



# SOPHON BMCV 开发参考手册

文件编号	RDF-TP-23-01-0-0
发布单位	产品经营部
发布时间	2025/10/27

## 注意

本文件所含信息均具有保密性质并仅限于内部使用。不得对本文件、本文件的任何部分或本文件所含的任何信息进行未经授权地使用、披露或复制。

## NOTICE

The information contained in this document is confidential and is intended only for internal use. Unauthorized use, disclosure or copying of this document, any part hereof or any information contained herein is strictly prohibited.

## 目录

1	BMCV 介绍	2
2	BMCV 公共组件	2
2.1	BMCV 接口总览	2
2.2	BMCV API 返回值	7
2.3	BMCV 尺寸限制	7
3	BMCV 基础结构体	8
3.1	bm_image	8
3.2	bm_image_format_ext	8
3.3	bm_data_format_ext	11
4	BMCV 基础函数	13
4.1	bm_image_create	13
4.2	bm_image_destroy	17
4.3	bm_image_copy_host_to_device	18
4.4	bm_image_copy_device_to_host	20
4.5	bm_image_attach	21
4.6	bm_image_detach	22
4.7	bm_image_alloc_dev_mem	23
4.8	bm_image_alloc_dev_mem_heap_mask	24
4.9	bm_image_get_byte_size	25
4.10	bm_image_get_device_mem	26
4.11	bm_image_alloc_contiguous_mem	27
4.12	bm_image_alloc_contiguous_mem_heap_mask	28
4.13	bm_image_free_contiguous_mem	29
4.14	bm_image_attach_contiguous_mem	30
4.15	bm_image_detach_contiguous_mem	31
4.16	bm_image_get_contiguous_device_mem	32
4.17	bm_image_get_format_info	33
4.18	bm_image_get_stride	34
4.19	bm_image_get_plane_num	35
4.20	bm_image_is_attached	36
4.21	bm_image_get_handle	37
4.22	bm_image_write_to_bmp	37
5	BMCV API	38
5.1	bmcv_base64_enc(dec)	38
5.2	bmcv_as_strided	40
5.3	bmcv_attribute_filter_topk	43
5.4	bmcv_batch_topk	49

5.5	bmcv_calc_hist . . . . .	51
5.5.1	直方图 . . . . .	51
5.5.2	带权重的直方图 . . . . .	54
5.6	bmcv_cmulp . . . . .	55
5.7	bmcv_distance . . . . .	58
5.8	bmcv_feature_match_normalized . . . . .	61
5.9	bmcv_feature_match . . . . .	65
5.10	bmcv_fft . . . . .	68
5.10.1	创建 . . . . .	68
5.10.2	执行 . . . . .	70
5.10.3	销毁 . . . . .	71
5.10.4	示例代码 . . . . .	71
5.11	bmcv_gemm . . . . .	72
5.12	bmcv_gemm_ext . . . . .	76
5.13	bmcv_hist_balance . . . . .	80
5.14	bmcv_hm_distance . . . . .	82
5.15	bmcv_matmul . . . . .	84
5.16	bmcv_min_max . . . . .	88
5.17	bmcv_nms . . . . .	89
5.18	bmcv_sort . . . . .	92
5.19	bmcv_stft . . . . .	95
5.20	bmcv_istft . . . . .	98
5.21	bmcv_faiss_indexflatIP . . . . .	101
5.22	bmcv_faiss_indexflatL2 . . . . .	106
5.23	bmcv_faiss_indexPQ_ADC . . . . .	112
5.24	bmcv_faiss_indexPQ_SDC . . . . .	118
5.25	bmcv_faiss_indexPQ_encode . . . . .	124
5.26	bmcv_ldc_rot . . . . .	128
5.27	bmcv_ldc_gdc . . . . .	132
5.28	bmcv_ldc_load_mesh . . . . .	136
5.29	bmcv_ldc_gen_mesh . . . . .	141
5.30	bmcv_dwa_rot . . . . .	146
5.31	bmcv_dwa_gdc . . . . .	149
5.32	bmcv_dwa_affine . . . . .	155
5.33	bmcv_dwa_fisheye . . . . .	159
5.34	bmcv_dwa_dewarp . . . . .	167
5.35	bmcv_blend . . . . .	170
5.36	bmcv_image_axpy . . . . .	178
5.37	bmcv_image_resize . . . . .	180
5.38	bmcv_image_watermark_superpose . . . . .	182
5.39	bmcv_image_absdiff . . . . .	189
5.40	bmcv_image_add_weighted . . . . .	192
5.41	bmcv_image_bayer2rgb . . . . .	196
5.42	bmcv_image_bitwise_and . . . . .	199
5.43	bmcv_image_bitwise_or . . . . .	202
5.44	bmcv_image_bitwise_xor . . . . .	206
5.45	bmcv_image_circle . . . . .	210

5.46	bmcv_image_convert_to . . . . .	212
5.47	bmcv_image_copy_to . . . . .	219
5.48	bmcv_image_csc_convert_to . . . . .	223
5.49	bmcv_image_csc_overlay . . . . .	229
5.50	bmcv_image_draw_lines . . . . .	232
5.51	bmcv_image_draw_point . . . . .	236
5.52	bmcv_image_draw_rectangle . . . . .	238
5.53	bmcv_image_fill_rectangle . . . . .	242
5.54	bmcv_image_flip . . . . .	245
5.55	bmcv_image_gaussian_blur . . . . .	248
5.56	bmcv_image_jpeg_dec . . . . .	251
5.57	bmcv_image_jpeg_enc . . . . .	255
5.58	bmcv_image_laplacian . . . . .	259
5.59	bmcv_image_mosaic . . . . .	261
5.60	bmcv_image_put_text . . . . .	265
5.61	bmcv_image_pyramid_down . . . . .	269
5.62	bmcv_image_quantify . . . . .	271
5.63	bmcv_image_storage_convert . . . . .	274
5.64	bmcv_image_threshold . . . . .	275
5.65	bmcv_image_transpose . . . . .	279
5.66	bmcv_image_rotate_trans . . . . .	282
5.67	bmcv_image_rotate . . . . .	286
5.68	bmcv_image_vpp_basic . . . . .	290
5.69	bmcv_image_vpp_convert . . . . .	296
5.70	bmcv_image_vpp_convert_padding . . . . .	299
5.71	bmcv_image_vpp_csc_matrix_convert . . . . .	302
5.72	bmcv_image_vpp_stitch . . . . .	306
5.73	bmcv_image_overlay . . . . .	309
5.74	bmcv_image_warp_affine . . . . .	313
5.75	bmcv_image_warp_affine_padding . . . . .	319
5.76	bmcv_image_warp_perspective . . . . .	320
5.77	bmcv_width_align . . . . .	329
5.78	bmcv_ive_bersen . . . . .	334
5.79	bmcv_ive_16bit_to_8bit . . . . .	337
5.80	bmcv_ive_dma . . . . .	342
5.81	bmcv_ive_filter . . . . .	347
5.82	bmcv_ive_csc . . . . .	351
5.83	bmcv_ive_filter_and_csc . . . . .	355
5.84	bmcv_ive_dilate . . . . .	358
5.85	bmcv_ive_erode . . . . .	362
5.86	bmcv_ive_ccl . . . . .	365
5.87	bmcv_ive_integ . . . . .	370
5.88	bmcv_ive_hist . . . . .	374
5.89	bmcv_ive_gradfg . . . . .	376
5.90	bmcv_ive_lbp . . . . .	381
5.91	bmcv_ive_normgrad . . . . .	385
5.92	bmcv_ive_sad . . . . .	390

---

5.93	bmcv_ive_stcandicorner . . . . .	396
5.94	bmcv_ive_mag_and_ang . . . . .	399
5.95	bmcv_ive_map . . . . .	404
5.96	bmcv_ive_ncc . . . . .	408
5.97	bmcv_ive_ord_stat_filter . . . . .	412
5.98	bmcv_ive_sobel . . . . .	415
5.99	bmcv_ive_gmm . . . . .	420
5.100	bmcv_ive_gmm2 . . . . .	425
5.101	bmcv_ive_resize . . . . .	433
5.102	bmcv_ive_thresh . . . . .	436
5.103	bmcv_ive_add . . . . .	445
5.104	bmcv_ive_sub . . . . .	448
5.105	bmcv_ive_and . . . . .	452
5.106	bmcv_ive_or . . . . .	455
5.107	bmcv_ive_xor . . . . .	458
5.108	bmcv_ive_canny . . . . .	461
5.109	bmcv_ive_match_bgmodel . . . . .	465
5.110	bmcv_ive_update_bgmodel . . . . .	472
5.111	bmcv_ive_frame_diff_motion . . . . .	475
5.112	bmcv_dpu_sgbm_disp . . . . .	479
5.113	bmcv_dpu_fgs_disp . . . . .	483
5.114	bmcv_dpu_online_disp . . . . .	486
5.115	bmcv_tde_fill . . . . .	488
5.116	bmcv_tde_convert . . . . .	490
5.117	bmcv_tde_line . . . . .	494
5.118	bmcv_tde_draw . . . . .	496

---

## 1 BMCV 介绍

BMCV 提供了一套基于 Sophon 硬件优化的机器视觉库，通过利用 VPSS、TPU、IVE、DPU、DWA 等模块，可以完成色彩空间转换、尺度变换、仿射变换、透射变换、线性变换、画框、JPEG 编解码、BASE64 编解码、NMS、排序、特征匹配等操作。

表 1.1: 修订记录

Revision	Date	Comments
1.1.0	2023/11	初版, 加入 vpss 模块相关接口
1.2.0	2023/12	加入 cv 算子  ive dpu ldc dwa blend 模块相关接口
1.3.0	2024/01	加入 ldc_gen_mesh dwa_affine dwa_fisheye 等接口
1.4.0	2024/02	加入 cv 算子 hist_balance quantify 等接口
1.5.0	2024/04	加入 cv 算子 put_text draw_lines 等接口
1.6.0	2024/05	加入 flip 的接口
1.7.0	2024/07	加入 overlay 的接口
1.8.0	2024/11	加入 cv 算子 faiss fft stft istft rotate 等接口
1.9.0	2025/01	各接口文档增加示例代码, put_text 支持中文, vpss 支持 pcie 模式, 优化 overlay 接口性能
2.0.0	2025/04	加入 circle point csc_overlay 等接口, 优化 draw_rect fill_rect 接口性能, vpss 接口多 crop 任务支持多核调用

## 2 BMCV 公共组件

本章节主要用于介绍 BMCV 一些公共组件。

### 2.1 BMCV 接口总览

将 BMCV API 按实现的硬件分类，并简要说明功能。

表 2.1: 多模块实现接口

API	功能	硬件
1 bmcv_image_copy_to	拷贝到目的图指定区域	TPU / VPSS
2 bmcv_image_put_text	写字	CPU / VPSS
3 bmcv_image_convert_to	线性变换	TPU / VPSS
4 bmcv_image_rotate	旋转	LDC / VPSS
5 bmcv_image_rotate_trans	旋转 (指定 TPU 实现)	TPU / VPSS

表 2.2: VPSS 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_image_vpp_convert	crop+ 色彩转换 + 缩放
bmcv_image_vpp_convert_padding	填充背景色
2 bmcv_image_vpp_csc_matrix_convert	自定义矩阵色彩转换
3 bmcv_image_vpp_basic	多功能组合
4 bmcv_image_csc_convert_to	多功能组合 + 线性变换
5 bmcv_image_draw_rectangle	画空心框
6 bmcv_image_fill_rectangle	画实心框
7 bmcv_image_flip	图像横向/纵向翻转
8 bmcv_image_mosaic	图像局部马赛克
9 bmcv_image_storage_convert	色彩转换 + 缩放
bmcv_image_storage_convert_with_cstype	可选标准色彩转换 + 缩放
10 bmcv_image_resize	图像缩放
11 bmcv_image_watermark_superpose	水印叠加
bmcv_image_watermark_repeat_superpose	重复水印叠加
12 bmcv_image_overlay	OSD 叠加
13 bmcv_image_vpp_stitch	图像拼接 (无重叠)
14 bmcv_image_circle	画圆
15 bmcv_image_draw_point	画正方形实心点
16 bmcv_image_csc_overlay	多功能组合 + OSD

表 2.3: JPEG 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_image_jpeg_dec	图像解码
2 bmcv_image_jpeg_enc	图像编码

表 2.4: TPU 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_image_absdiff	两图相减取绝对值
2 bmcv_image_axpy	$F = A * X + Y$ 的缩放加法
3 bmcv_image_add_weighted	两图加权融合
4 bmcv_image_bitwise_and	按位与
bmcv_image_bitwise_or	按位或
bmcv_image_bitwise_xor	按位异或
5 bmcv_image_width_align	按指定宽对齐
6 bmcv_sort	浮点数据排序
7 bmcv_image_threshold	像素阈值化
8 bmcv_feature_match	特征比对 (INT8)
bmcv_feature_match_normalized	特征比对 (FP32)
9 bmcv_hist_balance	直方图均衡化
10 bmcv_image_quantify	FP32 转 INT8

续下页

表 2.4 – 接上页

11	bmcv_faiss_indexflatIP	向量相似度搜索
	bmcv_faiss_indexflatL2	L2 距离的平方
	bmcv_faiss_indexPQ_encode	向量量化编码
	bmcv_faiss_indexPQ_encode_ext	可选数据类型
	bmcv_faiss_indexPQ_ADC	原始向量查询
	bmcv_faiss_indexPQ_ADC_ext	可选数据类型
	bmcv_faiss_indexPQ_SDC	编码向量查询
	bmcv_faiss_indexPQ_SDC_ext	可选数据类型
12	bmcv_image_transpose	图像转置
13	bmcv_image_pyramid_down	高斯金字塔下采样
14	bmcv_image_bayer2rgb	Bayer 转 RGBP
15	bmcv_batch_topk	取前 K 个最大/小值
16	bmcv_matmul	矩阵乘法
17	bmcv_calc_hist	直方图计算
	bmcv_calc_hist_with_weight	带权重直方图
	bmcv_hist_balance	直方图均衡化
18	bmcv_as_strided	重指定行列和步长
19	bmcv_min_max	取最大/小值。
20	bmcv_cmulp	复数乘法
21	bmcv_image_laplacian	梯度计算
22	bmcv_gemm	广义矩阵乘加
	bmcv_gemm_ext	可选数据类型
23	bmcv_image_warp	仿射变换
	bmcv_image_warp_affine	仿射变换
	bmcv_image_warp_affine_padding	背景图填充
	bmcv_image_warp_affine_similar_to_opencv	矩阵对齐 opencv
	bmcv_image_warp_affine_similar_to_opencv_padding	功能结合
24	bmcv_image_warp_perspective	透射变换
	bmcv_image_warp_perspective_with_coordinate	坐标输入
	bmcv_image_warp_perspective_similar_to_opencv	矩阵对齐 opencv
25	bmcv_distance	多点到一点欧式距离
26	bmcv_fft_execute	快速傅里叶变换
	bmcv_fft_execute_real_input	实数输入
27	bmcv_stft	短时傅里叶变换
	bmcv_istft	逆短时傅里叶变换

表 2.5: IVE 实现接口

num API	功能
1 bmcv_ive_filter	5x5 模板滤波
2 bmcv_ive_csc	色彩空间转换
3 bmcv_ive_filter_and_csc	功能组合
4 bmcv_ive_dilate	5x5 模板膨胀
5 bmcv_ive_erode	5x5 模板腐蚀

续下页

表 2.5 – 接上页

6	bmcv_ive_ccl	连通区域标记
7	bmcv_ive_integ	积分图计算
8	bmcv_ive_hist	直方图计算
9	bmcv_ive_gradfg	梯度前景图计算
10	bmcv_ive_lbp	局部二值计算
11	bmcv_ive_normgrad	归一化梯度计算
12	bmcv_ive_sad	绝对差和
13	bmcv_ive_stcandidcorner	角点计算
14	bmcv_ive_mag_and_ang	5x5 模板幅值/幅角计算
15	bmcv_ive_map	映射赋值
16	bmcv_ive_ncc	归一化互相关系数计算
17	bmcv_ive_ord_stat_filter	3x3 模板顺序统计量滤波
18	bmcv_ive_sobel	5x5 模板梯度计算
19	bmcv_ive_gmm	背景建模
20	bmcv_ive_gmm2	背景建模支持更多参数
21	bmcv_ive_resize	图像缩放
22	bmcv_ive_thresh	阈值化
23	bmcv_ive_add	两图加权相加
24	bmcv_ive_sub	两图相减
25	bmcv_ive_and	两图相与
26	bmcv_ive_or	两图相或
27	bmcv_ive_xor	两图异或
28	bmcv_ive_canny	边缘图计算
29	bmcv_ive_match_bgmodel	获取前景数据
30	bmcv_ive_update_bgmodel	更新背景模型
31	bmcv_ive_frame_diff_motion	背景相减
32	bmcv_ive_bernsen	图像二值化
33	bmcv_ive_16bit_to_8bit	16Bit 转 8Bit
34	bmcv_ive_dma	内存拷贝

表 2.6: LDC 实现接口

num	API	功能
1	bmcv_ldc_rot	旋转
2	bmcv_ldc_gdc	畸变校正
3	bmcv_ldc_gdc_load_mesh	畸变校正 (加载已有 MESH 表)

表 2.7: DWA 实现接口

num	API	功能
1	bmcv_dwa_rot	旋转
2	bmcv_dwa_gdc	畸变校正
3	bmcv_dwa_affine	仿射校正
4	bmcv_dwa_fisheye	鱼眼校正
5	bmcv_dwa_dewarp	畸变校正 (输入 GRIDINFO)

表 2.8: DPU 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_dpu_sgbm_disp	半全局块匹配计算
2 bmcv_dpu_fgs_disp	快速全局平滑计算
3 bmcv_dpu_online_disp	功能组合

表 2.9: BLEND 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_blend	图像拼接 (重叠区域平滑过渡)

表 2.10: TDE 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_tde_fill	颜色填充
2 bmcv_tde_convert	crop/csc/缩放/旋转/叠图
3 bmcv_tde_line	绘制线条
4 bmcv_tde_draw	填充多边形

表 2.11: SPACC 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_base64_enc	base64 编码
2 bmcv_base64_dec	base64 解码

表 2.12: CPU 实现接口

<b>nui API</b>	<b>功能</b>
1 bmcv_image_draw_lines	划线
2 bmcv_fft_1d_create_plan bmcv_fft_2d_create_plan bmcv_fft_destroy_plan	一维 FFT 任务创建 二维 FFT 任务创建 FFT 任务销毁
3 bmcv_ldc_gdc_gen_mesh	畸变校正 MESH 表计算
4 bmcv_gen_text_watermark	生成文字水印
5 bm_image_write_to_bmp	保存 BMP 图像到文件
6 bm_read_bin	读取 RAW 图像到内存
7 bm_write_bin	保存 RAW 图像到文件

## 2.2 BMCV API 返回值

在代码中，我们定义了一个枚举类型 `bm_status_t`，其中包含不同的状态值。

表 2.13: bmcv 返回值表

返回值	描述
BM_SUCCESS	成功
BM_ERR_FAILURE	失败
BM_ERR_DEVNOTREADY	设备尚未准备好
BM_ERR_TIMEOUT	超时
BM_ERR_PARAM	参数无效
BM_ERR_NOMEM	内存不足
BM_ERR_DATA	数据错误
BM_ERR_BUSY	忙碌
BM_ERR_NOFEATURE	暂时不支持
BM_NOT_SUPPORTED	不支持

## 2.3 BMCV 尺寸限制

表 2.14: bmcv 参数范围表

模块	最小尺寸	最大尺寸	输入 stride 对齐要求	输出 stride 对齐要求
TPU	8*8	4096*4096	1	1
VPSS	16*16	8192*8192	1	1
LDC	64*64	4608*4608	64	64
DWA	32*32	4096*4096	32	32
IVE	64*64	1920*1080	16	16
DPU	64*64	1920*1080	16	16(sgbm), 32(fgs)
BLEND	64*64	4608*2160	16	16
TDE	16*16	8192*8192	32	1

### 3 BMCV 基础结构体

本章节主要用于介绍 BMCV 使用的基础结构体；我们不保证此文档更改后能及时告知用户，因此请使用最新版 released SDK 里面的文档。

#### 3.1 bm\_image

bm\_image 结构体定义如下：

```
struct bm_image {
    int width;
    int height;
    bm_image_format_ext image_format;
    bm_data_format_ext data_type;
    struct bm_image_private* image_private;
};
```

表 3.1: bm\_image 结构成员

序号	术语	定义说明
1	width	图像的宽
2	height	图像的高
3	image_format	图像的色彩格式
4	data_type	图像的数据格式
5	image_private	图像的私有数据

#### 3.2 bm\_image\_format\_ext

bm\_image\_format\_ext 有以下枚举类型。

```
typedef enum bm_image_format_ext_{
    FORMAT_YUV420P,
    FORMAT_YUV422P,
    FORMAT_YUV444P,
    FORMAT_NV12,
    FORMAT_NV21,
    FORMAT_NV16,
    FORMAT_NV61,
    FORMAT_NV24,
    FORMAT_RGB_PLANAR,
    FORMAT_BGR_PLANAR,
    FORMAT_RGB_PACKED,
    FORMAT_BGR_PACKED,
    FORMAT_RGBP_SEPARATE,
    FORMAT_BGRP_SEPARATE,
    FORMAT_GRAY,
    FORMAT_COMPRESSED,
```

(续下页)

(接上页)

```

FORMAT_HSV_PLANAR,
FORMAT_ARGB_PACKED,
FORMAT_ABGR_PACKED,
FORMAT_YUV444_PACKED,
FORMAT_YVU444_PACKED,
FORMAT_YUV422_YUYV,
FORMAT_YUV422_YVYU,
FORMAT_YUV422_UYVY,
FORMAT_YUV422_VYUY,
FORMAT_RGBYP_PLANAR,
FORMAT_HSV180_PACKED,
FORMAT_HSV256_PACKED,
FORMAT_BAYER,
FORMAT_BAYER_RG8,
FORMAT_ARGB4444_PACKED,
FORMAT_ABGR4444_PACKED,
FORMAT_ARGB1555_PACKED,
FORMAT_ABGR1555_PACKED,
} bm_image_format_ext;

```

表 3.2: 各个格式说明

序号	格式名称	说明
1	FORMAT_YUV420P	表示预创建一个 YUV420 格式的图片，有三个 plane。
2	FORMAT_YUV422P	表示预创建一个 YUV422 格式的图片，有三个 plane。
3	FORMAT_YUV444P	表示预创建一个 YUV444 格式的图片，有三个 plane。
4	FORMAT_NV12	表示预创建一个 NV12 格式的图片，有两个 plane。
5	FORMAT_NV21	表示预创建一个 NV21 格式的图片，有两个 plane。
6	FORMAT_NV16	表示预创建一个 NV16 格式的图片，有两个 plane。
7	FORMAT_NV61	表示预创建一个 NV61 格式的图片，有两个 plane。
8	FORMAT_NV24	表示预创建一个 NV24 格式的图片，有两个 plane。
9	FORMAT_RGB_PLANAR	表示预创建一个 RGB 格式的图片，RGB 分开排列，有一个 plane。
10	FORMAT_BGR_PLANAR	表示预创建一个 BGR 格式的图片，BGR 分开排列，有一个 plane。
11	FORMAT_RGB_PACKED	表示预创建一个 RGB 格式的图片，RGB 交错排列，有一个 plane。

续下页

表 3.2 - 接上页

序号	格式名称	说明
12	FORMAT_BGR_PACKED	表示预创建一个 BGR 格式的图片, BGR 交错排列, 有一个 plane。
13	FORMAT_RGBP_SEPARATE	表示预创建一个 RGB planar 格式的图片, RGB 分开排列并各占一个 plane, 共有 3 个 plane。
14	FORMAT_BGRP_SEPARATE	表示预创建一个 BGR planar 格式的图片, BGR 分开排列并各占一个 plane, 共有 3 个 plane。
15	FORMAT_GRAY	表示预创建一个灰度图格式的图片, 有一个 plane。
16	FORMAT_COMPRESSED	表示预创建一个 VPU 内部压缩格式的图片, 共有四个 plane, 分别存放内容如下: plane0: Y 压缩表、plane1: Y 压缩数据、plane2: CbCr 压缩表、plane3: CbCr 压缩数据。
17	FORMAT_HSV_PLANAR	表示预创建一个 HSV planar 格式的图片, H 的范围为 0~180, 有三个 plane。
18	FORMAT_ARGB_PACKED	表示预创建一个 ARGB 格式的图片, 该图片仅有一个 plane, 并且像素值以 BGRA 顺序交错连续排列, 即 BGRABGRA。
19	FORMAT_ABGR_PACKED	表示预创建一个 ABGR 格式的图片, 该图片仅有一个 plane, 并且像素值以 RGBA 顺序交错连续排列, 即 RGBARGBA。
20	FORMAT_YUV444_PACKED	表示预创建一个 YUV444 格式的图片, YUV 交错排列, 有一个 plane。
21	FORMAT_YVU444_PACKED	表示预创建一个 YVU444 格式的图片, YVU 交错排列, 有一个 plane。
22	FORMAT_YUV422_YUYV	表示预创建一个 YUV422 格式的图片, YUYV 交错排列, 有一个 plane。
23	FORMAT_YUV422_YVYU	表示预创建一个 YUV422 格式的图片, YVYU 交错排列, 有一个 plane。
24	FORMAT_YUV422_UYVY	表示预创建一个 YUV422 格式的图片, UYVY 交错排列, 有一个 plane。
25	FORMAT_YUV422_VYUY	表示预创建一个 YUV422 格式的图片, VYUY 交错排列, 有一个 plane。
26	FORMAT_RGBYP_PLANAR	表示预创建一个 RGBY planar 格式的图片, 有四个 plane。
27	FORMAT_HSV180_PACKED	表示预创建一个 HSV 格式的图片, H 的范围为 0~180, HSV 交错排列, 有一个 plane。
28	FORMAT_HSV256_PACKED	表示预创建一个 HSV 格式的图片, H 的范围为 0~255, HSV 交错排列, 有一个 plane。

续下页

表 3.2 - 接上页

序号	格式名称	说明
29	FORMAT_BAYER	表示预创建一个 bayer 格式的图片，有一个 plane，像素排列方式是 BGGR，且宽高需要是偶数。
30	FORMAT_BAYER_RG8	表示预创建一个 bayer 格式的图片，有一个 plane，像素排列方式是 RGGB，且宽高需要是偶数。
31	FOR-MAT_ARGB4444_PACKED	表示预创建一个 ARGB 格式的图片，包含四个通道，即 A: 透明度 (Alpha)、R: 红色 (Red)、G: 绿色 (Green)、B: 蓝色 (Blue)，每个通道占 4 位，共 16 位，有一个 plane。
32	FOR-MAT_ABGR4444_PACKED	表示预创建一个 ABGR 格式的图片，包含四个通道，即 A: 透明度 (Alpha)、B: 蓝色 (Blue)、G: 绿色 (Green)、R: 红色 (Red)，每个通道占 4 位，共 16 位，有一个 plane。
33	FOR-MAT_ARGB1555_PACKED	表示预创建一个 ARGB 格式的图片，包含四个通道，即 A: 透明度 (Alpha)、R: 红色 (Red)、G: 绿色 (Green)、B: 蓝色 (Blue)，各个通道分别占 1、5、5、5 位，共 16 位，有一个 plane。
34	FOR-MAT_ABGR1555_PACKED	表示预创建一个 ABGR 格式的图片，包含四个通道，即 A: 透明度 (Alpha)、B: 蓝色 (Blue)、G: 绿色 (Green)、R: 红色 (Red)，各个通道分别占 1、5、5、5 位，共 16 位，有一个 plane。

### 3.3 bm\_data\_format\_ext

bm\_data\_format\_ext 有以下枚举类型。

```
typedef enum bm_image_data_format_ext {
    DATA_TYPE_EXT_FLOAT32,
    DATA_TYPE_EXT_1N_BYTE,
    DATA_TYPE_EXT_1N_BYTE_SIGNED,
    DATA_TYPE_EXT_FP16,
    DATA_TYPE_EXT_BF16,
    DATA_TYPE_EXT_U16,
    DATA_TYPE_EXT_S16,
    DATA_TYPE_EXT_U32,
    DATA_TYPE_EXT_4N_BYTE = 254,
    DATA_TYPE_EXT_4N_BYTE_SIGNED,
}bm_image_data_format_ext;
```

表 3.3: 各个格式说明

序号	格式名称	说明
1	DATA_TYPE_EXT_FLOAT32	表示所创建的图片数据格式为单精度浮点数。
2	DATA_TYPE_EXT_1N_BYTE	表示所创建图片数据格式为普通无符号 UINT8。
3	DATA_TYPE_EXT_1N_BYTE_SIGNED	表示所创建图片数据格式为普通有符号 INT8。
4	DATA_TYPE_EXT_FP16	表示所创建的图片数据格式为半精度浮点数，5bit 表示指数，10bit 表示小数。
5	DATA_TYPE_EXT_BF16	表示所创建的图片数据格式为 16bit 浮点数，实际是对 FLOAT32 单精度浮点数截断数据，即用 8bit 表示指数，7bit 表示小数。
6	DATA_TYPE_EXT_U16	表示所创建图片数据格式为普通无符号 UINT16。
7	DATA_TYPE_EXT_S16	表示所创建图片数据格式为普通有符号 INT16。
8	DATA_TYPE_EXT_U32	表示所创建图片数据格式为普通无符号 UINT32。
9	DATA_TYPE_EXT_4N_BYTE	表示所创建图片数据格式为 4N UINT8，即四张无符号 UINT8 图片数据交错排列，一个 bm_image 对象其实含有四张属性相同的图片。
10	DATA_TYPE_EXT_4N_BYTE_SIGNED	表示所创建图片数据格式为 4N INT8，即四张有符号 INT8 图片数据交错排列。

## 4 BMCV 基础函数

本章节主要用于介绍 BMCV 常用的基础函数。

### 4.1 bm\_image\_create

#### 【描述】

创建一个 bm\_image 结构。

#### 【语法】

```
1 bm_status_t bm_image_create(
2     bm_handle_t handle,
3     int img_h,
4     int img_w,
5     bmcv_image_format_ext image_format,
6     bmcv_data_format_ext data_type,
7     bm_image *image,
8     int* stride = NULL);
```

#### 【参数】

表 4.1: bm\_image\_create 参数表

参数名称	输入/输出	描述
handle	输入	输入参数。设备环境句柄，通过调用 bm_dev_request 获取。
img_h	输入	图片高度。
img_w	输入	图片宽度。
image_format	输入	所需创建 bm_image 图片格式。
data_type	输入	所需创建 bm_image 数据格式。
*image	输出	输出填充的 bm_image 结构指针。
*stride	输入	所创建 bm_image 将要关联的 device memory 内存布局，即每个 plane 的 width stride 值。在传入 NULL 时默认与一行的数据宽度相同，以 byte 计数。更多用法请参阅注意事项。

#### 【返回值】

成功调用将返回 BM\_SUCCESS，并填充输出的 image 指针结构。这个结构中记录了图片的大小，以及相关格式。但此时并没有与任何 device memory 关联，也没有申请数据对应的 device memory。

### 【注意事项】

1. 以下图片格式的宽可以是奇数，接口内部会调整到偶数再完成相应功能。但建议尽量使用偶数的宽，这样可以发挥最大的效率。
  - FORMAT\_YUV420P
  - FORMAT\_YUV422P
  - FORMAT\_NV12
  - FORMAT\_NV21
  - FORMAT\_NV16
  - FORMAT\_NV61
2. 以下图片格式的高可以是奇数，接口内部会调整到偶数再完成相应功能。但建议尽量使用偶数的高，这样可以发挥最大的效率。
  - FORMAT\_YUV420P
  - FORMAT\_NV12
  - FORMAT\_NV21
3. FORMAT\_COMPRESSED 图片格式的图片创建时需要将 stride 参数的数组设置为各通道的数据长度。若 stride 设置为 NULL，后续则需要用 bm\_image\_attach 绑定 device memory。
4. 若 stride 参数设置为 NULL，此时默认各个 plane 的数据是 compact 排列，创建的 bm\_image 各 plane 的 stride 被设置为与 image\_format 对应的默认值。
5. 如果传入 stride 非 NULL，则会检测传入的各 stride 值是否大于 image\_format 对应的所有 plane 的默认 stride，若大于，则创建的 bm\_image 各 plane 的 stride 值设定为传入值；若反之，则设定为默认 stride 值向上对齐传入值的值。
6. 默认 stride 值的计算方法如下：

```
1 int data_size = 1;
2 switch (data_type) {
3     case DATA_TYPE_EXT_FLOAT32:
4     case DATA_TYPE_EXT_U32:
5         data_size = 4;
6         break;
7     case DATA_TYPE_EXT_FP16:
8     case DATA_TYPE_EXT_BF16:
9     case DATA_TYPE_EXT_U16:
10    case DATA_TYPE_EXT_S16:
```

(续下页)

(接上页)

```
11     data_size = 2;
12     break;
13 default:
14     data_size = 1;
15     break;
16 }
17 int default_stride[3] = {0};
18 switch (image_format) {
19     case FORMAT_YUV420P: {
20         plane_num = 3;
21         default_stride[0] = width * data_size;
22         default_stride[1] = (ALIGN(width, 2) >> 1) * data_size;
23         default_stride[2] = default_stride[1];
24         break;
25     }
26     case FORMAT_YUV422P: {
27         plane_num = 3;
28         default_stride[0] = width * data_size;
29         default_stride[1] = (ALIGN(width, 2) >> 1) * data_size;
30         default_stride[2] = default_stride[1];
31         break;
32     }
33     case FORMAT_YUV444P:
34     case FORMAT_BGRP_SEPARATE:
35     case FORMAT_RGBP_SEPARATE:
36     case FORMAT_HSV_PLANAR: {
37         plane_num = 3;
38         default_stride[0] = width * data_size;
39         default_stride[1] = width * data_size;
40         default_stride[2] = default_stride[1];
41         break;
42     }
43     case FORMAT_NV24: {
44         plane_num = 2;
45         default_stride[0] = width * data_size;
46         default_stride[1] = width * 2 * data_size;
47         break;
48     }
49     case FORMAT_NV12:
50     case FORMAT_NV21: {
51         plane_num = 2;
52         default_stride[0] = width * data_size;
53         default_stride[1] = ALIGN(width, 2) * data_size;
54         break;
55     }
56     case FORMAT_NV16:
57     case FORMAT_NV61: {
58         plane_num = 2;
59         default_stride[0] = width * data_size;
60         default_stride[1] = ALIGN(width, 2) * data_size;
61         break;
62 }
```

(续下页)

(接上页)

```
62     }
63     case FORMAT_GRAY:
64     case FORMAT_BAYER:
65     case FORMAT_BAYER_RG8: {
66         plane_num = 1;
67         default_stride[0] = width * data_size;
68         break;
69     }
70     case FORMAT_COMPRESSED: {
71         plane_num = 4;
72         break;
73     }
74     case FORMAT_YUV444_PACKED:
75     case FORMAT_YVU444_PACKED:
76     case FORMAT_HSV180_PACKED:
77     case FORMAT_HSV256_PACKED:
78     case FORMAT_BGR_PACKED:
79     case FORMAT_RGB_PACKED: {
80         plane_num = 1;
81         default_stride[0] = width * 3 * data_size;
82         break;
83     }
84     case FORMAT_ABGR_PACKED:
85     case FORMAT_ARGB_PACKED: {
86         plane_num = 1;
87         default_stride[0] = width * 4 * data_size;
88         break;
89     }
90     case FORMAT_BGR_PLANAR:
91     case FORMAT_RGB_PLANAR: {
92         plane_num = 1;
93         default_stride[0] = width * data_size;
94         break;
95     }
96     case FORMAT_RGBYP_PLANAR: {
97         plane_num = 4;
98         default_stride[0] = width * data_size;
99         default_stride[1] = width * data_size;
100        default_stride[2] = width * data_size;
101        default_stride[3] = width * data_size;
102        break;
103    }
104    case FORMAT_YUV422_YUYV:
105    case FORMAT_YUV422_YVYU:
106    case FORMAT_YUV422_UYVY:
107    case FORMAT_YUV422_VYUY:
108    case FORMAT_ARGB4444_PACKED:
109    case FORMAT_ABGR4444_PACKED:
110    case FORMAT_ARGB1555_PACKED:
111    case FORMAT_ABGR1555_PACKED: {
112        plane_num = 1;
```

(续下页)

(接上页)

```
113     default_stride[0] = width * 2 * data_size;  
114     break;  
115 }  
116 }
```

## 4.2 bm\_image\_destroy

### 【描述】

销毁 bm\_image 对象，避免内存泄漏，与 bm\_image\_create 成对使用。

### 【语法】

```
1 bm_status_t bm_image_destroy(  
2     bm_image* image  
3 );
```

### 【参数】

表 4.2: bm\_image\_destroy 参数表

参数名称	输入/输出	描述
*image	输入	待销毁的 bm_image 对象指针。

### 【返回值】

成功返回将销毁该 bm\_image 对象。

注意，如果该对象的 device memory 是直接以该对象用 bm\_image\_alloc\_dev\_mem 系列函数申请的，则 bm\_image\_destroy 释放该内存空间。

如果该对象的 device memory 是使用 bm\_image\_attach 绑定而来的，则被视为由用户自己管理的内存空间，将不会被释放。

### 4.3 bm\_image\_copy\_host\_to\_device

#### 【描述】

该 API 将 host 端数据拷贝到 bm\_image 结构对应的 device memory 中。

#### 【语法】

```

1  bm_status_t bm_image_copy_host_to_device(
2      bm_image image,
3      void* buffers[]
4 );

```

#### 【参数】

表 4.3: bm\_image\_copy\_host\_to\_device 参数表

参数名称	输入/输出	描述
image	输入	待填充 device memory 数据的 bm_image 对象。
* buffers[]	输入	host 端指针, buffers 为指向不同 plane 数据的指针。

其中, buffers[] 的数组长度应等于创建 bm\_image 结构时 image\_format 对应的 plane 数。每个 plane 数据量的具体计算方法如下(以紧凑排列的默认 stride 为例) :

```

1  switch (image_format) {
2      case FORMAT_YUV420P: {
3          size[0] = width * height * data_size;
4          size[1] = ALIGN(width, 2) * ALIGN(height, 2) / 4 * data_size;
5          size[2] = ALIGN(width, 2) * ALIGN(height, 2) / 4 * data_size;
6          break;
7      }
8      case FORMAT_YUV422P: {
9          size[0] = width * height * data_size;
10         size[1] = ALIGN(width, 2) * ALIGN(height, 2) / 2 * data_size;
11         size[2] = ALIGN(width, 2) * ALIGN(height, 2) / 2 * data_size;
12         break;
13     }
14     case FORMAT_YUV444P:
15     case FORMAT_BGRP_SEPARATE:
16     case FORMAT_RGBP_SEPARATE:
17     case FORMAT_HSV_PLANAR: {
18         size[0] = width * height * data_size;
19         size[1] = width * height * data_size;

```

(续下页)

(接上页)

```
20     size[2] = width * height * data_size;
21     break;
22 }
23 case FORMAT_NV24: {
24     size[0] = width * height * data_size;
25     size[1] = width * 2 * height * data_size;
26     break;
27 }
28 case FORMAT_NV12:
29 case FORMAT_NV21: {
30     size[0] = width * height * data_size;
31     size[1] = ALIGN(width, 2) * ALIGN(height, 2) / 2 * data_size;
32     break;
33 }
34 case FORMAT_NV16:
35 case FORMAT_NV61: {
36     size[0] = width * height * data_size;
37     size[1] = ALIGN(width, 2) * height * data_size;
38     break;
39 }
40 case FORMAT_GRAY:
41 case FORMAT_BAYER:
42 case FORMAT_BAYER_RG8: {
43     size[0] = width * height * data_size;
44     break;
45 }
46 case FORMAT_COMPRESSED: {
47     break;
48 }
49 case FORMAT_YUV444_PACKED:
50 case FORMAT_YVU444_PACKED:
51 case FORMAT_HSV180_PACKED:
52 case FORMAT_HSV256_PACKED:
53 case FORMAT_BGR_PACKED:
54 case FORMAT_RGB_PACKED: {
55     size[0] = width * 3 * height * data_size;
56     break;
57 }
58 case FORMAT_ABGR_PACKED:
59 case FORMAT_ARGB_PACKED: {
60     size[0] = width * 4 * height * data_size;
61     break;
62 }
63 case FORMAT_BGR_PLANAR:
64 case FORMAT_RGB_PLANAR: {
65     size[0] = width * height * 3 * data_size;
66     break;
67 }
68 case FORMAT_RGBYP_PLANAR: {
69     size[0] = width * height * data_size;
70     size[1] = width * height * data_size;
```

(续下页)

(接上页)

```
71     size[2] = width * height * data_size;
72     size[3] = width * height * data_size;
73     break;
74 }
75 case FORMAT_YUV422_YUYV:
76 case FORMAT_YUV422_YVYU:
77 case FORMAT_YUV422_UYVY:
78 case FORMAT_YUV422_VYUY:
79 case FORMAT_ARGB4444_PACKED:
80 case FORMAT_ABGR4444_PACKED:
81 case FORMAT_ARGB1555_PACKED:
82 case FORMAT_ABGR1555_PACKED: {
83     size[0] = width * 2 * height * data_size;
84     break;
85 }
86 }
```

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 备注:

1. 如果 bm\_image 未由 bm\_image\_create 创建，则返回失败。
2. 如果所传入的 bm\_image 对象还没有与 device memory 相关联的话，会自动为每个 plane 申请对应 image\_private->plane\_byte\_size 大小的 device memory，并将 host 端数据拷贝到申请的 device memory 中。如果申请 device memory 失败，则该 API 调用失败。
3. 如果所传入的 bm\_image 对象图片格式为 FORMAT\_COMPRESSED，且该 bm\_image 在 create 时没有设置 stride 参数，则直接返回失败。
4. 如果拷贝失败，则该 API 调用失败。

## 4.4 bm\_image\_copy\_device\_to\_host

### 【描述】

该 API 将 device memory 中的图片数据拷贝到 host 端。

### 【语法】

```

1 bm_status_t bm_image_copy_device_to_host(
2     bm_image image,
3     void* buffers[]
4 );

```

## 【参数】

表 4.4: bm\_image\_copy\_device\_to\_host 参数表

参数名称	输入/输出	描述
image	输入	待传输数据的 bm_image 对象。
* buffers[]	输出	host 端指针, buffers 为指向不同 plane 数据的指针, 每个 plane 需要传输的数据量可以通过 bm_image_get_byte_size API 获取。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 备注:

- 如果 bm\_image 未由 bm\_image\_create 创建, 则返回失败。
- 如果 bm\_image 没有与 device memory 相关联的话, 将返回失败。
- 数据传输失败的话, API 将调用失败。
- 如果该函数成功返回, 会将所关联的 device memory 中的数据拷贝到 host 端 buffers 中。
- 调用之前, 请为 buffers 中的各指针提前分配虚拟内存, 内存大小应为每个 plane 需要传输的数据量。

## 4.5 bm\_image\_attach

### 【描述】

如果用户希望自己管理 device memory, 或者 device memory 由外部组件 (VPU/VPP 等) 产生, 则可以调用以下 API 将这块 device memory 与 bm\_image 相关联。

### 【语法】

```
1 bm_status_t bm_image_attach(  
2     bm_image image,  
3     bm_device_mem_t* device_memory  
4 );
```

## 【参数】

表 4.5: bm\_image\_attach 参数表

参数名称	输入/输出	描述
image	输入	待关联的 bm_image 对象。
* device_memory	输入	填充 bm_image 所需的 device_memory，数量应由创建 bm_image 结构时 image_format 对应的 plane 数所决定。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 备注:

1. 如果 bm\_image 未由 bm\_image\_create 创建, 则返回失败。
2. 该函数成功调用时, bm\_image 对象将与传入的 device\_memory 对象相关联。
3. bm\_image 不会对通过这种方式关联的 device\_memory 进行管理, 即在销毁的时候并不会释放对应的 device\_memory, 需要用户自行管理这块 device\_memory。

## 4.6 bm\_image\_detach

### 【描述】

该 API 用于将 device memory 与 bm\_image 解关联。

### 【语法】

1 bm\_status\_t bm\_image\_detach(bm\_image image);

## 【参数】

表 4.6: bm\_image\_detach 参数表

参数名称	输入/输出	描述
image	输入	待解关联的 bm_image 对象。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意事项】

- 如果传入的 bm\_image 对象未被创建, 则返回失败。
- 该函数成功返回时, 会对 bm\_image 对象关联的 device\_memory 进行解关联, bm\_image 对象将不再关联 device\_memory。
- 如果解关联的 device\_memory 不是由 bm\_image\_attach 关联来的, 而是由该 bm\_image 对象通过 bm\_image\_alloc\_dev\_mem 系列函数申请的, 该 API 则会释放这块 device memory。
- 如果对象未关联任何 device memory, 则直接返回成功。

## 4.7 bm\_image\_alloc\_dev\_mem

### 【描述】

该 API 为 bm\_image 对象申请内部管理内存, 所申请 device memory 大小为各个 plane 所需 device memory 大小之和。plane\_byte\_size 计算方法在 bm\_image\_copy\_host\_to\_device 中已经介绍, 或者通过调用 bm\_image\_get\_byte\_size API 来确认。

### 【语法】

1 bm\_status\_t bm\_image\_alloc\_dev\_mem(  
2     bm\_image image,

(续下页)

(接上页)

```

3     int heap_id
4 );

```

## 【参数】

表 4.7: bm\_image\_alloc\_dev\_mem 参数表

参数名称	输入/输出	描述
image	输入	待申请 device memory 的 bm_image 对象。
heap_id	输入	待申请 device memory 所在的 heap 的 id 号。

## 【注意事项】

1. 如果 bm\_image 对象未创建，则返回失败。
2. 如果 image format 为 FORMAT\_COMPRESSED，且该 bm\_image 在 create 时没有设置 stride 参数，则返回失败。
3. 如果 bm\_image 对象已关联 device memory，则会直接返回成功。
4. 所申请 device memory 由内部管理，在 destroy 或者不再使用时释放。

## 4.8 bm\_image\_alloc\_dev\_mem\_heap\_mask

### 【描述】

该 API 为 bm\_image 对象申请内部管理内存，所申请 device memory 大小为各个 plane 所需 device memory 大小之和。plane\_byte\_size 计算方法在 bm\_image\_copy\_host\_to\_device 中已经介绍，或者通过调用 bm\_image\_get\_byte\_size API 来确认。

### 【语法】

```

1 bm_status_t bm_image_alloc_dev_mem_heap_mask(
2     bm_image    image,
3     int         heap_mask
4 );

```

## 【参数】

表 4.8: bm\_image\_alloc\_dev\_mem\_heap\_mask 参数表

参数名称	输入/输出	描述
image	输入	待申请 device memory 的 bm_image 对象。
heap_mask	输入	选择一个或多个 heap id 的掩码，每一个 bit 表示一个 heap id 的是否有效，1 表示可以在该 heap 分配，0 则表示不可以。最低位表示 heap0，以此类推。比如 heap_mask=0b10 则表示指定在 heap1 上分配空间，heap_mask=0b101 则表示指定在 heap0 或者 heap2 上分配空间。

**【注意事项】**

1. 如果 bm\_image 对象未创建，则返回失败。
2. 如果 image format 为 FORMAT\_COMPRESSED，且该 bm\_image 在 create 时没有设置 stride 参数，则返回失败。
3. 如果 bm\_image 对象已关联 device memory，则会直接返回成功。
4. 所申请 device memory 由内部管理，在 destroy 或者不再使用时释放。

**4.9 bm\_image\_get\_byte\_size****【描述】**

获取 bm\_image 对象各个 plane 字节大小。

**【语法】**

```

1  bm_status_t bm_image_get_byte_size(
2      bm_image image,
3      int* size
4 );

```

**【参数】**

表 4.9: bm\_image\_get\_byte\_size 参数表

参数名称	输入/输出	描述
image	输入	待获取 device memory 大小的 bm_image 对象。
*size	输出	返回的各个 plane 字节数结果。

**【注意事项】**

1. 如果 bm\_image 对象未创建, 则返回失败。
2. 如果 image format 为 FORMAT\_COMPRESSED 并且未关联外部 device memory, 则返回失败。
3. 该函数成功调用时将在 size 指针中填充各个 plane 所需的 device memory 字节大小。size 大小的计算方法在 bm\_image\_copy\_host\_to\_device 中已介绍。
4. 如果 bm\_image 对象未关联 device memory, 除了 FORMAT\_COMPRESSED 格式外, 其他格式仍能够成功返回并填充 size。

**4.10 bm\_image\_get\_device\_mem****【描述】**

获取 bm\_image 对象各个 plane 的 device memory。

**【语法】**

```

1  bm_status_t bm_image_get_device_mem(
2      bm_image image,
3      bm_device_mem_t* mem
4 );

```

**【参数】**

表 4.10: bm\_image\_get\_device\_mem 参数表

参数名称	输入/输出	描述
image	输入	待获取 device memory 的 bm_image 对象。
*mem	输出	返回的各个 plane 的 bm_device_mem_t 结构。

## 【注意事项】

1. 该函数将成功返回时，将在 mem 指针中填充 bm\_image 对象各个 plane 关联的 bm\_device\_mem\_t 结构。
2. 如果 bm\_image 对象未关联 device memory 的话，将返回失败。
3. 如果 bm\_image 对象未创建，则返回失败。
4. 如果是 bm\_image 自行申请的 device memory 结构，请不要将其释放，以免发生 double free。

## 4.11 `bm_image_alloc_contiguous_mem`

### 【描述】

为多个 image 分配连续的内存。

### 【语法】

```
1 bm_status_t bm_image_alloc_contiguous_mem(
2     int         image_num,
3     bm_image   *images
4 );
```

### 【参数】

表 4.11: `bm_image_alloc_contiguous_mem` 参数表

参数名称	输入/输出	描述
<code>image_num</code>	输入	待分配内存的 image 个数。
<code>*images</code>	输入	待分配内存的 image 的指针。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

## 【注意事项】

1. `image_num` 应该大于 0, 否则将返回错误。

2. 如传入的 image 已分配或者 attach 过内存，应先 detach 已有内存，否则将返回失败。
3. 所有待分配的 image 应该尺寸相同，否则将返回错误。
4. 当希望 destroy 的 image 是通过调用本 api 所分配的内存时，应先调用 bm\_image\_free\_contiguous\_mem 将分配内存释放，再用 bm\_image\_destroy 来实现 destroy image
5. bm\_image\_alloc\_contiguous\_mem 与 bm\_image\_free\_contiguous\_mem 应成对使用。

## 4.12 bm\_image\_alloc\_contiguous\_mem\_heap\_mask

### 【描述】

为多个 image 在指定的 heap 上分配连续的内存。

### 【语法】

```

1 bm_status_t bm_image_alloc_contiguous_mem_heap_mask(
2     int         image_num,
3     bm_image   *images,
4     int         heap_mask
5 );

```

### 【参数】

表 4.12: bm\_image\_alloc\_contiguous\_mem\_heap\_mask  
参数表

参数名称	输入/输出	描述
image_num	输入	待分配内存的 image 个数。
*images	输入	待分配内存的 image 的指针。
heap_mask	输入	选择一个或多个 heap id 的掩码，每一个 bit 表示一个 heap id 的是否有效，1 表示可以在该 heap 上分配，0 则表示不可以。最低位表示 heap0，以此类推。比如 heap_mask=0b10 则表示指定在 heap1 上分配空间，heap_mask=0b101 则表示指定在 heap0 或者 heap2 上分配空间。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

### 【注意事项】

1. image\_num 应该大于 0, 否则将返回错误。
2. 如传入的 image 已分配或者 attach 过内存, 应先 detach 已有内存, 否则将返回失败。
3. 所有待分配的 image 应该尺寸相同, 否则将返回错误。
4. 当希望 destory 的 image 是通过调用本 api 所分配的内存时, 应先调用 bm\_image\_free\_contiguous\_mem 将分配内存释放, 再用 bm\_image\_destroy 来实现 destory image
5. bm\_image\_alloc\_contiguous\_mem 与 bm\_image\_free\_contiguous\_mem 应成对使用。

## 4.13 bm\_image\_free\_contiguous\_mem

### 【描述】

释放通过 bm\_image\_alloc\_contiguous\_mem 所分配的多个 image 中的连续内存。

### 【语法】

```
1 bm_status_t bm_image_free_contiguous_mem(  
2     int image_num,  
3     bm_image *images  
4 );
```

### 【参数】

表 4.13: bm\_image\_free\_contiguous\_mem 参数表

参数名称	输入/输出	描述
image_num	输入	待释放内存的 image 个数。
*images	输入	待释放内存的 image 的指针。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

## 【注意事项】

1. image\_num 应该大于 0，否则将返回错误。
2. 所有待释放的 image 应该尺寸相同。
3. bm\_image\_alloc\_contiguous\_mem 与 bm\_image\_free\_contiguous\_mem 应成对使用。bm\_image\_free\_contiguous\_mem 所要释放的内存必须是通过 bm\_image\_alloc\_contiguous\_mem 所分配的。
4. 应先调用 bm\_image\_free\_contiguous\_mem, 将 image 中内存释放, 再调 bm\_image\_destroy 去 destory image。

## 4.14 bm\_image\_attach\_contiguous\_mem

### 【描述】

将一块连续内存 attach 到多个 image 中。

### 【语法】

```
1 bm_status_t bm_image_attach_contiguous_mem(  
2     int image_num,  
3     bm_image *images,  
4     bm_device_mem_t dmem  
5 );
```

### 【参数】

表 4.14: bm\_image\_attach\_contiguous\_mem 参数表

参数名称	输入/输出	描述
image_num	输入	待 attach 内存的 image 个数。
*images	输入	待 attach 内存的 image 的指针。
dmem	输入	已分配好的 device memory 信息。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

## 【注意事项】

1. image\_num 应该大于 0，否则将返回错误。
2. 所有待 attach 的 image 应该尺寸相同，否则将返回错误。

## 4.15 bm\_image\_detach\_contiguous\_mem

### 【描述】

将一块连续内存从多个 image 中 detach。

### 【语法】

```
1 bm_status_t bm_image_detach_contiguous_mem(  
2     int image_num,  
3     bm_image *images  
4 );
```

### 【参数】

表 4.15: bm\_image\_detach\_contiguous\_mem 参数表

参数名称	输入/输出	描述
image_num	输入	待 attach 内存的 image 个数。
*images	输入	待 attach 内存的 image 的指针。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

### 【注意事项】

1. image\_num 应该大于 0，否则将返回错误。
2. 所有待 detach 的 image 应该尺寸相同，否则将返回错误。
3. bm\_image\_attach\_contiguous\_mem 与 bm\_image\_detach\_contiguous\_mem 应成对使用。bm\_image\_detach\_contiguous\_mem 所要 detach 的 device memory 必须是通过 bm\_image\_attach\_contiguous\_mem attach 到 image 中的。

## 4.16 bm\_image\_get\_contiguous\_device\_mem

### 【描述】

从多个内存连续的 image 中得到连续内存的 device memory 信息。

### 【语法】

```
1 bm_status_t bm_image_get_contiguous_device_mem(  
2     int image_num,  
3     bm_image *images,  
4     bm_device_mem_t *mem  
5 );
```

### 【参数】

表 4.16: bm\_image\_get\_contiguous\_device\_mem 参数表

参数名称	输入/输出	描述
image_num	输入	待获取信息的 image 个数。
*images	输入	待获取信息的 image 指针。
*mem	输出	得到的连续内存的 device memory 信息。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

### 【注意事项】

1. image\_num 应该大于 0，否则将返回错误。
2. 所填入的 image 应该尺寸相同，否则将返回错误。
3. 所填入的 image 必须是内存连续的，否则返回错误。
4. 所填入的 image 内存必须是通过 bm\_image\_alloc\_contiguous\_mem 或者 bm\_image\_attach\_contiguous\_mem 获得。

## 4.17 bm\_image\_get\_format\_info

### 【描述】

该接口用于获取 bm\_image 的一些信息。

### 【语法】

```
1 bm_status_t bm_image_get_format_info(  
2     bm_image_t* src,  
3     bm_image_format_info_t* info  
4 );
```

### 【参数】

表 4.17: bm\_image\_get\_format\_info 参数表

参数名称	输入/输出	描述
*src	输入	所要获取信息的目标 bm_image。
*info	输出	保存所需信息的数据结构，返回给用户，具体内容见下面的数据结构说明。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

### 【数据类型说明】

```
1 typedef struct bm_image_format_info {  
2     int plane_nb;  
3     bm_device_mem_t plane_data[8];  
4     int stride[8];  
5     int width;  
6     int height;  
7     bm_image_format_ext image_format;  
8     bm_image_data_format_ext data_type;  
9     bool default_stride;  
10 } bm_image_format_info_t;
```

- int plane\_nb  
该 image 的 plane 数量
- bm\_device\_mem\_t plane\_data[8]  
各个 plane 的 device memory
- int stride[8];  
各个 plane 的 stride 值
- int width;  
图片的宽度
- int height;  
图片的高度
- bm\_image\_format\_ext image\_format;  
图片的格式
- bm\_image\_data\_format\_ext data\_type;  
图片的存储数据类型
- bool default\_stride;  
是否使用了默认的 stride

#### 4.18 bm\_image\_get\_stride

##### 【描述】

该接口用于获取目标 bm\_image 的 stride 信息。

##### 【语法】

```
1 bm_status_t bm_image_get_stride(  
2     bm_image image,  
3     int *stride  
4 );
```

##### 【参数】

表 4.18: bm\_image\_get\_stride 参数表

参数名称	输入/输出	描述
image	输入	所要获取 stride 信息的目标 bm_image。
*stride	输出	存放各个 plane 的 stride 的指针。

### 【返回值】

函数调用成功时返回 BM\_SUCCESS。

## 4.19 bm\_image\_get\_plane\_num

### 【描述】

该接口用于获取目标 bm\_image 的 plane 数量。

### 【语法】

```
1 int bm_image_get_plane_num(bm_image image);
```

### 【参数】

表 4.19: bm\_image\_get\_plane\_num 参数表

参数名称	输入/输出	描述
image	输入	所要获取 plane 数量的目标 bm_image。

### 【返回值】

返回值即为目标 bm\_image 的 plane 数量。

## 4.20 bm\_image\_is\_attached

### 【描述】

该接口用于判断目标是否已经 attach 存储空间。

### 【语法】

```
1 bool bm_image_is_attached(bm_image image);
```

### 【参数】

表 4.20: bm\_image\_is\_attached 参数表

参数名称	输入/输出	描述
image	输入	所要判断是否 attach 存储空间的目标 bm_image。

### 【返回值】

若目标 bm\_image 已经 attach 存储空间则返回 true, 否则返回 false。

### 【注意事项】

1. 一般情况而言, 调用 bmcv api 要求输入 bm\_image 对象关联 device memory, 否则返回失败。而输出 bm\_image 对象如果未关联 device memory, 则会在内部调用 bm\_image\_alloc\_dev\_mem 函数, 内部申请内存。
2. bm\_image 调用 bm\_image\_alloc\_dev\_mem 所申请的内存都由内部自动管理, 在调用 bm\_image\_destroy、bm\_image\_detach 或者 bm\_image\_attach 其他 device memory 时自动释放, 无需调用者管理。相反, 如果 bm\_image\_attach 一块 device memory 时, 表示这块 memory 将由调用者自己管理。无论是 bm\_image\_destroy、bm\_image\_detach, 或者再调用 bm\_image\_attach 其他 device memory, 均不会释放, 需要调用者手动释放。

## 4.21 bm\_image\_get\_handle

### 【描述】

该接口用于通过 bm\_image 获取句柄 handle。

### 【语法】

```
1 bm_handle_t bm_image_get_handle(bm_image* image);
```

### 【参数】

表 4.21: bm\_image\_get\_handle 参数表

参数名称	输入/输出	描述
image	输入	所要获取 handle 的目标 bm_image。

### 【返回值】

返回值即为目标 bm\_image 的句柄 handle。

## 4.22 bm\_image\_write\_to\_bmp

### 【描述】

该接口用于将 bm\_image 的数据保存成 bmp 格式存到文件中。

### 【语法】

```
1 bm_status_t bm_image_write_to_bmp(bm_image image, const char *filename);
```

### 【参数】

表 4.22: bm\_image\_write\_to\_bmp 参数表

参数名称	输入/输出	描述
image	输入	目标 bm_image。
filename	输入	保存 bmp 文件的名称。

### 【返回值】

成功则返回 BM\_SUCCESS。

### 【注意事项】

1. 该 api 要求输入 bm\_image 对象关联 device memory，否则返回失败。

BMCV 还提供类似接口读入或写出 bm\_image 的数据。如下：

```
1 void bm_read_bin(bm_image src, const char *input_name);
2 void bm_read_compact_bin(bm_image src, const char *input_name);
3
4 void bm_write_bin(bm_image dst, const char *output_name);
5 void bm_write_compact_bin(bm_image dst, const char *output_name);
```

其中，bm\_read\_bin 和 bm\_write\_bin 可以读入和写出 raw 数据文件，读入写出数据按照每行间距为 bm\_image 中各 plane 的 stride 来进行。(若 stride 大于默认 stride，可能会读取或写出无效数据)

bm\_read\_compact\_bin 和 bm\_write\_compact\_bin 适合读入和写出紧凑排列的 raw 数据文件，读入写出数据按照每行间距为 bm\_image 中各 plane 的默认 stride 来进行。

## 5 BMCV API

本章介绍 BMCV 常用 API

### 5.1 bmcv\_base64\_enc(dec)

#### 【描述】

base64 网络传输中常用的编码方式，利用 64 个常用字符来对 6 位二进制数编码。

#### 【语法】

```

1 bm_status_t bmcv_base64_enc(bm_handle_t handle,
2     bm_device_mem_t src,
3     bm_device_mem_t dst,
4     unsigned long len[2])
5
6 bm_status_t bmcv_base64_dec(bm_handle_t handle,
7     bm_device_mem_t src,
8     bm_device_mem_t dst,
9     unsigned long len[2])

```

## 【参数】

表 5.1: bmcv\_base64 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
src	输入	输入字符串所在地址，类型为 bm_device_mem_t。需要调用 bm_mem_from_system() 将数据地址转化成转化为 bm_device_mem_t 所对应的结构。
dst	输出	输出字符串所在地址，类型为 bm_device_mem_t。需要调用 bm_mem_from_system() 将数据地址转化成转化为 bm_device_mem_t 所对应的结构。
len[2]	输出	进行 base64 编码或解码的长度，单位是字节。其中 len[0] 代表输入长度，需要调用者给出。而 len[1] 为输出长度，由 api 计算后给出。

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 代码示例：

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

```

(续下页)

(接上页)

```

int main() {
    int original_len = (rand() % 134217728) + 1; //128M
    int encoded_len = (original_len + 2) / 3 * 4;
    char* src = (char*)malloc((original_len + 3) * sizeof(char));
    char* dst = (char*)malloc((encoded_len + 3) * sizeof(char));
    for (int j = 0; j < original_len; j++) {
        src[j] = (char)((rand() % 256) + 1);
    }
    bm_handle_t handle;
    int ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(1);
    }
    unsigned long lenth[2];
    lenth[0] = (unsigned long)original_len;

    bmcv_base64_enc(handle, bm_mem_from_system(src), bm_mem_from_
    ↵system(dst), lenth);
    bmcv_base64_dec(handle, bm_mem_from_system(dst), bm_mem_from_
    ↵system(src), lenth);

    bm_dev_free(handle);
    free(src);
    free(dst);
    return 0;
}

```

## 5.2 bmcv\_as\_strided

该接口可以根据现有矩阵以及给定的步长来创建一个视图矩阵。

**接口形式：**

```

bm_status_t bmcv_as_strided(
    bm_handle_t handle,
    bm_device_mem_t input,
    bm_device_mem_t output,
    int input_row,
    int input_col,
    int output_row,
    int output_col,
    int row_stride,
    int col_stride);

```

**参数说明：**

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_device_mem_t input`  
输入参数。存放输入矩阵 `input` 数据的设备内存地址。
- `bm_device_mem_t output`  
输入参数。存放输出矩阵 `output` 数据的设备内存地址。
- `int input_row`  
输入参数。输入矩阵 `input` 的行数。
- `int input_col`  
输入参数。输入矩阵 `input` 的列数。
- `int output_row`  
输入参数。输出矩阵 `output` 的行数。
- `int output_col`  
输入参数。输出矩阵 `output` 的列数。
- `int row_stride`  
输入参数。输出矩阵行之间的步长。
- `int col_stride`  
输入参数。输出矩阵列之间的步长。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项:

1. 该接口可通过设置环境变量启用双核计算，运行程序前：`export TPU_CORES=2` 或 `export TPU_CORES=both` 即可。

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    int input_row = 5;
    int input_col = 5;
```

(续下页)

(接上页)

```
int output_row = 3;
int output_col = 3;
int row_stride = 1;
int col_stride = 2;
int input_size = input_row * input_col;
int output_size = output_row * output_col;
int ret = 0;
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);

float* input_data = (float*)malloc(input_size * sizeof(float));
float* tpu_output = (float*)malloc(output_size * sizeof(float));

for (int i = 0; i < input_size; i++) {
    input_data[i] = (float)rand() / (float)RAND_MAX * 100;
}

memset(tpu_output, 0, output_size * sizeof(float));

bm_device_mem_t input_dev_mem, output_dev_mem;
ret = bm_malloc_device_byte(handle, &input_dev_mem, input_size * [F
    ↵sizeof(float)];
ret = bm_malloc_device_byte(handle, &output_dev_mem, output_size * [F
    ↵sizeof(float));

ret = bm_memcpy_s2d(handle, input_dev_mem, input_data);

ret = bmcv_as_strided(handle, input_dev_mem, output_dev_mem, input_row, \
    input_col, output_row, output_col, row_stride, col_stride);

ret = bm_memcpy_d2s(handle, tpu_output, output_dev_mem);

bm_free_device(handle, input_dev_mem);
bm_free_device(handle, output_dev_mem);

free(input_data);
free(tpu_output);
bm_dev_free(handle);
return ret;
}
```

### 5.3 bmcv\_attribute\_filter\_topk

#### 描述:

海康-属性过滤排序算子。输入数据具有空间，时间，属性值和相似度这四个维度的属性，每个维度数据先按照特定的标准进行属性过滤，然后按照相似度值进行排序。

#### 语法:

```
1 bm_status_t bmcv_attribute_filter_topk(  
2     bm_handle_t handle,  
3     int data_type,  
4     int data_count,  
5     int max_space_points,  
6     int time_range_min,  
7     int time_range_max,  
8     int attr3_flag,  
9     int topk,  
10    float threshold,  
11    int8_t attr_mask,  
12    bm_device_mem_t space_bits_dev_mem,  
13    bm_device_mem_t space_points_dev_mem,  
14    bm_device_mem_t time_points_dev_mem,  
15    bm_device_mem_t attributes_dev_mem,  
16    bm_device_mem_t similarity_data_dev_mem,  
17    bm_device_mem_t filtered_idx_dev_mem,  
18    bm_device_mem_t filtered_similarity_data_dev_mem,  
19    bm_device_mem_t topk_idx_dev_mem,  
20    bm_device_mem_t topk_data_dev_mem);
```

#### 参数:

表 5.2: bmcv\_attribute\_filter\_topk 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
data_type	输入	输入数据类型。
data_count	输入	输入数据数量。
max_space_points	输入	最大空间点个数上限。
time_range_min	输入	时间范围下限。
time_range_max	输入	时间范围上限。
attr3_flag	输入	属性值过滤标志位。
topk	输入	排序并获取到的最大或最小输出数据的个数。
threshold	输入	相似度过滤阈值。
attr_mask	输入	属性掩码。
space_bits_dev_mem	输入	空间位图数据的设备地址。
space_points_dev_mem	输入	空间点数据的设备地址。
time_points_dev_mem	输入	时间点数据的设备地址。
atributes_dev_mem	输入	属性值数据的设备地址。
similarity_data_dev_mem	输入	相似度数据的设备地址。
filtered_idx_dev_mem	输入	用于暂存过滤后索引数据的设备地址。
filtered_similarity_data	输入	用于暂存过滤后相似度数据的设备地址。
topk_idx_dev_mem	输出	最大或最小 topk 个输出索引数据的设备地址。
topk_data_dev_mem	输出	最大或最小 topk 个输出相似度数据的设备地址。

**返回值：**

该函数成功调用时，返回 BM\_SUCCESS，调用失败时，返回 BM\_ERR\_PARAM 等错误码。

**注意事项：**

1. data\_type 支持 fp32 和 uint32 两种数据类型，其中 fp32 数据将进行降序排序，uint32 数据类型将进行升序排序。
2. data\_count 需是 1024 的倍数，理论最大支持数据量受内存容量限制，目前压测过的范围为 1024-4096000。
3. max\_space\_points 设定范围为 -1 和 (0,300000]，其中设定为 -1 时表示空间属性不进行过滤。
4. time\_range\_min 和 time\_range\_max 的支持范围为 (0,315360000)，其中 time\_range\_min 小于等于 time\_range\_max。
5. attr3\_flag 支持 0 和 1 两种模式，0 表示属性值和属性掩码按位与操作后，数值不等于属性掩码，该数据被过滤；1 表示属性值和属性掩码按位与操作后，数值小于等于 0，该数据被过滤。

6. topk 值支持 5, 10, 100, 1000, 10000, 20000, 50000, 且 topk 值须小于等于 data\_count。
7. threshold 相似度阈值默认为 float 类型, 如进行 uint32 类型数据过滤, 接口内会将 threshold 数值强制转为 uint32 类型。
8. attr\_mask 支持范围为 [-128,127]。

#### 代码示例:

```

1 #include "stdio.h"
2 #include "stdlib.h"
3 #include <time.h>
4 #include "bmvc_api_ext_c.h"
5 #include <stdint.h>
6 #ifdef __linux
7 #include <sys/time.h>
8 #else
9 #include <windows.h>
10 #include "time.h"
11 #endif
12
13 #define TIME_COST_US(start, end) ((end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - [F]
14 → start.tv_usec))
15 #define MAX_BITS 128
16 #define MAX_HD MAX_BITS
17 #define TYPICAL_HD 30
18 uint32_t generate_hamming_distance(int max_bits) {
19     uint32_t hd = 0;
20     for(int i = 0; i < max_bits; i++) {
21         if((float)rand() / RAND_MAX < 0.3) {
22             hd++;
23         }
24     }
25     return hd;
26 }
27 void generate_hamming_data(uint32_t* out, size_t count) {
28     for(size_t i = 0; i < count; i++) {
29         out[i] = generate_hamming_distance(MAX_BITS);
30         if(i == 0) out[i] = 0;
31         if(i == 1) out[i] = MAX_HD;
32         if(i == 2) out[i] = TYPICAL_HD-1;
33         if(i == 3) out[i] = TYPICAL_HD+1;
34         if(i % 10000 == 0) out[i] = UINT32_MAX;
35     }
36 }
37 void generate_test_data(size_t data_count, int max_space_points, int64_t** space_bits, int32_t
38 → t** time_points, int32_t** space_points,
39 → int8_t** attributes, int active_points_num, float** similarity_data_fp32,[F]
40 → uint32_t** similarity_data_u32) {
41     const size_t bitmap_size = (max_space_points + 64 - 1) / 64;
42     *space_bits = (int64_t*)calloc(bitmap_size, sizeof(int64_t));

```

(续下页)

(接上页)

```

42 *time_points = (int32_t*)malloc(data_count * sizeof(int32_t));
43 *space_points = (int32_t*)malloc(data_count * sizeof(int32_t));
44 *attributes = (int8_t*)malloc(data_count * sizeof(int8_t));
45 *similarity_data_fp32 = (float*)malloc(data_count * sizeof(float));
46 *similarity_data_u32 = (uint32_t*)malloc(data_count * sizeof(uint32_t));
47
48 //Bitmap initialization
49 int32_t* valid_points = (int32_t*)malloc(active_points_num * sizeof(int32_t));
50 for (size_t i = 0; i < active_points_num; ++i) {
51     valid_points[i] = rand() % max_space_points;
52 }
53
54 // Fisher-Yates Shuffle
55 for (int i = active_points_num - 1; i > 0; --i) {
56     int j = rand() % (i + 1);
57     int32_t temp = valid_points[i];
58     valid_points[i] = valid_points[j];
59     valid_points[j] = temp;
60 }
61
62 for (size_t i = 0; i < active_points_num; ++i) {
63     int32_t point = valid_points[i];
64     int group = point / 64;
65     int offset = point % 64;
66     (*space_bits)[group] |= (1ULL << offset);
67 }
68 free(valid_points);
69 generate_hamming_data(*similarity_data_u32, data_count);
70 //Fill in the attributes of each feature
71 for (size_t i = 0; i < data_count; ++i) {
72     (*time_points)[i] = rand() % 315360001;
73     (*space_points)[i] = rand() % 300001;
74     (*attributes)[i] = (int8_t)(rand() % 256 - 128);
75     (*similarity_data_fp32)[i] = (float)rand() / RAND_MAX * 3000000.0f - 1500000.0f;
76 }
77 }
78
79 void convert_int64_to_int32(const int64_t* input, int32_t* output, size_t n) {
80     for (size_t i = 0; i < n; i++) {
81         uint64_t value = (uint64_t)input[i]; //Use unsigned to ensure correct shifting
82         output[2 * i] = (int32_t)(value & 0xFFFFFFFF);
83         output[2 * i + 1] = (int32_t)((value >> 32) & 0xFFFFFFFF);
84     }
85 }
86
87 int main(int argc, char* argv[]) {
88     int data_type = 5 + 3 * (rand() % 2);
89     int topk_num[] = {5, 10, 100, 1000, 10000, 50000};
90     int size = sizeof(topk_num) / sizeof(topk_num[0]);
91     int rand_num = rand() % size;
92     int topk = topk_num[rand_num];

```

(续下页)

(接上页)

```

93 int data_count = 1024 * (1 + rand() % 4000);
94 int max_space_points = 300000;
95 float threshold = 3.0;
96 int time_range_min = 0;
97 int time_range_max = rand() % 315360000;
98 int attr3_flag = (int)rand() % 2;
99 int8_t attr_mask = (int8_t)rand();
100 int64_t* space_bits = NULL;
101 int active_points_num = 5000;
102 int32_t* time_points = NULL;
103 int32_t* space_points = NULL;
104 int8_t* attributes = NULL;
105 float* similarity_data_fp32 = NULL;
106 uint32_t* similarity_data_u32 = NULL;
107 bm_handle_t handle;
108 bm_status_t ret = bm_dev_request(&handle, 0);
109 if (ret != BM_SUCCESS) {
110     printf("Create bm handle failed. ret = %d\n", ret);
111     return -1;
112 }
113
114 generate_test_data(data_count, max_space_points, &space_bits, &time_points, &space_
→points,
115                     &attributes, active_points_num, &similarity_data_fp32, &similarity_data_
→u32);
116 void* topk_data_tpu = (void*)malloc(topk * 4);
117 int* topk_idx_tpu = (int*)malloc(topk * sizeof(int));
118 bm_status_t bm_ret;
119 bm_device_mem_t space_bits_dev_mem;
120 bm_device_mem_t space_points_dev_mem;
121 bm_device_mem_t time_points_dev_mem;
122 bm_device_mem_t attributes_dev_mem;
123 bm_device_mem_t similarity_data_dev_mem;
124 bm_device_mem_t filtered_similarity_data_dev_mem;
125 bm_device_mem_t filtered_idx_dev_mem;
126 bm_device_mem_t topk_data_dev_mem;
127 bm_device_mem_t topk_idx_dev_mem;
128 size_t type_bytes = (data_type == 5 ? sizeof(float) : sizeof(uint32_t));
129 const size_t bitmap_size = (max_space_points + 64 - 1) / 64;
130 struct timeval t1, t2;
131 int per_row_num = 1024;
132 int per_row_num_bits_map = per_row_num / 32;
133 int space_bits_num = ((bitmap_size * 2 + per_row_num_bits_map - 1) / per_row_num_
→bits_map) * per_row_num_bits_map;
134 int32_t* space_bits_int32 = (int32_t*)calloc(space_bits_num * sizeof(int32_t), sizeof(int32_
→t));
135 convert_int64_to_int32(space_bits, space_bits_int32, bitmap_size);
136 bm_malloc_device_byte(handle, &space_bits_dev_mem, space_bits_num * sizeof(int32_t));
137 bm_malloc_device_byte(handle, &space_points_dev_mem, data_count * sizeof(int32_t));
138 bm_malloc_device_byte(handle, &time_points_dev_mem, data_count * sizeof(int32_t));
139 bm_malloc_device_byte(handle, &attributes_dev_mem, data_count * sizeof(int8_t));

```

(续下页)

(接上页)

```

140     bm_malloc_device_byte(handle, &similarity_data_dev_mem, data_count * type_bytes);
141     bm_malloc_device_byte(handle, &filtered_similarity_data_dev_mem, data_count * type_
142     ↪bytes);
142     bm_malloc_device_byte(handle, &filtered_idx_dev_mem, data_count * sizeof(int));
143     bm_malloc_device_byte(handle, &topk_idx_dev_mem, topk * sizeof(unsigned int));
144     bm_malloc_device_byte(handle, &topk_data_dev_mem, topk * type_bytes);
145     bm_memcpy_s2d(handle, space_bits_dev_mem, space_bits_int32);
146     bm_memcpy_s2d(handle, space_points_dev_mem, space_points);
147     bm_memcpy_s2d(handle, time_points_dev_mem, time_points);
148     bm_memcpy_s2d(handle, attributes_dev_mem, attributes);
149     if (data_type == 5) {
150         bm_memcpy_s2d(handle, similarity_data_dev_mem, similarity_data_fp32);
151     } else {
152         bm_memcpy_s2d(handle, similarity_data_dev_mem, similarity_data_u32);
153     }
154     gettimeofday(&t1, NULL);
155     bm_ret = bmcv_attribute_filter_topk(handle, data_type, data_count, max_space_points,F
156     ↪time_range_min,
156     ↪time_range_max, attr3_flag, topk, threshold, attr_mask, space_
157     ↪bits_dev_mem,
157     ↪space_points_dev_mem, time_points_dev_mem, attributes_dev_
158     ↪mem,
158     ↪similarity_data_dev_mem, filtered_idx_dev_mem, filtered_
159     ↪similarity_data_dev_mem,
159     ↪topk_idx_dev_mem, topk_data_dev_mem);
160     if(bm_ret != BM_SUCCESS) {
161         printf("bmcv_attribute_filter_topk failed\n");
162     }
163     gettimeofday(&t2, NULL);
164     printf("attribute_filter_topk TPU using time = %ld(us)\n", TIME_COST_US(t1, t2));
165     bm_memcpy_d2s(handle, topk_data_tpu, topk_data_dev_mem);
166     bm_memcpy_d2s(handle, topk_idx_tpu, topk_idx_dev_mem);
167     bm_free_device(handle, space_bits_dev_mem);
168     bm_free_device(handle, space_points_dev_mem);
169     bm_free_device(handle, time_points_dev_mem);
170     bm_free_device(handle, attributes_dev_mem);
171     bm_free_device(handle, similarity_data_dev_mem);
172     bm_free_device(handle, topk_data_dev_mem);
173     bm_free_device(handle, topk_idx_dev_mem);
174     free(space_bits_int32);
175     free(space_bits);
176     free(time_points);
177     free(space_points);
178     free(attributes);
179     free(similarity_data_fp32);
180     free(similarity_data_u32);
181     bm_dev_free(handle);
182     return ret;
183 }
```

## 5.4 bmcv\_batch\_topk

### 描述:

计算每个 batch 中最大或最小的 k 个数，并返回 index。

### 语法:

```

1  bm_status_t bmcv_batch_topk(
2      bm_handle_t handle,
3      bm_device_mem_t src_data_addr,
4      bm_device_mem_t src_index_addr,
5      bm_device_mem_t dst_data_addr,
6      bm_device_mem_t dst_index_addr,
7      bm_device_mem_t buffer_addr,
8      int k,
9      int batch,
10     int *per_batch_cnt,
11     int src_batch_stride,
12     bool src_index_valid,
13     bool same_batch_cnt,
14     bool descending);

```

### 参数:

表 5.3: bmcv\_batch\_topk 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获得。
src_data_addr	输入	输入数据的地址信息。
src_index_addr	输入	输入数据索引的地址信息，当 src_index_valid 为 true 时，设置该参数。
dst_data_addr	输出	输出数据的地址信息。
dst_index_addr	输出	输出数据索引的地址信息。
buffer_addr	输入	缓冲区地址信息。
k	输入	取每个 batch 中最大或最小数的数量，取值范围为 1-100。
batch	输入	batch 的数量，取值范围为 1-20。
*per_batch_cnt	输入	每个 batch 的数据数量，取值范围为 1-100000。
src_batch_stride	输入	两个 batch 之间的间隔数。
src_index_valid	输入	如果为 true，则使用 src_index，否则使用自动生成的 index。
same_batch_cnt	输入	每个 batch 数据是否相同。
descending	输入	对数据进行升序或者降序。

### 返回值:

该函数成功调用时，返回 BM\_SUCCESS。

### 格式支持:

该接口目前仅支持 float32 类型数据。

### 代码示例：

```

1 #include <stdio.h>
2 #include "stdlib.h"
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5
6
7 int main() {
8     int batch_num = 10 + rand() % 999990;
9     int k = 1 + rand() % 100;
10    k = k < batch_num ? k : batch_num;
11    int descending = rand() % 2;
12    int batch = rand() % 32 + 1;
13    bool bottom_index_valid = true;
14
15    printf("batch_topk_params: \n");
16    printf("batch_num = %d, k = %d, descending = %d, batch = %d, bottom_index_valid = %d \n"
17        , batch_num, k, descending, batch, bottom_index_valid);
18    bm_handle_t handle;
19    bm_status_t ret = bm_dev_request(&handle, 0);
20    if (ret != BM_SUCCESS) {
21        printf("Create bm handle failed. ret = %d\n", ret);
22        return -1;
23    }
24
25    int batch_stride = batch_num;
26
27    float* bottom_data = (float*)malloc(batch * batch_stride * sizeof(float));
28    int* bottom_index = (int*)malloc(batch * batch_stride * sizeof(int));
29    float* top_data = (float*)malloc(batch * batch_stride * sizeof(float));
30    int* top_index = (int*)malloc(batch * batch_stride * sizeof(int));
31    float* top_data_ref = (float*)malloc(batch * k * sizeof(float));
32    int* top_index_ref = (int*)malloc(batch * k * sizeof(int));
33    float* buffer = (float*)malloc(3 * batch_stride * sizeof(float));
34
35    for(int i = 0; i < batch; i++){
36        for(int j = 0; j < batch_num; j++){
37            bottom_data[i * batch_stride + j] = rand() % 10000 * 1.0f;
38            bottom_index[i * batch_stride + j] = i * batch_stride + j;
39        }
40
41    ret = bmvc_batch_topk(handle, bm_mem_from_system((void*)bottom_data), bm_mem_
42        _from_system((void*)bottom_index),
43        bm_mem_from_system((void*)top_data), bm_mem_from_system((void*)top_
44        _index),
45        bm_mem_from_system((void*)buffer), bottom_index_valid, k, batch, &batch_
46        _num,
47        true, batch_stride, descending);

```

(续下页)

(接上页)

```
45     bm_dev_free(handle);  
46  
47     free(bottom_data);  
48     free(bottom_index);  
49     free(top_data);  
50     free(top_data_ref);  
51     free(top_index);  
52     free(top_index_ref);  
53     free(buffer);  
54  
55     return ret;  
56 }  
57 }
```

## 5.5 bmcv\_calc\_hist

### 5.5.1 直方图

接口形式：

```
bm_status_t bmcv_calc_hist(  
    bm_handle_t handle,  
    bm_device_mem_t input,  
    bm_device_mem_t output,  
    int C,  
    int H,  
    int W,  
    const int* channels,  
    int dims,  
    const int* histSizes,  
    const float* ranges,  
    int inputDtype);
```

参数说明：

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_device_mem_t input`  
输入参数。该 device memory 空间存储了输入数据，类型可以是 `float32` 或者 `uint8`，由参数 `inputDtype` 决定。其大小为  $C * H * W * \text{sizeof}(Dtype)$ 。
- `bm_device_mem_t output`  
输出参数。该 device memory 空间存储了输出结果，类型为 `float`，其大小为  $\text{histSizes}[0] * \text{histSizes}[1] * \dots * \text{histSizes}[n] * \text{sizeof}(\text{float})$ 。
- `int C`

输入参数。输入数据的通道数量。

- int H

输入参数。输入数据每个通道的高度。

- int W

输入参数。输入数据每个通道的宽度。

- const int\* channels

输入参数。需要计算直方图的 channel 列表，其长度为 dims，每个元素的值必须小于 C。

- int dims

输入参数。输出的直方图维度，要求不大于 3。

- const int\* histSizes

输入参数。对应每个 channel 统计直方图的份数，其长度为 dims。

- const float\* ranges

输入参数。每个通道参与统计的范围，其长度为 2 \* dims。

- int inputDtype

输入参数。输入数据的类型：0 表示 float，1 表示 uint8。

#### 返回值说明：

- BM\_SUCCESS: 成功

- 其他: 失败

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <math.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

int main() {
    int H = 1024;
    int W = 1024;
    int C = 3;
    int dim = 3;
    int data_type = 0; // 0: float; 1: uint8

    int channels[3] = {0, 1, 2};
    int histSizes[3] = {32, 32, 32};
    float ranges[6] = {0, 256, 0, 256, 0, 256};
```

(续下页)

(接上页)

```
int totalHists = 1;
int ret = 0;
int i, j;
int Num = 0;
bm_handle_t handle;

ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
    printf("bm_dev_request failed. ret = %d\n", ret);
    return -1;
}
for (i = 0; i < dim; i++) {
    totalHists *= histSizes[i];
}
float* output_tpu = (float*)malloc(totalHists * sizeof(float));
memset(output_tpu, 0, totalHists * sizeof(float));

float* input_host = (float*)malloc(C * H * W * sizeof(float));

// randomly initialize input_host
for (i = 0; i < C; i++) {
    for (j = 0; j < H * W; j++) {
        Num = (int)ranges[2*C-1];
        input_host[i * H * W + j] = (float)(rand() % Num);
    }
}

bm_device_mem_t input, output;
ret = bm_malloc_device_byte(handle, &output, totalHists * sizeof(float));
ret = bm_malloc_device_byte(handle, &input, C * H * W * sizeof(float));
ret = bm_memcpy_s2d(handle, input, input_host);
ret = bmcv_calc_hist(handle, input, output, C, H, W, channels, dim, histSizes, ranges, data_type);

ret = bm_memcpy_d2s(handle, output_tpu, output);
if (ret != BM_SUCCESS) {
    printf("test calc hist failed. ret = %d\n", ret);
    bm_free_device(handle, input);
}

free(input_host);
free(output_tpu);
bm_dev_free(handle);
return ret;
}
```

### 5.5.2 带权重的直方图

接口形式：

```
bm_status_t bmcv_calc_hist_with_weight(  
    bm_handle_t handle,  
    bm_device_mem_t input,  
    bm_device_mem_t output,  
    const float* weight,  
    int C,  
    int H,  
    int W,  
    const int* channels,  
    int dims,  
    const int* histSizes,  
    const float* ranges,  
    int inputDtype);
```

参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t input  
输入参数。该 device memory 空间存储了输入数据，其大小为  $C * H * W * \text{sizeof}(Dtype)$ 。
- bm\_device\_mem\_t output  
输出参数。该 device memory 空间存储了输出结果，类型为 float，其大小为  $\text{histSizes}[0] * \text{histSizes}[1] * \dots * \text{histSizes}[n] * \text{sizeof}(float)$ 。
- const float\* weight  
输入参数。channel 内部每个元素在统计直方图时的权重，其大小为  $H * W * \text{sizeof}(float)$ ，如果所有值全为 1 则与普通直方图功能相同。
- int C  
输入参数。输入数据的通道数量。
- int H  
输入参数。输入数据每个通道的高度。
- int W  
输入参数。输入数据每个通道的宽度。
- const int\* channels  
输入参数。需要计算直方图的 channel 列表，其长度为 dims，每个元素的值必须小于 C。
- int dims  
输入参数。输出的直方图维度，要求不大于 3。

- const int\* histSizes  
输入参数。对应每个 channel 统计直方图的份数，其长度为 dims。
- const float\* ranges  
输入参数。每个通道参与统计的范围，其长度为 2 \* dims。
- int inputDtype  
输入参数。输入数据的类型：0 表示 float，1 表示 uint8。

#### 返回值说明：

- BM\_SUCCESS: 成功
- 其他: 失败

### 5.6 bmcv\_cmulp

该接口实现复数乘法运算，运算公式如下：

$$\text{outputReal} + \text{outputImag} \times i = (\text{inputReal} + \text{inputImag} \times i) \times (\text{pointReal} + \text{pointImag} \times i)$$

$$\text{outputReal} = \text{inputReal} \times \text{pointReal} - \text{inputImag} \times \text{pointImag}$$

$$\text{outputImag} = \text{inputReal} \times \text{pointImag} + \text{inputImag} \times \text{pointReal}$$

其中， $i$  是虚数单位，满足公式  $i^2 = -1$ .

#### 接口形式：

```
bm_status_t bmcv_cmulp(
    bm_handle_t handle,
    bm_device_mem_t inputReal,
    bm_device_mem_t inputImag,
    bm_device_mem_t pointReal,
    bm_device_mem_t pointImag,
    bm_device_mem_t outputReal,
    bm_device_mem_t outputImag,
    int batch,
    int len);
```

#### 参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t inputReal  
输入参数。存放输入实部的 device 地址。

- `bm_device_mem_t inputImg`  
输入参数。存放输入虚部的 device 地址。
- `bm_device_mem_t pointReal`  
输入参数。存放另一个输入实部的 device 地址。
- `bm_device_mem_t pointImg`  
输入参数。存放另一个输入虚部的 device 地址。
- `bm_device_mem_t outputReal`  
输出参数。存放输出实部的 device 地址。
- `bm_device_mem_t outputImg`  
输出参数。存放输出虚部的 device 地址。
- `int batch`  
输入参数。batch 的数量。
- `int len`  
输入参数。一个 batch 中复数的数量。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项:

1. 数据类型仅支持 float。
2. 该接口可通过设置环境变量启用双核计算，运行程序前：`export TPU_CORES=2` 或 `export TPU_CORES=both` 即可。

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    int L = 1 + rand() % 4096;
    int batch = 1 + rand() % 1980;
    int ret = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }
}
```

