AW G2D
开发指南

版本号: 2.4
发布日期: 2024.8.20

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.6.30	AWA1572	创建该文档。
2.0	2020.11.18	AWA1639	更新适配 linux5.4。
2.1	2021.4.10	AWA1693	添加输出宽度限制说明。
2.2	2022.7.11	AWA1836	更新适配 linux-5.10。
2.3	2024.3.14	AWA2072	删除部分冗余内容，新增各版本 g2d 性能指标，新增各功能使用 demo，新增限制条件说明，新增 FAQ 记录。
2.4	2024.8.20	AWA2072	修复使用 demo 中有误的参数

目　录

插　图

1 前言

1.1 文档简介

本文主要介绍 sunxi 平台 G2D 模块的设计指标、功能、限制条件、模块配置和调用方法。

1.2 目标读者

- G2D 驱动开发人员/维护人员
- 使用 G2D 模块进行 2D 图形操作加速的方案开发者

1.3 适用范围

本文档的某些特性仅适用于最新版本的 G2D 驱动，如发现文档和所使用驱动有不同处，请联系我司获取最新版本 G2D 驱动

2 2. 概述

G2D 模块主要实现图像旋转、镜像、图像裁剪和拼接、图像数据下采样和无极缩放、数据格式转换、颜色空间转换、图层合成、ColorKey、矩形区域颜色填充、点/线绘制、二元光栅操作、三元光栅操作等加速功能，具体功能与 IP 版本有关。

2.1 相关术语介绍

术语	说明
Rotate/Mirror	对图像做旋转/镜像
Crop/Stitching	图像裁剪、拼接
Scaler/Down_Sampling	图像缩放/图像数据下采样，比如 Y8 每隔一行、一列采样一次，2048*2048->1024/1024
PixelFormat_Conversion	像素格式转换，比如 ARGB8888 转 NV12
ColorSpace_Conversion	颜色空间转换，比如 rgb bt709 转 yuv bt601
Point/Line DrawingRect_ColorFilling	画点/画线，往某一矩形区域填充指定颜色
ColorKey	两个图像叠加混合的时候，对特殊色做特殊过滤，符合条件的区域叫 match 区，在 match 区就全部使用另外一个图层的颜色值，不符合条件的区域就是非 match 区，非 match 区就是走普通的 alpha 混合
Alpha Blending	对任意两张图像可以合成为一张图像，合成图像的像素取值根据数学公式：RGB3 = (1- a) * RGB1 + a * RGB2
ROI	ROI 代表感兴趣区域（Region of Interest），通常用 ROI(x, y, w, h) 来表示。在图像处理领域，ROI 指的是对图像中的特定区域感兴趣，而不是整个图像。. 通过指定 ROI，可以针对图像中的特定区域进行操作，例如裁剪、处理或提取感兴趣的信息。x 表示矩形区域的左上角点的 x 坐标，y 表示矩形区域的左上角点的 y 坐标，w 表示矩形区域的宽度（width）h 表示矩形区域的高度（height）
RCQ/AHB	rcq，ahb 为两种更新寄存器的方式，rcq 为硬件自更新寄存器，能够下发批处理任务，节省中断响应时间

2.2 设计指标

Version	Chip	Function	Source Resolution(Max)	Destination Resoultion(Max)	Pixels/Cycle	Clock Frequency
G2D100 (MixerProcessor-AHB)	T7 V5 A50 R311 MR133 B300 V5V200 V316(SDV) V536(CDR) V526(CDR) A20E V40 T3 T3A T3L R40 A40I	Rotate/Mirror Crop/Stitching Scaler/Down_Sampling PixelFormat_Conversion ColorSpace_Conversion Alpha Blending Point/Line Drawing Rect_ColorFilling ColorKey Rop2 Rop3	2048x2048	2048x2048	1	300M
G2D101r1p0(Mixer-RCQ, Rotate-AHB)	V833(IPC) V831(IPC) V535(CDR) V533(CDR) V459 QG2101A R128 V853 V821	Rotate/Mirror Crop/Stitching Scaler/Down_Sampling PixelFormat_Conversion ColorSpace_Conversion Alpha Blending Point/Line Drawing Rect_ColorFilling ColorKey Rop2 Rop3	Rotate/Mirror: 4096x4096 others: 2048x2048	Rotate/Mirror: 4096x4096 others: 2048x2048	Rotate/Mirror: 4 others:	300M
	H3pro H616 H313 H700 VMP1002 IK316 H618 T5					

图 2-1: design_spec0

G2D101r1p0 (Rotate-AHB)	T507 T503 T513 T517 A100 A133 A133P Y509 MR813 R818 B810 A523 A527 T527 MR527 AI985 H728	Rotate/Mirror Crop/Stitching	4096x4096	2048x2048	4	300M

图 2-2: design_spec1

注：
1). Crop/Stiching 功能有两种实现方式，性能有所差异，具体见 Crop/Stiching 功能说明
2). G2D100 可以实现 rotate 和 mixer 一次完成，G2D101r1p0 需分两次去做
3). Pixels/Cycle 和 Clock Frequency 可用于计算理论性能，但实际运行表现与带宽、cpu 占用率有关。
4). 关于输入输出分辨率，如果用到 Down Sampling 功能，输入最大分辨率可达到 8176x8176。

2.3 图像格式支持

Version	Chip	Input Data Format	Output Data Format
MixerProcessor (G2D100)	T7 V5A50 R311 MR133 B300 V5V200 V316(SDV) V536(CDR) V526(CDR) A20E V40 T3 T3A T3L R40 A40I	G2D_FMT_ARGB_AYUV8888 G2D_FMT_BGRA_VUYA8888 G2D_FMT_ABGR_AVUY8888 G2D_FMT_RGBA_YUVA8888 G2D_FMT_XRGB8888 G2D_FMT_BGRX8888 G2D_FMT_XBGR8888 G2D_FMT_RGBX8888 G2D_FMT_ARGB4444 G2D_FMT_ABGR4444 G2D_FMT_RGBA4444 G2D_FMT_BGRA4444 G2D_FMT_ARGB1555 G2D_FMT_ABGR1555 G2D_FMT_RGBA5551 G2D_FMT_BGRA5551 G2D_FMT_RGB565 G2D_FMT_BGR565 G2D_FMT_IYUV422 G2D_FMT_8BPP_MONO G2D_FMT_4BPP_MONO G2D_FMT_2BPP_MONO G2D_FMT_1BPP_MONO G2D_FMT_PYUV422UVC G2D_FMT_PYUV420UVC G2D_FMT_PYUV411UVC /* just for input format */ G2D_FMT_8BPP_PALETTE G2D_FMT_4BPP_PALETTE G2D_FMT_2BPP_PALETTE G2D_FMT_1BPP_PALETTE G2D_FMT_PYUV422UVC_MB16	G2D_FMT_ARGB_AYUV8888 G2D_FMT_BGRA_VUYA8888 G2D_FMT_ABGR_AVUY8888 G2D_FMT_RGBA_YUVA8888 G2D_FMT_XRGB8888 G2D_FMT_BGRX8888 G2D_FMT_XBGR8888 G2D_FMT_RGBX8888 G2D_FMT_ARGB4444 G2D_FMT_ABGR4444 G2D_FMT_RGBA4444 G2D_FMT_BGRA4444 G2D_FMT_ARGB1555 G2D_FMT_ABGR1555 G2D_FMT_RGBA5551 G2D_FMT_BGRA5551 G2D_FMT_RGB565 G2D_FMT_BGR565 G2D_FMT_IYUV422 G2D_FMT_8BPP_MONO G2D_FMT_4BPP_MONO G2D_FMT_2BPP_MONO G2D_FMT_1BPP_MONO G2D_FMT_PYUV422UVC G2D_FMT_PYUV420UVC G2D_FMT_PYUV411UVC