(续下页)

(接上页)

```

float* XRHost = (float*)malloc(L * batch * sizeof(float));
float* XIHost = (float*)malloc(L * batch * sizeof(float));
float* PRHost = (float*)malloc(L * sizeof(float));
float* PIHost = (float*)malloc(L * sizeof(float));
float* tpu_YR = (float*)malloc(L * batch * sizeof(float));
float* tpu_YI = (float*)malloc(L * batch * sizeof(float));
int i;

for (i = 0; i < L * batch; ++i) {
    XRHost[i] = rand() % 5 - 2;
    XIHost[i] = rand() % 5 - 2;
}
for (i = 0; i < L; ++i) {
    PRHost[i] = rand() % 5 - 2;
    PIHost[i] = rand() % 5 - 2;
}

bm_device_mem_t XRDev, XIDev, PRDev, PIDev, YRDev, YIDev;

ret = bm_malloc_device_byte(handle, &XRDev, L * batch * sizeof(float));
ret = bm_malloc_device_byte(handle, &XIDev, L * batch * sizeof(float));
ret = bm_malloc_device_byte(handle, &PRDev, L * sizeof(float));
ret = bm_malloc_device_byte(handle, &PIDev, L * sizeof(float));
ret = bm_malloc_device_byte(handle, &YRDev, L * batch * sizeof(float));
ret = bm_malloc_device_byte(handle, &YIDev, L * batch * sizeof(float));
ret = bm_memcpy_s2d(handle, XRDev, XRHost);
ret = bm_memcpy_s2d(handle, XIDev, XIHost);
ret = bm_memcpy_s2d(handle, PRDev, PRHost);
ret = bm_memcpy_s2d(handle, PIDev, PIHost);

ret = bmcv_cmulp(handle, XRDev, XIDev, PRDev, PIDev, YRDev, YIDev, batch, F
↪L);
if (ret != BM_SUCCESS) {
    printf("bmcv_cmulp failed. ret = %d\n", ret);
    return -1;
}

ret = bm_memcpy_d2s(handle, tpu_YR, YRDev);
ret = bm_memcpy_d2s(handle, tpu_YI, YIDev);

if (ret) {
    printf("the tpu cuml failed!, ret = %d\n", ret);
    return ret;
}

printf("test whether correct: XRHOST=%f, XIHOST=%f, PRHOST=%f, PIHOST=%
↪f, tpu_YR=%f, tpu_YI=%f\n", XRHost[0], XIHost[0], PRHost[0], PIHost[0], tpu_
↪_YR[0], tpu_YI[0]);

bm_free_device(handle, XRDev);
bm_free_device(handle, XIDev);

```

(续下页)

(接上页)

```
bm_free_device(handle, YRDev);
bm_free_device(handle, YIDev);
bm_free_device(handle, PRDev);
bm_free_device(handle, PIDev);
free(XRHost);
free(XIHost);
free(PRHost);
free(PIHost);
free(tpu_YR);
free(tpu_YI);

bm_dev_free(handle);
return ret;
}
```

## 5.7 bmcv\_distance

计算多维空间下多个点与特定一个点的欧式距离，前者坐标存放在连续的 device memory 中，而特定一个点的坐标通过参数传入。坐标值为 float 类型。

接口形式：

```
bm_status_t bmcv_distance(
    bm_handle_t handle,
    bm_device_mem_t input,
    bm_device_mem_t output,
    int dim,
    const float* pnt,
    int len);
```

参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄
- bm\_device\_mem\_t input  
输入参数。存放 len 个点坐标的 device 空间。其大小为 len\*dim\*sizeof(float)。
- bm\_device\_mem\_t output  
输出参数。存放 len 个距离的 device 空间。其大小为 len\*sizeof(float)。
- int dim  
输入参数。空间维度大小。
- const float\* pnt  
输入参数。特定一个点的坐标，长度为 dim。

- int len

输入参数。待求坐标的数量。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 示例代码

```
#include "bmvc_api_ext_c.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "test_misc.h"

#define DIM_MAX 8

enum op {
    FP32 = 0,
    FP16 = 1,
};

int main() {
    int dim = 1 + rand() % 8;
    int len = 1 + rand() % 40960;
    int ret = 0;
    enum op op = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }

    float pnt32[DIM_MAX] = {0};
    float* XHost = (float*)malloc(len * dim * sizeof(float));
    float* tpu_out = (float*)malloc(len * sizeof(float));
    fp16* XHost16 = (fp16*)malloc(len * dim * sizeof(fp16));
    fp16* tpu_out_fp16 = (fp16*)malloc(len * sizeof(fp16));
    fp16 pnt16[DIM_MAX] = {0};
    int round = 1;
    int i;

    printf("len = %d, dim = %d, op = %d\n", len, dim, op);

    for (i = 0; i < dim; ++i) {
        pnt32[i] = (rand() % 2 ? 1.f : -1.f) * (rand() % 100 + (rand() % 100) * 0.01);
        pnt16[i] = fp32tofp16(pnt32[i], round);
    }
}
```

(续下页)

(接上页)

```
for (i = 0; i < len * dim; ++i) {
    XHost[i] = (rand() % 2 ? 1.f : -1.f) * (rand() % 100 + (rand() % 100) * 0.01);
    XHost16[i] = fp32tofp16(XHost[i], round);
}
if (op == FP32) {
    enum bm_data_type_t dtype = DT_FP32;
    bm_device_mem_t XDev, YDev;

    ret = bm_malloc_device_byte(handle, &XDev, len * dim * sizeof(float));
    ret = bm_malloc_device_byte(handle, &YDev, len * sizeof(float));
    ret = bm_memcpy_s2d(handle, XDev, XHost);
    ret = bmcv_distance(handle, XDev, YDev, dim, pnt32, len, dtype);
    ret = bm_memcpy_d2s(handle, tpu_out, YDev);

    bm_free_device(handle, XDev);
    bm_free_device(handle, YDev);
} else {
    enum bm_data_type_t dtype = DT_FP16;
    bm_device_mem_t XDev, YDev;

    ret = bm_malloc_device_byte(handle, &XDev, len * dim * sizeof(float) / 2);
    ret = bm_malloc_device_byte(handle, &YDev, len * sizeof(float) / 2);
    ret = bm_memcpy_s2d(handle, XDev, XHost16);
    ret = bmcv_distance(handle, XDev, YDev, dim, (const void *)pnt16, len, dtype);
    ret = bm_memcpy_d2s(handle, tpu_out_fp16, YDev);

    bm_free_device(handle, XDev);
    bm_free_device(handle, YDev);

    for (i = 0; i < len; ++i) {
        tpu_out[i] = fp16tofp32(tpu_out_fp16[i]);
    }
}

free(XHost16);
free(tpu_out_fp16);
free(XHost);
free(tpu_out);

bm_dev_free(handle);
return ret;
}
```

## 5.8 bmcv\_feature\_match\_normalized

该接口用于将网络得到特征点（float 格式）与数据库中特征点（float 格式）进行比对，输出最佳匹配。

接口形式：

```
bm_status_t bmcv_feature_match_normalized(  
    bm_handle_t handle,  
    bm_device_mem_t input_data_global_addr,  
    bm_device_mem_t db_data_global_addr,  
    bm_device_mem_t db_feature_global_addr,  
    bm_device_mem_t output_similarity_global_addr,  
    bm_device_mem_t output_index_global_addr,  
    int batch_size,  
    int feature_size,  
    int db_size);
```

参数说明：

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_device_mem_t input_data_global_addr`  
输入参数。所要比对的特征点数据存储的地址。该数据按照 `batch_size * feature_size` 的数据格式进行排列。`batch_size`, `feature_size` 具体含义将在下面进行介绍。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。
- `bm_device_mem_t db_data_global_addr`  
输入参数。数据库的特征点数据存储的地址。该数据按照 `feature_size * db_size` 的数据格式进行排列。`feature_size`, `db_size` 具体含义将在下面进行介绍。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。
- `bm_device_mem_t db_feature_global_addr`  
输入参数。数据库的特征点的 `feature_size` 方向模的倒数的地址。该数据按照 `db_size` 的数据格式进行排列。
- `bm_device_mem_t output_similarity_global_addr`  
输出参数。每个 batch 得到的比对结果的值中最大值存储地址。该数据按照 `batch_size` 的数据格式进行排列。`batch_size` 具体含义将在下面进行介绍。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。
- `bm_device_mem_t output_index_global_addr`  
输出参数。每个 batch 得到的比对结果的在数据库中的序号的存储地址。如对于 batch 0，如果 `output_sorted_similarity_global_addr` 中 batch 0 的数据是由输入数据与数

据库的第 800 组特征点进行比对得到的，那么 output\_sorted\_index\_global\_addr 所在地址对应 batch 0 的数据为 800。output\_sorted\_similarity\_global\_addr 中的数据按照 batch\_size\_c 的数据格式进行排列。batch\_size 具体含义将在下面进行介绍。bm\_device\_mem\_t 为内置表示地址的数据类型，可以使用函数 bm\_mem\_from\_system(addr) 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- int batch\_size  
输入参数。待输入数据的 batch 个数，如输入数据有 4 组特征点，则该数据的 batch\_size 为 4。batch\_size 最大值不应超过 10。
- int feature\_size  
输入参数。每组数据的特征点个数。feature\_size 最大值不应该超过 1000。
- int db\_size  
输入参数。数据库中数据特征点的组数。db\_size 最大值不应该超过 90000。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 注意事项:

1. 输入数据和数据库中数据的数据类型为 float 类型。
2. 输出的比对结果数据类型为 float，输出的序号类型为 int。
3. 数据库中的数据在内存的排布为 feature\_size \* db\_size，因此需要将一组特征点进行转置之后再放入数据库中。
4. db\_feature\_global\_addr 模的倒数计算方法为:  $1 / \sqrt{y_1 * y_1 + y_2 * y_2 + \dots + y_n * y_n}$ ;

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

static int calc_sqrt_transposed(float** feature, int rows, int cols, float* db_feature)
{
    int i, j;
    float tmp;
    float result;

    for (i = 0; i < cols; ++i) {
        tmp = 0.f;
        for (j = 0; j < rows; ++j) {
            tmp += feature[j][i] * feature[j][i];
        }
        result = sqrt(tmp);
        db_feature[i] = result;
    }
}
```

(续下页)

(接上页)

```

        tmp += feature[j][i] * feature[j][i];
    }
    result = 1.f / sqrt(tmp);
    db_feature[i] = result;
}

return 0;
}

int main()
{
    int batch_size = rand() % 8 + 1;
    int feature_size = rand() % 1000 + 1;
    int db_size = (rand() % 90 + 1) * 1000;
    bm_handle_t handle;
    int ret = 0;

    ret = (int)bm_dev_request(&handle, 0);
    if (ret) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return ret;
    }

    float* input_data = (float*)malloc(sizeof(float) * batch_size * feature_size);
    float* db_data = (float*)malloc(sizeof(float) * db_size * feature_size);
    float* db_feature = (float*)malloc(sizeof(float) * db_size);
    float* output_similarity = (float*)malloc(sizeof(float) * batch_size); /*float*/
    int* output_index = (int*)malloc(sizeof(int) * batch_size);
    int i, j;

    float** db_content_vec = (float**)malloc(feature_size * sizeof(float*)); /*row = F
    ↪feature_size col = db_size*/
    for (i = 0; i < feature_size; ++i) {
        db_content_vec[i] = (float*)malloc(db_size * sizeof(float));
        for (j = 0; j < db_size; ++j) {
            db_content_vec[i][j] = rand() % 20 - 10;
        }
    }

    float** input_content_vec = (float**)malloc(batch_size * sizeof(float*)); /*row F
    ↪= batch_size col = feature_size*/
    for (i = 0; i < batch_size; ++i) {
        input_content_vec[i] = (float*)malloc(feature_size * sizeof(float));
        for (j = 0; j < feature_size; ++j) {
            input_content_vec[i][j] = rand() % 20 - 10;
        }
    }

    float** ref_res = (float**)malloc(sizeof(float) * batch_size); /* row = batch_
    ↪size col = db_size */

```

(续下页)

(接上页)

```
for (i = 0; i < batch_size; ++i) {
    ref_res[i] = (float*)malloc(db_size * sizeof(float));
}

for (i = 0; i < feature_size; ++i) {
    for (j = 0; j < db_size; ++j) {
        db_data[i * db_size + j] = db_content_vec[i][j];
    }
}

ret = calc_sqrt_transposed(db_content_vec, feature_size, db_size, db_feature);

for (i = 0; i < batch_size; i++) {
    for (j = 0; j < feature_size; j++) {
        input_data[i * feature_size + j] = input_content_vec[i][j];
    }
}

ret = bmcv_feature_match_normalized(handle, bm_mem_from_system(input_data),
                                     bm_mem_from_system(db_data),
                                     bm_mem_from_system(db_feature), bm_mem_from_system(output_similarity),
                                     bm_mem_from_system(output_index), batch_size, feature_size, db_size);

free(input_data);
free(db_data);
free(db_feature);
free(output_similarity);
free(output_index);
for(i = 0; i < batch_size; i++) {
    free(input_content_vec[i]);
    free(ref_res[i]);
}
for(i = 0; i < feature_size; i++) {
    free(db_content_vec[i]);
}
free(input_content_vec);
free(db_content_vec);
free(ref_res);

bm_dev_free(handle);
return ret;
}
```

## 5.9 bmcv\_feature\_match

该接口用于将网络得到特征点（int8 格式）与数据库中特征点（int8 格式）进行比对，输出最佳匹配的 top-k。

接口形式：

```
bm_status_t bmcv_feature_match(  
    bm_handle_t handle,  
    bm_device_mem_t input_data_global_addr,  
    bm_device_mem_t db_data_global_addr,  
    bm_device_mem_t output_sorted_similarity_global_addr,  
    bm_device_mem_t output_sorted_index_global_addr,  
    int batch_size,  
    int feature_size,  
    int db_size,  
    int sort_cnt,  
    int rshiftbits);
```

参数说明：

- bm\_handle\_t handle

输入参数。bm\_handle 句柄。

- bm\_device\_mem\_t input\_data\_global\_addr

输入参数。所要比对的特征点数据存储的地址。该数据按照 batch\_size \* feature\_size 的数据格式进行排列。batch\_size, feature\_size 具体含义将在下面进行介绍。bm\_device\_mem\_t 为内置表示地址的数据类型，可以使用函数 bm\_mem\_from\_system(addr) 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- bm\_device\_mem\_t db\_data\_global\_addr

输入参数。数据库的特征点数据存储的地址。该数据按照 feature\_size \* db\_size 的数据格式进行排列。feature\_size, db\_size 具体含义将在下面进行介绍。bm\_device\_mem\_t 为内置表示地址的数据类型，可以使用函数 bm\_mem\_from\_system(addr) 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- bm\_device\_mem\_t output\_sorted\_similarity\_global\_addr

输出参数。每个 batch 得到的比对结果的值中最大几个值（降序排列）存储地址，具体取多少个值由 sort\_cnt 决定。该数据按照 batch\_size \* sort\_cnt 的数据格式进行排列。batch\_size 具体含义将在下面进行介绍。bm\_device\_mem\_t 为内置表示地址的数据类型，可以使用函数 bm\_mem\_from\_system(addr) 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- bm\_device\_mem\_t output\_sorted\_index\_global\_addr

输出参数。每个 batch 得到的比对结果的在数据库中的序号的存储地址。如对于 batch 0，如果 output\_sorted\_similarity\_global\_addr 中 batch 0 的数据是由输入数据与数据库的第 800 组特征点进行比对得到的，那么 output\_sorted\_index\_global\_addr 所在地址对应 batch 0 的数据为 800. output\_sorted\_similarity\_global\_addr 中的

数据按照 batch\_size \* sort\_cnt 的数据格式进行排列。batch\_size 具体含义将在下面进行介绍。bm\_device\_mem\_t 为内置表示地址的数据类型，可以使用函数 bm\_mem\_from\_system(addr) 将普通用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- int batch\_size

输入参数。待输入数据的 batch 个数，如输入数据有 4 组特征点，则该数据的 batch\_size 为 4。batch\_size 最大值不应超过 10。

- int feature\_size

输入参数。每组数据的特征点个数。feature\_size 最大值不应该超过 3000。

- int db\_size

输入参数。数据库中数据特征点的组数。db\_size 最大值不应该超过 100000。

- int sort\_cnt

输入参数。每个 batch 对比结果中所要排序个数，也就是输出结果个数，如需要最大的 3 个比对结果，则 sort\_cnt 设置为 3。该值默认为 1。sort\_cnt 最大值不应该超过 30。

- int rshiftbits

输入参数。对结果进行右移处理的位数，右移采用 round 对小数进行取整处理。该参数默认为 0。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 注意事项:

1. 输入数据和数据库中数据的数据类型为 int8 类型。
2. 输出的比对结果数据类型为 short，输出的序号类型为 int。
3. 数据库中的数据在内存的排布为 feature\_size \* db\_size，因此需要将一组特征点进行转置之后再放入数据库中。
4. sort\_cnt 的取值范围为 1 ~ 30。

#### 示例代码

```
#include "bmvc_api_ext_c.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

int main()
{
    int batch_size = rand() % 10 + 1;
```

(续下页)

(接上页)

```

int feature_size = rand() % 1000 + 1;
int rshiftbits = rand() % 3;
int sort_cnt = rand() % 30 + 1;
int db_size = (rand() % 90 + 1) * 1000;
bm_handle_t handle;
int ret = 0;

ret = (int)bm_dev_request(&handle, 0);

int8_t* input_data = (int8_t*)malloc(sizeof(int8_t) * batch_size * feature_size);
int8_t* db_data = (int8_t*)malloc(sizeof(int8_t) * db_size * feature_size);
short* output_similarity = (short*)malloc(sizeof(short) * batch_size * db_size);
int* output_index = (int*)malloc(sizeof(int) * batch_size * db_size);
int i, j;
int8_t temp_val;

int8_t** db_content_vec = (int8_t**)malloc(sizeof(int8_t*) * feature_size);
for (i = 0; i < feature_size; ++i) {
    db_content_vec[i] = (int8_t*)malloc(sizeof(int8_t) * db_size);
}
int8_t** input_content_vec = (int8_t**)malloc(sizeof(int8_t*) * batch_size);
for (i = 0; i < batch_size; ++i) {
    input_content_vec[i] = (int8_t*)malloc(sizeof(int8_t) * feature_size);
}

short** ref_res = (short**)malloc(sizeof(short*) * batch_size);
for (i = 0; i < batch_size; ++i) {
    ref_res[i] = (short*)malloc(sizeof(short) * db_size);
}

for (i = 0; i < feature_size; ++i) {
    for (j = 0; j < db_size; ++j) {
        temp_val = rand() % 20 - 10;
        db_content_vec[i][j] = temp_val;
    }
}

for (i = 0; i < batch_size; ++i) {
    for (j = 0; j < feature_size; ++j) {
        temp_val = rand() % 20 - 10;
        input_content_vec[i][j] = temp_val;
    }
}

for (i = 0; i < feature_size; ++i) {
    for (j = 0; j < db_size; ++j) {
        db_data[i * db_size + j] = db_content_vec[i][j];
    }
}

for (i = 0; i < batch_size; ++i) {

```

(续下页)

(接上页)

```

for (j = 0; j < feature_size; ++j) {
    input_data[i * feature_size + j] = input_content_vec[i][j];
}
}

ret = bmcv_feature_match(handle, bm_mem_from_system(input_data), bm_
    ↵mem_from_system(db_data),
    bm_mem_from_system(output_similarity), bm_mem_from_
    ↵system(output_index),
        batch_size, feature_size, db_size, sort_cnt, rshiftbits);

free(input_data);
free(db_data);
free(output_similarity);
free(output_index);
for(i = 0; i < batch_size; ++i) {
    free(input_content_vec[i]);
    free(ref_res[i]);
}
for(i = 0; i < feature_size; ++i) {
    free(db_content_vec[i]);
}
free(input_content_vec);
free(db_content_vec);
free(ref_res);

bm_dev_free(handle);
return ret;
}
}

```

## 5.10 bmcv\_fft

FFT 运算。完整的使用步骤包括创建、执行、销毁三步。

### 5.10.1 创建

支持一维或者两维的 FFT 计算，其区别在于创建过程中，后面的执行和销毁使用相同的接口。

对于一维的 FFT，支持多 batch 的运算，接口形式如下：

```

bm_status_t bmcv_fft_1d_create_plan(
    bm_handle_t handle,
    int batch,
    int len,
    bool forward,
    void** plan);

```

**参数说明：**

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `int batch`  
输入参数。`batch` 的数量。
- `int len`  
输入参数。每个 `batch` 的长度。`len` 需要是 2、3、4 或 5 的幂。
- `bool forward`  
输入参数。是否为正向变换，`false` 表示逆向变换。
- `void** plan`  
输出参数。执行阶段需要使用的句柄。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

对于两维  $M * N$  的 FFT 运算，接口形式如下：

```
bm_status_t bmcv_fft_2d_create_plan(  
    bm_handle_t handle,  
    int M,  
    int N,  
    bool forward,  
    void** plan);
```

#### 参数说明:

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `int M`  
输入参数。第一个维度的大小。
- `int N`  
输入参数。第二个维度的大小。
- `bool forward`  
输入参数。是否为正向变换，`false` 表示逆向变换。
- `void** plan`  
输出参数。执行阶段需要使用的句柄。

#### 返回值说明:

- `BM_SUCCESS`: 成功

- 其他: 失败

### 5.10.2 执行

使用上述创建后的 plan 就可以开始真正的执行阶段了，支持复数输入和实数输入两种接口，其格式分别如下：

```
bm_status_t bmcv_fft_execute(
    bm_handle_t handle,
    bm_device_mem_t inputReal,
    bm_device_mem_t inputImag,
    bm_device_mem_t outputReal,
    bm_device_mem_t outputImag,
    const void *plan);

bm_status_t bmcv_fft_execute_real_input(
    bm_handle_t handle,
    bm_device_mem_t inputReal,
    bm_device_mem_t outputReal,
    bm_device_mem_t outputImag,
    const void *plan);
```

#### 参数说明：

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_device_mem_t inputReal`  
输入参数。存放输入数据实数部分的 device memory 空间，对于一维的 FFT，其大小为  $\text{batch} \times \text{len} \times \text{sizeof}(\text{float})$ ，对于两维 FFT，其大小为  $M \times N \times \text{sizeof}(\text{float})$ 。
- `bm_device_mem_t inputImag`  
输入参数。存放输入数据虚数部分的 device memory 空间，对于一维的 FFT，其大小为  $\text{batch} \times \text{len} \times \text{sizeof}(\text{float})$ ，对于两维 FFT，其大小为  $M \times N \times \text{sizeof}(\text{float})$ 。
- `bm_device_mem_t outputReal`  
输出参数。存放输出结果实数部分的 device memory 空间，对于一维的 FFT，其大小为  $\text{batch} \times \text{len} \times \text{sizeof}(\text{float})$ ，对于两维 FFT，其大小为  $M \times N \times \text{sizeof}(\text{float})$ 。
- `bm_device_mem_t outputImag`  
输出参数。存放输出结果虚数部分的 device memory 空间，对于一维的 FFT，其大小为  $\text{batch} \times \text{len} \times \text{sizeof}(\text{float})$ ，对于两维 FFT，其大小为  $M \times N \times \text{sizeof}(\text{float})$ 。
- `const void* plan`  
输入参数。创建阶段所得到的句柄。

#### 返回值说明：

- `BM_SUCCESS`: 成功

- 其他: 失败

### 5.10.3 销毁

当执行完成后需要销毁所创建的句柄。

```
void bmcv_fft_destroy_plan(bm_handle_t handle, void* plan);
```

### 5.10.4 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

int main()
{
    bm_handle_t handle;
    int ret = 0;
    int i;
    int L = 100;
    int batch = 100;
    bool forward = true;
    bool realInput = false;

    ret = (int)bm_dev_request(&handle, 0);
    if (ret) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return ret;
    }

    float* XRHost = (float*)malloc(L * batch * sizeof(float));
    float* XIHost = (float*)malloc(L * batch * sizeof(float));
    float* YRHost_tpu = (float*)malloc(L * batch * sizeof(float));
    float* YIHost_tpu = (float*)malloc(L * batch * sizeof(float));

    for (i = 0; i < L * batch; ++i) {
        XRHost[i] = (float)rand() / RAND_MAX;
        XIHost[i] = realInput ? 0 : ((float)rand() / RAND_MAX);
    }

    bm_device_mem_t XRDev, XIDev, YRDev, YIDev;
    void* plan = NULL;

    ret = bm_malloc_device_byte(handle, &XRDev, L * batch * sizeof(float));
    ret = bm_malloc_device_byte(handle, &XIDev, L * batch * sizeof(float));
    ret = bm_malloc_device_byte(handle, &YRDev, L * batch * sizeof(float));
```

(续下页)

(接上页)

```

ret = bm_malloc_device_byte(handle, &YIDev, L * batch * sizeof(float));

ret = bm_memcpy_s2d(handle, XRDev, XRHost);
ret = bm_memcpy_s2d(handle, XIDev, XIHost);

ret = bmcv_fft_2d_create_plan(handle, L, batch, forward, &plan);

ret = bmcv_fft_execute(handle, XRDev, XIDev, YRDev, YIDev, plan);
if (ret != BM_SUCCESS) {
    printf("bmcv_fft_execute failed!\n");
    if (plan != NULL) {
        bmcv_fft_destroy_plan(handle, plan);
    }
}
ret = bm_memcpy_d2s(handle, (void*)YRHost_tpu, YRDev);
ret = bm_memcpy_d2s(handle, (void*)YIHost_tpu, YIDev);

if (plan != NULL) {
    bmcv_fft_destroy_plan(handle, plan);
}

free(XRHost);
free(XIHost);
free(YRHost_tpu);
free(YIHost_tpu);

bm_dev_free(handle);
return ret;
}

```

## 5.11 bmcv\_gemm

该接口可以实现 float32 类型矩阵的通用乘法计算，如下公式：

$$C = \alpha \times A \times B + \beta \times C$$

其中，A、B、C 均为矩阵， $\alpha$  和  $\beta$  均为常系数

**接口形式：**

```

bm_status_t bmcv_gemm(
    bm_handle_t handle,
    bool is_A_trans,
    bool is_B_trans,
    int M,
    int N,
    int K,
)

```

(续下页)

(接上页)

```
float alpha,  
bm_device_mem_t A,  
int lda,  
bm_device_mem_t B,  
int ldb,  
float beta,  
bm_device_mem_t C,  
int ldc);
```

### 参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄
- bool is\_A\_trans  
输入参数。设定矩阵 A 是否转置
- bool is\_B\_trans  
输入参数。设定矩阵 B 是否转置
- int M  
输入参数。矩阵 A 和矩阵 C 的行数
- int N  
输入参数。矩阵 B 和矩阵 C 的列数
- int K  
输入参数。矩阵 A 的列数和矩阵 B 的行数
- float alpha  
输入参数。数乘系数
- bm\_device\_mem\_t A  
输入参数。根据数据存放位置保存左矩阵 A 数据的 device 地址或者 host 地址。如果数据存放于 host 空间则内部会自动完成 s2d 的搬运
- int lda  
输入参数。矩阵 A 的 leading dimension, 即第一维度的大小, 在行与行之间没有 stride 的情况下即为 A 的列数 (不做转置) 或行数 (做转置)
- bm\_device\_mem\_t B  
输入参数。根据数据存放位置保存右矩阵 B 数据的 device 地址或者 host 地址。如果数据存放于 host 空间则内部会自动完成 s2d 的搬运。
- int ldb

输入参数。矩阵 C 的 leading dimension, 即第一维度的大小, 在行与行之间没有 stride 的情况下即为 B 的列数 (不做转置) 或行数 (做转置)。

- float beta  
输入参数。数乘系数。
- bm\_device\_mem\_t C

输出参数。根据数据存放位置保存矩阵 C 数据的 device 地址或者 host 地址。如果是 host 地址, 则当 beta 不为 0 时, 计算前内部会自动完成 s2d 的搬运, 计算后再自动完成 d2s 的搬运。

- int ldc  
输入参数。矩阵 C 的 leading dimension, 即第一维度的大小, 在行与行之间没有 stride 的情况下即为 C 的列数。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define GEMM_INPUT_NUM 3
#define GEMM_OUTPUT_NUM 1

static int assign_values_to_matrix(float* matrix, int size)
{
    int i;

    if (matrix == NULL || size <= 0) {
        printf("the assign_values_to_matrix func error!\n");
        return -1;
    }

    for (i = 0; i < size; ++i) {
        matrix[i] = (rand() % 100) * 0.01f;
    }

    return 0;
}

int main()
{
    int M = 1 + rand() % 800;
```

(续下页)

(接上页)

```
int N = 1 + rand() % 800;
int K = 1 + rand() % 800;
int rand_sign_a = (rand() % 2 == 0) ? 1 : -1;
int rand_sign_b = (rand() % 2 == 0) ? 1 : -1;
float alpha = rand_sign_a * (rand() % 100) * 0.05;
float beta = rand_sign_b * (rand() % 100) * 0.05;
bool if_A_trans = rand() % 2;
bool if_B_trans = rand() % 2;
int ret = 0;
bm_handle_t handle;

if (if_A_trans) {
    if_B_trans = true;
}

ret = bm_dev_request(&handle, 0);
if (ret) {
    printf("bm_dev_request failed. ret = %d\n", ret);
    return ret;
}

float* src_A = (float*)malloc(M * K * sizeof(float));
float* src_B = (float*)malloc(N * K * sizeof(float));
float* src_C = (float*)malloc(M * N * sizeof(float));
int lda = if_A_trans ? M : K;
int ldb = if_B_trans ? K : N;

ret = assign_values_to_matrix(src_A, M * K);
ret = assign_values_to_matrix(src_B, N * K);
ret = assign_values_to_matrix(src_C, M * N);

ret = bmcv_gemm(handle, if_A_trans, if_B_trans, M, N, K, alpha, bm_mem_
from_system((void *)src_A),
                lda, bm_mem_from_system((void *)src_B), ldb, beta,
                bm_mem_from_system((void *)src_C), N);

free(src_A);
free(src_B);
free(src_C);

bm_dev_free(handle);
return ret;
}
```

## 5.12 bmcv\_gemm\_ext

该接口可以实现 fp32/fp16 类型矩阵的通用乘法计算，如下公式：

$$Y = \alpha \times A \times B + \beta \times C$$

其中，A、B、C、Y 均为矩阵， $\alpha$  和  $\beta$  均为常系数

**接口形式：**

```
bm_status_t bmcv_gemm_ext(
    bm_handle_t handle,
    bool is_A_trans,
    bool is_B_trans,
    int M,
    int N,
    int K,
    float alpha,
    bm_device_mem_t A,
    bm_device_mem_t B,
    float beta,
    bm_device_mem_t C,
    bm_device_mem_t Y,
    bm_image_data_format_ext input_dtype,
    bm_image_data_format_ext output_dtype);
```

**参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄
- bool is\_A\_trans  
输入参数。设定矩阵 A 是否转置
- bool is\_B\_trans  
输入参数。设定矩阵 B 是否转置
- int M  
输入参数。矩阵 A、C、Y 的行数
- int N  
输入参数。矩阵 B、C、Y 的列数
- int K  
输入参数。矩阵 A 的列数和矩阵 B 的行数
- float alpha  
输入参数。数乘系数

- `bm_device_mem_t A`

输入参数。根据数据存放位置保存左矩阵 A 数据的 device 地址，需在使用前完成数据 s2d 搬运。

- `bm_device_mem_t B`

输入参数。根据数据存放位置保存右矩阵 B 数据的 device 地址，需在使用前完成数据 s2d 搬运。

- `float beta`

输入参数。数乘系数。

- `bm_device_mem_t C`

输入参数。根据数据存放位置保存矩阵 C 数据的 device 地址，需在使用前完成数据 s2d 搬运。

- `bm_device_mem_t Y`

输出参数。矩阵 Y 数据的 device 地址，保存输出结果。

- `bm_image_data_format_ext input_dtype`

输入参数。输入矩阵 A、B、C 的数据类型。支持输入 FP16-输出 FP16 或 FP32，输入 FP32-输出 FP32。

- `bm_image_data_format_ext output_dtype`

输入参数。输出矩阵 Y 的数据类型。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项:

1. 该接口在 FP16 输入、A 矩阵转置的情况下，M 仅支持小于等于 64 的取值。
2. 该接口不支持 FP32 输入且 FP16 输出。

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define GEMM_INPUT_NUM 3
#define GEMM_OUTPUT_NUM 1

static int assign_values_to_matrix(float* matrix, int size)
```

(续下页)

(接上页)

```

{
    int i;

    if (matrix == NULL || size <= 0) {
        printf("the assign_values_to_matrix func error!\n");
        return -1;
    }

    for (i = 0; i < size; ++i) {
        matrix[i] = (rand() % 100) * 0.01f;
    }

    return 0;
}

int main()
{
    int M = 1 + rand() % 800;
    int N = 1 + rand() % 800;
    int K = 1 + rand() % 800;
    int rand_sign_a = (rand() % 2 == 0) ? 1 : -1;
    int rand_sign_b = (rand() % 2 == 0) ? 1 : -1;
    float alpha = rand_sign_a * (rand() % 100) * 0.05;
    float beta = rand_sign_b * (rand() % 100) * 0.05;
    bool is_A_trans = rand() % 2;
    bool is_B_trans = rand() % 2;
    int ret = 0;
    bm_handle_t handle;

    if (is_A_trans) {
        is_B_trans = true;
    }

    ret = bm_dev_request(&handle, 0);
    if (ret) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return ret;
    }

    float* A = (float*)malloc(M * K * sizeof(float));
    float* B = (float*)malloc(N * K * sizeof(float));
    float* C = (float*)malloc(M * N * sizeof(float));
    float* tpu_C = (float*)malloc(M * N * sizeof(float));
    bm_image_data_format_ext in_dtype, out_dtype;

    ret = assign_values_to_matrix(A, M * K);
    ret = assign_values_to_matrix(B, N * K);
    ret = assign_values_to_matrix(C, M * N);
    memset(tpu_C, 0.f, sizeof(float) * M * N);
}

```

(续下页)

(接上页)

```

in_dtype = DATA_TYPE_EXT_FLOAT32;
out_dtype = DATA_TYPE_EXT_FLOAT32;
memset(tpu_C, 0.f, sizeof(float) * M * N);

if (in_dtype == DATA_TYPE_EXT_FP16 && is_A_trans && M > 64) {
    printf("Error! It only support M <= 64 when A is trans and input_dtype is FP16\n");
    return -1;
}

unsigned short* A_fp16 = (unsigned short*)malloc(M * K * sizeof(unsigned short));
unsigned short* B_fp16 = (unsigned short*)malloc(N * K * sizeof(unsigned short));
unsigned short* C_fp16 = (unsigned short*)malloc(M * N * sizeof(unsigned short));
unsigned short* Y_fp16 = (unsigned short*)malloc(M * N * sizeof(unsigned short));
bm_device_mem_t input_dev_buffer[GEMM_INPUT_NUM];
bm_device_mem_t output_dev_buffer[GEMM_OUTPUT_NUM];

ret = bm_malloc_device_byte(handle, &input_dev_buffer[0], M * K * sizeof(float));
ret = bm_malloc_device_byte(handle, &input_dev_buffer[1], N * K * sizeof(float));
ret = bm_malloc_device_byte(handle, &input_dev_buffer[2], M * N * sizeof(float));
ret = bm_memcpy_s2d(handle, input_dev_buffer[0], (void*)A);
ret = bm_memcpy_s2d(handle, input_dev_buffer[1], (void*)B);
ret = bm_memcpy_s2d(handle, input_dev_buffer[2], (void*)C);

ret = bm_malloc_device_byte(handle, &output_dev_buffer[0], M * N * sizeof(float));

ret = bmcv_gemm_ext(handle, is_A_trans, is_B_trans, M, N, K, alpha, input_dev_buffer[0],
                     input_dev_buffer[1], beta, input_dev_buffer[2], output_dev_buffer[0],
                     in_dtype, out_dtype);

ret = bm_memcpy_d2s(handle, (void*)tpu_C, output_dev_buffer[0]);

free(A_fp16);
free(B_fp16);
free(C_fp16);
free(Y_fp16);

free(A);
free(B);
free(C);
free(tpu_C);

bm_dev_free(handle);
return ret;

```

(续下页)

(接上页)

}

### 5.13 bmcv\_hist\_balance

对图像进行直方图均衡化操作，提高图像的对比度。

**接口形式：**

```
bm_status_t bmcv_hist_balance(  
    bm_handle_t handle,  
    bm_device_mem_t input,  
    bm_device_mem_t output,  
    int H,  
    int W);
```

**参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle句柄
- bm\_device\_mem\_t input  
输入参数。存放输入图像的device空间。其大小为H \* W \* sizeof(uint8\_t)。
- bm\_device\_mem\_t output  
输出参数。存放输出图像的device空间。其大小为H \* W \* sizeof(uint8\_t)。
- int H  
输入参数。图像的高。
- int W  
输入参数。图像的宽。

**返回值说明：**

- BM\_SUCCESS: 成功
- 其他: 失败

**注意事项：**

1. 数据类型仅支持uint8\_t。
2. 支持的最小图像尺寸为H = 1, W = 1。
3. 支持的最大图像尺寸为H = 8192, W = 8192。

**示例代码**

```
#include <math.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "bmvc_api_ext_c.h"

int main()
{
    int height = rand() % 8192 + 1;
    int width = rand() % 8192 + 1;
    int ret = 0;
    bm_handle_t handle;
    int i;

    ret = bm_dev_request(&handle, 0);
    if (ret) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return ret;
    }

    int len = height * width;

    uint8_t* inputHost = (uint8_t*)malloc(len * sizeof(uint8_t));
    uint8_t* output_tpu = (uint8_t*)malloc(len * sizeof(uint8_t));

    for (i = 0; i < len; ++i) {
        inputHost[i] = (uint8_t)(rand() % 256);
    }

    bm_device_mem_t input, output;
    int H = height;
    int W = width;

    ret = bm_malloc_device_byte(handle, &output, H * W * sizeof(uint8_t));
    ret = bm_malloc_device_byte(handle, &input, H * W * sizeof(uint8_t));
    ret = bm_memcpy_s2d(handle, input, inputHost);

    ret = bmvc_hist_balance(handle, input, output, H, W);
    ret = bm_memcpy_d2s(handle, output_tpu, output);

    bm_free_device(handle, input);
    bm_free_device(handle, output);

    free(inputHost);
    free(output_tpu);

    bm_dev_free(handle);
    return ret;
}
```

## 5.14 bmcv\_hm\_distance

### 描述:

该接口用来计算两个向量中各个元素的汉明距离，该接口支持启用双核处理。

### 语法:

```

1 bm_status_t bmcv_hamming_distance(
2     bm_handle_t handle,
3     bm_device_mem_t input1,
4     bm_device_mem_t input2,
5     bm_device_mem_t output,
6     int bits_len,
7     int input1_num,
8     int input2_num);

```

### 参数:

表 5.4: bmcv\_hm\_distance 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	向量 1 数据的地址信息。
input2	输入	向量 2 数据的地址信息。
output	输出	output 向量数据的地址信息。
bits_len	输入	向量中的每个元素的长度。
input1_num	输入	向量 1 的数据个数。
input2_num	输入	向量 2 的数据个数。

### 返回值:

该函数成功调用时，返回 BM\_SUCCESS。

### 注意事项:

1. bits\_len 支持到 4, 8, 16, 32。
2. input1\_num 支持 1 ~ 16, input2\_num 支持 2 ~ 50000000, 但因 TPU 能调度的存储空间容量有限，不同 bits\_len 和 input1\_num 的组合下，input2\_num 能支持的最大值可能受限。
3. 该接口支持启用双核处理，在运行程序前可通过设置环境变量来改变使用的 TPU core，例如：export TPU\_CORES=0/1/2/both，如不设置环境变量，默认使用 core0 处理。其中 TPU\_CORES=0 代表仅启用 TPU core0 进行处理，TPU\_CORES=1 代表仅启用 TPU core1 进行处理，TPU\_CORES=2 和 TPU\_CORES=both 代表启用双核进行处理。

### 代码示例:

```

1 #include <math.h>
2 #include "stdio.h"
3 #include "stdlib.h"
4 #include "string.h"
5 #include "bmcv_api_ext_c.h"
6
7 int main() {
8     int bits_len = 8;
9     int input1_num = 1 + rand() % 16;
10    int input2_num = 1 + rand() % 10000;
11    bm_handle_t handle;
12    bm_status_t ret = bm_dev_request(&handle, 0);
13    if (ret != BM_SUCCESS) {
14        printf("Create bm handle failed. ret = %d\n", ret);
15        return -1;
16    }
17
18    bm_device_mem_t input1_dev_mem;
19    bm_device_mem_t input2_dev_mem;
20    bm_device_mem_t output_dev_mem;
21
22    uint32_t* input1_data = (uint32_t*)malloc(input1_num * bits_len * sizeof(uint32_t));
23    uint32_t* input2_data = (uint32_t*)malloc(input2_num * bits_len * sizeof(uint32_t));
24    uint32_t* output_tpu = (uint32_t*)malloc(input1_num * input2_num * sizeof(uint32_t));
25
26    printf("bits_len is %u\n", bits_len);
27    printf("input1_data len is %u\n", input1_num);
28    printf("input2_data len is %u\n", input2_num);
29    memset(input1_data, 0, input1_num * bits_len * sizeof(uint32_t));
30    memset(input2_data, 0, input2_num * bits_len * sizeof(uint32_t));
31    memset(output_tpu, 0, input1_num * input2_num * sizeof(uint32_t));
32
33 // fill data
34 for(int i = 0; i < input1_num * bits_len; i++) {
35     input1_data[i] = rand() % 10;
36 }
37 for(int i = 0; i < input2_num * bits_len; i++) {
38     input2_data[i] = rand() % 20 + 1;
39 }
40 // tpu_cal
41 bm_malloc_device_byte(handle, &input1_dev_mem, input1_num * bits_len * sizeof(uint32_t));
42 bm_malloc_device_byte(handle, &input2_dev_mem, input2_num * bits_len * sizeof(uint32_t));
43 bm_malloc_device_byte(handle, &output_dev_mem, input1_num * input2_num * [F]sizeof(uint32_t));
44 bm_memcpy_s2d(handle, input1_dev_mem, input1_data);
45 bm_memcpy_s2d(handle, input2_dev_mem, input2_data);
46
47 bmcv_hamming_distance(handle, input1_dev_mem, input2_dev_mem, output_dev_mem, [F]bits_len, input1_num, input2_num);

```

(续下页)

(接上页)

```

48     bm_memcpy_d2s(handle, output_tpu, output_dev_mem);
49
50     for (int i = 0; i < 8; i++) {
51         printf("output_tpu[%d] is: %d\n", i, output_tpu[i]);
52     }
53     free(input1_data);
54     free(input2_data);
55     free(output_tpu);
56     bm_free_device(handle, input1_dev_mem);
57     bm_free_device(handle, input2_dev_mem);
58     bm_free_device(handle, output_dev_mem);
59
60     bm_dev_free(handle);
61     return ret;
62 }

```

## 5.15 bmcv\_matmul

该接口可以实现 8-bit 数据类型矩阵的乘法计算，如下公式：

$$C = (A \times B) \gg rshift\_bit \quad (5.1)$$

或者

$$C = alpha \times (A \times B) + beta \quad (5.2)$$

其中，

- A 是输入的左矩阵，其数据类型可以是 unsigned char 或者 signed char 类型的 8-bit 数据，大小为 (M, K)；
- B 是输入的右矩阵，其数据类型可以是 unsigned char 或者 signed char 类型的 8-bit 数据，大小为 (K, N)；
- C 是输出的结果矩阵，其数据类型长度可以是 int8、int16 或者 float32，用户配置决定。

当 C 是 int8 或者 int16 时，执行上述公式 (5.1) 的功能，而其符号取决于 A 和 B，当 A 和 B 均为无符号时 C 才为无符号数，否则为有符号；

当 C 是 float32 时，执行上述公式 (5.2) 的功能。

- rshift\_bit 是矩阵乘积的右移数，当 C 是 int8 或者 int16 时才有效，由于矩阵的乘积有可能会超出 8-bit 或者 16-bit 的范围，所以用户可以配置一定的右移数，通过舍弃部分精度来防止溢出。
- alpha 和 beta 是 float32 的常系数，当 C 是 float32 时才有效。

## 接口形式:

```
bm_status_t bmcv_matmul(  
    bm_handle_t handle,  
    int M,  
    int N,  
    int K,  
    bm_device_mem_t A,  
    bm_device_mem_t B,  
    bm_device_mem_t C,  
    int A_sign,  
    int B_sign,  
    int rshift_bit,  
    int result_type,  
    bool is_B_trans,  
    float alpha = 1,  
    float beta = 0);
```

## 参数说明:

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄
- int M  
输入参数。矩阵 A 和矩阵 C 的行数
- int N  
输入参数。矩阵 B 和矩阵 C 的列数
- int K  
输入参数。矩阵 A 的列数和矩阵 B 的行数
- bm\_device\_mem\_t A  
输入参数。根据左矩阵 A 数据存放位置保存其 device 地址或者 host 地址。如果数据存放于 host 空间则内部会自动完成 s2d 的搬运
- bm\_device\_mem\_t B  
输入参数。根据右矩阵 B 数据存放位置保存其 device 地址或者 host 地址。如果数据存放于 host 空间则内部会自动完成 s2d 的搬运。
- bm\_device\_mem\_t C  
输出参数。根据矩阵 C 数据存放位置保存其 device 地址或者 host 地址。如果是 host 地址，则当 beta 不为 0 时，计算前内部会自动完成 s2d 的搬运，计算后再自动完成 d2s 的搬运。
- int A\_sign  
输入参数。左矩阵 A 的符号，1 表示有符号，0 表示无符号。
- int B\_sign

输入参数。右矩阵 B 的符号，1 表示有符号，0 表示无符号。

- int rshift\_bit

输入参数。矩阵乘积的右移数，为非负数。只有当 result\_type 等于 0 或者 1 时才有效。

- int result\_type

输入参数。输出的结果矩阵数据类型，0 表示是 int8，1 表示 int16, 2 表示 float32。

- bool is\_B\_trans

输入参数。输入右矩阵 B 是否需要计算前做转置。

- float alpha

常系数，输入矩阵 A 和 B 相乘之后再乘上该系数，只有当 result\_type 等于 2 时才有效，默认值为 1。

- float beta

常系数，在输出结果矩阵 C 之前，加上该偏移量，只有当 result\_type 等于 2 时才有效，默认值为 0。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 示例代码

```
#include "bmvc_api_ext_c.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "test_misc.h"

static void assign_fix8b_matrix(void* mat, int size, int is_16bit)
{
    int i;

    for (i = 0; i < size; i++) {
        if (is_16bit) {
            *((signed short*)mat + i) = rand() % 65536 - 32768;
        } else {
            *((signed char*)mat + i) = rand() % 256 - 128;
        }
    }
}

int main()
{
    int M = 1 + rand() % 6;
    int K = 1 + rand() % 512;
```

(续下页)

(接上页)

```
int N = 1 + rand() % 9216;
// int trans_A = 0; //whether to transpose
int trans_B = 0;
int A_sign = 0; //unsigned or singned
int B_sign = 0;
int result_type = 0; //0-int8 1-int16 2-float
int right_shift_bit = 1;
float alpha = 1;
float beta = 0;
int ret = 0;
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);

signed char* input_A;
signed char* input_B;
void* tpu_out;
input_A = (signed char*)malloc(M * K * sizeof(signed char));
input_B = (signed char*)malloc(K * N * sizeof(signed char));

if (result_type == 0) {
    tpu_out = (signed char*)malloc(M * N * sizeof(signed char));
    memset(tpu_out, 0, M * N * sizeof(signed char));
}

assign_fix8b_matrix((void*)input_A, M * K, 0);
assign_fix8b_matrix((void*)input_B, K * N, 0);

ret = bmcv_matmul(handle, M, N, K, bm_mem_from_system((void*)input_A),
                  bm_mem_from_system((void*)input_B), bm_mem_from_
                  system(tpu_out), A_sign,
                  B_sign, right_shift_bit, result_type, trans_B, alpha, beta);

if (ret != BM_SUCCESS) {
    printf("Create bm handle failed. ret = %d\n", ret);
    bm_dev_free(handle);
    return -1;
}

free(input_A);
free(input_B);
free(tpu_out);

bm_dev_free(handle);
return ret;
}
```

## 5.16 bmcv\_min\_max

对于存储于 device memory 中连续空间的一组数据，该接口可以获取这组数据中的最大值和最小值。

**接口形式：**

```
bm_status_t bmcv_min_max(  
    bm_handle_t handle,  
    bm_device_mem_t input,  
    float* minVal,  
    float* maxVal,  
    int len);
```

**参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄
- bm\_device\_mem\_t input  
输入参数。输入数据的 device 地址。
- float\* minVal  
输出参数。运算后得到的最小值结果，如果为 NULL 则不计算最小值。
- float\* maxVal  
输出参数。运算后得到的最大值结果，如果为 NULL 则不计算最大值。
- int len  
输入参数。输入数据的长度。

**返回值说明：**

- BM\_SUCCESS: 成功
- 其他: 失败

**注意事项：**

1. 该接口可通过设置环境变量启用双核计算，运行程序前：export TPU\_CORES=2 或 export TPU\_CORES=both 即可。

**示例代码**

```
#include "bmcv_api_ext_c.h"  
#include "stdio.h"  
#include "stdlib.h"  
  
int main() {  
    int L = 50 + rand() % 260095;
```

(续下页)

(接上页)

```

int ret = 0;
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);

if (ret != BM_SUCCESS) {
    printf("Create bm handle failed. ret = %d\n", ret);
    return -1;
}

float min_tpu = 0, max_tpu = 0;
float* XHost = (float*)malloc(L * sizeof(float));
int i;

for (i = 0; i < L; ++i)
    XHost[i] = (float)((rand() % 2 ? 1 : -1) * (rand() % 1000 + (rand() % 100000) * 0.01));

bm_device_mem_t XDev;

ret = bm_malloc_device_byte(handle, &XDev, L * sizeof(float));
ret = bm_memcpy_s2d(handle, XDev, XHost);
ret = bmcv_min_max(handle, XDev, &min_tpu, &max_tpu, L);
if (ret != BM_SUCCESS) {
    printf("Calculate bm min_max failed. ret = %d\n", ret);
    return -1;
}

bm_free_device(handle, XDev);

free(XHost);

bm_dev_free(handle);
return ret;
}

```

### 5.17 bmcv\_nms

该接口用于消除网络计算得到过多的物体框，并找到最佳物体框。

**接口形式：**

```

bm_status_t bmcv_nms(bm_handle_t handle,
                      bm_device_mem_t input_proposal_addr,
                      int proposal_size,
                      float nms_threshold,
                      bm_device_mem_t output_proposal_addr);

```

**参数说明：**

- bm\_handle\_t handle

输入参数。bm\_handle 句柄。

- bm\_device\_mem\_t input\_proposal\_addr

输入参数。输入物体框数据所在地址，输入物体框数据结构为 face\_rect\_t，详见下面数据结构说明。需要调用 bm\_mem\_from\_system() 将数据地址转化成转化为 bm\_device\_mem\_t 所对应的结构。

- int proposal\_size

输入参数。物体框个数。

- float nms\_threshold

输入参数。过滤物体框的阈值，分数小于该阈值的物体框将会被过滤掉。

- bm\_device\_mem\_t output\_proposal\_addr

输出参数。输出物体框数据所在地址，输出物体框数据结构为 nms\_proposal\_t，详见下面数据结构说明。需要调用 bm\_mem\_from\_system() 将数据地址转化成转化为 bm\_device\_mem\_t 所对应的结构。

#### 返回值说明：

- BM\_SUCCESS: 成功
- 其他: 失败

#### 数据类型说明：

face\_rect\_t 描述了一个物体框坐标位置以及对应的分数。

```
struct face_rect_t {
    float x1;
    float y1;
    float x2;
    float y2;
    float score;
};
```

- x1 描述了物体框左边缘的横坐标
- y1 描述了物体框上边缘的纵坐标
- x2 描述了物体框右边缘的横坐标
- y2 描述了物体框下边缘的纵坐标
- score 描述了物体框对应的分数

nms\_proposal\_t 描述了输出物体框的信息。

```
struct nms_proposal_t {
    struct face_rect_t face_rect[MAX_PROPOSAL_NUM];
    int size;
    int capacity;
    struct face_rect_t* begin;
```

(续下页)

(接上页)

```
struct face_rect_t* end;
};
```

- face\_rect 描述了经过过滤后的物体框信息
- size 描述了过滤后得到的物体框个数
- capacity 描述了过滤后物体框最大个数
- begin 暂不使用
- end 暂不使用

#### 示例代码:

```
#include "bmvc_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "test_misc.h"

#define SCORE_RAND_LEN_MAX 50000

int main()
{
    int num = rand() % SCORE_RAND_LEN_MAX + 1;
    int i;
    int ret = 0;
    float nms_threshold = 0.7;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);

    if (ret != BM_SUCCESS) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return -1;
    }

    face_rect_t* input = (face_rect_t*)malloc(num * sizeof(face_rect_t));
    nms_proposal_t tpu_out[1];

    for (i = 0; i < num; ++i) {
        input[i].x1 = ((float)(rand() % 100)) / 10;
        input[i].x2 = input[i].x1 + ((float)(rand() % 100)) / 10;
        input[i].y1 = ((float)(rand() % 100)) / 10;
        input[i].y2 = input[i].y1 + ((float)(rand() % 100)) / 10;
        input[i].score = (float)rand() / (float)RAND_MAX;
    }

    ret = bmvc_nms(handle, bm_mem_from_system(input), num, nms_threshold, F
    ↵bm_mem_from_system(tpu_out));
    if (ret != BM_SUCCESS) {
```

(续下页)

(接上页)

```

printf("Calculate bm_nms failed.\n");
bm_dev_free(handle);
return -1;
}

free(input);
bm_dev_free(handle);
return ret;
}

```

**注意事项:**

1. 该 api 可输入的最大 proposal 数为 50000。

**5.18 bmcv\_sort**

该接口可以实现浮点数据的排序（升序/降序），并且支持排序后可以得到原数据所对应的索引值。

**接口形式:**

```

bm_status_t bmcv_sort(
    bm_handle_t handle,
    bm_device_mem_t src_index_addr,
    bm_device_mem_t src_data_addr,
    bm_device_mem_t dst_index_addr,
    bm_device_mem_t dst_data_addr,
    int data_cnt,
    int sort_cnt,
    int order,
    bool index_enable,
    bool auto_index);

```

**参数说明:**

- `bm_handle_t handle`

输入参数。`bm_handle_t handle` 设备环境句柄，通过调用 `bm_dev_request` 获取。

- `bm_device_mem_t src_index_addr`

输入参数。每个输入数据所对应 `index` 的地址。如果使能 `index_enable` 并且不使用 `auto_index` 时，则该参数有效。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- `bm_device_mem_t src_data_addr`

输入参数。待排序的输入数据所对应的地址。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- `bm_device_mem_t dst_index_addr`

输出参数。排序后输出数据所对应 index 的地址，如果使能 `index_enable` 并且不使用 `auto_index` 时，则该参数有效。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- `bm_device_mem_t dst_data_addr`

输出参数。排序后的输出数据所对应的地址。`bm_device_mem_t` 为内置表示地址的数据类型，可以使用函数 `bm_mem_from_system(addr)` 将用户使用的指针或地址转为该类型，用户可参考示例代码中的使用方式。

- `int data_cnt`

输入参数。待排序的输入数据的数量。

- `int sort_cnt`

输入参数。需要排序的数量，也就是输出结果的个数，包括排好序的数据和对应 index。比如降序排列，如果只需要输出最大的前三个数据，该参数设置为 3 即可。

- `int order`

输入参数。升序还是降序，0 表示升序，1 表示降序。

- `bool index_enable`

输入参数。是否使能 `index`。如果使能即可输出排序后数据所对应的 `index`，否则 `src_index_addr` 和 `dst_index_addr` 这两个参数无效。

- `bool auto_index`

输入参数。是否使能自动生成 `index` 功能。使用该功能的前提是 `index_enable` 参数为 `true`，如果该参数也为 `true` 则表示按照输入数据的存储顺序从 0 开始计数作为 `index`，参数 `src_index_addr` 便无效，输出结果中排好序数据所对应的 `index` 即存放于 `dst_index_addr` 地址中。

#### 返回值说明：

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项：

1. 要求输入参数满足： $0 < \text{sort\_cnt} \leq \text{data\_cnt}$ 。
2. 若需要使用 `auto index` 功能，前提是参数 `index_enable` 为 `true`。
3. 该接口可通过设置环境变量启用双核计算，运行程序前：`export TPU_CORES=2` 或 `export TPU_CORES=both` 即可。
4. 由于 TPU 排序类计算的底层机制限制，该接口在输入参数 `sort_cnt > 128` 时，输出数据地址和输出索引地址空间需要申请为 `data_num * sizeof(data_type)` 而不是 `sort_num * sizeof(data_type)`，否则接口输出将会出错。

#### 示例代码

```

#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "bmcv_api_ext_c.h"

#define MAX_SORT_NUM (500000)

typedef float bm_sort_data_type_t;
typedef enum { ASCEND_ORDER, DESCEND_ORDER } cdma_sort_order_e;

typedef struct {
    int index;
    float val;
} __attribute__((packed)) sort_t;

int32_t main() {
    int dev_id = 0;
    int data_num = 1 + rand() % 500000;
    int sort_num = 1 + rand() % data_num;
    int ret = 0;
    bm_handle_t handle;
    cdma_sort_order_e order1 = DESCEND_ORDER;
    // cdma_sort_order_e order2 = ASCEND_ORDER;
    ret = bm_dev_request(&handle, dev_id);

    bm_sort_data_type_t *src_data = (bm_sort_data_type_t*)malloc(data_num[* sizeof(float)]);
    int *src_index_p = (int*)malloc(data_num * sizeof(int));
    sort_t *ref_res = (sort_t*)malloc(data_num * sizeof(sort_t));
    sort_t *cdma_res = (sort_t*)malloc(sort_num * sizeof(sort_t));
    bm_sort_data_type_t *dst_data = (bm_sort_data_type_t*)malloc(sort_num[* sizeof(bm_sort_data_type_t)]);
    int *dst_data_index = (int*)malloc(sort_num * sizeof(int));
    bool index_enable = rand() % 2 ? true : false;
    bool auto_index = rand() % 2 ? true : false;
    printf("data num: %d, sort num: %d\n", data_num, sort_num);

    // produce src data and index
    for (int32_t i = 0; i < data_num; i++) {
        if(auto_index){
            src_index_p[i] = i;
        }else{
            src_index_p[i] = rand() % MAX_SORT_NUM;
        }
        ref_res[i].index = src_index_p[i];
        ref_res[i].val = ((float)(rand() % MAX_SORT_NUM)) / 100;
        src_data[i] = ref_res[i].val;
    }

    int *dst_index_p = NULL;
}

```

(续下页)

(接上页)

```

bm_sort_data_type_t *dst_data_p = NULL;
dst_index_p = (int*)malloc(sort_num * sizeof(int));
dst_data_p = (bm_sort_data_type_t*)malloc(sort_num * sizeof(int));

bmcv_sort(handle, bm_mem_from_system(src_index_p), bm_mem_from_
system(src_data), data_num,
          bm_mem_from_system(dst_index_p), bm_mem_from_system(dst_data_
p), sort_num, (int)order1,
          index_enable, auto_index);

for (int i = 0; i < sort_num; i++) {
    cdma_res[i].index = dst_index_p[i];
    cdma_res[i].val = dst_data_p[i];
}
free(dst_index_p);
free(dst_data_p);

// release memory
free(src_data);
free(src_index_p);
free(ref_res);
free(cdma_res);
free(dst_data);
free(dst_data_index);

bm_dev_free(handle);
return ret;
}

```

## 5.19 bmcv\_stft

接口形式如下：

```

bm_status_t bmcv_stft(
    bm_handle_t handle,
    float* XRHost, float* XIHost,
    float* YRHost, float* YIHost,
    int batch, int L,
    bool realInput, int pad_mode,
    int n_fft, int win_mode, int hop_len,
    bool normalize);

```

**参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle句柄。
- float\* XRHost  
输入参数。输入信号的实部地址，。空间大小为batch \* L。

- float\* XIHost  
输入参数。输入信号的虚部地址，空间大小为 batch \* L。
- float\* YRHost  
输入参数。输出信号的实部地址，空间大小为  $(n_{fft} / 2 + 1) * (1 + L / hop\_len)$ 。
- float\* YIHost  
输入参数。输出信号的虚部地址，空间大小为  $(n_{fft} / 2 + 1) * (1 + L / hop\_len)$ 。
- int batch  
输入参数。batch 数量。
- int L  
输入参数。每个 batch 的信号长度。L 需要是 2、3、4 或 5 的幂。
- bool realInput  
输入参数。输入是否为实数，false 为复数，true 为实数。
- int pad\_mode  
输入参数。信号填充模式，0 为 constant, 1 为 reflect。
- int n\_fft  
输入参数。每个 L 信号长度做 FFT 的长度
- int win\_mode  
输入参数。信号加窗模式，0 为 hanning 窗，1 为 hamming 窗。
- int hop\_len  
输入参数。信号做 FFT 的滑动距离，一般为  $n_{fft} / 4$  或者  $n_{fft} / 2$ 。
- bool normalize  
输入参数。输出结果是否要进行归一化。

**返回值说明:**

- BM\_SUCCESS: 成功
- 其他: 失败

**注意事项:**

1. 本接口只处理一维信号的 STFT 变换。

**示例代码**

```
#include "bmcv_api_ext_c.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

(续下页)

(接上页)

```

#include <math.h>
#include <string.h>

enum Pad_mode {
    CONSTANT = 0,
    REFLECT = 1,
};

enum Win_mode {
    HANN = 0,
    HAMM = 1,
};

static void readBin_4b(const char* path, float* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 4, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

int main()
{
    bm_handle_t handle;
    int ret = 0;
    int i;
    int L = 4096;
    int batch = 1;
    bool realInput = true;
    int pad_mode = REFLECT;
    int win_mode = HANN;
    int n_fft = 4096;
    int hop_length = 1024;
    bool norm = true;
    char* src_name = NULL;

    ret = (int)bm_dev_request(&handle, 0);
    if (ret) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return ret;
    }

    float* XRHost = (float*)malloc(L * batch * sizeof(float));
    float* XIHost = (float*)malloc(L * batch * sizeof(float));
    int num_frames = 1 + L / hop_length;
    int row_num = n_fft / 2 + 1;
    float* YRHost_tpu = (float*)malloc(batch * row_num * num_frames * [F
    ↵sizeof(float)];
    float* YIHost_tpu = (float*)malloc(batch * row_num * num_frames * [F
    ↵sizeof(float)]);
}

```

(续下页)

(接上页)

```

    ↵sizeof(float));

    memset(XRHost, 0, L * batch * sizeof(float));
    memset(XIHost, 0, L * batch * sizeof(float));

    if (src_name != NULL) {
        readBin_4b(src_name, XRHost, L * batch);
    } else {
        for (i = 0; i < L * batch; i++) {
            XRHost[i] = (float)rand() / RAND_MAX;
            XIHost[i] = realInput ? 0 : ((float)rand() / RAND_MAX);
        }
    }
    ret = bmcv_stft(handle, XRHost, XIHost, YRHost_tpu, YIHost_tpu, batch, L,
                    realInput, pad_mode, n_fft, win_mode, hop_length, norm);

    free(XRHost);
    free(XIHost);
    free(YRHost_tpu);
    free(YIHost_tpu);

    bm_dev_free(handle);
    return ret;
}

```

## 5.20 bmcv\_istft

接口形式如下：

```

bm_status_t bmcv_istft(
    bm_handle_t handle,
    float* XRHost, float* XIHost,
    float* YRHost, float* YIHost,
    int batch, int L,
    bool realInput, int pad_mode,
    int n_fft, int win_mode, int hop_len,
    bool normalize);

```

**参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle句柄。
- float\* XRHost  
输入参数。输入信号的实部地址，。空间大小为  $(n_{\text{fft}} / 2 + 1) * (1 + L / \text{hop\_len})$ 。
- float\* XIHost

输入参数。输入信号的虚部地址，空间大小为  $(n\_fft / 2 + 1) * (1 + L / hop\_len)$ 。

- float\* YRHost

输入参数。输出信号的实部地址，空间大小为 batch \* L。

- float\* YIHost

输入参数。输出信号的虚部地址，空间大小为 batch \* L。

- int batch

输入参数。batch 数量。

- int L

输入参数。每个 batch 的信号长度。L 需要是 2、3、4 或 5 的幂。

- bool realInput

输入参数。输出的信号是否为实数，false 为复数，true 为实数。

- int pad\_mode

输入参数。信号填充模式，0 为 constant，1 为 reflect。

- int n\_fft

输入参数。每个 L 信号长度做 FFT 的长度

- int win\_mode

输入参数。信号加窗模式，0 为 hanning 窗，1 为 hamming 窗。

- int hop\_len

输入参数。信号做 FFT 的滑动距离，一般为 n\_fft / 4 或者 n\_fft / 2。

- bool normalize

输入参数。输出结果是否要进行归一化。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 注意事项:

1. 本接口只处理一维信号的 ISTFT 变换。

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

(续下页)

(接上页)

```
#define M_PI 3.14159265358979323846

enum Pad_mode {
    CONSTANT = 0,
    REFLECT = 1,
};

enum Win_mode {
    HANN = 0,
    HAMM = 1,
};

typedef struct {
    float real;
    float imag;
} Complex;

int main()
{
    bm_handle_t handle;
    int ret = 0;
    int i;
    int L = 4096;
    bool real_input = true;
    int pad_mode = REFLECT;
    int win_mode = HANN;
    int n_fft = 4096;
    int hop_length = n_fft / 4;
    bool norm = true;

    ret = (int)bm_dev_request(&handle, 0);
    if (ret) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return ret;
    }

    int num_frames = 1 + L / hop_length;
    int row_num = n_fft / 2 + 1;

    float* input_R = (float*)malloc(row_num * num_frames * sizeof(float));
    float* input_I = (float*)malloc(row_num * num_frames * sizeof(float));
    float* YRHost_cpu = (float*)malloc(L * sizeof(float));
    float* YRHost_tpu = (float*)malloc(L * sizeof(float));
    float* YIHost_cpu = (float*)malloc(L * sizeof(float));
    float* YIHost_tpu = (float*)malloc(L * sizeof(float));

    memset(input_R, 0, row_num * num_frames * sizeof(float));
    memset(input_I, 0, row_num * num_frames * sizeof(float));

    for (i = 0; i < row_num * num_frames; i++) {
```

(续下页)

(接上页)

```

    input_R[i] = (float)rand() / RAND_MAX;
    input_I[i] = (float)rand() / RAND_MAX;
}

ret = bmcv_istft(handle, input_R, input_I, YRHost_tpu, YIHost_tpu, 1, L, real_
    ↵input,
    pad_mode, n_fft, win_mode, hop_length, norm);

free(input_R);
free(input_I);
free(YRHost_cpu);
free(YIHost_cpu);
free(YRHost_tpu);
free(YIHost_tpu);

bm_dev_free(handle);
return ret;
}

```

## 5.21 bmcv\_faiss\_indexflatIP

计算查询向量与数据库向量的内积距离, 输出前 K (sort\_cnt) 个最匹配的内积距离值及其对应的索引。

接口形式:

```

bm_status_t bmcv_faiss_indexflatIP(
    bm_handle_t handle,
    bm_device_mem_t input_data_global_addr,
    bm_device_mem_t db_data_global_addr,
    bm_device_mem_t buffer_global_addr,
    bm_device_mem_t output_sorted_similarity_global_addr,
    bm_device_mem_t output_sorted_index_global_addr,
    int vec_dims,
    int query_vecs_num,
    int database_vecs_num,
    int sort_cnt,
    int is_transpose,
    int input_dtype,
    int output_dtype);

```

输入参数说明:

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t input\_data\_global\_addr  
输入参数。存放查询向量组成的矩阵的 device 空间。

- `bm_device_mem_t db_data_global_addr`  
输入参数。存放底库向量组成的矩阵的 device 空间。
- `bm_device_mem_t buffer_global_addr`  
输入参数。存放计算出的内积值的缓存空间。
- `bm_device_mem_t output_sorted_similarity_global_addr`  
输出参数。存放排序后的最匹配的内积值的 device 空间。
- `bm_device_mem_t output_sorted_index_global_addr`  
输出参数。存储输出内积值对应索引的 device 空间。
- `int vec_dims`  
输入参数。向量维数。
- `int query_vecs_num`  
输入参数。查询向量的个数。
- `int database_vecs_num`  
输入参数。底库向量的个数。
- `int sort_cnt`  
输入参数。输出的前 `sort_cnt` 个最匹配的内积值。
- `int is_transpose`  
输入参数。0 表示底库矩阵不转置; 1 表示底库矩阵转置。
- `int input_dtype`  
输入参数。输入数据类型，支持 fp32,fp16,char, 5 表示 fp32, 3 表示 fp16, 1 表示 char。
- `int output_dtype`  
输出参数。输出数据类型，支持 fp32,fp16,char, 5 表示 fp32, 3 表示 fp16, 9 表示 char。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项:

1. 输入数据(查询向量)和底库数据(底库向量)的数据类型为 fp32, fp16 或 char。
2. 输出的排序后的相似度的数据类型为 fp32, fp16 或 int, 相对应的索引的数据类型为 int。
3. 底库数据通常以 `database_vecs_num * vec_dims` 的形式排布在内存中。此时, 参数 `is_transpose` 需要设置为 1。
4. 查询向量和数据库向量内积距离值越大, 表示两者的相似度越高。因此, 在 TopK 过程中对内积距离值按降序排序。

5. 输入参数中 sort\_cnt 和 query\_vecs\_num 需要小于或等于 database\_vecs\_num。
6. 该接口可通过设置环境变量启用双核计算，运行程序前：export TPU\_CORES=2 或 export TPU\_CORES=both 即可。

示例代码：

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "bmcv_api_ext_c.h"
#include "test_misc.h"

int main() {
    int sort_cnt = 100;
    int database_vecs_num = 10000;
    int query_vecs_num = 1;
    int vec_dims = 256;
    int is_transpose = 1;
    int ret;

    int input_dtype = 1;
    int output_dtype = 9;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (BM_SUCCESS != ret) {
        printf("request dev failed\n");
        return BM_ERR_FAILURE;
    }

    int i, j;
    signed char* input_data = (signed char*)malloc(query_vecs_num * vec_dims * sizeof(signed char));
    signed char* db_data = (signed char*)malloc(database_vecs_num * vec_dims * sizeof(signed char));
    int* output_similarity = (int*)malloc(query_vecs_num * sort_cnt * sizeof(int));
    signed char** input_content_vec = (signed char**)malloc(query_vecs_num * sizeof(signed char));
    for (i = 0; i < query_vecs_num; i++) {
        input_content_vec[i] = (signed char*)malloc(vec_dims * sizeof(signed char));
        for (j = 0; j < vec_dims; j++) {
            #ifdef __linux__
            signed char temp_val = random() % 20 - 10;
            #else
            signed char temp_val = rand() % 20 - 10;
            #endif
            input_content_vec[i][j] = temp_val;
        }
    }
    signed char** db_content_vec = (signed char**)malloc((is_transpose ? database_vecs_num : vec_dims) * sizeof(signed char));
    signed char** db_content_vec_trans = (signed char**)malloc((is_transpose ? vec_dims : database_vecs_num) * sizeof(signed char));
    for (i = 0; i < query_vecs_num; i++) {
        for (j = 0; j < vec_dims; j++) {
            db_content_vec[i][j] = input_data[i * vec_dims + j];
        }
    }
    for (i = 0; i < database_vecs_num; i++) {
        for (j = 0; j < vec_dims; j++) {
            db_content_vec_trans[i * vec_dims + j] = db_data[i * vec_dims + j];
        }
    }
    ret = bm_exec(handle, 0, input_data, query_vecs_num * vec_dims, output_similarity, sort_cnt, db_content_vec, database_vecs_num * vec_dims, db_content_vec_trans, vec_dims);
    if (BM_SUCCESS != ret) {
        printf("bm_exec failed\n");
        return BM_ERR_FAILURE;
    }
    for (i = 0; i < query_vecs_num; i++) {
        free(input_content_vec[i]);
    }
    free(input_content_vec);
    free(db_content_vec);
    free(db_content_vec_trans);
    free(input_data);
    free(db_data);
}
```

(续下页)

(接上页)

```

    ↵ dims : database_vecs_num) * sizeof(signed char*));

    for(i = 0; i < vec_dims; i++) {
        db_content_vec_trans[i] = (signed char*)malloc(database_vecs_num * [F
    ↵ sizeof(signed char));
    }
    for (i = 0; i < database_vecs_num; i++) {
        db_content_vec[i] = (signed char*)malloc(vec_dims * sizeof(signed char));
        for (j = 0; j < vec_dims; j++) {
            #ifdef __linux__
            signed char temp_val = random() % 20 + 1;
            #else
            signed char temp_val = rand() % 20 - 10;
            #endif
            db_content_vec[i][j] = temp_val;
            db_content_vec_trans[j][i] = temp_val;
        }
    }

    for (i = 0; i < query_vecs_num; ++i) {
        for (j = 0; j < vec_dims; ++j) {
            input_data[i * vec_dims + j] = input_content_vec[i][j];
        }
    }

    for (i = 0; i < database_vecs_num; ++i) {
        for (j = 0; j < vec_dims; ++j) {
            db_data[i * vec_dims + j] = db_content_vec[i][j];
        }
    }

    int* output_index = (int*)malloc(query_vecs_num * sort_cnt * sizeof(int));
    int** ref_result = (int**)calloc(query_vecs_num, sizeof(int*));
    for (i = 0; i < query_vecs_num; i++) {
        ref_result[i] = (int*)calloc(database_vecs_num, sizeof(int));
    }

    bm_device_mem_t input_data_global_addr_device,
                    db_data_global_addr_device,
                    buffer_global_addr_device,
                    output_sorted_similarity_global_addr_device,
                    output_sorted_index_global_addr_device;
    bm_malloc_device_byte(handle,
                          &input_data_global_addr_device,
                          dtype_size((enum bm_data_type_t)input_dtype) * query_vecs_
    ↵ num * vec_dims);
    bm_malloc_device_byte(handle,
                          &db_data_global_addr_device,
                          dtype_size((enum bm_data_type_t)input_dtype) * database_
    ↵ vecs_num * vec_dims);
    bm_malloc_device_byte(handle,
                          &buffer_global_addr_device,

```

(续下页)

(接上页)

```

        dtype_size((enum bm_data_type_t)DT_FP32) * query_vecs_
num * database_vecs_num);
bm_malloc_device_byte(handle,
    &output_sorted_similarity_global_addr_device,
    dtype_size((enum bm_data_type_t)output_dtype) * query_
vecs_num * sort_cnt);
bm_malloc_device_byte(handle,
    &output_sorted_index_global_addr_device,
    dtype_size(DT_INT32) * query_vecs_num * sort_cnt);
bm_memcpy_s2d(handle,
    input_data_global_addr_device,
    bm_mem_get_system_addr(bm_mem_from_system(input_data)));
bm_memcpy_s2d(handle,
    db_data_global_addr_device,
    bm_mem_get_system_addr(bm_mem_from_system(db_data)));

ret = bmcv_faiss_indexflatIP(handle,
    input_data_global_addr_device,
    db_data_global_addr_device,
    buffer_global_addr_device,
    output_sorted_similarity_global_addr_device,
    output_sorted_index_global_addr_device,
    vec_dims,
    query_vecs_num,
    database_vecs_num,
    sort_cnt,
    is_transpose,
    input_dtype,
    output_dtype);

bm_memcpy_d2s(handle,
    bm_mem_get_system_addr(bm_mem_from_system(output_
similarity)),
    output_sorted_similarity_global_addr_device);
bm_memcpy_d2s(handle,
    bm_mem_get_system_addr(bm_mem_from_system(output_index)),
    output_sorted_index_global_addr_device);

bm_free_device(handle, input_data_global_addr_device);
bm_free_device(handle, db_data_global_addr_device);
bm_free_device(handle, buffer_global_addr_device);
bm_free_device(handle, output_sorted_similarity_global_addr_device);
bm_free_device(handle, output_sorted_index_global_addr_device);

free(input_data);
free(db_data);
free(output_similarity);
for (i = 0; i < query_vecs_num; ++i) {
    free(input_content_vec[i]);
    free(ref_result[i]);
}
}

```

(续下页)

(接上页)

```

for(i = 0; i < query_vecs_num; ++i){
    free(db_content_vec[i]);
}
free(input_content_vec);
free(db_content_vec);
free(output_index);
free(ref_result);

bm_dev_free(handle);
return 0;
}

```

## 5.22 bmcv\_faiss\_indexflatL2

计算查询向量与数据库向量 L2 距离的平方，输出前 K (sort\_cnt) 个最匹配的 L2 距离的平方值及其对应的索引。

**接口形式：**

```

bm_status_t bmcv_faiss_indexflatL2(
    bm_handle_t handle,
    bm_device_mem_t input_data_global_addr,
    bm_device_mem_t db_data_global_addr,
    bm_device_mem_t query_L2norm_global_addr,
    bm_device_mem_t db_L2norm_global_addr,
    bm_device_mem_t buffer_global_addr,
    bm_device_mem_t output_sorted_similarity_global_addr,
    bm_device_mem_t output_sorted_index_global_addr,
    int vec_dims,
    int query_vecs_num,
    int database_vecs_num,
    int sort_cnt,
    int is_transpose,
    int input_dtype,
    int output_dtype);

```

**输入参数说明：**

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t input\_data\_global\_addr  
输入参数。存放查询向量组成的矩阵的 device 空间。
- bm\_device\_mem\_t db\_data\_global\_addr  
输入参数。存放底库向量组成的矩阵的 device 空间。
- bm\_device\_mem\_t query\_L2norm\_global\_addr

输入参数。存放计算出的内积值的缓存空间。

- `bm_device_mem_t db_L2norm_global_addr`

输入参数。Device addr information of the database norm\_L2sqr vector.

- `bm_device_mem_t buffer_global_addr`

输入参数。Squared L2 values stored in the buffer

- `bm_device_mem_t output_sorted_similarity_global_addr`

输出参数。存放排序后的最匹配的内积值的 device 空间。

- `bm_device_mem_t output_sorted_index_global_addr`

输出参数。存储输出内积值对应索引的 device 空间。

- `int vec_dims`

输入参数。向量维数。

- `int query_vecs_num`

输入参数。查询向量的个数。

- `int database_vecs_num`

输入参数。底库向量的个数。

- `int sort_cnt`

输入参数。输出的前 `sort_cnt` 个最匹配的 L2 距离的平方值。

- `int is_transpose`

输入参数。0 表示底库矩阵不转置; 1 表示底库矩阵转置。

- `int input_dtype`

输入参数。输入数据类型，支持 fp32 和 fp16, 5 表示 fp32, 3 表示 fp16。

- `int output_dtype`

输出参数。输出数据类型，支持 fp32 和 fp16, 5 表示 fp32, 3 表示 fp16。

#### 返回值说明：

- `BM_SUCCESS`: 成功

- 其他: 失败

#### 注意事项：

1. 输入数据 (查询向量) 和底库数据 (底库向量) 的数据类型为 float。
2. 输出的排序后的相似度结果的数据类型为 float, 相对应的索引的数据类型为 int。
3. 假设输入数据和底库数据的 L2 范数的平方值已提前计算完成，并存储在处理器上。
4. 底库数据通常以 `database_vecs_num * vec_dims` 的形式排布在内存中。此时，参数 `is_transpose` 需要设置为 1。

5. 查询向量和数据库向量 L2 距离的平方值越小, 表示两者的相似度越高。因此, 在 TopK 过程中对 L2 距离的平方值按升序排序。
6. database\_vecs\_num 与 sort\_cnt 的取值需要满足条件: database\_vecs\_num > sort\_cnt。
7. 输入参数中 sort\_cnt 和 query\_vecs\_num 需要小于或等于 database\_vecs\_num。
8. 该接口可通过设置环境变量启用双核计算, 运行程序前: export TPU\_CORES=2 或 export TPU\_CORES=both 即可。

### 示例代码:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "bmvc_api_ext_c.h"
#include "test_misc.h"
#include "string.h"

typedef unsigned int u32;
typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned long long u64;

typedef struct {
    int L_row_num;
    int L_col_num;
    int R_col_num;
    int transpose;
    enum bm_data_type_t L_dtype;
    enum bm_data_type_t R_dtype;
    enum bm_data_type_t Y_dtype;
} matmul_param_t;

void fvec_norm_L2sqr_ref(float* vec, float* matrix, int row_num, int col_num) {
    for (int i = 0; i < row_num; i++) {
        for (int j = 0; j < col_num; j++) {
            vec[i] += matrix[i * col_num + j] * matrix[i * col_num + j];
        }
    }
}

void matrix_trans(void* src, void* dst, int row_num, int col_num, enum bm_data_
→type_t dtype) {
    for (int i = 0; i < row_num; i++) {
        for (int j = 0; j < col_num; j++) {
            if (dtype == DT_INT8 || dtype == DT_UINT8) {
                ((u8*)dst)[j * row_num + i] = ((u8*)src)[i * col_num + j];
            } else if (dtype == DT_INT16 || dtype == DT_UINT16) {
                ((u16*)dst)[j * row_num + i] = ((u16*)src)[i * col_num + j];
            } else if (dtype == DT_FP32) {
                ((float*)dst)[j * row_num + i] = ((float*)src)[i * col_num + j];
            }
        }
    }
}
```

(续下页)

(接上页)

```

} else if (dtype == DT_INT32 || dtype == DT_UINT32) {
    ((u32*)dst)[j * row_num + i] = ((u32*)src)[i * col_num + j];
} else if (dtype == DT_FP16) {
    ((fp16*)dst)[j * row_num + i] = ((fp16*)src)[i * col_num + j];
}
}

void matrix_gen_data(float* data, u32 len) {
    for (u32 i = 0; i < len; i++) {
        data[i] = ((float)rand() / (float)RAND_MAX);
    }
}

int main() {
    int sort_cnt = 100;
    int database_vecs_num = 20000;
    int query_vecs_num = 1;
    int vec_dims = 256;
    int is_transpose = 1;
    int input_dtype = 5;
    int output_dtype = 5;

    int ret;

    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (BM_SUCCESS != ret) {
        printf("request dev failed\n");
        return BM_ERR_FAILURE;
    }

    float* input_data = (float*)malloc(query_vecs_num * vec_dims * sizeof(float));
    float* db_data = (float*)malloc(database_vecs_num * vec_dims * sizeof(float));
    float* db_data_trans = (float*)malloc(vec_dims * database_vecs_num * [F]
    ↵sizeof(float));
    float* vec_query = (float*)malloc(1 * query_vecs_num * sizeof(float));
    float* vec_db = (float*)malloc(1 * database_vecs_num * sizeof(float));

    unsigned char* output_dis = (unsigned char*)malloc(query_vecs_num * sort_
    ↵cnt * dtype_size((enum bm_data_type_t)output_dtype));
    int* output_idx = (int*)malloc(query_vecs_num * sort_cnt * dtype_size(DT_
    ↵INT32));

    float* blob_Y_ref = (float*)malloc(query_vecs_num * database_vecs_num * [F]
    ↵sizeof(float));
    unsigned char* blob_dis_ref = (unsigned char*)malloc(query_vecs_num * sort_
    ↵cnt * dtype_size((enum bm_data_type_t)output_dtype)); //////
    int* blob_inx_ref = (int*)malloc(query_vecs_num * sort_cnt * sizeof(int));
}

```

(续下页)

(接上页)

```

matrix_gen_data(input_data, query_vecs_num * vec_dims);
matrix_gen_data(db_data, vec_dims * database_vecs_num);
matrix_trans(db_data, db_data_trans, database_vecs_num, vec_dims, (enum F
↪bm_data_type_t)input_dtype);
fvec_norm_L2sqr_ref(vec_query, input_data, query_vecs_num, vec_dims);
fvec_norm_L2sqr_ref(vec_db, db_data, database_vecs_num, vec_dims);
bm_device_mem_t query_data_dev_mem,
    db_data_dev_mem,
    query_L2norm_dev_mem,
    db_L2norm_dev_mem,
    buffer_dev_mem,
    sorted_similarity_dev_mem,
    sorted_index_dev_mem;

bm_malloc_device_byte(handle,
    &query_data_dev_mem,
    dtype_size((enum bm_data_type_t)input_dtype) * query_vecs_
↪num * vec_dims);
bm_malloc_device_byte(handle,
    &db_data_dev_mem,
    dtype_size((enum bm_data_type_t)input_dtype) * database_
↪vecs_num * vec_dims);
bm_malloc_device_byte(handle,
    &query_L2norm_dev_mem,
    dtype_size((enum bm_data_type_t)input_dtype) * query_vecs_
↪num * 1);
bm_malloc_device_byte(handle,
    &db_L2norm_dev_mem,
    dtype_size((enum bm_data_type_t)input_dtype) * database_
↪vecs_num * 1);

bm_malloc_device_byte(handle,
    &buffer_dev_mem,
    dtype_size((enum bm_data_type_t)DT_FP32) * query_vecs_
↪num * database_vecs_num);
bm_malloc_device_byte(handle,
    &sorted_similarity_dev_mem,
    dtype_size((enum bm_data_type_t)output_dtype) * query_vecs_
↪num * sort_cnt);
bm_malloc_device_byte(handle,
    &sorted_index_dev_mem,
    dtype_size((enum bm_data_type_t)DT_INT32) * query_vecs_
↪num * sort_cnt);
bm_memcpy_s2d(handle,
    query_data_dev_mem,
    bm_mem_get_system_addr(bm_mem_from_system(input_data)));
bm_memcpy_s2d(handle,
    db_data_dev_mem,
    bm_mem_get_system_addr(bm_mem_from_system(db_data)));
bm_memcpy_s2d(handle,
    query_L2norm_dev_mem,

```

(续下页)

(接上页)

```
    bm_mem_get_system_addr(bm_mem_from_system(vec_query)));
bm_memcpy_s2d(handle,
    db_L2norm_dev_mem,
    bm_mem_get_system_addr(bm_mem_from_system(vec_db)));
ret = bmcv_faiss_indexflatL2(handle,
    query_data_dev_mem,
    db_data_dev_mem,
    query_L2norm_dev_mem,
    db_L2norm_dev_mem,
    buffer_dev_mem,
    sorted_similarity_dev_mem,
    sorted_index_dev_mem,
    vec_dims,
    query_vecs_num,
    database_vecs_num,
    sort_cnt,
    is_transpose,
    input_dtype,
    output_dtype);
bm_memcpy_d2s(handle,
    bm_mem_get_system_addr(bm_mem_from_system(output_dis)),
    sorted_similarity_dev_mem);
bm_memcpy_d2s(handle,
    bm_mem_get_system_addr(bm_mem_from_system(output_idx)),
    sorted_index_dev_mem);
matmul_param_t param;
memset(&param, 0, sizeof(matmul_param_t));

param.L_row_num = query_vecs_num,
param.L_col_num = vec_dims;
param.R_col_num = database_vecs_num;
param.transpose = is_transpose;
param.L_dtype = (enum bm_data_type_t)input_dtype;
param.R_dtype = (enum bm_data_type_t)input_dtype;
param.Y_dtype = (enum bm_data_type_t)output_dtype;

bm_free_device(handle, query_data_dev_mem);
bm_free_device(handle, db_data_dev_mem);
bm_free_device(handle, query_L2norm_dev_mem);
bm_free_device(handle, db_L2norm_dev_mem);
bm_free_device(handle, buffer_dev_mem);
bm_free_device(handle, sorted_similarity_dev_mem);
bm_free_device(handle, sorted_index_dev_mem);

free(input_data);
free(db_data);
free(db_data_trans);
free(vec_query);
free(vec_db);
free(output_dis);
free(output_idx);
```

(续下页)

(接上页)

```
free(blob_Y_ref);
free(blob_dis_ref);
free(blob_inx_ref);

bm_dev_free(handle);
return 0;
}
```

### 5.23 bmcv\_faiss\_indexPQ\_ADC

该接口通过 query 和 centroids 计算距离表，对底库编码查表并排序，输出前 K (sort\_cnt) 个最匹配的向量索引及其对应的距离。

接口形式：

```
bm_status_t bmcv_faiss_indexPQ_ADC(
    bm_handle_t handle,
    bm_device_mem_t centroids_input_dev,
    bm_device_mem_t nxquery_input_dev,
    bm_device_mem_t nycodes_input_dev,
    bm_device_mem_t distance_output_dev,
    bm_device_mem_t index_output_dev,
    int vec_dims,
    int slice_num,
    int centroids_num,
    int database_num,
    int query_num,
    int sort_cnt,
    int IP_metric);
```

输入参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t centroids\_input\_dev  
输入参数。存储聚类中心数据的 deivce 空间。
- bm\_device\_mem\_t nxquery\_input\_dev  
输入参数。存储查询向量组成的矩阵的 deivce 空间。
- bm\_device\_mem\_t nycodes\_input\_dev  
输入参数。存放底库向量组成的矩阵的 device 空间。
- bm\_device\_mem\_t distance\_output\_dev  
输出参数。存放输出距离的 device 空间。

- `bm_device_mem_t index_output_dev`  
输出参数。存放输出排序的 device 空间。
- `int vec_dims`  
输入参数。原始向量的维度。
- `int slice_num`  
输入参数。原始维度切分数量。
- `int centroids_num`  
输入参数。聚类中心的数量。
- `int database_num`  
输入参数。数据底库的数量。
- `int query_num`  
输入参数。检索向量的数量。
- `int sort_cnt`  
输入参数。输出前 `sort_cnt` 个最匹配的底库向量。
- `int IP_metric`  
输入参数。0 表示 L2 距离计算; 1 表示 IP 距离计算。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项:

- 1、输入数据 (查询向量) 和聚类中心的数据类型为 `float`, 底库数据 (底库编码) 的数据类型为 `uint8`, 都存储在设备内存上。
- 2、输出的排序后的相似度结果的数据类型为 `float`, 相对应的索引的数据类型为 `int`, 存储在设备内存上。
- 3、`L2_metric` 计算是 L2 距离平方, 没有进行开方 (参考 faiss 源码的实现)。
- 4、查询向量和数据库向量 L2 距离越小, 表示两者的相似度越高。输出 L2 topk 距离按升序排序。
- 5、查询向量和数据库向量 IP 距离越大, 表示两者的相似度越高。输出 IP topk 距离按降序排序。
- 6、输入参数中 `sort_cnt` 和 `query_num` 需要小于或等于 `database_num`。
- 7、原始维度切分数量 `slice_num` 仅支持 8/16/32/64 且需要小于 `vec_dims`。

8、faiss 系列算子有多个输入参数，每个参数都有一个使用范围限制，超过该范围的入参对应 tpu 输出会出错，我们选择了三个主要参数做了测试，固定其中两个维度，测试了第三个维度的最大值，测试结果如下表格所示：

query_num	vec_dims	max_database_num
1	128	3500 万
1	256	3500 万
1	512	3500 万
64	128	250 万
64	256	250 万
64	512	250 万
128	128	130 万
128	256	130 万
128	512	130 万
256	128	70 万
256	256	70 万
256	512	70 万

database_num	vec_dims	max_query_num
1000	128	1000
1000	256	1000
1000	512	1000
1 万	128	1 万
1 万	256	1 万
1 万	512	1 万
10 万	128	1875
10 万	256	1872
10 万	512	1869

database_num	query_num	max_vec_dims
1万	1	512
1万	64	512
1万	128	512
1万	256	512
10万	1	512
10万	32	512
10万	64	512
10万	128	512
10万	256	512
100万	1	512
100万	32	512
100万	64	512
100万	128	512

## 示例代码

```
#include "bmvc_api_ext_c.h"
#include "test_misc.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <sys/time.h>

int main() {
    int sort_cnt = 100;
    int vec_dims = 256;
    int query_num = 1;
    int slice_m = 32;
    int ksub = 256;
    int dsub = vec_dims / slice_m;
    int database_num = 2000000;
    int input_dtype = 5; // 5:float
    int output_dtype = 5;
    int IP_metric = 0;
    int show_result = 1;
    struct timespec tp;
    clock_gettime(0, &tp);
    unsigned int seed = tp.tv_nsec;

    bm_handle_t handle;
    bm_status_t ret = BM_SUCCESS;
    ret = bm_dev_request(&handle, 0);
    if (BM_SUCCESS != ret)
    {
        printf("request dev failed\n");
        return BM_ERR_FAILURE;
    }
}
```

(续下页)

(接上页)

```

    }

    srand(seed);
    int round = 1;
    fp16 *centroids_input_sys_fp16 = (fp16*)malloc(slice_m * ksub * dsub * [F
    ↵sizeof(fp16));
    fp16 *nxquery_input_sys_fp16 = (fp16*)malloc(query_num * vec_dims * [F
    ↵sizeof(fp16));
    float *centroids_input_sys_fp32 = (float*)malloc(slice_m * ksub * dsub * [F
    ↵sizeof(float));
    float *nxquery_input_sys_fp32 = (float*)malloc(query_num * vec_dims * [F
    ↵sizeof(float));

    unsigned char *nycodes_input_sys = (unsigned char*)malloc(database_num * [F
    ↵slice_m * sizeof(unsigned char));
    unsigned char *distance_output_sys = (unsigned char*)malloc(query_num * [F
    ↵database_num * dtype_size((enum bm_data_type_t )output_dtype));
    int *index_output_sys = (int*)malloc(query_num * database_num * sizeof(int));

    for (int i = 0; i < slice_m; i++) {
        for (int j = 0; j < ksub; j++) {
            for (int n = 0; n < dsub; n++) {
                float value = (float)rand() / RAND_MAX * 20.0 - 10.0;
                centroids_input_sys_fp32[i * dsub * ksub + j * dsub + n] = value;
                centroids_input_sys_fp16[i * dsub * ksub + j * dsub + n] = [F
    ↵fp32tofp16(value, round);
            }
        }
    }
    for (int i = 0; i < query_num; i++) {
        for (int j = 0; j < vec_dims; j++) {
            float value = (float)rand() / RAND_MAX * 20.0 - 10.0;
            nxquery_input_sys_fp32[i * vec_dims + j] = value;
            nxquery_input_sys_fp16[i * vec_dims + j] = fp32tofp16(value, round);
        }
    }
    for (int i = 0; i < database_num; i++) {
        for (int j = 0; j < slice_m; j++) {
            nycodes_input_sys[i * slice_m + j] = rand() % 256;
        }
    }
}

bm_device_mem_t centroids_input_dev, nxquery_input_dev, nycodes_input_
    ↵dev, distance_output_dev, index_output_dev;
int centroids_size = slice_m * ksub * dsub * dtype_size((enum bm_data_type_
    ↵t )input_dtype);
int nxquery_size = query_num * vec_dims * dtype_size((enum bm_data_type_
    ↵t )input_dtype);
int nycodes_size = database_num * slice_m * sizeof(char);
int distance_size = query_num * database_num * dtype_size((enum bm_data_
    ↵type_t )output_dtype);

```

(续下页)

(接上页)

```

int index_size = query_num * database_num * sizeof(int);

bm_malloc_device_byte(handle, &centroids_input_dev, centroids_size);
bm_malloc_device_byte(handle, &nxquery_input_dev, nxquery_size);
bm_malloc_device_byte(handle, &nycodes_input_dev, nycodes_size);
bm_malloc_device_byte(handle, &distance_output_dev, distance_size);
bm_malloc_device_byte(handle, &index_output_dev, index_size);

if (input_dtype == DT_FP16) {
    bm_memcpy_s2d(handle, centroids_input_dev, centroids_input_sys_fp16);
    bm_memcpy_s2d(handle, nxquery_input_dev, nxquery_input_sys_fp16);
} else {
    bm_memcpy_s2d(handle, centroids_input_dev, centroids_input_sys_fp32);
    bm_memcpy_s2d(handle, nxquery_input_dev, nxquery_input_sys_fp32);
}
bm_memcpy_s2d(handle, nycodes_input_dev, nycodes_input_sys);

struct timeval t1, t2;
gettimeofday(&t1, NULL);
ret = bmcv_faiss_indexPQ_ADC_ext(handle,
                                   centroids_input_dev,
                                   nxquery_input_dev,
                                   nycodes_input_dev,
                                   distance_output_dev,
                                   index_output_dev,
                                   vec_dims, slice_m, ksub, database_num, query_num, sort_cnt,
→ IP_metric, input_dtype, output_dtype);
gettimeofday(&t2, NULL);
printf("TPU using time(us): %ld(us)\n", (long)((t2.tv_sec - t1.tv_sec) * 1000000[F
→ + t2.tv_usec - t1.tv_usec));
printf("TPU using time(ms): %ld(ms)\n", (long)((t2.tv_sec - t1.tv_sec) * [F
→ 1000000 + t2.tv_usec - t1.tv_usec) / 1000);

if(ret != BM_SUCCESS){
    bm_free_device(handle, centroids_input_dev);
    bm_free_device(handle, nxquery_input_dev);
    bm_free_device(handle, nycodes_input_dev);
    bm_free_device(handle, distance_output_dev);
    bm_free_device(handle, index_output_dev);

    free(centroids_input_sys_fp32);
    free(centroids_input_sys_fp16);
    free(nxquery_input_sys_fp32);
    free(nxquery_input_sys_fp16);
    free(nycodes_input_sys);
    free(distance_output_sys);
    free(index_output_sys);

    bm_dev_free(handle);
    return BM_ERR_FAILURE;
}

```

(续下页)

(接上页)

```

bm_memcpy_d2s(handle, distance_output_sys, distance_output_dev);
bm_memcpy_d2s(handle, index_output_sys, index_output_dev);

if (show_result) {
    printf("ADCsearch result:\n");
    for (int i = 0; i < sort_cnt; i++) {
        printf("top: %d\n", i + 1);
        printf("index: %d\t", index_output_sys[i]);
        if (output_dtype == DT_FP16) {
            printf("distance: %f", fp16tofp32(((fp16*)distance_output_sys)[i]));
        } else {
            printf("distance: %f", ((float*)distance_output_sys)[i]);
        }
        printf("\n");
    }
}

bm_free_device(handle, centroids_input_dev);
bm_free_device(handle, nxquery_input_dev);
bm_free_device(handle, nycodes_input_dev);
bm_free_device(handle, distance_output_dev);
bm_free_device(handle, index_output_dev);

free(centroids_input_sys_fp32);
free(centroids_input_sys_fp16);
free(nxquery_input_sys_fp32);
free(nxquery_input_sys_fp16);
free(nycodes_input_sys);
free(distance_output_sys);
free(index_output_sys);

bm_dev_free(handle);

return 0;
}

```

## 5.24 bmcv\_faiss\_indexPQ\_SDC

该接口通过检索向量编码和底库编码在 sdc\_table 中查表并累加，输出前 K (sort\_cnt) 个最匹配的向量索引及其对应的距离。

**接口形式：**

```

bm_status_t bmcv_faiss_indexPQ_SDC(
    bm_handle_t handle,
    bm_device_mem_t sdc_table_input_dev,
    bm_device_mem_t nxcodes_input_dev,
    bm_device_mem_t nycodes_input_dev,
)

```

(续下页)

(接上页)

```
bm_device_mem_t distance_output_dev,  
bm_device_mem_t index_output_dev,  
int slice_num,  
int centroids_num,  
int database_num,  
int query_num,  
int sort_cnt,  
int IP_metric);
```

### 输入参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t sdc\_table\_input\_dev  
输入参数。存放对称距离表的 device 空间。
- bm\_device\_mem\_t nxcodes\_input\_dev  
输入参数。存放检索向量编码的 device 空间。
- bm\_device\_mem\_t nycodes\_input\_dev  
输入参数。存放底库编码的 device 空间。
- bm\_device\_mem\_t distance\_output\_dev  
输出参数。存放输出距离的 device 空间。
- bm\_device\_mem\_t index\_output\_dev  
输出参数。存放输出排序的 device 空间。
- int slice\_num  
输入参数。原始维度切分数量。
- int centroids\_num  
输入参数。聚类中心的数量。
- int database\_num  
输入参数。数据底库的数量。
- int query\_num  
输入参数。检索向量的数量。
- int sort\_cnt  
输入参数。输出的前 sort\_cnt 个最匹配底库向量。
- int IP\_metric  
输入参数。0 表示 L2 距离计算；1 表示 IP 距离计算。

### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

### 注意事项:

- 1、输入数据 (查询向量) 和对称距离表的数据类型为 float, 底库数据 (底库编码) 的数据类型为 uint8, 存储在设备内存上。
- 2、输出的排序后的相似度结果的数据类型为 float, 相对应的索引的数据类型为 int, 存储在设备内存上。
- 3、SDC 检索过程中 metric 的选择没有区别, 因为距离在于输入的 sdc table, 主要区别在于其 topk 结果是降序, L2 的结果为升序。
- 4、查询向量和数据库向量 L2 距离越小, 表示两者的相似度越高。输出 L2 topk 距离按升序排序。
- 5、查询向量和数据库向量 IP 距离越大, 表示两者的相似度越高。输出 IP topk 距离按降序排序。
- 6、输入参数中 sort\_cnt 和 query\_num 需要小于或等于 database\_num。
- 7、原始维度切分数量 slice\_num 仅支持 8/16/32/64 且需要小于 vec\_dims。
- 8、faiss 系列算子有多个输入参数, 每个参数都有一个使用范围限制, 超过该范围的入参对应 tpu 输出会出错, 我们选择了三个主要参数做了测试, 固定其中两个维度, 测试了第三个维度的最大值, 测试结果如下表格所示:

query_num	vec_dims	max_database_num
1	128	3500 万
1	256	3500 万
1	512	3500 万
64	128	250 万
64	256	250 万
64	512	250 万
128	128	130 万
128	256	130 万
128	512	130 万
256	128	70 万
256	256	70 万
256	512	70 万

database_num	vec_dims	max_query_num
1000	128	1000
1000	256	1000
1000	512	1000
1 万	128	1 万
1 万	256	1 万
1 万	512	1 万
10 万	128	1875
10 万	256	1872
10 万	512	1869

database_num	query_num	max_vec_dims
1 万	1	512
1 万	64	512
1 万	128	512
1 万	256	512
10 万	1	512
10 万	32	512
10 万	64	512
10 万	128	512
10 万	256	512
100 万	1	512
100 万	32	512
100 万	64	512
100 万	128	512

## 示例代码

```
#include "bmvc_api_ext_c.h"
#include "test_misc.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <sys/time.h>

int main() {
    int sort_cnt = 100;
    int query_num = 1;
    int slice_m = 32;
    int ksub = 256;
    int database_num = 2000000;
    int input_dtype = 5; // 5: float
    int output_dtype = 5;
    int IP_metric = 0;
```

(续下页)

(接上页)

```

int show_result = 1;
struct timespec tp;
clock_gettime(CLOCK_REALTIME, &tp);
unsigned int seed = tp.tv_nsec;

bm_handle_t handle;
bm_status_t ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS)
{
    printf("request dev failed\n");
    return BM_ERR_FAILURE;
}

srand(seed);
int round = 1;
fp16 *sdc_table_input_sys_fp16 = (fp16*)malloc(slice_m * ksub * ksub * sizeof(fp16));
float *sdc_table_input_sys_fp32 = (float*)malloc(slice_m * ksub * ksub * sizeof(float));
unsigned char *nxcodes_input_sys = (unsigned char*)malloc(query_num * slice_m);
unsigned char *nycodes_input_sys = (unsigned char*)malloc(database_num * slice_m);
unsigned char *distance_output_sys = (unsigned char*)malloc(query_num * database_num * dtype_size((enum bm_data_type_t)output_dtype));
int *index_output_sys = (int*)malloc(query_num * database_num * sizeof(int));

for (int i = 0; i < slice_m; i++) {
    for (int j = 0; j < ksub; j++) {
        for (int n = 0; n < ksub; n++) {
            float value = (n > j) ? (float)rand() / RAND_MAX * 20.0 : 0.0;
            sdc_table_input_sys_fp32[i * ksub * ksub + j * ksub + n] = value;
            sdc_table_input_sys_fp16[i * ksub * ksub + j * ksub + n] = fp32tofp16(value, round);
        }
    }
}
for (int i = 0; i < query_num; i++) {
    for (int j = 0; j < slice_m; j++) {
        nxcodes_input_sys[i * slice_m + j] = rand() % 256;
    }
}
for (int i = 0; i < database_num; i++) {
    for (int j = 0; j < slice_m; j++) {
        nycodes_input_sys[i * slice_m + j] = rand() % 256;
    }
}

int sdc_table_size = slice_m * ksub * ksub * dtype_size((enum bm_data_type_t)input_dtype);

```

(续下页)

(接上页)

```

int nxcodes_size = query_num * slice_m;
int nycodes_size = database_num * slice_m;
int output_distance_size = query_num * database_num * dtype_size((enum bm_
↪data_type_t )output_dtype);
int output_index_size = query_num * database_num * sizeof(int);

bm_device_mem_t sdc_table_input_dev, nxcodes_input_dev, nycodes_input_
↪dev, distance_output_dev, index_output_dev;

bm_malloc_device_byte(handle, &sdc_table_input_dev, sdc_table_size);
bm_malloc_device_byte(handle, &nxcodes_input_dev, nxcodes_size);
bm_malloc_device_byte(handle, &nycodes_input_dev, nycodes_size);
bm_malloc_device_byte(handle, &distance_output_dev, output_distance_size);
bm_malloc_device_byte(handle, &index_output_dev, output_index_size);

if (input_dtype == DT_FP16) {
    bm_memcpy_s2d(handle, sdc_table_input_dev, sdc_table_input_sys_fp16);
} else {
    bm_memcpy_s2d(handle, sdc_table_input_dev, sdc_table_input_sys_fp32);
}
bm_memcpy_s2d(handle, nxcodes_input_dev, nxcodes_input_sys);
bm_memcpy_s2d(handle, nycodes_input_dev, nycodes_input_sys);

struct timeval t1, t2;
gettimeofday(&t1, NULL);
ret = bmcv_faiss_indexPQ_SDC_ext(handle,
                                   sdc_table_input_dev,
                                   nxcodes_input_dev,
                                   nycodes_input_dev,
                                   distance_output_dev,
                                   index_output_dev,
                                   slice_m, ksub, database_num, query_num, sort_cnt, IP_
↪metric, input_dtype, output_dtype);
gettimeofday(&t2, NULL);
printf("TPU using time(us): %ld(us)\n", (t2.tv_sec - t1.tv_sec) * 1000000 + t2.tv_
↪usec - t1.tv_usec);
printf("TPU using time(ms): %ld(ms)\n", ((t2.tv_sec - t1.tv_sec) * 1000000 + t2.
↪tv_usec - t1.tv_usec) / 1000);

if(ret != BM_SUCCESS){
    bm_free_device(handle, sdc_table_input_dev);
    bm_free_device(handle, nxcodes_input_dev);
    bm_free_device(handle, nycodes_input_dev);
    bm_free_device(handle, distance_output_dev);
    bm_free_device(handle, index_output_dev);

    free(sdc_table_input_sys_fp32);
    free(sdc_table_input_sys_fp16);
    free(nxcodes_input_sys);
    free(nycodes_input_sys);
    free(distance_output_sys);
}

```

(续下页)

(接上页)

```

        free(index_output_sys);

        bm_dev_free(handle);
        return BM_ERR_FAILURE;
    }

    bm_memcpy_d2s(handle, distance_output_sys, distance_output_dev);
    bm_memcpy_d2s(handle, index_output_sys, index_output_dev);

    if (show_result) {
        printf("SDCsearch result:\n");
        for (int i = 0; i < sort_cnt; i++) {
            printf("top: %d\n", i + 1);
            printf("index: %d\t", index_output_sys[i]);
            if (output_dtype == DT_FP16) {
                printf("distance: %f\n", fp16tofp32(((fp16*)distance_output_sys)[i]));
            } else {
                printf("distance: %f\n", ((float*)distance_output_sys)[i]);
            }
        }
    }

    bm_free_device(handle, sdc_table_input_dev);
    bm_free_device(handle, nxcodes_input_dev);
    bm_free_device(handle, nycodes_input_dev);
    bm_free_device(handle, distance_output_dev);
    bm_free_device(handle, index_output_dev);

    free(sdc_table_input_sys_fp32);
    free(sdc_table_input_sys_fp16);
    free(nxcodes_input_sys);
    free(nycodes_input_sys);
    free(distance_output_sys);
    free(index_output_sys);

    bm_dev_free(handle);
    return 0;
}

```

## 5.25 bmcv\_faiss\_indexPQ\_encode

该接口输入 vectors 和 centroids 计算距离表并排序，输出 vectors 的量化编码。

**接口形式：**

```

bm_status_t bmcv_faiss_indexPQ_encode(
    bm_handle_t handle,
    bm_device_mem_t vector_input_dev,
    bm_device_mem_t centroids_input_dev,

```

(续下页)

(接上页)

```
bm_device_mem_t buffer_table_dev,  
bm_device_mem_t codes_output_dev,  
int encode_vec_num,  
int vec_dims,  
int slice_num,  
int centroids_num,  
int IP_metric);
```

#### 输入参数说明:

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_device\_mem\_t vector\_input\_dev  
输入参数。存放待编码向量的 device 空间。
- bm\_device\_mem\_t centroids\_input\_dev  
输入参数。存储聚类中心数据的 deivce 空间。
- bm\_device\_mem\_t buffer\_table\_dev  
输入参数。存放计算出的距离表的缓存空间。
- bm\_device\_mem\_t codes\_output\_dev  
输出参数。存放向量编码结果的 device 空间。
- int encode\_vec\_num  
输入参数。待编码向量的个数。
- int vec\_dims  
输入参数。原始向量的维度。
- int slice\_num  
输入参数。原始维度切分数量。
- int centroids\_num  
输入参数。聚类中心的数量。
- int IP\_metric  
输入参数。0 表示 L2 距离计算; 1 表示 IP 距离计算。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 注意事项:

1、输入数据(查询向量)和聚类中心的数据类型为 float，输出向量编码的数据类型为 uint8，存储在设备内存上。

2、buffer\_table 的大小为 slice\_num \* centroids\_num，数据类型为 float。

3、原始维度切分数量 slice\_num 仅支持 8/16/32/64 且需要小于 vec\_dims。

### 示例代码

```
#include "bmvc_api_ext_c.h"
#include "test_misc.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <sys/time.h>

int main() {
    int vec_dims = 256;
    int encode_vec_num = 1;
    int slice_m = 32;
    int ksub = 256;
    int dsub = vec_dims / slice_m;
    int input_dtype = 5; // 5: float
    int IP_metric = 0;
    struct timespec tp;
    clock_gettime(CLOCK_REALTIME, &tp);
    unsigned int seed = tp.tv_nsec;

    bm_handle_t handle;
    bm_status_t ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS)
    {
        printf("request dev failed\n");
        return BM_ERR_FAILURE;
    }

    srand(seed);
    float *centroids_input_sys_fp32 = (float*)malloc(slice_m * ksub * dsub * sizeof(float));
    unsigned char *nxcodes_input_sys = (unsigned char*)malloc(encode_vec_num * vec_dims);
    unsigned char *output_codes_sys = (unsigned char*)malloc(encode_vec_num * slice_m);
    for (int i = 0; i < slice_m; i++) {
        for (int j = 0; j < ksub; j++) {
            for (int n = 0; n < dsub; n++) {
                float value = (float)rand() / RAND_MAX * 20.0 - 10.0;
                centroids_input_sys_fp32[i * dsub * ksub + j * dsub + n] = value;
            }
        }
    }
    for (int i = 0; i < encode_vec_num; i++) {
```

(续下页)

(接上页)

```

for (int j = 0; j < slice_m; j++) {
    nxcodes_input_sys[i * slice_m + j] = rand() % 256;
}
}

int centroids_size = slice_m * ksub * dsub * dtype_size((enum bm_data_type_
↪)input_dtype);
int nxcodes_size = encode_vec_num * vec_dims * dtype_size((enum bm_data_
↪type_t )input_dtype);
int buffer_table_size = slice_m * ksub * dtype_size((enum bm_data_type_t F
↪)input_dtype);
int output_codes_size = encode_vec_num * slice_m;

bm_device_mem_t centroids_input_dev, nxcodes_input_dev, buffer_table_dev,
↪ codes_output_dev;

bm_malloc_device_byte(handle, &centroids_input_dev, centroids_size);
bm_malloc_device_byte(handle, &nxcodes_input_dev, nxcodes_size);
bm_malloc_device_byte(handle, &buffer_table_dev, buffer_table_size);
bm_malloc_device_byte(handle, &codes_output_dev, output_codes_size);
bm_memcpy_s2d(handle, centroids_input_dev, centroids_input_sys_fp32);
bm_memcpy_s2d(handle, nxcodes_input_dev, nxcodes_input_sys);

struct timeval t1, t2;
gettimeofday(&t1, NULL);
ret = bmvc_faiss_indexPQ_encode(handle,
    nxcodes_input_dev,
    centroids_input_dev,
    buffer_table_dev,
    codes_output_dev,
    encode_vec_num,
    vec_dims,
    slice_m,
    ksub,
    IP_metric);
gettimeofday(&t2, NULL);
printf("TPU using time(us): %ld(us)\n", (t2.tv_sec - t1.tv_sec) * 1000000 + t2.tv_
↪usec - t1.tv_usec);
printf("TPU using time(ms): %ld(ms)\n", ((t2.tv_sec - t1.tv_sec) * 1000000 + t2.
↪tv_usec - t1.tv_usec) / 1000);

if(ret != BM_SUCCESS){
    bm_free_device(handle, centroids_input_dev);
    bm_free_device(handle, nxcodes_input_dev);
    bm_free_device(handle, buffer_table_dev);
    bm_free_device(handle, codes_output_dev);
    free(centroids_input_sys_fp32);
    free(nxcodes_input_sys);
    free(output_codes_sys);
    bm_dev_free(handle);
    return BM_ERR_FAILURE;
}

```

(续下页)

(接上页)

```

}
bm_memcpy_d2s(handle, output_codes_sys, codes_output_dev);

printf("finish encode\n");
bm_free_device(handle, centroids_input_dev);
bm_free_device(handle, nxcodes_input_dev);
bm_free_device(handle, buffer_table_dev);
bm_free_device(handle, codes_output_dev);
free(centroids_input_sys_fp32);
free(npcodes_input_sys);
free(output_codes_sys);
bm_dev_free(handle);
return 0;
}

```

## 5.26 bmcv\_ldc\_rot

### 【描述】

镜头畸变校正 (LDC) 模块的旋转功能，通过围绕固定点旋转图像，改变其方向与角度，从而使其在平面上发生旋转。

### 【语法】

```

1 bm_status_t bmcv_ldc_rot(bm_handle_t handle,
2                      bm_image in_image,
3                      bm_image out_image,
4                      bmcv_rot_mode rot_mode);

```

### 【参数】

表 5.5: bmcv\_ldc\_rot 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
in_image	输入	输入的待旋转图像，通过调用 bm_image_create 创建
out_image	输出	输出的旋转后图像，通过调用 bm_image_create 创建
rot_mode	输入	旋转类型，取值为 [0, 4] 内的整数，其中 0 为 0°，1 为 90°，2 为 180°，3 为 270°，4 为 xy flip

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【数据类型说明】

```
1 typedef enum bmcv_rot_mode {
2     BMCV_ROTATION_0 = 0,
3     BMCV_ROTATION_90,
4     BMCV_ROTATION_180,
5     BMCV_ROTATION_270,
6     BMCV_ROTATION_XY_FLIP,
7     BMCV_ROTATION_MAX
8 } bmcv_rot_mode;
```

- BMCV\_ROTATION\_0 代表 0 度的旋转, 即图像不进行旋转, 保持原状。
- BMCV\_ROTATION\_90 代表 90 度的旋转, 即图像顺时针旋转 90 度。
- BMCV\_ROTATION\_180 代表 180 度的旋转, 即图像顺时针旋转 180 度。
- BMCV\_ROTATION\_270 代表 270 度的旋转, 即图像顺时针旋转 270 度。
- BMCV\_ROTATION\_XY\_FLIP 代表 XY 翻转, 即图像在 X 轴和 Y 轴上都进行翻转(镜像翻转)。
- BMCV\_ROTATION\_MAX 代表枚举最大值, 用于指示枚举的结束或作为范围检查的标记。

## 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_GRAY

**【注意】**

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
2. 支持图像的分辨率为 64x64~4608x4608，且要求 64 位对齐。
3. 当前 ldc 模块固件原生支持 90° 与 270° 旋转，所以 rot\_mode 选择 0, 1, 3。

**【代码示例】**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 #define LDC_ALIGN 64
8 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
9
10 int main() {
11     int dev_id = 0;
12     int height = 1080, width = 1920;
13     bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_GRAY;
14     bmvc_rot_mode rot_mode = BMVC_ROTATION_0;
15     char *src_name = "path/to/src", *dst_name = "path/to/dst";
16     bm_handle_t handle = NULL;
17     int ret = (int)bm_dev_request(&handle, dev_id);
18     if (ret != 0) {
19         printf("Create bm handle failed. ret = %d\n", ret);
20         exit(-1);
21     }
22
23     bm_image src, dst;
24     int src_stride[4];
25     int dst_stride[4];
26
27     // align
28     int align_width = (width + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
29
30     int data_size = 1;
31     src_stride[0] = align_up(width, 16) * data_size;
32     dst_stride[0] = align_up(align_width, 16) * data_size;
33     // create bm image
34     bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, src_
35     ↪stride);
36     bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst, [F
37     ↪dst_stride]);
38
39     ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
```

(续下页)

(接上页)

```
38     ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
39
40     int image_byte_size[4] = {0};
41     bm_image_get_byte_size(src, image_byte_size);
42     int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
43     ↪byte_size[3];
44     unsigned char *input_data = (unsigned char *)malloc(byte_size);
45     FILE *fp_src = fopen(src_name, "rb");
46     if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
47         printf("file size is less than required bytes%d\n", byte_size);
48     };
49     fclose(fp_src);
50     void* in_ptr[4] = {(void *)input_data,
51                         (void *)((unsigned char *)input_data + image_byte_size[0]),
52                         (void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
53     ↪size[1]),
54                         (void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
55     ↪size[1] + image_byte_size[2])};
56     bm_image_copy_host_to_device(src, in_ptr);
57
58     ret = bmcv_ldc_rot(handle, src, dst, rot_mode);
59
60     bm_image_get_byte_size(dst, image_byte_size);
61     byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
62     ↪size[3];
63     unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
64     void* out_ptr[4] = {(void*)output_ptr,
65                         (void*)((unsigned char*)output_ptr + image_byte_size[0]),
66                         (void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
67     ↪size[1]),
68                         (void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
69     ↪size[1] + image_byte_size[2])};
70     bm_image_copy_device_to_host(src, (void **)out_ptr);
71
72     FILE *fp_dst = fopen(dst_name, "wb");
73     if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
74         printf("file size is less than %d required bytes\n", byte_size);
75     };
76     fclose(fp_dst);
77
78     free(input_data);
79     free(output_ptr);
80 }
```

## 5.27 bmcv\_ldc\_gdc

### 【描述】

镜头畸变校正 (LDC) 模块的几何畸变校正功能，通过校正镜头引起的图像畸变（针对桶形畸变 (Barrel Distortion) 及枕形畸变 (Pincushion Distortion)），使图像中的直线变得更加准确和几何正确，提高图像的质量和可视化效果。

其中，提供两种畸变校正的方式供用户选择，分别为：1. 用户根据图像畸变的类型及校正强度输入配置参数列表对图像进行校正；2. 用户使用 Grid\_Info(输入输出图像坐标映射关系描述) 文件校正图像，以获得更好的图像校正效果。

### 【语法】

```

1 bm_status_t bmcv_ldc_gdc(bm_handle_t      handle,
2                      bm_image       in_image,
3                      bm_image       out_image,
4                      bmcv_gdc_attr ldc_attr);

```

### 【参数】

表 5.6: bmcv\_ldc\_gdc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
in_image	输入	输入的畸变图像，通过调用 bm_image_create 创建
out_image	输出	输出的畸变校正后的图像，通过调用 bm_image_create 创建
ldc_attr	输入	GDC 功能的参数配置列表，包括 bAspect XRatio YRatio XYRatio CenterXOffset CenterYOffset DistortionRatio *grid_info

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【数据类型说明】

表 5.7: bmcv\_gdc\_attr 参数表

参数名称	输入/输出	描述
bAspect	输入	畸变校正的过程中是否保持幅型比, 0: 不保持, 1: 保持
s32XRatio	输入	配置范围:[0, 100], 水平方向视野大小参数, bAspect=0 时有效
s32YRatio	输入	配置范围:[0, 100], 垂直方向视野大小参数, bAspect=0 时有效
s32XYRatio	输入	配置范围:[0, 100], 视野大小参数, bAspect=1 时有效
s32CenterXOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的水平偏移
s32CenterYOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的垂直偏移
s32DistortionRatio	输入	配置范围:[-300, 500], 畸变校正强度参数, 畸变类型为桶形时配置为负, 畸变类型为枕形时配置为负
grid_info	输入	用于存储 grid_info 的信息, 包含 grid_info 的大小和数据

### 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_GRAY

### 【注意】

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。
2. 支持图像的分辨率为 64x64~4608x4608, 且要求 64 位对齐。

3. 若用户决定使用第一种方式进行图像校正，需根据图像畸变的类型及校正强度自行输入配置参数列表 ldc\_attr，此时要将 grid\_info 设置为空。
4. 若用户决定使用第二种方式进行图像校正，需提供 Grid\_Info 文件，具体使用方式请参考下面的代码示例。

### 【代码示例】

#### 1. 通过配置参数列表进行图像校正

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 #define LDC_ALIGN 64
8 #define ALIGN(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
9
10 int main() {
11     int dev_id = 0;
12     int height = 1080, width = 1920;
13     bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_GRAY;
14     bmvc_gdc_attr stLDCAttr = {0};
15     char *src_name = "path/to/src", *dst_name = "path/to/dst";
16     bm_handle_t handle = NULL;
17     int ret = (int)bm_dev_request(&handle, dev_id);
18     if (ret != 0) {
19         printf("Create bm handle failed. ret = %d\n", ret);
20         exit(-1);
21     }
22     bm_image src, dst;
23     int src_stride[4];
24     int dst_stride[4];
25     // align
26     int align_height = (height + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
27     int align_width = (width + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
28
29     // calc image stride
30     int data_size = 1;
31     src_stride[0] = ALIGN(width, 16) * data_size;
32     dst_stride[0] = ALIGN(align_width, 16) * data_size;
33     // create bm image
34     bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, src_
35     ↪stride);
36     bm_image_create(handle, align_height, align_width, dst_fmt, DATA_TYPE_EXT_1N_
37     ↪BYTE, &dst, dst_stride);
```

(续下页)

(接上页)

```

37     ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
38     ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
39
40     int image_byte_size[4] = {0};
41     bm_image_get_byte_size(src, image_byte_size);
42     int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
43     →byte_size[3];
44     unsigned char *input_data = (unsigned char *)malloc(byte_size);
45     FILE *fp_src = fopen(src_name, "rb");
46     if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
47         printf("file size is less than required bytes%d\n", byte_size);
48     };
49     fclose(fp_src);
50     void* in_ptr[4] = {(void *)input_data,
51                         (void *)((unsigned char *)input_data + image_byte_size[0]),
52                         (void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
53     →size[1]),
54                         (void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
55     →size[1] + image_byte_size[2])};
56     bm_image_copy_host_to_device(src, in_ptr);
57
58     ret = bmcv_ldc_gdc(handle, src, dst, stLDCAttr);
59
60     bm_image_get_byte_size(src, image_byte_size);
61     byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
62     →size[3];
63     unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
64     void* out_ptr[4] = {(void *)output_ptr,
65                         (void *)((unsigned char *)output_ptr + image_byte_size[0]),
66                         (void *)((unsigned char *)output_ptr + image_byte_size[0] + image_byte_
67     →size[1]),
68                         (void *)((unsigned char *)output_ptr + image_byte_size[0] + image_byte_
69     →size[1] + image_byte_size[2])};
70     bm_image_copy_device_to_host(src, (void **)out_ptr);
71
72     FILE *fp_dst = fopen(dst_name, "wb");
73     if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
74         printf("file size is less than %d required bytes\n", byte_size);
75     };
76     fclose(fp_dst);
77
78     free(input_data);
79     free(output_ptr);
80
81     bm_image_destroy(&src);
82     bm_image_destroy(&dst);
83
84     bm_dev_free(handle);
85
86     return 0;

```

(续下页)

## 5.28 bmcv\_ldc\_load\_mesh

### 【描述】

镜头畸变校正 (LDC) 模块的几何畸变校正功能，通过校正镜头引起的图像畸变（针对桶形畸变 (Barrel Distortion) 及枕形畸变 (Pincushion Distortion)），使图像中的直线变得更加准确和几何正确，提高图像的质量和可视化效果。在这个过程中，需要通过 CPU 计算 MESH 表用于后续的畸变校正，由于该过程较为耗时，因此可以通过该函数直接加载已有的 MESH 表进行图像的畸变矫正。

### 【语法】

```

1 bm_status_t bmcv_ldc_gdc_load_mesh(bm_handle_t handle,
2                                bm_image in_image,
3                                bm_image out_image,
4                                bm_device_mem_t dmem);
```

### 【参数】

表 5.8: bmcv\_ldc\_gdc\_load\_mesh 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取
in_image	输入	输入的畸变图像，通过调用 <code>bm_image_create</code> 创建
out_image	输出	输出的畸变校正后的图像，通过调用 <code>bm_image_create</code> 创建
dmem	输入	用于存储 MESH 表的设备内存，通过调用 <code>bm_malloc_device_byte</code> 创建

### 【返回值】

该函数成功调用时，返回 `BM_SUCCESS`。

## 【格式支持】

- 输入和输出的数据类型：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

- 输入和输出的色彩格式必须保持一致，可支持：

num	image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_GRAY

## 【注意】

- 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
- 支持图像的分辨率为 64x64~4608x4608，且要求 64 位对齐。
- 若要保证几何畸变校正的效果，用户需确认 MESH 表的正确性。

## 【代码示例】

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6 #include <errno.h>
7
8 #define TILESIZE 64 // HW: data Tile Size
9 #define LDC_ALIGN 64
10 #define HW_MESH_SIZE 8
11 #define MESH_NUM_ATILE (TILESIZE / HW_MESH_SIZE) // how many mesh in A TILE
12
13 typedef unsigned int u32;
14 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
15
16 typedef struct COORD2D_INT_HW {
17     unsigned char xcor[3]; // s13.10, 24bit
18 } __attribute__((packed)) COORD2D_INT_HW;
19

```

(续下页)

(接上页)

```

20 void test_mesh_gen_get_1st_size(u32 u32Width, u32 u32Height, u32 *mesh_1st_size)
21 {
22     if (!mesh_1st_size)
23         return;
24
25     u32 ori_src_width = u32Width;
26     u32 ori_src_height = u32Height;
27
28     // In LDC Processing, width & height aligned to TILESIZE
29     u32 src_width_s1 = ((ori_src_width + TILESIZE - 1) / TILESIZE) * TILESIZE;
30     u32 src_height_s1 = ((ori_src_height + TILESIZE - 1) / TILESIZE) * TILESIZE;
31
32     // modify frame size
33     u32 dst_height_s1 = src_height_s1;
34     u32 dst_width_s1 = src_width_s1;
35     u32 num_tilex_s1 = dst_width_s1 / TILESIZE;
36     u32 num_tiley_s1 = dst_height_s1 / TILESIZE;
37
38     // 4 = 4 knots in a mesh
39     *mesh_1st_size = sizeof(struct COORD2D_INT_HW) * MESH_NUM_ATILE * MESH_
40     →NUM_ATILE * num_tilex_s1 * num_tiley_s1 * 4;
41 }
42
43 void test_mesh_gen_get_2nd_size(u32 u32Width, u32 u32Height, u32 *mesh_2nd_size)
44 {
45     if (!mesh_2nd_size)
46         return;
47
48     u32 ori_src_width = u32Width;
49     u32 ori_src_height = u32Height;
50
51     // In LDC Processing, width & height aligned to TILESIZE
52     u32 src_width_s1 = ((ori_src_width + TILESIZE - 1) / TILESIZE) * TILESIZE;
53     u32 src_height_s1 = ((ori_src_height + TILESIZE - 1) / TILESIZE) * TILESIZE;
54
55     // modify frame size
56     u32 dst_height_s1 = src_height_s1;
57     u32 dst_width_s1 = src_width_s1;
58     u32 src_height_s2 = dst_width_s1;
59     u32 src_width_s2 = dst_height_s1;
60     u32 dst_height_s2 = src_height_s2;
61     u32 dst_width_s2 = src_width_s2;
62     u32 num_tilex_s2 = dst_width_s2 / TILESIZE;
63     u32 num_tiley_s2 = dst_height_s2 / TILESIZE;
64
65     // 4 = 4 knots in a mesh
66     *mesh_2nd_size = sizeof(struct COORD2D_INT_HW) * MESH_NUM_ATILE * MESH_
67     →NUM_ATILE * num_tilex_s2 * num_tiley_s2 * 4;
68 }
```

(续下页)

(接上页)

```

69             u32 u32Height,
70             u32 *mesh_1st_size,
71             u32 *mesh_2nd_size)
72 {
73     if (!mesh_1st_size || !mesh_2nd_size)
74         return;
75
76     test_mesh_gen_get_1st_size(u32Width, u32Height, mesh_1st_size);
77     test_mesh_gen_get_2nd_size(u32Width, u32Height, mesh_2nd_size);
78 }
79
80 int main() {
81     int dev_id = 0;
82     int height = 1080, width = 1920;
83     bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_GRAY;
84     char *src_name = "path/to/src", *dst_name = "path/to/dst";
85     bm_handle_t handle = NULL;
86     int ret = (int)bm_dev_request(&handle, dev_id);
87     if (ret != 0) {
88         printf("Create bm handle failed. ret = %d\n", ret);
89         exit(-1);
90     }
91
92     bm_image src, dst;
93     int src_stride[4];
94     int dst_stride[4];
95
96     // align
97     int align_height = (height + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
98     int align_width = (width + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
99
100    // calc image stride
101    int data_size = 1;
102    src_stride[0] = align_up(width, 16) * data_size;
103    dst_stride[0] = align_up(align_width, 16) * data_size;
104    // create bm image
105    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, src_
106    ↪stride);
106    bm_image_create(handle, align_height, align_width, dst_fmt, DATA_TYPE_EXT_1N_
107    ↪BYTE, &dst, dst_stride);
107
108    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
109    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
110
111    int image_byte_size[4] = {0};
112    bm_image_get_byte_size(src, image_byte_size);
113    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
114    ↪byte_size[3];
114    unsigned char *input_data = (unsigned char *)malloc(byte_size);
115    FILE *fp_src = fopen(src_name, "rb");
115    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
116

```

(续下页)

(接上页)

```
117     printf("file size is less than required bytes%d\n", byte_size);
118 };
119 fclose(fp_src);
120 void* in_ptr[4] = {(void *)input_data,
121                     ((void *)((unsigned char*)input_data + image_byte_size[0])),
122                     ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
123 →size[1])),
124                     ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
125 →size[1] + image_byte_size[2]))};
126 bm_image_copy_host_to_device(src, in_ptr);
127
128 bm_device_mem_t dmem;
129 u32 mesh_1st_size = 0, mesh_2nd_size = 0;
130 test_mesh_gen_get_size(width, height, &mesh_1st_size, &mesh_2nd_size);
131 u32 mesh_size = mesh_1st_size + mesh_2nd_size;
132 ret = bm_malloc_device_byte_heap(handle, &dmem, BMCV_HEAP1_ID, mesh_size);
133
134 bm_image_get_byte_size(src, image_byte_size);
135 byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
136 →size[3];
137 unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
138 void* out_ptr[4] = {(void*)output_ptr,
139                     ((void*)((unsigned char*)output_ptr + image_byte_size[0])),
140                     ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
141 →size[1])),
142                     ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
143 →size[1] + image_byte_size[2]))};
143 bm_image_copy_device_to_host(src, (void **)out_ptr);
144
145 FILE *fp_dst = fopen(dst_name, "wb");
146 if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
147     printf("file size is less than %d required bytes\n", byte_size);
148 }
149 fclose(fp_dst);
150
151 free(input_data);
152 free(output_ptr);
153
154 bm_image_destroy(&src);
155 bm_image_destroy(&dst);
156
157 bm_dev_free(handle);
158
159 return 0;
160 }
```

## 5.29 bmcv\_ldc\_gen\_mesh

### 【描述】

镜头畸变校正 (LDC) 模块的几何畸变校正功能，通过校正镜头引起的图像畸变（针对桶形畸变 (Barrel Distortion) 及枕形畸变 (Pincushion Distortion)），使图像中的直线变得更加准确和几何正确，提高图像的质量和可视化效果。在这个过程中，需要通过 CPU 计算 MESH 表用于后续的畸变校正，因此可以通过该函数计算并保存 MESH 表用于后续的畸变校正。

### 【语法】

```

1 bm_status_t bmcv_ldc_gdc_gen_mesh(bm_handle_t handle,
2                                bm_image in_image,
3                                bm_image out_image,
4                                bmcv_gdc_attr ldc_attr,
5                                bm_device_mem_t dmem);

```

### 【参数】

表 5.9: bmcv\_ldc\_gdc\_gen\_mesh 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
in_image	输入	输入的畸变图像，通过调用 bm_image_create 创建
out_image	输出	输出的畸变校正后的图像，通过调用 bm_image_create 创建
ldc_attr	输入	GDC 功能的参数配置列表，包括 bAspect XRatio YRatio XYRatio CenterXOffset CenterYOffset DistortionRatio
dmem	输入	用于存储 MESH 表的设备内存，通过调用 bm_malloc_device_byte 创建

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【数据类型说明】

表 5.10: bmcv\_gdc\_attr 参数表

参数名称	输入/输出	描述
bAspect	输入	畸变校正的过程中是否保持幅型比, 0: 不保持, 1: 保持
s32XRatio	输入	配置范围:[0, 100], 水平方向视野大小参数, bAspect=0 时有效
s32YRatio	输入	配置范围:[0, 100], 垂直方向视野大小参数, bAspect=0 时有效
s32XYRatio	输入	配置范围:[0, 100], 视野大小参数, bAspect=1 时有效
s32CenterXOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的水平偏移
s32CenterYOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的垂直偏移
s32DistortionRatio	输入	配置范围:[-300, 500], 畸变校正强度参数, 畸变类型为桶形时配置为负, 畸变类型为枕形时配置为负
grid_info	输入	用于存储 grid_info 的信息, 包含 grid_info 的大小和数据

### 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_NV12
2	FORMAT_NV21
3	FORMAT_GRAY

### 【注意】

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。
2. 支持图像的分辨率为 64x64~4608x4608, 且要求 64 位对齐。

3. 用户需根据图像畸变的类型及校正强度自行输入配置 GDC 的参数配置列表 ldc\_attr，此时要将 grid\_info 设置为空。

### 【代码示例】

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmcv_api_ext_c.h"
5 #include <unistd.h>
6 #include <inttypes.h>
7
8 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
9 #define HW_MESH_SIZE 8
10
11 #define TILESIZE 64 // HW: data Tile Size
12 #define MESH_NUM_ATILE (TILESIZE / HW_MESH_SIZE) // how many mesh in A TILE
13 #define LDC_ALIGN 64
14 typedef unsigned int      u32;
15 typedef unsigned char     u8;
16
17 typedef struct COORD2D_INT_HW {
18     u8 xcor[3]; // s13.10, 24bit
19 } __attribute__((packed)) COORD2D_INT_HW;
20
21 void test_mesh_gen_get_1st_size(u32 u32Width, u32 u32Height, u32 *mesh_1st_size)
22 {
23     if (!mesh_1st_size)
24         return;
25
26     u32 ori_src_width = u32Width;
27     u32 ori_src_height = u32Height;
28
29     // In LDC Processing, width & height aligned to TILESIZE
30     u32 src_width_s1 = ((ori_src_width + TILESIZE - 1) / TILESIZE) * TILESIZE;
31     u32 src_height_s1 = ((ori_src_height + TILESIZE - 1) / TILESIZE) * TILESIZE;
32
33     // modify frame size
34     u32 dst_height_s1 = src_height_s1;
35     u32 dst_width_s1 = src_width_s1;
36     u32 num_tilex_s1 = dst_width_s1 / TILESIZE;
37     u32 num_tiley_s1 = dst_height_s1 / TILESIZE;
38
39     // 4 = 4 knots in a mesh
40     *mesh_1st_size = sizeof(struct COORD2D_INT_HW) * MESH_NUM_ATILE * MESH_
41     →NUM_ATILE * num_tilex_s1 * num_tiley_s1 * 4;
42 }
43
44 void test_mesh_gen_get_2nd_size(u32 u32Width, u32 u32Height, u32 *mesh_2nd_size)
```

(续下页)

(接上页)

```

44 {
45     if (!mesh_2nd_size)
46         return;
47
48     u32 ori_src_width = u32Width;
49     u32 ori_src_height = u32Height;
50
51     // In LDC Processing, width & height aligned to TILESIZE
52     u32 src_width_s1 = ((ori_src_width + TILESIZE - 1) / TILESIZE) * TILESIZE;
53     u32 src_height_s1 = ((ori_src_height + TILESIZE - 1) / TILESIZE) * TILESIZE;
54
55     // modify frame size
56     u32 dst_height_s1 = src_height_s1;
57     u32 dst_width_s1 = src_width_s1;
58     u32 src_height_s2 = dst_width_s1;
59     u32 src_width_s2 = dst_height_s1;
60     u32 dst_height_s2 = src_height_s2;
61     u32 dst_width_s2 = src_width_s2;
62     u32 num_tilex_s2 = dst_width_s2 / TILESIZE;
63     u32 num_tiley_s2 = dst_height_s2 / TILESIZE;
64
65     // 4 = 4 knots in a mesh
66     *mesh_2nd_size = sizeof(struct COORD2D_INT_HW) * MESH_NUM_ATILE * MESH_
67     →NUM_ATILE * num_tilex_s2 * num_tiley_s2 * 4;
68 }
69
70 void test_mesh_gen_get_size(u32 u32Width,
71                             u32 u32Height,
72                             u32 *mesh_1st_size,
73                             u32 *mesh_2nd_size)
74 {
75     if (!mesh_1st_size || !mesh_2nd_size)
76         return;
77
78     test_mesh_gen_get_1st_size(u32Width, u32Height, mesh_1st_size);
79     test_mesh_gen_get_2nd_size(u32Width, u32Height, mesh_2nd_size);
80 }
81
82 int main() {
83     int dev_id = 0;
84     int height = 1080, width = 1920;
85     bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_GRAY;
86     bmcv_gdc_attr stLDCAttr = {0};
87     char *src_name = "path/to/src";
88     bm_handle_t handle = NULL;
89     int ret = (int)bm_dev_request(&handle, dev_id);
90     if (ret != 0) {
91         printf("Create bm handle failed. ret = %d\n", ret);
92         exit(-1);
93     }

```

(续下页)

(接上页)

```

94
95     bm_image src, dst;
96     int src_stride[4];
97     int dst_stride[4];
98
99     // align
100    int align_height = (height + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
101    int align_width = (width + (LDC_ALIGN - 1)) & ~(LDC_ALIGN - 1);
102
103    // calc image stride
104    int data_size = 1;
105    src_stride[0] = align_up(width, 64) * data_size;
106    dst_stride[0] = align_up(width, 64) * data_size;
107    // create bm image
108    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, src_
→stride);
109    bm_image_create(handle, align_height, align_width, dst_fmt, DATA_TYPE_EXT_1N_
→BYTE, &dst, dst_stride);
110
111    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
112    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
113
114    int byte_size = width * height;
115    unsigned char *input_data = (unsigned char *)malloc(byte_size);
116    FILE *fp_src = fopen(src_name, "rb");
117    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
118        printf("file size is less than required bytes%d\n", byte_size);
119    };
120    fclose(fp_src);
121    void *in_ptr[1] = {(void *)input_data};
122    bm_image_copy_host_to_device(src, in_ptr);
123
124    bm_device_mem_t dmem;
125    u32 mesh_1st_size = 1, mesh_2nd_size = 1;
126    test_mesh_gen_get_size(width, height, &mesh_1st_size, &mesh_2nd_size);
127    u32 mesh_size = mesh_1st_size + mesh_2nd_size;
128    ret = bm_malloc_device_byte(handle, &dmem, mesh_size);
129
130    ret = bmcv_ldc_gdc_gen_mesh(handle, src, dst, stLDCAttr, dmem);
131    unsigned char *buffer = (unsigned char *)malloc(mesh_size);
132    memset(buffer, 0, mesh_size);
133    ret = bm_memcpy_d2s(handle, (void *)buffer, dmem);
134
135    char mesh_name[128];
136    snprintf(mesh_name, 128, "./test_mesh_%dx%d_%d_%d_%d_%d_%d_%d.mesh"
137           , width, height, stLDCAttr.bAspect, stLDCAttr.s32XRatio, stLDCAttr.s32YRatio
138           , stLDCAttr.s32XYRatio, stLDCAttr.s32CenterXOffset, stLDCAttr.s32CenterYOffset, F
→stLDCAttr.s32DistortionRatio);
139
140    FILE *fp = fopen(mesh_name, "wb");
141    if (fwrite((void *)buffer, mesh_size, 1, fp) != 1) {

```

(续下页)

(接上页)

```

142     printf("fwrite mesh data error\n");
143     free(buffer);
144 }
145 fclose(fp);
146 free(buffer);
147
148 bm_image_destroy(&src);
149 bm_image_destroy(&dst);
150
151 bm_dev_free(handle);
152
153 return 0;
154 }
```

### 5.30 bmcv\_dwa\_rot

#### 【描述】

去畸变仿射 (DWA) 模块的旋转功能，通过围绕固定点旋转图像，改变其方向与角度，从而使其在平面上发生旋转。

#### 【语法】

```

1 bm_status_t bmcv_dwa_rot( bm_handle_t      handle,
2                           bm_image       input_image,
3                           bm_image       output_image,
4                           bmcv_rot_mode rot_mode);
```

#### 【参数】

表 5.11: bmcv\_dwa\_rot 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取
input_image	输入	输入的待旋转图像，通过调用 <code>bm_image_create</code> 创建
output_image	输出	输出的旋转后图像，通过调用 <code>bm_image_create</code> 创建
rot_mode	输入	旋转类型，取值为 [0, 4] 内的整数，其中 0 为 0°，1 为 90°，2 为 180°，3 为 270°，4 为 xy flip

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【数据类型说明】

```
1 typedef enum bmcv_rot_mode_ {
2     BMCV_ROTATION_0 = 0,
3     BMCV_ROTATION_90,
4     BMCV_ROTATION_180,
5     BMCV_ROTATION_270,
6     BMCV_ROTATION_XY_FLIP,
7     BMCV_ROTATION_MAX
8 } bmcv_rot_mode;
```

- BMCV\_ROTATION\_0 代表 0 度的旋转, 即图像不进行旋转, 保持原状。
- BMCV\_ROTATION\_90 代表 90 度的旋转, 即图像顺时针旋转 90 度。
- BMCV\_ROTATION\_180 代表 180 度的旋转, 即图像顺时针旋转 180 度。
- BMCV\_ROTATION\_270 代表 270 度的旋转, 即图像顺时针旋转 270 度。
- BMCV\_ROTATION\_XY\_FLIP 代表 XY 翻转, 即图像在 X 轴和 Y 轴上都进行翻转(镜像翻转)。
- BMCV\_ROTATION\_MAX 代表枚举最大值, 用于指示枚举的结束或作为范围检查的标记。

## 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_YUV420P
3	FORMAT_YUV444P
4	FORMAT_GRAY

**【注意】**

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
2. 支持图像的分辨率为 32x32~4096x4096，且要求 32 对齐。

**【代码示例】**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 int main() {
8     int src_h = 1080, src_w = 1920, dev_id = 0;
9     bm_image_format_ext fmt = FORMAT_GRAY;
10    char *src_name = "path/to/src", *dst_name = "path/to/dst";
11    bm_handle_t handle = NULL;
12    bmcv_rot_mode rot_mode = BMCV_ROTATION_0;
13    bm_status_t ret;
14    bm_image src, dst;
15    int dst_w, dst_h;
16    ret = bm_dev_request(&handle, dev_id);
17
18    bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
19    dst_w = src_w;
20    dst_h = src_h;
21
22    bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
23
24    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
25    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
26
27    int image_byte_size[4] = {0};
28    bm_image_get_byte_size(src, image_byte_size);
29    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
30    ↪byte_size[3];
31    unsigned char *input_data = (unsigned char *)malloc(byte_size);
32    FILE *fp_src = fopen(src_name, "rb");
33    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
34        printf("file size is less than required bytes%d\n", byte_size);
35    };
36    fclose(fp_src);
37    void* in_ptr[4] = {(void *)input_data,
38                      ((void *)((unsigned char*)input_data + image_byte_size[0])),
39                      ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
39    ↪size[1])),
```

(续下页)

(接上页)

```
39         (void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
40 ->size[1] + image_byte_size[2])};
41     bm_image_copy_host_to_device(src, in_ptr);
42
43     bmcv_dwa_rot(handle, src, dst, rot_mode);
44
45     bm_image_get_byte_size(src, image_byte_size);
46     byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
47 ->size[3];
48     unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
49     void* out_ptr[4] = {(void*)output_ptr,
50             (void*)((unsigned char*)output_ptr + image_byte_size[0]),
51             (void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
52 ->size[1]),
53             (void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
54 ->size[1] + image_byte_size[2])};
54     bm_image_copy_device_to_host(src, (void **)out_ptr);
55
56     FILE *fp_dst = fopen(dst_name, "wb");
57     if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
58         printf("file size is less than %d required bytes\n", byte_size);
59     };
60     fclose(fp_dst);
61
62     free(input_data);
63     free(output_ptr);
64     bm_image_destroy(&src);
65     bm_image_destroy(&dst);
66
67     bm_dev_free(handle);
68
69     return ret;
70 }
```

### 5.31 bmcv\_dwa\_gdc

#### 【描述】

去畸变仿射 (DWA) 模块的几何畸变校正功能，通过校正镜头引起的图像畸变（针对桶形畸变 (Barrel Distortion) 及枕形畸变 (Pincushion Distortion)），使图像中的直线变得更加准确和几何正确，提高图像的质量和可视化效果。

其中，提供两种畸变校正的方式供用户选择，分别为：1. 用户根据图像畸变的类型及校正强度输入配置参数列表对图像进行校正；2. 用户使用 Grid\_Info(输入输出图像坐标映射关系描述) 文件校正图像，以获得更好的图像校正效果。

#### 【语法】

```
1 bm_status_t bmcv_dwa_gdc(bm_handle_t handle,
2                                bm_image      input_image,
3                                bm_image      output_image,
4                                bmcv_gdc_attr ldc_attr);
```

## 【参数】

表 5.12: bmcv\_dwa\_gdc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取
input_image	输入	输入的畸变图像，通过调用 <code>bm_image_create</code> 创建
output_image	输出	输出的畸变校正后的图像，通过调用 <code>bm_image_create</code> 创建
ldc_attr	输入	GDC 功能的参数配置列表，包括 <code>bAspect</code> <code>s32XRatio</code> <code>s32YRatio</code> <code>s32XYRatio</code> <code>s32CenterXOffset</code> <code>s32CenterYOffset</code> <code>s32DistortionRatio</code> <code>*grid_info</code>

## 【返回值】

该函数成功调用时，返回 `BM_SUCCESS`。

## 【数据类型说明】

表 5.13: bmcv\_gdc\_attr 参数表

参数名称	输入/输出	描述
bAspect	输入	畸变校正的过程中是否保持幅型比, 0: 不保持, 1: 保持
s32XRatio	输入	配置范围:[0, 100], 水平方向视野大小参数, bAspect=0 时有效
s32YRatio	输入	配置范围:[0, 100], 垂直方向视野大小参数, bAspect=0 时有效
s32XYRatio	输入	配置范围:[0, 100], 视野大小参数, bAspect=1 时有效
s32CenterXOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的水平偏移
s32CenterYOffset	输入	配置范围:[-511, 511], 畸变中心点相对于图像中心点的垂直偏移
s32DistortionRatio	输入	配置范围:[-300, 500], 畸变校正强度参数, 畸变类型为桶形时配置为负, 畸变类型为枕形时配置为负
grid_info	输入	用于存储 grid_info 的信息, 包含 grid_info 的大小和数据

**【格式支持】**

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_YUV420P
3	FORMAT_YUV444P
4	FORMAT_GRAY

**【注意】**

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。
2. 支持图像的分辨率为 32x32~4096x4096, 且要求宽 64 stride 对齐。

3. 若用户决定使用第一种方式进行图像校正，需根据图像畸变的类型及校正强度自行输入配置参数列表 ldc\_attr，此时要将 grid\_info 设置为空。
4. 若用户决定使用第二种方式进行图像校正，需提供 Grid\_Info 文件，具体使用方式请参考下面的代码示例。

### 【代码示例】

#### 1. 通过配置参数列表进行图像校正

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmcv_api_ext_c.h"
5 #include <unistd.h>
6
7 int main() {
8     int src_h = 1080, src_w = 1920, dev_id = 0;
9     bm_image_format_ext fmt = FORMAT_YUV420P;
10    char *src_name = "path/to/src", *dst_name = "path/to/dst";
11    bm_handle_t handle = NULL;
12    bmcv_gdc_attr ldc_attr = {true, 0, 0, 0, 0, 0, -200, };
13    fmt = FORMAT_RGB_PLANAR;
14    ldc_attr.grid_info.size = 0;
15    ldc_attr.grid_info.u.system.system_addr = NULL;
16    int ret = (int)bm_dev_request(&handle, dev_id);
17    if (ret != 0) {
18        printf("Create bm handle failed. ret = %d\n", ret);
19        exit(-1);
20    }
21
22    bm_image src, dst;
23    int dst_w, dst_h;
24
25    bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
26
27    dst_w = src_w;
28    dst_h = src_h;
29    bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
30
31    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
32    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
33
34    int image_byte_size[4] = {0};
35    bm_image_get_byte_size(src, image_byte_size);
36    for (int i = 0; i < 4; i++) {
37        printf("image_byte_size[%d] is : %d\n", i, image_byte_size[i]);
38    }
```

(续下页)

(接上页)

```

39     int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
40     →byte_size[3];
41     // int byte_size = src_w * src_h * 3 / 2;
42     unsigned char *input_data = (unsigned char *)malloc(byte_size);
43     FILE *fp_src = fopen(src_name, "rb");
44     if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
45         printf("file size is less than required bytes\n", byte_size);
46     };
47     fclose(fp_src);
48     bm_image_copy_host_to_device(src, (void *)&input_data);
49
50     bmcv_dwa_gdc(handle, src, dst, ldc_attr);
51
52     unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
53     void* out_ptr[4] = {(void*)output_ptr,
54                         ((void*)((unsigned char*)output_ptr + dst_w * dst_h)),
55                         ((void*)((unsigned char*)output_ptr + 5 / 4 * dst_w * dst_w))};
56     bm_image_copy_device_to_host(dst, (void **)out_ptr);
57
58     FILE *fp_dst = fopen(dst_name, "wb");
59     if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
60         printf("file size is less than %d required bytes\n", byte_size);
61     };
62     fclose(fp_dst);
63
64     free(input_data);
65     free(output_ptr);
66     bm_image_destroy(&src);
67     bm_image_destroy(&dst);
68
69     bm_dev_free(handle);
70
71     return 0;
}

```



## 2. 通过 Grid\_Info 文件进行图像校正

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmcv_api_ext_c.h"
5 #include <unistd.h>
6
7 int main() {
8     int src_h = 1080, src_w = 1920, dst_h = 1080, dst_w = 1920, dev_id = 0;
9     bm_image_format_ext fmt = FORMAT_YUV420P;
10    char *src_name = "path/to/src", *dst_name = "path/to/dst", *grid_name = "path/to/grid_"

```

(续下页)

(接上页)

```

11  ↳info";
12   bm_handle_t handle = NULL;
13   bmcv_gdc_attr ldc_attr = {true, 0, 0, 0, 0, 0, -200, };
14   fmt = FORMAT_RGB_PLANAR;
15   ldc_attr.grid_info.size = 0;
16   ldc_attr.grid_info.u.system.system_addr = NULL;
17   int ret = (int)bm_dev_request(&handle, dev_id);
18   if (ret != 0) {
19     printf("Create bm handle failed. ret = %d\n", ret);
20     exit(-1);
21   }
22
23   FILE *fp = fopen(grid_name, "rb");
24   if (!fp) {
25     printf("open file:%s failed.\n", grid_name);
26     exit(-1);
27   }
28   u32 grid_size = 32768; // grid_info文件的字节数
29   char *grid_data = (char *)malloc(grid_size);
30   fread(grid_data, 1, grid_size, fp);
31
32   fclose(fp);
33
34   bm_image src, dst;
35   bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYT
36   e, &src, NULL);
37   bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYT
38   e, &dst, NULL);
39
40   int image_byte_size[4] = {0};
41   bm_image_get_byte_size(src, image_byte_size);
42   for (int i = 0; i < 4; i++) {
43     printf("image_byte_size[%d] is : %d\n", i, image_byte_size[i]);
44   }
45   int byte_size = image_byte_size[0] + image_byte_size[1] + image_
46   ↳byte_size[2];
47   // int byte_size = src_w * src_h * 3 / 2;
48   unsigned char *input_data = (unsigned char *)malloc(byte_size);
49   FILE *fp_src = fopen(src_name, "rb");
50   if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
51     printf("file size is less than required bytes%d\n", byte_size);
52   };
53   fclose(fp_src);
54   bm_image_copy_host_to_device(src, (void *)&input_data);
55
56   ldc_attr.grid_info.u.system.system_addr = (void *)grid_data;
57   ldc_attr.grid_info.size = grid_size;
58
59   bmcv_dwa_gdc(handle, src, dst, ldc_attr);

```

(续下页)

(接上页)

```
60 unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
61 void* out_ptr[4] = { (void*)output_ptr,
62                     ((void*)((unsigned char*)output_ptr + dst_w * dst_h)),
63                     ((void*)((unsigned char*)output_ptr + 5 / 4 * dst_w * dst_w))};
64 bm_image_copy_device_to_host(dst, (void**)out_ptr);
65
66 FILE *fp_dst = fopen(dst_name, "wb");
67 if (fwrite((void*)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
68     printf("file size is less than %d required bytes\n", byte_size);
69 }
70 fclose(fp_dst);
71
72 free(input_data);
73 free(output_ptr);
74 bm_image_destroy(&src);
75 bm_image_destroy(&dst);
76
77 bm_dev_free(handle);
78
79 return 0;
80 }
```

### 5.32 bmcv\_dwa\_affine

#### 【描述】

去畸变仿射 (DWA) 模块的仿射校正功能，通过线性组合和平移操作，来保持图像中的平行线仍然平行，并保持源图像两点之间的距离比例，从而对图像进行旋转、缩放、平移和倾斜等变换。

#### 【语法】

```
1 bm_status_t bmcv_dwa_affine(bm_handle_t handle,
2                                bm_image input_image,
3                                bm_image output_image,
4                                bmcv_affine_attr_s affine_attr);
```

#### 【参数】

表 5.14: bmcv\_dwa\_affine 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
input_image	输入	输入的源图像，通过调用 bm_image_create 创建
output_image	输出	输出的仿射变化校正后的图像，通过调用 bm_image_create 创建
affine_attr	输入	affine 功能的参数配置列表，包括 u32RegionNum astRegionAttr stDestSize

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【数据类型说明】

表 5.15: bmcv\_affine\_attr\_s 参数表

参数名称	输入/输出	描述
u32RegionNum	输入	配置范围:[1, 32]，在源图像中进行 Affine 操作的区域数量
astRegionAttr	输入	结构体数组，存储源图像中每个 Affine 区域的四个顶点坐标
stDestSize	输入	结构体，存储 Affine 操作后的目标区域尺寸

### 【格式支持】

1. 输入和输出的数据类型：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致，可支持：

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_YUV420P
3	FORMAT_YUV444P
4	FORMAT_GRAY

**【注意】**

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败；
2. 支持图像的分辨率为 32x32~4096x4096，且要求 32 对齐；
3. 用户需根据源图像中想要进行 Affine 操作的位置及目标区域尺寸自行输入配置参数列表 affine\_attr；
4. astRegionAttr 数组中，原图中某区域四个顶点的坐标顺序分别为左上、右上、左下、右下；
5. 推荐 stDestSize 的 width 与输出图像 output\_image 的 width 相同，否则结果需要裁剪。

**【代码示例】**

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmcv_api_ext_c.h"
5 #include <unistd.h>
6
7 int main() {
8     int src_h = 1080, src_w = 1920, dst_w = 1920, dst_h = 1080, dev_id = 0;
9     bm_image_format_ext fmt = FORMAT_YUV420P;
10    char *src_name = "path/to/src", *dst_name = "path/to/dst";
11    bm_handle_t handle = NULL;
12    bmcv_affine_attr_s affine_attr = {0};
13    affine_attr.u32RegionNum = 4;
14    affine_attr.stDestSize.u32Width = 400;
15    affine_attr.stDestSize.u32Height = 400;
16    // bmcv_point2f_s faces[9][4] = {0};
17    bmcv_point2f_s faces[9][4] = {
18        { .x = 722.755, .y = 65.7575}, { .x = 828.402, .y = 80.6858}, { .x = 707.827, .y = 171.405}, { .
19        .x = 813.474, .y = 186.333 } },
20        { .x = 494.919, .y = 117.918}, { .x = 605.38, .y = 109.453}, { .x = 503.384, .y = 228.378}, { .
21        .x = 613.845, .y = 219.913 } },
22        { .x = 1509.06, .y = 147.139}, { .x = 1592.4, .y = 193.044}, { .x = 1463.15, .y = 230.48 }, { .

```

(续下页)

(接上页)

```

21    →x = 1546.5, .y = 276.383} },
22    { {.x = 1580.21, .y = 66.7939}, {.x = 1694.1, .y = 70.356}, {.x = 1576.65, .y = 180.682}, {.x = 1690.54, .y = 184.243},
23    { {.x = 178.76, .y = 90.4814}, {.x = 286.234, .y = 80.799}, {.x = 188.442, .y = 197.955}, {.x = 295.916, .y = 188.273},
24    { {.x = 1195.57, .y = 139.226}, {.x = 1292.69, .y = 104.122}, {.x = 1230.68, .y = 236.34}, {.x = 1327.79, .y = 201.236},
25    { {.x = 398.669, .y = 109.872}, {.x = 501.93, .y = 133.357}, {.x = 375.184, .y = 213.133}, {.x = 478.445, .y = 236.618},
26    { {.x = 845.989, .y = 94.591}, {.x = 949.411, .y = 63.6143}, {.x = 876.966, .y = 198.013}, {.x = 980.388, .y = 167.036},
27    { {.x = 1060.19, .y = 58.7882}, {.x = 1170.61, .y = 61.9105}, {.x = 1057.07, .y = 169.203}, {.x = 1167.48, .y = 172.325},
28    };
29    memcpy(affine_attr.astRegionAttr, faces, sizeof(faces));
30
31    int ret = (int)bm_dev_request(&handle, dev_id);
32    if (ret != 0) {
33        printf("Create bm handle failed. ret = %d\n", ret);
34        exit(-1);
35    }
36
37    bm_image src, dst;
38
39    bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
40    bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
41
42    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
43    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
44
45    int image_byte_size[4] = {0};
46    bm_image_get_byte_size(src, image_byte_size);
47    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_size[3];
48    unsigned char *input_data = (unsigned char*)malloc(byte_size);
49    FILE *fp_src = fopen(src_name, "rb");
50    if (fread((void*)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
51        printf("file size is less than required bytes%d\n", byte_size);
52    }
53    fclose(fp_src);
54    void* in_ptr[4] = {((void *)input_data,
55                      ((void *)((unsigned char*)input_data + image_byte_size[0])),
56                      ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_size[1])),
57                      ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_size[1] + image_byte_size[2]))};
58    bm_image_copy_host_to_device(src, in_ptr);
59
60    bmcv_dwa_affine(handle, src, dst, affine_attr);
61
62    bm_image_get_byte_size(dst, image_byte_size);

```

(续下页)

(接上页)

```

62     byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
63     →size[3];
64     unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
65     void* out_ptr[4] = {(void*)output_ptr,
66                          ((void*)((unsigned char*)output_ptr + image_byte_size[0])),
67                          ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
68     →size[1])),
69                          ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
70     →size[1] + image_byte_size[2]))};
71     bm_image_copy_device_to_host(dst, (void**)out_ptr);
72
73     FILE *fp_dst = fopen(dst_name, "wb");
74     if (fwrite((void*)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
75         printf("file size is less than %d required bytes\n", byte_size);
76     };
77     fclose(fp_dst);
78
79     bm_image_destroy(&zsrc);
80     bm_image_destroy(&dst);
81
82     free(input_data);
83     free(output_ptr);
84
85     return 0;
}

```

### 5.33 bmcv\_dwa\_fisheye

#### 【描述】

去畸变仿射 (DWA) 模块的鱼眼畸变校正功能，通过配置校正参数获取适当的校正模型来消除鱼眼镜头造成的图像畸变，从而使弯曲的图像呈现出人眼能够感受到的更真实的形式。

其中，提供两种校正的方式供用户选择，分别为：1. 用户根据鱼眼畸变的类型及校正模型输入配置参数列表对图像进行校正；2. 用户使用 Grid\_Info(输入输出图像坐标映射关系描述)文件校正图像，以获得更好的图像校正效果。

#### 【语法】

```

1  bm_status_t bmcv_dwa_fisheye(bm_handle_t handle,
2                                bm_image input_image,
3                                bm_image output_image,
4                                bmcv_fisheye_attr_s fisheye_attr);

```

## 【参数】

表 5.16: bmcv\_dwa\_fisheye 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
input_image	输入	输入的畸变图像，通过调用 bm_image_create 创建
output_image	输出	输出的畸变校正后的图像，通过调用 bm_image_create 创建
fisheye_attr	输入	Fisheye 功能的参数配置列表，包括 bEnable bBgColor u32BgColor s32HorOffset s32VerOffset u32TrapezoidCoef s32FanStrength enMountMode enUseMode u32RegionNum enViewMode *grid_info

## 【返回值】

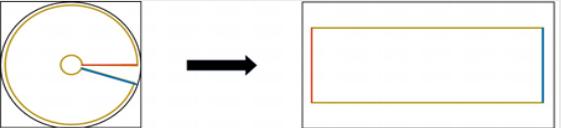
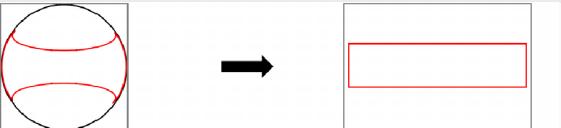
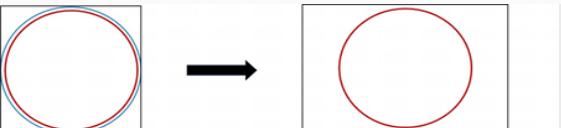
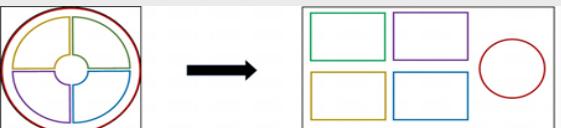
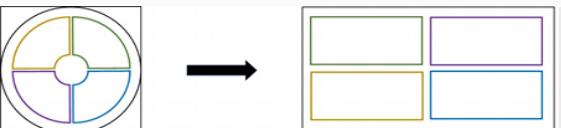
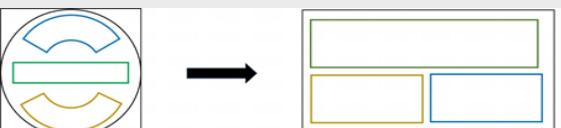
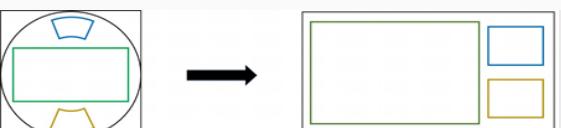
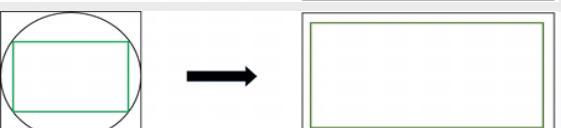
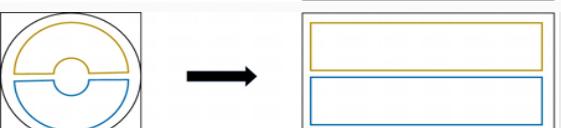
该函数成功调用时，返回 BM\_SUCCESS。

## 【数据类型说明】

表 5.17: bmcv\_fisheye\_attr\_s 参数表

参数名称	输入/输出	描述
bEnable	输入	配置范围:bool, 是否开启鱼眼校正功能
bBgColor	输入	配置范围:bool, 是否使用背景颜色功能
u32BgColor	输入	配置范围:[0, 0xffffffff], 背景颜色的 RGB888 值, bBgColor=1 时生效
s32HorOffset	输入	图像中心点相对于物理中心点的水平偏移
s32VerOffset	输入	图像中心点相对于物理中心点的垂直偏移
u32TrapezoidCoef	输入	配置范围:[0, 32], 梯形校正强度系数, 默认输入配置为 0
s32FanStrength	输入	配置范围: [-760, 760], 扇形校正强度系数, 默认输入配置为 0
enMountMode	输入	配置范围:[0, 2], 鱼眼校正的安装模式, 包括地装、顶装、和壁装三种安装模式
enUseMode	输入	配置范围:[1, 9], 鱼眼校正的应用模式
u32RegionNum	输入	配置范围:[1, 4], 鱼眼校正的区域数量
enViewMode	输入	配置范围:[0, 3], 客户端的鱼眼视图模式, 即不同的鱼眼摄像头观看模式
grid_info	输入	用于存储 grid_info 的信息, 包含 grid_info 的大小和数据

表 5.18: 鱼眼校正的应用模式

应用模式	效果示例
BMCV_MODE_PANORAMA_360	
BMCV_MODE_PANORAMA_180	
BMCV_MODE_01_1O	
BMCV_MODE_02_1O4R	
BMCV_MODE_03_4R	
BMCV_MODE_04_1P2R	
BMCV_MODE_05_1P2R	
BMCV_MODE_06_1P	
BMCV_MODE_07_2P	

注: BMCV\_MODE\_STEREO\_FIT 暂不支持。

表 5.19: Fisheye Correction 客户端的鱼眼视图模式 enView-Mode 参数表

参数名称	输入/输出	描述
BMCV_FISHEYE_	输入	代表 360 度全景模式的鱼眼视图
BMCV_FISHEYE_	输入	代表 180 度全景模式的鱼眼视图
BMCV_FISHEYE_	输入	代表普通视图模式
BMCV_FISHEYE_	输入	代表没有任何变换的鱼眼视图模式
BMCV_FISHEYE_	输入	表示枚举的结束标记

### 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_YUV420P
3	FORMAT_YUV444P
4	FORMAT_GRAY

### 【注意】

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。
2. 支持图像的分辨率为 32x32~4096x4096, 且要求 32 对齐。
3. 若用户决定使用第一种方式进行图像校正, 用户需根据鱼眼畸变的类型及校正模型自行输入配置参数列表 fisheye\_attr, 此时要将 grid\_info 设置为空。
4. 若用户决定使用第二种方式进行图像校正, 需提供 Grid\_Info 文件, 具体使用方式请参考下面的代码示例。

### 【代码示例】

1. 通过配置参数列表进行图像校正

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmcv_api_ext_c.h"
5 #include <unistd.h>
6
7 #define YUV_8BIT(y, u, v) (((y)&0xff) << 16) | (((u)&0xff) << 8) | ((v)&0xff))
8
9 int main() {
10     int src_h = 1024, src_w = 1024, dst_w = 1280, dst_h = 720, dev_id = 0;
11     int yuv_8bit_y = 0, yuv_8bit_u = 0, yuv_8bit_v = 0;
12     bm_image_format_ext fmt = FORMAT_YUV420P;
13     char *src_name = "path/to/src", *dst_name = "path/to/dst";
14     bm_handle_t handle = NULL;
15     bmcv_fisheye_attr_s fisheye_attr = {0};
16     dst_w = 1280;
17     dst_h = 720;
18     fmt = FORMAT_RGB_PLANAR;
19     fisheye_attr.bEnable = 1;
20     fisheye_attr.bBgColor = 1;
21     yuv_8bit_y = 0;
22     yuv_8bit_u = 128;
23     yuv_8bit_v = 128;
24     fisheye_attr.u32BgColor = YUV_8BIT(yuv_8bit_y, yuv_8bit_u, yuv_8bit_v);
25     fisheye_attr.s32HorOffset = src_w / 2;
26     fisheye_attr.s32VerOffset = src_h / 2;
27     fisheye_attr.u32TrapezoidCoef = 0;
28     fisheye_attr.s32FanStrength = 0;
29     fisheye_attr.enMountMode = 0;
30     fisheye_attr.enUseMode = 1;
31     fisheye_attr.enViewMode = 0;
32     fisheye_attr.u32RegionNum = 1;
33     fisheye_attr.grid_info.u.system.system_addr = NULL;
34     fisheye_attr.grid_info.size = 0;
35     dst_name = "dwa_fisheye_output_rand.yuv";
36     // rand_mode = 1;
37     int ret = (int)bm_dev_request(&handle, dev_id);
38     if (ret != 0) {
39         printf("Create bm handle failed. ret = %d\n", ret);
40         exit(-1);
41     }
42     bm_image src, dst;
43
44     bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
45     bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
46
47     ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
48     ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
49
50     int image_byte_size[4] = {0};
51     bm_image_get_byte_size(src, image_byte_size);
```

(续下页)

(接上页)

```

52 int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
→byte_size[3];
53 unsigned char *input_data = (unsigned char *)malloc(byte_size);
54 FILE *fp_src = fopen(src_name, "rb");
55 if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
56     printf("file size is less than required bytes%od\n", byte_size);
57 }
58 fclose(fp_src);
59 void* in_ptr[4] = { (void *)input_data,
60                     ((unsigned char*)input_data + image_byte_size[0]),
61                     ((unsigned char*)input_data + image_byte_size[0] + image_byte_
→size[1]),
62                     ((unsigned char*)input_data + image_byte_size[0] + image_byte_
→size[1] + image_byte_size[2])};
63 bm_image_copy_host_to_device(src, in_ptr);
64
65 bmcv_dwa_fisheye(handle, src, dst, fisheye_attr);
66
67 bm_image_get_byte_size(src, image_byte_size);
68 byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
→size[3];
69 unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
70 void* out_ptr[4] = { (void *)output_ptr,
71                     ((unsigned char*)output_ptr + image_byte_size[0]),
72                     ((unsigned char*)output_ptr + image_byte_size[0] + image_byte_size[1]),
73                     ((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
→size[1] + image_byte_size[2])};
74 bm_image_copy_device_to_host(src, (void **)out_ptr);
75
76 FILE *fp_dst = fopen(dst_name, "wb");
77 if (fwrite((void *)input_data, 1, byte_size, fp_dst) < (unsigned int)byte_size){
78     printf("file size is less than %d required bytes\n", byte_size);
79 }
80 fclose(fp_dst);
81
82 free(input_data);
83 free(output_ptr);
84 bm_image_destroy(&src);
85 bm_image_destroy(&dst);
86
87 bm_dev_free(handle);
88
89 return 0;
90 }
```

## 2. 通过 Grid\_Info 文件进行图像校正

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 int main() {
8     bm_status_t ret = BM_SUCCESS;
9     bm_handle_t handle = NULL;
10    int dev_id = 0;
11
12    char *src_name = "path/to/src", *dst_name = "path/to/dst", *grid_name = "path/to/grid_"
13    ↪info_dat";
14
15    int src_h = 2240, src_w = 2240;
16    int dst_w = 2240, dst_h = 2240;
17    bm_image src, dst;
18    bm_image_format_ext fmt = FORMAT_YUV420P;
19    ret = (int)bm_dev_request(&handle, dev_id);
20    bmvc_fisheye_attr_s fisheye_attr = {0};
21    fisheye_attr.grid_info.size = 446496; // 注意：用户需根据实际的Grid_
22    ↪Info文件大小（字节数）进行设置
23    fisheye_attr.bEnable = true;
24
25    // create bm image
26    bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
27    bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
28    ret = bm_image_alloc_dev_mem(src, BMVC_HEAP_ANY);
29    ret = bm_image_alloc_dev_mem(dst, BMVC_HEAP_ANY);
30    // read image data from input files
31    int image_byte_size[4] = {0};
32    bm_image_get_byte_size(src, image_byte_size);
33    int byte_size = src_w * src_h * 3 / 2;
34    unsigned char *input_data = (unsigned char *)malloc(byte_size);
35    FILE *fp_src = fopen(src_name, "rb");
36    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
37        printf("file size is less than required bytes%d\n", byte_size);
38    };
39    fclose(fp_src);
40    void* in_ptr[3] = {(void *)input_data, (void *)((unsigned char*)input_data + src_w * src_h),
41    ↪(void *)((unsigned char*)input_data + 5 / 4 * src_w * src_h)};
42    bm_image_copy_host_to_device(src, in_ptr);
43    // read grid_info data
44    char *buffer = (char *)malloc(fisheye_attr.grid_info.size);
45    memset(buffer, 0, fisheye_attr.grid_info.size);
46
47    FILE *fp = fopen(grid_name, "rb");
48    fread(buffer, 1, fisheye_attr.grid_info.size, fp);
49    fclose(fp);
50    fisheye_attr.grid_info.u.system.system_addr = (void *)buffer;

```

(续下页)

(接上页)

```

49     bmcv_dwa_fisheye(handle, src, dst, fisheye_attr);
50     bm_image_get_byte_size(dst, image_byte_size);
51     unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
52     void* out_ptr[3] = {(void*)output_ptr, (void*)((unsigned char*)output_ptr + dst_w * dst_h),
53     ↪ (void*)((unsigned char*)output_ptr + 5 / 4 * dst_w * dst_h)};
54     bm_image_copy_device_to_host(dst, (void**)out_ptr);
55
56     FILE *fp_dst = fopen(dst_name, "wb");
57     if (fwrite((void*)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
58         printf("file size is less than %d required bytes\n", byte_size);
59     };
60     fclose(fp_dst);
61     return ret;
}

```

### 5.34 bmcv\_dwa\_dewarp

#### 【描述】

去畸变校正 (DWA) 模块的去畸变校正功能，通过 Grid\_Info(输入输出图像坐标映射关系描述) 文件对畸变图像进行校正，使图像中的直线变得更加准确和几何正确，提高图像的质量和可视化效果。

#### 【语法】

```

1  bm_status_t bmcv_dwa_dewarp(bm_handle_t handle,
2                                bm_image input_image,
3                                bm_image output_image,
4                                char *grid_info);

```

#### 【参数】

表 5.20: bmcv\_dwa\_dewarp 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
input_image	输入	输入的畸变图像，通过调用 bm_image_create 创建
output_image	输出	输出的畸变校正后的图像，通过调用 bm_image_create 创建
*grid_info	输入	输入的 Grid_Info 的对象指针

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【格式支持】

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_YUV420P
3	FORMAT_YUV444P
4	FORMAT_GRAY

## 【注意】

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。
2. 支持图像的分辨率为 32x32~4096x4096, 且要求 32 对齐。
3. 用户需提供 Grid\_Info 文件以进行 DEWARP 功能, 具体使用方式请参考下面的代码示例。

## 【代码示例】

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 typedef unsigned int u32;
8
9 int main() {
10     int src_h = 1080, src_w = 1920, dst_w = 1920, dst_h = 1080;
```

(续下页)

(接上页)

```

11 int dev_id = 0;
12 bm_image_format_ext fmt = FORMAT_GRAY;
13 char *src_name = "path/to/src";
14 char *dst_name = "path/to/dst";
15 char *grid_name = "path/to/grid_info";
16 bm_handle_t handle = NULL;
17 u32 grid_size = 0;
18 int ret = (int)bm_dev_request(&handle, dev_id);
19 if (ret != 0) {
20     printf("Create bm handle failed. ret = %d\n", ret);
21     exit(-1);
22 }
23 FILE *fp = fopen(grid_name, "rb");
24 if (!fp) {
25     printf("open file:%s failed.\n", grid_name);
26     exit(-1);
27 }
28 char *grid_data = (char *)malloc(grid_size);
29 fread(grid_data, 1, grid_size, fp);
30
31 fclose(fp);
32
33 bm_image src, dst;
34
35 dst_w = src_w;
36 dst_h = src_h;
37 bm_image_create(handle, src_h, src_w, fmt, DATA_TYPE_EXT_1N_BYTE, &src, NULL);
38 bm_image_create(handle, dst_h, dst_w, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, NULL);
39
40 ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
41 ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
42
43 int image_byte_size[4] = {0};
44 bm_image_get_byte_size(src, image_byte_size);
45 int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
46 →byte_size[3];
47 unsigned char *input_data = (unsigned char *)malloc(byte_size);
48 FILE *fp_src = fopen(src_name, "rb");
49 if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
50     printf("file size is less than required bytes%d\n", byte_size);
51 }
52 fclose(fp_src);
53 void* in_ptr[4] = {(void *)input_data,
54                     ((void *)((unsigned char*)input_data + image_byte_size[0])),
55                     ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
56 →size[1])),
57                     ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
58 →size[1] + image_byte_size[2]))};
59 bm_image_copy_host_to_device(src, in_ptr);
60
61 bm_device_mem_t dmem;

```

(续下页)

(接上页)

```
59 dmem.u.system.system_addr = (void *)grid_data;
60 dmem.size = grid_size;
61
62 bmcv_dwa_dewarp(handle, src, dst, dmem);
63
64 bm_image_get_byte_size(dst, image_byte_size);
65 byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_
→size[3];
66 unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
67 void* out_ptr[4] = {(void*)output_ptr,
68                     ((void*)((unsigned char*)output_ptr + image_byte_size[0])),
69                     ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
→size[1])),
70                     ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_byte_
→size[1] + image_byte_size[2]))};
71 bm_image_copy_device_to_host(dst, (void **)out_ptr);
72
73 FILE *fp_dst = fopen(dst_name, "wb");
74 if (fwrite((void *)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
75     printf("file size is less than %d required bytes\n", byte_size);
76 }
77 fclose(fp_dst);
78
79 free(grid_data);
80 free(input_data);
81 free(output_ptr);
82 bm_image_destroy(&src);
83 bm_image_destroy(&dst);
84
85 bm_dev_free(handle);
86
87 return 0;
88 }
```

### 5.35 bmcv\_blend

该接口可调用芯片上的 stitch 硬件模块，实现 2 ~ 4 路图像拼接功能，下图展示了 2 路图像拼接的示例。



图 a: 2 路图像拼接示例

算法流程示例:

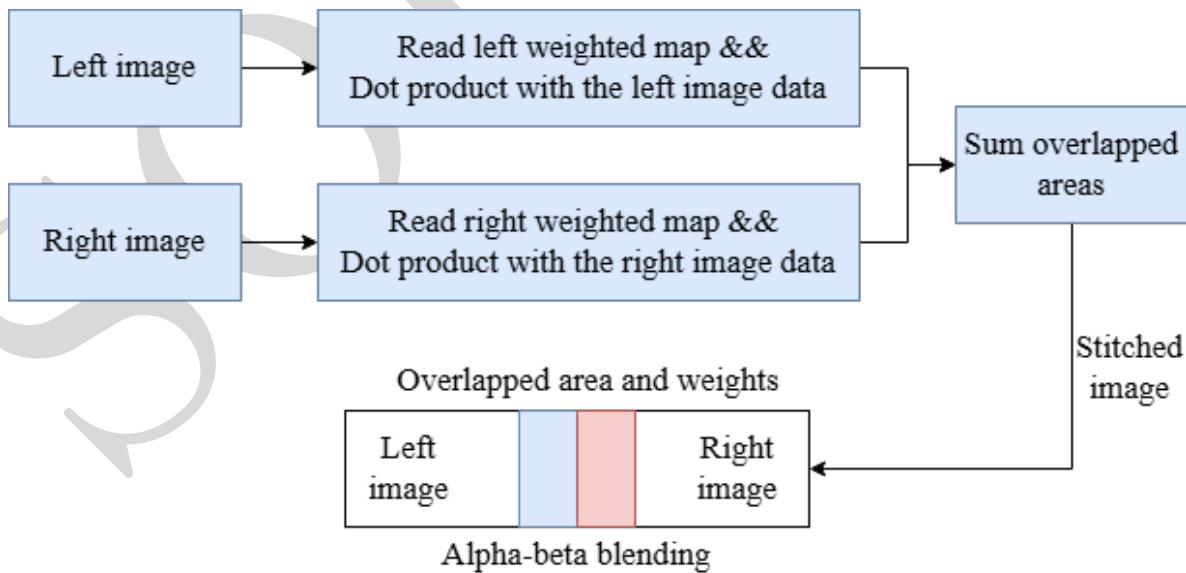


图 b: Image Stitching 算法流程示例

图 b 展示了基于 Image Stitching 模块的 2 路图片拼接的原理。在进行拼接之前，用户需要指定重叠区域及左右权重图 (weighted map)；接着，将左右权重图分别与输入的左右图像进

行相乘；最后，将结果相加以获得最终的图片拼接结果。

当多路图像输入 Image Blend 模块前，或需要通过其他模块对多路图片进行影像几何畸变校正 (DWA), 抠图 (Crop), 图像缩放 (resize) 等操作。

Image Blend 模块接受双路图片作为输入，通过 Image Blend 算法根据相同的图像区间，将这些图片拼接在一起。

#### 加速原理介绍：

图片融合模块 (Image Blend) 用于加速影像拼接的相关计算。对于有着不同图片输入数量的图片拼接任务，拼接方式有多种方案。下图别展示了常用的 2 路与 4 路图片拼接方案。

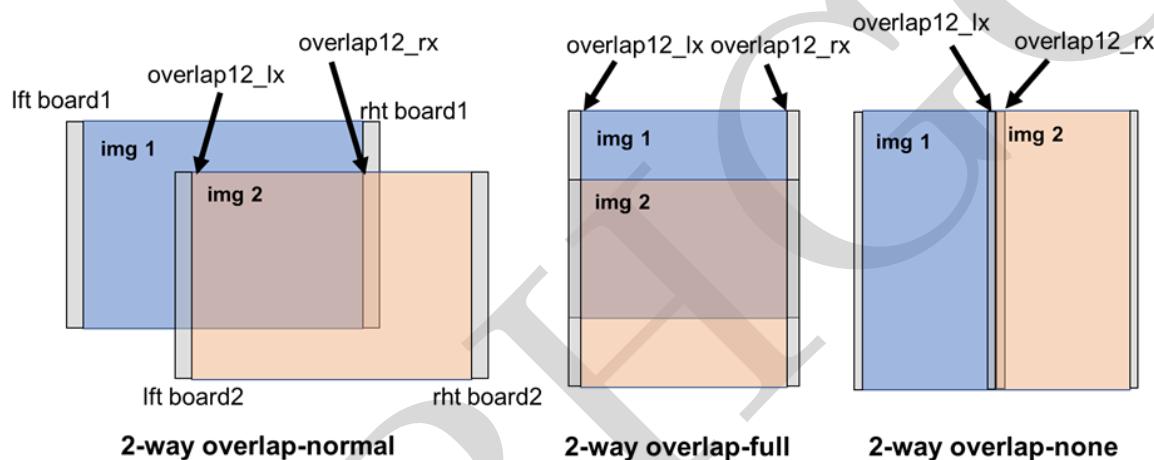


图 c: Image Blend 模块双路图片拼接方案示例

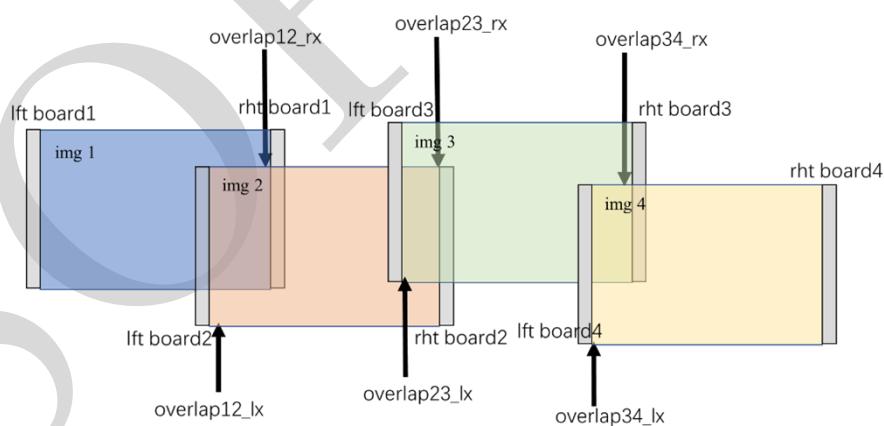


图 d: Image Blend 模块四路图片拼接方案示例

#### 接口函数介绍：

##### 【接口形式】

```

1 bm_status_t bmcv_blending(
2     bm_handle_t handle,
3     int input_num,
4     bm_image* input,
5     bm_image output,
6     struct stitch_param stitch_config);

```

## 【参数】

表 5.21: bmcv\_blending 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input_num	输入	输入 image 数量，2 或 3 或 4。
*input	输入	输入待拼接 bm_image 对象指针，其指向空间的长度由 input_num 决定。
output	输出	输出拼接后 bm_image 对象。
stitch_config	输入	该结构描述了拼接融合中所需要设置的参数。

## 【数据类型说明】

```

1 struct stitch_param {
2     struct bm_stitch_src_ovlp_attr ovlp_attr;
3     struct bm_stitch_src_bd_attr bd_attr;
4     bm_device_mem_t wgt_phy_mem[3][2];
5     enum bm_stitch_wgt_mode wgt_mode;
6 };

```

表 5.22: stitch\_param 参数介绍

参数名称	描述
ovlp_attr	输入源重叠区域属性。
bd_attr	输入域左右边界区域属性，目前参数未开放，设为 0。
wgt_phy_mem	权重图的物理地址和大小。
wgt_mode	权重图的模式。

```

1 struct bm_stitch_src_ovlp_attr {
2     short ovlp_lx[3];
3     short ovlp_rx[3];
4 };

```

表 5.23: bm\_stitch\_src\_ovlp\_attr 参数介绍

参数名称	描述
ovlp_lx	重叠区域左边界点 x 坐标。
ovlp_rx	重叠区域右边界点 x 坐标。

```

1 struct bm_stitch_src_bd_attr {
2     short bd_lx[4];
3     short bd_rx[4];
4 };

```

表 5.24: bm\_stitch\_src\_bd\_attr 参数介绍

参数名称	描述
bd_lx	左侧图片的黑边宽度。
bd_rx	右侧图片的黑边宽度。

```

1 typedef struct bm_mem_desc {
2     union {
3         struct {
4             #ifdef __linux__
5                 unsigned long device_addr;
6             #else
7                 unsigned long long device_addr;
8             #endif
9                 unsigned int reserved;
10                int dmabuf_fd;
11            } device;
12
13            struct {
14                void *system_addr;
15                unsigned int reserved0;
16                int reserved1;
17            } system;
18        } u;
19
20        bm_mem_flags_t flags;
21        unsigned int size;
22    } bm_mem_desc_t;

```

表 5.25: bm\_device\_mem\_t wgt\_phy\_mem 参数介绍

参数名称	描述
wgt_phy_mem[0][0]	第 1 处融合区域, y 通道权重图数据。
wgt_phy_mem[0][1]	第 1 处融合区域, uv 通道权重图数据。
device_addr	权重图的物理地址。
size	权重图的字节数大小。

```

1 enum bm_stitch_wgt_mode {
2     BM_STITCH_WGT_YUV_SHARE = 0,
3     BM_STITCH_WGT_UV_SHARE,
4     BM_STITCH_WGT_SEP,
5 };

```

表 5.26: bm\_stitch\_wgt\_mode 参数介绍

参数名称	描述
BM_STITCH_WGT_YUV_SHARE	YUV share alpha and beta ( $1 \alpha + 1 \beta$ ), 优先推荐这个。
BM_STITCH_WGT_UV_SHARE	UV share alpha and beta ( $2 \alpha + 2 \beta$ ), 不推荐选这个。

**【返回值】**

- BM\_SUCCESS: 成功
- 其他: 失败

**【格式支持】**

- 输入、输出图像格式支持:
  - FORMAT\_RGBP\_SEPARATE、FORMAT\_BGRP\_SEPARATE
  - FORMAT\_YUV420P、FORMAT\_YUV422P
  - FORMAT\_YUV444P、FORMAT\_GRAY

**注意事项:**

- 1、图像 stride 要求 16byte 对齐。输入图片最小像素支持 64x64，最大支持 4608x8192。
- 2、输入输出数据类型要求: DATA\_TYPE\_EXT\_1N\_BYTE。
- 3、输入输出格式需保持一致；输入输出图片高需要相等。
- 4、wgt\_phy\_mem 参数，请用 bmlib 相关 api 设置，不要直接赋值。使用例如: bm\_malloc\_device\_byte, bm\_memcpy\_s2d 等接口。传入的权重数据须为 char 类型数据，对于重叠区域的每个像素，左权重 + 右权重数值须为 255。

**示例代码:**

```

#include "stdio.h"
#include "stdlib.h"
#include <unistd.h>
#include "string.h"
#include "getopt.h"
#include "signal.h"
#include "bmcv_api_ext_c.h"

```

(续下页)

(接上页)

```
#include <stdatomic.h>

#define ALIGN(x, a)    (((x) + ((a)-1)) & ~((a)-1))
void bm_dem_read_bin(bm_handle_t handle, bm_device_mem_t* dmem, const char *input_
→name, unsigned int size)
{
    if (access(input_name, F_OK) != 0 || strlen(input_name) == 0 || 0 >= size)
    {
        return;
    }

    char* input_ptr = (char *)malloc(size);
    FILE *fp_src = fopen(input_name, "rb+");
    if (fread((void *)input_ptr, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };
    fclose(fp_src);

    if (BM_SUCCESS != bm_malloc_device_byte(handle, dmem, size)){
        printf("bm_malloc_device_byte failed\n");
    }

    if (BM_SUCCESS != bm_memcpy_s2d(handle, *dmem, input_ptr)){
        printf("bm_memcpy_s2d failed\n");
    }

    free(input_ptr);
    return;
}

void blend_HandleSig(int signum)
{
    signal(SIGINT, SIG_IGN);
    signal(SIGTERM, SIG_IGN);

    printf("signal happen %d \n", signum);

    exit(-1);
}

int main()
{
    bm_handle_t handle = NULL;
    bm_image_src[2], dst;
    int src_h[2] = {288, 288}, src_w[2] = {2304, 4608}, dst_w = 4608, dst_h = 288, wgtWidth, F
    →wgtHeight;
    bm_image_format_ext src_fmt = FORMAT_YUV420P, dst_fmt = FORMAT_YUV420P;
    char *src_name[2] = {"path/to/src1", "path/to/src2"}, *dst_name = "path/to/dst", *wgt_
    →name[2] = {"path/to/wgt1", "path/to/wgt2"};
}
```

(续下页)

(接上页)

```

int dev_id = 0, ret = 0, wgt_len = 0, input_num = 2;

struct stitch_param stitch_config;
memset(&stitch_config, 0, sizeof(stitch_config));
stitch_config.wgt_mode = BM_STITCH_WGT_YUV_SHARE;

signal(SIGINT, blend_HandleSig);
signal(SIGTERM, blend_HandleSig);

bm_status_t ret1 = bm_dev_request(&handle, dev_id);
if (ret1 != BM_SUCCESS) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
stitch_config.ovlp_attr.ovlp_rx[0] = 2303;
stitch_config.ovlp_attr.ovlp_lx[0] = 0;
wgtWidth = ALIGN(stitch_config.ovlp_attr.ovlp_rx[0] - stitch_config.ovlp_attr.ovlp_lx[0][F
→+ 1, 16]);
wgtHeight = src_h[0];

int byte_size = 0;
for(int i = 0; i < 2; i++) {
    bm_image_create(handle, src_h[i], src_w[i], src_fmt, DATA_TYPE_EXT_1N_BYTE, &
→src[i], NULL);
    bm_image_alloc_dev_mem(src[i], 1);
    byte_size = src_w[i] * src_h[i] * 3 / 2;
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes% d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = { (void *)input_data, (void *)((unsigned char *)input_data + src_w[i] * src_
→h[i]), (void *)((unsigned char *)input_data + src_w[i] * src_h[i] * 5 / 4)};
    bm_image_copy_host_to_device(src[i], in_ptr);
    wgt_len = wgtWidth * wgtHeight;
    if (stitch_config.wgt_mode == BM_STITCH_WGT_UV_SHARE)
        wgt_len = wgt_len << 1;

    bm_dem_read_bin(handle, &stitch_config.wgt_phy_mem[0][i], wgt_name[i], wgt_len);
}
bm_image_create(handle, dst_h, dst_w, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst,
→NULL);
bm_image_alloc_dev_mem(dst, 1);

ret = bmcv_blending(handle, input_num, src, dst, stitch_config);

unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
void* out_ptr[4] = { (void *)output_ptr, (void *)((unsigned char *)output_ptr + dst_w * dst_h),
→ (void *)((unsigned char *)output_ptr + dst_w * dst_h * 5 / 4)};

```

(续下页)

(接上页)

```

bm_image_copy_device_to_host(dst, (void **)out_ptr);

FILE *fp_dst = fopen(dst_name, "wb");
if (fwrite((void *)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
    printf("file size is less than %d required bytes\n", byte_size);
}
fclose(fp_dst);

bm_image_destroy(&src[0]);
bm_image_destroy(&src[1]);
bm_image_destroy(&dst);
bm_dev_free(handle);

return ret;
}

```

### 5.36 bmcv\_image\_axpy

该接口实现  $F = A * X + Y$ , 其中 A 是常数, 大小为  $n * c$ , F、X、Y 都是大小为  $n * c * h * w$  的矩阵。

**接口形式:**

```

bm_status_t bmcv_image_axpy(
    bm_handle_t handle,
    bm_device_mem_t tensor_A,
    bm_device_mem_t tensor_X,
    bm_device_mem_t tensor_Y,
    bm_device_mem_t tensor_F,
    int input_n,
    int input_c,
    int input_h,
    int input_w);

```

**参数说明:**

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_device_mem_t tensor_A`  
输入参数。存放常数 A 的设备内存地址。
- `bm_device_mem_t tensor_X`  
输入参数。存放矩阵 X 的设备内存地址。
- `bm_device_mem_t tensor_Y`  
输入参数。存放矩阵 Y 的设备内存地址。

- `bm_device_mem_t tensor_F`  
输出参数。存放结果矩阵 F 的设备内存地址。
- `int input_n`  
输入参数。n 维度大小。
- `int input_c`  
输入参数。c 维度大小。
- `int input_h`  
输入参数。h 维度大小。
- `int input_w`  
输入参数。w 维度大小。

#### 返回值说明：

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 示例代码

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

#define N (10)
#define C 256 // (64 * 2 + (64 >> 1))
#define H 8
#define W 8
#define TENSOR_SIZE (N * C * H * W)

int main(){
    int trials = 1;
    int ret = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }

    float *tensor_X = malloc(TENSOR_SIZE * sizeof(float));
    float *tensor_A = malloc(N * C * sizeof(float));
    float *tensor_Y = malloc(TENSOR_SIZE * sizeof(float));
    float *tensor_F = malloc(TENSOR_SIZE * sizeof(float));
    int idx_trial;
```

(续下页)

(接上页)

```

for (idx_trial = 0; idx_trial < trials; idx_trial++) {
    for (int idx = 0; idx < TENSOR_SIZE; idx++) {
        tensor_X[idx] = (float)idx - 5.0f;
        tensor_Y[idx] = (float)idx/3.0f - 8.2f; //y
    }

    for (int idx = 0; idx < N*C; idx++) {
        tensor_A[idx] = (float)idx * 1.5f + 1.0f;
    }

    ret = bmcv_image_axpy(handle,
        bm_mem_from_system((void *)tensor_A),
        bm_mem_from_system((void *)tensor_X),
        bm_mem_from_system((void *)tensor_Y),
        bm_mem_from_system((void *)tensor_F),
        N, C, H, W);
}

free(tensor_X);
free(tensor_A);
free(tensor_Y);
free(tensor_F);

bm_dev_free(handle);
return ret;
}

```

### 5.37 bmcv\_image\_resize

该接口用于实现图像尺寸的变化，如放大、缩小、抠图等功能。

**接口形式：**

```

bm_status_t bmcv_image_resize(
    bm_handle_t handle,
    int input_num,
    bmcv_resize_image_resize_attr[4],
    bm_image* input,
    bm_image* output
);

```

**参数说明：**

- bm\_handle\_t handle

输入参数。bm\_handle 句柄。

- int input\_num

输入参数。输入图片数，最多支持 4。

- bmcv\_resize\_image\_resize\_attr [4]

输入参数。每张图片对应的 resize 参数, 最多支持 4 张图片。

- `bm_image*` input

输入参数。输入 `bm_image`。每个 `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `bm_image*` output

输出参数。输出 `bm_image`。每个 `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存, 如果不主动分配将在 api 内部进行自行分配。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 数据类型说明:

```
typedef struct bmcv_resize_s{
    int start_x;
    int start_y;
    int in_width;
    int in_height;
    int out_width;
    int out_height;
}bmcv_resize_t;

typedef struct bmcv_resize_image_s{
    bmcv_resize_t *resize_img_attr;
    int roi_num;
    unsigned char stretch_fit;
    unsigned char padding_b;
    unsigned char padding_g;
    unsigned char padding_r;
    unsigned int interpolation;
}bmcv_resize_image;
```

- `bmcv_resize_image` 描述了一张图中 resize 配置信息。
- `roi_num` 描述了一副图中需要进行 resize 的子图总个数。
- `stretch_fit` 表示是否按照原图比例对图片进行缩放, 1 表示无需按照原图比例进行缩放, 0 表示按照原图比例进行缩放, 当采用这种方式的时候, 结果图片中为进行缩放的地方将会被填充成特定值。
- `padding_b` 表示当 `stretch_fit` 设成 0 的情况下, `b` 通道上被填充的值。
- `padding_r` 表示当 `stretch_fit` 设成 0 的情况下, `r` 通道上被填充的值。
- `padding_g` 表示当 `stretch_fit` 设成 0 的情况下, `g` 通道上被填充的值。

- interpolation 表示缩图所使用的算法。BMCV\_INTER\_NEAREST 表示最近邻算法，BMCV\_INTER\_LINEAR 表示线性插值算法。
- start\_x 描述了 resize 起始横坐标（相对于原图），常用于抠图功能。
- start\_y 描述了 resize 起始纵坐标（相对于原图），常用于抠图功能。
- in\_width 描述了 crop 图像的宽。
- in\_height 描述了 crop 图像的高。
- out\_width 描述了输出图像的宽。
- out\_height 描述了输出图像的高。

#### 注意事项:

该接口的注意事项与 bmcv\_image\_vpp\_basic 接口相同。

### 5.38 bmcv\_image\_watermark\_superpose

#### 【描述】

以下接口用于在图像上叠加一个或多个水印。

#### 【描述接口一】

接口一可实现在不同的输入图的指定位置，叠加不同的水印。

#### 【语法】

```
1 bm_status_t bmcv_image_watermark_superpose(
2     bm_handle_t handle,
3     bm_image *image,
4     bm_device_mem_t *bitmap_mem,
5     int bitmap_num,
6     int bitmap_type,
7     int pitch,
8     bmcv_rect_t *rects,
9     bmcv_color_t color);
```

#### 【描述接口二】

此接口为接口一的简化版本，可在一张图中的不同位置重复叠加一种水印。

## 【语法】

```
1 bm_status_t bmcv_image_watermark_repeat_superpose(
2     bm_handle_t handle,
3     bm_image image,
4     bm_device_mem_t bitmap_mem,
5     int bitmap_num,
6     int bitmap_type,
7     int pitch,
8     bmcv_rect_t * rects,
9     bmcv_color_t color);
```

### 传入参数说明:

- bm\_handle\_t handle  
输入参数。设备环境句柄，通过调用 bm\_dev\_request 获取。
- bm\_image\* image  
输入参数。需要打水印的 bm\_image 对象指针。
- bm\_device\_mem\_t\* bitmap\_mem  
输入参数。水印的 bm\_device\_mem\_t 对象指针。
- int bitmap\_num  
输入参数。水印数量，指 rects 指针中所包含的 bmcv\_rect\_t 对象个数、也是 image 指针中所包含的 bm\_image 对象个数、也是 bitmap\_mem 指针中所包含的 bm\_device\_mem\_t 对象个数。
- int bitmap\_type  
输入参数。水印类型，值 0 表示水印为 8bit 数据类型（有透明度信息），值 1 表示水印为 1bit 数据类型（无透明度信息）。
- int pitch  
输入参数。水印文件每行的 byte 数，可理解为水印的宽。
- bmcv\_rect\_t\* rects  
输入参数。水印位置指针，包含每个水印起始点和宽高。具体内容参考下面的数据类型说明。
- bmcv\_color\_t color  
输入参数。水印的颜色。具体内容参考下面的数据类型说明。

### 返回值说明:

- BM\_SUCCESS: 成功

- 其他: 失败

#### 数据类型说明:

```
typedef struct bmcv_rect {  
    int start_x;  
    int start_y;  
    int crop_w;  
    int crop_h;  
} bmcv_rect_t;  
  
typedef struct {  
    unsigned char r;  
    unsigned char g;  
    unsigned char b;  
} bmcv_color_t;
```

- start\_x 描述了水印在原图中所在的起始横坐标。自左而右从 0 开始，取值范围 [0, width)。
- start\_y 描述了水印在原图中所在的起始纵坐标。自上而下从 0 开始，取值范围 [0, height)。
- crop\_w 描述的水印的宽度。
- crop\_h 描述的水印的高度。
- r 颜色的 r 分量。
- g 颜色的 g 分量。
- b 颜色的 b 分量。

#### 注意事项:

1. 该 API 要求如下：
  - 底图的数据类型必须为：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

- 底图的色彩格式可支持：

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

如果不满足输入输出格式要求，则返回失败。

- 水印图两种格式的数据排列：

8bit 数据格式：

用每个 byte [0-255] 存储单个像素的透明度信息，0 代表完全透明，255 代表完全不透明。其数据量为 width \* height byte。

$$dst = (\alpha * color + (255 - \alpha) * src) / 255$$

dst 表示输出图该位置像素点的值，alpha 表示水印图存储的透明度信息，color 代表接口中 color 参数传入的值，src 代表底图该位置像素点的值。

如下代码，A\_8bit 是一个宽高为 8\*8 的 8bit 水印，内容为字母 A。

```
unsigned char A_8bit[8][8] = {
{0, 0, 255, 255, 0, 0, 0, 0},
{0, 255, 0, 0, 255, 0, 0, 0},
{0, 255, 0, 0, 255, 0, 0, 0},
{0, 255, 255, 255, 255, 0, 0, 0},
{0, 255, 0, 0, 255, 0, 0, 0},
{0, 255, 0, 0, 255, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0};
```

1bit 数据格式：

用每个 bit [0-1] 存储单个像素是否在有效区域，0 代表在无效区域，1 代表在有效区域。其数据量为 width \* height / 8 byte。

$$dst = (\alpha == 1 ? color : src)$$

dst 表示输出图该位置像素点的值，alpha 表示水印图存储的区域信息，color 代表接口中 color 参数传入的值，src 代表底图该位置像素点的值。

将上述 A\_8bit 矩阵用 1bit 存储，即如下代码。

```
unsigned char A_1bit[8] = {
    0xc, //二进制表示 0000 1100
    0x12,//(二进制表示 0001 0010)
    0x12,//(二进制表示 0001 0010)
    0x1e,//(二进制表示 0001 1110)
    0x12,//(二进制表示 0001 0010)
    0x12,//(二进制表示 0001 0010)
    0x0, //二进制表示 0000 0000
    0x0}//(二进制表示 0000 0000)
```

2. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
3. 水印数量最多可设置 512 个。
4. 如果水印区域超出原图宽高，会返回失败。
5. 不支持对单底图进行位置重叠的多图叠加。

#### 示例代码：

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "bmvc_api_ext_c.h"

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    len = fwrite((void*)output_data, 1, size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_dst);
    return 0;
}

bm_status_t open_water(
    bm_handle_t handle,
    char * src_name,
    int src_size,
    bm_device_mem_t * dst)
{
    bm_status_t ret = BM_SUCCESS;
```

(续下页)

(接上页)

```

unsigned char * src = (unsigned char *)malloc(sizeof(unsigned char) * src_size);
ret = bm_malloc_device_byte(handle, dst, src_size);
if(ret != BM_SUCCESS){
    printf("bm_malloc_device_byte fail %s: %s: %d\n", __FILE__, __func__, __LINE__);
    goto fail;
}

FILE * fp_src = fopen(src_name, "rb");
size_t read_size = fread((void *)src, src_size, 1, fp_src);
printf("fread %ld byte\n", read_size);
fclose(fp_src);

#ifdef _FPGA
    ret = bm_memcpy_s2d_fpga(handle, dst[0], (void*)src);
#else
    ret = bm_memcpy_s2d(handle, dst[0], (void*)src);
#endif
if(ret != BM_SUCCESS){
    printf("bm_memcpy_s2d fail %s: %s: %d\n", __FILE__, __func__, __LINE__);
}
fail:
    free(src);
    return ret;
}

int main() {
    char *filename_src = "path/to/src";
    char *filename_water = "path/to/water_file";
    char *filename_dst = "path/to/dst";
    int in_width = 1920;
    int in_height = 1080;
    int water_width = 800;
    int water_height = 600;
    bm_image_format_ext src_format = FORMAT_RGB_PLANAR;
    bmcv_rect_t rects = {
        .start_x = 200,
        .start_y = 200,
        .crop_w = 800,
        .crop_h = 600};
    bmcv_color_t color = {
        .r = 0,
        .g = 0,
        .b = 0};

    bm_status_t ret = BM_SUCCESS;

    int src_size = in_width * in_height * 3;
    int water_size = water_width * water_height * 3;
    unsigned char *src_data = (unsigned char *)malloc(src_size);

```

(续下页)

(接上页)

```
unsigned char *water_data = (unsigned char *)malloc(water_size);

FILE *file;
file = fopen(filename_water, "rb");
fread(water_data, sizeof(unsigned char), water_size, file);
fclose(file);

file = fopen(filename_src, "rb");
fread(src_data, sizeof(unsigned char), src_size, file);
fclose(file);

bm_handle_t handle = NULL;
int dev_id = 0;
bm_image src;
bm_device_mem_t water;
ret = bm_dev_request(&handle, dev_id);

open_water(handle, filename_water, water_size, &water);

bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_
↪IN_BYTE, &src, NULL);
bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);

int src_image_byte_size[4] = {0};
bm_image_get_byte_size(src, src_image_byte_size);

void *src_in_ptr[4] = {(void *)src_data,
                      (void *)((char *)src_data + src_image_byte_size[0]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_
↪byte_size[1]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_
↪byte_size[1] + src_image_byte_size[2])};

bm_image_copy_host_to_device(src, (void **)src_in_ptr);
ret = bmcv_image_watermark_superpose(handle, &src, &water, 1, 0, water_width,
↪&rects, color);
bm_image_copy_device_to_host(src, (void **)src_in_ptr);

writeBin(filename_dst, src_data, src_size);

bm_image_destroy(&src);
bm_dev_free(handle);

free(src_data);
free(water_data);

return ret;
}
```

### 5.39 bmcv\_image\_absdiff

两张大小相同的图片对应像素值相减并取绝对值。

**接口形式：**

```
bm_status_t bmcv_image_absdiff(
    bm_handle_t handle,
    bm_image input1,
    bm_image input2,
    bm_image output);
```

**参数说明：**

- `bm_handle_t handle`

输入参数。`bm_handle` 设备环境句柄，通过调用 `bm_dev_request` 获取。

- `bm_image input1`

输入参数。输入第一张图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `bm_image input2`

输入参数。输入第二张图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `bm_image output`

输出参数。输出 `bm_image`。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

**返回值说明：**

- `BM_SUCCESS`: 成功
- 其他: 失败

**格式支持：**

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

### 注意事项：

1. 在调用 bmcv\_image\_absdiff() 之前必须确保输入的 image 内存已经申请。
2. 输入、输出的 data\_type, image\_format 必须相同。

### 示例代码

```
#include "bmcv_api_ext_c.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include <assert.h>

void readBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_dst);
}
```

(续下页)

(接上页)

```
}
```

```
int main() {
    int height = 1080;
    int width = 1920;
    int format = 8;
    char *src1_name = "path/to/src1";
    char *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    bm_handle_t handle;
    bm_status_t ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("Create bm handle failed. ret = %d\n", ret);
        return -1;
    }
    int img_size = width * height * 3;
    unsigned char* input1_data = malloc(width * height * 3);
    unsigned char* input2_data = malloc(width * height * 3);
    unsigned char* output_tpu = malloc(width * height * 3);
    readBin(src1_name, input1_data, img_size);
    readBin(src2_name, input2_data, img_size);
    memset(output_tpu, 0, width * height * 3);
    bm_image input1_img;
    bm_image input2_img;
    bm_image output_img;

    bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_
    ↪TYPE_EXT_1N_BYTE, &input1_img, NULL);
    bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_
    ↪TYPE_EXT_1N_BYTE, &input2_img, NULL);
    bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_
    ↪TYPE_EXT_1N_BYTE, &output_img, NULL);
    bm_image_alloc_dev_mem(input1_img, 2);
    bm_image_alloc_dev_mem(input2_img, 2);
    bm_image_alloc_dev_mem(output_img, 2);
    unsigned char *in1_ptr[3] = {input1_data, input1_data + height * width, input1_
    ↪data + 2 * height * width};
    unsigned char *in2_ptr[3] = {input2_data, input2_data + width * height, input2_
    ↪data + 2 * height * width};
    bm_image_copy_host_to_device(input1_img, (void **)(in1_ptr));
    bm_image_copy_host_to_device(input2_img, (void **)(in2_ptr));

    ret = bmcv_image_absdiff(handle, input1_img, input2_img, output_img);
    if (BM_SUCCESS != ret) {
        bm_image_destroy(&input1_img);
        bm_image_destroy(&input2_img);
        bm_image_destroy(&output_img);
        return -1;
    }
    unsigned char *out_ptr[3] = {output_tpu, output_tpu + height * width, output_
    ↪data + 2 * height * width};
```

(续下页)

(接上页)

```

    ↪tpu + 2 * height * width};
    bm_image_copy_device_to_host(output_img, (void **)out_ptr);

    bm_image_destroy(&input1_img);
    bm_image_destroy(&input2_img);
    bm_image_destroy(&output_img);

    writeBin(dst_name, output_tpu, img_size);
    free(input1_data);
    free(input2_data);
    free(output_tpu);
    bm_dev_free(handle);
    return ret;
}

```

## 5.40 bmcv\_image\_add\_weighted

实现两张相同大小图像的加权融合，具体如下：

$$output = \alpha * input1 + \beta * input2 + \gamma$$

**接口形式：**

```

bm_status_t bmcv_image_add_weighted(
    bm_handle_t handle,
    bm_image input1,
    float alpha,
    bm_image input2,
    float beta,
    float gamma,
    bm_image output);

```

**参数说明：**

- `bm_handle_t handle`

输入参数。`bm_handle` 设备环境句柄，通过调用 `bm_dev_request` 获取。

- `bm_image input1`

输入参数。输入第一张图像的 `bm_image`，`bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 `attach` 已有的内存。

- `float alpha`

输入参数。第一张图像的权重。

- `bm_image input2`

输入参数。输入第二张图像的 bm\_image, bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存, 或者使用 bmcv\_image\_attach 来 attach 已有的内存。

- float beta  
输入参数。第二张图像的权重。
- float gamma  
输入参数。融合之后的偏移量。
- bm\_image output  
输出参数。输出 bm\_image。image 内存可以通过 bm\_image\_alloc\_dev\_mem 来开辟新的内存, 或者使用 bmcv\_image\_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 格式支持:

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

**注意事项：**

1. 在调用 bmcv\_image\_add\_weighted() 之前必须确保输入的 image 内存已经申请。
2. 输入、输出的 data\_type, image\_format 必须相同。

**示例代码**

```
#include "bmcv_api_ext_c.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size) {
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    }
    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size) {
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    }
    fclose(fp_dst);
}

int main() {
    int height = 1080;
    int width = 1920;
    int format = 8;
    float alpha = roundf((float)rand() / RAND_MAX * 10)/ 10.0;
    float beta = 1.0f - alpha;
    float gamma = ((float)rand() / RAND_MAX) * 255.0f;
    char *src1_name = "path/to/src1";
    char *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    int ret = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }
    int img_size = height * width * 3;
```

(续下页)

(接上页)

```
unsigned char *input1_data = (unsigned char *)malloc(width * height * 3);
unsigned char *input2_data = (unsigned char *)malloc(width * height * 3);
unsigned char *output_tpu = (unsigned char *)malloc(width * height * 3);

readBin(src1_name, input1_data, width * height);
readBin(src2_name, input2_data, img_size);

memset(output_tpu, 0, img_size * sizeof(unsigned char));

bm_image input1_img;
bm_image input2_img;
bm_image output_img;

bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_EXT_1N_BYTE, &input1_img, NULL);
bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_EXT_1N_BYTE, &input2_img, NULL);
bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_EXT_1N_BYTE, &output_img, NULL);
ret = bm_image_alloc_dev_mem(input1_img, 2);
ret = bm_image_alloc_dev_mem(input2_img, 2);
ret = bm_image_alloc_dev_mem(output_img, 2);

unsigned char* in1_ptr[3] = {input1_data, input1_data + height * width, input1_data + 2 * height * width};
unsigned char* in2_ptr[3] = {input2_data, input2_data + height * width, input2_data + 2 * height * width};
ret = bm_image_copy_host_to_device(input1_img, (void **)in1_ptr);
ret = bm_image_copy_host_to_device(input2_img, (void **)in2_ptr);
ret = bmcv_image_add_weighted(handle, input1_img, alpha, input2_img, beta, gamma, output_img);
unsigned char* out_ptr[3] = {output_tpu, output_tpu + height * width, output_tpu + 2 * height * width};
ret = bm_image_copy_device_to_host(output_img, (void **)out_ptr);

bm_image_destroy(&input1_img);
bm_image_destroy(&input2_img);
bm_image_destroy(&output_img);

if (ret) {
    printf("tpu_add_weighted failed!\n");
    return ret;
}
writeBin(dst_name, output_tpu, img_size);

free(input1_data);
free(input2_data);
free(output_tpu);
```

(续下页)

(接上页)

```

bm_dev_free(handle);
return ret;
}

```

## 5.41 bmcv\_image\_bayer2rgb

### 描述:

该接口可以将 bayerBG8 或 bayerRG8 格式图像转化为 rgb planar 格式图像。

### 语法:

```

1 bm_status_t bmcv_image_bayer2rgb(
2     bm_handle_t handle,
3     unsigned char* convd_kernel,
4     bm_image input,
5     bm_image output);

```

### 参数:

表 5.27: bmcv\_image\_bayer2rgb 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
convd_kernel	输入	卷积核数值。
input	输入	输入参数。输入 bayer 格式图像的 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。
output	输出	输出图像的 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。

### 返回值:

该函数成功调用时，返回 BM\_SUCCESS。

### 注意事项:

1. input 的格式目前支持 bayerBG8 或 bayerRG8，bm\_image\_create 步骤中 bayerBG8 创建为 FORMAT\_BAYER 格式，bayerRG8 创建为 FORMAT\_BAYER\_RG8 格式。

2. output 的格式是 rgb plannar, data\_type 均为 uint8 类型。
3. 输入图像的宽高需为偶数，且最大宽高均为 4096。
4. 如调用该接口的程序为多线程程序，需要在创建 bm\_image 前和销毁 bm\_image 后加线程锁。

### 代码示例：

```

1 #include "bmvc_api_ext_c.h"
2 #include "stdio.h"
3 #include "stdlib.h"
4 #include "string.h"
5 #include <assert.h>
6
7 #define NPU_NUM 32
8 #define KERNEL_SIZE 3 * 3 * 3 * 4 * 32
9 #define CONVD_MATRIX 12 * 9
10
11 const unsigned char convd_kernel_bg8[CONVD_MATRIX] = {1, 0, 1, 0, 0, 0, 1, 0, 1, //Rb
12                                         0, 0, 2, 0, 0, 0, 0, 0, 2, //Rg1
13                                         0, 0, 0, 0, 0, 2, 0, 2, //Rg2
14                                         0, 0, 0, 0, 0, 0, 0, 4, //Rr
15                                         4, 0, 0, 0, 0, 0, 0, 0, //Bb
16                                         2, 0, 2, 0, 0, 0, 0, 0, //Bg1
17                                         2, 0, 0, 0, 0, 2, 0, 0, //Bg2
18                                         1, 0, 1, 0, 0, 0, 1, 0, 1, //Br
19                                         0, 1, 0, 1, 0, 1, 0, 1, 0, //Gb
20                                         0, 0, 0, 0, 4, 0, 0, 0, //Gg1
21                                         0, 0, 0, 0, 0, 0, 4, 0, //Gg2
22                                         0, 1, 0, 1, 0, 1, 0, 1, 0};//Gr
23 const unsigned char convd_kernel_rg8[CONVD_MATRIX] = {4, 0, 0, 0, 0, 0, 0, 0, 0, //Rr
24                                         2, 0, 2, 0, 0, 0, 0, 0, 0, //Rg1
25                                         2, 0, 0, 0, 0, 2, 0, 0, //Rg2
26                                         1, 0, 1, 0, 0, 0, 1, 0, 1, //Rb
27                                         1, 0, 1, 0, 0, 0, 1, 0, 1, //Br
28                                         0, 0, 2, 0, 0, 0, 0, 0, 2, //Bg1
29                                         0, 0, 0, 2, 0, 2, 0, 0, 0, //Bg2
30                                         0, 0, 0, 0, 0, 0, 0, 0, 4, //Bb
31                                         1, 0, 1, 0, 0, 0, 1, 0, 1, //Gr
32                                         0, 0, 0, 0, 4, 0, 0, 0, //Gg1
33                                         0, 0, 0, 0, 0, 0, 4, 0, //Gg2
34                                         0, 1, 0, 1, 0, 1, 0, 1, 0};//Gb
35 void readBin(const char *path, unsigned char* input_data, int size) {
36     FILE *fp_src = fopen(path, "rb");
37     if (fp_src == NULL) {
38         printf("无法打开输出文件 %s\n", path);
39         return;
40     }
41     if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
42         printf("file size is less than %d required bytes\n", size);
43     };
44     fclose(fp_src);

```

(续下页)

(接上页)

```

45 }
46
47
48 int main() {
49     int width = 6000;
50     int height = 4000;
51     int src_type = 0;
52     char *src_name = "path/to/src";
53     char *output_path = "path/to/output";
54
55     bm_handle_t handle;
56     bm_status_t ret = bm_dev_request(&handle, 0);
57     if (ret != BM_SUCCESS) {
58         printf("Create bm handle failed. ret = %d\n", ret);
59         return -1;
60     }
61     unsigned char* output = (unsigned char*)malloc(width * height * 3);
62     unsigned char* input = (unsigned char*)malloc(width * height);
63     unsigned char kernel_data[KERNEL_SIZE] = {0};
64
65     // reconstructing kernel data
66     for (int i = 0; i < 12; i++) {
67         for (int j = 0; j < 9; j++) {
68             if (src_type == 0) {
69                 kernel_data[i * 9 * NPU_NUM + NPU_NUM * j] = convd_kernel_bg8[i * 9 + j];
70             } else {
71                 kernel_data[i * 9 * NPU_NUM + NPU_NUM * j] = convd_kernel_rg8[i * 9 + j];
72             }
73         }
74     }
75
76     readBin(src_name, input, height * width);
77     bm_image input_img;
78     bm_image output_img;
79
80     ret = bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_EXT_1N_
81     ↪BYTE, &input_img, NULL);
82     bm_image_create(handle, height, width, FORMAT_RGB_PLANAR, DATA_TYPE_EXT_
83     ↪1N_BYT, &output_img, NULL);
84     bm_image_alloc_dev_mem(input_img, BMCV_HEAP1_ID);
85     bm_image_alloc_dev_mem(output_img, BMCV_HEAP1_ID);
86     bm_image_copy_host_to_device(input_img, (void**)(&input));
87     bmcv_image_bayer2rgb(handle, kernel_data, input_img, output_img);
88
89     void* out_ptr[3] = {(void*)output,
90                         ((void*)((unsigned char*)output + width * height)),
91                         ((void*)((unsigned char*)output + 2 * width * height))};
92     bm_image_copy_device_to_host(output_img, (void**)(&out_ptr));
93     bm_image_destroy(&input_img);
94     bm_image_destroy(&output_img);

```

(续下页)

(接上页)

```
94  
95     FILE *fp_dst = fopen(output_path, "wb");  
96     fwrite((void *)output, 1, height * width * 3, fp_dst);  
97     free(output);  
98     free(input);  
99     bm_dev_free(handle);  
100    return ret;  
101 }
```

## 5.42 bmcv\_image\_bitwise\_and

两张大小相同的图片对应像素值进行按位与操作。

接口形式：

```
bm_status_t bmcv_image_bitwise_and(  
    bm_handle_t handle,  
    bm_image input1,  
    bm_image input2,  
    bm_image output);
```

参数说明：

- `bm_handle_t handle`  
输入参数。`bm_handle` 设备环境句柄，通过调用 `bm_dev_request` 获取。
- `input1`  
输入参数。输入第一张图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。
- `input2`  
输入参数。输入第二张图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。
- `output`  
输出参数。输出 `bm_image`。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

返回值说明：

- `BM_SUCCESS`: 成功
- 其他: 失败

### 格式支持:

- 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

- 输入和输出的色彩格式

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

### 注意事项:

- 在调用 bmcv\_image\_bitwise\_and() 之前必须确保输入的 image 内存已经申请。
- 输入、输出的 data\_type, image\_format 必须相同。

### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");

    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    }
}
```

(续下页)

(接上页)

```
fclose(fp_src);  
}  
  
static void writeBin(const char * path, unsigned char* input_data, int size)  
{  
    FILE *fp_dst = fopen(path, "wb");  
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){  
        printf("file size is less than %d required bytes\n", size);  
    }  
  
    fclose(fp_dst);  
}  
  
int main()  
{  
    int height = 1080;  
    int width = 1920;  
    char* src1_name = "path/to/src1";  
    char* src2_name = "path/to/src2";  
    char* dst_name = "path/to/dst";  
    int format = 8;  
    int ret = 0;  
    bm_handle_t handle;  
    ret = bm_dev_request(&handle, 0);  
    if (ret != BM_SUCCESS) {  
        printf("bm_dev_request failed. ret = %d\n", ret);  
        return -1;  
    }  
  
    int img_size = height * width * 3;  
  
    unsigned char* input1 = (unsigned char*)malloc(width * height * 3);  
    unsigned char* input2 = (unsigned char*)malloc(width * height * 3);  
    unsigned char* output = (unsigned char*)malloc(width * height * 3);  
  
    readBin(src1_name, input1, img_size);  
    readBin(src2_name, input2, img_size);  
  
    memset(output, 0, img_size * sizeof(unsigned char));  
  
    bm_image input1_img;  
    bm_image input2_img;  
    bm_image output_img;  
  
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,  
    ↵DATA_TYPE_EXT_1N_BYTE, &input1_img, NULL);  
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,  
    ↵DATA_TYPE_EXT_1N_BYTE, &input2_img, NULL);  
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,  
    ↵DATA_TYPE_EXT_1N_BYTE, &output_img, NULL);
```

(续下页)

(接上页)

```

ret = bm_image_alloc_dev_mem(input1_img, 2);
ret = bm_image_alloc_dev_mem(input2_img, 2);
ret = bm_image_alloc_dev_mem(output_img, 2);
unsigned char* in1_ptr[3] = {input1, input1 + height * width, input1 + 2 * [F]
                           ↵height * width};
unsigned char* in2_ptr[3] = {input2, input2 + height * width, input2 + 2 * [F]
                           ↵height * width};
ret = bm_image_copy_host_to_device(input1_img, (void **)in1_ptr);
ret = bm_image_copy_host_to_device(input2_img, (void **)in2_ptr);

ret = bmcv_image_bitwise_and(handle, input1_img, input2_img, output_img);

unsigned char* out_ptr[3] = {output, output + height * width, output + 2 * [F]
                           ↵height * width};
ret = bm_image_copy_device_to_host(output_img, (void **)out_ptr);

bm_image_destroy(&input1_img);
bm_image_destroy(&input2_img);
bm_image_destroy(&output_img);

if (ret) {
    printf("tpu_bitwise failed!\n");
    return ret;
}

writeBin(dst_name, output, img_size);
free(input1);
free(input2);
free(output);

bm_dev_free(handle);
return ret;
}

```

### 5.43 bmcv\_image\_bitwise\_or

两张大小相同的图片对应像素值进行按位或操作。

**接口形式：**

```

bm_status_t bmcv_image_bitwise_or(
    bm_handle_t handle,
    bm_image input1,
    bm_image input2,
    bm_image output);

```

**参数说明：**

- bm\_handle\_t handle

输入参数。bm\_handle 设备环境句柄，通过调用 bm\_dev\_request 获取。

- input1

输入参数。输入第一张图像的 bm\_image，bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。

- input2

输入参数。输入第二张图像的 bm\_image，bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。

- output

输出参数。输出 bm\_image。image 内存可以通过 bm\_image\_alloc\_dev\_mem 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 格式支持:

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

### 注意事项：

- 在调用 bmcv\_image\_bitwise\_or() 之前必须确保输入的 image 内存已经申请。
- 输入、输出的 data\_type, image\_format 必须相同。

### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");

    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };
}
```

(续下页)

(接上页)

```

    fclose(fp_dst);
}

int main()
{
    int height = 1080;
    int width = 1920;
    char* src1_name = "path/to/src1";
    char* src2_name = "path/to/src2";
    char* dst_name = "path/to/dst";
    int format = 8;
    int ret = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }

    int img_size = height * width * 3;

    unsigned char* input1 = (unsigned char*)malloc(width * height * 3);
    unsigned char* input2 = (unsigned char*)malloc(width * height * 3);
    unsigned char* output = (unsigned char*)malloc(width * height * 3);

    readBin(src1_name, input1, img_size);
    readBin(src2_name, input2, img_size);

    memset(output, 0, img_size * sizeof(unsigned char));

    bm_image input1_img;
    bm_image input2_img;
    bm_image output_img;

    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format, F
    ↵DATA_TYPE_EXT_1N_BYTE, &input1_img, NULL);
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format, F
    ↵DATA_TYPE_EXT_1N_BYTE, &input2_img, NULL);
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format, F
    ↵DATA_TYPE_EXT_1N_BYTE, &output_img, NULL);

    ret = bm_image_alloc_dev_mem(input1_img, 2);
    ret = bm_image_alloc_dev_mem(input2_img, 2);
    ret = bm_image_alloc_dev_mem(output_img, 2);
    unsigned char* in1_ptr[3] = {input1, input1 + height * width, input1 + 2 * F
    ↵height * width};
    unsigned char* in2_ptr[3] = {input2, input2 + height * width, input2 + 2 * F
    ↵height * width};
    ret = bm_image_copy_host_to_device(input1_img, (void **)in1_ptr);
    ret = bm_image_copy_host_to_device(input2_img, (void **)in2_ptr);
}

```

(续下页)

(接上页)

```

ret = bmcv_image_bitwise_or(handle, input1_img, input2_img, output_img);

unsigned char* out_ptr[3] = {output, output + height * width, output + 2 * [F]
                           height * width};
ret = bm_image_copy_device_to_host(output_img, (void **)out_ptr);

bm_image_destroy(&input1_img);
bm_image_destroy(&input2_img);
bm_image_destroy(&output_img);

if (ret) {
    printf("tpu_bitwise failed!\n");
    return ret;
}

writeBin(dst_name, output, img_size);
free(input1);
free(input2);
free(output);

bm_dev_free(handle);
return ret;
}

```

#### 5.44 bmcv\_image\_bitwise\_xor

两张大小相同的图片对应像素值进行按位异或操作。

**接口形式：**

```

bm_status_t bmcv_image_bitwise_xor(
    bm_handle_t handle,
    bm_image input1,
    bm_image input2,
    bm_image output);

```

**参数说明：**

- bm\_handle\_t handle

输入参数。bm\_handle 设备环境句柄，通过调用 bm\_dev\_request 获取。

- input1

输入参数。输入第一张图像的 bm\_image，bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。

- input2

输入参数。输入第二张图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `output`

输出参数。输出 `bm_image`。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 格式支持:

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

#### 注意事项:

1. 在调用 `bmcv_image_bitwise_xor()` 之前必须确保输入的 `image` 内存已经申请。
2. 输入、输出的 `data_type`, `image_format` 必须相同。

#### 示例代码

```
#include "bmvc_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");

    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_dst);
}

int main()
{
    int height = 1080;
    int width = 1920;
    char* src1_name = "path/to/src1";
    char* src2_name = "path/to/src2";
    char* dst_name = "path/to/dst";
    int format = 8;
    int ret = 0;
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    if (ret != BM_SUCCESS) {
        printf("bm_dev_request failed. ret = %d\n", ret);
        return -1;
    }

    int img_size = height * width * 3;

    unsigned char* input1 = (unsigned char*)malloc(width * height * 3);
    unsigned char* input2 = (unsigned char*)malloc(width * height * 3);
    unsigned char* output = (unsigned char*)malloc(width * height * 3);

    readBin(src1_name, input1, img_size);
    readBin(src2_name, input2, img_size);
```

(续下页)

(接上页)

```
memset(output, 0, img_size * sizeof(unsigned char));

bm_image input1_img;
bm_image input2_img;
bm_image output_img;

ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,F
↳DATA_TYPE_EXT_1N_BYTE, &input1_img, NULL);
ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,F
↳DATA_TYPE_EXT_1N_BYTE, &input2_img, NULL);
ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,F
↳DATA_TYPE_EXT_1N_BYTE, &output_img, NULL);

ret = bm_image_alloc_dev_mem(input1_img, 2);
ret = bm_image_alloc_dev_mem(input2_img, 2);
ret = bm_image_alloc_dev_mem(output_img, 2);
unsigned char* in1_ptr[3] = {input1, input1 + height * width, input1 + 2 * F
↳height * width};
unsigned char* in2_ptr[3] = {input2, input2 + height * width, input2 + 2 * F
↳height * width};
ret = bm_image_copy_host_to_device(input1_img, (void **)in1_ptr);
ret = bm_image_copy_host_to_device(input2_img, (void **)in2_ptr);

ret = bmcv_image_bitwise_xor(handle, input1_img, input2_img, output_img);

unsigned char* out_ptr[3] = {output, output + height * width, output + 2 * F
↳height * width};
ret = bm_image_copy_device_to_host(output_img, (void **)out_ptr);

bm_image_destroy(&input1_img);
bm_image_destroy(&input2_img);
bm_image_destroy(&output_img);

if (ret) {
    printf("tpu_bitwise failed!\n");
    return ret;
}

writeBin(dst_name, output, img_size);
free(input1);
free(input2);
free(output);

bm_dev_free(handle);
return ret;
}
```

## 5.45 bmcv\_image\_circle

### 【描述】

在图像上绘制圆形。支持绘制空心圆 (CIR\_LINE)、实心圆 (CIR\_SHAPE)、和圆形相框 (CIR\_EMPTY)。

### 【语法】

```

1 bm_status_t bmcv_image_circle(
2     bm_handle_t handle,
3     bm_image_t image,
4     bmcv_point_t center,
5     int radius,
6     bmcv_color_t color,
7     int line_width)

```

### 【参数】

表 5.28: bmcv\_image\_circle 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取。
image	输入/输出	输入/输出 <code>bm_image</code> 对象。
center	输入	圆心的位置 (x, y) (以图片左上角为原点)。
radius	输入	圆的半径。
color	输入	所填充颜色的 R/G/B 值。
line_width	输入	画圆的线宽，取值范围为 [-2, 15]，当传入 -1 时表示绘制实心圆，当传入 -2 时表示绘制相框。

### 【数据类型说明】

```

1 typedef struct {
2     int x;
3     int y;
4 } bmcv_point_t;
5
6 typedef struct {

```

(续下页)

(接上页)

```

7   unsigned char r;
8   unsigned char g;
9   unsigned char b;
10 } bmcv_color_t;
11
12 typedef enum {
13     CIR_EMPTY = -2,
14     CIR_SHAPE = -1,
15 } bmcv_cir_mode;

```

1. x, y 代表坐标点的横纵坐标。
2. CIR\_EMPTY 表示相框模式，即填充圆形外部区域，圆形内部不变；
3. CIR\_SHAPE 表示绘制实心圆；

#### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

#### 【注意】

1. 该 API 所需要满足的图像格式、尺寸等要求与 bmcv\_image\_vpp\_basic 中的表格相同。
2. 输入必须关联 device memory，否则返回失败。

#### 【代码示例】

```

bm_handle_t handle;
int src_w = 1920, src_h = 1080, dev_id = 0;
bmcv_point_t center = {960, 540};
bmcv_color_t color = {151, 255, 152};
int radius = 250, line_width = 10;
bm_image_format_ext src_fmt = FORMAT_YUV420P;
char *src_name = "1920x1080_yuv420.bin", *dst_name = "dst.bin";

bm_image src;
bm_dev_request(&handle, 0);
bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, [F
→NULL];
bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);

```

(续下页)

(接上页)

```
bm_read_bin(src,src_name);
bmcv_image_circle(handle, src, center, radius, color, line_width);
bm_write_bin(src, dst_name);

bm_image_destroy(&src);
bm_dev_free(handle);
```

#### 5.46 bmcv\_image\_convert\_to

该接口用于实现图像像素线性变化，具体数据关系可用如下公式表示：

$$y = kx + b$$

**接口形式：**

```
bm_status_t bmcv_image_convert_to (
    bm_handle_t handle,
    int input_num,
    bmcv_convert_to_attr convert_to_attr,
    bm_image* input,
    bm_image* output);
```

**参数说明：**

- **bm\_handle\_t handle**  
输入参数。bm\_handle 句柄。
- **int input\_num**  
输入参数。输入图片数，如果 input\_num > 1，那么多个输入图像必须是连续存储的（可以使用 bm\_image\_alloc\_contiguous\_mem 给多张图申请连续空间）。
- **bmcv\_convert\_to\_attr convert\_to\_attr**  
输入参数。每张图片对应的配置参数。
- **bm\_image\* input**  
输入参数。输入 bm\_image。每个 bm\_image 外部需要调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。
- **bm\_image\* output**  
输出参数。输出 bm\_image。每个 bm\_image 外部需要调用 bmcv\_image\_create 创建。image 内存可以通过 bm\_image\_alloc\_dev\_mem 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

**数据类型说明：**

```
struct bmcv_convert_to_attr_s{
    float alpha_0;
    float beta_0;
    float alpha_1;
    float beta_1;
    float alpha_2;
    float beta_2;
} bmcv_convert_to_attr;
```

#### 参数说明：

- float alpha\_0  
第 0 个 channel 进行线性变换的系数
- float beta\_0  
第 0 个 channel 进行线性变换的偏移
- float alpha\_1  
第 1 个 channel 进行线性变换的系数
- float beta\_1  
第 1 个 channel 进行线性变换的偏移
- float alpha\_2  
第 2 个 channel 进行线性变换的系数
- float beta\_2  
第 2 个 channel 进行线性变换的偏移

#### 返回值说明：

- BM\_SUCCESS: 成功
- 其他: 失败

#### 示例代码

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

typedef enum {
    UINT8_C1 = 0,
    UINT8_C3,
    INT8_C1,
    INT8_C3,
    FLOAT32_C1,
```

(续下页)

(接上页)

```

FLOAT32_C3,
MAX_COLOR_TYPE
} cv_color_e;

typedef enum {
    CONVERT_1N_TO_1N = 0
} convert_storage_mode_e;

typedef struct {
    int idx;
    int trials;
    bm_handle_t handle;
} convert_to_thread_arg_t;

typedef struct {
    float alpha;
    float beta;
} convert_to_arg_t;

#define BM1688_MAX_W (2048)
#define BM1688_MAX_H (2048)
#define MIN_W (16)
#define MIN_H (16)

#define __ALIGN_MASK(x, mask) (((x) + (mask)) & ~(mask))

#ifndef __linux
#define ALIGN(x, a) __ALIGN_MASK(x, (__typeof__(x))(a)-1)
#else
#define ALIGN(x, a) __ALIGN_MASK(x, (int)(a)-1)
#endif

typedef enum { MIN_TEST = 0, MAX_TEST, RAND_TEST, MAX_RAND_-
    MODE } rand_mode_e;

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }
    len = fwrite((void*)output_data, sizeof(unsigned char), size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_dst);
}

```

(续下页)

(接上页)

```

    return 0;
}

uint8_t fp32_to_u8(float fp32) {
    // Parse the binary representation of the float as an integer using pointer casting
    union {
        float f;
        uint32_t i;
    } u;
    u.f = fp32;

    // Extract sign, exponent, and mantissa
    uint32_t sign = (u.i >> 31) & 0x1;      // Sign bit
    uint32_t exponent = (u.i >> 23) & 0xFF; // Exponent part
    uint32_t fraction = u.i & 0x7FFFFFF; // Mantissa (fraction part)

    // If exponent is 0, it's a denormal number or zero - map directly to 0
    if (exponent == 0) {
        return 0;
    }

    // Calculate the actual float value
    float value = pow(-1, sign) * pow(2, exponent - 127) * (1 + fraction / pow(2, 23));

    // Map to u8 range (0-255)
    if (value < 0) value = 0;                // Clamp minimum value
    if (value > 255) value = 255;           // Clamp maximum value

    return (uint8_t)(value + 0.5f);         // Round and return
}

static bm_image_data_format_ext convert_to_type_translation(int cv_color, int cv_data_type, int if_input) {
    bm_image_data_format_ext image_format = DATA_TYPE_EXT_1N_BYTE;
    if (((UINT8_C3 == cv_color) || (UINT8_C1 == cv_color)) &&
        (CONVERT_1N_TO_1N == data_type)) {
        image_format = DATA_TYPE_EXT_1N_BYTE;
    } else if ((FLOAT32_C1 == cv_color) || (FLOAT32_C3 == cv_color)) {
        image_format = DATA_TYPE_EXT_FLOAT32;
    } else if (((INT8_C3 == cv_color) || (INT8_C1 == cv_color)) &&
               (CONVERT_1N_TO_1N == data_type)) {
        image_format = DATA_TYPE_EXT_1N_BYTE_SIGNED;
    }

    return image_format;
}

int main(){
    int dev_id = 0;
    bm_handle_t handle = NULL;
}

```

(续下页)

(接上页)

```

int image_n = 3;
int image_c = 3;
int image_w = 1920;
int image_h = 1080;

convert_to_arg_t convert_to_arg[image_c];
int convert_format = CONVERT_1N_TO_1N;

bm_status_t ret = bm_dev_request(&handle, dev_id);

memset(convert_to_arg, 0x0, sizeof(convert_to_arg));
for(int i = 0; i < image_c; i++){
    convert_to_arg[i].alpha = ((float)(rand() % 20)) / (float)10;
    convert_to_arg[i].beta = ((float)(rand() % 20)) / (float)10 - 1;
}

int image_len = 0;
image_len = image_n * image_c * image_w * image_h;

int input_data_type = UINT8_C3;
unsigned char *input = malloc(image_len * sizeof(unsigned char));
memset(input, 0x0, image_len);

int output_data_type = FLOAT32_C3;
float *bmvc_res = malloc(image_len * sizeof(float));
for(size_t i = 0; i < image_len; ++i) {
    bmvc_res[i] = 0;
}

for(int i = 0; i < image_len; i++){
    input[i] = rand() % 255;
}

bm_image_data_format_ext input_data_format_ext, output_data_format_ext;
bmvc_convert_to_attr convert_to_attr;
convert_to_attr.alpha_0 = convert_to_arg[0].alpha;
convert_to_attr.beta_0 = convert_to_arg[0].beta;
convert_to_attr.alpha_1 = convert_to_arg[1].alpha;
convert_to_attr.beta_1 = convert_to_arg[1].beta;
convert_to_attr.alpha_2 = convert_to_arg[2].alpha;
convert_to_attr.beta_2 = convert_to_arg[2].beta;

input_data_format_ext = convert_to_type_translation(input_data_type, F
    ↵convert_format, 1);
output_data_format_ext = convert_to_type_translation(output_data_type, F
    ↵convert_format, 0);

bm_image input_images[image_n];
bm_image output_images[32];

for(int img_idx = 0; img_idx < image_n; img_idx++){

}

```

(续下页)

(接上页)

```

    bm_image_create(handle,
                    image_h,
                    image_w,
                    FORMAT_RGB_PLANAR,
                    input_data_format_ext,
                    &input_images[img_idx],
                    NULL);
}
for(int img_idx = 0; img_idx < image_n; img_idx++){
    unsigned char *input_img_data = input + image_len / image_n * img_idx;
    bm_image_copy_host_to_device(input_images[img_idx], (void **)&input_
→img_data);
}
for(int img_idx = 0; img_idx < image_n; img_idx++){
    bm_image_create(handle,
                    image_h,
                    image_w,
                    FORMAT_RGB_PLANAR,
                    output_data_format_ext,
                    &output_images[img_idx],
                    NULL);
}
bm_image_alloc_contiguous_mem(image_n, output_images, BMCV_HEAP0_
→ID);

bmcv_image_convert_to(handle, image_n, convert_to_attr, input_images,[F
→output_images);

char filename[256];

unsigned char *res_u8_data = (unsigned char*)malloc(image_len *[F
→sizeof(unsigned char));
for(int img_idx = 0; img_idx < image_n; img_idx++){
    float *res_img_data = bmcv_res + image_len / image_n * img_idx;
    int image_byte_size[4] = {0};
    bm_image_get_byte_size(output_images[img_idx], image_byte_size);
    void* out_addr[4] = {(void *)res_img_data,
                         (void*)((float*)res_img_data + image_byte_size[0]/
→sizeof(float)),
                         (void*)((float*)res_img_data + (image_byte_size[0] + image_
→byte_size[1])/sizeof(float)),
                         (void*)((float*)res_img_data + (image_byte_size[0] + image_
→byte_size[1] + image_byte_size[2])/sizeof(float))};

    bm_image_copy_device_to_host(output_images[img_idx], (void **)&out_
→addr);
    for (int j = 0; j < image_len / image_n; j++) {
        res_u8_data[j] = fp32_to_u8(res_img_data[j]);
    }
    sprintf(filename, sizeof(filename), "daily_test_convert_to_dst_%d.bin", img_
→idx);
}

```

(续下页)

(接上页)

```

    ↵idx);
        writeBin(filename, res_u8_data, image_len / image_n);
    }

    free(res_u8_data);
    free(input);
    free(bmcv_res);

    for(int i = 0; i < image_n; i++){
        bm_image_destroy(&input_images[i]);
    }
    for(int i = 0; i < image_n; i++){
        bm_image_destroy(&output_images[i]);
    }
    bm_dev_free(handle);

    return ret;
}

```

**格式支持:**

1. 该接口支持下列 image\_format:
  - FORMAT\_BGR\_PLANAR
  - FORMAT\_RGB\_PLANAR
  - FORMAT\_BGR\_PACKED
  - FORMAT\_RGB\_PACKED
  - FORMAT\_YUV420P
  - FORMAT\_GRAY
2. 该接口支持下列情形 data type 之间的转换:
  - DATA\_TYPE\_EXT\_1N\_BYTE —> DATA\_TYPE\_EXT\_FLOAT32
  - DATA\_TYPE\_EXT\_FLOAT32 —> DATA\_TYPE\_EXT\_1N\_BYTE
  - DATA\_TYPE\_EXT\_1N\_BYTE —> DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED
  - DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED —> DATA\_TYPE\_EXT\_1N\_BYTE
  - DATA\_TYPE\_EXT\_FLOAT32 —> DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED
  - DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED —> DATA\_TYPE\_EXT\_FLOAT32

**注意事项:**

1. 在调用 bmcv\_image\_convert\_to() 之前必须确保输入的 image 内存已经申请。
2. 输入的各个 image 的宽、高以及 data\_type、image\_format 必须相同。
3. 输出的各个 image 的宽、高以及 data\_type、image\_format 必须相同。

4. 输入 image 宽、高必须等于输出 image 宽高。
5. 输入 image 的 image\_format 与输出 image 的 image\_format 必须相同。
6. image\_num 必须大于 0。
7. 输入图片的最小尺寸为 16 \* 16，最大尺寸为 4096 \* 4096，当输入 data\_type 为 DATA\_TYPE\_EXT\_1N\_BYT 时，最大尺寸可以支持 8192 \* 8192。
8. 输出图片的最小尺寸为 16 \* 16，最大尺寸 8192 \* 8192。

## 5.47 bmcv\_image\_copy\_to

### 描述：

该接口实现将一幅图像拷贝到目的图像的对应内存区域。

### 语法：

```

1 bm_status_t bmcv_image_copy_to(
2     bm_handle_t handle,
3     bmcv_copy_to_attr_t copy_to_attr,
4     bm_image      input,
5     bm_image      output
6 );

```

### 参数：

表 5.29: bmcv\_image\_copy\_to 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
copy_to_attr	输入	api 所对应的属性配置。
input	输入	输入 bm_image。
output	输出	输出 bm_image。

### 返回值：

该函数成功调用时，返回 BM\_SUCCESS。

### 数据类型说明：

```

1 typedef struct bmcv_copy_to_attr_s {
2     int      start_x;
3     int      start_y;
4     unsigned char padding_r;
5     unsigned char padding_g;
6     unsigned char padding_b;
7     int if_padding;
8 } bmcv_copy_to_attr_t;

```

- padding\_b 表示当 input 的图像要小于输出图像的情况下，多出来的图像 b 通道上被填充的值。
- padding\_r 表示当 input 的图像要小于输出图像的情况下，多出来的图像 r 通道上被填充的值。
- padding\_g 表示当 input 的图像要小于输出图像的情况下，多出来的图像 g 通道上被填充的值。
- start\_x 描述了 copy\_to 拷贝到输出图像所在的起始横坐标。
- start\_y 描述了 copy\_to 拷贝到输出图像所在的起始纵坐标。
- if\_padding 表示当 input 的图像要小于输出图像的情况下，是否需要对多余的图像区域填充特定颜色，0 表示不需要，1 表示需要。当该值填 0 时，padding\_r, padding\_g, padding\_b 的设置将无效。

#### 格式支持：

##### 1. 输入和输出的数据类型

num	input data type	output data type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16
6	DATA_TYPE_EXT_FLOAT32	DATA_TYPE_EXT_FLOAT32
7	DATA_TYPE_EXT_1N_BYTE_SIGN	DATA_TYPE_EXT_1N_BYTE_SIGNED

##### 2. 输入和输出支持的色彩格式

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_BGR_PACKED
5	FORMAT_GRAY

#### 注意：

1. 在调用 bmcv\_image\_copy\_to() 之前必须确保输入的 image 内存已经申请。
2. 为了避免内存越界，输入图像 width + start\_x 必须小于等于输出图像 width stride。

#### 代码示例：

```

1 #include <assert.h>
2 #include <stdint.h>
```

(续下页)

(接上页)

```

3 #include <stdio.h>
4 #include <stdlib.h>
5 #include "bmvc_api_ext_c.h"
6
7 typedef enum { COPY_TO_GRAY = 0, COPY_TO_BGR, COPY_TO_RGB } padding_
8 ↪corlor_e;
9 typedef enum { PLANNER = 0, PACKED } padding_format_e;
10
11 static int writeBin(const char* path, void* output_data, int size)
12 {
13     int len = 0;
14     FILE* fp_dst = fopen(path, "wb+");
15
16     if (fp_dst == NULL) {
17         perror("Error opening file\n");
18         return -1;
19     }
20
21     len = fwrite((void*)output_data, 1, size, fp_dst);
22     if (len < size) {
23         printf("file size = %d is less than required bytes = %d\n", len, size);
24         return -1;
25     }
26
27     fclose(fp_dst);
28     return 0;
29 }
30
31 int main() {
32     int dev_id = 0;
33     bm_handle_t handle;
34     bm_status_t ret = bm_dev_request(&handle, dev_id);
35
36     int in_w    = 400;
37     int in_h    = 400;
38     int out_w   = 800;
39     int out_h   = 800;
40     int start_x = 200;
41     int start_y = 200;
42
43     int image_format = FORMAT_RGB_PLANAR;
44     int data_type = DATA_TYPE_EXT_FLOAT32;
45     int channel   = 3;
46
47     int image_n = 1;
48
49     float* src_data = (float *)malloc(image_n * channel * in_w * in_h * sizeof(float));
50     float* res_data = (float *)malloc(image_n * channel * out_w * out_h * sizeof(float));
51
52     for (int i = 0; i < image_n * channel * in_w * in_h; i++) {

```

(续下页)

(接上页)

```
53     src_data[i] = rand() % 255;
54 }
55 // calculate res
56 bmcv_copy_to_attr_t copy_to_attr;
57 copy_to_attr.start_x = start_x;
58 copy_to_attr.start_y = start_y;
59 copy_to_attr.padding_r = 0;
60 copy_to_attr.padding_g = 0;
61 copy_to_attr.padding_b = 0;
62 copy_to_attr.if_padding = 1;
63 bm_image input, output;
64 bm_image_create(handle,
65     in_h,
66     in_w,
67     (bm_image_format_ext)image_format,
68     (bm_image_data_format_ext)data_type,
69     &input, NULL);
70 bm_image_alloc_dev_mem(input, BMCV_HEAP1_ID);
71 bm_image_copy_host_to_device(input, (void **)&src_data);
72 bm_image_create(handle,
73     out_h,
74     out_w,
75     (bm_image_format_ext)image_format,
76     (bm_image_data_format_ext)data_type,
77     &output, NULL);
78 bm_image_alloc_dev_mem(output, BMCV_HEAP1_ID);
79 ret = bmcv_image_copy_to(handle, copy_to_attr, input, output);
80
81 if (BM_SUCCESS != ret) {
82     printf("bmcv_copy_to error 1 !!!\n");
83     bm_image_destroy(&input);
84     bm_image_destroy(&output);
85     free(src_data);
86     free(res_data);
87     return -1;
88 }
89 bm_image_copy_device_to_host(output, (void **)&res_data);
90
91 char *dst_name = "path/to/dst";
92 writeBin(dst_name, res_data, out_w * out_h * 3);
93 writeBin("path/to/src", src_data, in_h * in_w * 3);
94
95 bm_image_destroy(&input);
96 bm_image_destroy(&output);
97 free(src_data);
98 free(res_data);
99
100 return ret;
101 }
```

## 5.48 bmcv\_image\_csc\_convert\_to

描述：

该 API 可以实现对多张图片的 crop、color-space-convert、resize、padding、convert\_to 及其任意若干个功能的组合。

语法：

```
1 bm_status_t bmcv_image_csc_convert_to(
2     bm_handle_t handle,
3     int in_img_num,
4     bm_image* input,
5     bm_image* output,
6     int* crop_num_vec = NULL,
7     bmcv_rect_t* crop_rect = NULL,
8     bmcv_padding_attr_t* padding_attr = NULL,
9     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
10    csc_type_t csc_type = CSC_MAX_ENUM,
11    csc_matrix_t* matrix = NULL,
12    bmcv_convert_to_attr* convert_to_attr = NULL);
```

参数：

表 5.30: bmcv\_image\_csc\_convert\_to 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
in_img_num	输入	输入 bm_image 数量。
*input	输入	输入 bm_image 对象指针，其指向空间的长度由 in_img_num 决定。
* output	输出	输出 bm_image 对象指针，其指向空间的长度由 in_img_num 和 crop_num_vec 共同决定，即所有输入图片 crop 数量之和。
* crop_num_vec	输入	该指针指向对每张输入图片进行 crop 的数量，其指向空间的长度由 in_img_num 决定，如果不使用 crop 功能可填 NULL。
* crop_rect	输入	输出 bm_image 对象所对应的在输入图像上 crop 的参数。
* padding_attr	输入	所有 crop 的目标小图在 dst image 中的位置信息以及要 padding 的各通道像素值，若不使用 padding 功能则设置为 NULL。
algorithm	输入	resize 算法选择，包括 BMCV_INTER_NEAREST、BMCV_INTER_LINEAR 和 BMCV_INTER_BICUBIC 三种，默认情况下是双线性差值。
csc_type	输入	color space convert 参数类型选择，填 CSC_MAX_ENUM 则使用默认值，默认为 CSC_YCbCr2RGB_BT601 或者 CSC_RGB2YCbCr_BT601，支持的类型见表 csc_type
*matrix	输入	如果 csc_type 选择 CSC_USER_DEFINED_MATRIX，则需要传入系数矩阵。
*convert_to_attr	输入	线性变换系数。当 convert_to_attr 参数非空时，输出格式仅支持 FORMAT_RGB_PLANAR 和 FORMAT_BGR_PLANAR。

表 5.31: csc\_type

CSC_YCbCr2RGB_BT601
CSC_YPbPr2RGB_BT601
CSC_RGB2YCbCr_BT601
CSC_YCbCr2RGB_BT709
CSC_RGB2YCbCr_BT709
CSC_RGB2YPbPr_BT601
CSC_YPbPr2RGB_BT709
CSC_RGB2YPbPr_BT709
CSC_FANCY_PbPr_BT601
CSC_FANCY_PbPr_BT709
CSC_USER_DEFINED_MATRIX
CSC_MAX_ENUM

**数据类型说明：**

```

1 typedef struct bmcv_rect {
2     int start_x;
3     int start_y;
4     int crop_w;
5     int crop_h;
6 } bmcv_rect_t;
```

start\_x、start\_y、crop\_w、crop\_h 分别表示每个输出 bm\_image 对象所对应的在输入图像上 crop 的参数，包括起始点 x 坐标、起始点 y 坐标、crop 图像的宽度以及 crop 图像的高度。图像左上顶点作为坐标原点。如果不使用 crop 功能可填 NULL。

```

1 typedef struct bmcv_padding_attr_s {
2     unsigned int dst_crop_stx;
3     unsigned int dst_crop_sty;
4     unsigned int dst_crop_w;
5     unsigned int dst_crop_h;
6     unsigned char padding_r;
7     unsigned char padding_g;
8     unsigned char padding_b;
9     int         if_memset;
10 } bmcv_padding_attr_t;
```

1. 目标小图的左上角顶点相对于 dst image 原点（左上角）的 offset 信息：dst\_crop\_stx 和 dst\_crop\_sty；
2. 目标小图经 resize 后的宽高：dst\_crop\_w 和 dst\_crop\_h；
3. dst image 如果是 RGB 格式，各通道需要 padding 的像素值信息：padding\_r、padding\_g、padding\_b，当 if\_memset=1 时有效，如果是 GRAY 图像可以将三个值均设置为同一个值；

4. if\_memset 表示要不要在该 api 内部对 dst image 按照各个通道的 padding 值做 memset，仅支持 RGB 和 GRAY 格式的图像。如果设置为 0 则用户需要在调用该 api 前，根据需要 padding 的像素值信息，调用 bmlib 中的 api 直接对 device memory 进行 memset 操作，如果用户对 padding 的值不关心，可以设置为 0 忽略该步骤。

```

1  typedef struct {
2      short csc_coe00;
3      short csc_coe01;
4      short csc_coe02;
5      unsigned char csc_add0;
6      unsigned char csc_sub0;
7      short csc_coe10;
8      short csc_coe11;
9      short csc_coe12;
10     unsigned char csc_add1;
11     unsigned char csc_sub1;
12     short csc_coe20;
13     short csc_coe21;
14     short csc_coe22;
15     unsigned char csc_add2;
16     unsigned char csc_sub2;
17 } csc_matrix_t;

```

自定义 csc\_matrix 的系数。

```

1  typedef struct bmcv_convert_to_attr_s{
2      float alpha_0;
3      float beta_0;
4      float alpha_1;
5      float beta_1;
6      float alpha_2;
7      float beta_2;
8 } bmcv_convert_to_attr;

```

- alpha\_0 描述了第 0 个 channel 进行线性变换的系数
- beta\_0 描述了第 0 个 channel 进行线性变换的偏移
- alpha\_1 描述了第 1 个 channel 进行线性变换的系数
- beta\_1 描述了第 1 个 channel 进行线性变换的偏移
- alpha\_2 描述了第 2 个 channel 进行线性变换的系数
- beta\_2 描述了第 2 个 channel 进行线性变换的偏移

**返回值：**

该函数成功调用时，返回 BM\_SUCCESS。

**格式支持：**

1. 支持的数据类型为：

num	input data_type	output data_type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

2. 输入支持色彩格式为：

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_COMPRESSED
16	FORMAT_YUV422_YUYV
17	FORMAT_YUV422_YVYU
18	FORMAT_YUV422_UYVY
19	FORMAT_YUV422_VYUY

3. 输出支持色彩格式为：

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_YUV422_YUYV
16	FORMAT_YUV422_YVYU
17	FORMAT_YUV422_UYVY
18	FORMAT_YUV422_VYUY
19	FORMAT_HSV_PLANAR

4. 图片缩放倍数 ((crop.width / output.width) 以及 (crop.height / output.height)) 限制在 1/128 ~ 128 之间。
5. 输入输出的宽高 (src.width, src.height, dst.widht, dst.height) 限制在 16 ~ 8192 之间。
6. 输入必须关联 device memory，否则返回失败。
7. FORMAT\_COMPRESSED 是 VPU 解码后内置的一种压缩格式，它包括 4 个部分：Y compressed table、Y compressed data、CbCr compressed table 以及 CbCr compressed data。请注意 bm\_image 中这四部分存储的顺序与 FFMPEG 中 AVFrame 稍有不同，如果需要 attach AVFrame 中 device memory 数据到 bm\_image 中时，对应关系如下，关于 AVFrame 详细内容请参考 VPU 的用户手册。

```

bm_device_mem_t src_plane_device[4];
src_plane_device[0] = bm_mem_from_device((u64)avframe->data[6],
                                         avframe->linesize[6]);
src_plane_device[1] = bm_mem_from_device((u64)avframe->data[4],
                                         avframe->linesize[4] * avframe->h);
src_plane_device[2] = bm_mem_from_device((u64)avframe->data[7],
                                         avframe->linesize[7]);
src_plane_device[3] = bm_mem_from_device((u64)avframe->data[5],
                                         avframe->linesize[4] * avframe->h / 2);

bm_image_attach(*compressed_image, src_plane_device);

```

## 5.49 bmcv\_image\_csc\_overlay

描述：

该 API 可以实现对单张图片的 crop、color-space-convert、resize、padding、convert\_to、flip、draw/fill rect、circle、overlay 任意若干个功能的组合。

语法：

```
1 DECL_EXPORT bm_status_t bmcv_image_csc_overlay(
2     bm_handle_t           handle,
3     int                  crop_num,
4     bm_image              input,
5     bm_image*             output,
6     bmcv_rect_t*          crop_rect = NULL,
7     bmcv_padding_attr_t*  padding_attr = NULL,
8     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
9     csc_type_t            csc_type = CSC_MAX_ENUM,
10    bmcv_flip_mode         flip_mode = NO_FLIP,
11    bmcv_convert_to_attr*  convert_to_attr = NULL,
12    bmcv_overlay_attr*    overlay_attr = NULL,
13    bmcv_draw_rect_attr*   draw_rect_attr = NULL,
14    bmcv_fill_rect_attr*  fill_rect_attr = NULL,
15    bmcv_circle_attr*     circle_attr = NULL);
```

参数：

表 5.32: bmcv\_image\_csc\_overlay 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
crop_num	输入	对输入 bm_image 进行 crop 的数量。
input	输入	输入 bm_image 对象。
* output	输出	输出 bm_image 对象指针，其指向空间的长度由 crop_num 决定。
* crop_rect	输入	输出 bm_image 对象所对应的在输入图像上 crop 的参数，其指向空间的长度由 crop_num 决定。若不使用 crop 功能则设置为 NULL。
* padding_attr	输入	所有 crop 的目标小图在 dst image 中的位置信息以及要 padding 的各通道像素值，其指向空间的长度由 crop_num 决定。若不使用 padding 功能则设置为 NULL。
algorithm	输入	resize 算法选择，包括 BMCV_INTER_NEAREST、BMCV_INTER_LINEAR 和 BMCV_INTER_BICUBIC 三种，默认情况下是双线性差值。
csc_type	输入	color space convert 参数类型选择，填 CSC_MAX_ENUM 则使用默认值，默认为 CSC_YCbCr2RGB_BT601 或者 CSC_RGB2YCbCr_BT601。
flip_mode	输入	flip 模式，可选择水平翻转，垂直翻转和旋转 180 度。
*convert_to_attr	输入	线性变换系数，其指向空间的长度为 1。若不使用 convert_to 功能则设置为 NULL。当 convert_to_attr 参数非空时，输出格式仅支持 FORMAT_RGB_PLANAR 和 FORMAT_BGR_PLANAR。
*overlay_attr	输入	overlay 功能参数，其指向空间的长度由 crop_num 决定。若不使用 overlay 功能则设置为 NULL。
*draw_rect_attr	输入	绘制长方形空心框功能参数，其指向空间的长度由 crop_num 决定。若不使用画框功能则设置为 NULL。
*fill_rect_attr	输入	绘制长方形实心框功能参数，其指向空间的长度由 crop_num 决定。若不使用填框功能则设置为 NULL。
*circle_attr	输入	绘制圆功能参数，其指向空间的长度由 crop_num 决定。若不使用画圆功能则设置为 NULL。

## 数据类型说明：

除下述数据类型之外，未说明的数据类型请参阅 bmcv\_image\_csc\_convert\_to 接口文档。

```

1  typedef enum {
2      NO_FLIP = 0,
3      HORIZONTAL_FLIP = 1,
4      VERTICAL_FLIP = 2,
5      ROTATE_180 = 3,
6  } bmcv_flip_mode;

```

NO\_FLIP 表示不做处理，HORIZONTAL\_FLIP 表示水平翻转，VERTICAL\_FLIP 表示垂直翻转，ROTATE\_180 表示旋转 180 度。

```

1  typedef struct bmcv_overlay_attr {
2      char overlay_num;
3      bmcv_rect_t* overlay_info;
4      bm_image* overlay_image;
5  } bmcv_overlay_attr;

```

- overlay\_num 描述需要叠图的数量，该接口最高支持在单张目的图上叠 16 张图。
- overlay\_info 描述叠图位置信息指针，其指向空间的长度由 overlay\_num 决定。
- overlay\_image 描述需要叠图的 bm\_image 指针，其指向空间的长度由 overlay\_num 决定。

```

1  typedef struct bmcv_draw_rect_attr {
2      char rect_num;
3      bmcv_rect_t draw_rect[4];
4      unsigned int color[4];
5      short line_width[4];
6  } bmcv_draw_rect_attr;

```

- rect\_num 描述需要画框的数量，该接口最高支持在单张目的图上画 4 个框。
- draw\_rect 描述画框位置信息。
- color 描述画框的颜色信息，可由 ((r [F] 16) | (g [F] 8) | b) 表示。
- line\_width 描述画框的线宽。

```

1  typedef struct bmcv_fill_rect_attr {
2      char rect_num;
3      bmcv_rect_t fill_rect[4];
4      unsigned int color[4];
5  } bmcv_fill_rect_attr;

```

- rect\_num 描述需要填框的数量，该接口最高支持在单张目的图上填 4 个框。
- draw\_rect 描述填框位置信息。
- color 描述填框的颜色信息，可由 ((r [F] 16) | (g [F] 8) | b) 表示。

```

1 typedef struct bmcv_circle_attr {
2     bmcv_point_t center;
3     short radius;
4     bmcv_color_t color;
5     signed char line_width;
6 } bmcv_circle_attr;

```

- 该接口最高支持在单张目的图上画 1 个圆。
- center 描述画圆中心位置信息。
- radius 描述画圆半径。
- color 描述画圆的颜色信息。
- line\_width 描述画圆的线宽，取值为 [-2, 15]，-2 表示画圆形相框，-1 表示画实心圆。

#### 返回值：

该函数成功调用时，返回 BM\_SUCCESS。

#### 格式支持：

本接口格式支持情况与 bmcv\_image\_csc\_convert\_to 接口相同，请参阅该接口文档。

### 5.50 bmcv\_image\_draw\_lines

可以实现在一张图像上画一条或多条线段，从而可以实现画多边形的功能，并支持指定线的颜色和线的宽度。

#### 接口形式：

```

typedef struct {
    int x;
    int y;
} bmcv_point_t;

typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} bmcv_color_t;

bm_status_t bmcv_image_draw_lines(
    bm_handle_t handle,
    bm_image img,
    const bmcv_point_t* start,
    const bmcv_point_t* end,
    int line_num,

```

(续下页)

(接上页)

```
bmcv_color_t color,  
int thickness);
```

#### 参数说明：

- `bm_handle_t handle`

输入参数。`bm_handle` 句柄。

- `bm_image img`

输入/输出参数。需处理图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `const bmcv_point_t* start`

输入参数。线段起始点的坐标指针, 指向的数据长度由 `line_num` 参数决定。图像左上角为原点, 向右延伸为 x 方向, 向下延伸为 y 方向。

- `const bmcv_point_t* end`

输入参数。线段结束点的坐标指针, 指向的数据长度由 `line_num` 参数决定。图像左上角为原点, 向右延伸为 x 方向, 向下延伸为 y 方向。

- `int line_num`

输入参数。需要画线的数量。

- `bmcv_color_t color`

输入参数。画线的颜色, 分别为 RGB 三个通道的值。

- `int thickness`

输入参数。画线的宽度, 对于 YUV 格式的图像建议设置为偶数。

#### 返回值说明：

- `BM_SUCCESS`: 成功

- 其他: 失败

#### 格式支持：

该接口目前支持以下 `image_format`:

num	image_format
1	FORMAT_GRAY
2	FORMAT_YUV420P
3	FORMAT_YUV422P
4	FORMAT_YUV444P
5	FORMAT_NV12
6	FORMAT_NV21
7	FORMAT_NV16
8	FORMAT_NV61

目前支持以下 data\_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

### 示例代码

```
#include "bmvc_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>

#define ALIGN(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
#define SATURATE(a, s, e) ((a) > (e) ? (e) : ((a) < (s) ? (s) : (a)))
#define IMAGE_CHN_NUM_MAX 3

static int line_num = 6;
static bmcv_point_t start[6] = {{0, 0}, {500, 0}, {500, 500}, {0, 500}, {0, 0}, {0, 500}};
static bmcv_point_t end[6] = {{500, 0}, {500, 500}, {0, 500}, {0, 0}, {500, 500}, {500, 0}};
static unsigned char color[3] = {255, 0, 0};
static int thickness = 4;

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    len = fwrite((void*)output_data, 1, size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }
}
```

(续下页)

(接上页)

```
}

fclose(fp_dst);
return 0;
}

static int readBin(const char* path, void* input_data)
{
int len;
int size;
FILE* fp_src = fopen(path, "rb");

if (fp_src == NULL) {
    perror("Error opening file\n");
    return -1;
}

fseek(fp_src, 0, SEEK_END);
size = ftell(fp_src);
fseek(fp_src, 0, SEEK_SET);

len = fread((void*)input_data, 1, size, fp_src);
if (len < size) {
    printf("file size = %d is less than required bytes = %d\n", len, size);
    return -1;
}

fclose(fp_src);
return 0;
}

int main()
{
int width = 1920;
int height = 1080;
int format = 0;
int ret = 0;
char *input_path = "path/to/input";
char *output_path = "path/to/output";
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);
if (ret != BM_SUCCESS) {
    printf("bm_dev_request failed. ret = %d\n", ret);
    return -1;
}
unsigned char* data_tpu = (unsigned char*)malloc(width * height * IMAGE_
CHN_NUM_MAX * sizeof(unsigned char));
int offset_list[IMAGE_CHN_NUM_MAX] = {0};
int total_size = 0;
ret = readBin(input_path, data_tpu);

unsigned char* input = data_tpu;
```

(续下页)

(接上页)

```

const bmcv_point_t* p1 = start;
const bmcv_point_t* p2 = end;

bm_image input_img;
unsigned char* in_ptr[IMAGE_CHN_NUM_MAX] = {0};
bmcv_color_t rgb = {color[0], color[1], color[2]};

ret = bm_image_create(handle, height, width, (bm_image_format_ext)format, F
DATA_TYPE_EXT_1N_BYTE, &input_img, NULL);
ret = bm_image_alloc_dev_mem(input_img, BMCV_HEAP1_ID);

offset_list[0] = width * height;
offset_list[1] = ALIGN(width, 2) * ALIGN(height, 2) >> 2;
offset_list[2] = ALIGN(width, 2) * ALIGN(height, 2) >> 2;

in_ptr[0] = input;
in_ptr[1] = input + offset_list[0];
in_ptr[2] = input + offset_list[0] + offset_list[1];

ret = bm_image_copy_host_to_device(input_img, (void**)in_ptr);
bmcv_image_draw_lines(handle, input_img, p1, p2, line_num, rgb, thickness);
ret = bm_image_copy_device_to_host(input_img, (void**)in_ptr);

bm_image_destroy(&input_img);

for (int i = 0; i < IMAGE_CHN_NUM_MAX; ++i) {
    total_size += offset_list[i];
}

ret = writeBin(output_path, data_tpu, total_size);

free(data_tpu);
bm_dev_free(handle);
return ret;
}

```

## 5.51 bmcv\_image\_draw\_point

该接口用于在图像上填充一个或者多个 point。

**接口形式：**