图 2-3: pixel_format0

		G2D_FMT_PYUV420UVC_MB16 G2D_FMT_PYUV411UVC_MB16 G2D_FMT_PYUV422UVC_MB32 G2D_FMT_PYUV420UVC_MB32 G2D_FMT_PYUV411UVC_MB32 G2D_FMT_PYUV422UVC_MB64 G2D_FMT_PYUV420UVC_MB64 G2D_FMT_PYUV411UVC_MB64 G2D_FMT_PYUV422UVC_MB128 G2D_FMT_PYUV420UVC_MB128 G2D_FMT_PYUV411UVC_MB128	/* just for output format */ G2D_FMT_PYUV422 G2D_FMT_PYUV420 G2D_FMT_PYUV4112048x2048
G2D101r1p0（Mixer-Rcq）	V833(IPC) V831(IPC) V535(CDR) V533(CDR) V459 QG2101A R128	Rotate/Mirror: 同 G2D101r1p0 (Rotate-Only) Others: G2D_FORMAT_ARGB8888 G2D_FORMAT_ABGR8888 G2D_FORMAT_RGBA8888 G2D_FORMAT_BGRA8888 G2D_FORMAT_XRGB8888 G2D_FORMAT_XBGR8888 G2D_FORMAT_RGBX8888 G2D_FORMAT_BGRX8888 G2D_FORMAT_RGB888 G2D_FORMAT_BGR888 G2D_FORMAT_RGB565 G2D_FORMAT_BGR565 G2D_FORMAT_ARGB4444 G2D_FORMAT_ABGR4444 G2D_FORMAT_RGBA4444 G2D_FORMAT_BGRA4444 G2D_FORMAT_ARGB1555 G2D_FORMAT_ABGR1555 G2D_FORMAT_RGBA5551 G2D_FORMAT_BGRA5551	Rotate/Mirror: 同 G2D101r1p0 (Rotate-Only) Others: G2D_FORMAT_ARGB8888 G2D_FORMAT_ABGR8888 G2D_FORMAT_RGBA8888 G2D_FORMAT_BGRA8888 G2D_FORMAT_XRGB8888 G2D_FORMAT_XBGR8888 G2D_FORMAT_RGBX8888 G2D_FORMAT_BGRX8888 G2D_FORMAT_RGB888 G2D_FORMAT_BGR888 G2D_FORMAT_RGB565 G2D_FORMAT_BGR565 G2D_FORMAT_ARGB4444 G2D_FORMAT_ABGR4444 G2D_FORMAT_RGBA4444 G2D_FORMAT_BGRA4444 G2D_FORMAT_ARGB1555 G2D_FORMAT_ABGR1555 G2D_FORMAT_RGBA5551 G2D_FORMAT_BGRA5551

图 2-4: pixel_format1

	V853 V821	/* invailed for UI channel */ G2D_FORMAT_IYUV422_V0Y1U0Y0 G2D_FORMAT_IYUV422_Y1V0Y0U0 G2D_FORMAT_IYUV422_U0Y1V0Y0 G2D_FORMAT_IYUV422_Y1U0Y0V0 G2D_FORMAT_YUV422UVC_V1U1V0U0 G2D_FORMAT_YUV422UVC_U1V1U0V0 G2D_FORMAT_YUV422_PLANAR G2D_FORMAT_YUV420UVC_V1U1V0U0 G2D_FORMAT_YUV420UVC_U1V1U0V0 G2D_FORMAT_YUV420_PLANAR G2D_FORMAT_YUV411UVC_V1U1V0U0 G2D_FORMAT_YUV411UVC_U1V1U0V0 G2D_FORMAT_YUV411_PLANAR G2D_FORMAT_Y8 = 0x30	/* invailed for UI channel */ G2D_FORMAT_IYUV422_V0Y1U0Y0 G2D_FORMAT_IYUV422_Y1V0Y0U0 G2D_FORMAT_IYUV422_U0Y1V0Y0 G2D_FORMAT_IYUV422_Y1U0Y0V0 G2D_FORMAT_YUV422UVC_V1U1V0U0 G2D_FORMAT_YUV422UVC_U1V1U0V0 G2D_FORMAT_YUV422_PLANAR G2D_FORMAT_YUV420UVC_V1U1V0U0 G2D_FORMAT_YUV420UVC_U1V1U0V0 G2D_FORMAT_YUV420_PLANAR G2D_FORMAT_YUV411UVC_V1U1V0U0 G2D_FORMAT_YUV411UVC_U1V1U0V0 G2D_FORMAT_YUV411_PLANAR G2D_FORMAT_Y8 = 0x30
	H3pro H616 H313 H700 VMP1002	G2D_FORMAT_ARGB8888 G2D_FORMAT_ABGR8888 G2D_FORMAT_RGBA8888 G2D_FORMAT_BGRA8888 G2D_FORMAT_XRGB8888 G2D_FORMAT_XBGR8888 G2D_FORMAT_RGBX8888 G2D_FORMAT_BGRX8888 G2D_FORMAT_RGB888 G2D_FORMAT_BGR888 G2D_FORMAT_RGB565 G2D_FORMAT_BGR565 G2D_FORMAT_ARGB4444 G2D_FORMAT_ABGR4444 G2D_FORMAT_RGBA4444 G2D_FORMAT_BGRA4444 G2D_FORMAT_ARGB1555	G2D_FORMAT_ARGB8888 G2D_FORMAT_ABGR8888 G2D_FORMAT_RGBA8888 G2D_FORMAT_BGRA8888 G2D_FORMAT_XRGB8888 G2D_FORMAT_XBGR8888 G2D_FORMAT_RGBX8888 G2D_FORMAT_BGRX8888 G2D_FORMAT_RGB888 G2D_FORMAT_BGR888 G2D_FORMAT_RGB565 G2D_FORMAT_BGR565 G2D_FORMAT_ARGB4444 G2D_FORMAT_ABGR4444 G2D_FORMAT_RGBA4444 G2D_FORMAT_BGRA4444 G2D_FORMAT_ARGB1555