```

bm_status_t bmcv_image_draw_point(
    bm_handle_t handle,
    bm_image image,
    int point_num,
    bmcv_point_t* coord,
    int length,
    unsigned char r,
)

```

(续下页)

(接上页)

```
unsigned char g,  
unsigned char b);
```

#### 传入参数说明:

- `bm_handle_t handle`  
输入参数。设备环境句柄，通过调用 `bm_dev_request` 获取。
- `bm_image image`  
输入参数。需要在其上填充 point 的 `bm_image` 对象。
- `int point_num`  
输入参数。需填充 point 的数量，指 `coord` 指针中所包含的 `bmcv_point_t` 对象个数。
- `bmcv_point_t* rect`  
输入参数。point 位置指针。具体内容参考下面的数据类型说明。
- `int length`  
输入参数。point 的边长。
- `unsigned char r`  
输入参数。矩形填充颜色的 r 分量。
- `unsigned char g`  
输入参数。矩形填充颜色的 g 分量。
- `unsigned char b`  
输入参数。矩形填充颜色的 b 分量。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 数据类型说明:

```
typedef struct {  
    int x;  
    int y;  
} bmcv_point_t;
```

- `x` 描述了 point 在原图中所在的起始横坐标。自左而右从 0 开始，取值范围  $[0, width)$ 。
- `y` 描述了 point 在原图中所在的起始纵坐标。自上而下从 0 开始，取值范围  $[0, height)$ 。

#### 注意事项:

1. 该接口底图的格式与尺寸限制于 `bmcv_image_vpp_basic` 接口相同。

2. 所有输入 point 对象区域须在图像以内。

**代码示例：**

```
#include "bmvc_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int src_w = 1920, src_h = 1080, dev_id = 0;
    bmvc_point_t center = {128, 128};
    unsigned char r = 0, g = 255, b = 128;
    int length = 2;
    bm_image_format_ext src_fmt = FORMAT_YUV420P;
    bm_handle_t handle = NULL;
    bm_image src;
    char *src_name = "/opt/sophon/libphon-current/bin/res/1920x1080_yuv420.bin";
    ↪;
    char *dst_name = "dst.bin";
    bmvc_point_t coord[9] = {{center.x - 60, center.y - 40}, {center.x, center.y - 40},
    ↪{center.x + 60, center.y - 40}, {center.x - 60, center.y}, {center.x, center.y}, {center.x + 60, center.y},
    ↪{center.x - 60, center.y + 40}, {center.x, center.y + 40}, {center.x + 60, center.y + 40}};
    ↪center.y + 40};

    bm_dev_request(&handle, dev_id);
    bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_
    ↪BYTE, &src, NULL);
    bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    bm_read_bin(src, src_name);

    bmvc_image_draw_point(handle, src, 9, coord, length, r, g, b);

    bm_write_bin(src, dst_name);
    bm_image_destroy(&src);
    bm_dev_free(handle);
    return 0;
}
```

## 5.52 bmvc\_image\_draw\_rectangle

**描述：**

该接口用于在图像上画一个或多个矩形框。

**语法：**

```
1  bm_status_t bmvc_image_draw_rectangle(
2      bm_handle_t handle,
3      bm_image image,
4      int rect_num,
```

(续下页)

(接上页)

```

5     bmcv_rect_t *rects,
6     int      line_width,
7     unsigned char r,
8     unsigned char g,
9     unsigned char b)

```

**参数:**

表 5.33: bmcv\_image\_draw\_rectangle 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image	输入	需要在其上画矩形框的 bm_image 对象。
rect_num	输入	矩形框数量，指 rect_num 指针中所包含的 bmcv_rect_t 对象个数。
*rect	输出	矩形框对象指针，包含矩形起始点和宽高。具体内容参考下面的数据类型说明。
line_width	输入	矩形框线宽。line_width 取值范围: [1, min{crop_w/2, crop_h/2}]
r	输入	矩形框颜色的 r 分量。
g	输入	矩形框颜色的 g 分量。
b	输入	矩形框颜色的 b 分量。

**返回值:**

该函数成功调用时, 返回 BM\_SUCCESS。

**数据类型说明:**

```

1 typedef struct bmcv_rect {
2     int start_x;
3     int start_y;
4     int crop_w;
5     int crop_h;
6 } bmcv_rect_t;

```

- start\_x 描述了 crop 图像在原图中所在的起始横坐标。自左而右从 0 开始, 取值范围 [0, width)。
- start\_y 描述了 crop 图像在原图中所在的起始纵坐标。自上而下从 0 开始, 取值范围 [0, height)。
- crop\_w 描述的 crop 图像的宽度, 也就是对应输出图像的宽度。
- crop\_h 描述的 crop 图像的高度, 也就是对应输出图像的高度。

**格式支持:**

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致，可支持：

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

### 注意：

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
2. 如果 image 为 NV12/NV21/NV16/NV61/YUV420P 格式，则线宽 line\_width 会自动偶数对齐。
3. 如果 rect\_num 为 0，则自动返回成功。
4. 如果 line\_width 小于零，则返回失败。
5. 所有输入矩形对象部分在 image 之外，则只会画出在 image 之内的线条，并返回成功。

### 代码示例：

```

1 #include <limits.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 #include "bmcv_api_ext_c.h"
7
8 int main() {
9     char* filename_src = "path/to/src";
10    char* filename_dst = "path/to/dst";
11    int in_width = 1920;
12    int in_height = 1080;
13    int rect_num = 1;
14    int line_width = 10;
15    bm_image_format_ext src_format = 8;

```

(续下页)

(接上页)

```
16 bmcv_rect_t crop_rect = {  
17     .start_x = 100,  
18     .start_y = 100,  
19     .crop_w = 200,  
20     .crop_h = 200};  
21 unsigned char r = 0;  
22 unsigned char g = 0;  
23 unsigned char b = 0;  
24  
25 bm_status_t ret = BM_SUCCESS;  
26  
27 int src_size = in_height * in_width * 3;  
28 unsigned char *input_data = (unsigned char *)malloc(src_size * sizeof(unsigned char));  
29  
30 FILE *file;  
31 file = fopen(filename_src, "rb");  
32 fread(input_data, sizeof(unsigned char), src_size, file);  
33 fclose(file);  
34  
35 bm_handle_t handle;  
36 int dev_id = 0;  
37 bm_image src;  
38  
39 ret = bm_dev_request(&handle, dev_id);  
40 if (ret != BM_SUCCESS) {  
41     printf("Create bm handle failed. ret = %d\n", ret);  
42     return ret;  
43 }  
44  
45 bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_1N_BYTE, F  
→&src, NULL);  
46 bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);  
47  
48 void *src_in_ptr[3] = { (void *)input_data,  
49                         ((void *)((char *)input_data + in_height * in_width)),  
50                         ((void *)((char *)input_data + 2 * in_height * in_width))};  
51  
52 bm_image_copy_host_to_device(src, (void **)src_in_ptr);  
53 ret = bmcv_image_draw_rectangle(handle, src, rect_num, &crop_rect, line_width, r, g, b);  
54 bm_image_copy_device_to_host(src, (void **)src_in_ptr);  
55  
56 bm_image_destroy(&src);  
57 bm_dev_free(handle);  
58  
59 file = fopen(filename_dst, "wb");  
60 fwrite(input_data, sizeof(unsigned char), src_size, file);  
61 fclose(file);  
62  
63 free(input_data);  
64 return ret;
```

(续下页)

### 5.53 bmcv\_image\_fill\_rectangle

#### 描述:

该接口用于在图像上填充一个或者多个矩形。

#### 语法:

```

1 bm_status_t bmcv_image_fill_rectangle(
2     bm_handle_t handle,
3     bm_image    image,
4     int         rect_num,
5     bmcv_rect_t *rects,
6     unsigned char r,
7     unsigned char g,
8     unsigned char b)

```

#### 参数:

表 5.34: bmcv\_image\_fill\_rectangle 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image	输入	需要在其上画矩形框的 bm_image 对象。
rect_num	输入	矩形框数量，指 rects 指针中所包含的 bmcv_rect_t 对象个数。
*rect	输出	矩形框对象指针，包含矩形起始点和宽高。具体内容参考下面的数据类型说明。
r	输入	矩形框颜色的 r 分量。
g	输入	矩形框颜色的 g 分量。
b	输入	矩形框颜色的 b 分量。

#### 返回值:

该函数成功调用时，返回 BM\_SUCCESS。

#### 数据类型说明:

```

1 typedef struct bmcv_rect {
2     int start_x;
3     int start_y;
4     int crop_w;
5     int crop_h;
6 } bmcv_rect_t;

```

- start\_x 描述了 crop 图像在原图中所在的起始横坐标。自左而右从 0 开始，取值范围 [0, width-1)。
- start\_y 描述了 crop 图像在原图中所在的起始纵坐标。自上而下从 0 开始，取值范围 [0, height-1)。
- crop\_w 描述的 crop 图像的宽度，也就是对应输出图像的宽度。
- crop\_h 描述的 crop 图像的高度，也就是对应输出图像的高度。

#### 格式支持：

##### 1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

##### 2. 输入和输出的色彩格式必须保持一致，可支持：

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

#### 注意：

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
2. 如果 rect\_num 为 0，则自动返回成功。
3. 如果 line\_width 小于零，则返回失败。
4. 所有输入矩形对象部分在 image 之外，则只会画出在 image 之内的线条，并返回成功。

#### 代码示例：

```

1 #include <limits.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 #include "bmvc_api_ext_c.h"

```

(续下页)

(接上页)

```
7 int main() {
8     char* filename_src = "path/to/src";
9     char* filename_dst = "path/to/dst";
10    int in_width = 1920;
11    int in_height = 1080;
12    int rect_num = 1;
13    bm_image_format_ext src_format = 8;
14    bmcv_rect_t crop_rect = {
15        .start_x = 100,
16        .start_y = 100,
17        .crop_w = 100,
18        .crop_h = 100};
19    unsigned char r = 0;
20    unsigned char g = 0;
21    unsigned char b = 0;
22
23    bm_status_t ret = BM_SUCCESS;
24
25    int src_size = in_width * in_height * 3;
26    unsigned char *input_data = (unsigned char *)malloc(src_size);
27
28    FILE *file;
29    file = fopen(filename_src, "rb");
30    fread(input_data, sizeof(unsigned char), src_size, file);
31    fclose(file);
32
33    bm_handle_t handle = NULL;
34    int dev_id = 0;
35    bm_image src;
36
37    ret = bm_dev_request(&handle, dev_id);
38    if (ret != BM_SUCCESS) {
39        printf("Create bm handle failed. ret = %d\n", ret);
40        return ret;
41    }
42
43    bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_1N_BYTE, F
44    ↪&src, NULL);
45    bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
46
47    void *src_in_ptr[3] = { (void *)input_data,
48                           (void *)((char *)input_data + in_height * in_width),
49                           (void *)((char *)input_data + 2 * in_height * in_width)};
50
51    bm_image_copy_host_to_device(src, (void **)src_in_ptr);
52    ret = bmcv_image_fill_rectangle(handle, src, rect_num, &crop_rect, r, g, b);
53    bm_image_copy_device_to_host(src, (void **)src_in_ptr);
54
55    bm_image_destroy(&src);
56    bm_dev_free(handle);
```

(续下页)

(接上页)

```
57  
58  
59 file = fopen(filename_dst, "wb");  
60 fwrite(input_data, sizeof(unsigned char), src_size, file);  
61 fclose(file);  
62  
63  
64 free(input_data);  
65 return ret;  
66 }
```

## 5.54 bmcv\_image\_flip

### 【描述】

该 API 可将输入图像进行水平翻转、垂直翻转或旋转 180 度的处理。

### 【语法】

```
1 bm_status_t bmcv_image_flip(  
2     bm_handle_t      handle,  
3     bm_image         input,  
4     bm_image         output,  
5     bmcv_flip_mode   flip_mode)
```

### 【参数】

表 5.35: bmcv\_image\_flip 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象。
output	输出	输出 bm_image 对象。
flip_mode	输入	flip 模式，可选择水平翻转，垂直翻转和旋转 180 度。

### 【数据类型说明】

```
1 typedef enum {
2     NO_FLIP = 0,
3     HORIZONTAL_FLIP = 1,
4     VERTICAL_FLIP = 2,
5     ROTATE_180 = 3,
6 } bmcv_flip_mode;
```

NO\_FLIP 表示不做处理，HORIZONTAL\_FLIP 表示水平翻转，VERTICAL\_FLIP 表示垂直翻转，ROTATE\_180 表示旋转 180 度。

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

1. 该 API 所需要满足的格式以及部分要求与 bmcv\_image\_vpp\_basic 中的表格相同。
2. 输入输出的宽高 (src.width, src.height, dst.widht, dst.height) 限制在 8 ~ 8192 之间，缩放 128 倍。
3. 输入必须关联 device memory，否则返回失败。

### 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

int readBin(const char * path, unsigned char* input_data, int size) {
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    }
    fclose(fp_src);
    return 0;
}

int writeBin(const char * path, unsigned char* input_data, int size) {
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
```

(续下页)

(接上页)

```
    printf("file size is less than %d required bytes\n", size);
};

fclose(fp_dst);
return 0;
}

int main() {
    int src_h = 1080;
    int src_w = 1920;
    int dst_w = 1920;
    int dst_h = 1080;
    bm_image_format_ext src_fmt = FORMAT_GRAY;
    bm_image_format_ext dst_fmt = 14;
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bmcv_flip_mode flip_mode = HORIZONTAL_FLIP;
    bm_handle_t handle;
    bm_status_t ret = 0;
    bm_image src, dst;
    unsigned char* data_tpu = (unsigned char*)malloc(src_w * src_h * sizeof(unsigned char));
    ret = bm_dev_request(&handle, 0);

    ret = readBin(src_name, data_tpu, src_h * src_w);

    bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src,[F
→NULL);
    bm_image_create(handle, dst_h, dst_w, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst,[F
→NULL);

    ret = bm_image_alloc_dev_mem(src,BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst,BMCV_HEAP1_ID);

    unsigned char* in_ptr[1] = {0};

    in_ptr[0] = data_tpu;

    ret = bm_image_copy_host_to_device(src, (void**)in_ptr);
    bmcv_image_flip(handle, src, dst, flip_mode);
    ret = bm_image_copy_device_to_host(dst, (void**)in_ptr);
    bm_image_destroy(&src);
    bm_image_destroy(&dst);

    ret = writeBin(dst_name, data_tpu, src_w * src_h);

    return ret;
}
```

## 5.55 bmcv\_image\_gaussian\_blur

### 描述:

该接口用于对图像进行高斯滤波操作。

### 语法:

```

1 bm_status_t bmcv_image_gaussian_blur(
2     bm_handle_t handle,
3     bm_image input,
4     bm_image output,
5     int kw,
6     int kh,
7     float sigmaX,
8     float sigmaY = 0);

```

### 参数:

表 5.36: bmcv\_image\_gaussian\_blur 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入图像的 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。
output	输出	输出图像的 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以通过 bm_image_alloc_dev_mem 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。
kw	输入	kernel 宽的大小。
kh	输入	kernel 高的大小。
sigmaX	输入	X 方向上的高斯核标准差，取值范围为 0-4.0。
sigmaY = 0	输入	Y 方向上的高斯核标准差，取值范围为 0-4.0。如果为 0 则表示与 X 方向上的高斯核标准差相同。

### 返回值:

该函数成功调用时，返回 BM\_SUCCESS。

### 格式支持:

该接口目前支持以下图像格式:

num	image_format
1	FORMAT_BGR_PLANAR
2	FORMAT_RGB_PLANAR
3	FORMAT_RGBP_SEPARATE
4	FORMAT_BGRP_SEPARATE
5	FORMAT_GRAY

该接口目前支持的数据格式：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

### 注意事项：

- 在调用该接口之前必须确保输入图像的内存已经申请。
- 输入输出图像的数据格式，图像格式必须相同。
- 目前卷积核支持的大小有 3\*3, 5\*5, 7\*7, 当卷积核大小为 3 时，支持的宽高范围为 8\*8 ~ 4096\*8192，核大小为 5 时支持的宽高范围为 8\*8 ~ 2048\*8192，核大小为 7 时支持的宽高范围为 8\*8 ~ 1500\*8192。

### 代码示例：

```

1 #include <stdio.h>
2 #include "bmvc_api_ext_c.h"
3 #include "stdlib.h"
4 #include "string.h"
5 #include <assert.h>
6 #include <float.h>
7 #include <math.h>
8
9 static void read_bin(const char *input_path, unsigned char *input_data, int width, int height) {
10    FILE *fp_src = fopen(input_path, "rb");
11    if (fp_src == NULL) {
12        printf("无法打开输出文件 %s\n", input_path);
13        return;
14    }
15    if(fread(input_data, sizeof(char), width * height, fp_src) != 0) {
16        printf("read image success\n");
17    }
18    fclose(fp_src);
19}
20
21 static void write_bin(const char *output_path, unsigned char *output_data, int width, int height) {
22    FILE *fp_dst = fopen(output_path, "wb");
23    if (fp_dst == NULL) {

```

(续下页)

(接上页)

```

24     printf("无法打开输出文件 %s\n", output_path);
25     return;
26 }
27 fwrite(output_data, sizeof(char), width * height, fp_dst);
28 fclose(fp_dst);
29 }

30
31 int main() {
32     int width = 1920;
33     int height = 1080;
34     int format = FORMAT_GRAY;
35     float sigmaX = (float)rand() / RAND_MAX * 5.0f;
36     float sigmaY = (float)rand() / RAND_MAX * 5.0f;
37     int ret = 0;
38     char *input_path = "path/to/input";
39     char *output_path = "path/to/output";
40     bm_handle_t handle;
41     ret = bm_dev_request(&handle, 0);
42     if (ret != BM_SUCCESS) {
43         printf("bm_dev_request failed. ret = %d\n", ret);
44         return -1;
45     }

46     int kw = 3, kh = 3;

47
48     unsigned char *input_data = (unsigned char*)malloc(width * height);
49     unsigned char *output_tpu = (unsigned char*)malloc(width * height);

50
51     read_bin(input_path, input_data, width, height);

52
53     bm_image img_i;
54     bm_image img_o;

55
56     bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_
57     →EXT_1N_BYT, &img_i, NULL);
58     bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_
59     →EXT_1N_BYT, &img_o, NULL);
60     bm_image_alloc_dev_mem(img_i, 2);
61     bm_image_alloc_dev_mem(img_o, 2);

62     unsigned char *input_addr[3] = {input_data, input_data + height * width, input_data + 2[F]
63     →* height * width};
64     bm_image_copy_host_to_device(img_i, (void **)(input_addr));

65     ret = bmcv_image_gaussian_blur(handle, img_i, img_o, kw, kh, sigmaX, sigmaY);
66     unsigned char *output_addr[3] = {output_tpu, output_tpu + height * width, output_tpu + [F]
67     →2 * height * width};
68     bm_image_copy_device_to_host(img_o, (void **)(output_addr));

69     bm_image_destroy(&img_i);
70     bm_image_destroy(&img_o);

```

(续下页)

(接上页)

```

71
72
73     write_bin(output_path, output_tpu, width, height);
74     free(input_data);
75     free(output_tpu);
76
77     bm_dev_free(handle);
78     return ret;
79 }
```

## 5.56 bmcv\_image\_jpeg\_dec

### 【描述】

该接口可以实现对多张图片的 JPEG 解码过程。

### 【语法】

```

1  bm_status_t bmcv_image_jpeg_dec(
2      bm_handle_t handle,
3      void * p_jpeg_data[],
4      size_t * in_size,
5      int image_num,
6      bm_image * dst
7 );
```

### 【参数】

表 5.37: bmcv\_image\_jpeg\_dec 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*p_jpeg_data[]	输入	待解码的图片数据指针，由于该接口支持对多张图片的解码，因此为指针数组。
*in_size	输入	待解码各张图片的大小（以 byte 为单位）存放在该指针中，也就是上述 p_jpeg_data 每一维指针所指向空间的大小。
image_num	输入	输入图片数量，最多支持 4。
*dst	输出	输出 bm_image 的指针。每个 dst bm_image 用户可以选择自行调用 bm_image_create 创建，也可以选择不创建。如果用户只声明而不创建则由接口内部根据待解码图片信息自动创建，默认的 format 如表 default_format 所示，当不再需要时仍然需要用户调用 bm_image_destory 来销毁。

表 5.38: default\_format

码流	默认输出 format
YUV420	FORMAT_YUV420P
YUV422	FORMAT_YUV422P
YUV444	FORMAT_YUV444P
YUV400	FORMAT_GRAY

**【返回值】**

该函数成功调用时, 返回 BM\_SUCCESS。

**【注意】**

1. 如果用户没有使用 bmcv\_image\_create 创建 dst 的 bm\_image, 那么需要将参数传入指针所指向的空间置 0。
2. 目前解码支持的图片格式及其输出格式对应如下, 如果用户需要指定以下某一种输出格式, 可通过使用 bmcv\_image\_create 自行创建 dst bm\_image, 从而实现将图片解码到以下对应的某一格式。

码流	输出 format
YUV420	FORMAT_YUV420P
	FORMAT_NV12
	FORMAT_NV21
YUV422	FORMAT_YUV422P
	FORMAT_NV16
	FORMAT_NV61
YUV444	FORMAT_YUV444P
YUV400	FORMAT_GRAY

**【代码示例】**

```

1 #include <stdio.h>
2 #include <stdint.h>
3 #include <stdlib.h>
4 #include <memory.h>
5 #include "bmcv_api_ext_c.h"
6 #include <assert.h>
7 #include <math.h>
8
9 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
10
11 int main() {
12     int dev_id = -1;
13     int dev_cnt;
14     int ret = 0;
15     bm_dev_getcount(&dev_cnt);

```

(续下页)

(接上页)

```

16     if (dev_id >= dev_cnt) {
17         printf("[TEST JPEG] dev_id should less than device count, only detect %d devices\n", dev_
18             →cnt);
19         exit(-1);
20     }
21     printf("device count = %d\n", dev_cnt);
22     bm_handle_t handle[dev_cnt];
23
24     for (int i = 0; i < dev_cnt; i++) {
25         int id;
26         if (dev_id != -1) {
27             dev_cnt = 1;
28             id = dev_id;
29         } else {
30             id = i;
31         }
32         bm_status_t req = bm_dev_request(handle + i, id);
33         if (req != BM_SUCCESS) {
34             printf("create bm handle for dev%d failed!\n", id);
35             exit(-1);
36         }
37     }
38     for (int j = 0; j < dev_cnt; j++) {
39         char *dst_name = "path/to/dst";
40         char *src_name = "path/to/src";
41         int format = FORMAT_YUV420P;
42         int image_n = 1;
43         int image_h = 1080;
44         int image_w = 1920;
45         int* stride = (int*)malloc(3 * sizeof(int));
46         stride[0] = align_up(image_w, 16);
47         stride[1] = align_up(image_w / 2, 16);
48         stride[2] = align_up(image_w / 2, 16);
49
50         bm_image src[4];
51         for (int i = 0; i < image_n; i++) {
52             bm_image_create(handle[j], image_h, image_w, (bm_image_format_ext)format, F
53             →DATA_TYPE_EXT_1N_BYTE, src + i, stride);
54             bm_image_alloc_dev_mem(src[i], BMCV_HEAP1_ID);
55         }
56         int image_byte_size[4] = {0};
57         bm_image_get_byte_size(src[0], image_byte_size);
58         for (int i = 0; i < 4; i++) {
59             printf("src_%d_byte_size: %d\n", i, image_byte_size[i]);
60         }
61         int byte_size = image_w * image_h * 3 / 2;
62         unsigned char *input_data = (unsigned char *)malloc(byte_size);
63         FILE *fp_src = fopen(src_name, "rb");
64         if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
65             printf("file size is less than required bytes%d\n", byte_size);

```

(续下页)

(接上页)

```

65     };
66     fclose(fp_src);
67     void* in_ptr[3] = { (void *)input_data, (void *)((unsigned char*)input_data + image_w * F
68 → image_h), (void *)((unsigned char*)input_data + 5 / 4 * image_w * image_h)};
69     bm_image_copy_host_to_device(src[0], in_ptr);
70     void* jpeg_data[4] = {NULL, NULL, NULL, NULL};
71     size_t* size = (size_t*)malloc(image_n * sizeof(size_t));
72     ret = bmcv_image_jpeg_enc(handle[j], image_n, src, jpeg_data, size, 95);
73     if (ret != BM_SUCCESS) exit(-1);

74     // test decode
75     bm_image dst[4];
76     // get dst format
77     if (true) { // create_dst_image
78         int dst_format = FORMAT_NV12;
79         stride[0] = align_up(image_w, 16);
80         stride[1] = align_up(image_w, 16);
81         for (int i = 0; i < image_n; i++) {
82             bm_image_create(handle[j], image_h, image_w, (bm_image_format_ext)dst_format,
83 → DATA_TYPE_EXT_1N_BYTE, dst + i, stride);
84         }
85     }
86     ret = bmcv_image_jpeg_dec(handle[j], (void**)jpeg_data, size, image_n, dst);
87     assert(ret == BM_SUCCESS);

88     bm_image_get_byte_size(dst[0], image_byte_size);
89     for (int i = 0; i < 4; i++) {
90         printf("dst_bytesize[%d]: %d\n", i, image_byte_size[i]);
91     }
92     byte_size = image_h * image_w * 3;
93     uint8_t* output_ptr = (uint8_t*)malloc(byte_size * image_n);
94     for (int i = 0; i < image_n; i++) {
95         void* out_ptr[3] = { (void *)output_ptr, (void *)((uint8_t*)output_ptr + image_w * F
96 → image_h), (void *)((uint8_t*)output_ptr + 5 / 4 * image_w * image_h)};
97         bm_image_copy_device_to_host(dst[i], out_ptr);
98     }

99     FILE *fp_dst = fopen(dst_name, "wb");
100    if (fwrite((void *)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
101        printf("file size is less than %d required bytes\n", byte_size);
102    };
103    fclose(fp_dst);

104    for (int i = 0; i < image_n; i++) {
105        bm_image_destroy(&dst[i]);
106    }

107    for (int i = 0; i < image_n; i++) {
108        free(jpeg_data[i]);
109        bm_image_destroy(&src[i]);
110    }
111
112

```

(续下页)

(接上页)

```
113     }
114     free(input_data);
115     free(output_ptr);
116     free(size);
117     free(stride);
118 }
119 for (int i = 0; i < dev_cnt; i++) {
120     bm_dev_free(handle[i]);
121 }
122 return ret;
123 }
```

## 5.57 bmcv\_image\_jpeg\_enc

### 【描述】

该接口可以实现对多张 bm\_image 的 JPEG 编码过程。

### 【语法】

```
1 bm_status_t bmcv_image_jpeg_enc(
2     bm_handle_t handle,
3     int         image_num,
4     bm_image *  src,
5     void *      p_jpeg_data[],
6     size_t *    out_size,
7     int         quality_factor = 85
8 );
```

### 【参数】

表 5.39: bmcv\_image\_jpeg\_enc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image_num	输入	输入图片数量，最多支持 4。
*src	输入	输入 bm_image 的指针。每个 bm_image 需要外部调用 bmcv_image_create 创建，image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。
*p_jpeg_data[]	输出	编码后图片的数据指针，由于该接口支持对多张图片的编码，因此为指针数组，数组的大小即为 image_num。用户可以选择不为其申请空间（即数组每个元素均为 NULL），在 api 内部会根据编码后数据的大小自动分配空间，但当不再使用时需要用户手动释放该空间。当然用户也可以选择自己申请足够的空间。
*out_size	输入	完成编码后各张图片的大小（以 byte 为单位）存放在该指针中。
quality_factor	输入	编码后图片的质量因子。取值 0 ~ 100 之间，值越大表示图片质量越高，但数据量也就越大，反之值越小图片质量越低，数据量也就越少。该参数为可选参数，默认值为 85。

**【返回值】**

该函数成功调用时，返回 BM\_SUCCESS。

**备注:**

目前编码支持的图片格式包括以下几种：

FORMAT\_YUV420P  
 FORMAT\_YUV422P  
 FORMAT\_YUV444P  
 FORMAT\_NV12  
 FORMAT\_NV21  
 FORMAT\_NV16  
 FORMAT\_NV61  
 FORMAT\_GRAY

目前编码支持的数据格式如下：

## DATA\_TYPE\_EXT\_1N\_BYTE

## 【代码示例】

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include <stdlib.h>
4 #include <memory.h>
5 #include "bmvc_api_ext_c.h"
6 #include <assert.h>
7 #include <math.h>
8
9 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
10
11 int main() {
12     int dev_id = -1;
13     int dev_cnt;
14     int ret = 0;
15     bm_dev_getcount(&dev_cnt);
16     if (dev_id >= dev_cnt) {
17         printf("[TEST JPEG] dev_id should less than device count, only detect %d devices\n", dev_
18             ↪cnt);
19         exit(-1);
20     }
21     printf("device count = %d\n", dev_cnt);
22     bm_handle_t handle[dev_cnt];
23
24     for (int i = 0; i < dev_cnt; i++) {
25         int id;
26         if (dev_id != -1) {
27             dev_cnt = 1;
28             id = dev_id;
29         } else {
30             id = i;
31         }
32         bm_status_t req = bm_dev_request(handle + i, id);
33         if (req != BM_SUCCESS) {
34             printf("create bm handle for dev%d failed!\n", id);
35             exit(-1);
36         }
37     }
38     for (int j = 0; j < dev_cnt; j++) {
39         char *src_name = "path/to/src";
40         int format = FORMAT_YUV420P;
41         int image_n = 1;
42         int image_h = 1080;
43         int image_w = 1920;
44         int* stride = (int*)malloc(3 * sizeof(int));
45         stride[0] = align_up(image_w, 16);
46         stride[1] = align_up(image_w / 2, 16);
47         stride[2] = align_up(image_w / 2, 16);
```

(续下页)

(接上页)

```
47
48
49     bm_image src[4];
50     for (int i = 0; i < image_n; i++) {
51         bm_image_create(handle[j], image_h, image_w, (bm_image_format_ext)format,F
52         →DATA_TYPE_EXT_1N_BYTE, src + i, stride);
53         bm_image_alloc_dev_mem(src[i], BMCV_HEAP1_ID);
54     }
55     int image_byte_size[4] = {0};
56     bm_image_get_byte_size(src[0], image_byte_size);
57     for (int i = 0; i < 4; i++) {
58         printf("src %d byte size: %d\n", i, image_byte_size[i]);
59     }
60     int byte_size = image_w * image_h * 3 / 2;
61     unsigned char *input_data = (unsigned char *)malloc(byte_size);
62     FILE *fp_src = fopen(src_name, "rb");
63     if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
64         printf("file size is less than required bytes%d\n", byte_size);
65     };
66     fclose(fp_src);
67     void* in_ptr[4] = {(void *)input_data,
68                         (void *)((unsigned char*)input_data + image_byte_size[0]),
69                         (void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
70             →size[1]),
71                         (void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_
72             →size[1] + image_byte_size[2])};
73     bm_image_copy_host_to_device(src[0], in_ptr);
74     void* jpeg_data[4] = {NULL, NULL, NULL, NULL};
75     size_t* size = (size_t*)malloc(image_n * sizeof(size_t));
76     ret = bmcv_image_jpeg_enc(handle[j], image_n, src, jpeg_data, size, 95);
77     if (ret != BM_SUCCESS) exit(-1);
78
79     for (int i = 0; i < image_n; i++) {
80         free(jpeg_data[i]);
81         bm_image_destroy(&src[i]);
82     }
83     free(input_data);
84     free(size);
85     free(stride);
86 }
87 return ret;
88 }
```

## 5.58 bmcv\_image\_laplacian

梯度计算 laplacian 算子。

**接口形式：**

```
bm_status_t bmcv_image_laplacian(
    bm_handle_t handle,
    bm_image input,
    bm_image output,
    unsigned int ksize);
```

**参数说明：**

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_image input`  
输入参数。输入图像的 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。
- `bm_image output`  
输出参数。输出 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。
- `int ksize = 1 / 3`  
Laplacian 核的大小, 必须是 1 或 3。

**返回值说明：**

- `BM_SUCCESS`: 成功
- 其他: 失败

**格式支持：**

1. 该接口目前支持以下 `image_format`:

num	input image_format	output image_format
1	FORMAT_GRAY	FORMAT_GRAY

2. 目前支持以下 `data_type`:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

**注意事项：**

1. 在调用该接口之前必须确保输入的 image 内存已经申请。
2. input output 的 data\_type 必须相同。
3. 目前支持图像宽高范围为 2x2-4096x4096。

**示例代码**

```
#include "bmvc_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    }

    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    }

    fclose(fp_dst);
}

int main()
{
    int height = 1080;
    int width = 1920;
    unsigned int ksize = 3;
    bm_image_format_ext fmt = FORMAT_GRAY; /* 14 */
    int ret = 0;
    char* src_name = "path/to/src";
    char* dst_name = "path/to/dst";
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);

    unsigned char* input = (unsigned char*)malloc(width * height * sizeof(unsigned char));
    unsigned char* tpu_out = (unsigned char*)malloc(width * height * sizeof(unsigned char));
}
```

(续下页)

(接上页)

```

unsigned char* cpu_out = (unsigned char*)malloc(width * height *F
↳sizeof(unsigned char));
int len;

len = width * height;
readBin(src_name, input, len);
memset(tpu_out, 0, len * sizeof(unsigned char));

bm_image_data_format_ext data_type = DATA_TYPE_EXT_1N_BYTE;
bm_image input_img, output_img;

ret = bm_image_create(handle, height, width, fmt, data_type, &input_img,F
↳NULL);
ret = bm_image_create(handle, height, width, fmt, data_type, &output_img,F
↳NULL);
ret = bm_image_alloc_dev_mem(input_img, 2);
ret = bm_image_alloc_dev_mem(output_img, 2);

ret = bm_image_copy_host_to_device(input_img, (void **)(&input));
ret = bmcv_image_laplacian(handle, input_img, output_img, kszie);
ret = bm_image_copy_device_to_host(output_img, (void **)(&tpu_out));

bm_image_destroy(&input_img);
bm_image_destroy(&output_img);

writeBin(dst_name, tpu_out, len);
free(input);
free(tpu_out);
free(cpu_out);

bm_dev_free(handle);
return ret;
}

```

## 5.59 bmcv\_image\_mosaic

### 描述:

该接口用于在图像上打一个或多个马赛克。

### 语法:

```

1 bm_status_t bmcv_image_mosaic(
2     bm_handle_t handle,
3     int mosaic_num,
4     bm_image input,

```

(续下页)

(接上页)

```

5     bmcv_rect_t *mosaic_rect,
6     int is_expand)

```

**参数：**

表 5.40: bmcv\_image\_mosaic 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
mosaic_num	输入	马赛克数量，指 mosaic_rect 指针中所包含的 bmcv_rect_t 对象个数。
input	输入	需要打马赛克的 bm_image 对象。
*mosaic_rect	输入	马赛克对象指针，包含每个马赛克起始点和宽高。具体内容参考下面的数据类型说明。
is_expand	输入	是否扩列。值为 0 时表示不扩列，值为 1 时表示在原马赛克周围扩列一个宏块 (8 个像素)。

**数据类型说明：**

```

1 typedef struct bmcv_rect {
2     int start_x;
3     int start_y;
4     int crop_w;
5     int crop_h;
6 } bmcv_rect_t;

```

- start\_x 描述了马赛克在原图中所在的起始横坐标。自左而右从 0 开始，取值范围 [0, width)。
- start\_y 描述了马赛克在原图中所在的起始纵坐标。自上而下从 0 开始，取值范围 [0, height)。
- crop\_w 描述的马赛克的宽度。
- crop\_h 描述的马赛克的高度。

**返回值：**

该函数成功调用时，返回 BM\_SUCCESS。

**格式支持：**

1. 输入和输出的数据类型

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式

num	image_format
1	FORMAT_YUV420P
2	FORMAT_YUV444P
3	FORMAT_NV12
4	FORMAT_NV21
5	FORMAT_RGB_PLANAR
6	FORMAT_BGR_PLANAR
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_RGBP_SEPARATE
10	FORMAT_BGRP_SEPARATE
11	FORMAT_GRAY

**注意：**

1. 输入输出所有 bm\_image 结构必须提前创建，否则返回失败。
2. 如果马赛克宽高非 8 对齐，则会自动向上 8 对齐，若在边缘区域，则 8 对齐时会往非边缘方向延展。
3. 如果马赛克区域超出原图宽高，超出部分会自动贴到原图边缘。
4. 仅支持 8x8 以上的马赛克尺寸。

**示例代码**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

void readBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_dst);
}
```

(续下页)

(接上页)

```
}

int main() {
    int src_h = 1080, src_w = 1920, dev_id = 0;
    bm_image_format_ext src_fmt = FORMAT_RGB_PACKED;
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";

    bmcv_rect_t rect = {.start_x = 100, .start_y = 100, .crop_w = 500, .crop_h = F
    ↪500};
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);

    unsigned char* input_data = malloc(src_h * src_w * 3);
    unsigned char* output_tpu = malloc(src_h * src_w * 3);
    readBin(src_name, input_data, src_h * src_w * 3);
    memset(output_tpu, 0, src_h * src_w * 3);

    // convert_ctx ctx = *(convert_ctx*)arg;
    bm_image src;
    bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_
    ↪BYTE, &src, NULL);
    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    unsigned char *in1_ptr[1] = {input_data};
    bm_image_copy_host_to_device(src, (void **)(in1_ptr));
    bmcv_image_mosaic(handle, 1, src, &rect, false);

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2]F
    ↪+ image_byte_size[3];
    unsigned char* output_ptr = (unsigned char*)malloc(byte_size);
    void* out_ptr[4] = {(void*)output_ptr,
                        ((void*)((unsigned char*)output_ptr + image_byte_size[0])),
                        ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_
    ↪byte_size[1])),
                        ((void*)((unsigned char*)output_ptr + image_byte_size[0] + image_
    ↪byte_size[1] + image_byte_size[2]))};
    bm_image_copy_device_to_host(src, (void **)out_ptr);
    writeBin(dst_name, output_ptr, src_h * src_w * 3);

    free(input_data);
    free(output_ptr);
    bm_image_destroy(&src);
    return ret;
}
```

## 5.60 bmcv\_image\_put\_text

可以实现在一张图像上写字的功能，并支持指定字的颜色、大小和宽度。

接口形式：

```
typedef struct {
    int x;
    int y;
} bmcv_point_t;

typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} bmcv_color_t;

bm_status_t bmcv_image_put_text(
    bm_handle_t handle,
    bm_image image,
    const char* text,
    bmcv_point_t org,
    bmcv_color_t color,
    float fontScale,
    int thickness);
```

参数说明：

- bm\_handle\_t handle  
输入参数。bm\_handle 句柄。
- bm\_image image  
输入/输出参数。需处理图像的 bm\_image，bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存，或者使用 bmcv\_image\_attach 来 attach 已有的内存。
- const char\* text  
输入参数。待写入的文本内容。当文本内容中有中文时，thickness 请设置为 0。
- bmcv\_point\_t org  
输入参数。第一个字符左下角的坐标位置。图像左上角为原点，向右延伸为 x 方向，向下延伸为 y 方向。
- bmcv\_color\_t color  
输入参数。画线的颜色，分别为 RGB 三个通道的值。
- float fontScale  
输入参数。字体大小。

- int thickness

输入参数。画线的宽度，对于 YUV 格式的图像建议设置为偶数。开启中文字库请将该参数设置为 0。

#### 返回值说明：

- BM\_SUCCESS: 成功
- 其他: 失败

#### 格式支持：

1. 该接口目前支持以下 image\_format:

num	image_format
1	FORMAT_GRAY
2	FORMAT_YUV420P
3	FORMAT_YUV422P
4	FORMAT_YUV444P
5	FORMAT_NV12
6	FORMAT_NV21
7	FORMAT_NV16
8	FORMAT_NV61

thickness 参数配置为 0，即开启中文字库后，image\_format 扩展支持 bmcv\_image\_overlay API 底图支持的格式。

2. 目前支持以下 data\_type:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

3. 若文字内容不变，推荐使用 bmcv\_gen\_text\_watermark 与 bmcv\_image\_overlay 搭配的文字绘制方式，文字生成水印图，重复使用水印图进行 osd 叠加，以提高处理效率。示例参见 bmcv\_image\_overlay 接口文档。

#### 示例代码

```
#include "bmcv_api_ext_c.h"
#include <stdio.h>
#include <stdlib.h>

#define IMAGE_CHN_NUM_MAX 3
#define __ALIGN_MASK(x, mask) (((x) + (mask)) & ~(mask))
#define ALIGN(x, a) __ALIGN_MASK(x, (__typeof__(x))(a)-1)

static bmcv_point_t org = {500, 500};
static int fontScale = 5;
```

(续下页)

(接上页)

```
static const char text[30] = "Hello, world!";
static unsigned char color[3] = {255, 0, 0};
static int thickness = 2;

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    len = fwrite((void*)output_data, 1, size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_dst);
    return 0;
}

static int readBin(const char* path, void* input_data)
{
    int len;
    int size;
    FILE* fp_src = fopen(path, "rb");

    if (fp_src == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    fseek(fp_src, 0, SEEK_END);
    size = ftell(fp_src);
    fseek(fp_src, 0, SEEK_SET);

    len = fread((void*)input_data, 1, size, fp_src);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_src);
    return 0;
}

int main()
```

(续下页)

(接上页)

```
{  
    int width = 1920;  
    int height = 1080;  
    int format = FORMAT_YUV420P;  
    int ret = 0;  
    char* input_path = "path/to/input";  
    char* output_path = "path/to/output";  
    int i;  
    bm_handle_t handle;  
    ret = bm_dev_request(&handle, 0);  
  
    int offset_list[IMAGE_CHN_NUM_MAX] = {0};  
    offset_list[0] = width * height;  
    offset_list[1] = ALIGN(width, 2) * ALIGN(height, 2) >> 2;  
    offset_list[2] = ALIGN(width, 2) * ALIGN(height, 2) >> 2;  
  
    int total_size = 0;  
    unsigned char* data_tpu = (unsigned char*)malloc(width * height * IMAGE_  
    ↵CHN_NUM_MAX * sizeof(unsigned char));  
  
    ret = readBin(input_path, data_tpu);  
  
    bm_image input_img;  
    unsigned char* in_ptr[IMAGE_CHN_NUM_MAX] = {0};  
    bmcv_color_t rgb = {color[0], color[1], color[2]};  
  
    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format, F  
    ↵DATA_TYPE_EXT_1N_BYTE, &input_img, NULL);  
    ret = bm_image_alloc_dev_mem(input_img, BMCV_HEAP1_ID);  
  
    in_ptr[0] = data_tpu;  
    in_ptr[1] = data_tpu + offset_list[0];  
    in_ptr[2] = data_tpu + offset_list[0] + offset_list[1];  
  
    ret = bm_image_copy_host_to_device(input_img, (void**)in_ptr);  
    ret = bmcv_image_put_text(handle, input_img, text, org, rgb, fontScale, F  
    ↵thickness);  
    ret = bm_image_copy_device_to_host(input_img, (void**)in_ptr);  
  
    bm_image_destroy(&input_img);  
  
    for (i = 0; i < IMAGE_CHN_NUM_MAX; ++i) {  
        total_size += offset_list[i];  
    }  
    ret = writeBin(output_path, data_tpu, total_size);  
  
    free(data_tpu);  
    bm_dev_free(handle);  
    return ret;  
}
```

## 5.61 bmcv\_image\_pyramid\_down

该接口实现图像高斯金字塔操作中的向下采样。

**接口形式：**

```
bm_status_t bmcv_image_pyramid_down(
    bm_handle_t handle,
    bm_image input,
    bm_image output);
```

**参数说明：**

- `bm_handle_t handle`  
输入参数。`bm_handle` 句柄。
- `bm_image input`  
输入参数。输入图像 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`bm_image` 的内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。
- `bm_image output`  
输出参数。输出图像 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`bm_image` 的内存可以使用 `bm_image_alloc_dev_mem` 或者 `bm_image_copy_host_to_device` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。

**返回值说明：**

- `BM_SUCCESS`: 成功
- 其他: 失败

**格式支持：**

该接口目前支持以下 `image_format` 与 `data_type`:

num	image_format	data_type
1	FORMAT_GRAY	DATA_TYPE_EXT_1N_BYTE

**注意事项：**

1. 目前支持图像的最小宽为 3, 最小高为 3, 最大宽为 2043, 最大高为 4096。

**示例代码**

```
#include <assert.h>
#include "bmcv_api_ext_c.h"
#include <math.h>
```

(续下页)

(接上页)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    };

    fclose(fp_dst);
}

int main()
{
    int height = 1080;
    int width = 1920;
    int ret = 0;
    char* src_name = "path/to/src";
    char* dst_name = "path/to/dst";
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);

    int ow = width / 2;
    int oh = height / 2;
    int channel = 1;
    unsigned char* input_data = (unsigned char*)malloc(width * height * channel * [F
    ↵sizeof(unsigned char));
    unsigned char* output_tpu = (unsigned char*)malloc(ow * oh * channel * [F
    ↵sizeof(unsigned char));
    unsigned char* output_cpu = (unsigned char*)malloc(ow * oh * channel * [F
    ↵sizeof(unsigned char));
    readBin(src_name, input_data, width * height * channel);
    memset(output_tpu, 0, ow * oh * channel * sizeof(unsigned char));

    bm_image_format_ext fmt = FORMAT_GRAY;
    bm_image img_i;
    bm_image img_o;

    ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_

```

(续下页)

(接上页)

```

→BYTE, &img_i, NULL);
    ret = bm_image_create(handle, oh, ow, fmt, DATA_TYPE_EXT_1N_BYTEx,
    ↪img_o, NULL);

    ret = bm_image_alloc_dev_mem(img_i, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(img_o, BMCV_HEAP1_ID);

    ret = bm_image_copy_host_to_device(img_i, (void **)(&input_data));
    ret = bmcv_image_pyramid_down(handle, img_i, img_o);
    ret = bm_image_copy_device_to_host(img_o, (void **)(&output_tpu));

    bm_image_destroy(&img_i);
    bm_image_destroy(&img_o);

    writeBin(dst_name, output_tpu, ow * oh * channel);

    free(input_data);
    free(output_tpu);
    free(output_cpu);

    bm_dev_free(handle);
    return ret;
}

```

## 5.62 bmcv\_image\_quantify

将 float 类型数据转化成 int 类型 (舍入模式为小数点后直接截断), 并将小于 0 的数变为 0, 大于 255 的数变为 255。

**接口形式:**

```

bm_status_t bmcv_image_quantify(
    bm_handle_t handle,
    bm_image input,
    bm_image output);

```

**参数说明:**

- **bm\_handle\_t handle**  
输入参数。bm\_handle 句柄。
- **bm\_image input**  
输入参数。输入图像的 bm\_image, bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者 bm\_image\_copy\_host\_to\_device 来开辟新的内存, 或者使用 bmcv\_image\_attach 来 attach 已有的内存。
- **bm\_image output**

输出参数。输出 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存, 或者使用 `bmcv_image_attach` 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 格式支持:

该接口目前支持以下 `image_format`:

num	input image_format	output image_format
1	FORMAT_YUV444P	FORMAT_YUV444P
2	FORMAT_RGB_PLANAR	FORMAT_RGB_PLANAR
3	FORMAT_BGR_PLANAR	FORMAT_BGR_PLANAR
4	FORMAT_RGB_PACKED	FORMAT_RGB_PACKED
5	FORMAT_BGR_PACKED	FORMAT_BGR_PACKED
6	FORMAT_RGBP_SEPARATE	FORMAT_RGBP_SEPARATE
7	FORMAT_BGRP_SEPARATE	FORMAT_BGRP_SEPARATE
8	FORMAT_GRAY	FORMAT_GRAY

输入数据目前支持以下 `data_type`:

num	data_type
1	DATA_TYPE_EXT_FLOAT32

输出数据目前支持以下 `data_type`:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

#### 注意事项:

1. 在调用该接口之前必须确保输入的 `image` 内存已经申请。
2. 如调用该接口的程序为多线程程序, 需要在创建 `bm_image` 前和销毁 `bm_image` 后加线程锁。
3. 该接口支持图像宽高范围为  $1 \times 1 \sim 8192 \times 8192$ 。
4. 该接口可通过设置环境变量启用双核计算, 运行程序前: `export TPU_CORES=2` 或 `export TPU_CORES=both` 即可。

#### 代码示例:

```
#include <stdio.h>
#include "bmcv_api_ext_c.h"
#include "stdlib.h"
#include "string.h"
#include <assert.h>
#include <float.h>

static void read_bin(const char *input_path, float *input_data, int width, int height)
{
    FILE *fp_src = fopen(input_path, "rb");
    if (fp_src == NULL)
    {
        printf("无法打开输出文件 %s\n", input_path);
        return;
    }
    if(fread(input_data, sizeof(float), width * height, fp_src) != 0)
        printf("read image success\n");
    fclose(fp_src);
}

static void write_bin(const char *output_path, unsigned char *output_data, int[F]
width, int height)
{
    FILE *fp_dst = fopen(output_path, "wb");
    if (fp_dst == NULL)
    {
        printf("无法打开输出文件 %s\n", output_path);
        return;
    }
    fwrite(output_data, sizeof(int), width * height, fp_dst);
    fclose(fp_dst);
}

int main()
{
    int height = 1080;
    int width = 1920;
    char *input_path = "path/to/input";
    char *output_path = "path/to/output";
    bm_handle_t handle;
    bm_status_t ret = bm_dev_request(&handle, 0);

    float* input_data = (float*)malloc(width * height * 3 * sizeof(float));
    unsigned char* output_tpu = (unsigned char*)malloc(width * height * 3 * [F]
sizeof(unsigned char));
    read_bin(input_path, input_data, width, height);

    bm_image input_img;
    bm_image output_img;
    bm_image_create(handle, height, width, (bm_image_format_ext)FORMAT_
RGB_PLANAR, DATA_TYPE_EXT_FLOAT32, &input_img, NULL);
```

(续下页)

(接上页)

```
bm_image_create(handle, height, width, (bm_image_format_ext)FORMAT_
    ↵RGB_PLANAR, DATA_TYPE_EXT_1N_BYTE, &output_img, NULL);
bm_image_alloc_dev_mem(input_img, 1);
bm_image_alloc_dev_mem(output_img, 1);
float* in_ptr[1] = {input_data};

bm_image_copy_host_to_device(input_img, (void **)in_ptr);
bmcv_image_quantify(handle, input_img, output_img);
unsigned char* out_ptr[1] = {output_tpu};

bm_image_copy_device_to_host(output_img, (void **)out_ptr);
bm_image_destroy(&input_img);
bm_image_destroy(&output_img);

write_bin(output_path, output_tpu, width, height);
free(input_data);
free(output_tpu);
bm_dev_free(handle);
return ret;
}
```

### 5.63 bmcv\_image\_storage\_convert

#### 【描述】

该接口将源图像格式的数据转换为目的图像的格式数据，并填充在目的图像关联的 device memory 中。

#### 【语法】

```
1 bm_status_t bmcv_image_storage_convert(
2     bm_handle_t handle,
3     int image_num,
4     bm_image* input_image,
5     bm_image* output_image
6 );
```

#### 【参数】

表 5.41: bmcv\_image\_storage\_convert 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image_num	输入	输入/输出 image 数量。
*input	输入	输入 bm_image 对象指针。
* output	输出	输出 bm_image 对象指针。

**【返回值】**

该函数成功调用时，返回 BM\_SUCCESS。

**【注意】**

该接口的参数数据类型与注意事项与 bmcv\_image\_vpp\_basic 接口相同。

## 5.64 bmcv\_image\_threshold

**描述：**

该接口用于对图像进行像素阈值化操作。

**语法：**

```

1  bm_status_t bmcv_image_threshold(
2      bm_handle_t handle,
3      bm_image input,
4      bm_image output,
5      unsigned char thresh,
6      unsigned char max_value,
7      bm_thresh_type_t type);

```

其中 thresh 类型如下：

```

1  typedef enum {
2      BM_THRESH_BINARY = 0,
3      BM_THRESH_BINARY_INV,
4      BM_THRESH_TRUNC,
5      BM_THRESH_TOZERO,
6      BM_THRESH_TOZERO_INV,
7      BM_THRESH_TYPE_MAX
8  } bm_thresh_type_t;

```

各个类型对应的具体公式如下：

Enumerator	
THRESH_BINARY	$dst(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$dst(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$

参数：

表 5.42: bmcv\_image\_threshold 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入参数。输入图像的 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。
output	输出	输出参数。输出 bm_image，bm_image 需要外部调用 bmcv_image_create 创建。image 内存可以通过 bm_image_alloc_dev_mem 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配。
thresh	输入	像素阈值，取值范围为 0-255。
max_value	输入	阈值化操作后的像素最大值，取值范围为 0-255。
type	输入	阈值化类型，取值范围为 0-4。

返回值：

该函数成功调用时，返回 BM\_SUCCESS。

格式支持：

该接口目前支持以下图像格式：

num	image_format
1	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY
8	FORMAT_YUV420P
9	FORMAT_YUV422P
10	FORMAT_YUV444P
11	FORMAT_NV12
12	FORMAT_NV21
13	FORMAT_NV16
14	FORMAT_NV61
15	FORMAT_NV24

该接口目前支持的数据格式：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

### 注意事项：

- 在调用该接口之前必须确保输入图像的内存已经申请。
- 输入输出图像的数据格式，图像格式必须相同。
- 目前支持的图像最大宽和高为 4096。

### 代码示例：

```

1 #include <stdio.h>
2 #include "bmvc_api_ext_c.h"
3 #include "test_misc.h"
4 #include "stdlib.h"
5 #include "string.h"
6 #include <assert.h>
7 #include <float.h>

8 static void read_bin(const char *input_path, unsigned char *input_data, int width, int height)
9 {
10     FILE *fp_src = fopen(input_path, "rb");
11     if (fp_src == NULL)
12     {
13         printf("无法打开输出文件 %s\n", input_path);
14         return;
15     }

```

(续下页)

(接上页)

```

16     }
17     if(fread(input_data, sizeof(char), width * height, fp_src) != 0)
18         printf("read image success\n");
19     fclose(fp_src);
20 }
21
22 static void write_bin(const char *output_path, unsigned char *output_data, int width, int height)
23 {
24     FILE *fp_dst = fopen(output_path, "wb");
25     if (fp_dst == NULL)
26     {
27         printf("无法打开输出文件 %s\n", output_path);
28         return;
29     }
30     fwrite(output_data, sizeof(char), width * height, fp_dst);
31     fclose(fp_dst);
32 }
33
34 int main() {
35     int height = 1080;
36     int width = 1920;
37     int type = rand() % 5;
38     char *input_path = "path/to/input";
39     char *output_path = "path/to/output";
40     bm_handle_t handle;
41     bm_status_t ret = bm_dev_request(&handle, 0);
42     if (ret != BM_SUCCESS) {
43         printf("Create bm handle failed. ret = %d\n", ret);
44         return -1;
45     }
46     int channel = 1;
47
48     unsigned char threshold = 50;
49     unsigned char max_value = 228;
50     printf("type: %d\n", type);
51     printf("threshold: %d , max_value: %d\n", threshold, max_value);
52     printf("width: %d , height: %d , channel: %d \n", width, height, channel);
53
54     unsigned char* input_data = (unsigned char*)malloc(width * height);
55     unsigned char* output_tpu = (unsigned char*)malloc(width * height);
56     read_bin(input_path, input_data, width, height);
57
58     bm_image input_img;
59     bm_image output_img;
60
61     bm_image_create(handle, height, width, (bm_image_format_ext)FORMAT_GRAY, DATA_
62     ↪TYPE_EXT_1N_BYTE, &input_img, NULL);
63     bm_image_create(handle, height, width, (bm_image_format_ext)FORMAT_GRAY, DATA_
64     ↪TYPE_EXT_1N_BYTE, &output_img, NULL);
65     bm_image_alloc_dev_mem(input_img, 1);
66     bm_image_alloc_dev_mem(output_img, 1);

```

(续下页)

(接上页)

```
65     unsigned char* in_ptr[1] = {input_data};
66     bm_image_copy_host_to_device(input_img, (void **)in_ptr);
67     bmcv_image_threshold(handle, input_img, output_img, threshold, max_value, (bm_thresh_
68     ↪type_t)type);
69     unsigned char* out_ptr[1] = {output_tpu};
70     bm_image_copy_device_to_host(output_img, (void **)out_ptr);
71
72     bm_image_destroy(&input_img);
73     bm_image_destroy(&output_img);
74
75     write_bin(output_path, output_tpu, width, height);
76
77     free(input_data);
78     free(output_tpu);
79     bm_dev_free(handle);
80     return ret;
81 }
```

## 5.65 bmcv\_image\_transpose

该接口可以实现图片宽和高的转置。

**接口形式:**

```
bm_status_t bmcv_image_transpose(
    bm_handle_t handle,
    bm_image input,
    bm_image output);
```

**参数说明:**

- bm\_handle\_t handle  
输入参数。设备环境句柄，通过调用 bm\_dev\_request 获取。
- bm\_image input  
输入参数。输入图像的 bm\_image 结构体。
- bm\_image output  
输出参数。输出图像的 bm\_image 结构体。

**返回值说明:**

- BM\_SUCCESS: 成功
- 其他: 失败

**示例代码**

```
#include "bmvc_api_ext_c.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <pthread.h>

#define DTTYPE_F32 4
#define DTTYPE_U8 1
#define ALIGN(a, b) (((a) + (b) - 1) / (b) * (b))
#define TIME_COST_US(start, end) ((end.tv_sec - start.tv_sec) * 1000000 + (end.
→tv_usec - start.tv_usec))

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    }

    fclose(fp_src);
}

static void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb+");
    if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    }

    fclose(fp_dst);
}

int main()
{
    int channel = 1;
    int height = 1080;
    int width = 1920;
    int stride = width;
    int dtype = 1;
    int ret = 0;
    char* src_name = "path/to/src";
    char* dst_name = "path/to/dst";
    bm_handle_t handle;
    ret = bm_dev_request(&handle, 0);
    unsigned char* input = (unsigned char*) malloc(channel * height * stride * [F]
```

(续下页)

(接上页)

```
→sizeof(unsigned char));
    unsigned char* output = (unsigned char*) malloc(channel * height * width * [F]
→sizeof(unsigned char));
    readBin(src_name, input, channel * height * stride);
    memset(output, 0, channel * height * width);

    int format;
    int type = 0;
    int stride_byte;
    bm_image input_img, output_img;

    format = (channel == 3) ? FORMAT_BGR_PLANAR : FORMAT_GRAY;

    if (dtype == DTYPE_F32) {
        type = DATA_TYPE_EXT_FLOAT32;
        stride_byte = stride * sizeof(float);
    } else if (dtype == DTYPE_U8) {
        type = DATA_TYPE_EXT_1N_BYTE;
        stride_byte = stride * sizeof(unsigned char);
    }

    ret = bm_image_create(handle, height, width, (bm_image_format_ext)format,[F]
→(bm_image_data_format_ext)type, &input_img, &stride_byte);
    ret = bm_image_alloc_dev_mem(input_img, 2);
    ret = bm_image_copy_host_to_device(input_img, (void**)&input);
    ret = bm_image_create(handle, width, height, (bm_image_format_ext)format,[F]
→(bm_image_data_format_ext)type, &output_img, NULL);

    ret = bm_image_alloc_dev_mem(output_img, 2);

    ret = bmcv_image_transpose(handle, input_img, output_img);
    ret = bm_image_copy_device_to_host(output_img, (void**)&output);

    bm_image_destroy(&input_img);
    bm_image_destroy(&output_img);

    writeBin(dst_name, output, channel * height * width);

    free(input);
    free(output);

    bm_dev_free(handle);
    return ret;
}
```

### 注意事项:

1. 该 API 要求输入和输出的 bm\_image 图像格式相同，支持以下格式：

num	image_format
1	FORMAT_RGB_PLANAR
2	FORMAT_BGR_PLANAR
3	FORMAT_GRAY

2. 该 API 要求输入和输出的 bm\_image 数据类型相同，支持以下类型：

num	data_type
1	DATA_TYPE_EXT_FLOAT32
2	DATA_TYPE_EXT_1N_BYTE
3	DATA_TYPE_EXT_4N_BYTE
4	DATA_TYPE_EXT_1N_BYTE_SIGNED
5	DATA_TYPE_EXT_4N_BYTE_SIGNED

3. 输出图像的 width 必须等于输入图像的 height，输出图像的 height 必须等于输入图像的 width；
4. 输入图像支持带有 stride；
5. 输入输出 bm\_image 结构必须提前创建，否则返回失败。
6. 输入 bm\_image 必须 attach device memory，否则返回失败
7. 如果输出对象未 attach device memory，则会内部调用 bm\_image\_alloc\_dev\_mem 申请内部管理的 device memory，并将转置后的数据填充到 device memory 中。

## 5.66 bmcv\_image\_rotate\_trans

### 【描述】

实现图像顺时针旋转 90 度，180 度和 270 度

### 【语法】

```

1  bm_status_t bmcv_image_rotate_trans(
2      bm_handle_t handle,
3      bm_image input,
4      bm_image output,
5      int rotation_angle);

```

### 【参数】

表 5.43: bmcv\_image\_rotate 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
input	输出	输入的待旋转图像，需要外部调用 bmcv_image_create 创建。image 内存可以使用 bm_image_alloc_dev_mem 或者 bm_image_copy_host_to_device 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。
output	输出	输出的旋转后图像，需要外部调用 bmcv_image_create 创建。image 内存可以通过 bm_image_alloc_dev_mem 来开辟新的内存，或者使用 bmcv_image_attach 来 attach 已有的内存。如果不主动分配将在 api 内部进行自行分配
rotation_angle	输入	顺时针旋转角度。可选角度 90 度, 180 度, 270 度

#### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

#### 【格式支持】

1. 输入和输出的数据类型：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致，可支持：

num	image_format
1	FORMAT_YUV444P
2	FORMAT_NV12
3	FORMAT_NV21
4	FORMAT_RGB_PLANAR
5	FORMAT_BGR_PLANAR
6	FORMAT_RGBP_SEPARATE
7	FORMAT_BGRP_SEPARATE
8	FORMAT_GRAY

### 【注意】

- 在调用 bmcv\_image\_rotate\_trans() 之前必须确保输入的 image 内存已经申请。
- 输入输出图像的 data\_type、image\_format 必须相同。
- 输入输出图像的宽高尺寸支持 8\*8-8192\*8192。
- 输入输出图像为 nv12 和 nv21 图像格式时，因处理过程中会经过多次色彩变换，输出图像像素值将存在误差，但肉眼观察差异不大。

### 【代码示例】

```

1 #include <stdio.h>
2 #include "bmcv_api_ext_c.h"
3 #include "stdlib.h"
4 #include "string.h"
5 #include <assert.h>
6 #include <float.h>
7
8 #define BM1688 0x1686a200
9 #define TIME_COST_US(start, end) ((end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec -
10   start.tv_usec))
11
12 static void read_bin(const char *input_path, unsigned char *input_data, int width, int height,
13   float channel) {
14     FILE *fp_src = fopen(input_path, "rb");
15     if (fp_src == NULL)
16     {
17       printf("Can not open file! %s\n", input_path);
18       return;
19     }
20     if(fread(input_data, sizeof(unsigned char), width * height * channel, fp_src) != 0)
21       printf("read image success\n");

```

(续下页)

(接上页)

```

20     fclose(fp_src);
21 }
22
23 static void write_bin(const char *output_path, unsigned char *output_data, int width, int height, int channel) {
24     FILE *fp_dst = fopen(output_path, "wb");
25     if (fp_dst == NULL)
26     {
27         printf("Can not open file! %s\n", output_path);
28         return;
29     }
30     fwrite(output_data, sizeof(unsigned char), width * height * channel, fp_dst);
31     fclose(fp_dst);
32 }
33
34 int main() {
35     int width = 1920;
36     int height = 1080;
37     int format = 14;      // FORMAT_GRAY
38     int rotation_angle = 90;    // chosen from {90, 180, 270}
39     char *input_path = "path/to/input";
40     char *output_path = "path/to/output";
41     bm_handle_t handle;
42     bm_status_t ret = bm_dev_request(&handle, 0);
43     if (ret != BM_SUCCESS) {
44         printf("Create bm handle failed. ret = %d\n", ret);
45         return -1;
46     }
47
48     unsigned char* input_data = (unsigned char*)malloc(width * height * 3 * sizeof(unsigned char));
49     unsigned char* output_tpu = (unsigned char*)malloc(width * height * 3 * sizeof(unsigned char));
50     read_bin(input_path, input_data, width, height, 3);
51
52     bm_image input_img, output_img;
53     bm_image_create(handle, height, width, (bm_image_format_ext)format, DATA_TYPE_
54     EXT_1N_BYTE, &input_img, NULL);
55     bm_image_create(handle, width, height, (bm_image_format_ext)format, DATA_TYPE_
56     EXT_1N_BYTE, &output_img, NULL);
57
58     bm_image_alloc_dev_mem(input_img, 2);
59     bm_image_alloc_dev_mem(output_img, 2);
60
61     unsigned char *input_addr[3] = {input_data, input_data + height * width, input_data + height * width * 2};
62     bm_image_copy_host_to_device(input_img, (void **)(input_addr));
63
64     ret = bmcv_image_rotate_trans(handle, input_img, output_img, rotation_angle);
65     if (ret != BM_SUCCESS) {
66         printf("bmcv_image_rotate error!");
67     }

```

(续下页)

(接上页)

```
65     bm_image_destroy(&input_img);
66     bm_image_destroy(&output_img);
67     bm_dev_free(handle);
68     return -1;
69 }
70
71     unsigned char *output_addr[3] = {output_tpu, output_tpu + height * width, output_tpu[F
72 →+ 2 * height * width];
73     bm_image_copy_device_to_host(output_img, (void **)output_addr);
74
75     bm_image_destroy(&input_img);
76     bm_image_destroy(&output_img);
77
78     write_bin(output_path, output_tpu, width, height, 1);
79
80     free(input_data);
81     free(output_tpu);
82     bm_dev_free(handle);
83     return ret;
84 }
```

## 5.67 bmcv\_image\_rotate

### 【描述】

实现图像顺时针旋转 90 度，180 度和 270 度

### 【语法】

```
1 bm_status_t bmcv_image_rotate(
2     bm_handle_t handle,
3     bm_image input,
4     bm_image output,
5     int rotation_angle);
```

### 【参数】

表 5.44: bmcv\_image\_rotate 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取
input	输出	输入的待旋转图像，通过调用 bm_image_create 创建
output	输出	输出的旋转后图像，通过调用 bm_image_create 创建
rotation_angle	输入	顺时针旋转角度。可选角度 90 度, 180 度, 270 度

**【返回值】**

该函数成功调用时, 返回 BM\_SUCCESS。

**【格式支持】**

1. 输入和输出的数据类型:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE

2. 输入和输出的色彩格式必须保持一致, 可支持:

num	image_format
1	FORMAT_YUV444P
2	FORMAT_NV12
3	FORMAT_NV21
4	FORMAT_RGB_PLANAR
5	FORMAT_BGR_PLANAR
6	FORMAT_RGBP_SEPARATE
7	FORMAT_BGRP_SEPARATE
8	FORMAT_GRAY

**【注意】**

1. 输入输出所有 bm\_image 结构必须提前创建, 否则返回失败。

2. 输入输出图像的 data\_type、image\_format 必须相同。
3. 支持图像的分辨率为 64x64~4608x4608。

### 【代码示例】

```
1 #include "bmvc_api_ext_c.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <unistd.h>
6
7 static int writeBin(const char* path, void* output_data, int size)
8 {
9     int len = 0;
10    FILE* fp_dst = fopen(path, "wb+");
11
12    if (fp_dst == NULL) {
13        perror("Error opening file\n");
14        return -1;
15    }
16
17    len = fwrite((void*)output_data, 1, size, fp_dst);
18    if (len < size) {
19        printf("file size = %d is less than required bytes = %d\n", len, size);
20        return -1;
21    }
22
23    fclose(fp_dst);
24    return 0;
25}
26
27 static int readBin(const char* path, void* input_data)
28 {
29     int len;
30     int size;
31     FILE* fp_src = fopen(path, "rb");
32
33     if (fp_src == NULL) {
34         perror("Error opening file\n");
35         return -1;
36     }
37
38     fseek(fp_src, 0, SEEK_END);
39     size = ftell(fp_src);
40     fseek(fp_src, 0, SEEK_SET);
41
42     len = fread((void*)input_data, 1, size, fp_src);
43     if (len < size) {
```

(续下页)

(接上页)

```
44     printf("file size = %d is less than required bytes = %d\n", len, size);
45     return -1;
46 }
47
48 fclose(fp_src);
49 return 0;
50 }
51
52
53
54
55 int main() {
56     int rot_angle = 90;
57     int src_w = 1920, src_h = 1080, dst_w = 1080, dst_h = 1920, dev_id = 0;
58     bm_image_format_ext src_fmt = FORMAT_RGB_PLANAR, dst_fmt = FORMAT_RGB_
59     ↪PLANAR;
60     char *src_name = "/path/to/src";
61     char *dst_name = "path/to/dst";
62     bm_handle_t handle = NULL;
63     bm_status_t ret;
64     ret = bm_dev_request(&handle, dev_id);
65     bm_image src, dst;
66
66     bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src,[F]
67     ↪NULL);
67     bm_image_create(handle, dst_h, dst_w, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst,[F]
68     ↪NULL);
68
69     ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
70     ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
71
72     unsigned char* input_data = malloc(src_w * src_h * 3);
73     unsigned char* in_ptr[3] = {input_data, input_data + src_w * src_h, input_data + src_w[F]
74     ↪* src_h * 2};
74     readBin(src_name, input_data);
75     bm_image_copy_host_to_device(src, (void **)in_ptr);
76     bmcv_image_rotate(handle, src, dst, rot_angle);
77     bm_image_copy_device_to_host(dst, (void **)in_ptr);
78
79     writeBin(dst_name, input_data, src_w * src_h * 3);
80
81     bm_image_destroy(&src);
82     bm_image_destroy(&dst);
83
84     free(input_data);
85     bm_dev_free(handle);
86
87     return ret;
88 }
```

## 5.68 bmcv\_image\_vpp\_basic

### 【描述】

该 API 可以实现对多张图片的 crop、color-space-convert、resize、padding 及其任意若干个功能的组合。

### 【语法】

```
1 bm_status_t bmcv_image_vpp_basic(
2     bm_handle_t handle,
3     int in_img_num,
4     bm_image* input,
5     bm_image* output,
6     int* crop_num_vec = NULL,
7     bmcv_rect_t* crop_rect = NULL,
8     bmcv_padding_attr_t* padding_attr = NULL,
9     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
10    csc_type_t csc_type = CSC_MAX_ENUM,
11    csc_matrix_t* matrix = NULL);
```

### 【参数】

表 5.45: bmcv\_image\_vpp\_basic 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image_num	输入	输入 image 数量。
*input	输入	输入 bm_image 对象指针，其指向空间的长度由 in_img_num 决定。
* output	输出	输出 bm_image 对象指针，其指向空间的长度由 in_img_num 和 crop_num_vec 共同决定，即所有输入图片 crop 数量之和。
* crop_num_vec	输入	该指针指向对每张输入图片进行 crop 的数量，其指向空间的长度由 in_img_num 决定，如果不使用 crop 功能可填 NULL。
* crop_rect	输入	每个输出 bm_image 对象所对应的在输入图像上 crop 的参数。
* padding_attr	输入	所有 crop 的目标小图在 dst image 中的位置信息以及要 padding 的各通道像素值，若不使用 padding 功能则设置为 NULL。
algorithm	输入	resize 算法选择，包括 BMCV_INTER_NEAREST、BMCV_INTER_LINEAR 和 BMCV_INTER_BICUBIC 三种，默认情况下是双线性差值。
csc_type	输入	color space convert 参数类型选择，填 CSC_MAX_ENUM 则使用默认值，默认为 CSC_YCbCr2RGB_BT601 或者 CSC_RGB2YCbCr_BT601。
*matrix	输入	如果 csc_type 选择 CSC_USER_DEFINED_MATRIX，则需要用该指针传入系数矩阵。

## 【数据类型说明】

```

1  typedef enum csc_type {
2      CSC_YCbCr2RGB_BT601 = 0,
3      CSC_YPbPr2RGB_BT601,
4      CSC_RGB2YCbCr_BT601,
5      CSC_YCbCr2RGB_BT709,
6      CSC_RGB2YCbCr_BT709,
7      CSC_RGB2YPbPr_BT601,
8      CSC_YPbPr2RGB_BT709,
9      CSC_RGB2YPbPr_BT709,
10     CSC_FANCY_PbPr_BT601 = 100,
```

(续下页)

(接上页)

```

11 CSC_FANCY_PbPr_BT709,
12 CSC_USER_DEFINED_MATRIX = 1000,
13 CSC_MAX_ENUM
14 } csc_type_t;

```

可选的 csc\_type 参数。

其中 CbCr 表示 Limit range, 其应用 YUV 数据像素范围为 [16, 235], PbPr 表示 Full range, 其应用 YUV 数据像素范围为 [0, 255]。

BT601/BT709 代表相应的视频颜色空间标准。

FANCY 表示采用与 opencv 相同的处理方式计算上采样 (如从 YUV420P 上采样到 RGB)。

```

1 typedef struct bmcv_rect {
2     int start_x;
3     int start_y;
4     int crop_w;
5     int crop_h;
6 } bmcv_rect_t;

```

start\_x、start\_y、crop\_w、crop\_h 分别表示每个输出 bm\_image 对象所对应的在输入图像上 crop 的参数，包括起始点 x 坐标、起始点 y 坐标、crop 图像的宽度以及 crop 图像的高度。图像左上顶点作为坐标原点。如果不使用 crop 功能可填 NULL。

```

1 typedef struct bmcv_padding_attr_s {
2     unsigned int dst_crop_stx;
3     unsigned int dst_crop_sty;
4     unsigned int dst_crop_w;
5     unsigned int dst_crop_h;
6     unsigned char padding_r;
7     unsigned char padding_g;
8     unsigned char padding_b;
9     int         if_memset;
10 } bmcv_padding_attr_t;

```

1. 目标小图的左上角顶点相对于 dst image 原点 (左上角) 的 offset 信息: dst\_crop\_stx 和 dst\_crop\_sty;
2. 目标小图经 resize 后的宽高: dst\_crop\_w 和 dst\_crop\_h;
3. dst image 如果是 RGB 格式，各通道需要 padding 的像素值信息: padding\_r、padding\_g、padding\_b，当 if\_memset=1 时有效，如果是 GRAY 图像可以将三个值均设置为同一个值；
4. if\_memset 表示要不要在该 api 内部对 dst image 按照各个通道的 padding 值做 memset。如果设置为 0 则用户需要在调用该 api 前，根据需要 padding 的像素值信息，调用 bmlib 中的 api 直接对 device memory 进行 memset 操作，如果用户对 padding 的值不关心，可以设置为 0 忽略该步骤。

```

1  typedef struct {
2      short csc_coe00;
3      short csc_coe01;
4      short csc_coe02;
5      unsigned char csc_add0;
6      unsigned char csc_sub0;
7      short csc_coe10;
8      short csc_coe11;
9      short csc_coe12;
10     unsigned char csc_add1;
11     unsigned char csc_sub1;
12     short csc_coe20;
13     short csc_coe21;
14     short csc_coe22;
15     unsigned char csc_add2;
16     unsigned char csc_sub2;
17 } csc_matrix_t;

```

自定义 csc\_matrix 的系数。其中，矩阵变换关系如下：

$$\begin{cases} dst_0 = (coe_{00} * (src_0 - sub_0) + coe_{01} * (src_1 - sub_1) + coe_{02} * (src_2 - sub_2)) >> 10 + add_0 \\ dst_1 = (coe_{10} * (src_0 - sub_0) + coe_{11} * (src_1 - sub_1) + coe_{12} * (src_2 - sub_2)) >> 10 + add_1 \\ dst_2 = (coe_{20} * (src_0 - sub_0) + coe_{21} * (src_1 - sub_1) + coe_{22} * (src_2 - sub_2)) >> 10 + add_2 \end{cases}$$

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【格式支持】

1. 支持的数据类型为：

num	input data_type	output data_type
1	DATA_TYPE_EXT_1N_BYTE	DATA_TYPE_EXT_FLOAT32
2		DATA_TYPE_EXT_1N_BYTE
3		DATA_TYPE_EXT_1N_BYTE_SIGNED
4		DATA_TYPE_EXT_FP16
5		DATA_TYPE_EXT_BF16

2. 输入支持色彩格式为：

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_COMPRESSED
16	FORMAT_YUV444_PACKED
17	FORMAT_YVU444_PACKED
18	FORMAT_YUV422_YUYV
19	FORMAT_YUV422_YVYU
20	FORMAT_YUV422_UYVY
21	FORMAT_YUV422_VYUY

3. 输出支持色彩格式为：

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV21
6	FORMAT_NV16
7	FORMAT_NV61
8	FORMAT_RGB_PLANAR
9	FORMAT_BGR_PLANAR
10	FORMAT_RGB_PACKED
11	FORMAT_BGR_PACKED
12	FORMAT_RGBP_SEPARATE
13	FORMAT_BGRP_SEPARATE
14	FORMAT_GRAY
15	FORMAT_YUV422_YUYV
16	FORMAT_YUV422_YVYU
17	FORMAT_YUV422_UYVY
18	FORMAT_YUV422_VYUY
19	FORMAT_HSV_PLANAR

### 【注意】

1. 图片实际缩放倍数 ((crop.width / output.width) 以及 (crop.height / output.height)) 限制在 1/128 ~ 128 之间。
2. 输入输出的宽高 (src.width, src.height, dst.width, dst.height) 限制在 16 ~ 8192 之间。
3. 输入必须关联 device memory , 否则返回失败。
4. 该接口支持 byte align 的数据输入。但当输入为 YUV420P / NV12 / NV21 格式时，使用 stride 32 byte align 的数据作为输入可使转换效率更高。
5. FORMAT\_COMPRESSED 是 VPU 解码后内置的一种压缩格式，它包括 4 个部分：Y compressed table、Y compressed data、CbCr compressed table 以及 CbCr compressed data。请注意 bm\_image 中这四部分存储的顺序与 FFMPEG 中 AVFrame 稍有不同，如果需要 attach AVFrame 中 device memory 数据到 bm\_image 中时，对应关系如下，关于 AVFrame 详细内容请参考 VPU 的用户手册。

```

bm_device_mem_t src_plane_device[4];
src_plane_device[0] = bm_mem_from_device((u64)avframe->data[6],
                                         avframe->linesize[6]);
src_plane_device[1] = bm_mem_from_device((u64)avframe->data[4],
                                         avframe->linesize[4] * avframe->h);
src_plane_device[2] = bm_mem_from_device((u64)avframe->data[7],
                                         avframe->linesize[7]);

```

(续下页)

(接上页)

```

1 src_plane_device[3] = bm_mem_from_device((u64)avframe->data[5],
2     avframe->linesize[4] * avframe->h / 2);
3
4 bm_image_attach(*compressed_image, src_plane_device);

```

## 5.69 bmcv\_image\_vpp\_convert

### 【描述】

该 API 将输入图像格式转化为输出图像格式，并支持 crop + resize 功能，支持从 1 张输入中 crop 多张输出并 resize 到输出图片大小。

### 【语法】

```

1 bm_status_t bmcv_image_vpp_convert(
2     bm_handle_t handle,
3     int output_num,
4     bm_image input,
5     bm_image *output,
6     bmcv_rect_t *crop_rect,
7     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);

```

### 【参数】

表 5.46: bmcv\_image\_vpp\_convert 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
output_num	输出	输出 bm_image 数量，和 src image 的 crop 数量相等，一个 src crop 输出一个 dst bm_image。
input	输入	输入 bm_image 对象。
* output	输出	输出 bm_image 对象指针。
* crop_rect	输入	每个输出 bm_image 对象所对应的在输入图像上 crop 的参数。
algorithm	输入	resize 算法选择，包括 BMCV_INTER_NEAREST、BMCV_INTER_LINEAR 和 BMCV_INTER_BICUBIC 三种，默认情况下是双线性差值。

**【注意】**

该接口的参数数据类型与注意事项与 bmcv\_image\_vpp\_basic 接口相同。

**【代码示例】**

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "bmcv_api_ext_c.h"

int main() {
    char *filename_src = "path/to/src";
    char *filename_dst = "path/to/dst";

    int in_width = 1920;
    int in_height = 1080;
    int out_width = 1920;
    int out_height = 1080;

    bm_image_format_ext src_format = 0; // FORMAT_YUV420P
    bm_image_format_ext dst_format = 0;
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR;

    bmcv_rect_t crop_rect = {
        .start_x = 500,
        .start_y = 500,
        .crop_w = 200,
        .crop_h = 200};

    bm_status_t ret = BM_SUCCESS;

    int src_size = in_width * in_height * 3 / 2;
    int dst_size = in_width * in_height * 3 / 2;
    unsigned char *src_data = (unsigned char *)malloc(src_size);
    unsigned char *dst_data = (unsigned char *)malloc(dst_size);

    FILE *file;
    file = fopen(filename_src, "rb");
    fread(src_data, sizeof(unsigned char), src_size, file);
    fclose(file);

    bm_handle_t handle;
    int dev_id = 0;
    bmcv_image src, dst;
```

(续下页)

(接上页)

```

ret = bm_dev_request(&handle, dev_id);

bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_1N_BYTE,
→ &src, NULL);
bm_image_create(handle, out_height, out_width, dst_format, DATA_TYPE_EXT_1N_
→ BYTE, &dst, NULL);
bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int src_image_byte_size[4] = {0};
bm_image_get_byte_size(src, src_image_byte_size);
void *src_in_ptr[4] = {(void *)src_data,
                      (void *)((char *)src_data + src_image_byte_size[0]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→ size[1]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→ size[1] + src_image_byte_size[2])};

bm_image_copy_host_to_device(src, (void **)src_in_ptr);
ret = bmcv_image_vpp_csc_matrix_convert(handle, 1, src, &dst, CSC_MAX_ENUM, NULL,
→ algorithm, &crop_rect);

int dst_image_byte_size[4] = {0};
bm_image_get_byte_size(dst, dst_image_byte_size);
void *dst_in_ptr[4] = {(void *)dst_data,
                      (void *)((char *)dst_data + dst_image_byte_size[0]),
                      (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→ size[1]),
                      (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→ size[1] + dst_image_byte_size[2])};

bm_image_copy_device_to_host(dst, (void **)dst_in_ptr);

FILE *fp_dst = fopen(filename_dst, "wb");
if (fwrite((void *)dst_data, 1, dst_size, fp_dst) < (unsigned int)dst_size){
    printf("file size is less than %d required bytes\n", dst_size);
}
fclose(fp_dst);

bm_image_destroy(&src);
bm_image_destroy(&dst);
bm_dev_free(handle);

free(src_data);
free(dst_data);

return ret;

```

(续下页)

(接上页)

{}

## 5.70 bmcv\_image\_vpp\_convert\_padding

### 【描述】

该 API 通过对 dst image 做 memset 操作，实现图像 padding 的效果。通俗的讲是将一张小图填充到大图中。可以从一张 src image 上 crop 多个目标图像，对于每一个目标小图，可以一次性完成 csc+resize 操作，然后根据其在大图中的 offset 信息，填充到大图中。一次 crop 的数量不能超过 512。

### 【语法】

```
1 bm_status_t bmcv_image_vpp_convert_padding(  
2     bm_handle_t      handle,  
3     int              output_num,  
4     bm_image          input,  
5     bm_image *        output,  
6     bmcv_padding_attr_t * padding_attr,  
7     bmcv_rect_t *      crop_rect = NULL,  
8     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);
```

### 【参数】

表 5.47: bmcv\_image\_vpp\_convert\_padding 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
output_num	输出	输出 bm_image 数量，和 src image 的 crop 数量相等，一个 src crop 输出一个 dst bm_image。
input	输入	输入 bm_image 对象。
* output	输出	输出 bm_image 对象指针。
* padding_attr	输入	src crop 的目标小图在 dst image 中的位置信息以及要 padding 的各通道像素值。
* crop_rect	输入	每个输出 bm_image 对象所对应的在输入图像上 crop 的参数。
algorithm	输入	resize 算法选择，包括 BMVC_INTER_NEAREST、 BMVC_INTER_LINEAR 和 BMVC_INTER_BICUBIC 三种， 默认情况下是双线性差值。

**【注意】**

该接口的参数数据类型与注意事项与 bmcv\_image\_vpp\_basic 接口相同。

**【代码示例】**

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "bmcv_api_ext_c.h"

int main() {
    char *filename_src = "path/to/src";
    char *filename_dst = "path/to/dst";
    int in_width = 1920;
    int in_height = 1080;
    int out_width = 1920;
    int out_height = 1080;
    bm_image_format_ext src_format = 0; // FORMAT_YUV420P
    bm_image_format_ext dst_format = 0;
    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR;
```

(续下页)

(接上页)

```

bmcv_rect_t crop_rect = {
    .start_x = 100,
    .start_y = 100,
    .crop_w = 500,
    .crop_h = 500
};

bmcv_padding_attr_t padding_rect = {
    .dst_crop_stx = 0,
    .dst_crop_sty = 0,
    .dst_crop_w = 1000,
    .dst_crop_h = 1000,
    .padding_r = 155,
    .padding_g = 20,
    .padding_b = 36,
    .if_memset = 1
};

bm_status_t ret = BM_SUCCESS;

int src_size = in_height * in_width * 3 / 2;
int dst_size = in_height * in_width * 3 / 2;
unsigned char *src_data = (unsigned char *)malloc(src_size);
unsigned char *dst_data = (unsigned char *)malloc(dst_size);

FILE *file;
file = fopen(filename_src, "rb");
fread(src_data, sizeof(unsigned char), src_size, file);
fclose(file);

bm_handle_t handle = NULL;
int dev_id = 0;
bm_image src, dst;

ret = bm_dev_request(&handle, dev_id);

bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_
    ↵1N_BYTE, &src, NULL);
bm_image_create(handle, out_height, out_width, dst_format, DATA_TYPE_
    ↵EXT_1N_BYTE, &dst, NULL);
bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int src_image_byte_size[4] = {0};
bm_image_get_byte_size(src, src_image_byte_size);
void *src_in_ptr[4] = {(void *)src_data,
    ↵(void *)((char *)src_data + src_image_byte_size[0]),
    ↵(void *)((char *)src_data + src_image_byte_size[0] + src_image_
        ↵byte_size[1])),
    ↵(void *)((char *)src_data + src_image_byte_size[0] + src_image_
        ↵byte_size[1] + src_image_byte_size[2])};

```

(续下页)

(接上页)

```

bm_image_copy_host_to_device(src, (void **)src_in_ptr);
ret = bmcv_image_vpp_convert_padding(handle, 1, src, &dst, &padding_rect, &
crop_rect, algorithm);

int dst_image_byte_size[4] = {0};
bm_image_get_byte_size(dst, dst_image_byte_size);
void *dst_in_ptr[4] = {(void *)dst_data,
    ((void *)((char *)dst_data + dst_image_byte_size[0])),
    ((void *)((char *)dst_data + dst_image_byte_size[0] + dst_
image_byte_size[1])),
    ((void *)((char *)dst_data + dst_image_byte_size[0] + dst_
image_byte_size[1] + dst_image_byte_size[2]))};

bm_image_copy_device_to_host(dst, (void **)dst_in_ptr);

FILE *fp_dst = fopen(filename_dst, "wb");
if (fwrite((void *)dst_data, 1, dst_size, fp_dst) < (unsigned int)dst_size){
    printf("file size is less than %d required bytes\n", dst_size);
}
fclose(fp_dst);

bm_image_destroy(&src);
bm_image_destroy(&dst);
bm_dev_free(handle);

free(src_data);
free(dst_data);

return ret;
}

```

### 5.71 bmcv\_image\_vpp\_csc\_matrix\_convert

#### 【描述】

默认情况下，bmcv\_image\_vpp\_convert 使用的是 BT\_601 标准进行色域转换。有些情况下需要使用其他标准，或者用户自定义 csc 参数。

#### 【语法】

```

1 bm_status_t bmcv_image_vpp_csc_matrix_convert(
2     bm_handle_t handle,
3     int output_num,

```

(续下页)

(接上页)

```

4   bm_image input,
5   bm_image *output,
6   csc_type_t csc,
7   csc_matrix_t *matrix = NULL,
8   bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR,
9   bmcv_rect_t *crop_rect = NULL);

```

**【参数】**

表 5.48: bmcv\_image\_vpp\_csc\_matrix\_convert 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取。
image_num	输入	输入 <code>bm_image</code> 数量。
input	输入	输入 <code>bm_image</code> 对象。
*output	输出	输出 <code>bm_image</code> 对象指针。
csc	输入	色域转换枚举类型。
*matrix	输入	色域转换自定义矩阵，当且仅当 <code>csc</code> 为 <code>CSC_USER_DEFINED_MATRIX</code> 时这个值才生效。
algorithm	输入	resize 算法选择，包括 <code>BMCV_INTER_NEAREST</code> 、 <code>BMCV_INTER_LINEAR</code> 和 <code>BMCV_INTER_BICUBIC</code> 三种， 默认情况下是双线性差值。

**【注意】**

该接口的参数数据类型与注意事项与 `bmcv_image_vpp_basic` 接口相同。

**【返回值】**

该函数成功调用时，返回 `BM_SUCCESS`。

**【代码示例】**

```
1 #include <limits.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 #include "bmcv_api_ext_c.h"
7
8 int main() {
9     char* filename_src = "path/to/src_file";
10    char* filename_dst = "path/to/dst_file";
11    int in_width = 1920;
12    int in_height = 1080;
13    int out_width = 1920;
14    int out_height = 1080;
15    // int use_real_img = 1;
16    bm_image_format_ext src_format = 0;
17    bm_image_format_ext dst_format = 0;
18    bmcv_rect_t crop_rect = {
19        .start_x = 0,
20        .start_y = 0,
21        .crop_w = 200,
22        .crop_h = 200};
23    bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR;
24    bm_image_data_format_ext data_format = 1;
25
26    bm_status_t ret = BM_SUCCESS;
27
28    int src_size = in_width * in_height * 3 / 2;
29    int dst_size = out_width * out_height * 3 / 2;
30    unsigned char *src_data = (unsigned char *)malloc(src_size);
31    unsigned char *dst_data = (unsigned char *)malloc(dst_size);
32
33    FILE *file;
34    file = fopen(filename_src, "rb");
35    fread(src_data, sizeof(unsigned char), src_size, file);
36    fclose(file);
37
38    bm_handle_t handle = NULL;
39    int dev_id = 0;
40    bm_image src, dst;
41
42    ret = bm_dev_request(&handle, dev_id);
43    if (ret != BM_SUCCESS) {
44        printf("Create bm handle failed. ret = %d\n", ret);
45        return ret;
46    }
47
48    bm_image_create(handle, in_height, in_width, src_format, DATA_TYPE_EXT_1N_BYTE,
49    ↪ &src, NULL);
50    bm_image_create(handle, out_height, out_width, dst_format, data_format, &dst, NULL);
51    bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
```

(续下页)

(接上页)

```
51     bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
52
53     int src_image_byte_size[4] = {0};
54     bm_image_get_byte_size(src, src_image_byte_size);
55
56     void *src_in_ptr[4] = { (void *)src_data,
57                           (void *)((char *)src_data + src_image_byte_size[0]),
58                           (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→size[1]),
59                           (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→size[1] + src_image_byte_size[2])};
60
61     bm_image_copy_host_to_device(src, (void **)src_in_ptr);
62
63 // ret = bmcv_image_vpp_convert(handle, 1, src, &dst, crop_rect, algorithm);
64 ret = bmcv_image_vpp_csc_matrix_convert(handle, 1, src, &dst, CSC_MAX_ENUM, NULL,
→ algorithm, &crop_rect);
65
66     int dst_image_byte_size[4] = {0};
67     bm_image_get_byte_size(dst, dst_image_byte_size);
68
69     void *dst_in_ptr[4] = { (void *)dst_data,
70                           (void *)((char *)dst_data + dst_image_byte_size[0]),
71                           (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→size[1]),
72                           (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→size[1] + dst_image_byte_size[2])};
73
74     bm_image_copy_device_to_host(dst, (void **)dst_in_ptr);
75
76     bm_image_destroy(&src);
77     bm_image_destroy(&dst);
78     bm_dev_free(handle);
79
80     file = fopen(filename_dst, "wb");
81     fwrite(dst_data, sizeof(unsigned char), dst_size, file);
82     fclose(file);
83
84
85     free(src_data);
86     free(dst_data);
87
88     return ret;
89 }
```

## 5.72 bmcv\_image\_vpp\_stitch

### 【描述】

该 API 可实现图像拼接的效果，对输入 image 可以一次完成 src crop + csc + resize + dst crop 操作。dst image 中拼接的小图像数量不能超过 512。

### 【语法】

```
1 bm_status_t bmcv_image_vpp_stitch(
2     bm_handle_t      handle,
3     int              input_num,
4     bm_image*        input,
5     bm_image         output,
6     bmcv_rect_t*    dst_crop_rect,
7     bmcv_rect_t*    src_crop_rect = NULL,
8     bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR);
```

### 【参数】

表 5.49: bmcv\_image\_vpp\_stitch 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input_num	输入	输入 bm_image 数量。
*input	输入	输入 bm_image 对象指针。
output	输出	输出 bm_image 对象。
*dst_crop_rect	输入	在 dst images 上，各个目标小图的坐标和宽高信息。
*src_crop_rect	输入	在 src image 上，各个目标小图的坐标和宽高信息。
algorithm	输入	resize 算法选择，包括 BMCV_INTER_NEAREST、BMCV_INTER_LINEAR 和 BMCV_INTER_BICUBIC 三种，默认情况下是双线性差值。

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

1. 如果对 src image 做 crop 操作，一张 src image 只 crop 一个目标。
2. 该接口的参数数据类型与其他注意事项与 bmcv\_image\_vpp\_basic 接口相同。

### 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

int src_h = 1080, src_w = 1920, dst_w = 1920, dst_h = 2160, dev_id = 0;
bm_image_format_ext src_fmt = FORMAT_YUV420P, dst_fmt = FORMAT_YUV420P;
char *src_name = "path/to/src", *dst_name = "/path/to/dst";
bmcv_rect_t dst_rect0 = {.start_x = 0, .start_y = 0, .crop_w = 1920, .crop_h = 1080};
bmcv_rect_t dst_rect1 = {.start_x = 0, .start_y = 1080, .crop_w = 1920, .crop_h = 1080};
bmcv_resize_algorithm algorithm = BMCV_INTER_LINEAR;
bm_handle_t handle = NULL;

int main() {
    bm_status_t ret;
    bm_image src, dst;
    ret = bm_dev_request(&handle, dev_id);
    bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, F
    ↵NULL);
    bm_image_create(handle, dst_h, dst_w, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst, F
    ↵NULL);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    int src_size = src_h * src_w * 3 / 2;
    int dst_size = src_h * src_w * 3 / 2;
    unsigned char *src_data = (unsigned char *)malloc(src_size);
    unsigned char *dst_data = (unsigned char *)malloc(dst_size);

    FILE *file;
    file = fopen(src_name, "rb");
    fread(src_data, sizeof(unsigned char), src_size, file);
    fclose(file);

    int src_image_byte_size[4] = {0};
```

(续下页)

(接上页)

```
bm_image_get_byte_size(src, src_image_byte_size);
void *src_in_ptr[4] = { (void *)src_data,
                      (void *)((char *)src_data + src_image_byte_size[0]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→size[1]),
                      (void *)((char *)src_data + src_image_byte_size[0] + src_image_byte_
→size[1] + src_image_byte_size[2])};
bm_image_copy_host_to_device(src, (void **)src_in_ptr);

bmcv_rect_t rect = {.start_x = 0, .start_y = 0, .crop_w = src_w, .crop_h = src_h};
bmcv_rect_t src_rect[2] = {rect, rect};
bmcv_rect_t dst_rect[2] = {dst_rect0, dst_rect1};

bm_image input[2] = {src, src};

bmcv_image_vpp_stitch(handle, 2, input, dst, dst_rect, src_rect, BMCV_INTER_LINEAR);

int dst_image_byte_size[4] = {0};
bm_image_get_byte_size(dst, dst_image_byte_size);
void *dst_in_ptr[4] = { (void *)dst_data,
                      (void *)((char *)dst_data + dst_image_byte_size[0]),
                      (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→size[1]),
                      (void *)((char *)dst_data + dst_image_byte_size[0] + dst_image_byte_
→size[1] + dst_image_byte_size[2])};
bm_image_copy_device_to_host(dst, (void **)dst_in_ptr);

FILE *fp_dst = fopen(dst_name, "wb");
if (fwrite((void *)dst_data, 1, dst_size, fp_dst) < (unsigned int)dst_size){
    printf("file size is less than %d required bytes\n", dst_size);
}
fclose(fp_dst);

bm_image_destroy(&src);
bm_image_destroy(&dst);
bm_dev_free(handle);

return ret;
}
```

### 5.73 bmcv\_image\_overlay

#### 【描述】

该 API 实现在图片上叠加带透明通道的水印图。该接口可搭配 bmcv\_gen\_text\_watermark 接口实现绘制中英文的功能，参照代码示例 2。

#### 【语法】

```

1 bm_status_t bmcv_image_overlay(
2     bm_handle_t      handle,
3     bm_image         image,
4     int              overlay_num,
5     bmcv_rect_t*    overlay_info,
6     bm_image*        overlay_image);

```

#### 【参数】

表 5.50: bmcv\_image\_overlay 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image	输入/输出	输入/输出 bm_image 对象。
overlay_num	输入	水印图的个数。
* overlay_info	输入	输入水印图在输入图片上区域信息。
* overlay_image	输入	输入 overlay_image 对象。

#### 【数据类型说明】

```

1 typedef struct bmcv_rect {
2     unsigned int start_x;
3     unsigned int start_y;
4     unsigned int crop_w;
5     unsigned int crop_h;
6 } bmcv_rect_t;

```

start\_x、start\_y、crop\_w、crop\_h 分别表示输入水印图对象在输入图像上信息，包括起始点 x 坐标、起始点 y 坐标、水印图像的宽度以及水印图像的高度。图像左上顶点作为坐标原点。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 水印图色彩格式为:

num	input image_format
1	FORMAT_ARGB_PACKED
2	FORMAT_ARGB444_PACKED
3	FORMAT_ARGB1555_PACKED

2. 水印图的 stride 需要 16 对齐。
3. 该 API 所需要满足的底图格式以及尺寸要求与 bmcv\_image\_vpp\_basic 一致。
4. 输入必须关联 device memory, 否则返回失败。
5. 不支持对单底图进行位置重叠的多图叠加。

## 【代码示例 1】

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "stb_image.h"

#include "bmcv_api_ext_c.h"

void readBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size){
        printf("read_Bin: file size is less than %d required bytes\n", size);
    };
    fclose(fp_src);
}

void writeBin(const char * path, unsigned char* input_data, int size)
{
    FILE *fp_dst = fopen(path, "wb");
```

(续下页)

(接上页)

```

if (fwrite((void *)input_data, 1, size, fp_dst) < (unsigned int)size){
    printf("write_Bin: file size is less than %d required bytes\n", size);
}

fclose(fp_dst);
}

int main(){
bm_image_format_ext image_fmt = FORMAT_RGB_PACKED;
bm_image_format_ext overlay_fmt = FORMAT_ARGB_PACKED;

int img_w = 1920, img_h = 1080;
int overlay_w = 300, overlay_h = 300;
int overlay_num = 1;
int x = 500, y = 500;
char *image_name = "path/to/image";
char *overlay_name = "path/to/overlay";
char *output_image = "path/to/output";
bm_handle_t handle = NULL;
bm_status_t ret = bm_dev_request(&handle, 0);

// config setting
bmvc_rect_t overlay_info;
memset(&overlay_info, 0, sizeof(bmvc_rect_t));

overlay_info.start_x = x;
overlay_info.start_y = y;
overlay_info.crop_h = overlay_h;
overlay_info.crop_w = overlay_w;

unsigned char* img = malloc(img_h * img_w * 3);
unsigned char* output_tpu = malloc(img_h * img_w * 3);
readBin(image_name, img, img_h * img_w * 3);
memset(output_tpu, 0, img_h * img_w * 3);

// create bm image struct & alloc dev mem
bm_image image;
bm_image_create(handle, img_h, img_w, image_fmt, DATA_TYPE_EXT_1N_BYTE, &
→image, NULL);
ret = bm_image_alloc_dev_mem(image, BMCV_HEAP1_ID);
unsigned char *in1_ptr[1] = {img};
bm_image_copy_host_to_device(image, (void **)(in1_ptr));

unsigned char* overlay_ptr = malloc(overlay_w * overlay_h * 4);
readBin(overlay_name, overlay_ptr, overlay_w * overlay_h * 4);

bm_image overlay_image[overlay_num];
bm_image_create(handle, overlay_h, overlay_w, overlay_fmt, DATA_TYPE_EXT_1N_
→BYTE, overlay_image, NULL);
ret = bm_image_alloc_dev_mem(overlay_image[0], BMCV_HEAP1_ID);

```

(续下页)

(接上页)

```

unsigned char* in_overlay[4] = {overlay_ptr, overlay_ptr + overlay_h * overlay_w, overlay_
→ptr + 2 * overlay_w * overlay_h, overlay_ptr + overlay_w * overlay_h * 3};
bm_image_copy_host_to_device(overlay_image[0], (void **)(in_overlay));

ret = bmcv_image_overlay(handle, image, overlay_num, &overlay_info, overlay_image);

unsigned char *out_ptr[3] = {output_tpu, output_tpu + img_h * img_w, output_tpu + 2 * [F
→img_h * img_w];
bm_image_copy_device_to_host(image, (void **)out_ptr);

writeBin(output_image, output_tpu, img_h * img_w * 3);

free(img);
free(output_tpu);
bm_dev_free(handle);
return ret;
}

```

## 【代码示例 2】

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <bmcv_api_ext_c.h>

int main(int argc, char* args[]){

    setlocale(LC_ALL, "");
    bm_status_t ret = BM_SUCCESS;
    wchar_t hexcode[256];
    int r = 255, g = 255, b = 0;
    unsigned char fontScale = 2;
    char* output_path = "out.bmp";
    mbstowcs(hexcode, "北京beijing" sizeof(hexcode) / sizeof(wchar_t)); //unsigned
    printf("Received wide character string: %ls\n", hexcode);
    printf("output path: %s\n", output_path);

    bm_image image;
    bm_handle_t handle = NULL;
    bm_dev_request(&handle, 0);
    bm_image_create(handle, 1080, 1920, FORMAT_YUV420P, DATA_TYPE_EXT_1N_BYTE,
→ &image, NULL);
    bm_image_alloc_dev_mem(image, BMCV_HEAP1_ID);
}

```

(续下页)

(接上页)

```

bm_read_bin(image, "/opt/sophon/libsonphon-current/bin/res/1920x1080_yuv420.bin");
bmcv_point_t org = {.x = 0, .y = 500};
bmcv_color_t color = {.r = r, .g = g, .b = b};

bm_image watermark;
ret = bmvc_gen_text_watermark(handle, hexcode, color, fontScale, FORMAT_ARGB_
→PACKED, &watermark);
if (ret != BM_SUCCESS) {
    printf("bmvc_gen_text_watermark fail\n");
    goto fail1;
}

bmvc_rect_t rect = {.start_x = org.x, .start_y = org.y, .crop_w = watermark.width, .crop_
→h = watermark.height};
ret = bmvc_image_overlay(handle, image, 1, &rect, &watermark);
if (ret != BM_SUCCESS) {
    printf("bmvc_image_overlay fail\n");
    goto fail2;
}
bm_image_write_to_bmp(image, output_path);

fail2:
bm_image_destroy(&watermark);
fail1:
bm_image_destroy(&image);
bm_dev_free(handle);
return ret;
}

```

## 5.74 bmvc\_image\_warp\_affine

该接口实现图像的仿射变换，可实现旋转、平移、缩放等操作。仿射变换是一种二维坐标  $(x, y)$  到二维坐标  $(x_0, y_0)$  的线性变换，该接口的实现是针对输出图像的每一个像素点找到在输入图像中对应的坐标，从而构成一幅新的图像，其数学表达式形式如下：

$$\begin{cases} x_0 = a_1x + b_1y + c_1 \\ y_0 = a_2x + b_2y + c_2 \end{cases}$$

对应的齐次坐标矩阵表示形式为：

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

坐标变换矩阵是一个 6 点的矩阵，该矩阵是从输出图像坐标推导输入图像坐标的系数矩阵，可以通过输入输出图像上对应的 3 个点坐标来获取。在人脸检测中，通过获取人脸定位点来获取变换矩阵。

bmcv\_affine\_matrix 定义了一个坐标变换矩阵，其顺序为 float m[6] = {a1, b1, c1, a2, b2, c2}。而 bmcv\_affine\_image\_matrix 定义了一张图片里面有几个变换矩阵，通常来说一张图片有多个脸时，会对应多个变换矩阵。

```
typedef struct bmcv_affine_matrix_s{
    float m[6];
} bmcv_warp_matrix;

typedef struct bmcv_affine_image_matrix_s{
    bmcv_affine_matrix *matrix;
    int matrix_num;
} bmcv_affine_image_matrix;
```

**接口形式一：**

```
bm_status_t bmcv_image_warp_affine(
    bm_handle_t handle,
    int image_num,
    bmcv_affine_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0);
```

**接口形式二：**

```
bm_status_t bmcv_image_warp_affine_similar_to_opencv(
    bm_handle_t handle,
    int image_num,
    bmcv_affine_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0);
```

本接口是对齐 opencv 仿射变换的接口，该矩阵是从输入图像坐标推导输出图像坐标的系数矩阵。

### 参数说明

- bm\_handle\_t handle  
输入参数。输入的 bm\_handle 句柄。
- int image\_num  
输入参数。输入图片数，最多支持 4。
- bmcv\_affine\_image\_matrix matrix[4]  
输入参数。每张图片对应的变换矩阵数据结构，最多支持 4 张图片。
- bm\_image\* input  
输入参数。输入 bm\_image，对于 1N 模式，最多 4 个 bm\_image，对于 4N 模式，最多一个 bm\_image。

- `bm_image*` output

输出参数。输出 `bm_image`, 外部需要调用 `bmcv_image_create` 创建, 建议用户调用 `bmcv_image_attach` 来分配 device memory。如果用户不调用 `attach`, 则内部分配 device memory。对于输出 `bm_image`, 其数据类型和输入一致, 即输入是 4N 模式, 则输出也是 4N 模式, 输入 1N 模式, 输出也是 1N 模式。所需要的 `bm_image` 大小是所有图片的变换矩阵之和。比如输入 1 个 4N 模式的 `bm_image`, 4 张图片的变换矩阵数目为 **【3,0,13,5】**, 则共有变换矩阵  $3+0+13+5=21$ , 由于输出是 4N 模式, 则需要  $(21+4-1)/4=6$  个 `bm_image` 的输出。

- `int use_bilinear`

输入参数。是否使用 bilinear 进行插值, 若为 0 则使用 nearest 插值, 若为 1 则使用 bilinear 插值, 默认使用 nearest 插值。选择 nearest 插值的性能会优于 bilinear, 因此建议首选 nearest 插值, 除非对精度有要求时可选择使用 bilinear 插值。

#### 返回值说明:

- `BM_SUCCESS`: 成功
- 其他: 失败

#### 注意事项

1. 该接口所支持的 `image_format` 包括:

num	<code>image_format</code>
1	<code>FORMAT_BGR_PLANAR</code>
2	<code>FORMAT_RGB_PLANAR</code>

2. 该接口所支持的 `data_type` 包括:

num	<code>data_type</code>
1	<code>DATA_TYPE_EXT_1N_BYTE</code>
2	<code>DATA_TYPE_EXT_4N_BYTE</code>

3. 该接口的输入以及输出 `bm_image` 均支持带有 stride。
4. 要求该接口输入 `bm_image` 的 width、height、`image_format` 以及 `data_type` 必须保持一致。
5. 要求该接口输出 `bm_image` 的 width、height、`image_format`、`data_type` 以及 stride 必须保持一致。

#### 示例代码

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
```

(续下页)

(接上页)

```

#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    len = fwrite((void*)output_data, 1, size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_dst);
    return 0;
}

static unsigned char* image_read_2(
    int image_n,
    int image_c,
    int image_sh,
    int image_sw,
    int image_dh,
    int image_dw) {

    unsigned char* res = (unsigned char*) malloc(image_n * image_c * image_
        _sh * image_sw * sizeof(unsigned char));
    unsigned char* res_temp = (unsigned char*) malloc(image_n * image_c * [F]
        _image_dh * image_dw * sizeof(unsigned char));
    unsigned char* res_temp_bak = (unsigned char*) malloc(image_n * image_c * [F]
        _image_dh * image_dw * sizeof(unsigned char));

    for (int i = 0; i < image_n * image_c * image_sh * image_sw; i++)
    {
        res[i] = i % 255;
    }

    if (image_dh <= image_sh && image_dw <= image_sw)
        return res;

    if (image_dh > image_sh){
        int pad_h_value = (image_dh - image_sh) / 2;
        for (int i = 0;i < pad_h_value * image_sw;i++)
            res_temp[i] = 0;
    }
}

```

(续下页)

(接上页)

```

        for (int i = pad_h_value * image_sw, j = 0; i < pad_h_value * image_sw + [F]
            ↵image_n * image_c * image_sh * image_sw;i++,j++)
            res_temp[i] = res[j];

        for (int i = pad_h_value * image_sw + image_n * image_c * image_sh * [F]
            ↵image_sw;i < pad_h_value * image_sw + image_n * image_c * image_sh * [F]
            ↵image_sw + pad_h_value * image_sw;i++)
            res_temp[i] = 0;
    }

    if (image_dw > image_sw){
        int pad_w_value = (image_dw - image_sw) / 2;
        int j = 0;
        for (int i = 0;i < image_dh;i++){
            for (j < pad_w_value + i * image_dw;j++)
                res_temp_bak[j] = 0;
            for (j < pad_w_value + image_sw + i * image_dw;j++)
                res_temp_bak[j] = res_temp[j - pad_w_value - i * image_dw + i * [F]
            ↵image_sw];
            for (j < pad_w_value + pad_w_value + image_sw + i * image_dw;j++)
                res_temp_bak[j] = 0;
        }
    }

    free(res);
    free(res_temp);
    return res_temp_bak;
}

int main() {
    int image_sh = 1080;
    int image_sw = 1920;
    int image_dh = 112;
    int image_dw = 112;
    int is_bilinear = 0;
    bm_handle_t handle;
    int dev_id = 0;
    bm_status_t ret = bm_dev_request(&handle, dev_id);

    int image_c = 3;
    int image_n = 1;
    int output_num = 1;

    unsigned char* src_data = image_read_2(image_n, image_c, image_sh, image_
        ↵sw, image_dh, image_dw);
    float* trans_mat = (float*)malloc(output_num * 6 * sizeof(float));

    for (int i = 0; i < output_num; i++){
        trans_mat[0 + i * 6] = 3.84843f;
}

```

(续下页)

(接上页)

```

trans_mat[1 + i * 6] = -0.0248411f;
trans_mat[2 + i * 6] = 916.203f;
trans_mat[3 + i * 6] = 0.0248411;
trans_mat[4 + i * 6] = 3.84843f;
trans_mat[5 + i * 6] = 55.9748f;
}

// int* map = (int*)malloc(output_num *image_dh *image_dw * 2 * sizeof(int));
unsigned char* dst_image_tpu = (unsigned char*)malloc(output_num * image_c[F]
→* image_dh * image_dw * sizeof(unsigned char));

bmcv_affine_image_matrix matrix_image[4];
bmcv_affine_matrix * matrix = (bmcv_affine_matrix *) (trans_mat);
for (int i = 0; i < image_n; i++) {
    matrix_image[i].matrix_num = 1;
    matrix_image[i].matrix = matrix;
    matrix += 1;
}

bm_image src_img[4];
bm_image_format_ext image_format = FORMAT_BGR_PLANAR;
bm_image_data_format_ext data_type = DATA_TYPE_EXT_1N_BYTE;

for (int i = 0; i < image_n; i++) {
    bm_image_create(handle, image_sh, image_sw, image_format, data_type, src_
→img + i, NULL);
    int stride = 0;
    bm_image_get_stride(src_img[i], &stride);
    void *ptr = (void *) (src_data + 3 * stride * image_sh * i);
    bm_image_copy_host_to_device(src_img[i], (void **)(&ptr));
}

// create dst image.
bm_image* dst_img = (bm_image*)malloc(image_n * sizeof(bm_image));

for (int i = 0; i < image_n; i++) {
    bm_image_create(handle, image_dh, image_dw, image_format, data_type,[F]
→dst_img + i, NULL);
}
ret = bmcv_image_warp_affine(handle, image_n, matrix_image, src_img, dst_
→img, is_bilinear);

int size = 0;
bm_image_get_byte_size(dst_img[0], &size);
unsigned char* temp_out = (unsigned char*)malloc(output_num * size * [F]
→sizeof(unsigned char));
for (int i = 0; i < image_n; i++) {
    void *ptr = (void *) (temp_out + size * i);
    bm_image_copy_device_to_host(dst_img[i], (void **)(&ptr));
}

```

(续下页)

(接上页)

```

    memcpy(dst_image_tpu, temp_out, image_n * image_c * image_dh * image_
dw);

    for (int i = 0; i < image_n; i++){
        bm_image_destroy(&src_img[i]);
    }
    char *dst_name = "path/to/dst";
    writeBin(dst_name, temp_out, size);
    writeBin("path/to/src", src_data, image_sh * image_sw * 3);
    free(src_data);
    free(dst_img);
    free(temp_out);

    // free(map);
    free(dst_image_tpu);

}

return ret;
}

```

## 5.75 bmcv\_image\_warp\_affine\_padding

### 接口说明

- 所有的使用方式均和上述的 bmcv\_image\_warp\_affine 相同，仅仅改变了接口名字，具体的 padding zero 的接口名字如下：

### 接口形式一：

```

bm_status_t bmcv_image_warp_affine_padding(
    bm_handle_t handle,
    int image_num,
    bmcv_affine_image_matrix matrix[4],
    bm_image *input,
    bm_image *output,
    int use_bilinear);

```

### 接口形式一：

```

bm_status_t bmcv_image_warp_affine_similar_to_opencv_padding(
    bm_handle_t handle,
    int image_num,
    bmcv_affine_image_matrix matrix[4],
    bm_image *input,
    bm_image *output,
    int use_bilinear);

```

### 代码示例说明

- 同 bmcv\_image\_warp\_affine 接口使用方式相同，只需

要将接口名字换成 bmcv\_image\_warp\_affine\_padding 或 bmcv\_image\_warp\_affine\_similar\_to\_opencv\_padding 即可。

## 5.76 bmcv\_image\_warp\_perspective

该接口实现图像的透射变换，又称投影变换或透视变换。透射变换将图片投影到一个新的视平面，是一种二维坐标  $(x_0, y_0)$  到二维坐标  $(x, y)$  的非线性变换，该接口的实现是针对输出图像的每一个像素点坐标得到对应输入图像的坐标，然后构成一幅新的图像，其数学表达式形式如下：

$$\begin{cases} x' = a_1x + b_1y + c_1 \\ y' = a_2x + b_2y + c_2 \\ w' = a_3x + b_3y + c_3 \\ x_0 = x'/w' \\ y_0 = y'/w' \end{cases}$$

对应的齐次坐标矩阵表示形式为：

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{cases} x_0 = x'/w' \\ y_0 = y'/w' \end{cases}$$

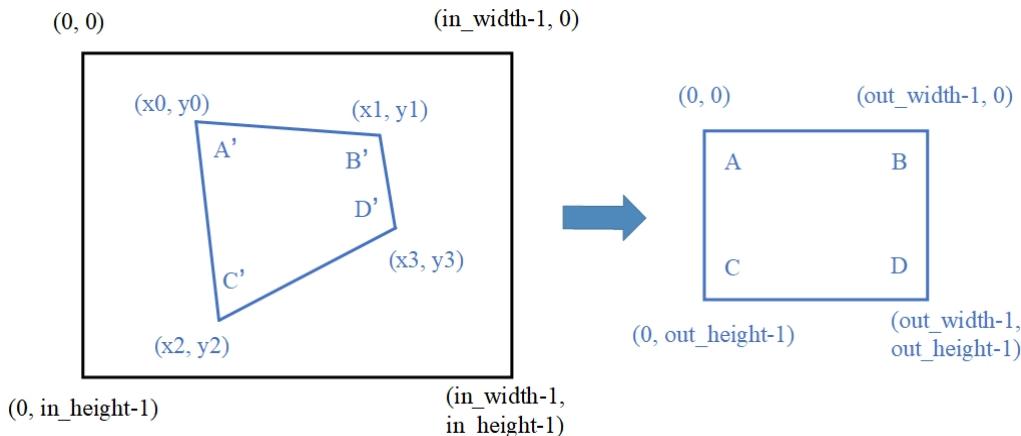
坐标变换矩阵是一个 9 点的矩阵（通常  $c_3 = 1$ ），利用该变换矩阵可以从输出图像坐标推导出对应的输入原图坐标，该变换矩阵可以通过输入输出图像对应的 4 个点的坐标来获取。

为了更方便地完成透射变换，该库提供了两种形式的接口供用户使用：一种是用户提供变换矩阵给接口作为输入；另一种接口是提供输入图像中 4 个点的坐标作为输入，适用于将一个不规则的四边形透射为一个与输出大小相同的矩形，如下图所示，可以将输入图像 A' B' C' D' 映射为输出图像 ABCD，用户只需要提供输入图像中 A'、B'、C'、D' 四个点的坐标即可，该接口内部会根据这四个的坐标和输出图像四个顶点的坐标自动计算出变换矩阵，从而完成该功能。

**接口形式一：**

```
bm_status_t bmcv_image_warp_perspective(
    bm_handle_t handle,
    int image_num,
    bmcv_perspective_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0);
```

其中，`bmcv_perspective_matrix` 定义了一个坐标变换矩阵，其顺序为 `float m[9] = {a1, b1, c1, a2, b2, c2, a3, b3, c3}`。而 `bmcv_perspective_image_matrix` 定义了一张图片里面有几个变换矩阵，可以实现对一张图片里的多个小图进行透射变换。



```

typedef struct bmcv_perspective_matrix_s{
    float m[9];
} bmcv_perspective_matrix;

typedef struct bmcv_perspective_image_matrix_s{
    bmcv_perspective_matrix *matrix;
    int matrix_num;
} bmcv_perspective_image_matrix;

```

### 接口形式二：

```

bm_status_t bmcv_image_warp_perspective_with_coordinate(
    bm_handle_t handle,
    int image_num,
    bmcv_perspective_image_coordinate coord[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0);

```

其中，`bmcv_perspective_coordinate` 定义了四边形四个顶点的坐标，按照左上、右上、左下、右下的顺序存储。而 `bmcv_perspective_image_coordinate` 定义了一张图片里面有几组四边形的坐标，可以实现对一张图片里的多个小图进行透射变换。

```

typedef struct bmcv_perspective_coordinate_s{
    int x[4];
    int y[4];
} bmcv_perspective_coordinate;

typedef struct bmcv_perspective_image_coordinate_s{
    bmcv_perspective_coordinate *coordinate;
    int coordinate_num;
} bmcv_perspective_image_coordinate;

```

### 接口形式三：

```
bm_status_t bmcv_image_warp_perspective_similar_to_opencv(
    bm_handle_t handle,
    int image_num,
    bmcv_perspective_image_matrix matrix[4],
    bm_image* input,
    bm_image* output,
    int use_bilinear = 0);
```

本接口中 bmcv\_perspective\_image\_matrix 定义的变换矩阵与 opencv 的 warpPerspective 接口要求输入的变换矩阵相同，且与接口一中同名结构体定义的矩阵互为逆矩阵，其余参数与接口一相同。

```
typedef struct bmcv_perspective_matrix_s{
    float m[9];
} bmcv_perspective_matrix;

typedef struct bmcv_perspective_image_matrix_s{
    bmcv_perspective_matrix *matrix;
    int matrix_num;
} bmcv_perspective_image_matrix;
```

## 参数说明

- `bm_handle_t handle`  
输入参数。输入的 `bm_handle` 句柄。
- `int image_num`  
输入参数。输入图片数，最多支持 4。
- `bmcv_perspective_image_matrix matrix[4]`  
输入参数。每张图片对应的变换矩阵数据结构，最多支持 4 张图片。
- `bmcv_perspective_image_coordinate coord[4]`  
输入参数。每张图片对应的四边形坐标信息，最多支持 4 张图片。
- `bm_image* input`  
输入参数。输入 `bm_image`，对于 1N 模式，最多 4 个 `bm_image`，对于 4N 模式，最多一个 `bm_image`。
- `bm_image* output`  
输出参数。输出 `bm_image`，外部需要调用 `bmcv_image_create` 创建，建议用户调用 `bmcv_image_attach` 来分配 device memory。如果用户不调用 `attach`，则内部分配 device memory。对于输出 `bm_image`，其数据类型和输入一致，即输入是 4N 模式，则输出也是 4N 模式，输入 1N 模式，输出也是 1N 模式。所需要的 `bm_image` 大小是所有图片的变换矩阵之和。比如输入 1 个 4N 模式的 `bm_image`，4 张图片的变换矩阵数目为 **【3,0,13,5】**，则共有变换矩阵  $3+0+13+5=21$ ，由于输出是 4N 模式，则需要  $(21+4-1)/4=6$  个 `bm_image` 的输出。

- int use\_bilinear

输入参数。是否使用 bilinear 进行插值，若为 0 则使用 nearest 插值，若为 1 则使用 bilinear 插值，默认使用 nearest 插值。选择 nearest 插值的性能会优于 bilinear，因此建议首选 nearest 插值，除非对精度有要求时可选择使用 bilinear 插值。

#### 返回值说明:

- BM\_SUCCESS: 成功
- 其他: 失败

#### 注意事项

1. 该接口要求输出图像的所有坐标点都能在输入的原图中找到对应的坐标点，不能超出原图大小，建议优先使用接口二，可以自动满足该条件。
2. 该接口所支持的 image\_format 包括：

num	image_format
1	FORMAT_BGR_PLANAR
2	FORMAT_RGB_PLANAR

3. 该接口所支持的 data\_type 包括：

num	data_type
1	DATA_TYPE_EXT_1N_BYTE
2	DATA_TYPE_EXT_4N_BYTE

4. 该接口的输入以及输出 bm\_image 均支持带有 stride。
5. 要求该接口输入 bm\_image 的 width、height、image\_format 以及 data\_type 必须保持一致。
6. 要求该接口输出 bm\_image 的 width、height、image\_format、data\_type 以及 stride 必须保持一致。

#### 示例代码

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"

#define BM_MIN(x, y) (((x)) < ((y)) ? (x) : (y))
#define BM_MAX(x, y) (((x)) > ((y)) ? (x) : (y))

#define NO_USE 0
```

(续下页)

(接上页)

```

#define MAX_INT (float)(pow(2, 31) - 2)
#define MIN_INT (float)(1 - pow(2, 31))
#define UNUSED(x) (void)(x)

static int image_sh = 500;
static int image_sw = 500;
static int image_dh = 200;
static int image_dw = 200;
static int use_bilinear = 0;
static bm_handle_t handle;

static int writeBin(const char* path, void* output_data, int size)
{
    int len = 0;
    FILE* fp_dst = fopen(path, "wb+");

    if (fp_dst == NULL) {
        perror("Error opening file\n");
        return -1;
    }

    len = fwrite((void*)output_data, 1, size, fp_dst);
    if (len < size) {
        printf("file size = %d is less than required bytes = %d\n", len, size);
        return -1;
    }

    fclose(fp_dst);
    return 0;
}

static void my_get_perspective_transform(int* sx, int* sy, int dw, int dh, float* F
matrix) {
    int A = sx[3] + sx[0] - sx[1] - sx[2];
    int B = sy[3] + sy[0] - sy[1] - sy[2];
    int C = sx[2] - sx[3];
    int D = sy[2] - sy[3];
    int E = sx[1] - sx[3];
    int F = sy[1] - sy[3];
    matrix[8] = 1;
    matrix[7] = ((float)(A * F - B * E) / (float)(dh)) / (float)(C * F - D * E);
    matrix[6] = ((float)(A * D - B * C) / (float)(dw)) / (float)(D * E - C * F);
    matrix[0] = (matrix[6] * dw * sx[1] + sx[1] - sx[0]) / dw;
    matrix[1] = (matrix[7] * dh * sx[2] + sx[2] - sx[0]) / dh;
    matrix[2] = sx[0];
    matrix[3] = (matrix[6] * dw * sy[1] + sy[1] - sy[0]) / dw;
    matrix[4] = (matrix[7] * dh * sy[2] + sy[2] - sy[0]) / dh;
    matrix[5] = sy[0];
}

```

(续下页)

(接上页)

```

unsigned char* read_pixel_data(const char* filename, size_t* data_size) {
    FILE* file = fopen(filename, "rb");
    if (!file) {
        fprintf(stderr, "Can not open: %s\n", filename);
        return NULL;
    }
    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);

    if (file_size <= 0) {
        fprintf(stderr, "invalid file size: %ld\n", file_size);
        fclose(file);
        return NULL;
    }

    unsigned char* res = (unsigned char*)malloc(file_size);
    if (!res) {
        fprintf(stderr, "malloc failed\n");
        fclose(file);
        return NULL;
    }

    size_t bytes_read = fread(res, 1, file_size, file);
    if (bytes_read != (size_t)file_size) {
        fprintf(stderr, "Read file error: %s (expect %ld bytes, actual %zu bytes)\n",
                filename, file_size, bytes_read);
        free(res);
        fclose(file);
        return NULL;
    }

    fclose(file);
    *data_size = (size_t)file_size;
    return res;
}

static bm_status_t src_data_generation(int i, int* coordinate, float* trans_mat, F
→float* tensor_S) {
    int border_x1 = rand() % image_sw;
    int border_x2 = rand() % image_sw;
    while (border_x1 == border_x2) border_x2 = rand() % image_sw;
    int border_y1 = rand() % image_sh;
    int border_y2 = rand() % image_sh;
    while (border_y1 == border_y2) border_y2 = rand() % image_sh;
    int x_min = BM_MIN(border_x1, border_x2);
    int x_max = BM_MAX(border_x1, border_x2);
    int y_min = BM_MIN(border_y1, border_y2);
    int y_max = BM_MAX(border_y1, border_y2);

    int x[4], y[4];

```

(续下页)

(接上页)

```

int sx[4], sy[4];
int idx = rand() % 4;
x [0] = x_min + rand() % (x_max - x_min);
y [0] = y_min;
x [1] = x_max;
y [1] = y_min + rand() % (y_max - y_min);
x [2] = x_max - rand() % (x_max - x_min);
y [2] = y_max;
x [3] = x_min;
y [3] = y_max - rand() % (y_max - y_min);
sx [0] = x[(0 + idx) % 4];
sy [0] = y[(0 + idx) % 4];
sx [1] = x[(1 + idx) % 4];
sy [1] = y[(1 + idx) % 4];
sx [2] = x[(3 + idx) % 4];
sy [2] = y[(3 + idx) % 4];
sx [3] = x[(2 + idx) % 4];
sy [3] = y[(2 + idx) % 4];
printf("src coordinate: (%d %d) (%d %d) (%d %d)\n", sx[0], sy[0], sx[1], sy[1], sx[2], sy[2], sx[3], sy[3]);

coordinate[0 + i * 8] = sx[0];
coordinate[1 + i * 8] = sx[1];
coordinate[2 + i * 8] = sx[2];
coordinate[3 + i * 8] = sx[3];
coordinate[4 + i * 8] = sy[0];
coordinate[5 + i * 8] = sy[1];
coordinate[6 + i * 8] = sy[2];
coordinate[7 + i * 8] = sy[3];

float matrix_cv[9];
my_get_perspective_transform(sx, sy, image_dw-1, image_dh-1, matrix_cv);
trans_mat[0 + i * 9] = matrix_cv[0];
trans_mat[1 + i * 9] = matrix_cv[1];
trans_mat[2 + i * 9] = matrix_cv[2];
trans_mat[3 + i * 9] = matrix_cv[3];
trans_mat[4 + i * 9] = matrix_cv[4];
trans_mat[5 + i * 9] = matrix_cv[5];
trans_mat[6 + i * 9] = matrix_cv[6];
trans_mat[7 + i * 9] = matrix_cv[7];
trans_mat[8 + i * 9] = matrix_cv[8];

printf("trans_mat[0 + i * 9] = %f\n", trans_mat[0 + i * 9]);
printf("trans_mat[1 + i * 9] = %f\n", trans_mat[1 + i * 9]);
printf("trans_mat[2 + i * 9] = %f\n", trans_mat[2 + i * 9]);
printf("trans_mat[3 + i * 9] = %f\n", trans_mat[3 + i * 9]);
printf("trans_mat[4 + i * 9] = %f\n", trans_mat[4 + i * 9]);
printf("trans_mat[5 + i * 9] = %f\n", trans_mat[5 + i * 9]);
printf("trans_mat[6 + i * 9] = %f\n", trans_mat[6 + i * 9]);
printf("trans_mat[7 + i * 9] = %f\n", trans_mat[7 + i * 9]);
printf("trans_mat[8 + i * 9] = %f\n", trans_mat[8 + i * 9]);

```

(续下页)

(接上页)

```

float* tensor_SX = tensor_S;
float* tensor_SY = tensor_SX + image_dh * image_dw;
for (int y = 0; y < image_dh; y++) {
    for (int x = 0; x < image_dw; x++) {
        float dx = tensor_SX[y * image_dw + x] * trans_mat[0 + i * 9] +
            tensor_SY[y * image_dw + x] * trans_mat[1 + i * 9] + trans_mat[2 + F
i * 9];
        float dy = tensor_SX[y * image_dw + x] * trans_mat[3 + i * 9] +
            tensor_SY[y * image_dw + x] * trans_mat[4 + i * 9] + trans_mat[5 F
+ i * 9];
        float dz = tensor_SX[y * image_dw + x] * trans_mat[6 + i * 9] +
            tensor_SY[y * image_dw + x] * trans_mat[7 + i * 9] + trans_mat[8 F
+ i * 9];

        dx = dx / dz;
        dy = dy / dz;

        if (dx < MIN_INT || dx > MAX_INT || dy < MIN_INT || dy > MAX_
INT || fabs(dz) == 0) {
            printf("----- the input data is not legal! ----- \n");
            return BM_ERR_DATA;
        }
    }
}
return BM_SUCCESS;
}

int main() {
    int dev_id = 0;
    bm_status_t ret = bm_dev_request(&handle, dev_id);
    if (ret != BM_SUCCESS) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    int matrix_num[4] = {1, 1, 1, 1};
    int image_n = 1;

    int output_num = 0;
    for (int i = 0; i < image_n; i++) {
        output_num += matrix_num[i];
    }

    const char* filename = "/home/linaro/output.bin";
    size_t data_size = 0;
    unsigned char* src_data = read_pixel_data(filename, &data_size);
    if (src_data) {
        printf("Get %zu bytes\n", data_size);
    }
    float* trans_mat = (float*) malloc(output_num * 9 * sizeof(float));
    int* coordinate = (int*) malloc(output_num * 8 * sizeof(int));
}

```

(续下页)

(接上页)

```

float* tensor_S = (float*) malloc(image_dh * image_dw * 2 * sizeof(float));
float* tensor_SX = tensor_S;
float* tensor_SY = tensor_SX + image_dh * image_dw;
for (int y = 0; y < image_dh; y++) {
    for (int x = 0; x < image_dw; x++) {
        tensor_SX[y * image_dw + x] = (float)x;
        tensor_SY[y * image_dw + x] = (float)y;
    }
}

for (int i = 0; i < output_num; i++) {
    ret = src_data_generation(i, coordinate, trans_mat, tensor_S);
    while (BM_ERR_DATA == ret)
        ret = src_data_generation(i, coordinate, trans_mat, tensor_S);
}

bmvc_perspective_image_matrix matrix_image[4];
bmvc_perspective_matrix* matrix = (bmvc_perspective_matrix*)(trans_mat);
for (int i = 0; i < image_n; i++) {
    matrix_image[i].matrix_num = matrix_num[i];
    matrix_image[i].matrix = matrix;
    matrix += matrix_num[i];
}

bm_image src_img[4];
bm_image_format_ext image_format = FORMAT_BGR_PLANAR;
bm_image_data_format_ext data_type = DATA_TYPE_EXT_1N_BYTE;
int in_image_num = image_n;
int out_image_num = output_num;

for (int i = 0; i < in_image_num; i++) {
    bm_image_create(
        handle, image_sh, image_sw, image_format, data_type, src_img + i, [F]
        ↵NULL);
    int stride = 0;
    // debug
    bm_image_get_stride(src_img[i], &stride);
    void* ptr = (void*)(src_data + 3 * stride * image_sh * i);
    bm_image_copy_host_to_device(src_img[i], (void**)(&ptr));
}

// create dst image.
bm_image* dst_img = (bm_image*)malloc(out_image_num * sizeof(bm_image));

for (int i = 0; i < out_image_num; i++) {
    bm_image_create(handle, image_dh, image_dw, image_format, data_type, [F]
    ↵dst_img + i, NULL);
}
printf("No coordinate\n");
ret = bmvc_image_warp_perspective(handle, image_n, matrix_image, src_img, [F]
    ↵dst_img, use_bilinear);

```

(续下页)

(接上页)

```

int size = 0;
bm_image_get_byte_size(dst_img[0], &size);
unsigned char* temp_out = (unsigned char*)malloc(out_image_num * size * sizeof(unsigned char));

for (int i = 0; i < out_image_num; i++) {
    void *ptr = (void*)(temp_out + size * i);
    bm_image_copy_device_to_host(dst_img[i], (void**)&ptr);
}

char *dst_name = "/home/linaro/output_warp.bin";
writeBin(dst_name, temp_out, size);

free(temp_out);
free(dst_img);
free(src_data);

free(trans_mat);
free(coordinate);
free(tensor_S);

bm_dev_free(handle);

return ret;
}

```

## 5.77 bmcv\_width\_align

将内存区域的数据按照指定的宽的布局转换并存储到目标内存区域

**处理器型号支持:**

该接口支持 BM1684X/BM1688/BM1690。

**接口形式:**

```

bm_status_t bmcv_width_align(
    bm_handle_t handle,
    bm_image input,
    bm_image output)

```

**参数说明:**

- bm\_handle\_t handle

输入参数。bm\_handle 句柄。

- bm\_image input

输入参数。输入图像的 bm\_image, bm\_image 需要外部调用 bmcv\_image\_create 创建。image 内存可以使用 bm\_image\_alloc\_dev\_mem 或者

`bm_image_copy_host_to_device` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。

- `bm_image output`

输出参数。输出 `bm_image`, `bm_image` 需要外部调用 `bmcv_image_create` 创建。`image` 内存可以通过 `bm_image_alloc_dev_mem` 来开辟新的内存，或者使用 `bmcv_image_attach` 来 attach 已有的内存。

**返回值说明:**

- `BM_SUCCESS`: 成功
- 其他: 失败

**格式支持:**

该接口目前支持以下 `image_format`:

num	input image_format	output image_format
1	FORMAT_BGR_PACKED	FORMAT_BGR_PACKED
2	FORMAT_BGR_PLANAR	FORMAT_BGR_PLANAR
3	FORMAT_RGB_PACKED	FORMAT_RGB_PACKED
4	FORMAT_RGB_PLANAR	FORMAT_RGB_PLANAR
5	FORMAT_RGBP_SEPARATE	FORMAT_RGBP_SEPARATE
6	FORMAT_BGRP_SEPARATE	FORMAT_BGRP_SEPARATE
7	FORMAT_GRAY	FORMAT_GRAY
8	FORMAT_YUV420P	FORMAT_YUV420P
9	FORMAT_YUV422P	FORMAT_YUV422P
10	FORMAT_YUV444P	FORMAT_YUV444P
11	FORMAT_NV12	FORMAT_NV12
12	FORMAT_NV21	FORMAT_NV21
13	FORMAT_NV16	FORMAT_NV16
14	FORMAT_NV61	FORMAT_NV61
15	FORMAT_NV24	FORMAT_NV24

目前支持以下 `data_type`:

num	data_type
1	DATA_TYPE_EXT_1N_BYTE
2	DATA_TYPE_EXT_FLOAT32
3	DATA_TYPE_EXT_FP16

**注意事项:**

- 1、在调用该接口之前必须确保输入的 `image` 内存已经申请。
- 2、`input output` 的 `image_format` 以及 `data_type` 必须相同。

## 代码示例：

```
#include "bmvc_api_ext_c.h"
#include "stdio.h"
#include "stdlib.h"
#include <sys/time.h>
#include <time.h>
#include <pthread.h>
#include "bmvc_internal.h"

static void readBin(const char* path, unsigned char* input_data, int size)
{
    FILE *fp_src = fopen(path, "rb");
    if (fread((void *)input_data, 1, size, fp_src) < (unsigned int)size) {
        printf("file size is less than %d required bytes\n", size);
    };
    fclose(fp_src);
}

static void write_bin(const char *output_path, unsigned char *output_data, int size)
{
    FILE *fp_dst = fopen(output_path, "wb");

    if (fp_dst == NULL) {
        printf("unable to open output file %s\n", output_path);
        return;
    }

    if(fwrite(output_data, sizeof(unsigned char), size, fp_dst) != 0) {
        printf("write image success\n");
    }
    fclose(fp_dst);
}

static int bmvc_width_align_cmp(unsigned char *p_exp, unsigned char *p_got, int F
                                ↵count) {
    int ret = 0;
    for (int j = 0; j < count; j++) {
        if (p_exp[j] != p_got[j]) {
            printf("error: when idx=%d, exp=%d but got=%d\n", j, (int)p_exp[j], F
                  ↵(int)p_got[j]);
            return -1;
        }
    }
    return ret;
}

int main() {
    int image_w = 1920;
    int image_h = 1080;

    int default_stride[3] = {0};
```

(续下页)

(接上页)

```

int          src_stride[3]    = {0};
int          dst_stride[3]    = {0};
bm_image_format_ext image_format      = FORMAT_GRAY;
bm_image_data_format_ext data_type     = DATA_TYPE_EXT_1N_
BYTE;
int          raw_size        = 0;

default_stride[0] = image_w;
src_stride[0]     = image_w + rand() % 16;
dst_stride[0]     = image_w + rand() % 16;

raw_size = image_h * image_w;
unsigned char* raw_image = (unsigned char*)malloc(image_w * image_h * [F
sizeof(unsigned char));
unsigned char* dst_data = (unsigned char*)malloc(image_w * image_h * [F
sizeof(unsigned char));
unsigned char* src_image = (unsigned char*)malloc(src_stride[0] * image_h * [F
sizeof(unsigned char));
unsigned char* dst_image = (unsigned char*)malloc(dst_stride[0] * image_h * [F
sizeof(unsigned char));

const char* input_path = "/home/linaro/gray.bin";
const char* output_path = "/home/linaro/gray_out.bin";
readBin(input_path, raw_image, raw_size);

int ret = 0;
bm_handle_t handle;
ret = bm_dev_request(&handle, 0);
bm_image src_img, dst_img;

// calculate use reference for compare.
unsigned char *src_s_offset;
unsigned char *src_d_offset;
for (int i = 0; i < image_h; i++) {
    src_s_offset = raw_image + i * default_stride[0];
    src_d_offset = src_image + i * src_stride[0];
    memcpy(src_d_offset, src_s_offset, image_w);
}

// create source image.
bm_image_create(handle, image_h, image_w, image_format, data_type, &src_
img, src_stride);
bm_image_create(handle, image_h, image_w, image_format, data_type, &dst_
img, dst_stride);

int size[3] = {0};
bm_image_get_byte_size(src_img, size);
u8 *host_ptr_src[] = {src_image,
                     src_image + size[0],
                     src_image + size[0] + size[1]};
bm_image_get_byte_size(dst_img, size);

```

(续下页)

(接上页)

```
u8 *host_ptr_dst[] = {dst_image,
                      dst_image + size[0],
                      dst_image + size[0] + size[1]};

ret = bm_image_copy_host_to_device(src_img, (void **)(host_ptr_src));
if (ret != BM_SUCCESS) {
    printf("test data prepare failed");
    return ret;
}

ret = bmcv_width_align(handle, src_img, dst_img);
if (ret != BM_SUCCESS) {
    printf("bmcv width align failed");
    return ret;
}
ret = bm_image_copy_device_to_host(dst_img, (void **)(host_ptr_dst));
if (ret != BM_SUCCESS) {
    printf("test data copy back failed");
    return ret;
}
bm_image_destroy(&src_img);
bm_image_destroy(&dst_img);

unsigned char *dst_s_offset;
unsigned char *dst_d_offset;
for (int i = 0; i < image_h; i++) {
    dst_s_offset = dst_image + i * dst_stride[0];
    dst_d_offset = dst_data + i * default_stride[0];
    memcpy(dst_d_offset, dst_s_offset, image_w);
}
write_bin(output_path, dst_data, raw_size);

// compare.
int cmp_res = bmcv_width_align_cmp(raw_image, dst_data, raw_size);
if (cmp_res != 0) {
    printf("cv_width_align comparing failed\n");
    ret = BM_ERR_FAILURE;
    return ret;
}
printf("-----[TEST WIDTH ALIGN] ALL TEST PASSED!\n");
free(raw_image);
free(dst_data);
free(src_image);
free(dst_image);
return 0;
}
```

## 5.78 bmcv\_ive\_bersen

### 【描述】

该 API 使用 ive 硬件资源，实现利用 Bernsen 二值法，对图像进行二值化求解，实现图像的二值化。

### 【语法】

```

1 bm_status_t bmcv_ive_bernsen(
2     bm_handle_t      handle,
3     bm_image        input,
4     bm_image        output,
5     bmcv_ive_bernsen_attr attr);

```

### 【参数】

表 5.51: bmcv\_ive\_bernsen 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	16x16~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

### 【数据类型说明】

【说明】 定义 Bernsen 阈值计算的不同方式。

```

1 typedef enum bmcv_ive_bernsen_mode_e{
2     BM_IVE_BERNSEN_NORMAL = 0x0,
3     BM_IVE_BERNSEN_THRESH = 0x1,

```

(续下页)

(接上页)

```

4     BM_IVE_BERNSEN_PAPER = 0x2,
5 } bmcv_ive_bernsen_mode;

```

**【说明】** 定义 Bernsen 二值法的控制参数。

```

1 typedef struct bmcv_ive_bernsen_attr_s{
2     bmcv_ive_bernsen_mode en_mode;
3     unsigned char u8_win_size;
4     unsigned char u8_thr;
5     unsigned char u8_contrast_threshold;
6 } bmcv_ive_bernsen_attr;

```

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

- 输入输出图像的 width 都需要 16 对齐。

### 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    bmcv_ive_bernsen_mode mode = BM_IVE_BERNSEN_NORMAL;
    int winsize = 5;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src, dst;
    int stride[4];

```

(续下页)

(接上页)

```

// config setting
bmcv_ive_bernsen_attr attr;
memset(&attr, 0, sizeof(bmcv_ive_bernsen_attr));

attr.en_mode = mode;
attr.u8_thr = 128;
attr.u8_contrast_threshold = 15;
attr.u8_win_size = winsize;

// calc ive image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src, stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
dst, stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
// read image data from input files
int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
    (void *)((unsigned char *)input_data + image_byte_size[0]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_bernsen(handle, src, dst, attr);

unsigned char *ive_add_res = (unsigned char *)malloc(width * height * [F]
sizeof(unsigned char));
memset(ive_add_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_add_res);
FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_add_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

```

(续下页)

(接上页)

```

bm_image_destroy(&src);
bm_image_destroy(&dst);

return 0;
}

```

## 5.79 bmcv\_ive\_16bit\_to\_8bit

### 【描述】

该 API 使用 ive 硬件资源，创建 16bit 图像数据到 8bit 图像数据的线性转化任务。

### 【语法】

```

1 bm_status_t bmcv_ive_16bit_to_8bit(
2     bm_handle_t           handle,
3     bm_image              input,
4     bm_image              output,
5     bmcv_ive_16bit_to_8bit_attr attr);

```

### 【参数】

表 5.52: bmcv\_ive\_16bit\_to\_8bit 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_S16 DATA_TYPE_EXT_U16	16x16~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE DATA_TYPE_EXT_1N_BYTE_SIGNED	同 input

## 【数据类型说明】

【说明】定义 16bit 图像数据到 8bit 图像数据的转换模式。

```

1 typedef enum bmcv_ive_16bit_to_8bit_mode_e{
2     BM_IVE_S16_TO_S8 = 0x0,
3     BM_IVE_S16_TO_U8_ABS = 0x1,
4     BM_IVE_S16_TO_U8_BIAS = 0x2,
5     BM_IVE_U16_TO_U8 = 0x3,
6 } bmcv_ive_16bit_to_8bit_mode;

```

成员名称	描述
BM_IVE_S16_TO_S8	S16 数据到 S8 数据的线性变换。
BM_IVE_S16_TO_U8_ABS	S16 数据线性变化到 S8 数据后取绝对值得到 S8 数据。
BM_IVE_S16_TO_U8_BIAS	S16 数据线性变化到 S8 数据且平移后截断得到 U8 数据。
BM_IVE_U16_TO_U8	U16 数据线性变化到 U8 数据。

【说明】定义 16bit 图像数据到 8bit 图像数据的转化控制参数。

```

1 typedef struct bmcv_ive_16bit_to_8bit_attr_s{
2     bmcv_ive_16bit_to_8bit_mode mode;
3     unsigned short u16_denominator;
4     unsigned char u8_numerator;
5     signed char s8_bias;
6 } bmcv_ive_16bit_to_8bit_attr;

```

成员名称	描述
mode	16bit 数据到 8bit 数据的转换模式。
u16_denominator	线性变换中的分母，取值范围：{max{1, u8Numerator}, 65535}。
u8_numerator	线性变化中的分子，取值范围：[0, 255]。
s8_bias	线性变化中的平移项，取值范围：[-128, 127]。

## 【注意】

- $u8Numerator \leq u16Denominator$  且  $u16Denominator \neq 0$  ,

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。

2. 可配置 4 种模式，具体参考 bmcv\_ive\_16bit\_to\_8bit\_mode 说明。

3. 计算公式：

- BM\_IVE\_S16\_TO\_S8:

$$I_{\text{out}}(x, y) = \begin{cases} -128 & (\frac{a}{b}I(x, y) < -128) \\ \frac{a}{b}I(x, y) & (-128 \leq \frac{a}{b}I(x, y) \leq 127) \\ 127 & (\frac{a}{b}I(x, y) > 127) \end{cases}$$

- BM\_IVE\_S16\_TO\_U8\_ABS:

$$I_{\text{out}}(x, y) = \begin{cases} \left| \frac{a}{b}I(x, y) \right| & \left( \left| \frac{a}{b}I(x, y) \right| \leq 255 \right) \\ 255 & \left( \left| \frac{a}{b}I(x, y) \right| \geq 255 \right) \end{cases}$$

- BM\_IVE\_S16\_TO\_U8\_BIAS:

$$I_{\text{out}}(x, y) = \begin{cases} 0 & (\frac{a}{b}I(x, y) + bais < 0) \\ \frac{a}{b}I(x, y) + bais & (0 \leq \frac{a}{b}I(x, y) + bais \leq 255) \\ 255 & (\frac{a}{b}I(x, y) + bais > 255) \end{cases}$$

- BM\_IVE\_U16\_TO\_U8:

$$I_{\text{out}}(x, y) = \begin{cases} 0 & (\frac{a}{b}I(x, y) < 0) \\ \frac{a}{b}I(x, y) & (0 \leq \frac{a}{b}I(x, y) \leq 255) \\ 255 & (\frac{a}{b}I(x, y) > 255) \end{cases}$$

其中， $I(x, y)$  对应 input， $I_{\text{out}}(x, y)$  对应 output，a、b 和 bais 分别对应 attr 的 u8\_numerator、u16\_denominator、s8\_bias。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

static int readBin(const char* path, void* input_data)
{
    int len;
```

(续下页)

(接上页)

```

int size;
FILE* fp_src = fopen(path, "rb");

if (fp_src == NULL) {
    perror("Error opening file\n");
    return -1;
}

fseek(fp_src, 0, SEEK_END);
size = ftell(fp_src);
fseek(fp_src, 0, SEEK_SET);

len = fread((void*)input_data, 1, size, fp_src);
if (len < size) {
    printf("file size = %d is less than required bytes = %d\n", len, size);
    return -1;
}

fclose(fp_src);
return 0;
}

int main() {
    int dev_id = 0;
    bmcv_ive_16bit_to_8bit_mode mode = BM_IVE_S16_TO_S8;
    unsigned char u8Numerator = 41;
    unsigned short u16Denominator = 18508;
    signed char s8Bias = 0;
    int height = 1080, width = 1920;
    bm_image_data_format_ext srcDtype = DATA_TYPE_EXT_S16;
    bm_image_data_format_ext dstDtype = DATA_TYPE_EXT_1N_BYTE_
    ↵SIGNED;
    char *src_name = "path/to/src";
    char *ref_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src, dst;
    int srcStride[4], dstStride[4];

    // config setting
    bmcv_ive_16bit_to_8bit_attr attr;
    attr.mode = mode;
    attr.u16_denominator = u16Denominator;
    attr.u8_numerator = u8Numerator;
    attr.s8_bias = s8Bias;
}

```

(续下页)

(接上页)

```
int data_size = 1;
srcStride[0] = align_up(width, 16) * data_size;
dstStride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, FORMAT_GRAY, srcDtype, &src,[F
↳srcStride);
bm_image_create(handle, height, width, FORMAT_GRAY, dstDtype, &dst,[F
↳dstStride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

unsigned char* input_data = (unsigned char*)malloc(height * width *[F
↳sizeof(unsigned char));
readBin(src_name, input_data);
bm_image_copy_host_to_device(src, (void**)&input_data);

ret = bmcv_ive_16bit_to_8bit(handle, src, dst, attr);

int image_byte_size[4] = {0};
bm_image_get_byte_size(dst, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F
↳+ image_byte_size[3];
unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
memset(output_ptr, 0, sizeof(byte_size));

void* out_ptr[4] = {((void*)output_ptr,
                    ((void*)((char*)output_ptr + image_byte_size[0])),
                    ((void*)((char*)output_ptr + image_byte_size[0] + image_byte_
↳size[1])),
                    ((void*)((char*)output_ptr + image_byte_size[0] + image_byte_
↳size[1] + image_byte_size[2]))};

ret = bm_image_copy_device_to_host(dst, (void**)out_ptr);

FILE *fp_dst = fopen(ref_name, "wb");
if (fwrite((void*)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
    printf("file size is less than %d required bytes\n", byte_size);
}
fclose(fp_dst);

free(input_data);
free(output_ptr);

bm_image_destroy(&src);
bm_image_destroy(&dst);

return 0;
}
```

## 5.80 bmcv\_ive\_dma

### 【描述】

该 API 使用 ive 硬件资源，创建直接内存访问任务，支持快速拷贝、间隔拷贝；可实现数据从一块内存快速拷贝到另一块内存，或者从一块内存有规律的拷贝一些数据到另一块内存。

### 【语法】

```

1 bm_status_t bmcv_ive_dma(
2     bm_handle_t           handle,
3     bm_image              input,
4     bm_image              output,
5     bmcv_ive_dma_mode     dma_mode,
6     bmcv_ive_interval_dma_attr * attr);

```

### 【参数】

表 5.53: bmcv\_ive\_dma 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
dma_mode	输入	定义 DMA copy 操作模式。
attr	输入	DMA 在间隔拷贝模式下需要的控制参数结构体，直接拷贝可以为空。

【分辨率】: 32x1 ~ 1920x1080

### 【数据类型说明】

【说明】 定义 DMA 操作模式。

```

1 typedef enum bmcv_ive_dma_mode_e {
2     IVE_DMA_DIRECT_COPY = 0x0,
3     IVE_DMA_INTERVAL_COPY = 0x1
4 } bmcv_ive_dma_mode;

```

成员名称	描述
IVE_DMA_DIRECT_COPY	直接快速拷贝模式。
IVE_DMA_INTERVAL_COPY	间隔拷贝模式。

**【说明】** 定义 DMA 间隔拷贝需要的控制信息。

```

1 typedef struct bmcv_ive_interval_dma_attr_s {
2     unsigned char hor_seg_size;
3     unsigned char elem_size;
4     unsigned char ver_seg_rows;
5 } bmcv_ive_interval_dma_attr;
```

成员名称	描述
hor_seg_size	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}。
elem_size	仅间隔拷贝模式使用, 分割的每一段中前 elem_size 为有效拷贝字段。 取值范围: [1, hor_seg_size - 1]。
ver_seg_rows	仅间隔拷贝模式使用, 将每 ver_seg_rows 行中第一行数据分割为 hor_seg_size 大小的段, 拷贝每段的前 elem_size 大小的字节。 取值范围: [1, min{65535/srcStride, srcHeight}]。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

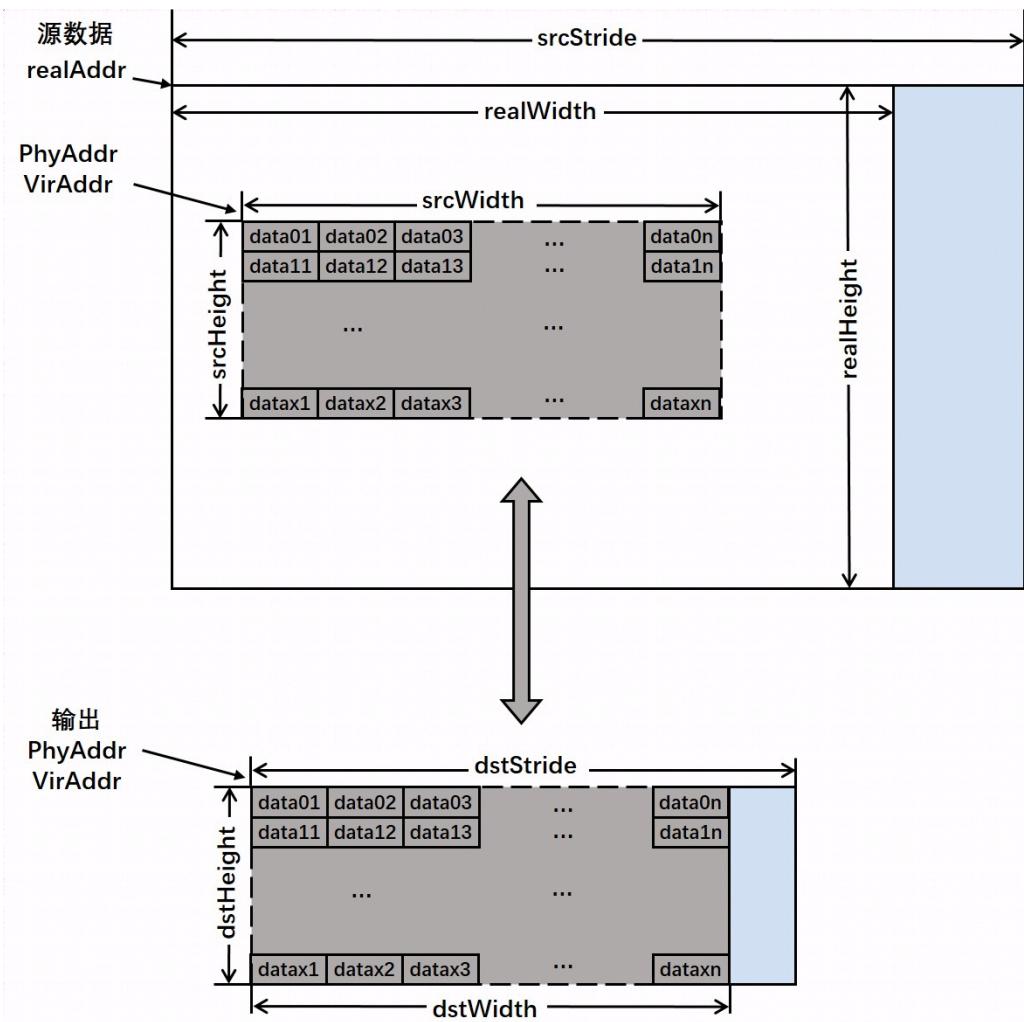
### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 模式是指 IVE\_DMA\_DIRECT\_COPY 和 IVE\_DMA\_INTERVAL\_COPY 模式;
3. IVE\_DMA\_DIRECT\_COPY : 快速拷贝模式, 可实现直接从大块内存中扣取小块内存, 如图 1-1 所示, 计算公式如下:

$$I_{\text{out}}(x, y) = I(x, y) \quad (0 \leq x \leq width, 0 \leq y \leq height)$$

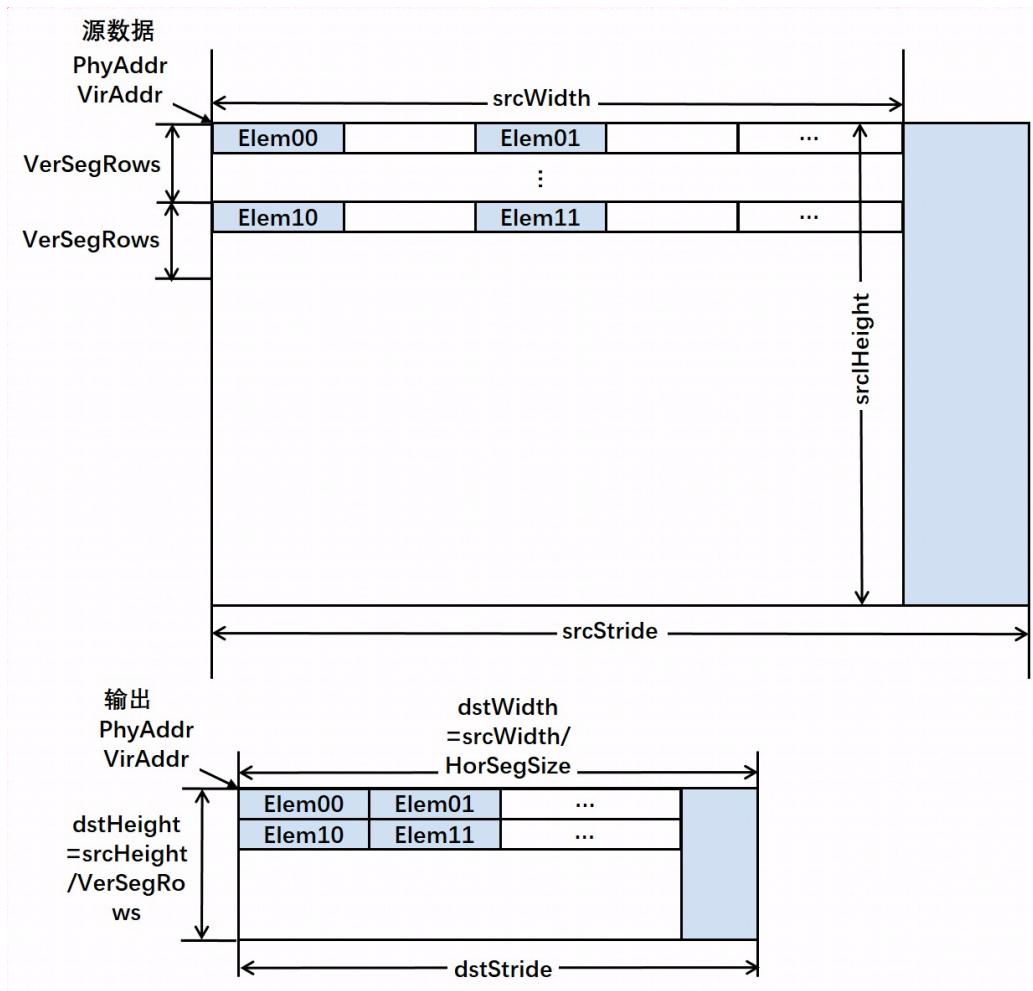
其中,  $I(x, y)$  对应 input,  $I_{\text{out}}(x, y)$  对应 output。

图 1-1 快速拷贝示意图



4. IVE\_DMA\_INTERVAL\_COPY : 间隔拷贝模式，要求输入数据宽度为 hor\_seg\_size 的倍数；间隔拷贝的方式，即将每 ver\_seg\_rows 行中第一行数据分割成 hor\_seg\_size 大小的段，拷贝每段的前 elem\_size 大小的字节，如图 1-2 所示。

图 1-2 间隔拷贝示意图



### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
GRAY;
    char *src_name = "path/to/src", *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
}
```

(续下页)

(接上页)

```

bm_image src, dst;

// create bm image struct
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
↪BYTE, &src, NULL);
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
↪BYTE, &dst, NULL);

// alloc bm image memory
ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

// read image data from input files
int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
↪+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
printf("file size is less than required bytes%d\n", byte_size);
};
fclose(fp_src);
void* in_ptr[4] = { (void *)input_data,
                   (void *)((unsigned char *)input_data + image_byte_size[0]),
                   (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
↪image_byte_size[1]),
                   (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
↪image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_dma(handle, src, dst, 0, NULL);

unsigned char *ive_res = (unsigned char *) malloc (width * height * [F]
↪sizeof(unsigned char));
memset(ive_res, 0, width * height * sizeof(unsigned char));
ret = bm_image_copy_device_to_host(dst, (void**)&ive_res);
if(ret != BM_SUCCESS){
    printf("dst bm_image_copy_device_to_host is failed \n");
    exit(-1);
}

FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_res);

bm_image_destroy(&src);
bm_image_destroy(&dst);

```

(续下页)

(接上页)

```
bm_dev_free(handle);

return ret;
}
```

## 5.81 bmcv\_ive\_filter

### 【描述】

该 API 使用 ive 硬件资源, 创建 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。

### 【语法】

```
1 bm_status_t bmcv_ive_filter(
2     bm_handle_t handle,
3     bm_image    input,
4     bm_image    output,
5     bmcv_ive_filter_ctrl filter_attr);
```

### 【参数】

表 5.54: bmcv\_ive\_filter 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体, 不能为空。
output	输出	输出 bm_image 对象结构体, 不能为空, 宽、高同 input。
filter_attr	输入	控制信息结构体, 不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
	NV21		
	NV61		
output	同 input	DATA_TYPE_EXT_1N_BYTE	同 input

### 【数据类型说明】

【说明】 定义模板滤波控制信息。

```

1 typedef struct bmcv_ive_filter_ctrl_s{
2     signed char as8_mask[25];
3     unsigned char u8_norm;
4 } bmcv_ive_filter_ctrl;
```

成员名称	描述
as8_mask	5x5 模板系数，外围系数设为 0，可实现 3x3 模板滤波。
u8_norm	归一化参数，取值范围: [0, 13]。

### 【注意】

- 通过配置不同的模板系数可以达到不同的滤波效果。

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- 当输入数据为 FORMAT\_NV21、FORMAT\_NV61 类型时，要求输出数据跨度一致。
- Filter 计算公式：

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

- 经典高斯模板如下，其中 norm\_u8 分别等于 4、8、8:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 5 & 6 & 5 & 2 \\ 3 & 6 & 8 & 6 & 3 \\ 2 & 5 & 6 & 5 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix} \times 3 \quad \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    unsigned char u8Norm = 4;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY; // 14 4 6
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;

    signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 2, 4,
                                2, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0 };

    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src, dst;
    int stride[4];

    bmcv_ive_filter_ctrl filterAttr;
    memcpy(filterAttr.as8_mask, arr3by3, 5 * 5 * sizeof(signed char));
    filterAttr.u8_norm = u8Norm;

    // calculate image stride && create bm image struct
    int data_size = 1;
    stride[0] = align_up(width, 16) * data_size;

    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
                    src, stride);
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
                    dst, stride);
```

(续下页)

(接上页)

```

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + F
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
                   (void *)((unsigned char *)input_data + image_byte_size[0]),
                   (void *)((unsigned char *)input_data + image_byte_size[0] + F
image_byte_size[1]),
                   (void *)((unsigned char *)input_data + image_byte_size[0] + F
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);
ret = bmcv_ive_filter(handle, src, dst, filterAttr);

bm_image_get_byte_size(dst, image_byte_size);
byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + F
+ image_byte_size[3];
unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
memset(output_ptr, 0, sizeof(byte_size));

void* out_ptr[4] = {(void *)output_ptr,
                   (void *)((char *)output_ptr + image_byte_size[0]),
                   (void *)((char *)output_ptr + image_byte_size[0] + image_byte_
size[1]),
                   (void *)((char *)output_ptr + image_byte_size[0] + image_byte_
size[1] + image_byte_size[2])};

ret = bm_image_copy_device_to_host(dst, (void **)out_ptr);

FILE *filter_fp = fopen(dst_name, "wb");
fwrite((void *)output_ptr, 1, width * height, filter_fp);
fclose(filter_fp);

free(input_data);
free(output_ptr);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}

```

## 5.82 bmcv\_ive\_csc

### 【描述】

该 API 使用 ive 硬件资源的 csc 功能, 创建色彩空间转换任务, 可实现 YUV2RGB、YUV2HSV、RGB2YUV 的色彩空间转换。

### 【语法】

```

1 bm_status_t bmcv_ive_csc(
2     bm_handle_t    handle,
3     bm_image      input,
4     bm_image      output,
5     csc_type_t    csc_type);

```

### 【参数】

表 5.55: bmcv\_ive\_csc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体, 不能为空。
output	输出	输出 bm_image 对象结构体, 不能为空, 宽、高同 input。
csc_type	输入	csc 工作模式。

参数名称	图像格式	数据类型	分辨率
input	FORMAT_NV21 FORMAT_NV61 FORMAT_RGB_PACKED FORMAT_RGB_PLANAR FORMAT_RGBP_SEPARATE	U8	64x64~1920x1080
output	FORMAT_NV21 FORMAT_NV61 FORMAT_RGB_PACKED FORMAT_RGB_PLANAR FORMAT_RGBP_SEPARATE	U8	同 input

**【补充】** FORMAT\_YUV444P 可以转 FORMAT\_HSV\_PLANAR, 转换模式色彩空间转换控制模式选择 CSC\_YCbCr2RGB\_BT601 或者 CSC\_YCbCr2RGB\_BT709。

## 【数据类型说明】

**【说明】** 定义色彩空间转换控制模式。

```

1  typedef enum csc_type {
2      CSC_YCbCr2RGB_BT601 = 0,
3      CSC_YPbPr2RGB_BT601,
4      CSC_RGB2YCbCr_BT601,
5      CSC_YCbCr2RGB_BT709,
6      CSC_RGB2YCbCr_BT709,
7      CSC_RGB2YPbPr_BT601,
8      CSC_YPbPr2RGB_BT709,
9      CSC_RGB2YPbPr_BT709,
10     CSC_FANCY_PbPr_BT601 = 100,
11     CSC_FANCY_PbPr_BT709,
12     CSC_USER_DEFINED_MATRIX = 1000,
13     CSC_MAX_ENUM
14 } csc_type_t;

```

成员名称	描述
CSC_YCbCr2RGB_BT601	BT601 的 YUV2RGB 图片变换
CSC_YPbPr2RGB_BT601	BT601 的 YUV2RGB 视频变换。
CSC_RGB2YCbCr_BT601	BT601 的 RGB2YUV 图像变换。
CSC_YCbCr2RGB_BT709	BT709 的 YUV2RGB 图像变换。
CSC_RGB2YCbCr_BT709	BT709 的 RGB2YUV 图像变换。
CSC_RGB2YPbPr_BT601	BT601 的 RGB2YUV 视频变换。
CSC_YPbPr2RGB_BT709	BT709 的 YUV2RGB 视频变换。
CSC_RGB2YPbPr_BT709	BT709 的 RGB2YUV 视频变换。

## 【注意】

- CSC\_YPbPr2RGB\_BT601 和 CSC\_YPbPr2RGB\_BT709 模式，输出满足  $16 \leq R, G, B \leq 235$ 。
- CSC\_YCbCr2RGB\_BT601 和 CSC\_YCbCr2RGB\_BT709 模式，输出满足  $0 \leq R, G, B \leq 255$ 。
- CSC\_RGB2YPbPr\_BT601 和 CSC\_RGB2YPbPr\_BT709 模式，输出满足  $0 \leq Y, U, V \leq 255$ 。
- CSC\_RGB2YCbCr\_BT601 和 CSC\_RGB2YCbCr\_BT709 模式。, 输出满足  $0 \leq Y \leq 235, 0 \leq U, V \leq 240$ 。

**【返回值】**

该函数成功调用时，返回 BM\_SUCCESS。

**【注意】**

1. 输入输出图像的 width 都需要 16 对齐。
2. 当输出数据为 FORMAT\_RGB\_PLANAR、FORMAT\_NV21、FORMAT\_NV61 类型时，要求输出数据跨度一致。
3. 不同的模式其输出的取值范围不一样。

**示例代码**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    csc_type_t csc_type = CSC_YCbCr2RGB_BT601;
    bm_image_format_ext src_fmt = 2; // FORMAT_NV21: 4; yuv444p : 2[F]
    ↪FORMAT_RGB_PACKED: 10
    bm_image_format_ext dst_fmt = 10;
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src, dst;
    int src_stride[4], dst_stride[4];

    // calculate image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    src_stride[1] = align_up(width, 16) * data_size;
    src_stride[2] = align_up(width, 16) * data_size;
    dst_stride[0] = align_up(width*3, 16) * data_size;

    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↪BYTE, &src, src_stride);
```

(续下页)

(接上页)

```
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
←BYTE, &dst, dst_stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);

int byte_size = width * height * 3;
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[3] = { (void *)input_data,
                    (void *)input_data + width * height,
                    (void *)input_data + width * height * 2 };
bm_image_copy_host_to_device(src, (void **)in_ptr);

ret = bmcv_ive_csc(handle, src, dst, csc_type);

bm_image_get_byte_size(dst, image_byte_size);

byte_size = width * height * 3;
unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
memset(output_ptr, 0, sizeof(byte_size));
void* out_ptr[1] = { (void *)output_ptr };
ret = bm_image_copy_device_to_host(dst, (void **)out_ptr);

FILE *fp_dst = fopen(dst_name, "wb");
if (fwrite((void *)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
    printf("file size is less than %d required bytes\n", byte_size);
}
fclose(fp_dst);

bm_image_destroy(&src);
bm_image_destroy(&dst);
free(input_data);
free(output_ptr);

bm_dev_free(handle);
return 0;
}
```

### 5.83 bmcv\_ive\_filter\_and\_csc

#### 【描述】

该 API 使用 ive 硬件资源，创建 5x5 模板滤波和 YUV2RGB 色彩空间转换复合任务，通过一次创建完成两种功能。

#### 【语法】

```

1 bm_status_t bmcv_ive_filter_and_csc(
2     bm_handle_t        handle,
3     bm_image           input,
4     bm_image           output,
5     bmcv_ive_filter_ctrl filter_attr,
6     csc_type_t         csc_type);

```

#### 【参数】

表 5.56: bmcv\_ive\_filter\_and\_csc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空，宽、高同 input。
filter_attr	输入	滤波控制参数结构体，不能为空。
csc_type	输入	色彩转换模式选择，不能为空。

参数名称	图像格式	数据类型	分辨率
input	NV21 NV61	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	RGB_PLANAR RGB_PACKAGE	DATA_TYPE_EXT_1N_BYTE	同 input

#### 【数据类型说明】

【说明】 定义模板滤波加色彩空间转换复合功能控制信息。

```

1 typedef struct bmcv_ive_filter_ctrl_s{
2     signed char as8_mask[25];
3     unsigned char u8_norm;
4 } bmcv_ive_filter_ctrl;

```

成员名称	描述
as8_mask	5x5 模板系数。
u8_norm	归一化参数，取值范围：[0, 13]。

【说明】 定义色彩空间转换控制模式。

```

1 typedef enum csc_type {
2     CSC_YCbCr2RGB_BT601 = 0,
3     CSC_YPbPr2RGB_BT601,
4     CSC_RGB2YCbCr_BT601,
5     CSC_YCbCr2RGB_BT709,
6     CSC_RGB2YCbCr_BT709,
7     CSC_RGB2YPbPr_BT601,
8     CSC_YPbPr2RGB_BT709,
9     CSC_RGB2YPbPr_BT709,
10    CSC_YCbCr2HSV_BT601,
11    CSC_YCbCr2HSV_BT709,
12    CSC_YCbCr2LAB_BT601,
13    CSC_YCbCr2LAB_BT709,
14    CSC_RGB2HSV,
15    CSC_RGB2GRAY,
16    CSC_FANCY_PbPr_BT601 = 100,
17    CSC_FANCY_PbPr_BT709,
18    CSC_USER_DEFINED_MATRIX = 1000,
19    CSC_MAX_ENUM
20 } csc_type_t;

```

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

1. 仅支持 YUV2RGB 的 4 种工作模式，具体参见 bmcv\_ive\_csc。
2. 输入输出图像的 width 都需要 16 对齐。

### 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

(续下页)

(接上页)

```

#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    unsigned char u8Norm = 4;
    int height = 1080, width = 1920;
    csc_type_t csc_type = CSC_YCbCr2RGB_BT601;
    bm_image_format_ext src_fmt = 2, dst_fmt = 10; // FORMAT_NV21: 4; [F]
    ↪yuv444p : 2 FORMAT_RGB_PACKED: 10
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src, dst;
    int src_stride[4], dst_stride[4];

    // filter data
    signed char arr3by3[25] = {0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 2,
                               4, 2, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0};

    // config setting
    bmcv_ive_filter_ctrl filterAttr;

    filterAttr.u8_norm = u8Norm;
    memcpy(filterAttr.as8_mask, arr3by3, 5 * 5 * sizeof(signed char));

    // calculate image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    src_stride[1] = align_up(width, 16) * data_size;
    src_stride[2] = align_up(width, 16) * data_size;
    dst_stride[0] = align_up(width * 3, 16) * data_size;

    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↪BYTE, &src, src_stride);
    bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
    ↪BYTE, &dst, dst_stride);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    int byte_size = width * height * 3;
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");

```

(续下页)

(接上页)

```
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[3] = {(void *)input_data,
                    ((unsigned char *)input_data + width * height),
                    ((unsigned char *)input_data + 2 * width * height)};
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_filter_and_csc(handle, src, dst, filterAttr, csc_type);

unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
memset(output_ptr, 0, sizeof(byte_size));

void* out_ptr[3] = {output_ptr,
                    ((char *)output_ptr + width * height),
                    ((char *)output_ptr + 2 * width * height)};

ret = bm_image_copy_device_to_host(dst, (void **)out_ptr);
FILE *fp_dst = fopen(dst_name, "wb");
if (fwrite((void *)output_ptr, 1, byte_size, fp_dst) < (unsigned int)byte_size){
    printf("file size is less than %d required bytes\n", byte_size);
}
fclose(fp_dst);

free(input_data);
free(output_ptr);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);

return ret;
}
```

#### 5.84 bmcv\_ive\_dilate

##### 【描述】

该 API 使用 ive 硬件资源，创建二值图像 5x5 模板膨胀任务，将模板覆盖的区域中的最大像素值赋给参考点。

##### 【语法】

```
1 bm_status_t bmcv_ive_dilate(
2     bm_handle_t handle,
3     bm_image    input,
4     bm_image    output,
5     unsigned char dilate_mask[25]);
```

## 【参数】

表 5.57: bmcv\_ive\_dilate 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空，宽、高同 input。
dilate_mask	输入	5x5 膨胀模板系数数组，不能为空，取值范围：0 或 255。

## 【数据类型说明】

参数名称	图像格式	数据类型	分辨率
input	GRAY 的二值图	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY 的二值图	DATA_TYPE_EXT_1N_BYTE	同 input

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 模板系数只能为 0 或 255。
3. 接口可以处理灰度图的输入，对于处理的结果不可保证，所以不提倡，但是也可以尝试。
4. 模板样例

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 255 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 255 & 255 & 255 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src", *dst_name = "path/to/dst";

    unsigned char arr3by3[25] = { 0, 0, 0, 0, 0, 0, 0, 255, 0, 0, 0, 255, 255,
                                255, 0, 0, 0, 255, 0, 0, 0, 0, 0, 0 };

    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
}
```

(续下页)

(接上页)

```

bm_image src, dst;
int stride[4];

// calculate image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src, stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
dst, stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
    (void *)((unsigned char *)input_data + image_byte_size[0]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_dilate(handle, src, dst, arr3by3);

unsigned char* ive_dilate_res = malloc(width * height * sizeof(unsigned char));
memset(ive_dilate_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_dilate_res);

FILE *ive_result_fp = fopen(dst_name, "wb");
fwrite((void *)ive_dilate_res, 1, width * height, ive_result_fp);
fclose(ive_result_fp);

free(input_data);
free(ive_dilate_res);

bm_image_destroy(&src);

```

(续下页)

(接上页)

```
bm_image_destroy(&dst);

bm_dev_free(handle);
return ret;
}
```

## 5.85 bmcv\_ive\_erode

### 【描述】

该 API 使用 ive 硬件资源，创建二值图像 5x5 模板腐蚀任务，将模板覆盖的区域中的最小像素值赋给参考点。

### 【语法】

```
1 bm_status_t bmcv_ive_erode(
2     bm_handle_t handle,
3     bm_image    input,
4     bm_image    output,
5     unsigned char erode_mask[25]);
```

### 【参数】

表 5.58: bmcv\_ive\_erode 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空，宽、高同 input。
erode_mask	输入	腐蚀模板系数，不能为空，取值范围: 0 或 255。

### 【数据类型说明】

参数名称	图像格式	数据类型	分辨率
input	GRAY 的二值图	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY 的二值图	DATA_TYPE_EXT_1N_BYTE	同 input

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 模板系数只能为 0 或 255。
3. 模板样例

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 255 & 0 & 0 \\ 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 255 & 255 & 255 & 0 \\ 0 & 0 & 255 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 255 & 255 & 255 & 0 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 0 & 255 & 255 & 255 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>
```

(续下页)

(接上页)

```

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src", *dst_name = "path/to/dst";

    unsigned char arr3by3[25] = { 0, 0, 0, 0, 0, 0, 0, 255, 0, 0, 0, 255, 255,
                                255, 0, 0, 0, 255, 0, 0, 0, 0, 0, 0, 0 };

    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src, dst;
    int stride[4];
    // calculate image stride && create bm image struct
    int data_size = 1;
    stride[0] = align_up(width, 16) * data_size;

    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
                    src, stride);
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
                    dst, stride);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    // bm_load_bin(src, src_name);
    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
    + image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void *)((unsigned char*)input_data + image_byte_size[0]),
                       (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
                     image_byte_size[1]),
                       (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
                     image_byte_size[1] + image_byte_size[2])};
    bm_image_copy_host_to_device(src, in_ptr);
    ret = bmcv_ive_erode(handle, src, dst, arr3by3);

```

(续下页)

(接上页)

```
unsigned char* iveErodeRes = malloc(width * height * sizeof(unsigned char));
memset(iveErodeRes, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&iveErodeRes);

FILE *ive_result_fp = fopen(dst_name, "wb");
fwrite((void *)iveErodeRes, 1, width * height, ive_result_fp);
fclose(ive_result_fp);

free(input_data);
free(iveErodeRes);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}
```

## 5.86 bmcv\_ive\_ccl

### 【描述】

该 API 使用 ive 硬件资源的 ccl 功能, 创建二值图像的连通区域标记任务, 即图像中具有相同像素值且位置相邻的前景像素点组成的图像区域。

### 【语法】

```
1 bm_status_t bmcv_ive_ccl(
2     bm_handle_t    handle,
3     bm_image      src_dst_image,
4     bm_device_mem_t ccblob_output,
5     bmcv_ive_ccl_attr ccl_attr);
```

### 【参数】

表 5.59: bmcv\_ive\_ccl 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
src_dst_image	输入、输出	输入 bm_image 对象结构体，连通区域标记在输入图像上进行，即输入图像同时也是标记图像输出，不能为空。
ccblob_output	输出	连通区域信息数据结构体，不能为空，内存至少需要配置 sizoef(bm_ive_ccblob) 大小，最多输出 254 个有效连通区域。
ccl_attr	输入	连通区域标记控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
srcDstImage	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080

### 【数据类型说明】

【说明】 定义连通区域信息。