图 2-5: pixel_format2

G2D101r1p0 (Rotate-Only)	IK316 H618 T5 T507 T503 T513 T517 A100 A133 A133P Y509 MR813 R818 B810 A523 A527 T527 MR527 AI985 H728	G2D_FORMAT_ABGR1555 G2D_FORMAT_RGBA5551 G2D_FORMAT_BGRA5551 G2D_FORMAT_ARGB2101010 G2D_FORMAT_ABGR2101010 G2D_FORMAT_RGBA1010102 G2D_FORMAT_BGRA1010102 /* invailed for UI channel / *G2D_FORMAT_IYUV422_V0Y1U0Y0* *G2D_FORMAT_IYUV422_Y1V0Y0U0* *G2D_FORMAT_IYUV422_U0Y1V0Y0* *G2D_FORMAT_IYUV422_Y1U0Y0V0* *G2D_FORMAT_YUV422UVC_V1U1V0U0* *G2D_FORMAT_YUV422UVC_U1V1U0V0* *G2D_FORMAT_YUV422_PLANAR* *G2D_FORMAT_YUV420UVC_V1U1V0U0* *G2D_FORMAT_YUV420UVC_U1V1U0V0* *G2D_FORMAT_YUV420_PLANAR* *G2D_FORMAT_YUV411UVC_V1U1V0U0* *G2D_FORMAT_YUV411UVC_U1V1U0V0* *G2D_FORMAT_YUV411_PLANAR* *G2D_FORMAT_Y8 = 0x30* / YUV 10bit format */ G2D_FORMAT_YVU10_P010 G2D_FORMAT_YVU10_P210 G2D_FORMAT_YVU10_444	G2D_FORMAT_ABGR1555 G2D_FORMAT_RGBA5551 G2D_FORMAT_BGRA5551 G2D_FORMAT_ARGB2101010 G2D_FORMAT_ABGR2101010 G2D_FORMAT_RGBA1010102 G2D_FORMAT_BGRA1010102 /* invailed for UI channel / *G2D_FORMAT_IYUV422_V0Y1U0Y0* *G2D_FORMAT_IYUV422_Y1V0Y0U0* *G2D_FORMAT_IYUV422_U0Y1V0Y0* *G2D_FORMAT_IYUV422_Y1U0Y0V0* *G2D_FORMAT_YUV422UVC_V1U1V0U0* *G2D_FORMAT_YUV422UVC_U1V1U0V0* *G2D_FORMAT_YUV422_PLANAR* *G2D_FORMAT_YUV420UVC_V1U1V0U0* *G2D_FORMAT_YUV420UVC_U1V1U0V0* *G2D_FORMAT_YUV420_PLANAR* *G2D_FORMAT_YUV411UVC_V1U1V0U0* *G2D_FORMAT_YUV411UVC_U1V1U0V0* *G2D_FORMAT_YUV411_PLANAR* *G2D_FORMAT_Y8 = 0x30* / YUV 10bit format */ G2D_FORMAT_YVU10_P010 G2D_FORMAT_YVU10_P210 G2D_FORMAT_YVU10_444

图 2-6: pixel_format3

2.4 限制条件

2.4.1 地址对齐问题

- Rotate/Mirror 输入输出地址必须保证 4 bytes 对齐，其他功能无限制。

2.4.2 宽度对齐问题

- Crop/Stitching, Scaler/Down_Sampling, PixelFormat_Conversion, ColorSpace_Conversion, Point/Line Drawing, Rect_ColorFilling 功能，对 RGB 格式输入输出没有限制，但 YUV420 宽高都需要 2 bytes 对齐，YUV422 宽需要 2 bytes 对齐，YUV411 宽需要 4 bytes 对齐。
- Rotate/Mirror 输入输出的宽度要 8 bytes 对齐，a40i 需要满足 64 bytes 对齐。注意如果是 yuv 采样格式，uv plane 的宽度也需要满足对齐规则，比如 YUV420 需要宽高都 16 bytes 对齐，YUV422 宽 16 bytes 对齐。