```

1 typedef struct bmcv_ive_region_s{
2     int u32_area;
3     unsigned short u16_left;
4     unsigned short u16_right;
5     unsigned short u16_top;
6     unsigned short u16_bottom;
7 } bmcv_ive_region;

```

成员名称	描述
u32_area	连通区域面积，以及连通区域像素数目。
u16_left	连通区域外接矩形的最左边坐标。
u16_right	连通区域外接矩形的最右边坐标。
u16_top	连通区域外接矩形的最上边坐标。
u16_bottom	连通区域外接矩形的最下边坐标。

【说明】 定义连通区域标记的输出信息。

```

1 typedef struct bmcv_ive_ccblob_s{
2     unsigned short u16_cur_aera_thr;
3     signed char s8_label_status;
4     unsigned char u8_region_num;
5     bmcv_ive_region ast_region[BM_IVE_MAX_REGION_NUM];
6 } bmcv_ive_ccblob;

```

成员名称	描述
u16_cur_aera_thr	有效连通区域的面积阈值，astRegion 中面积小于这个阈值的都被置为 0。
s8_label_status	连通区域标记是否成功。1：标记失败；0：标记成功。
u8_region_num	有效连通区域个数。
ast_region	连通区域信息：有效的连通区域其面积大于 0，对应标记为数组下标加 1。

【说明】定义连通区域模式。

```

1 typedef enum bmcv_ive_ccl_mode_e{
2     BM_IVE_CCL_MODE_4C = 0x0,
3     BM_IVE_CCL_MODE_8C = 0x1,
4 } bmcv_ive_ccl_mode;

```

成员名称	描述
BM_IVE_CCL_MODE_4C	4 连通。
BM_IVE_CCL_MODE_8C	8 连通。

【说明】定义连通区域标记控制参数。

```

1 typedef struct bmcv_ive_ccl_attr_s{
2     bmcv_ive_ccl_mode en_mode;
3     unsigned short u16_init_area_thr;
4     unsigned short u16_step;
5 } bmcv_ive_ccl_attr;

```

成员名称	描述
en_mode	连通区域模式。
u16_init_area_thr	初始面积阈值。 取值范围：[0, 65535]，参考取值：4。
u16_step	面积阈值增长步长。 取值范围：[1, 65535]，参考取值：2。

【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

【注意】

- 输入输出图像的 width 都需要 16 对齐。
- 连通区域的信息保存在 ccblob\_output.ast\_region 中。

3. ccblob\_output.u8\_region\_num 表示有效的连通区域数目，最多 254 个有效连通区域；有效的连通区域的面积大于 ccblob\_output.u16\_cur\_aera\_thr，标记号为其所在 ccblob\_output.ast\_region 数组元素的下标 + 1。有效的连通区域并不一定连续地存储在数组中，而很可能是间断的分布在数组中。
4. 若 ccblob\_output.s8\_label\_status 为 0，则标记成功（一个区域一个标记）；若为 -1，则标记失败（一个区域多个标记或者多个区域共用一个标记）。对于后者，若用户需要正确的标记号，还需要再次根据 ccblob\_output 中的外接矩形信息重新标记。不管标记是否成功，连通区域的外接矩形信息一定是正确可用的。
5. 输出的连通区域会用 ccl\_attr.u16\_init\_area\_thr 进行筛选，面积小于等于 pstCclCtrl→u16\_init\_area\_thr 均会被置为 0。
6. 当连通区域数目大于 254，会用 ccl\_attr.u16\_init\_area\_thr 删除面积小的连通区域；若 ccl\_attr.u16\_init\_area\_thr 不满足删除条件，会以 pstCclCtrl→u16\_step 为步长，增大删除连通区域的面积阈值。
7. 最终的面积阈值存储在 ccblob\_output.u16\_cur\_aera\_thr 中。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    bmcv_ive_ccl_mode_enMode = BM_IVE_CCL_MODE_8C;
    unsigned short u16InitAreaThr = 4;
    unsigned short u16Step = 2;

    char *src_name = "path/to/src";
    char *ive_res_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src_dst_img;
    bm_device_mem_t pst_blob;
    bmcv_ive_ccl_attr ccl_attr;
    bmcv_ive_ccblob *ccblob = NULL;
    int stride[4];
}
```

(续下页)

(接上页)

```
ccl_attr.en_mode = enMode;
ccl_attr.u16_init_area_thr = u16InitAreaThr;
ccl_attr.u16_step = u16Step;

ccblob = (bmcv_ive_ccblob *)malloc(sizeof(bmcv_ive_ccblob));
memset(ccblob, 0, sizeof(bmcv_ive_ccblob));

// calc iver image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src_dst_img, stride);

ret = bm_image_alloc_dev_mem(src_dst_img, BMCV_HEAP1_ID);
ret = bm_malloc_device_byte(handle, &pst_blob, sizeof(bmcv_ive_ccblob));

// bm_ive_read_bin(src_dst_img, src_name);
int image_byte_size[4] = {0};
bm_image_get_byte_size(src_dst_img, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {((void *)input_data,
                    ((void *)((unsigned char*)input_data + image_byte_size[0])),
                    ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_size[1])),
                    ((void *)((unsigned char*)input_data + image_byte_size[0] + image_byte_size[1] + image_byte_size[2]))};
bm_image_copy_host_to_device(src_dst_img, in_ptr);

ret = bmcv_ive_ccl(handle, src_dst_img, pst_blob, ccl_attr);

ret = bm_memcpy_d2s(handle, (void*)ccblob, pst_blob);
FILE *fp = fopen(ive_res_name, "wb");
fwrite((void *)ccblob, 1, sizeof(bmcv_ive_ccblob), fp);
fclose(fp);

bm_dev_free(handle);
return ret;
}
```

## 5.87 bmcv\_ive\_integ

### 【描述】

该 API 使用 ive 硬件资源，创建灰度图像的积分图计算任务，积分像素值等于灰度图的左上角与当前点所围成的矩形区域内所有像素点灰度值之和/平方和。

### 【语法】

```

1 bm_status_t bmcv_ive_integ(
2     bm_handle_t    handle,
3     bm_image       input,
4     bm_device_mem_t output,
5     bmcv_ive_integ_ctrl_s integ_attr);

```

### 【参数】

表 5.60: bmcv\_ive\_integ 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输入 bm_device_mem_t 对象数据结构体，宽、高同 input。
integ_attr	输入	积分图计算控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	32x16~1920x1080

### 【数据类型说明】

【说明】 定义积分图输出控制参数。

```

1 typedef enum _ive_integ_out_ctrl_e {
2     IVE_INTEG_MODE_COMBINE = 0x0,
3     IVE_INTEG_MODE_SUM = 0x1,

```

(续下页)

(接上页)

```

4   IVE_INTEG_MODE_SQSUM = 0x2,
5   IVE_INTEG_MODE_BUTT
6 } ive_integ_out_ctrl_e;

```

成员名称	描述
IVE_INTEG_MODE_COMBINE	和、平方和积分图组合输出。
IVE_INTEG_MODE_SUM	仅和积分图输出。
IVE_INTEG_MODE_SQSUM	仅平方和积分图输出。

### 【说明】定义积分图计算控制参数

```

1 typedef struct bmcv_integ_ctrl_t{
2     ive_integ_out_ctrl_e en_out_ctrl;
3 } bmcv_ive_integ_ctrl_s;

```

成员名称	描述
en_out_ctrl	积分图输出控制参数。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入图像的 width 都需要 16 对齐。
2. IVE\_INTEG\_MODE\_COMBINE, 组合输出模式, 输出图像数据类型必须为 u64, 计算公式如下:

$$\begin{aligned}
 I_{\text{sum}}(x, y) &= \sum_{i \geq 0}^{\lfloor x \rfloor} \sum_{j \geq 0}^{\lfloor y \rfloor} I(i, j) \\
 I_{\text{sq}}(x, y) &= \sum_{i \geq 0}^{\lfloor x \rfloor} \sum_{j \geq 0}^{\lfloor y \rfloor} (I(i, j) \cdot I(i, j)) \\
 I_{\text{out}}(x, y) &= (I_{\text{sq}}(x, y) \ll 28 \mid I_{\text{sum}}(x, y) \& 0xFFFFFFFF)
 \end{aligned}$$

3. IVE\_INTEG\_MODE\_SUM, 仅和积分图输出模式, 输出图像数据类型必须为 u32。

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

$$I_{\text{out}}(x, y) = I_{\text{sum}}(x, y)$$

4. IVE\_INTEG\_MODE\_SQSUM, 仅平方和积分图输出模式, 输出图像数据类型必须为 u32。

$$I_{\text{sq}}(x, y) = \sum_{i \geq 0}^{\lfloor x \rfloor} \sum_{j \geq 0}^{\lfloor y \rfloor} (I(i, j) \cdot I(i, j))$$

$$I_{\text{out}}(x, y) = I_{\text{sq}}(x, y)$$

其中,  $I(x, y)$  对应 input,  $I_{\text{sum}}$  对应 output。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext src_fmt = FORMAT_GRAY;
    ive_integ_out_ctrl_e integ_mode = 0;
    char* src_name = "path/to/src", *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bmcv_ive_integ_ctrl_s integ_attr;
    bm_image src;
    bm_device_mem_t dst;
    int src_stride[4];

    // calc ive image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;

    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src, src_stride);
```

(续下页)

(接上页)

```
ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
if (ret != BM_SUCCESS) {
    printf("bm_image_alloc_dev_mem_src. ret = %d\n", ret);
    exit(-1);
}

int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = { (void *)input_data,
                    (void *)((unsigned char *)input_data + image_byte_size[0]),
                    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
                    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);
data_size = sizeof(unsigned int);

ret = bm_malloc_device_byte(handle, &dst, height * width * data_size);
integ_attr.en_out_ctrl = integ_mode;

ret = bmcv_ive_integ(handle, src, dst, integ_attr);

unsigned int *dst_intg_u32 = malloc(width * height * data_size);
ret = bm_memcpy_d2s(handle, dst_intg_u32, dst);
FILE *intg_result_fp = fopen(dst_name, "wb");
fwrite((void *)dst_intg_u32, data_size, width * height, intg_result_fp);
fclose(intg_result_fp);

free(dst_intg_u32);

bm_image_destroy(&src);
bm_free_device(handle, dst);
bm_dev_free(handle);
return 0;
}
```

## 5.88 bmcv\_ive\_hist

### 【描述】

该 API 使用 ive 硬件资源，创建灰度图像，遍历图像像素值实现统计图像相同像素值出现的次数，构建直方图。

### 【语法】

```
1 bm_status_t bmcv_ive_hist(
2     bm_handle_t    handle,
3     bm_image      input,
4     bm_device_mem_t output);
```

### 【参数】

表 5.61: bmcv\_ive\_hist 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输入 bm_device_mem_t 对象数据结构体，不能为空，内存至少配置 1024 字节，输出使用 u32 类型代表 0-255 每个数字的统计值。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

1. 输入图像的 width 都需要 16 对齐。

2. 计算公式如下：

$$I(x) = \sum_i \sum_j \begin{cases} 1 & \text{if } I(i,j) = x \\ 0 & \text{otherwise} \end{cases} \quad \text{for } x = 0 \dots 255$$

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext src_fmt = FORMAT_GRAY;
    char * src_name = "path/to/src", *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src;
    bm_device_mem_t dst;
    int src_stride[4];

    // calculate image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src, src_stride);
    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
    ↵+ image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
        (void *)((unsigned char*)input_data + image_byte_size[0])},
        (void *)((unsigned char*)input_data + image_byte_size[1])},
        (void *)((unsigned char*)input_data + image_byte_size[2])},
        (void *)((unsigned char*)input_data + image_byte_size[3])},
```

(续下页)

(接上页)

```

        (void *)((unsigned char*)input_data + image_byte_size[0] + F
        ↵image_byte_size[1]),
        (void *)((unsigned char*)input_data + image_byte_size[0] + F
        ↵image_byte_size[1] + image_byte_size[2]));
    bm_image_copy_host_to_device(src, in_ptr);
    // create result
    ret = bm_malloc_device_byte(handle, &dst, 1024);
    ret = bmcv_ive_hist(handle, src, dst);

    unsigned char *dst_hist_result = malloc(dst.size);

    ret = bm_memcpy_d2s(handle, dst_hist_result, dst);

    FILE *hist_result_fp = fopen(dst_name, "wb");
    fwrite((void *)dst_hist_result, 1, dst.size, hist_result_fp);
    fclose(hist_result_fp);

    free(input_data);
    free(dst_hist_result);

    bm_image_destroy(&src);
    bm_free_device(handle, dst);

    bm_dev_free(handle);

    return 0;
}

```

### 5.89 bmcv\_ive\_gradfg

#### 【描述】

该 API 使用 ive 硬件资源，根据背景图像和当前帧图像的梯度信息计算梯度前景图像。

#### 【语法】

```

1  bm_status_t bmcv_ive_gradfg(
2      bm_handle_t      handle,
3      bm_image         input_bgdiff_fg,
4      bm_image         input_curgrad,
5      bm_image         intput_bggrad,
6      bm_image         output_gradfg,
7      bmcv_ive_gradfg_attr gradfg_attr);

```

## 【参数】

表 5.62: bmcv\_ive\_gradfg 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input_bgdiff_fg	输入	背景差分前景图像 bm_image 结构体，不能为空。
input_curgrad	输入	当前帧梯度图像 bm_image 结构体，不能为空。
intput_bggrad	输入	背景梯度图像 bm_image 结构体，不能为空，宽、高同 input_curgrad。
output_gradfg	输出	梯度前景图像 bm_image 结构体，不能为空，宽、高同 input_curgrad。
gradfg_attr	输入	计算梯度前景控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input_bgdiff_fg	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
input_curgrad	GRAY	DATA_TYPE_EXT_U16	同 input
intput_bggrad	GRAY	DATA_TYPE_EXT_U16	同 input
output_gradfg	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

## 【数据类型说明】

【说明】 定义梯度前景计算模式。

```

1 typedef enum bmcv_ive_gradfg_mode_e{
2     GRAD_FG_MODE_USE_CUR_GRAD = 0x0,
3     GRAD_FG_MODE_FIND_MIN_GRAD = 0x1,
4 } bmcv_ive_gradfg_mode;

```

成员名称	描述
GRAD_FG_MODE_USE_CUR_GRAD	当前位置梯度计算模式。
GRAD_FG_MODE_FIND_MIN_GRAD	周边最小梯度计算模式。

【说明】 定义计算梯度前景控制参数。

```

1 typedef struct bmcv_ive_gradfg_attr_s{
2     bmcv_ive_gradfg_mode en_mode;
3     unsigned short u16_edw_factor;
4     unsigned char u8_crl_coef_thr;

```

(续下页)

(接上页)

```

5   unsigned char u8_mag_crl_thr;
6   unsigned char u8_min_mag_diff;
7   unsigned char u8_noise_val;
8   unsigned char u8_edw_dark;
9 } bmcv_ive_gradfg_attr;
```

成员名称	描述
en_mode	梯度前景计算模式。参考 bm_ive_gradfg_mode。
u16_edw_factor	边缘宽度调节因子。 取值范围: [500, 2000], 参考取值: 100。
u8_crl_coef_thr	梯度向量相关系数阈值。 取值范围: [50, 100], 参考取值: 80。
u8_mag_crl_thr	梯度幅度阈值。 取值范围: [0, 20], 参考取值: 4。
u8_min_mag_diff	梯度幅值差值阈值。 取值范围: [2, 8], 参考取值: 2。
u8_noise_val	梯度幅值差值阈值。 取值范围: [1, 8], 参考取值: 1。
u8_edw_dark	黑像素使能标志。 0 表示不开启, 1 表示开启, 参考取值: 1。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 背景梯度图像和当前梯度图像的类型为 DATA\_TYPE\_EXT\_U16, 水平和竖直方向梯度按照 [xyxyxyxy...] 格式存储。

## 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
```

(续下页)

(接上页)

```

int height = 1080, width = 1920;
bm_image_format_ext fmt = FORMAT_GRAY;
bmcv_ive_gradfg_mode gradfg_mode = GRAD_FG_MODE_USE_CUR_
↪GRAD;
char *src_name = "path/to/src";
char *dst_name = "path/to/dst";
bm_handle_t handle = NULL;
/* 3 by 3 */
signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, -2, 0,
                           2, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0 };
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}

bm_image stBgdifffFg;
bm_image stCurGrad, stBgGrad, stGragFg;
int u8Stride[4], u16Stride[4];

// config setting

// normGrad config
bmvc_ive_normgrad_ctrl normGradAttr;
normGradAttr.en_mode = BM_IVE_NORM_GRAD_OUT_COMBINE;
normGradAttr.u8_norm = 8;
memcpy(&normGradAttr.as8_mask, arr3by3, 5 * 5 * sizeof(signed char));

bmvc_ive_gradfg_attr gradFgAttr;
gradFgAttr.en_mode = gradfg_mode;
gradFgAttr.u16_edw_factor = 1000;
gradFgAttr.u8_crl_coef_thr = 80;
gradFgAttr.u8_mag_crl_thr = 4;
gradFgAttr.u8_min_mag_diff = 2;
gradFgAttr.u8_noise_val = 1;
gradFgAttr.u8_edw_dark = 1;

// calc ive image stride && create bm image struct
int data_size = 1;
u8Stride[0] = align_up(width, 16) * data_size;
data_size = 2;
u16Stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
↪stBgdifffFg, u8Stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_U16, &
↪stCurGrad, u16Stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_U16, &
↪stBgGrad, u16Stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
↪stGragFg, u8Stride);

```

(续下页)

(接上页)

```

↳stGragFg, u8Stride);

ret = bm_image_alloc_dev_mem(stBgdifffG, BMCV_HEAP1_ID);

int image_byte_size[4] = {0};
bm_image_get_byte_size(stBgdifffG, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
↳+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
                    ((void *)((unsigned char*)input_data + image_byte_size[0])),
                    ((void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1])),
                    ((void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1] + image_byte_size[2]))};

bm_image_copy_host_to_device(stBgdifffG, in_ptr);

ret = bm_image_alloc_dev_mem(stCurGrad, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stBgGrad, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stGragFg, BMCV_HEAP1_ID);

// Create Fake Data.
ret = bmcv_ive_norm_grad(handle, &stBgdifffG, NULL, NULL, &stCurGrad,[F]
↳normGradAttr);

normGradAttr.u8_norm = 2;
ret = bmcv_ive_norm_grad(handle, &stBgdifffG, NULL, NULL, &stBgGrad,[F]
↳normGradAttr);
ret = bmcv_ive_gradfg(handle, stBgdifffG, stCurGrad, stBgGrad, stGragFg,[F]
↳gradFgAttr);

unsigned char *gradFg_res = (unsigned char *)malloc(width * height * [F]
↳sizeof(unsigned char));
memset(gradFg_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(stGragFg, (void**)&gradFg_res);

FILE *ive_fp = fopen(dst_name, "wb");
fwrite((void *)gradFg_res, 1, width * height, ive_fp);
fclose(ive_fp);
free(gradFg_res);

bm_image_destroy(&stBgdifffG);
bm_image_destroy(&stCurGrad);
bm_image_destroy(&stBgGrad);

```

(续下页)

(接上页)

```

bm_image_destroy(&stGragFg);

bm_dev_free(handle);
return 0;
}

```

## 5.90 bmcv\_ive\_lbp

### 【描述】

该 API 使用 ive 硬件资源, lbp 是描述图像局部特征的算子, 邻域窗口内的像素值与中心像素值进行相减结果和阈值对比求和得到这一局部的中心像素值。

### 【语法】

```

1 bm_status_t bmcv_ive_lbp(
2     bm_handle_t      handle,
3     bm_image        input,
4     bm_image        output,
5     bmcv_ive_lbp_ctrl_attr  lbp_attr);

```

### 【参数】

表 5.63: bmcv\_ive\_lbp 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体, 不能为空。
output	输出	输出 bm_image 对象结构体, 不能为空, 宽、高同 input。
lbp_attr	输入	LBP 纹理计算控制参数结构体, 不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080

## 【数据类型说明】

**【说明】** 定义 LBP 计算的比较模式。

```

1 typedef enum bmcv_lbp_cmp_mode_e{
2     BM_IVE_LBP_CMP_MODE_NORMAL = 0x0,
3     BM_IVE_LBP_CMP_MODE_ABS    = 0x1,
4 } bmcv_lbp_cmp_mode;

```

成员名称	描述
BM_IVE_LBP_CMP_MODE_NORMAL	LBP 简单比较模式。
BM_IVE_LBP_CMP_MODE_ABS	LBP 绝对值比较模式。

**【说明】** 定义 8 bit 数据联合体。

```

1 typedef union bmcv_ive_8bit_u{
2     signed char s8_val;
3     unsigned char u8_val;
4 } bmcv_ive_8bit;

```

成员名称	描述
s8_val	有符号的 8bit 值。
u8_val	无符号的 8bit 值。

**【说明】** 定义 LBP 纹理计算控制参数。

```

1 typedef struct bmcv_lbp_ctrl_attr_s{
2     bmcv_lbp_cmp_mode en_mode;
3     bmcv_ive_8bit    un8_bit_thr;
4 } bmcv_ive_lbp_ctrl_attr;

```

成员名称	描述
en_mode	LBP 比较模式。
un8_bit_thr	LBP 比较阈值。 BM_IVE_LBP_CMP_MODE_NORMAL: 取值范围: [-128, 127]; BM_IVE_LBP_CMP_MODE_ABS: 取值范围: [0, 255]。

## 【返回值】

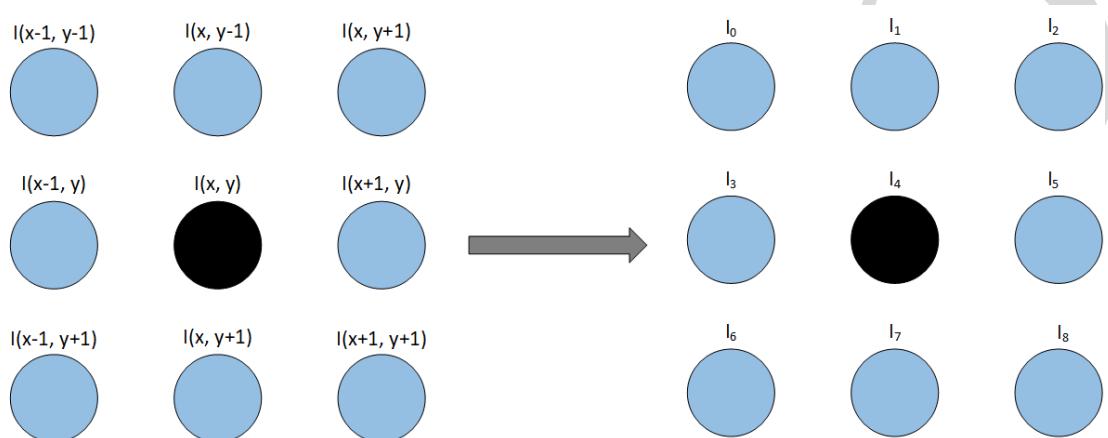
该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。

2. LBP 计算公式如图 1-3 所示:

图 1-3 LBP 计算公式示意图



- BM\_IVE\_LBP\_CMP\_MODE\_NORMAL:

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

- BM\_IVE\_LBP\_CMP\_MODE\_ABS:

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

其中,  $I(x, y)$  对应 input,  $lbp(x, y)$  对应 output,  $thr$  对应  $lbp\_attr.un8\_bit\_thr$ 。

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    bmcv_lbp_cmp_mode lbpCmpMode = BM_IVE_LBP_CMP_MODE_NORMAL;
```

(续下页)

(接上页)

```

char *src_name = "path/to/src", *dst_name = "path/to/dst";
bm_handle_t handle = NULL;
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}

bm_image src, dst;
int stride[4];

// calculate image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src, stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
dst, stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
    (void *)((unsigned char *)input_data + image_byte_size[0]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);
bmcv_ive_lbp_ctrl_attr lbp_ctrl;
lbp_ctrl.en_mode = lbpCmpMode;
lbp_ctrl.un8_bit_thr.s8_val = (lbpCmpMode == BM_IVE_LBP_CMP_
MODE_ABS ? 35 : 41);

ret = bmcv_ive_lbp(handle, src, dst, lbp_ctrl);

unsigned char* ive_lbp_res = malloc(width * height * sizeof(unsigned char));
memset(ive_lbp_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_lbp_res);

```

(续下页)

(接上页)

```
FILE *ive_result_fp = fopen(dst_name, "wb");
fwrite((void *)ive_lbp_res, 1, width * height, ive_result_fp);
fclose(ive_result_fp);

free(input_data);
free(ive_lbp_res);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}
```

## 5.91 bmcv\_ive\_normgrad

### 【描述】

该 API 使用 ive 硬件资源，创建归一化梯度计算任务，梯度分量均归一化到 S8。

### 【语法】

```
1 bm_status_t bmcv_ive_norm_grad(
2     bm_handle_t handle,
3     bm_image * input,
4     bm_image * output_h,
5     bm_image * output_v,
6     bm_image * output_hv,
7     bmcv_ive_normgrad_ctrl normgrad_attr);
```

### 【参数】

表 5.64: bmcv\_ive\_norm\_grad 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*input	输入	输入 bm_image 对象数据指针，不能为空。
*output_h	输出	输出 bm_image 对象数据指针，由模板直接滤波并归一到 s8 后得到的梯度分量图像 (H) 指针，根据 normgrad_attr.en_mode，若需要输出，则不能为空。
*output_v	输出	输出 bm_image 对象数据指针，由模板直接滤波并归一到 s8 后得到的梯度分量图像 (V) 指针，根据 normgrad_attr.en_mode，若需要输出则不能为空。
*output_hv	输出	输出 bm_image 对象数据指针，由模板和转置后的模板直接滤波，并归一到 s8 后采用 package 格式存储的图像指针。根据 normgrad_attr.en_mode，若需要输出则不为空。
normgrad_attr	输入	归一化梯度信息计算参数，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output_h	GRAY	DATA_TYPE_EXT_1N_BYTE_SIGNED	64x64~1920x1080
output_v	GRAY	DATA_TYPE_EXT_1N_BYTE_SIGNED	64x64~1920x1080
output_hv	GRAY	DATA_TYPE_EXT_U16	64x64~1920x1080

**【数据类型说明】****【说明】** 定义归一化梯度信息计算任务输出控制枚举类型。

```

1 typedef enum bmcv_ive_normgrad_outmode_e{
2     BM_IVE_NORM_GRAD_OUT_HOR_AND_VER = 0x0,
3     BM_IVE_NORM_GRAD_OUT_HOR        = 0x1,
4     BM_IVE_NORM_GRAD_OUT_VER       = 0x2,
5     BM_IVE_NORM_GRAD_OUT_COMBINE   = 0x3,
6 } bmcv_ive_normgrad_outmode;

```

成员名称	描述
BM_IVE_NORM_GRAD_OUT_HOR_AND_VER	同时输出梯度信息的 H、V 分量图。
BM_IVE_NORM_GRAD_OUT_HOR	仅输出梯度信息的 H 分量图。
BM_IVE_NORM_GRAD_OUT_VER	仅输出梯度信息的 V 分量图。
BM_IVE_NORM_GRAD_OUT_COMBINE	输出梯度信息以 package 存储的 HV 图。

**【说明】** 定义归一化梯度信息计算任务输出控制枚举类型。

```

1 typedef struct bmcv_ive_normgrad_ctrl_s{
2     bmcv_ive_normgrad_outmode en_mode;
3     signed char as8_mask[25];
4     unsigned char u8_norm;
5 } bmcv_ive_normgrad_ctrl;

```

成员名称	描述
en_mode	梯度信息输出控制模式。
as8_mask	as8Mask 计算梯度需要的模板。
u8_norm	归一化参数, 取值范围: [1, 13]。

**【返回值】**

该函数成功调用时, 返回 BM\_SUCCESS。

**【注意】**

1. 输入输出图像的 width 都需要 16 对齐。
2. 控制参数中输出模式如下: - BM\_IVE\_NORM\_GRAD\_OUT\_HOR\_AND\_VER 时, output\_h 和 output\_v 指针不能为空, 且要求跨度一致。
  - BM\_IVE\_NORM\_GRAD\_OUT\_HOR 时, output\_h 不能为空。
  - BM\_IVE\_NORM\_GRAD\_OUT\_VER 时, output\_v 不能为空。
  - BM\_IVE\_NORM\_GRAD\_OUT\_COMBINE 时, output\_hv 不能为空。
3. 计算公式如下:

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

**示例代码**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

```

(续下页)

(接上页)

```

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bmcv_ive_normgrad_outmode_enMode = BM_IVE_NORM_GRAD_OUT_
    ↵HOR_AND_VER;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src";
    char *dst_hName = "path/to/dst_h", *dst_vName = "path/to/dst_v";

    bm_handle_t handle = NULL;
    /* 3 by 3 */
    signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, -2, 0,
                                2, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0 };
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src;
    bm_image dst_H, dst_V, dst_combine_HV;
    int src_stride[4];
    int dst_stride[4];

    bmcv_ive_normgrad_ctrl normgrad_attr;
    normgrad_attr.u8_norm = 8;
    normgrad_attr.en_mode = enMode;
    (memcpy(normgrad_attr.as8_mask, arr3by3, 5 * 5 * sizeof(signed char)));

    // calc ive image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
    ↵src, src_stride);
    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
    ↵+ image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void *)((unsigned char *)input_data + image_byte_size[0])},

```

(续下页)

(接上页)

```
(void *)((unsigned char*)input_data + image_byte_size[0] + F
↳image_byte_size[1]),
         (void *)((unsigned char*)input_data + image_byte_size[0] + F
↳image_byte_size[1] + image_byte_size[2]));
bm_image_copy_host_to_device(src, in_ptr);

dst_stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE_
↳SIGNED, &dst_H, dst_stride);
ret = bm_image_alloc_dev_mem(dst_H, BMCV_HEAP1_ID);

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE_
↳SIGNED, &dst_V, dst_stride);
ret = bm_image_alloc_dev_mem(dst_V, BMCV_HEAP1_ID);

ret = bmcv_ive_norm_grad(handle, &src, &dst_H, &dst_V, &dst_combine_HV,
↳normgrad_attr);

signed char* ive_h_res = malloc(width * height * sizeof(signed char));
signed char* ive_v_res = malloc(width * height * sizeof(signed char));
unsigned short* ive_combine_res = malloc(width * height * sizeof(unsigned short));

memset(ive_h_res, 0, width * height * sizeof(signed char));
memset(ive_v_res, 0, width * height * sizeof(signed char));
memset(ive_combine_res, 1, width * height * sizeof(unsigned short));

ret = bm_image_copy_device_to_host(dst_H, (void**)&ive_h_res);
ret = bm_image_copy_device_to_host(dst_V, (void**)&ive_v_res);

FILE *iveH_fp = fopen(dst_hName, "wb");
fwrite((void *)ive_h_res, sizeof(signed char), width * height, iveH_fp);
fclose(iveH_fp);

FILE *iveV_fp = fopen(dst_vName, "wb");
fwrite((void *)ive_v_res, sizeof(signed char), width * height, iveV_fp);
fclose(iveV_fp);

free(input_data);
free(ive_h_res);
free(ive_v_res);
free(ive_combine_res);

bm_image_destroy(&dst_H);
bm_image_destroy(&dst_V);

bm_dev_free(handle);
return ret;
}
```

## 5.92 bmcv\_ive\_sad

### 【描述】

该 API 使用 ive 硬件资源, 计算两幅图像按照 4x4/8x8/16x16 分块的 16bit/8bit SAD 图像, 以及对 SAD 进行阈值化输出。

### 【语法】

```

1 bm_status_t bmcv_ive_sad(
2     bm_handle_t           handle,
3     bm_image *            input,
4     bm_image *            output_sad,
5     bm_image *            output_thr,
6     bmcv_ive_sad_attr *   sad_attr,
7     bmcv_ive_sad_thresh_attr* thresh_attr);

```

### 【参数】

表 5.65: bmcv\_ive\_sad 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
*input	输入	输入 bm_image 对象数组, 存放两个输入图像, 不能为空。
*output_sad	输出	输出 bm_image 对象数据指针, 输出 SAD 图像指针, 根据 sad_attr->en_out_ctrl, 若需要输出则不能为空。根据 sad_attr->en_mode, 对应 4x4、8x8、16x16 分块模式, 高、宽分别为 input1 的 1/4、1/8、1/16。
*output_thr	输出	输出 bm_image 对象数据指针, 输出 SAD 阈值化图像指针, 根据 sad_attr->en_out_ctrl, 若需要输出则不能为空。根据 sad_attr->en_mode, 对应 4x4、8x8、16x16 分块模式, 高、宽分别为 input1 的 1/4、1/8、1/16。
*sad_attr	输入	SAD 控制参数指针结构体, 不能为空。
*thresh_attr	输入	SAD thresh 模式需要的阈值参数结构体指针, 非 thresh 模式情况下, 传入值但是不起作用。

参数名称	图像格式	数据类型	分辨率
input1	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
input2	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input1
output_sad	GRAY	DATA_TYPE_EXT_1N_BYTE DATA_TYPE_EXT_U16	根据 sad_attr.en_mode, 对应 4x4、8x8、16x16 分块 模式, 高、宽分别是 input1 的 1/4、1/8、1/16。
output_thr	GRAY	DATA_TYPE_EXT_1N_BYTE	根据 sad_attr.en_mode, 对应 4x4、8x8、16x16 分块 模式, 高、宽分别是 input1 的 1/4、1/8、1/16。

### 【数据类型说明】

【说明】定义 SAD 计算模式。

```

1 typedef enum bmcv_ive_sad_mode_e{
2     BM_IVE_SAD_MODE_MB_4X4 = 0x0,
3     BM_IVE_SAD_MODE_MB_8X8 = 0x1,
4     BM_IVE_SAD_MODE_MB_16X16 = 0x2,
5 } bmcv_ive_sad_mode;

```

成员名称	描述
BM_IVE_SAD_MODE_MB_4X4	按 4x4 像素块计算 SAD。
BM_IVE_SAD_MODE_MB_8X8	按 8x8 像素块计算 SAD。
BM_IVE_SAD_MODE_MB_16X16	按 16x16 像素块计算 SAD。

【说明】定义 SAD 输出控制模式。

```

1 typedef enum bmcv_ive_sad_out_ctrl_s{
2     BM_IVE_SAD_OUT_BOTH = 0x0,
3     BM_IVE_SAD_OUT_SAD = 0x1,
4     BM_IVE_SAD_OUT_THRESH = 0x3,
5 } bmcv_ive_sad_out_ctrl;

```

成员名称	描述
BM_IVE_SAD_OUT_BOTH	SAD 图和阈值化图输出模式。
BM_IVE_SAD_OUT_SAD	SAD 图输出模式。
BM_IVE_SAD_OUT_THRESH	阈值化输出模式。

【说明】定义 SAD 控制参数。

```
1 typedef struct bmcv_ive_sad_attr_s{  
2     bm_ive_sad_mode en_mode;  
3     bm_ive_sad_out_ctrl en_out_ctrl;  
4 } bmcv_ive_sad_attr;
```

成员名称	描述
en_mode	SAD 计算模式。
en_out_ctrl	SAD 输出控制模式。

```
1 typedef struct bmcv_ive_sad_thresh_attr_s{  
2     unsigned short u16_thr;  
3     unsigned char u8_min_val;  
4     unsigned char u8_max_val;  
5 } bmcv_ive_sad_thresh_attr;
```

成员名称	描述
u16_thr	对计算的 SAD 图进行阈值化的阈值。取值范围依赖 enMode。
u8_min_val	阈值化不超过 u16Thr 时的取值。
u8_max_val	阈值化超过 u16Thr 时的取值。

### 【注意】

- BM\_IVE\_SAD\_OUT\_8BIT\_BOTH: u16\_thr 取值范围: [0, 255];
- BM\_IVE\_SAD\_OUT\_16BIT\_BOTH 或者 BM\_IVE\_SAD\_OUT\_THRESH: u16\_thr 取值范围: [0, 65535]。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 计算公式如下:

$$SAD_{\text{out}}(x, y) = \sum_{\substack{n \cdot x \leq i < n \cdot (x+1) \\ n \cdot y \leq j < n \cdot (j+1)}} |I_1(x, y) - I_2(x, y)|$$

$$THR_{\text{out}}(x, y) = \begin{cases} minVal & (SAD_{\text{out}}(x, y) \leq Thr) \\ maxVal & (SAD_{\text{out}}(x, y) > Thr) \end{cases}$$

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

const int sad_write_img(char* file_name, unsigned char *data,
                        int sad_width, int sad_height, int dsize,
                        int stride){
    unsigned char* tmp = malloc(sad_width * dsize);
    FILE *fp = fopen(file_name, "wb");
    for(int h = 0; h < sad_height; h++, data += stride){
        memcpy(tmp, data, sad_width * dsize);
        fwrite((void *)tmp, 1, sad_width * dsize, fp);
    }
    fclose(fp);
    free(tmp);
    return 0;
}

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    bmvc_ive_sad_mode sadMode = BM_IVE_SAD_MODE_MB_8X8;
    bmvc_ive_sad_out_ctrl sadOutCtrl = BM_IVE_SAD_OUT_BOTH;
    int u16_thr = 0x800;
    int u8_min_val = 2;
    int u8_max_val = 30;
    char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
    char *dstSad_name = "path/to/dstSad", *dstThr_name = "path/to/dstThr";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
}
```

(续下页)

(接上页)

```

int dst_width = 0, dst_height = 0;
bm_image src[2];
bm_image dstSad, dstSadThr;
int stride[4];
int dstSad_stride[4];

bmcv_ive_sad_attr sadAttr;
bmcv_ive_sad_thresh_attr sadthreshAttr;
sadAttr.en_mode = sadMode;
sadAttr.en_out_ctrl = sadOutCtrl;
sadthreshAttr.u16_thr = u16_thr;
sadthreshAttr.u8_min_val = u8_min_val;
sadthreshAttr.u8_max_val = u8_max_val;

bm_image_data_format_ext dst_sad_dtype = DATA_TYPE_EXT_1N_BYTE;
bool flag = false;

// calc ive image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;
bm_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src[0], stride);
bm_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src[1], stride);

dst_sad_dtype = DATA_TYPE_EXT_U16;
data_size = 2;
dstSad_stride[0] = align_up(width, 16) * data_size;
bm_create(handle, height, width, fmt, dst_sad_dtype, &dstSad, dstSad_
stride);
bm_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
dstSadThr, stride);

data_size = sizeof(unsigned short);
dst_width = width / 8;
dst_height = height / 8;

ret = bm_image_alloc_dev_mem(src[0], BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src[1], BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dstSad, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dstSadThr, BMCV_HEAP1_ID);
if (dstSad.data_type == DATA_TYPE_EXT_1N_BYTE){
    flag = true;
}

int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src[i], image_byte_size);
}

```

(续下页)

(接上页)

```
byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
+ image_byte_size[3];
input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(filename[i], "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
printf("file size is less than required bytes%d\n", byte_size);
};
fclose(fp_src);
void* in_ptr[4] = { (void *)input_data,
                    (void *)((unsigned char *)input_data + image_byte_size[0]),
                    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
                    (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src[i], in_ptr);
}

ret = bmcv_ive_sad(handle, src, &dstSad, &dstSadThr, &sadAttr, &
sadthreshAttr);

unsigned char* sad_res = (unsigned char*)malloc(width * height * data_size);
unsigned char* thr_res = (unsigned char*)malloc(width * height);
memset(sad_res, 0, width * height * data_size);
memset(thr_res, 0, width * height);

ret = bm_image_copy_device_to_host(dstSad, (void **)&sad_res);
ret = bm_image_copy_device_to_host(dstSadThr, (void **)&thr_res);

int sad_stride = (flag) ? dstSad_stride[0] / 2 : dstSad_stride[0];
sad_write_img(dstSad_name, sad_res, dst_width, dst_height, data_size, sad_
stride);
sad_write_img(dstThr_name, thr_res, dst_width, dst_height, sizeof(unsigned[F]
char), stride[0]);
free(sad_res);
free(thr_res);

bm_image_destroy(&src[0]);
bm_image_destroy(&src[1]);
bm_image_destroy(&dstSad);
bm_image_destroy(&dstSadThr);

bm_dev_free(handle);
return 0;
}
```

## 5.93 bmcv\_ive\_stcandidcorner

### 【描述】

该 API 使用 ive 硬件资源, 完成灰度图像 Shi-Tomasi-like 角点 (两条边缘的交点) 计算, 角点在任意一个方向上做微小移动, 都会引起该区域的梯度图的方向和幅值发生很大变化。

### 【语法】

```

1 bm_status_t bmcv_ive_stcandidcorner(
2     bm_handle_t          handle,
3     bm_image              input,
4     bm_image              output,
5     bmcv_ive_stcandidcorner_attr stcandidcorner_attr);

```

### 【参数】

表 5.66: bmcv\_ive\_stcandidcorner 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体, 不能为空。
output	输出	输出 bm_image 对象数据结构体, 候选角 点响应值图像指针, 不能为空, 宽、高同 input。
stcandidcorner_attr	输入	Shi-Tomas-like 候选角点计算控制参数结 构体, 不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64 ~ 1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

### 【数据类型说明】

【说明】 定义 Shi-Tomas-like 候选角点计算控制参数。

```

1 typedef struct bmcv_ive_stcandidcorner_attr_s{
2     bm_device_mem_t st_mem;
3     unsigned char u0q8_quality_level;
4 } bmcv_ive_stcandidcorner_attr;

```

成员名称	描述
st_mem	内存配置大小见 bmcv_ive_stcandidcorner 的注意。
u0q8_quality_level	ShiTomas 角点质量控制参数，角点响应值小于 u0q8_quality_level * 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。

【说明】定义 Shi-Tomas-like 角点计算时最大角点响应值结构体。

```

1 typedef struct bmcv_ive_st_max_eig_s{
2     unsigned short u16_max_eig;
3     unsigned char u8_reserved[14];
4 } bmcv_ive_st_max_eig;
```

成员名称	描述
u16_max_eig	最大角点响应值。
u8_reserved	保留位。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 与 OpenCV 中 ShiTomas 角点计算原理类似。
3. stcandidcorner\_attr.st\_mem 至少需要开辟的内存大小:

```
stcandidcorner_attr.st_mem.size = 4 * input_stride * input.height +  
sizeof(bmcv_ive_st_max_eig).
```

4. 该任务完成后, 得到的并不是真正的角点, 还需要进行下一步计算。

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>
```

(续下页)

(接上页)

```
#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    int u0q8QualityLevel = 25;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src, dst;
    int stride[4];

    bmcv_ive_stcandidcorner_attr stCandiCorner_attr;
    memset(&stCandiCorner_attr, 0, sizeof(bmcv_ive_stcandidcorner_attr));

    // calc ive image stride && create bm image struct
    int data_size = 1;
    stride[0] = align_up(width, 16) * data_size;
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &src, stride);
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &dst, stride);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    int attr_len = 4 * height * stride[0] + sizeof(bmcv_ive_st_max_eig);
    ret = bm_malloc_device_byte(handle, &stCandiCorner_attr.st_mem, attr_len * F
    ↪sizeof(unsigned char));
    stCandiCorner_attr.u0q8_quality_level = u0q8QualityLevel;

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] F
    ↪+ image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void*)((unsigned char*)input_data + image_byte_size[0])},

```

(续下页)

(接上页)

```

        (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1]),
        (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1] + image_byte_size[2]));
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_stcandidcorner(handle, src, dst, stCandiCorner_attr);

unsigned char *ive_res = (unsigned char *)malloc(width * height * [F]
↳sizeof(unsigned char));
memset(ive_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void**)&ive_res);

FILE *ive_fp = fopen(dst_name, "wb");
fwrite((void *)ive_res, 1, width * height, ive_fp);
fclose(ive_fp);

free(input_data);
free(ive_res);

bm_image_destroy(&src);
bm_image_destroy(&dst);
bm_free_device(handle, stCandiCorner_attr.st_mem);

bm_dev_free(handle);
return 0;
}

```

#### 5.94 bmcv\_ive\_mag\_and\_ang

##### 【描述】

该 API 使用 ive 硬件资源，创建 5x5 模板，计算灰度图的每个像素区域位置灰度值变化梯度的幅值和幅角任务。

##### 【语法】

```

1 bm_status_t bmcv_ive_mag_and_ang(
2     bm_handle_t handle,
3     bm_image * input,
4     bm_image * mag_output,
5     bm_image * ang_output,
6     bmcv_ive_mag_and_ang_ctrl attr);

```

## 【参数】

表 5.67: bmcv\_ive\_magandang 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*input	输入	输入 bm_image 对象指针，不能为空。
*mag_output	输出	输出 bm_image 对象数据指针，输出幅值图像，不能为空，宽、高同 input。
*ang_output	输出	输出 bm_image 对象数据指针，输出幅角图像。根据 attr.en_out_ctrl，需要输出则不能为空，宽、高同 input。
attr	输入	控制信息结构体，存放梯度幅值和幅角计算的控制信息，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
mag_output	GRAY	DATA_TYPE_EXT_U16	同 input
ang_output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

## 【数据类型说明】

【说明】定义梯度幅值与角度计算的输出格式。

```

1 typedef enum bmcv_ive_mag_and_ang_outctrl_e{
2     BM_IVE_MAG_AND_ANG_OUT_MAG = 0x0,
3     BM_IVE_MAG_AND_ANG_OUT_ALL = 0X1,
4 } bmcv_ive_mag_and_ang_outctrl;

```

成员名称	描述
BM_IVE_MAG_AND_ANG_OUT_MAG	仅输出幅值。
BM_IVE_MAG_AND_ANG_OUT_ALL	同时输出幅值和角度值。

【说明】定义梯度幅值和幅角计算的控制信息。

```

1 typedef struct bmcv_ive_mag_and_ang_ctrl_s{
2     bmcv_ive_mag_and_ang_outctrl    en_out_ctrl;
3     unsigned short                  u16_thr;
4     signed char                     as8_mask[25];
5 } bmcv_ive_mag_and_ang_ctrl;

```

成员名称	描述
en_out_ctrl	输出格式。
u16_thr	用于对幅值进行阈值化的阈值。
as8_mask	5x5 模板系数。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 可配置两种输出模式, 详细请参考 bm\_ive\_mag\_and\_ang\_outctrl。
3. 配置 BM\_IVE\_MAG\_AND\_ANG\_OUT\_ALL 时, 要求 mag\_output 与 ang\_output 跨度一致。
4. 用户可以通过 attr.u16\_thr 对幅值图进行阈值化操作 (可用来实现 EOH), 计算公式如下:

$$\text{Mag}(x, y) = \begin{cases} 0 & (\text{Mag}(x, y) < u16_{thr}) \\ \text{Mag}(x, y) & (\text{Mag}(x, y) \geq u16_{thr}) \end{cases}$$

其中,  $\text{Mag}(x, y)$  对应 mag\_output。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    // int templateSize = 0; // 0: 3x3 1:5x5
    unsigned short thrValue = 0;
    bm_image_format_ext fmt = FORMAT_GRAY;
    bmvc_ive_mag_and_ang_outctrl enMode = BM_IVE_MAG_AND_ANG_

```

(续下页)

(接上页)

```

    ↵OUT_ALL;
    char *src_name = "path/to/src";
    char *dst_magName = "path/to/mag_dst", *dst_angName = "path/to/ang_dst";
    /* 3 by 3 */
    signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, -2, 0,
                                2, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0 };
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src;
    bm_image dst_mag, dst_ang;
    int stride[4];
    int magStride[4];

    bmcv_ive_mag_and_ang_ctrl magAndAng_attr;
    memset(&magAndAng_attr, 0, sizeof(bmcv_ive_mag_and_ang_ctrl));
    magAndAng_attr.en_out_ctrl = enMode;
    magAndAng_attr.ul6_thr = thrValue;
    memcpy(magAndAng_attr.as8_mask, arr3by3, 5 * 5 * sizeof(signed char));

    // calc ive image stride && create bm image struct
    int data_size = 1;
    stride[0] = align_up(width, 16) * data_size;
    data_size = 2;
    magStride[0] = align_up(width, 16) * data_size;

    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
    ↵src, stride);
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_U16, &dst_
    ↵mag, magStride);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst_mag, BMCV_HEAP1_ID);

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
    ↵+ image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void *)((unsigned char *)input_data + image_byte_size[0]),
                       (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
    ↵image_byte_size[1])},

```

(续下页)

(接上页)

```
(void *)((unsigned char*)input_data + image_byte_size[0] + [F]
    ↵image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
    ↵dst_ang, stride);
ret = bm_image_alloc_dev_mem(dst_ang, BMCV_HEAP1_ID);

ret = bmcv_ive_mag_and_ang(handle, &src, &dst_mag, &dst_ang,[F]
    ↵magAndAng_attr);

unsigned short* magOut_res = malloc(width * height * sizeof(unsigned short));
unsigned char* angOut_res = malloc(width * height * sizeof(unsigned char));
memset(magOut_res, 0, width * height * sizeof(unsigned short));
memset(angOut_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst_mag, (void **)&magOut_res);
ret = bm_image_copy_device_to_host(dst_ang, (void **)&angOut_res);

FILE *mag_fp = fopen(dst_magName, "wb");
fwrite((void *)magOut_res, 1, width * height, mag_fp);
fclose(mag_fp);

FILE *ang_fp = fopen(dst_angName, "wb");
fwrite((void *)angOut_res, 1, width * height, ang_fp);
fclose(ang_fp);

free(input_data);
free(magOut_res);
free(angOut_res);

bm_image_destroy(&src);
bm_image_destroy(&dst_mag);
bm_image_destroy(&dst_ang);

bm_dev_free(handle);
return 0;
}
```

## 5.95 bmcv\_ive\_map

### 【描述】

该 API 使用 ive 硬件资源, 创建 Map (映射赋值) 任务, 对源图像中的每个像素, 查找 Map 查找表中的值, 赋予目标图像相应像素查找表中的值, 支持 U8->U8, U8->U16, U8->S16 三种模式的映射。

### 【语法】

```

1 bm_status_t bmcv_ive_map(
2     bm_handle_t    handle,
3     bm_image      input,
4     bm_image      output,
5     bm_device_mem_t map_table);

```

### 【参数】

表 5.68: bmcv\_ive\_map 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体, 不能为空。
output	输出	输出 bm_image 对象结构体, 不能为空, 宽、高同 input。
map_table	输入	bm_device_mem_t 类型结构体, 存储映射表信息。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE DATA_TYPE_EXT_U16 DATA_TYPE_EXT_S16	64x64~1920x1080

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。

2. 计算公式如下：

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

其中， $I(x, y)$  对应 input， $I_{\text{out}}(x, y)$ ，对应 output。

3. 映射表信息存放在大小为 256 元素的数组中，可以通过 s2d 实现，映射表拷贝到设备内存。

DATA\_TYPE\_EXT\_1N\_BYTE 的输出，数组元素对应 U8 类型；DATA\_TYPE\_EXT\_S16 的输出，数组元素对应 S16 类型；DATA\_TYPE\_EXT\_U16 的输出，数组元素对应 U16 类型。

## 【实例说明】

DATA\_TYPE\_EXT\_1N\_BYTE 的输出，数组元素对应 U8 类型，假设映射表信息，如下：

```

1 static unsigned char FixMap[256] = {
2     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
3     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
4     0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x02, 0x02, 0x03,
5     0x03, 0x04, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
6     0x0C, 0x0D, 0x0F, 0x10, 0x11, 0x12, 0x14, 0x15, 0x17, 0x18,
7     0x1A, 0x1B, 0x1D, 0x1E, 0x20, 0x21, 0x23, 0x24, 0x26, 0x27,
8     0x29, 0x2A, 0x2C, 0x2D, 0x2F, 0x31, 0x32, 0x34, 0x35, 0x37,
9     0x38, 0x3A, 0x3B, 0x3D, 0x3E, 0x40, 0x41, 0x43, 0x44, 0x45,
10    0x47, 0x48, 0x4A, 0x4B, 0x4D, 0x4E, 0x50, 0x51, 0x52, 0x54,
11    0x55, 0x56, 0x58, 0x59, 0x5A, 0x5B, 0x5D, 0x5E, 0x5F, 0x60,
12    0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x69, 0x6A, 0x6B, 0x6D,
13    0x6E, 0x70, 0x71, 0x73, 0x75, 0x76, 0x78, 0x7A, 0x7B, 0x7D,
14    0x7E, 0x80, 0x81, 0x83, 0x84, 0x86, 0x87, 0x88, 0x89, 0x8B,
15    0x8C, 0x8D, 0x8E, 0x90, 0x92, 0x94, 0x97, 0x9A, 0x9C, 0x9E,
16    0xA1, 0xA3, 0xA5, 0xA6, 0xA7, 0xA9, 0xAA, 0xAB, 0xAC, 0xAC,
17    0xAD, 0xAE, 0xAF, 0xB0, 0xB1, 0xB3, 0xB4, 0xB5, 0xB7, 0xB9,
18    0xBB, 0xBD, 0xBF, 0xC1, 0xC4, 0xC7, 0xCC, 0xD1, 0xD5, 0xDA,
19    0xDE, 0xE0, 0xE2, 0xE3, 0xE4, 0xE5, 0xE5, 0xE6, 0xE6, 0xE6,
20    0xE6, 0xE6, 0xE7, 0xE7, 0xE7, 0xE8, 0xE8, 0xE9, 0xEA, 0xEC,
21    0xED, 0xEE, 0xF0, 0xF2, 0xF4, 0xF5, 0xF7, 0xF8, 0xFA, 0xFB,
22    0xFD, 0xFE, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
23    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
24    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
25    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
26    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
27    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
28 };

```

假设输入图像的第一个像素大小是 10，那么就找到 FixMap[10] 元素的值幅值给输出图像的第一个元素；然后输入的第二个像素，如果大小是 20，那么就将 FixMap[20] 的值幅值给输出图像的第二个像素。

### 【代码示例】

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "bmvc_api_ext_c.h"
5 #include <unistd.h>
6
7 #define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
8
9
10 static unsigned char FixMap[256] = {
11     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
12     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
13     0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x02, 0x02, 0x03,
14     0x03, 0x04, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B,
15     0x0C, 0x0D, 0x0F, 0x10, 0x11, 0x12, 0x14, 0x15, 0x17, 0x18,
16     0x1A, 0x1B, 0x1D, 0x1E, 0x20, 0x21, 0x23, 0x24, 0x26, 0x27,
17     0x29, 0x2A, 0x2C, 0x2D, 0x2F, 0x31, 0x32, 0x34, 0x35, 0x37,
18     0x38, 0x3A, 0x3B, 0x3D, 0x3E, 0x40, 0x41, 0x43, 0x44, 0x45,
19     0x47, 0x48, 0x4A, 0x4B, 0x4D, 0x4E, 0x50, 0x51, 0x52, 0x54,
20     0x55, 0x56, 0x58, 0x59, 0x5A, 0x5B, 0x5D, 0x5E, 0x5F, 0x60,
21     0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x69, 0x6A, 0x6B, 0x6D,
22     0x6E, 0x70, 0x71, 0x73, 0x75, 0x76, 0x78, 0x7A, 0x7B, 0x7D,
23     0x7E, 0x80, 0x81, 0x83, 0x84, 0x86, 0x87, 0x88, 0x89, 0x8B,
24     0x8C, 0x8D, 0x8E, 0x90, 0x92, 0x94, 0x97, 0x9A, 0x9C, 0x9E,
25     0xA1, 0xA3, 0xA5, 0xA6, 0xA7, 0xA9, 0xAA, 0xAB, 0xAC, 0xAC,
26     0xAD, 0xAE, 0xAF, 0xB0, 0xB1, 0xB3, 0xB4, 0xB5, 0xB7, 0xB9,
27     0xBB, 0xBD, 0xBF, 0xC1, 0xC4, 0xC7, 0xCC, 0xD1, 0xD5, 0xDA,
28     0xDE, 0xE0, 0xE2, 0xE3, 0xE4, 0xE5, 0xE5, 0xE6, 0xE6, 0xE6,
29     0xE6, 0xE6, 0xE7, 0xE7, 0xE7, 0xE8, 0xE8, 0xE9, 0xEA, 0xEC,
30     0xED, 0xEE, 0xF0, 0xF2, 0xF4, 0xF5, 0xF7, 0xF8, 0xFA, 0xFB,
31     0xFD, 0xFE, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
32     0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
33     0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
34     0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
35     0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
36     0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
37 };
38
39 int main() {
40     int dev_id = 0;
41     int height = 1080, width = 1920;
42     bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_GRAY;
43     char *src_name = "path/to/src", *dst_name = "path/to/dst";

```

(续下页)

(接上页)

```

44
45     bm_handle_t handle = NULL;
46     int ret = (int)bm_dev_request(&handle, dev_id);
47     if (ret != 0) {
48         printf("Create bm handle failed. ret = %d\n", ret);
49         exit(-1);
50     }
51     bm_image src, dst;
52     bm_device_mem_t mapTable;
53     int src_stride[4];
54     int dst_stride[4];
55
56     // calculate image stride
57     int data_size = 1;
58     src_stride[0] = align_up(width, 16) * data_size;
59     dst_stride[0] = align_up(width, 16) * data_size;
60
61     // create bm image struct
62     bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_BYTE, &src, src_
63     ↪stride);
64     bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_BYTE, &dst, [F]
65     ↪dst_stride);
66
67     // alloc bm image memory
68     ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
69     ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);
70
71     ret = bm_malloc_device_byte(handle, &mapTable, MAP_TABLE_SIZE);
72     ret = bm_memcpy_s2d(handle, mapTable, FixMap);
73
74     // read image data from input files
75     int image_byte_size[4] = {0};
76     bm_image_get_byte_size(src, image_byte_size);
77     int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + image_
78     ↪byte_size[3];
79     unsigned char *input_data = (unsigned char *)malloc(byte_size);
80     FILE *fp_src = fopen(src_name, "rb");
81     if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
82         printf("file size is less than required bytes%d\n", byte_size);
83     };
84     fclose(fp_src);
85     void* in_ptr[4] = {(void *)input_data,
86                         ((void *)((unsigned char *)input_data + image_byte_size[0])),
87                         ((void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
88                         ↪size[1])),
89                         ((void *)((unsigned char *)input_data + image_byte_size[0] + image_byte_
90                         ↪size[1] + image_byte_size[2]))};
91     bm_image_copy_host_to_device(src, in_ptr);
92
93     ret = bmcv_ive_map(handle, src, dst, mapTable);
94
95

```

(续下页)

(接上页)

```
90 unsigned char* ive_res = (unsigned char*) malloc (width * height * sizeof(unsigned char));
91 memset(ive_res, 0, width * height * sizeof(unsigned char));
92
93 ret = bm_image_copy_device_to_host(dst, (void**)&ive_res);
94 FILE *fp = fopen(dst_name, "wb");
95 fwrite((void *)ive_res, 1, width * height * sizeof(unsigned char), fp);
96 fclose(fp);
97
98 free(input_data);
99 free(ive_res);
100
101 bm_image_destroy(&src);
102 bm_image_destroy(&dst);
103 bm_free_device(handle, mapTable);
104
105 bm_dev_free(handle);
106
107 return 0;
108 }
```

## 5.96 bmcv\_ive\_ncc

### 【描述】

该 API 使用 ive 硬件资源，创建两相同分辨率灰度图像的归一化互相关系数计算任务。

### 【语法】

```
1 bm_status_t bmcv_ive_ncc(
2     bm_handle_t     handle,
3     bm_image       input1,
4     bm_image       input2,
5     bm_device_mem_t output);
```

### 【参数】

表 5.69: bmcv\_ive\_ncc 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出数据对象结构体，不能为空，内存至少需要配置: sizeof(bmcv_ive_ncc_dst_mem_t)。

参数名称	图像格式	数据类型	分辨率
input1	GRAY	DATA_TYPE_EXT_1N_BYTE	32x32~1920x1080
input2	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input1
output	-	-	-

**【数据类型说明】**

**【说明】** 定义 LBP 计算的比较模式。

```

1 typedef struct _bmcv_ive_ncc_dst_mem_s{
2     unsigned long long u64_numerator;
3     unsigned long long u64_quad_sum1;
4     unsigned long long u64_quad_sum2;
5     unsigned char u8_reserved[8];
6 }bmcv_ive_ncc_dst_mem_t;
```

成员名称	描述
u64_numerator	$\sum_{i=1}^w \sum_{j=1}^h (I_{src1}(i, j) * I_{src2}(i, j))$
u64_quad_sum1	$\sum_{i=1}^w \sum_{j=1}^h I_{src1}^2(i, j)$
u64_quad_sum2	$\sum_{i=1}^w \sum_{j=1}^h I_{src2}^2(i, j)$
u8_reserved	保留字段。

**【返回值】**

该函数成功调用时，返回 BM\_SUCCESS。

**【注意】**

- 输入图像的 width 都需要 16 对齐。

- 计算公式如下：

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

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 288, width = 352;;
    bm_image_format_ext src_fmt = FORMAT_GRAY;
    char* src1_name = "path/to/src1", *src2_name = "path/to/src2";
    char* dst_name = "/path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src1, src2;
    bm_device_mem_t dst;
    int src_stride[4];

    // calculate image stride && create bm image struct
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;

    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src1, src_stride);
    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src2, src_stride);
    ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);

    int byte_size;
    unsigned char *input_data;
    int image_byte_size[4] = {0};
    char *filename[] = {src1_name, src2_name};
    bm_image src_images[] = {src1, src2};
```

(续下页)

(接上页)

```

for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] * F
    ↵+ image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                        (void *)((unsigned char *)input_data + image_byte_size[0]),
                        (void *)((unsigned char *)input_data + image_byte_size[0] + F
                        ↵image_byte_size[1]),
                        (void *)((unsigned char *)input_data + image_byte_size[0] + F
                        ↵image_byte_size[1] + image_byte_size[2])};
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

int data_len = sizeof(bmcv_ive_ncc_dst_mem_t);

ret = bm_malloc_device_byte(handle, &dst, data_len);

ret = bmcv_ive_ncc(handle, src1, src2, dst);

unsigned long long *ncc_result = malloc(data_len);
ret = bm_memcpy_d2s(handle, ncc_result, dst);
unsigned long long *numerator = ncc_result;
unsigned long long *quadSum1 = ncc_result + 1;
unsigned long long *quadSum2 = quadSum1 + 1;
float fr = (float)((double)*numerator / (sqrt((double)*quadSum1) * F
    ↵sqrt((double)*quadSum2)));
printf("bmcv ive NCC value is %f \n", fr);

FILE *ncc_result_fp = fopen(dst_name, "wb");
fwrite((void *)ncc_result, 1, data_len, ncc_result_fp);
fclose(ncc_result_fp);

free(input_data);
free(ncc_result);

bm_free_device(handle, dst);
bm_dev_free(handle);
return 0;
}

```

## 5.97 bmcv\_ive\_ord\_stat\_filter

### 【描述】

该 API 使用 ive 硬件资源，创建 3x3 模板顺序统计量滤波任务，可进行 Median、Max、Min 滤波。

### 【语法】

```

1 bm_status_t bmcv_ive_ord_stat_filter(
2     bm_handle_t          handle,
3     bm_image_t           input,
4     bm_image_t           output,
5     bmcv_ive_ord_stat_filter_mode mode);

```

### 【参数】

表 5.70: bmcv\_ive\_ord\_stat\_filter 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空，宽、高同 input。
mode	输入	控制顺序统计量滤波的模式。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input1

### 【数据类型说明】

【说明】 定义顺序统计量滤波模式。

```

1   bmcv_ord_stat_filter_mode_e{
2       BM_IVE_ORD_STAT_FILTER_MEDIAN = 0x0,

```

(续下页)

(接上页)

```

3   BM_IVE_ORD_STAT_FILTER_MAX = 0x1,
4   BM_IVE_ORD_STAT_FILTER_MIN = 0x2,
5 } bmcv_ive_ord_stat_filter_mode;

```

成员名称	描述
BM_IVE_ORD_STAT_FILTER_MEDIAN	中值滤波
BM_IVE_ORD_STAT_FILTER_MAX	最大值滤波, 等价于灰度图的膨胀。
BM_IVE_ORD_STAT_FILTER_MIN	最小值滤波, 等价于灰度图的腐蚀。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 可配置 3 种滤波模式, 参考 bmcv\_ive\_ord\_stat\_filter\_mode 说明。
3. 计算公式如下:
  - BM\_IVE\_ORD\_STAT\_FILTER\_MEDIAN:

$$I_{\text{out}}(x, y) = \text{median}\left\{ \underset{-1 < j < 1}{\underset{-1 < i < 1}{I(x + i, y + j)}} \right\}$$

- BM\_IVE\_ORD\_STAT\_FILTER\_MAX:

$$I_{\text{out}}(x, y) = \max\left\{ \underset{-1 < j < 1}{\underset{-1 < i < 1}{I(x + i, y + j)}} \right\}$$

- BM\_IVE\_ORD\_STAT\_FILTER\_MIN:

$$I_{\text{out}}(x, y) = \min\left\{ \underset{-1 < j < 1}{\underset{-1 < i < 1}{I(x + i, y + j)}} \right\}$$

其中,  $I(x, y)$  对应 input,  $I_{\text{out}}(x, y)$  对应 output。

### 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    bmvc_ive_ord_stat_filter_mode ordStatFilterMode = BM_IVE_ORD_STAT_
    ↵ FILTER_MEDIAN;
    char *src_name = "path/to/src", *dst_name = "path/to/dst";

    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src, dst;
    int stride[4];

    // calc ive image stride && create bm image struct
    int data_size = 1;
    stride[0] = align_up(width, 16) * data_size;
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
    ↵ src, stride);
    bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
    ↵ dst, stride);

    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    int image_byte_size[4] = {0};
    bm_image_get_byte_size(src, image_byte_size);
    int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
    ↵ + image_byte_size[3];
    unsigned char *input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(src_name, "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%ld\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void *)((unsigned char *)input_data + image_byte_size[0]),
                       (void *)((unsigned char *)input_data + image_byte_size[0] + image_

```

(续下页)

(接上页)

```

    ↵byte_size[1]),
    ↵        (void *)((unsigned char*)input_data + image_byte_size[0] + image_
    ↵byte_size[1] + image_byte_size[2]));
    ↵bm_image_copy_host_to_device(src, in_ptr);

    ret = bmcv_ive_ord_stat_filter(handle, src, dst, ordStatFilterMode);

    unsigned char* ordStatFilter_res = malloc(width * height * sizeof(unsigned char));
    memset(ordStatFilter_res, 0, width * height * sizeof(unsigned char));

    ret = bm_image_copy_device_to_host(dst, (void **)&ordStatFilter_res);

    FILE *ive_result_fp = fopen(dst_name, "wb");
    fwrite((void *)ordStatFilter_res, 1, width * height, ive_result_fp);
    fclose(ive_result_fp);

    free(ordStatFilter_res);

    bm_image_destroy(&src);
    bm_image_destroy(&dst);

    bm_dev_free(handle);
    return 0;
}

```

## 5.98 bmcv\_ive\_sobel

### 【描述】

该 API 使用 ive 硬件资源，创建 5x5 模板 Sobel-like 梯度计算任务，计算图像梯度方向的近似灰度值，计算过程与 mag\_and\_ang 完全相同。

### 【语法】

```

1  bm_status_t bmcv_ive_sobel(
2      bm_handle_t      handle,
3      bm_image *       input,
4      bm_image *       output_h,
5      bm_image *       output_v,
6      bmcv_ive_sobel_ctrl  sobel_attr);

```

### 【参数】

表 5.71: bmcv\_ive\_sobel 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*input	输入	输入 bm_image 对象指针，不能为空。
*output_h	输出	输出 bm_image 对象指针，由模板直接滤波得到的梯度分量图像 H 指针，根据 sobel_attr->enOutCtrl，若需要输出则不能为空，宽、高同 input。
*output_v	输出	输出 bm_image 对象指针，由模板直接滤波得到的梯度分量图像 V 指针，根据 sobel_attr->enOutCtrl，若需要输出则不能为空，宽、高同 input。
sobel_attr	输入	控制信息结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	16x16~1920x1080
hOutput	GRAY	DATA_TYPE_EXT_S16	同 input
vOutput	GRAY	DATA_TYPE_EXT_S16	同 input

### 【数据类型说明】

【说明】定义 Sobel 输出控制信息。

```

1 typedef enum bmcv_ive_sobel_out_mode_e{
2     BM_IVE_SOBEL_OUT_MODE_BOTH = 0x0,
3     BM_IVE_SOBEL_OUT_MODE_HOR = 0x1,
4     BM_IVE_SOBEL_OUT_MODE_VER = 0x2,
5 } bmcv_ive_sobel_out_mode;

```

成员名称	描述
BM_IVE_SOBEL_OUT_MODE_BOTH	同时输出用模板和转置模板滤波的结果。
BM_IVE_SOBEL_OUT_MODE_HOR	仅输出用模板直接滤波的结果。
BM_IVE_SOBEL_OUT_MODE_VER	仅输出用转置模板滤波的结果。

【说明】定义 Sobel-like 梯度计算控制信息。

```

1 typedef struct bmcv_ive_sobel_ctrl_s{
2     bmcv_ive_sobel_out_mode sobel_mode;
3     signed char as8_mask[25];
4 } bmcv_ive_sobel_ctrl;

```

成员名称	描述
sobel_mode	出控制枚举参数。
as8_mask	5x5 模板系数。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. 可配置 3 种滤波模式, 参考 bmcv\_ive\_sobel\_out\_mode 说明。
3. 当输出模式为 BM\_IVE\_SOBEL\_OUT\_MODE\_BOTH 时, 要求 output\_h 与 output\_v 跨度一致。
4. 计算公式如下:

$$Hout(x, y) = \sum_{-2 < i < 2} \sum_{-2 < j < 2} I(x + i, y + j) \cdot \text{coef}(i, j)$$

$$Vout(x, y) = \sum_{-2 < j < 2} \sum_{-2 < i < 2} I(x + i, y + j) \cdot \text{coef}(j, i)$$

其中,  $I(x, y)$  对应 input,  $Hout(x, y)$  对应 output\_h,  $Vout(x, y)$  对应 output\_v,  $\text{coef}(mask)$  是 sobel\_attr.as8\_mask。

5. 模板
  - Sobel 模板

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 2 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- ### · Scharr 模板

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 3 & 0 \\ 0 & -10 & 0 & 10 & 0 \\ 0 & -3 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & -10 & -3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 10 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- ## · 拉普拉斯模板

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & 0 \\ 0 & -1 & 8 & -1 & 0 \\ 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bmcv_ive_sobel_out_mode enMode = BM_IVE_SOBEL_OUT_MODE_
→ BOTH;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src";
    char *sobel_hName = "path/to/sobel_h", *sobel_vName = "path/to/sobel_v"
→ ";
    bm_handle_t handle = NULL;
    /* 3 by 3 */
    signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, -2, 0,
        2, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0 };
```

(续下页)

(接上页)

```

int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}

bm_image src;
bm_image dst_H, dst_V;
int src_stride[4];
int dst_stride[4];

bmcv_ive_sobel_ctrl sobelAtt;
sobelAtt.sobel_mode = enMode;
memcpy(sobelAtt.as8_mask, arr3by3, 5 * 5 * sizeof(signed char));

// calculate image stride && create bm image struct
int data_size = 1;
src_stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_
→BYTE, &src, src_stride);
ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_
→size[2] + image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_
→size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
    (void *)((unsigned char *)input_data + image_byte_size[0]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + F_
→image_byte_size[1]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + F_
→image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);

data_size = 2;
dst_stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_S16, &
→dst_H, dst_stride);
ret = bm_image_alloc_dev_mem(dst_H, BMCV_HEAP1_ID);

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_S16, &
→dst_V, dst_stride);
ret = bm_image_alloc_dev_mem(dst_V, BMCV_HEAP1_ID);

ret = bmcv_ive_sobel(handle, &src, &dst_H, &dst_V, sobelAtt);

```

(续下页)

(接上页)

```
signed short* iveSobel_h = malloc(width * height * sizeof(signed short));
signed short* iveSobel_v = malloc(width * height * sizeof(signed short));
memset(iveSobel_h, 0, width * height * sizeof(signed short));
memset(iveSobel_v, 0, width * height * sizeof(signed short));

ret = bm_image_copy_device_to_host(dst_H, (void**)&iveSobel_h);
ret = bm_image_copy_device_to_host(dst_V, (void**)&iveSobel_v);

FILE *sobelH_fp = fopen(sobel_hName, "wb");
fwrite((void *)iveSobel_h, sizeof(signed short), width * height, sobelH_fp);
fclose(sobelH_fp);

FILE *sobelV_fp = fopen(sobel_vName, "wb");
fwrite((void *)iveSobel_v, sizeof(signed short), width * height, sobelV_fp);
fclose(sobelV_fp);

free(input_data);
free(iveSobel_h);
free(iveSobel_v);

bm_image_destroy(&src);
if(enMode == BM_IVE_SOBEL_OUT_MODE_BOTH || enMode == BM_
IVE_SOBEL_OUT_MODE_HOR)
    bm_image_destroy(&dst_H);

if(enMode == BM_IVE_SOBEL_OUT_MODE_BOTH || enMode == BM_
IVE_SOBEL_OUT_MODE_VER)
    bm_image_destroy(&dst_V);

bm_dev_free(handle);
return ret;
}
```

### 5.99 bmcv\_ive\_gmm

#### 【描述】

该 API 使用 ive 硬件资源, 创建 GMM 背景建模任务, 支持灰度图、RGB\_PACKED 图像的 GMM 背景建模, 高斯模型个数为 3 或者 5。

#### 【语法】

```

1 bm_status_t bmcv_ive_gmm(
2     bm_handle_t     handle,
3     bm_image       input,
4     bm_image       output_fg,
5     bm_image       output_bg,
6     bm_device_mem_t output_model,
7     bmcv_ive_gmm_ctrl gmm_attr);

```

## 【参数】

表 5.72: bmcv\_ive\_gmm 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output_fg	输出	输出 bm_image 对象结构体，表示前景图像，不能为空，宽、高同 input。
output_bg	输出	输出 bm_image 对象结构体，表示背景图像，不能为空，宽、高同 input。
output_model	输入、输出	bm_device_mem_t 对象结构体，对应 GMM 模型参数，不能为空。
gmm_attr	输入	控制信息结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY RGB_PACKED	DATA_TYPE_EXT_1N_BYTl	64x64~1920x1080
output_fg	GRAY 二值图	DATA_TYPE_EXT_1N_BYTl	同 input
output_bg	同 input	DATA_TYPE_EXT_1N_BYTl	同 input

## 【数据类型说明】

【说明】 定义 GMM 背景建模的控制参数。

```

1 typedef struct bmcv_ive_gmm_ctrl_s{
2     unsigned int u22q10_noise_var;
3     unsigned int u22q10_max_var;
4     unsigned int u22q10_min_var;
5     unsigned short u0q16_learn_rate;
6     unsigned short u0q16_bg_ratio;
7     unsigned short u8q8_var_thr;

```

(续下页)

(接上页)

```

8   unsigned short u0q16_init_weight;
9   unsigned char u8_model_num;
10 } bmcv_ive_gmm_ctrl;
```

成员名称	描述
u22q10_noise_var	初始噪声方差。 取值范围: [0x1, 0xFFFFFFF]。
u22q10_max_var	模型方差的最大值。 取值范围: [0x1, 0xFFFFFFF]。
u22q10_min_var	模型方差的最小值。 取值范围: [0x1, u22q10MaxVar]。
u0q16_learn_rate	学习速率。 取值范围: [1, 65535]。
u0q16_bg_ratio	背景比例阈值。 取值范围: [1, 65535]。
u8q8_var_thr	方差阈值。 取值范围: [1, 65535]。
u0q16_init_weight	初始权重。 取值范围: [1, 65535]。
u8_model_num	模型个数。 取值范围: {3, 5}。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. GMM 的实现方式参考了 OpenCV 中的 MOG 和 MOG2。
3. 源图像类型只能为 U8C1 或 U8C3\_PACKAGE, 分别用于灰度图和 RGB 图的 GMM 背景建模。
4. 前景图像是二值图, 类型只能为 U8C1; 背景图像与源图像类型一致。
5. 灰度图像或 RGB 图像 GMM 采用 n 个 (n=3 或 5) 高斯模型。

## 示例代码

```

#include <stdio.h>
#include <stdlib.h>
```

(续下页)

(接上页)

```

#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext src_fmt = FORMAT_GRAY;
    char *input_name = "path/to/input";
    char *dstFg_name = "path/to/dst_Fg", *dstBg_name = "path/to/dst_Bg";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src;
    bm_image dst_fg, dst_bg;
    bm_device_mem_t dst_model;
    int stride[4];

    unsigned int u32FrameNumMax = 32;
    unsigned int u32FrmCnt = 0;

    bmvc_ive_gmm_ctrl gmmAttr;
    gmmAttr.u0q16_bg_ratio = 45875;
    gmmAttr.u0q16_init_weight = 3277;
    gmmAttr.u22q10_noise_var = 225 * 1024;
    gmmAttr.u22q10_max_var = 2000 * 1024;
    gmmAttr.u22q10_min_var = 200 * 1024;
    gmmAttr.u8q8_var_thr = (unsigned short)(256 * 6.25);
    gmmAttr.u8_model_num = 3;

    int model_len = width * height * gmmAttr.u8_model_num * 8;

    unsigned char* inputData = malloc(width * height * u32FrameNumMax * sizeof(unsigned char));
    FILE *input_fp = fopen(input_name, "rb");
    fread((void *)inputData, 1, width * height * u32FrameNumMax * sizeof(unsigned char), input_fp);
    fclose(input_fp);

    unsigned char* srcData = malloc(width * height * sizeof(unsigned char));
    unsigned char* ive_fg_res = malloc(width * height * sizeof(unsigned char));
    unsigned char* ive_bg_res = malloc(width * height * sizeof(unsigned char));
    unsigned char* model_data = malloc(model_len * sizeof(unsigned char));

    memset(srcData, 0, width * height * sizeof(unsigned char));

```

(续下页)

(接上页)

```

memset(ive_fg_res, 0, width * height * sizeof(unsigned char));
memset(ive_bg_res, 0, width * height * sizeof(unsigned char));
memset(model_data, 0, model_len * sizeof(unsigned char));

// calc ive image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src, stride);
bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
EXT_1N_BYTE, &dst_fg, stride);
bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
EXT_1N_BYTE, &dst_bg, stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst_fg, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst_bg, BMCV_HEAP1_ID);

ret = bm_malloc_device_byte(handle, &dst_model, model_len);

ret = bm_memcpy_s2d(handle, dst_model, model_data);
for(u32FrmCnt = 0; u32FrmCnt < u32FrameNumMax; u32FrmCnt++){
    if(width > 480 ){
        for(int j = 0; j < 288; j++){
            memcpy(srcData + (j * width),
                   inputData + (u32FrmCnt * 352 * 288 + j * 352), 352);
            memcpy(srcData + (j * width + 352),
                   inputData + (u32FrmCnt * 352 * 288 + j * 352), 352);
        }
    } else {
        for(int j = 0; j < 288; j++){
            memcpy(srcData + j * stride[0],
                   inputData + u32FrmCnt * width * 288 + j * width, width);
            int s = stride[0] - width;
            memset(srcData + j * stride[0] + width, 0, s);
        }
    }
}

ret = bm_image_copy_host_to_device(src, (void**)&srcData);

if(u32FrmCnt >= 500)
    gmmAttr.u0q16_learn_rate = 131; //0.02
else
    gmmAttr.u0q16_learn_rate = 65535 / (u32FrmCnt + 1);

ret = bmcv_ive_gmm(handle, src, dst_fg, dst_bg, dst_model, gmmAttr);
}

ret = bm_image_copy_device_to_host(dst_fg, (void**)&ive_fg_res);
ret = bm_image_copy_device_to_host(dst_bg, (void**)&ive_bg_res);

```

(续下页)

(接上页)

```
FILE *fg_fp = fopen(dstFg_name, "wb");
fwrite((void *)ive_fg_res, 1, width * height, fg_fp);
fclose(fg_fp);

FILE *bg_fp = fopen(dstBg_name, "wb");
fwrite((void *)ive_bg_res, 1, width * height, bg_fp);
fclose(bg_fp);

free(inputData);
free(ive_fg_res);
free(srcData);
free(ive_bg_res);
free(model_data);
bm_image_destroy(&src);
bm_image_destroy(&dst_fg);
bm_image_destroy(&dst_bg);
bm_free_device(handle, dst_model);

bm_dev_free(handle);
return 0;
}
```

## 5.100 bmcv\_ive\_gmm2

### 【描述】

该 API 使用 ive 硬件资源，创建 GMM 背景建模任务，支持 1-5 个高斯模型，支持灰度图和 RGB\_PACKED 图输入，支持全局及像素级别的灵敏度系数以及前景模型时长更新系数。

### 【语法】

```
1 bm_status_t bmcv_ive_gmm2(
2     bm_handle_t handle,
3     bm_image * input,
4     bm_image * input_factor,
5     bm_image * output_fg,
6     bm_image * output_bg,
7     bm_image * output_match_model_info,
8     bm_device_mem_t output_model,
9     bmcv_ive_gmm2_ctrl gmm2_attr);
```

### 【参数】

表 5.73: bmcv\_ive\_gmm2 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*input	输入	输入 bm_image 对象指针，不能为空。
*input_factor	输入	输入 bm_image 对象指针，表示模型更新参数，当且仅仅当 gmm2_attr.en_sns_factor_mode、gmm2_attr.en_life_update_factor_mode 均使用全局模式时可以为空。
*output_fg	输出	输出 bm_image 对象指针，表示前景图像，不能为空，宽、高同 input。
*output_bg	输出	输出 bm_image 对象指针，表示背景图像，不能为空，宽、高同 input。
*output_match_model_info	输出	输出 bm_image 对象指针，对应模型匹配系数，不能为空。
output_model	输入、输出	bm_device_mem_t 对象结构体，对应 GMM 模型参数，不能为空。
gmm2_attr	输入	控制信息结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY RGB PACKED	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
input_factor	GRAY	DATA_TYPE_EXT_U16	同 input
output_fg	GRAY 二值图	DATA_TYPE_EXT_1N_BYTE	同 input
output_bg	同 input	DATA_TYPE_EXT_1N_BYTE	同 input
match_model_info	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

## 【数据类型说明】

【说明】定义灵敏度系数模式。

```

1  typedef enum gmm2_sns_factor_mode_e{
2      SNS_FACTOR_MODE_GLB = 0x0,
3      SNS_FACTOR_MODE_PIX = 0x1,
4  } gmm2_sns_factor_mode;

```

成员名称	描述
SNS_FACTOR_MODE_GLB	全局灵敏度系数模式，每个像素在模型匹配过程中，方差灵敏度使用 gmm2_attr.u8_glb_sns_factor。
SNS_FACTOR_MODE_PIX	像素级灵敏度系数模式，每个像素在模型匹配过程中，方差灵敏度使用 input_factor 的灵敏度系数。

【说明】定义模型时长参数更新模式。

```

1 typedef enum gmm2_life_update_factor_mode_e{
2     LIFE_UPDATE_FACTOR_MODE_GLB = 0x0,
3     LIFE_UPDATE_FACTOR_MODE_PIX = 0x1,
4 } gmm2_life_update_factor_mode;

```

成员名称	描述
LIFE_UPDATE_FACTOR_MODE_GLB	模型时长参数全局更新模式，每个像素模型时长参数在更新时使用 gmm2_attr.u16_glb_life_update_factor。
LIFE_UPDATE_FACTOR_MODE_PIX	模型时长参数像素级更新模式，每个像素模型时长在更新时使用 input_factor 的模型更新参数。

【说明】定义 GMM2 背景建模的控制参数。

```

1 typedef struct bmcv_ive_gmm2_ctrl_s{
2     gmm2_sns_factor_mode en_sns_factor_mode;
3     gmm2_life_update_factor_mode en_life_update_factor_mode;
4     unsigned short u16_glb_life_update_factor;
5     unsigned short u16_life_thr;
6     unsigned short u16_freq_init_val;
7     unsigned short u16_freq_redu_factor;
8     unsigned short u16_freq_add_factor;
9     unsigned short u16_freq_thr;
10    unsigned short u16_var_rate;
11    unsigned short u9q7_max_var;
12    unsigned short u9q7_min_var;
13    unsigned char u8_glb_sns_factor;
14    unsigned char u8_model_num;
15 } bmcv_ive_gmm2_ctrl;

```

成员名称	描述
en_sns_factor_mode	灵敏度模式, 默认全局模式。 全局模式使用 u8_glb_sns_factor 作为灵敏度系数; 像素模式使用 input_factor 的低 8 bit 值作为灵敏度系数。
en_life_update_factor_mode	模型时长更新模式, 默认全局模式。 全局模式使用 u16_glb_life_update_factor 作为前进模型更新参数; 像素模式使用 input_factor 的高 8bit 值作为前进模型时长更新参数。
u16_glb_life_update_factor	全局模型更新参数。 取值范围: [0, 65535], 默认: 4。
u16_life_thr	背景模型生成时间, 表示一个模型从前景模型转成背景模型需要的时间。 取值范围: [0, 65535], 默认: 5000。
u16_freq_init_val	初始频率。 取值范围: [0, 65535], 默认: 20000。
u16_freq_redu_factor	频率衰减系数。 取值范围: [0, 65535], 默认: 0xFF00。
u16_freq_add_factor	模型匹配频率增加系数。 取值范围: [0, 65535], 默认: 0xEF。
u16_freq_thr	模型失效频率阈值。 取值范围: [0, 65535], 默认: 12000。
u16_var_rate	方差更新率。 取值范围: [0, 65535], 默认: 1。
u9q7_max_var	方差最大值。 取值范围: [0, 65535], 默认: (16x16) ≪ 7。
u9q7_min_var	方差最小值。 取值范围: [0, u9q7MaxVar], 默认: (8x8) ≪ 7。
u8_glb_sns_factor	方差最小值。 取值范围: [0, 255], 默认: 8。
u8_model_num	模型数量。 取值范围 [1, 5], 默认: 3。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

- 输入输出图像的 width 都需要 16 对齐。

2. GMM2 在参考了 OPENCV 的 MOG 和 MOG2 的基础上，增加了像素级别的参数控制。
3. 源图像 pstSrc 类型只能为 U8C1 或 U8C3\_PACKAGE，分别用于灰度图和 RGB 图的 GMM 背景建模。
4. 模型更新参数 pstFactor 为 U16C1 图像：每个元素用 16 bit 表示，低 8 bit 为灵敏度系数，用于控制模型匹配时方差倍数；高 8 bit 为前景模型时长更新参数，用于控制背景模型形成时间。
5. 模型匹配系数指针 pstMatchModelInfo 为 U8C1 图像：每个元素用 8bit 表示，低 1 bit 为高斯模型匹配标志，0 表示匹配失败，1 表示匹配成功；高 7 bit 为频率最大模型序号。
6. GMM2 的频率参数（pstGmm2Ctrl 中的 u16\_freq\_init\_val、u16\_freq\_redu\_factor、u16\_freq\_add\_factor、u16\_freq\_thr）用于控制模型排序和模型有效时间。
  - u16\_freq\_init\_val 越大，模型有效时间越大；
  - u16\_freq\_redu\_factor 越大，模型有效时间越长，模型频率通过乘以频率衰减系数
  - u16\_freq\_redu\_factor/65536，达到频率衰减的目的；
  - u16\_freq\_add\_factor 越大，模型有效时间越长；
  - u16\_freq\_thr 越大，模型有效时间越短。
7. GMM2 的模型时长参数（pstGmm2Ctrl 中的 u16\_life\_thr）用于控制前景模型成为背景的时间。
  - u16\_life\_thr 越大，前景持续时间越长；
  - 单高斯模型下，模型时长参数不生效。
8. 灰度图像 GMM2 采用 n 个（ $1 \leq n \leq 5$ ）高斯模型。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    bool pixel_ctrl = false;
    gmm2_life_update_factor_mode
        life_update_enMode = LIFE_UPDATE_FACTOR_MODE_GLB;
    int height = 1080, width = 1920;
    bm_image_format_ext_src_fmt = FORMAT_GRAY;
    char *input_name = "path/to/input";
```

(续下页)

(接上页)

```

char *dstFg_name = "path/to/dst_Fg", *dstBg_name = "path/to/dst_Bg";

bm_handle_t handle = NULL;
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
bm_image src, src_factor;
bm_image dst_fg, dst_bg, dst_model_match_model_info;
bm_device_mem_t dst_model;
int stride[4], factorStride[4];
unsigned int u32FrameNumMax = 32;
unsigned int u32FrmCnt = 0;
unsigned int u32FrmNum = 0;

bmcv_ive_gmm2_ctrl gmm2Attr;
gmm2Attr.u16_var_rate = 1;
gmm2Attr.u8_model_num = 3;
gmm2Attr.u9q7_max_var = (16 * 16) << 7;
gmm2Attr.u9q7_min_var = (8 * 8) << 7;
gmm2Attr.u8_glb_sns_factor = 8;
gmm2Attr.en_sns_factor_mode = SNS_FACTOR_MODE_GLB;
gmm2Attr.u16_freq_thr = 12000;
gmm2Attr.u16_freq_init_val = 20000;
gmm2Attr.u16_freq_add_factor = 0xEF;
gmm2Attr.u16_freq_redu_factor = 0xFF00;
gmm2Attr.u16_life_thr = 5000;
gmm2Attr.en_life_update_factor_mode = life_update_enMode;

unsigned char* inputData = malloc(width * height * u32FrameNumMax * sizeof(unsigned char));
FILE *input_fp = fopen(input_name, "rb");
fread((void *)inputData, sizeof(unsigned char), width * height * u32FrameNumMax,
      input_fp);
fclose(input_fp);

unsigned char* srcData = malloc(width * height * sizeof(unsigned char));
unsigned short* srcFactorData = malloc(width * height * sizeof(unsigned short));
memset(srcData, 0, width * height * sizeof(unsigned char));
memset(srcFactorData, 0, width * height * sizeof(unsigned short));

int model_len = width * height * gmm2Attr.u8_model_num * 8;
unsigned char* model_data = malloc(model_len * sizeof(unsigned char));
memset(model_data, 0, model_len * sizeof(unsigned char));

unsigned char* ive_fg_res = malloc(width * height * sizeof(unsigned char));
unsigned char* ive_bg_res = malloc(width * height * sizeof(unsigned char));
unsigned char* ive_pc_match_res = malloc(width * height * sizeof(unsigned char));

```

(续下页)

(接上页)

```

memset(ive_fg_res, 0, width * height * sizeof(unsigned char));
memset(ive_bg_res, 0, width * height * sizeof(unsigned char));
memset(ive_pc_match_res, 0, width * height * sizeof(unsigned char));

// calc ive image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src, stride);
ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);

factorStride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
    ↵EXT_U16, &src_factor, factorStride);
ret = bm_image_alloc_dev_mem(src_factor, BMCV_HEAP1_ID);
ret = bm_image_copy_host_to_device(src_factor, (void **)&srcFactorData);

bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
    ↵EXT_1N_BYTE, &dst_fg, stride);
ret = bm_image_alloc_dev_mem(dst_fg, BMCV_HEAP1_ID);

bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
    ↵EXT_1N_BYTE, &dst_bg, stride);
ret = bm_image_alloc_dev_mem(dst_bg, BMCV_HEAP1_ID);

bm_image_create(handle, height, width, FORMAT_GRAY, DATA_TYPE_
    ↵EXT_1N_BYTE, &dst_model_match_model_info, stride);
ret = bm_image_alloc_dev_mem(dst_model_match_model_info, BMCV_
    ↵HEAP1_ID);

ret = bm_malloc_device_byte(handle, &dst_model, model_len);

ret = bm_memcpy_s2d(handle, dst_model, model_data);

for(u32FrmCnt = 0; u32FrmCnt < u32FrameNumMax; u32FrmCnt++){
    if(width > 480){
        for(int i = 0; i < 288; i++){
            memcpy(srcData + (i * width),
                inputData + (u32FrmCnt * 352 * 288 + i * 352), 352);
            memcpy(srcData + (i * width + 352),
                inputData + (u32FrmCnt * 352 * 288 + i * 352), 352);

        }
    } else {
        for(int i = 0; i < 288; i++){
            memcpy(srcData + i * stride[0],
                inputData + u32FrmCnt * width * 288 + i * width, width);
            int s = stride[0] - width;
            memset(srcData + i * stride[0] + width, 0, s);
        }
    }
}

```

(续下页)

(接上页)

```
        }

    ret = bm_image_copy_host_to_device(src, (void**)&srcData);

    u32FrmNum = u32FrmCnt + 1;
    if(gmm2Attr.u8_model_num == 1)
        gmm2Attr.u16_freq_redu_factor = (u32FrmNum >= 500) ? 0xFFA0 : 0xFC00;
    else
        gmm2Attr.u16_glb_life_update_factor =
            (u32FrmNum >= 500) ? 4 : 0xFFFF / u32FrmNum;

    if(pixel_ctrl && u32FrmNum > 16)
        gmm2Attr.en_life_update_factor_mode = LIFE_UPDATE_FACTOR_MODE_PIX;

    ret = bmcv_ive_gmm2(handle, &src, &src_factor, &dst_fg, &dst_bg, &dst_
model_match_model_info, dst_model, gmm2Attr);
}

ret = bm_image_copy_device_to_host(dst_fg, (void**)&ive_fg_res);
ret = bm_image_copy_device_to_host(dst_bg, (void**)&ive_bg_res);
ret = bm_image_copy_device_to_host(dst_model_match_model_info, (void*)&ive_pc_match_res);

FILE *fg_fp = fopen(dstFg_name, "wb");
fwrite((void *)ive_fg_res, 1, width * height, fg_fp);
fclose(fg_fp);

FILE *bg_fp = fopen(dstBg_name, "wb");
fwrite((void *)ive_bg_res, 1, width * height, bg_fp);
fclose(bg_fp);

free(inputData);
free(srcData);
free(srcFactorData);
free(model_data);
free(ive_fg_res);
free(ive_bg_res);
free(ive_pc_match_res);
bm_image_destroy(&src);
bm_image_destroy(&src_factor);
bm_image_destroy(&dst_fg);
bm_image_destroy(&dst_model_match_model_info);
bm_free_device(handle, dst_model);

bm_dev_free(handle);
return 0;
}
```

### 5.101 bmcv\_ive\_resize

#### 【描述】

该 API 使用 ive 硬件资源，创建图像缩放任务，支持双线性插值、区域插值缩放，支持多张 GRAY 与 RGB PLANAR 图像同时输入做一种类型的缩放。

#### 【语法】

```

1 bm_status_t bmcv_ive_resize(
2     bm_handle_t          handle,
3     bm_image              input,
4     bm_image              output,
5     bmcv_resize_algorithm  resize_mode);

```

#### 【参数】

表 5.74: bmcv\_ive\_resize 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空，每张图像类型同 input。
resize_mode	输入	输入 bmcv_resize_algorithm 对象，枚举控制信息。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
	RGB_PLANAR		
output	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
	RGB_PLANAR		

#### 【数据类型说明】

【说明】 定义 Resize 模式。

```

1 typedef enum bmcv_resize_algorithm_ {
2     BMCV_INTER_NEAREST = 0,
3     BMCV_INTER_LINEAR = 1,
4     BMCV_INTER_BICUBIC = 2,
5     BMCV_INTER_AREA = 3,
6 } bmcv_resize_algorithm;

```

成员名称	描述
BMCV_INTER_NEAREST	最近邻插值模式。
IVE_RESIZE_AREA	双线性插值缩放模式。
IVE_RESIZE_LINEAR	双三次插值模式。
IVE_RESIZE_AREA	区域插值缩放模式。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 支持 GRAY、RGB\_PLANAR 混合图像数组输入, 但所有图像的缩放模式相同, 同时模式只支持双线性插值和区域插值缩放模式。
2. 最大支持 16 倍缩放。
3. 输入输出图像的 width 都需要 16 对齐。

### 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    bmcv_resize_algorithm resize_mode = BMCV_INTER_AREA;
    bm_image_format_ext format = FORMAT_GRAY;
    int src_width = 1920, src_height = 1080;
    int dst_width = 400;
    int dst_height = 300;

```

(续下页)

(接上页)

```

char *src_name = "path/to/src";
char *dst_name = "path/to/dst";
bm_handle_t handle = NULL;
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}

bm_image src, dst;
int src_stride[4], dst_stride[4];

int data_size= 1;
int byte_size;
int image_byte_size[4] = {0};
// calculate image stride && create bm image struct
src_stride[0] = align_up(src_width, 16) * data_size;
bm_image_create(handle, src_height, src_width, format, DATA_TYPE_EXT_
↪1N_BYTE, &src, src_stride);

dst_stride[0] = align_up(dst_width, 16) * data_size;
bm_image_create(handle, dst_height, dst_width, format, DATA_TYPE_EXT_
↪1N_BYTE, &dst, dst_stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

bm_image_get_byte_size(src, image_byte_size);
byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + [F]
↪image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%ld\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
                   ((void *)((unsigned char*)input_data + image_byte_size[0])),
                   ((void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↪image_byte_size[1])),
                   ((void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↪image_byte_size[1] + image_byte_size[2]))};
bm_image_copy_host_to_device(src, in_ptr);

ret = bmcv_ive_resize(handle, src, dst, resize_mode);

bm_image_get_byte_size(dst, image_byte_size);
byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] + [F]
↪image_byte_size[3];
unsigned char* output_ptr = (unsigned char *)malloc(byte_size);
memset(output_ptr, 0, sizeof(byte_size));

```

(续下页)

(接上页)

```

void* out_ptr[4] = { (void*)output_ptr,
                     ((void*)((char*)output_ptr + image_byte_size[0])),
                     ((void*)((char*)output_ptr + image_byte_size[0] + image_byte_
                     size[1])),
                     ((void*)((char*)output_ptr + image_byte_size[0] + image_byte_
                     size[1] + image_byte_size[2]))};

ret = bm_image_copy_device_to_host(dst, (void**)out_ptr);

FILE *ive_fp = fopen(dst_name, "wb");
fwrite((void*)output_ptr, 1, byte_size, ive_fp);
fclose(ive_fp);

free(input_data);
free(output_ptr);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}

```

## 5.102 bmcv\_ive\_thresh

### 【描述】

该 API 使用 ive 硬件资源, 支持 U8 数据到 U8 数据、S16 数据到 8bit 数据或 U16 数据到 U8 数据的阈值化任务。

### 【语法】

```

1 bm_status_t bmcv_ive_thresh(
2     bm_handle_t      handle,
3     bm_image        input,
4     bm_image        output,
5     bmcv_ive_thresh_mode thresh_mode,
6     bmcv_ive_thresh_attr attr);

```

### 【参数】

表 5.75: bmcv\_ive\_thresh 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	输入 bm_image 结构体，不能为空。
output	输出	输出 bm_image 结构体，由模板直接滤波得到的梯度分量图像 H 指针，根据 sobel_attr.enOutCtrl，若需要输出则不能为空，宽、高同 input。
thresh_mode	输入	控制阈值化计算模式。
attr	输入	定义图像二值化控制信息。

## 【数据类型说明】

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
		DATA_TYPE_EXT_S16	
		DATA_TYPE_EXT_U16	
output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input
		DATA_TYPE_EXT_1N_BYTE_SIGNED	

【说明】 定义阈值化计算方式。

```

1 typedef enum bmcv_ive_thresh_mode_e{
2     // u8
3     IVE_THRESH_BINARY = 0x0,
4     IVE_THRESH_TRUNC = 0x1,
5     IVE_THRESH_TO_MINAL = 0x2,
6     IVE_THRESH_MIN_MID_MAX = 0x3,
7     IVE_THRESH_ORI_MID_MAX = 0x4,
8     IVE_THRESH_MIN_MID_ORI = 0x5,
9     IVE_THRESH_MIN_ORI_MAX = 0x6,
10    IVE_THRESH_ORI_MID_ORI = 0x7,
11
12    // s16
13    IVE_THRESH_S16_TO_S8_MIN_MID_MAX = 0x8,
14    IVE_THRESH_S16_TO_S8_MIN_ORI_MAX = 0x9,
15    IVE_THRESH_S16_TO_U8_MIN_MID_MAX = 0x10,
16    IVE_THRESH_S16_TO_U8_MIN_ORI_MAX = 0x11,
17
18    // u16
19    IVE_THRESH_U16_TO_U8_MIN_MID_MAX = 0x12,
20    IVE_THRESH_U16_TO_U8_MIN_ORI_MAX = 0x13,
21
22 } bmcv_ive_thresh_mode;

```

1. MOD\_U8 支持以下计算模式:

- IVE\_THRESH\_BINARY:
  - src\_val ≤ low\_thr, dst\_val = min\_val;
  - src\_val > low\_thr, dst\_val = max\_val.
- IVE\_THRESH\_TRUNC:
  - src\_val ≤ low\_thr, dst\_val = src\_val;
  - src\_val > low\_thr, dst\_val = max\_val.
- IVE\_THRESH\_TO\_MINAL:
  - src\_val ≤ low\_thr, dst\_val = min\_val;
  - src\_val > low\_thr, dst\_val = src\_val.
- IVE\_THRESH\_MIN\_MID\_MAX:
  - src\_val ≤ low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val ≤ high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_ORI\_MID\_MAX:
  - src\_val ≤ low\_thr, dst\_val = src\_val;
  - low\_thr < src\_val ≤ high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_MIN\_MID\_ORI:
  - src\_val ≤ low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val ≤ high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = src\_val.
- IVE\_THRESH\_MIN\_ORI\_MAX:
  - src\_val ≤ low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val ≤ high\_thr, dst\_val = src\_val;
  - src\_val > high\_thr, dst\_val = max\_val
- IVE\_THRESH\_ORI\_MID\_ORI:
  - src\_val ≤ low\_thr, dst\_val = src\_val;
  - low\_thr < src\_val ≤ high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = src\_val.