2.4.3 最小最大宽高问题

最小宽高：宽度上，ARGB8888 格式需要大于 2，RGB888 格式需要大于 3，RGB565 格式需要大于 4。高度上，YUV420 格式需要大于 2。

最大宽高：rotate 功能支持最大 4k 输入输出，其他功能如果不涉及到裁剪和抽样，最大宽高为 2048*2048。如涉及到裁剪或抽样，最大可以达到 8k 输入。注意抽样仅支持对输入 buffer 取数时操作。

2.4.4 输出格式显示

yuv 格式，做旋转时，输出一律是 yuv420，比如 I422 会被强制转换成 I420，NV16 会被强制转换成 NV12。

2.5 模块功能概述

2.5.1 旋转和镜像 (Rotate/Mirror)

旋转镜像主要是实现如下 Horizontal-Flip、Vertical-Flip、Rotate0°、Rotate90°、Rotate180°、Rotate270°、Rotate90°+Horizontal-Flip、Rotate90°+Vertical-Flip 共 8 种操作。

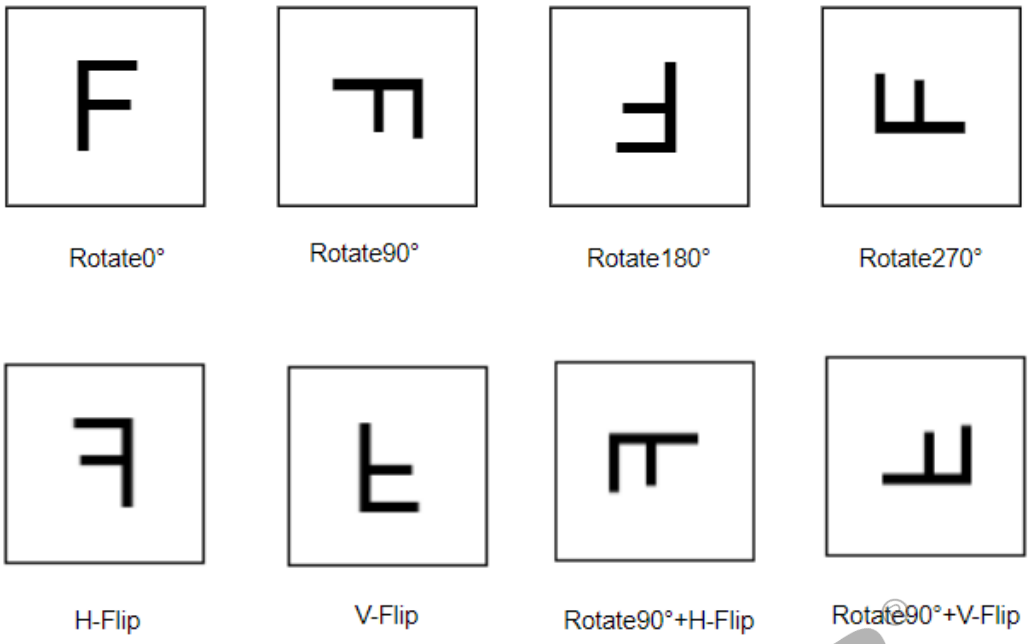

图 2-7: rotate

2.5.2 画点/画线/矩形填充 (Point/Line Drawing/ Rect_ColorFilling)

填充矩形区域功能可以实现对某块区域进行预订的颜色值填充，如下图就填充了 0xFF0080FF 的 ARGB 值，该功能还可以通过设定数据区域大小实现画点和直线，同时也可以通过设定 flag 实现一种填充颜色和目标做 alpha 运算。