## 2. MOD\_S16 支持以下计算模式:

- IVE\_THRESH\_S16\_TO\_S8\_MIN\_MID\_MAX:

- src\_val < low\_thr, dst\_val = min\_val;
- low\_thr < src\_val < high\_thr, dst\_val = mid\_val;
- src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_S16\_TO\_S8\_MIN\_ORI\_MAX:
  - src\_val < low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val < high\_thr, dst\_val = src\_val;
  - src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_S16\_TO\_U8\_MIN\_MID\_MAX:
  - src\_val < low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val < high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_S16\_TO\_U8\_MIN\_ORI\_MAX:
  - src\_val < low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val < high\_thr, dst\_val = src\_val;
  - src\_val > high\_thr, dst\_val = max\_val.

### 3. MOD\_U16 支持以下计算模式:

- IVE\_THRESH\_U16\_TO\_U8\_MIN\_MID\_MAX:
  - src\_val < low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val < high\_thr, dst\_val = mid\_val;
  - src\_val > high\_thr, dst\_val = max\_val.
- IVE\_THRESH\_U16\_TO\_U8\_MIN\_ORI\_MAX:
  - src\_val < low\_thr, dst\_val = min\_val;
  - low\_thr < src\_val < high\_thr, dst\_val = src\_val;
  - src\_val > high\_thr, dst\_val = max\_val.

**【说明】** 定义图像二值化控制信息。

```

1 typedef struct bmcv_ive_thresh_attr_s {
2     int low_thr;
3     int high_thr;
4     int min_val;
5     int mid_val;
6     int max_val;
7 } bmcv_ive_thresh_attr;

```

成员名称	描述
low_thr	低阈值。
high_thr	高阈值。
min_val	最小值。
mid_val	中间值。
max_val	最大值。

**【返回值】**

该函数成功调用时, 返回 BM\_SUCCESS。

**【注意】**

1. 输入输出图像的 width 都需要 16 对齐。
2. 可配置 3 种阈值化模式, 分别是 MOD\_U8、MOD\_U16、MOD\_S16, 每种模式支持的阈值计算模式详见 bmcv\_ive\_thresh\_mode。
3. 3 种阈值化模式的计算公式如下:
  - MOD\_U8
    - (1) IVE\_THRESH\_BINARY:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \max\_val & (I(x, y) > \text{low\_thr}) \end{cases}$$

*mid\_val*、*high\_thr* 无需赋值。

- (2) IVE\_THRESH\_TRUNC:

$$I_{\text{out}}(x, y) = \begin{cases} I(x, y) & (I(x, y) \leq \text{low\_thr}) \\ \max\_val & (I(x, y) > \text{low\_thr}) \end{cases}$$

*min\_val*、*mid\_val*、*high\_thr* 无需赋值。

- (3) IVE\_THRESH\_TO\_MINAL:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ I(x, y) & (I(x, y) > \text{low\_thr}) \end{cases}$$

*min\_val*、*max\_val*、*high\_thr* 无需赋值。

- (4) IVE\_THRESH\_MIN\_MID\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

(5) IVE\_THRESH\_ORI\_MID\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} I(x, y) & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

$\min\_val$  无需赋值。

(6) IVE\_THRESH\_MIN\_MID\_ORI:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ I(x, y) & (I(x, y) > \text{high\_thr}) \end{cases}$$

$\max\_val$  无需赋值。

(7) IVE\_THRESH\_MIN\_ORI\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ I(x, y) & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

$\text{mid\_val}$  无需赋值。

(8) IVE\_THRESH\_ORI\_MID\_ORI:

$$I_{\text{out}}(x, y) = \begin{cases} I(x, y) & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ I(x, y) & (I(x, y) > \text{high\_thr}) \end{cases}$$

$\min\_val$ 、 $\max\_val$  无需赋值。

MOD\_S16

(1) IVE\_THRESH\_S16\_TO\_S8\_MIN\_MID\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

要求:

$$-32768 \leq \text{low\_thr} \leq \text{high\_thr} \leq 32767;$$

$$-128 \leq \min\_val, \text{mid\_val}, \max\_val \leq 127.$$

(2) IVE\_THRESH\_S16\_TO\_S8\_MIN\_ORI\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ I(x, y) & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

**要求:**

$-129 \leq \text{low\_thr} \leq \text{high\_thr} \leq 127$ ;

$-128 \leq \min\_val, \max\_val \leq 127$ 。

(3) IVE\_THRESHOLD\_S16\_TO\_U8\_MIN\_MID\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

**要求:**

$-32768 \leq \text{low\_thr} \leq \text{high\_thr} \leq 32767$ ;

$0 \leq \min\_val, \text{mid\_val}, \max\_val \leq 255$ 。

(4) IVE\_THRESHOLD\_S16\_TO\_U8\_MIN\_ORI\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ I(x, y) & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

**要求:**

$-1 \leq \text{low\_thr} \leq \text{high\_thr} \leq 255$ ;

$0 \leq \min\_val, \max\_val \leq 255$ 。

· MOD\_U16

(1) IVE\_THRESHOLD\_U16\_TO\_U8\_MIN\_MID\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ \text{mid\_val} & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

要求:  $0 \leq \text{low\_thr} \leq \text{high\_thr} \leq 65535$ ;

(2) IVE\_THRESHOLD\_U16\_TO\_U8\_MIN\_ORI\_MAX:

$$I_{\text{out}}(x, y) = \begin{cases} \min\_val & (I(x, y) \leq \text{low\_thr}) \\ I(x, y) & (\text{low\_val} < I(x, y) \leq \text{high\_val}) \\ \max\_val & (I(x, y) > \text{high\_thr}) \end{cases}$$

要求:  $0 \leq \text{low\_thr} \leq \text{high\_thr} \leq 255$ ;

其中,  $I(x, y)$  对应 input,  $I_{\text{out}}(x, y)$  对应 output,  $\text{low\_thr}$ 、 $\text{high\_thr}$ 、 $\min\_val$ 、 $\text{mid\_val}$  和  $\max\_val$  分别对应 attr 的  $\text{low\_thr}$ 、 $\text{high\_thr}$ 、 $\min\_val$ 、 $\text{mid\_val}$ 、 $\max\_val$ ;

- bmcv\_ive\_thresh\_attr 中的 min\_val、mid\_val、max\_val 并不需要满足变量命名含义中的大小关系。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    bmcv_ive_thresh_mode thresh_mode = IVE_THRESH_MIN_MID_MAX; /*F
    ↵IVE_THRESH_MODE_MIN_MID_MAX */
    int low_thr = 236, high_thr = 249, min_val = 166, mid_val = 219, max_val = F
    ↵60;
    bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
    ↵GRAY;
    char *src_name = "path/to/src", *dst_name = "path/to/dst";

    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bmcv_ive_thresh_attr thresh_attr;
    bm_image src, dst;
    int src_stride[4];
    int dst_stride[4];

    // calc ive image stride
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    dst_stride[0] = align_up(width, 16) * data_size;
    // create bm image struct
    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &src, src_stride);
    bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &dst, dst_stride);

    // alloc bm image memory
    ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    // read image data from input files
    int image_byte_size[4] = {0};
```

(续下页)

(接上页)

```
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = { (void *)input_data,
                    ((void *)((unsigned char *)input_data + image_byte_size[0])),
                    ((void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1])),
                    ((void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2]))};
bm_image_copy_host_to_device(src, in_ptr);

memset(&thresh_attr, 0, sizeof(bmcv_ive_thresh_attr));
thresh_attr.low_thr = low_thr;
thresh_attr.high_thr = high_thr;
thresh_attr.min_val = min_val;
thresh_attr.mid_val = mid_val;
thresh_attr.max_val = max_val;

ret = bmcv_ive_thresh(handle, src, dst, thresh_mode, thresh_attr);

unsigned char *ive_res = (unsigned char *) malloc(width * height * [F]
sizeof(unsigned char));
memset(ive_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_res);

FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_res);

bm_image_destroy(&src);
bm_image_destroy(&dst);

bm_dev_free(handle);

return ret;
}
```

### 5.103 bmcv\_ive\_add

#### 【描述】

该 API 使用 ive 硬件资源，创建两张图像的加权相加操作。

#### 【语法】

```

1 bm_status_t bmcv_ive_add(
2     bm_handle_t      handle,
3     bm_image         input1,
4     bm_image         input2,
5     bm_image         output,
6     bmcv_ive_add_attr attr);

```

#### 【参数】

表 5.76: bmcv\_ive\_add 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
attr	输入	相加操作的需要的加权系数对应的结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input1	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	64x64~1920x1080
input2	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	同 input1
output	GRAY	DATA_TYPE_EXT_1N_BYTF	同 Input1

【说明】 定义两图像的加权加控制参数。

```

1 typedef struct bmcv_ive_add_attr_s {
2     unsigned short param_x;

```

(续下页)

(接上页)

```

3     unsigned short param_y;
4 } bmcv_ive_add_attr;
```

成员名称	描述
param_x	加权加 “xA + yB” 中的权重 x。
param_y	加权加 “xA + yB” 中的权重 y。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- **MOD\_ADD:**

$$I_{\text{out}}(i, j) = x * I_1(i, j) + y * I_2(i, j)$$

其中,  $I_1(i, j)$  对应 input1,  $I_2(i, j)$  对应 input2,  $x, y$  分别对应 bmcv\_ive\_add\_attr 中的 param\_x 与 param\_y。

若定点化前,  $x$  和  $y$  满足  $x + y > 1$ , 则当前计算结果超过 8bit 取低 8bit 作为最终结果。

## 示例代码

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    unsigned short x = 19584, y = 45952; /* x + y = 65536 */
    bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
GRAY;
    char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
```

(续下页)

(接上页)

```

int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
bm_image src1, src2, dst;
int src_stride[4];
int dst_stride[4];

bmcv_ive_add_attr add_attr;
memset(&add_attr, 0, sizeof(bmcv_ive_add_attr));

add_attr.param_x = x;
add_attr.param_y = y;
// calc ive image stride && create bm image struct
int data_size = 1;
src_stride[0] = align_up(width, 16) * data_size;
dst_stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
↪BYTE, &src1, src_stride);
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
↪BYTE, &src2, src_stride);
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
↪BYTE, &dst, dst_stride);

ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
bm_image src_images[] = {src1, src2};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
↪+ image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       ((void *)((unsigned char*)input_data + image_byte_size[0])),
                       ((void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↪image_byte_size[1])),
                       ((void *)((unsigned char*)input_data + image_byte_size[0] + [F])

```

(续下页)

(接上页)

```

    ↵image_byte_size[1] + image_byte_size[2])};
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

ret = bmcv_ive_add(handle, src1, src2, dst, add_attr);

unsigned char *ive_add_res = (unsigned char*)malloc(width * height * F
↵sizeof(unsigned char));
memset(ive_add_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void**)&ive_add_res);
FILE *fp = fopen(dst_name, "wb");
fwrite((void*)ive_add_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_add_res);

bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);

return 0;
}

```

### 5.104 bmcv\_ive\_sub

#### 【描述】

该 API 使用 ive 硬件资源，创建两张图像的相减操作。

#### 【语法】

```

1 bm_status_t bmcv_ive_sub(
2     bm_handle_t      handle,
3     bm_image        input1,
4     bm_image        input2,
5     bm_image        output,
6     bmcv_ive_sub_attr attr);

```

#### 【参数】

表 5.77: bmcv\_ive\_sub 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
attr	输入	相减操作对应模式的结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input1	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	64x64~1920x1080
input2	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	同 input1
output	GRAY	DATA_TYPE_EXT_1N_BYTF	同 Input1

【注意】 output 数据类型格式也有可能是 DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED。

### 【数据类型说明】

【说明】 定义两张图像相减输出格式。

```

1 typedef enum bmcv_ive_sub_mode_e{
2     IVE_SUB_ABS = 0x0,
3     IVE_SUB_SHIFT = 0x1,
4     IVE_SUB_BUTT
5 } bmcv_ive_sub_mode;

```

成员名称	描述
IVE_SUB_ABS	取差的绝对值。
IVE_SUB_SHIFT	将结果右移一位输出，保留符号位。

【说明】 定义两图像相减控制参数。

```

1 typedef struct bmcv_ive_sub_attr_s {
2     bmcv_ive_sub_mode en_mode;
3 } bmcv_ive_sub_attr;

```

成员名称	描述
en_mode	MOD_SUB 下两图像相减模式。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- MOD\_SUB:
  - IVE\_SUB\_ABS:

$$I_{\text{out}}(x, y) = \text{abs}(I_{\text{src1}}(x, y) - I_{\text{src2}}(x, y))$$

输出格式是 DATA\_TYPE\_EXT\_1N\_BYTE。

- IVE\_SUB\_SHIFT:

$$I_{\text{out}}(x, y) = (I_{\text{src1}}(x, y) - I_{\text{src2}}(x, y)) \gg 1$$

输出格式是 DATA\_TYPE\_EXT\_1N\_BYTE\_SIGNED。

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    bmcv_ive_sub_mode sub_mode = IVE_SUB_ABS; /* IVE_SUB_MODE_
→ABS */
    bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
→GRAY;
    char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
}
```

(续下页)

(接上页)

```

bmcv_ive_sub_attr sub_attr;
bm_image src1, src2, dst;
int src_stride[4];
int dst_stride[4];

// set ive sub params
memset(&sub_attr, 0, sizeof(bmcv_ive_sub_attr));
sub_attr.en_mode = sub_mode;

// calc ive image stride
int data_size = 1;
src_stride[0] = align_up(width, 16) * data_size;
dst_stride[0] = align_up(width, 16) * data_size;
// create bm image struct
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
←BYTE, &src1, src_stride);
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
←BYTE, &src2, src_stride);
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
←BYTE, &dst, dst_stride);

// alloc bm image memory
ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

// read image data from input files
int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
bm_image src_images[] = {src1, src2};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] ←F
    ←+ image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       ((void *)((unsigned char *)input_data + image_byte_size[0])),
                       ((void *)((unsigned char *)input_data + image_byte_size[0]) + ←F
    ←image_byte_size[1]),
                       ((void *)((unsigned char *)input_data + image_byte_size[0]) + ←F
    ←image_byte_size[1] + image_byte_size[2])};
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

```

(续下页)

(接上页)

```
ret = bmcv_ive_sub(handle, src1, src2, dst, sub_attr);

unsigned char* ive_sub_res = (unsigned char*)malloc(width * height * sizeof(unsigned char));
memset(ive_sub_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_sub_res);

FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_sub_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_sub_res);

bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}
```

### 5.105 bmcv\_ive\_and

#### 【描述】

该 API 使用 ive 硬件资源，创建两张图像的加权相与操作。

#### 【语法】

```
1 bm_status_t bmcv_ive_and(
2     bm_handle_t      handle,
3     bm_image        input1,
4     bm_image        input2,
5     bm_image        output);
```

#### 【参数】

表 5.78: bmcv\_ive\_and 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input1	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	64x64~1920x1080
input2	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	同 input1
output	GRAY	DATA_TYPE_EXT_1N_BYTF	同 Input1

**【返回值】**

该函数成功调用时，返回 BM\_SUCCESS。

**【注意】**

- 输入输出图像的 width 都需要 16 对齐。
- **MOD\_AND:**

$$I_{\text{out}}(x, y) = I_{\text{src1}}(x, y) \& I_{\text{src2}}(x, y)$$

其中， $I_{\text{src1}}(x, y)$  对应 input1， $I_{\text{src2}}(x, y)$  对应 input2， $I_{\text{out}}(x, y)$  对应 output。

**示例代码**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
```

(续下页)

(接上页)

```

bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
GRAY;
char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
char *dst_name = "path/to/dst";
bm_handle_t handle = NULL;
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
bm_image src1, src2, dst;
int src_stride[4];
int dst_stride[4];

// calculate image stride
int data_size = 1;
src_stride[0] = align_up(width, 16) * data_size;
dst_stride[0] = align_up(width, 16) * data_size;
// create bm image struct
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src1, src_stride);
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src2, src_stride);
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
BYTE, &dst, dst_stride);

// alloc bm image memory
ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

// read image data from input files
int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
bm_image src_images[] = {src1, src2};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
+ image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
        (void *)((unsigned char *)input_data + image_byte_size[0]),
        (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
        (void *)((unsigned char *)input_data + image_byte_size[1])},

```

(续下页)

(接上页)

```

        (void *)((unsigned char*)input_data + image_byte_size[0] + F
    ↵image_byte_size[1] + image_byte_size[2]));
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

ret = bmcv_ive_and(handle, src1, src2, dst);

unsigned char *ive_and_res = (unsigned char*)malloc(width * height * F
    ↵sizeof(unsigned char));
memset(ive_and_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_and_res);
FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_and_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_and_res);
bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}

```

### 5.106 bmcv\_ive\_or

#### 【描述】

该 API 使用 ive 硬件资源，创建两张图像的相或操作。

#### 【语法】

```

1 bm_status_t bmcv_ive_or(
2     bm_handle_t      handle,
3     bm_image        input1,
4     bm_image        input2,
5     bm_image        output);

```

#### 【参数】

表 5.79: bmcv\_ive\_or 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input1	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	64x64~1920x1080
input2	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	同 input1
output	GRAY	DATA_TYPE_EXT_1N_BYTF	同 Input1

### 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- **MOD\_OR:**

$$I_{\text{out}}(x, y) = I_{\text{src1}}(x, y) | I_{\text{src2}}(x, y)$$

其中， $I_{\text{src1}}(x, y)$  对应 input1， $I_{\text{src2}}(x, y)$  对应 input2， $I_{\text{out}}(x, y)$  对应 output。

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
```

(续下页)

(接上页)

```

bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
GRAY;
char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
char *dst_name = "path/to/dst";
bm_handle_t handle = NULL;
int ret = (int)bm_dev_request(&handle, dev_id);
if (ret != 0) {
    printf("Create bm handle failed. ret = %d\n", ret);
    exit(-1);
}
bm_image src1, src2, dst;
int src_stride[4];
int dst_stride[4];

// calculate image stride
int data_size = 1;
src_stride[0] = align_up(width, 16) * data_size;
dst_stride[0] = align_up(width, 16) * data_size;

// create bm image struct
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src1, src_stride);
bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src2, src_stride);
bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
BYTE, &dst, dst_stride);

// alloc bm image memory
ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

// read image data from input files
int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
bm_image src_images[] = {src1, src2};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] [F]
    + image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,

```

(续下页)

(接上页)

```

        (void *)((unsigned char*)input_data + image_byte_size[0]),
        (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1]),
        (void *)((unsigned char*)input_data + image_byte_size[0] + [F]
↳image_byte_size[1] + image_byte_size[2]));
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

ret = bmcv_ive_or(handle, src1, src2, dst);

unsigned char *ive_res = (unsigned char*) malloc(width * height * [F]
↳sizeof(unsigned char));
memset(ive_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void**)&ive_res);

FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_res);

bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);
return 0;
}

```

### 5.107 bmcv\_ive\_xor

【描述】

该 API 使用 ive 硬件资源，创建两张图像的加权异或操作。

【语法】

```

1 bm_status_t bmcv_ive_xor(
2     bm_handle_t      handle,
3     bm_image        input1,
4     bm_image        input2,
5     bm_image        output);

```

## 【参数】

表 5.80: bmcv\_ive\_xor 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input1	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	64x64~1920x1080
input2	GRAY GRAY 二值图	DATA_TYPE_EXT_1N_BYTF	同 input1
output	GRAY	DATA_TYPE_EXT_1N_BYTF	同 Input1

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

## 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- MOD\_XOR:

$$I_{\text{out}}(x, y) = I_{\text{src1}}(x, y) \oplus I_{\text{src2}}(x, y)$$

其中， $I_{\text{src1}}(x, y)$  对应 input1， $I_{\text{src2}}(x, y)$  对应 input2， $I_{\text{out}}(x, y)$  对应 output。

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))
```

(续下页)

(接上页)

```

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext src_fmt = FORMAT_GRAY, dst_fmt = FORMAT_
GRAY;
    char *src1_name = "path/to/src1", *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src1, src2, dst;
    int src_stride[4];
    int dst_stride[4];

    // calculate image stride
    int data_size = 1;
    src_stride[0] = align_up(width, 16) * data_size;
    dst_stride[0] = align_up(width, 16) * data_size;
    // create bm image struct
    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src1, src_stride);
    bm_image_create(handle, height, width, src_fmt, DATA_TYPE_EXT_1N_
BYTE, &src2, src_stride);
    bm_image_create(handle, height, width, dst_fmt, DATA_TYPE_EXT_1N_
BYTE, &dst, dst_stride);

    // alloc bm image memory
    ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
    ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

    // read image data from input files
    int byte_size;
    unsigned char *input_data;
    int image_byte_size[4] = {0};
    char *filename[] = {src1_name, src2_name};
    bm_image src_images[] = {src1, src2};
    for (int i = 0; i < 2; i++) {
        bm_image_get_byte_size(src_images[i], image_byte_size);
        byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
        input_data = (unsigned char *)malloc(byte_size);
        FILE *fp_src = fopen(filename[i], "rb");
        if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
            printf("file size is less than required bytes%d\n", byte_size);
        };
        fclose(fp_src);
        void* in_ptr[4] = {(void *)input_data,

```

(续下页)

(接上页)

```
(void *)((unsigned char*)input_data + image_byte_size[0]),
(void *)((unsigned char*)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
(void *)((unsigned char*)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2]));
bm_image_copy_host_to_device(src_images[i], in_ptr);
}

ret = bmcv_ive_xor(handle, src1, src2, dst);

unsigned char *ive_res = (unsigned char*) malloc(width * height * [F]
sizeof(unsigned char));
memset(ive_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void**)&ive_res);
if(ret != BM_SUCCESS){
    printf("dst bm_image_copy_device_to_host failed, ret is %d \n", ret);
    exit(-1);
}

FILE *fp = fopen(dst_name, "wb");
fwrite((void *)ive_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_res);

bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);

return ret;
}
```

### 5.108 bmcv\_ive\_canny

#### 【描述】

该 API 使用 ive 硬件资源的 ccl 功能，计算 canny 边缘图。

#### 【语法】

```

1 bm_status_t bmcv_ive_canny(
2     bm_handle_t           handle,
3     bm_image              input,
4     bm_device_mem_t       output_edge,
5     bmcv_ive_canny_hys_edge_ctrl  canny_hys_edge_attr);

```

## 【参数】

表 5.81: bmcv\_ive\_canny 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 <code>bm_dev_request</code> 获取。
input	输入	输入 <code>bm_image</code> 对象结构体，不能为空。
output_edge	输出	输出 <code>bm_device_mem_t</code> 对象结构体，不能为空。
canny_hys_edge_attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080

## 【数据类型说明】

【说明】 定义 Canny 边缘前半部分计算任务的控制参数。

```

1 typedef struct bmcv_ive_canny_hys_edge_ctrl_s{
2     bm_device_mem_t st_mem;
3     unsigned short u16_low_thr;
4     unsigned short u16_high_thr;
5     signed char as8_mask[25];
6 } bmcv_ive_canny_hys_edge_ctrl;

```

成员名称	描述
stmem	辅助内存。
u16_low_thr	低阈值，取值范围：[0, 255]。
u16_high_thr	高阈值，取值范围：[u16LowThr, 255]。
as8_mask	用于计算梯度的参数模板。

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【注意】

1. 输入输出图像的 width 都需要 16 对齐。
2. canny\_hys\_edge\_attr.st\_mem 至少需要分配的内存大小:  

$$\text{canny\_hys\_edge\_attr.st\_mem.size} = \text{input\_stride} * 3 * \text{input.height}.$$

### 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int thrSize = 0; //0 -> 3x3; 1 -> 5x5
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src_name = "path/to/src";
    char *dstEdge_name = "path/to/dst";
    bm_handle_t handle = NULL;

    signed char arr5by5[25] = { -1, -2, 0, 2, 1, -4, -8, 0, 8, 4, -6, -12, 0,
        12, 6, -4, -8, 0, 8, 4, -1, -2, 0, 2, 1 };
    signed char arr3by3[25] = { 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, -2, 0,
        2, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0 };
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }
    bm_image src;
    bm_device_mem_t canny_stmem;
    int stride[4];

    int stmem_len = width * height * 4 * (sizeof(unsigned short) + sizeof(unsigned char));
    unsigned char *edge_res = malloc(width * height * sizeof(unsigned char));
    memset(edge_res, 0, width * height * sizeof(unsigned char));

    bmvc_ive_canny_hys_edge_ctrl cannyHysEdgeAttr;
    memset(&cannyHysEdgeAttr, 0, sizeof(bmvc_ive_canny_hys_edge_ctrl));
    cannyHysEdgeAttr.u16_low_thr = (thrSize == 0) ? 42 : 108;
    cannyHysEdgeAttr.u16_high_thr = 3 * cannyHysEdgeAttr.u16_low_thr;
    (thrSize == 0) ? memcpy(cannyHysEdgeAttr.as8_mask, arr3by3, 5 * 5 * F
    ↵char);
```

(续下页)

(接上页)

```

→sizeof(signed char)) :
    memcpy(cannyHysEdgeAttr.as8_mask, arr5by5, 5 * 5 * sizeof(signed F
→char));
}

// calculate image stride && create bm image struct
int data_size = 1;
stride[0] = align_up(width, 16) * data_size;
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
→src, stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_malloc_device_byte(handle, &canny_stmem, stmem_len);

// bm_ive_read_bin(src, src_name);
int image_byte_size[4] = {0};
bm_image_get_byte_size(src, image_byte_size);
int byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2] F
→+ image_byte_size[3];
unsigned char *input_data = (unsigned char *)malloc(byte_size);
FILE *fp_src = fopen(src_name, "rb");
if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
    printf("file size is less than required bytes%d\n", byte_size);
}
fclose(fp_src);
void* in_ptr[4] = {(void *)input_data,
    (void *)((unsigned char *)input_data + image_byte_size[0]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + F
→image_byte_size[1]),
    (void *)((unsigned char *)input_data + image_byte_size[0] + F
→image_byte_size[1] + image_byte_size[2])};
bm_image_copy_host_to_device(src, in_ptr);
cannyHysEdgeAttr.st_mem = canny_stmem;

ret = bmcv_ive_canny(handle, src, bm_mem_from_system((void *)edge_res), F
→cannyHysEdgeAttr);

FILE *edge_fp = fopen(dstEdge_name, "wb");
fwrite((void *)edge_res, 1, width * height, edge_fp);
fclose(edge_fp);

free(input_data);
free(edge_res);
bm_image_destroy(&src);
bm_free_device(handle, cannyHysEdgeAttr.st_mem);

bm_dev_free(handle);
return 0;
}

```

## 5.109 bmcv\_ive\_match\_bgmodel

### 【描述】

该 API 使用 ive 硬件资源，输入当前图像和模型，通过当前帧数据和背景之间差异来，获取前景数据。

### 【语法】

```

1 bm_status_t bmcv_ive_match_bgmodel(
2     bm_handle_t          handle,
3     bm_image              cur_img,
4     bm_image              bgmodel_img,
5     bm_image              fgflag_img,
6     bm_image              diff_fg_img,
7     bm_device_mem_t      stat_data_mem,
8     bmcv_ive_match_bgmodel_attr attr);

```

### 【参数】

表 5.82: bmcv\_ive\_match\_bgmodel 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
cur_img	输入	输入 bm_image 对象结构体，表示当前图像，不能为空。
bgmodel_img	输入/输出	背景模型 bm_image 结构体，不能为空，高同 cur_img，宽 = cur_img.width * sizeof(bm_ive_bg_model_pix)。
fgflag_img	输入/输出	前景状态图像 bm_image 结构体，不能为空，宽、高同 cur_img。
diff_fg_img	输出	帧间差分前景图像 bm_image 结构体，不能为空，宽、高同 cur_img。
stat_data_mem	输出	前景状态数据 bm_device_mem_t 结构体，不能为空，内存至少需配置 sizeof(bm_ive_bg_stat_data)。
attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
cur_img	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
bgmodel_img	GRAY	DATA_TYPE_EXT_1N_BYTE	-
fgflag_img	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
diff_fg_img	GRAY	DATA_TYPE_EXT_S16	64x64~1920x1080
stat_data_mem	GRAY	DATA_TYPE_EXT_1N_BYTE	-

### 【数据类型说明】

【说明】定义背景模型数据。

```

1 typedef struct bmcv_ive_bg_model_pix_s{
2     bmcv_ive_work_bg_pix st_work_bg_pixel;
3     bmcv_ive_candi_bg_pix st_candi_pixel;
4     bmcv_ive_bg_life st_bg_life;
5 } bmcv_ive_bg_model_pix;
```

成员名称	描述
st_work_bg_pixel	工作背景数据。
st_candi_pixel	候选背景数据。
st_bg_life	背景生命力。

【说明】定义背景状态数据。

```

1 typedef struct bmcv_ive_bg_stat_data_s{
2     unsigned int u32_pix_num;
3     unsigned int u32_sum_lum;
4     unsigned char u8_reserved[8];
5 } bmcv_ive_bg_stat_data;
```

成员名称	描述
u32_pix_num	前景像素数目。
u32_sum_lum	输入图像的所有像素亮度累加和。
u8_reserved	保留字段。

【说明】定义背景匹配控制参数。

```

1 typedef struct bmcv_ive_match_bgmodel_attr_s{
2     unsigned int u32_cur_frm_num;
3     unsigned int u32_pre_frm_num;
4     unsigned short u16_time_thr;
5     unsigned char u8_diff_thr_crl_coeff;
```

(续下页)

(接上页)

```

6   unsigned char u8_diff_max_thr;
7   unsigned char u8_diff_min_thr;
8   unsigned char u8_diff_thr_inc;
9   unsigned char u8_fast_learn_rate;
10  unsigned char u8_det_chg_region;
11 } bmcv_ive_match_bgmodel_attr;

```

成员名称	描述
u32_cur_frm_num	当前帧时间。
u32_pre_frm_num	前一帧时间，要求 <code>u32_pre_frm_num &lt; u32_cur_frm_num</code>
u16_time_thr	潜在背景替换时间阈值。 取值范围：[2, 100]，参考取值：20。
u8_diff_thr_crl_coef	差分阈值与灰度值相关系数。 取值范围：[0, 5]，参考取值：0。
u8_diff_max_thr	背景差分阈值调整上限。 取值范围：[3, 15]，参考取值：6。
u8_diff_min_thr	背景差分阈值调整下限。 取值范围：[3, diff_max_thr_u8]，参考取值：4。
u8_diff_thr_inc	动态背景下差分阈值增量。 取值范围：[0, 6]，参考取值：0。
u8_fast_learn_rate	快速背景学习速率。 取值范围：[0, 4]，参考取值：2。
u8_det_chg_region	是否检测变化区域。 取值范围：{0, 1}，0 表示不检测，1 表示检测；参考取值为 0。

**【返回值】**

该函数成功调用时，返回 `BM_SUCCESS`。

**【注意】**

1. 输入输出图像的 width 都需要 16 对齐。

**示例代码**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "bmcv_api_ext_c.h"
#include <unistd.h>

```

(续下页)

(接上页)

```

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main(){
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *input_name = "path/to/input";
    char *ref_name = "path/to/dst";
    bm_handle_t handle = NULL;

    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    int srcStride[4];

    int frameNum;
    int frameNumMax = 100;
    int frmCnt = 0;

    int buf_size = frameNumMax * width * height;
    unsigned char *input_data = (unsigned char*)malloc(buf_size * sizeof(unsigned char));
    unsigned char *src_data = (unsigned char*)malloc(buf_size * sizeof(unsigned char));
    unsigned char *bgmodel_res = (unsigned char*) malloc(width * sizeof(bmcv_ive_bg_model_pix) * height);

    memset(input_data, 0, buf_size * sizeof(unsigned char));
    memset(src_data, 0, buf_size * sizeof(unsigned char));
    memset(bgmodel_res, 0, width * sizeof(bmcv_ive_bg_model_pix) * height);

    FILE *input_fp = fopen(input_name, "rb");
    if(input_fp == NULL){
        printf("%s : No such file \n", ref_name);
        exit(-1);
    }
    fread((void *)input_data, 1, buf_size * sizeof(unsigned char), input_fp);
    fclose(input_fp);

    // create src image
    bm_image src, stFgFlag, stDiffFg, stBgModel;
    bm_device_mem_t stStatData;
    bmcv_ive_match_bgmodel_attr matchBgmodel_attr;

    int u8Stride[4] = {0}, s16Stride[4] = {0}, bgModelStride[4] = {0};

    int data_size = 1;
    u8Stride[0] = align_up(width, 16) * data_size;

```

(续下页)

(接上页)

```

bgModelStride[0] = align_up(width * sizeof(bmcv_ive_bg_model_pix), 16) * F
↳ data_size;
data_size = 2;
s16Stride[0] = align_up(width, 16) * data_size;

ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_
↳ BYTE, &src, u8Stride);
ret = bm_image_create(handle, height, width * sizeof(bmcv_ive_bg_model_pix),
↳ fmt, DATA_TYPE_EXT_1N_BYTE, &stBgModel, bgModelStride);
ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_
↳ BYTE, &stFgFlag, u8Stride);
ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_S16, &
↳ stDiffFg, s16Stride);

ret = bm_image_alloc_dev_mem(src, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stFgFlag, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stDiffFg, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stBgModel, BMCV_HEAP1_ID);
ret = bm_malloc_device_byte(handle, &stStatData, sizeof(bmcv_ive_bg_stat_
↳ data));

// init match_bgmodel config
matchBgmodel_attr.u32_cur_frm_num = 0;
matchBgmodel_attr.u32_pre_frm_num = 0;
matchBgmodel_attr.u16_time_thr = 20;
matchBgmodel_attr.u8_diff_thr_crl_coef = 0;
matchBgmodel_attr.u8_diff_max_thr = 10;
matchBgmodel_attr.u8_diff_min_thr = 10;
matchBgmodel_attr.u8_diff_thr_inc = 0;
matchBgmodel_attr.u8_fast_learn_rate = 4;
matchBgmodel_attr.u8_det_chg_region = 1;

data_size = 1;
srcStride[0] = align_up(width, 16) * data_size;

// create dst image
bm_image stBgImg, stChgStaLife;
bmcv_ive_update_bgmodel_attr update_bgmodel_attr;

int bg_img_stride[4] = {0}, chg_sta_stride[4] = {0};

data_size = 1;
bg_img_stride[0] = align_up(width, 16) * data_size;

data_size = 4;
chg_sta_stride[0] = align_up(width, 16) * data_size;

ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_
↳ BYTE, &stBgImg, bg_img_stride);
ret = bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_U32, &
↳ stChgStaLife, chg_sta_stride);

```

(续下页)

(接上页)

```

ret = bm_image_alloc_dev_mem(stBgImg, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(stChgStaLife, BMCV_HEAP1_ID);

// init update_bgmodel config
update_bgmodel_attr.u32_cur_frm_num = 0;
update_bgmodel_attr.u32_pre_chk_time = 0;
update_bgmodel_attr.u32_frm_chk_period = 30;
update_bgmodel_attr.u32_init_min_time = 25;
update_bgmodel_attr.u32_sty_bg_min_blend_time = 100;
update_bgmodel_attr.u32_sty_bg_max_blend_time = 1500;
update_bgmodel_attr.u16_fg_max_fade_time = 15;
update_bgmodel_attr.u16_bg_max_fade_time = 60;
update_bgmodel_attr.u8_sty_bg_acc_time_rate_thr = 80;
update_bgmodel_attr.u8_chg_bg_acc_time_rate_thr = 60;
update_bgmodel_attr.u8_dyn_bg_acc_time_thr = 0;
update_bgmodel_attr.u8_dyn_bg_depth = 3;
update_bgmodel_attr.u8_bg_eff_sta_rate_thr = 90;
update_bgmodel_attr.u8_acce_bg_learn = 0;
update_bgmodel_attr.u8_det_chg_region = 1;

unsigned char* fg_flag = malloc(width * height);
unsigned char* bg_model = (unsigned char*)malloc(width * sizeof(bmcv_ive_bg_
model_pix) * height);

bmcv_ive_bg_stat_data *stat = malloc(sizeof(bmcv_ive_bg_stat_data));

memset(fg_flag, 0, width * height);
memset(bg_model, 0, width * sizeof(bmcv_ive_bg_model_pix) * height);

for(int i = 0; i < 1; i++){
    int updCnt = 5;
    int preUpdTime = 0;
    int preChkTime = 0;
    int frmUpdPeriod = 10;
    int frmChkPeriod = 30;

    ret = bm_image_copy_host_to_device(stFgFlag, (void**)&fg_flag);
    ret = bm_image_copy_host_to_device(stBgModel, (void**)&bg_model);

    // config setting
    matchBgmodel_attr.u32_cur_frm_num = frmCnt;
    for(frmCnt = 0; frmCnt < frameNumMax; frmCnt++){
        frameNum = frmCnt + 1;
        if(width > 480){
            for(int j = 0; j < 288; j++){
                memcpy(src_data + (j * width),
                       input_data + (frmCnt * 352 * 288 + j * 352), 352);
                memcpy(src_data + (j * width + 352),
                       input_data + (frmCnt * 352 * 288 + j * 352), 352);
            }
        } else {
    }
}

```

(续下页)

(接上页)

```

for(int j = 0; j < 288; j++){
    memcpy(src_data + j * srcStride[0],
           input_data + frmCnt * width * 288 + j * width, width);
    int s = srcStride[0] - width;
    memset(src_data + j * srcStride[0] + width, 0, s);
}
}

ret = bm_image_copy_host_to_device(src, (void**)&src_data);

matchBgmodel_attr.u32_pre_frm_num = matchBgmodel_attr.u32_cur_
↪frm_num;
matchBgmodel_attr.u32_cur_frm_num = frameNum;
ret = bmcv_ive_match_bgmodel(handle, src, stBgModel,
                             stFgFlag, stDiffFg, stStatData, matchBgmodel_attr);

ret = bm_memcpy_d2s(handle, (void*)stat, stStatData);

if(updCnt == 0 || frameNum >= preUpdTIme + frmUpdPeriod){
    updCnt++;
    preUpdTIme = frameNum;
    update_bgmodel_attr.u32_cur_frm_num = frameNum;
    update_bgmodel_attr.u32_pre_chk_time = preChkTime;
    update_bgmodel_attr.u32_frm_chk_period = 0;
    if(frameNum >= preChkTime + frmChkPeriod){
        update_bgmodel_attr.u32_frm_chk_period = frmChkPeriod;
        preChkTime = frameNum;
    }
}

ret = bmcv_ive_update_bgmodel(handle, &src, &stBgModel, &stFgFlag,
↪ &stBgImg, &stChgStaLife, stStatData, update_bgmodel_attr);
ret = bm_memcpy_d2s(handle, (void*)stat, stStatData);
}

ret = bm_image_copy_device_to_host(stBgModel, (void**)&bgmodel_res);
}

free(input_data);
free(src_data);
free(fg_flag);
free(bg_model);
free(stat);

FILE *fp = fopen(ref_name, "wb");
fwrite((void *)bgmodel_res, 1, width * sizeof(bmcv_ive_bg_model_pix) * height,
↪ fp);
fclose(fp);

free(bgmodel_res);

bm_image_destroy(&src);

```

(续下页)

(接上页)

```
bm_image_destroy(&stFgFlag);
bm_image_destroy(&stDiffFg);
bm_image_destroy(&stBgModel);
bm_free_device(handle, stStatData);

bm_image_destroy(&stBgImg);
bm_image_destroy(&stChgStaLife);

bm_dev_free(handle);
return ret;
}
```

### 5.110 bmcv\_ive\_update\_bgmodel

#### 【描述】

该 API 使用 ive 硬件资源，输入当前图像和模型，更新背景模型。

#### 【语法】

```
1 bm_status_t bmcv_ive_update_bgmodel(
2     bm_handle_t           handle,
3     bm_image_t*           cur_img,
4     bm_image_t*           bgmodel_img,
5     bm_image_t*           fgflag_img,
6     bm_image_t*           bg_img,
7     bm_image_t*           chgsta_img,
8     bm_device_mem_t       stat_data_mem,
9     bmcv_ive_update_bgmodel_attr attr);
```

#### 【参数】

表 5.83: bmcv\_ive\_update\_bgmodel 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*cur_img	输入	输入 bm_image 对象指针，当前图像指针，不能为空。
*bgmodel_img	输入/输出	背景模型数据 bm_image 对象指针，不能为空。
*fgflag_img	输入/输出	前景状态图像 bm_image 对象指针，不能为空。
*bg_img	输出	背景图像 bm_image 对象指针，不能为空，宽、高同 fgflag_img。
*chgsta_img	输出	前景更新状态图像 bm_image 对象指针，当 updateBgmodel_attr.u8DetChgRegion 为 0 时，可以为空，宽、高同 fgflag_img。
stat_data_mem	输出	前景状态数据结构体，不能为空，内存至少需配置 sizeof(bm_ive_bg_stat_data)。
attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
cur_img	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
bgmodel_img	GRAY	DATA_TYPE_EXT_1N_BYTE	-
fgflag_img	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
bg_img	GRAY	DATA_TYPE_EXT_S16	64x64~1920x1080
chgsta_img	GRAY	DATA_TYPE_EXT_U32	64x64~1920x1080

## 【数据类型说明】

【说明】 定义背景更新控制参数。

```

1 typedef struct bmcv_ive_update_bgmodel_attr_s{
2     unsigned int u32_cur_frm_num;
3     unsigned int u32_pre_chk_time;
4     unsigned int u32_frm_chk_period;
5     unsigned int u32_init_min_time;
6     unsigned int u32_sty_bg_min_blend_time;
7     unsigned int u32_sty_bg_max_blend_time;
8     unsigned int u32_dyn_bg_min_blend_time;
9     unsigned int u32_static_det_min_time;
10    unsigned short u16_fg_max_fade_time;
11    unsigned short u16_bg_max_fade_time;
12    unsigned char u8_sty_bg_acc_time_rate_thr;
13    unsigned char u8_chg_bg_acc_time_rate_thr;

```

(续下页)

(接上页)

```

14     unsigned char u8_dyn_bg_acc_time_thr;
15     unsigned char u8_dyn_bg_depth;
16     unsigned char u8_bg_eff_sta_rate_thr;
17     unsigned char u8_acce_bg_learn;
18     unsigned char u8_det_chg_region;
19 } bmcv_ive_update_bgmodel_attr;

```

成员名称	描述
u32_cur_frm_num	当前帧时间。
u32_pre_chk_time	上次进行背景状态检查的时间点。
u32_frm_chk_period	背景状态检查时间周期。 取值范围: [0, 2000]。参考取值: 50。
u32_init_min_time	背景初始化最短时间。 取值范围: [20, 6000]。参考取值: 100。
u32_sty_bg_min_blend_time	稳定背景融入最短时间 取值范围: [u32_init_min_time, 6000]。参考取值: 20。
u32_sty_bg_max_blend_time	稳定背景融入最长时间 取值范围: [u32_sty_bg_min_blend_time, 40000]。参考取值: 1500。
u32_dyn_bg_min_blend_time	动态背景融入最短时间。 取值范围: [0, 6000]。参考取值: 1500。
u32_static_det_min_time	静物检测最短时间。 取值范围: [20, 6000]。参考取值: 80。
u16_fg_max_fade_time	前景持续消失最长时间。 取值范围: [1, 255]。参考取值: 15。
u16_bg_max_fade_time	背景持续消失时间。 取值范围: [1, 255]。参考取值: 60。
u8_sty_bg_acc_time_rate	稳态背景访问时间比率阈值。 取值范围: [10, 100]。参考取值: 80。
u8_chg_bg_acc_time_rate	变化背景访问时间比率阈值。 取值范围: [10, 100]。参考取值: 60。
u8_dyn_bg_acc_time_th	动态背景访问时间比率阈值。 取值范围: [0, 50]。参考取值: 0。
u8_dyn_bg_depth	动态背景深度。 取值范围: [0, 3]。参考取值: 3。
u8_bg_eff_sta_rate_thr	背景初始化阶段背景状态时间比率阈值。 取值范围: [90, 100]。参考取值: 90。
u8_acce_bg_learn	是否加速背景学习。 取值范围: {0, 1}, 0 表示不加速, 1 表示加速。参考取值: 0。
u8_det_chg_region	是否检测变化区域。 取值范围: {0, 1}, 0 表示不检测, 1 表示检测。参考取值: 0。

## 【注意事项】

要求  $\leq u32InitMinTime \leq u32StyBgMinBlendTime \leq u32StyBgMaxBlendTime$

#### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

#### 【注意】

- 输入输出图像的 width 都需要 16 对齐。
- 变化状态指像素值发生变化而成为前景, 并且变化后的像素值较长时间都保持稳定的状态, 这一般是由静止遗留物或者静止移走物在图像中产生。

### 5.111 bmcv\_ive\_frame\_diff\_motion

#### 【描述】

该 API 使用 ive 硬件资源, 创建背景相减任务。

#### 【语法】

```
1 bm_status_t bmcv_ive_frame_diff_motion(  
2     bm_handle_t           handle,  
3     bm_image              input1,  
4     bm_image              input2,  
5     bm_image              output,  
6     bmcv_ive_frame_diff_motion_attr attr);
```

#### 【参数】

表 5.84: bmcv\_ive\_frame\_diff\_motion 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input1	输入	输入 bm_image 对象结构体，不能为空。
input2	输入	输入 bm_image 对象结构体，不能为空。
output	输出	输出 bm_image 对象结构体，不能为空。
attr	输入	控制参数结构体，不能为空。

参数名称	图像格式	数据类型	分辨率
input	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
output	GRAY	DATA_TYPE_EXT_1N_BYTE	同 input

## 【数据类型说明】

【说明】 定义该算子控制信息。

```

1 typedef struct bmcv_ive_frame_diff_motion_attr_s{
2     bmcv_ive_sub_mode sub_mode;
3     bmcv_ive_thresh_mode thr_mode;
4     unsigned char u8_thr_low;
5     unsigned char u8_thr_high;
6     unsigned char u8_thr_min_val;
7     unsigned char u8_thr_mid_val;
8     unsigned char u8_thr_max_val;
9     unsigned char au8_erode_mask[25];
10    unsigned char au8_dilate_mask[25];
11 } bmcv_ive_frame_diff_motion_attr;

```

成员名称	描述
sub_mode	定义两张图像相减输出格式。
thr_mode	定义阈值化计算方式，仅支持 MOD_U8 模式，即 U8 数据到 U8 数据的阈值化。
u8_thr_low	低阈值，取值范围：[0, 255]。
u8_thr_high	表示高阈值， $0 \leq u8ThrLow \leq u8ThrHigh \leq 255$ 。
u8_thr_min_val	表示最小值，取值范围：[0, 255]。
u8_thr_mid_val	表示中间值，取值范围：[0, 255]。
u8_thr_max_val	表示最大值，取值范围：[0, 255]。
au8_erode_mask[25]	表示腐蚀的 5x5 模板系数，取值范围：0 或 255。
au8_dilate_mask[25]	表示膨胀的 5x5 模板系数，取值范围：0 或 255。

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

## 【注意】

1. 输入输出图像的 width 都需要 16 对齐。

## 示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmvc_api_ext_c.h"
#include <unistd.h>

#define align_up(num, align) (((num) + ((align) - 1)) & ~((align) - 1))

int main() {
    int dev_id = 0;
    int height = 1080, width = 1920;
    bm_image_format_ext fmt = FORMAT_GRAY;
    char *src1_name = "path/to/src1";
    char *src2_name = "path/to/src2";
    char *dst_name = "path/to/dst";
    bm_handle_t handle = NULL;
    int ret = (int)bm_dev_request(&handle, dev_id);
    if (ret != 0) {
        printf("Create bm handle failed. ret = %d\n", ret);
        exit(-1);
    }

    bm_image src1, src2, dst;
    int stride[4];

    // mask data
    unsigned char arr[25] = {0, 0, 255, 0, 0, 0, 0, 255, 0, 0, 255,
                           255, 255, 255, 255, 0, 0, 255, 0, 0, 255, 0, 0};

    // config setting(Sub->threshold->erode->dilate)
    bmcv_ive_frame_diff_motion_attr attr;
    attr.sub_mode = IVE_SUB_ABS;
    attr.thr_mode = IVE_THRESH_BINARY;
    attr.u8_thr_min_val = 0;
    attr.u8_thr_max_val = 255;
    attr.u8_thr_low = 30;

    memcpy(&attr.au8_erode_mask, &arr, 25 * sizeof(unsigned char));
    memcpy(&attr.au8_dilate_mask, &arr, 25 * sizeof(unsigned char));

    // calc ive image stride && create bm image struct
    int data_size = 1;
```

(续下页)

(接上页)

```

stride[0] = align_up(width, 16) * data_size;

bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src1, stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
src2, stride);
bm_image_create(handle, height, width, fmt, DATA_TYPE_EXT_1N_BYTE, &
dst, stride);

ret = bm_image_alloc_dev_mem(src1, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(src2, BMCV_HEAP1_ID);
ret = bm_image_alloc_dev_mem(dst, BMCV_HEAP1_ID);

// read image data from input files
int byte_size;
unsigned char *input_data;
int image_byte_size[4] = {0};
char *filename[] = {src1_name, src2_name};
bm_image src_images[] = {src1, src2};
for (int i = 0; i < 2; i++) {
    bm_image_get_byte_size(src_images[i], image_byte_size);
    byte_size = image_byte_size[0] + image_byte_size[1] + image_byte_size[2][F]
+ image_byte_size[3];
    input_data = (unsigned char *)malloc(byte_size);
    FILE *fp_src = fopen(filename[i], "rb");
    if (fread((void *)input_data, 1, byte_size, fp_src) < (unsigned int)byte_size) {
        printf("file size is less than required bytes%d\n", byte_size);
    };
    fclose(fp_src);
    void* in_ptr[4] = {(void *)input_data,
                       (void *)((unsigned char *)input_data + image_byte_size[0]),
                       (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1]),
                       (void *)((unsigned char *)input_data + image_byte_size[0] + [F]
image_byte_size[1] + image_byte_size[2])};
    bm_image_copy_host_to_device(src_images[i], in_ptr);
}

ret = bmcv_ive_frame_diff_motion(handle, src1, src2, dst, attr);

unsigned char *ive_add_res = (unsigned char *)malloc(width * height *[F]
sizeof(unsigned char));
memset(ive_add_res, 0, width * height * sizeof(unsigned char));

ret = bm_image_copy_device_to_host(dst, (void **)&ive_add_res);
if(ret != BM_SUCCESS){
    printf("dst bm_image_copy_device_to_host failed, ret is %d \n", ret);
    exit(-1);
}
FILE *fp = fopen(dst_name, "wb");

```

(续下页)

(接上页)

```
fwrite((void *)ive_add_res, 1, width * height * sizeof(unsigned char), fp);
fclose(fp);

free(input_data);
free(ive_add_res);
bm_image_destroy(&src1);
bm_image_destroy(&src2);
bm_image_destroy(&dst);

bm_dev_free(handle);

return 0;
}
```

### 5.112 bmcv\_dpu\_sgbm\_disp

#### 【描述】

该 API 使用 DPU 硬件资源, 实现半全局块匹配算法 SGBM(Semi-Global Block Matching)。

#### 【语法】

```
1 bm_status_t bmcv_dpu_sgbm_disp(
2     bm_handle_t handle,
3     bm_image *left_image,
4     bm_image *right_image,
5     bm_image *disp_image,
6     bmcv_dpu_sgbm_attrs *dpu_attr,
7     bmcv_dpu_sgbm_mode sgbm_mode);
```

#### 【参数】

表 5.85: bmcv\_dpu\_sgbm\_disp 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*left_image	输入	bm_image 类型，参考图指针，不能为空。
*right_image	输入	bm_image 类型，搜索图指针，不能为空，宽、高同 left_image。
*disp_image	输出	bm_image 类型，视差图，不能为空，宽、高同 left_image。
*dpu_attr	输入	bmcv_dpu_sgbm_attrs 类型，SGBM 部分控制参数。
sgbm_mode	输入	DPU SGBM 输出模式。

参数名称	图像格式	数据类型	分辨率
left_image	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
right_image	GRAY	DATA_TYPE_EXT_1N_BYTE	同 left_image
disp_image	GRAY	DATA_TYPE_EXT_1N_BYTE	同 left_image
		DATA_TYPE_EXT_U16	

### 【数据类型说明】

【说明】定义 DPU SGBM 输出模式。

```

1 typedef enum bmcv_dpu_sgbm_mode_{
2     DPU_SGBM_MUX0 = 1,
3     DPU_SGBM_MUX1 = 2,
4     DPU_SGBM_MUX2 = 3,
5 } bmcv_dpu_sgbm_mode;

```

成员名称	描述
DPU_SGBM_MUX0	使用 SGBM 处理左图和右图，输出一张没有经过后处理的 8bit 视差图。
DPU_SGBM_MUX1	使用 SGBM 处理左图和右图，输出一张经过后处理的 16bit 视差图。
DPU_SGBM_MUX2	使用 SGBM 处理左图和右图，输出一张经过后处理的 8bit 视差图。

【说明】定义 DPU SGBM 右图搜索范围。

```

1 typedef enum bmcv_dpu_disp_range_{
2     BMCV_DPU_DISP_RANGE_DEFAULT,

```

(续下页)

(接上页)

```

3   BMCV_DPU_DISP_RANGE_16,
4   BMCV_DPU_DISP_RANGE_32,
5   BMCV_DPU_DISP_RANGE_48,
6   BMCV_DPU_DISP_RANGE_64,
7   BMCV_DPU_DISP_RANGE_80,
8   BMCV_DPU_DISP_RANGE_96,
9   BMCV_DPU_DISP_RANGE_112,
10  BMCV_DPU_DISP_RANGE_128,
11  BMCV_DPU_DISP_RANGE_BUTT
12 } bmcv_dpu_disp_range;

```

成员名称	描述
BMCV_DPU_DISP_RANGE_DEFAULT	默认右图搜索范围为 16 pixel。
BMCV_DPU_DISP_RANGE_16	右图搜索范围为 16 pixel。
BMCV_DPU_DISP_RANGE_32	右图搜索范围为 32 pixel。
BMCV_DPU_DISP_RANGE_48	右图搜索范围为 48 pixel。
BMCV_DPU_DISP_RANGE_64	右图搜索范围为 64 pixel。
BMCV_DPU_DISP_RANGE_80	右图搜索范围为 80 pixel。
BMCV_DPU_DISP_RANGE_96	右图搜索范围为 96 pixel。
BMCV_DPU_DISP_RANGE_112	右图搜索范围为 112 pixel。
BMCV_DPU_DISP_RANGE_128	右图搜索范围为 128 pixel。
BMCV_DPU_DISP_RANGE_BUTT	枚举数组最大值，用于判断输入是否在范围内。

【说明】定义 DPU SGBM BoxFilter 模式。

```

1 typedef enum bmcv_dpu_bfw_mode {
2     DPU_BFW_MODE_DEFAULT,
3     DPU_BFW_MODE_1x1,
4     DPU_BFW_MODE_3x3,
5     DPU_BFW_MODE_5x5,
6     DPU_BFW_MODE_7x7,
7     DPU_BFW_MODE_BUTT
8 } bmcv_dpu_bfw_mode;

```

成员名称	描述
DPU_BFW_MODE_DEFAULT	默认 BoxFilter 窗口大小为 7x7。
DPU_BFW_MODE_1x1	BoxFilter 窗口大小为 1x1。
DPU_BFW_MODE_3x3	BoxFilter 窗口大小为 3x3。
DPU_BFW_MODE_5x5	BoxFilter 窗口大小为 5x5。
DPU_BFW_MODE_7x7	BoxFilter 窗口大小为 7x7。
DPU_BFW_MODE_BUTT	枚举数值的最大值，用于判断输入的枚举值是否在范围内。

【说明】定义 DPU SGBM DCC 代价聚合的模式。

```
1 typedef enum bmcv_dpu_dcc_dir_{
2     BMCV_DPU_DCC_DIR_DEFAULT,
3     BMCV_DPU_DCC_DIR_A12,
4     BMCV_DPU_DCC_DIR_A13,
5     BMCV_DPU_DCC_DIR_A14,
6     BMCV_DPU_DCC_DIR_BUTT
7 } bmcv_dpu_dcc_dir;
```

成员名称	描述
BMCV_DPU_DCC_DIR_DEFAULT	DCC 默认代价聚合方向: A1+A2。
BMCV_DPU_DCC_DIR_A12	DCC 代价聚合方向: A1+A2。
BMCV_DPU_DCC_DIR_A13	DCC 代价聚合方向: A1+A3。
BMCV_DPU_DCC_DIR_A14	DCC 代价聚合方向: A1+A4。
DPU_BFW_MODE_BUTT	枚举数值的最大值，用于判断输入的枚举值是否在范围内。

【说明】定义 DPU SGBM 控制参数。

```
1 typedef struct bmcv_dpu_sgbm_attrs_{
2     bmcv_dpu_bfw_mode    bfw_mode_en;
3     bmcv_dpu_disp_range  disp_range_en;
4     unsigned short       disp_start_pos;
5     unsigned int          dpu_census_shift;
6     unsigned int          dpu_rshift1;
7     unsigned int          dpu_rshift2;
8     bmcv_dpu_dcc_dir     dcc_dir_en;
9     unsigned int          dpu_ca_p1;
10    unsigned int          dpu_ca_p2;
11    unsigned int          dpu_uniq_ratio;
12    unsigned int          dpu_disp_shift;
13 } bmcv_dpu_sgbm_attrs;
```

成员名称	描述
bfw_mode_en	DPU SGBM BoxFilter 窗口大小, 取值可参考 bmcv_dpu_bfw_mode 的说明。
disp_range_en	右图搜索范围, 取值可参考 bmcv_dpu_disp_range 的说明。
disp_start_pos	右图搜索的起始位置。
dpu_census_shift	Census Transform 偏移量。
dpu_rshift1	原图的 BTcost map 的权值。
dpu_rshift2	Census Transform 的 BTcost map 的权值。
dcc_dir_en	DCC 代价聚合方向, 取值可参考 bmcv_dpu_dcc_dir 的说明。
dpu_ca_p1	DCC 代价聚合 P1 惩罚因子。
dpu_ca_p2	DCC 代价聚合 P2 惩罚因子。
dpu_uniq_ratio	唯一性检查因子, 取值范围:[0, 100]。
dpu_disp_shift	视差偏移量。

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 右图像的 width 必须大于或等于视差搜索的起始位置与视差搜索范围的和, 即:  

$$right\_image->width \geq disp\_start\_pos + disp\_range\_en.$$
2. 左右图像的 height 和 width 必须相同。
3. 左右图像的 width 要求 4 对齐, height 要求 2 对齐, stride 需要 16 对齐。

## 5.113 bmcv\_dpu\_fgs\_disp

### 【描述】

该 API 使用 DPU 硬件资源, 快速全局平滑算法 FGS(Fast Global Smoothing)。

### 【语法】

```

1 bm_status_t bmcv_dpu_fgs_disp(
2     bm_handle_t handle,
3     bm_image *guide_image,
4     bm_image *smooth_image,
5     bm_image *disp_image,
6     bmcv_dpu_fgs_attrs *fgs_attr,
7     bmcv_dpu_fgs_mode fgs_mode);

```

## 【参数】

表 5.86: bmcv\_dpu\_fgs\_disp 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*guide_image	输入	bm_image 类型，指导图指针，不能为空。
*smooth_image	输入	bm_image 类型，视差图指针，不能为空，宽、高同 guide_image。
*disp_image	输出	bm_image 类型，平滑图，不能为空，宽、高同 guide_image。
*fgs_attr	输入	bmcv_dpu_fgs_attrs 类型，FGS 部分控制参数。
fgs_mode	输入	DPU FGS 输出模式。

参数名称	图像格式	数据类型	分辨率
guide_image	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
smooth_image	GRAY	DATA_TYPE_EXT_1N_BYTE	同 guide_image
disp_image	GRAY	DATA_TYPE_EXT_1N_BYTE DATA_TYPE_EXT_U16	同 guide_image

## 【数据类型说明】

【说明】 定义 DPU FGS 输出模式。

```

1 typedef enum bmcv_dpu_fgs_mode {
2     DPU_FGS_MUX0 = 7,
3     DPU_FGS_MUX1 = 8,
4 } bmcv_dpu_fgs_mode;

```

成员名称	描述
DPU_FGS_MUX0	使用 FGS 处理左图和右图，输出一张 8bit 视差图（也可用于图像的降噪，类似于引导滤波）。
DPU_FGS_MUX1	使用 FGS 处理左图和右图，输出一张 16bit 深度图。

【说明】定义 DPU FGS 深度单位。

```

1 typedef enum bmcv_dpu_depth_unit {
2     BMCV_DPU_DEPTH_UNIT_DEFAULT,
3     BMCV_DPU_DEPTH_UNIT_MM,
4     BMCV_DPU_DEPTH_UNIT_CM,
5     BMCV_DPU_DEPTH_UNIT_DM,
6     BMCV_DPU_DEPTH_UNIT_M,
7     BMCV_DPU_DEPTH_UNIT_BUTT
8 } bmcv_dpu_depth_unit;

```

成员名称	描述
BMCV_DPU_DEPTH_UNIT_DEFAULT	默认深度单位，mm。
BMCV_DPU_DEPTH_UNIT_MM	深度单位是 mm。
BMCV_DPU_DEPTH_UNIT_CM	深度单位是 cm。
BMCV_DPU_DEPTH_UNIT_DM	深度单位是 dm。
BMCV_DPU_DEPTH_UNIT_M	深度单位是 m。
BMCV_DPU_DEPTH_UNIT_BUTT	枚举数组最大值，用于判断输入是否在范围内。

【说明】定义 DPU FGS 控制参数。

```

1 typedef struct bmcv_dpu_fgs_attrs_ {
2     unsigned int      fgs_max_count;
3     unsigned int      fgs_max_t;
4     unsigned int      fxbase_line;
5     bmcv_dpu_depth_unit depth_unit_en;
6 } bmcv_dpu_fgs_attrs;

```

成员名称	描述
fgs_max_count	Fgs 中转变为 0 的最大次数。
fgs_max_t	Fgs 的最大迭代次数（至少设置为 2），由于 FGS 计算量较大，一般设置为 3。
fxbase_line	左右摄像头的基线值。
depth_unit_en	深度的度量单位，取值可参考 bmcv_dpu_depth_unit 说明。

【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【注意】

1. 左右图像的 height 和 width 必须相同。
2. 左右图像的 width 要求 4 对齐, height 要求 2 对齐, 输入图 stride 需要 16 对齐, 输出图 stride 需要 32 对齐。
3. Fgs 的最大迭代次数至少设置为 2。

### 5.114 bmcv\_dpu\_online\_disp

### 【描述】

该 API 使用 DPU 硬件资源, 实现半全局块匹配算法 SGBM(Semi-Global Block Matching) 跟快速全局平滑算法 FGS(Fast Global Smoothing)。

### 【语法】

```
1 bm_status_t bmcv_dpu_online_disp(  
2     bm_handle_t handle,  
3     bm_image *left_image,  
4     bm_image *right_image,  
5     bm_image *disp_image,  
6     bmcv_dpu_sgbm_attrs *dpu_attr,  
7     bmcv_dpu_fgs_attrs *fgs_attr,  
8     bmcv_dpu_online_mode online_mode);
```

### 【参数】

表 5.87: bmcv\_dpu\_online\_disp 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
*left_image	输入	bm_image 类型，指导图指针，不能为空。
*right_image	输入	bm_image 类型，平滑图指针，不能为空，宽、高同 left_image。
*disp_image	输出	bm_image 类型，视差图，不能为空，宽、高同 left_image。
*dpu_attr	输入	bmcv_dpu_sgbm_attrs 类型，SGBM 部分控制参数。
*fgs_attr	输入	bmcv_dpu_fgs_attrs 类型，FGS 部分控制参数。
online_mode	输入	DPU online (SGBM_TO_FGS) 输出模式。

参数名称	图像格式	数据类型	分辨率
left_image	GRAY	DATA_TYPE_EXT_1N_BYTE	64x64~1920x1080
right_image	GRAY	DATA_TYPE_EXT_1N_BYTE	同 left_image
disp_image	GRAY	DATA_TYPE_EXT_1N_BYTE DATA_TYPE_EXT_U16	同 left_image

**【数据类型说明】****【说明】** 定义 DPU FGS 输出模式。

```

1 typedef enum bmcv_dpu_online_mode {
2     DPU_ONLINE_MUX0 = 4,
3     DPU_ONLINE_MUX1 = 5,
4     DPU_ONLINE_MUX2 = 6,
5 } bmcv_dpu_online_mode;

```

成员名称	描述
DPU_ONLINE_MUX0	该模式下，使用 FGS 处理左图和右图，输出一张 8bit 视差图（也可用于图像的降噪，类似于引导滤波）。
DPU_ONLINE_MUX1	该模式下，使用 SGBM、FGS 处理左图和右图，输出一张 16bit 深度图。
DPU_ONLINE_MUX2	该模式下，单独使用 SGBM 处理左图和右图，输出一张 16bit 深度图。

**【注意】** bmcv\_dpu\_sgbm\_attrs 及 bmcv\_dpu\_fgs\_attrs 类型的说明，详见 bmcv\_dpu\_fgs\_disp 及 bmcv\_dpu\_sgbm\_disp。

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【注意】

1. 左右图像的 height 和 width 必须相同。
1. 左右图像的 width 要求 16 对齐, height 要求 2 对齐, 输入图 stride 需要 16 对齐, 输出图 stride 需要 32 对齐。

## 5.115 bmcv\_tde\_fill

### 【描述】

利用 TDE 硬件将 bm\_image 的指定位置填充指定颜色。

### 【语法】

```
1 bm_status_t bmcv_tde_fill(
2     bm_handle_t handle,
3     bm_image_t image,
4     bmcv_color_ext color,
5     bmcv_rect_t* rect);
```

### 【参数】

表 5.88: bmcv\_tde\_fill 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄, 通过调用 bm_dev_request 获取。
image	输入	待处理图像的 bm_image。
color	输入	指定颜色, 包含 A/R/G/B 四个值。当底图为非 ARGB 格式时, A 数值不会生效。
rect	输入	填充位置的指针, 传入 NULL 表示全图填充。

## 【结构体】

```
1 typedef struct {
2     unsigned char r;
3     unsigned char g;
4     unsigned char b;
5     unsigned char a;
6 } bmcv_color_ext;
```

## 【格式支持】

num	input image_format
1	FORMAT_RGB_PACKED
2	FORMAT_BGR_PACKED
3	FORMAT_ARGB_PACKED
4	FORMAT_ABGR_PACKED
5	FORMAT_ARGB1555_PACKED
6	FORMAT_ABGR1555_PACKED
7	FORMAT_ARGB4444_PACKED
8	FORMAT_ABGR4444_PACKED

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

int main() {
    bm_handle_t handle;
    bm_image image;
    bmcv_color_ext color = {.a = 255, .r = 255, .g = 0, .b = 0};
    bmcv_rect_t rect = {.start_x = 0, .start_y = 0, .crop_w = 1920, .crop_h = 1080};
```

(续下页)

(接上页)

```
int h = 1080, w = 1920;
bm_image_format_ext fmt = FORMAT_ARGB_PACKED;

bm_dev_request(&handle, 0);
bm_image_create(handle, h, w, fmt, DATA_TYPE_EXT_1N_BYTE, &image,[F
˓→NULL);
bm_image_alloc_dev_mem(image, BMCV_HEAP1_ID);

bmcv_tde_fill(handle, image, color, &rect);

bm_write_bin(image, "out.raw");
bm_image_destroy(&image);
bm_dev_free(handle);
return 0;
}
```

### 5.116 bmcv\_tde\_convert

#### 【描述】

利用 TDE 硬件完成 bm\_image 的 crop + resize + csc + rotate 变换，并以指定方式融合到目标 bm\_image 的指定位置。

#### 【语法】

```
1 bm_status_t bmcv_tde_convert(
2     bm_handle_t handle,
3     bm_image    input,
4     bm_image    output,
5     int         rot_angle,
6     int         global_alpha,
7     bmcv_rect_t* src_rect,
8     bmcv_rect_t* dst_rect);
```

#### 【参数】

表 5.89: bmcv\_tde\_convert 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
input	输入	待处理图像的 bm_image。
output	输入	输出图像的 bm_image。
rot_angle	输入	旋转角度，支持 90/180/270 度旋转，传入 0 表示不旋转。
global_alpha	输入	全局透明度。取值为 1-255 时，统一调整 Src 的 alpha 通道。取值为 -15-0 时，按指定公式处理。详见下文。
src_rect	输入	crop 位置的指针，传入 NULL 表示不进行 crop。
dst_rect	输入	输出位置的指针，传入 NULL 表示拉伸到输出图的宽高。

## 【输入格式支持】

num	input image_format
1	FORMAT_YUV420P
2	FORMAT_YUV422P
3	FORMAT_YUV444P
4	FORMAT_NV12
5	FORMAT_NV16
6	FORMAT_YUV422_YUYV
7	FORMAT_RGB_PACKED
8	FORMAT_BGR_PACKED
9	FORMAT_ARGB_PACKED
10	FORMAT_ABGR_PACKED
11	FORMAT_ARGB1555_PACKED
12	FORMAT_ABGR1555_PACKED
13	FORMAT_ARGB4444_PACKED
14	FORMAT_ABGR4444_PACKED

## 【输出格式支持】

num	input image_format
1	FORMAT_RGB_PACKED
2	FORMAT_BGR_PACKED
3	FORMAT_ARGB_PACKED
4	FORMAT_ABGR_PACKED
5	FORMAT_ARGB1555_PACKED
6	FORMAT_ABGR1555_PACKED
7	FORMAT_ARGB4444_PACKED
8	FORMAT_ABGR4444_PACKED

## 【注意事项】

该接口限制输入的 bm\_image 的 stride 为 32 对齐。

global\_alpha 取值为 0-255 时，表示统一调整 Src 的 alpha 通道，融合时计算公式为

$$\text{Src.alpha} = \text{Src.alpha} * (\text{global\_alpha} / 255)$$

$$\text{Out} = (\text{Src.alpha} / 255) * \text{Src} + ((255 - \text{Src.alpha}) / 255) * \text{Dst}$$

当 Src 没有 alpha 通道时，Src.alpha 取 255。

global\_alpha 取值为 -15-0 时，表示按指定公式处理，如下：

```

1  typedef enum bm_tde_overlay {
2      BM_BLEND_NONE          = 0, /*! S, No blend */
3      BM_BLEND_SRC_OVER       = -1, /*! S + (1 - S.a) * D */
4      BM_BLEND_DST_OVER       = -2, /*! (1 - D.a) * S + D */
5      BM_BLEND_SRC_IN         = -3, /*! D.a * S */
6      BM_BLEND_DST_IN         = -4, /*! S.a * D */
7      BM_BLEND_MULTIPLY       = -5, /*! S * (1 - D.a) + D * (1 - S.a) + S * D */
8      BM_BLEND_SCREEN          = -6, /*! S + D - S * D */
9      BM_BLEND_DARKEN          = -7, /*! min(SrcOver, DstOver) */
10     BM_BLEND_LIGHTEN         = -8, /*! max(SrcOver, DstOver) */
11     BM_BLEND_ADDITIVE        = -9, /*! S + D */
12     BM_BLEND_SUBTRACT        = -10, /*! D * (1 - S.a) */
13     BM_BLEND_SUBTRACT_LVGL   = -11, /*! D - S */
14     BM_BLEND_NORMAL_LVGL     = -12, /*! S * S.a + (1 - S.a) * D */
15     BM_BLEND_ADDITIVE_LVGL   = -13, /*! (S + D) * S.a + D * (1 - S.a) */
16     BM_BLEND_MULTIPLY_LVGL   = -14, /*! (S * D) * S.a + D * (1 - S.a) */
17     BM_BLEND_PREMULTIPLY_SRC_OVER = -15, /*! S * S.a + (1 - S.a) * D */
18 } bm_tde_overlay_t;

```

## 【返回值】

该函数成功调用时，返回 BM\_SUCCESS。

### 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

int main() {
    bm_handle_t handle;
    bm_image input, output;
    bmcv_rect_t src_rect = {.start_x = 0, .start_y = 0, .crop_w = 1920, .crop_h = [F]
    ↵1080};
    bmcv_rect_t dst_rect = {.start_x = 0, .start_y = 0, .crop_w = 2560, .crop_h = [F]
    ↵1440};
    int src_h = 1080, src_w = 1920, dst_h = 2560, dst_w = 1440;
    bm_image_format_ext src_fmt = FORMAT_RGB_PACKED;
    bm_image_format_ext dst_fmt = FORMAT_ARGB_PACKED;

    bm_dev_request(&handle, 0);
    bm_image_create(handle, src_h, src_w, src_fmt, DATA_TYPE_EXT_1N_BYTE,
    ↵ &input, NULL);
    bm_image_alloc_dev_mem(input, BMCV_HEAP1_ID);
    bm_read_bin(input, "/opt/sophon/current/bin/res/1920x1080_rgb.bin");

    bm_image_create(handle, dst_h, dst_w, dst_fmt, DATA_TYPE_EXT_1N_
    ↵BYTE, &output, NULL);
    bm_image_alloc_dev_mem(output, BMCV_HEAP1_ID);

    bmcv_tde_convert(handle, input, output, 0, 0, &src_rect, &dst_rect);

    bm_write_bin(output, "out.raw");
    bm_image_destroy(&input);
    bm_image_destroy(&output);
    bm_dev_free(handle);
    return 0;
}
```

### 5.117 bmcv\_tde\_line

#### 【描述】

利用 TDE 硬件在 bm\_image 的指定位置划线。

#### 【语法】

```
1 bm_status_t bmcv_tde_line(
2     bm_handle_t handle,
3     bm_image    image,
4     bmcv_point_t start,
5     bmcv_point_t end,
6     bmcv_color_ext color,
7     int         thick);
```

#### 【参数】

表 5.90: bmcv\_tde\_line 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image	输入	待处理图像的 bm_image。
start	输入	划线左上坐标。
end	输入	划线右下坐标。
color	输入	划线颜色，包含 A/R/G/B 四个值。可调节 A 来改变填充封闭图形的透明度。
thick	输入	划线宽度。

#### 【结构体】

```
1 typedef struct {
2     unsigned char r;
3     unsigned char g;
4     unsigned char b;
5     unsigned char a;
6 } bmcv_color_ext;
```

## 【格式支持】

num	input image_format
1	FORMAT_RGB_PACKED
2	FORMAT_BGR_PACKED
3	FORMAT_ARGB_PACKED
4	FORMAT_ABGR_PACKED
5	FORMAT_ARGB1555_PACKED
6	FORMAT_ABGR1555_PACKED
7	FORMAT_ARGB4444_PACKED
8	FORMAT_ABGR4444_PACKED

## 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

## 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

int main() {
    bm_handle_t handle;
    bm_image image;
    bmcv_color_ext color = {.a = 255, .r = 255, .g = 0, .b = 0};
    bmcv_point_t start = {.x = 0, .y = 0};
    bmcv_point_t end = {.x = 200, .y = 200};
    int h = 1080, w = 1920;
    bm_image_format_ext fmt = FORMAT_RGB_PACKED;
    int thick = 2;

    bm_dev_request(&handle, 0);
    bm_image_create(handle, h, w, fmt, DATA_TYPE_EXT_1N_BYTE, &image, F
    ↵NULL);
    bm_image_alloc_dev_mem(image, BMCV_HEAP1_ID);
    bm_read_bin(image, "/opt/sophon/libsonn-current/bin/res/1920x1080_rgb.bin");

    bmcv_tde_line(handle, image, start, end, color, thick);
```

(续下页)

(接上页)

```

bm_write_bin(image, "out.raw");
bm_image_destroy(&image);
bm_dev_free(handle);
return 0;
}

```

### 5.118 bmcv\_tde\_draw

#### 【描述】

利用 TDE 硬件在 bm\_image 的指定位置填充不规则图形。输入各顶点坐标，按顺序连接顶点，最后一个顶点则连接首顶点，填充围成的所有封闭部分。

#### 【语法】

```

1 bm_status_t bmcv_tde_draw(
2     bm_handle_t handle,
3     bm_image_t image,
4     bmcv_point_t *point,
5     int path_num,
6     bmcv_color_ext color)

```

#### 【参数】

表 5.91: bmcv\_tde\_draw 参数表

参数名称	输入/输出	描述
handle	输入	设备环境句柄，通过调用 bm_dev_request 获取。
image	输入	待处理图像的 bm_image。
point	输入	多边形顶点坐标指针。
path_num	输入	多边形顶点数。
color	输入	填充图形颜色，包含 A/R/G/B 四个值。可调节 A 来改变填充封闭图形的透明度。

#### 【结构体】

```
1 typedef struct {
2     unsigned char r;
3     unsigned char g;
4     unsigned char b;
5     unsigned char a;
6 } bmcv_color_ext;
```

### 【格式支持】

num	input image_format
1	FORMAT_RGB_PACKED
2	FORMAT_BGR_PACKED
3	FORMAT_ARGB_PACKED
4	FORMAT_ABGR_PACKED
5	FORMAT_ARGB1555_PACKED
6	FORMAT_ABGR1555_PACKED
7	FORMAT_ARGB4444_PACKED
8	FORMAT_ABGR4444_PACKED

### 【返回值】

该函数成功调用时, 返回 BM\_SUCCESS。

### 【代码示例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include "bmcv_api_ext_c.h"

int main() {
    bm_handle_t handle;
    bm_image image;
    bmcv_color_ext color = {.a = 255, .r = 255, .g = 0, .b = 0};
    bmcv_point_t point[5] =
        {{.x = 100, .y = 70},
         {.x = 128, .y = 90},
         {.x = 117, .y = 124},
```

(续下页)

(接上页)

```
{.x = 82, .y = 124},  
 {.x = 71, .y = 90}}; //五边形  
int h = 1080, w = 1920;  
bm_image_format_ext fmt = FORMAT_RGB_PACKED;  
int point_num = 5;  
  
bm_dev_request(&handle, 0);  
bm_image_create(handle, h, w, fmt, DATA_TYPE_EXT_1N_BYTE, &image,F  
←NULL);  
bm_image_alloc_dev_mem(image, BMCV_HEAP1_ID);  
bm_read_bin(image, "/opt/sophon/libphon-current/bin/res/1920x1080_rgb.bin");  
  
bmcv_tde_draw(handle, image, point, point_num, color);  
  
bm_write_bin(image, "out.raw");  
bm_image_destroy(&image);  
bm_dev_free(handle);  
return 0;  
}
```