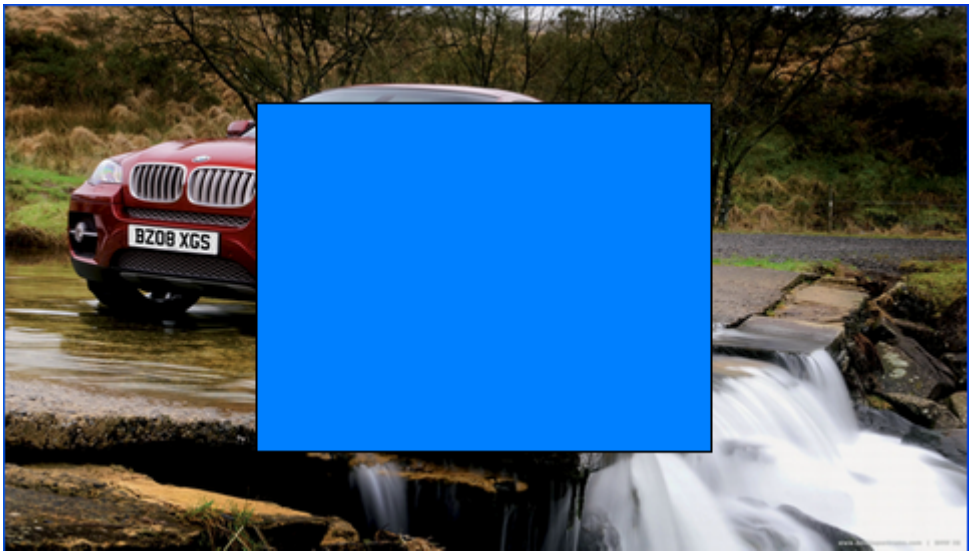

图 2-8: fill rectangle

2.5.3 图层混合 (Alpha Blending)

不同的图层之间可以做 alpha blending。Alpha 分为 pixel alpha、plane alpha、multi alpha 三种：

pixel alpha 意为每个像素自带有一个专属 alpha 值；

plane alpha 则是一个图层中所有像素共用一个 globe alpha 值；

multi alpha 则每个像素在代入 alpha 运算时的值为 globe alpha*pixel alpha

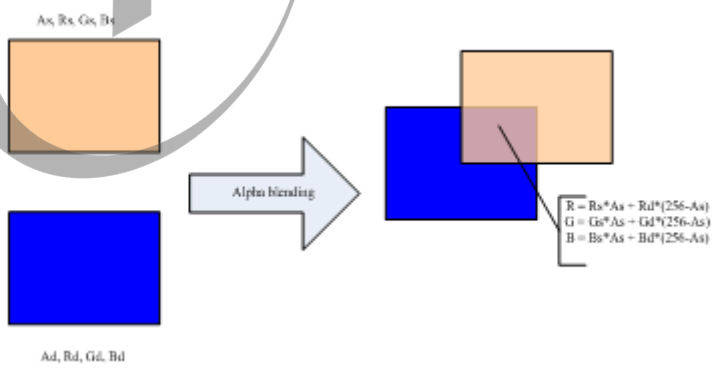

图 2-9: alpha blending 1

2.5.4 色键 (Colorkey)

Colorkey 技术是作用在两个图像叠加混合的时候，对特殊色做特殊过滤。符合条件的区域叫 match 区，在 match 区就全部使用另外一个图层的颜色值；不符合条件的区域就是非 match 区，非 match 区就是走普通的 alpha 混合。Alpha 值越大就是越不透明。

不同 image 之间可以做 colorkey 效果：

- 左图中 destination 中 match 部分（橙色五角星部分），被选择透过，显示为 source 与 destination 做 alpha blending 后的效果图。
- 右图中 source 中 match 部分（深红色五角星部分），被选择透过，直接显示 destination 与 source 做 alpha blending 后的效果图。

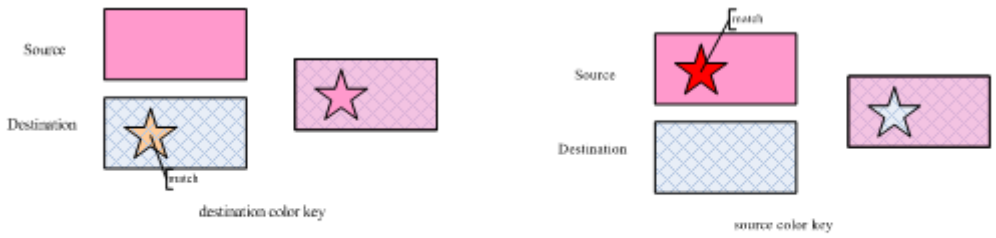

图 2-10: colorkey

2.5.5 缩放 (Scaler/Down_Sampling)

缩放主要是把 source 按照 destination 的 size 进行缩放，并最终与 destination 做 alpha blending、colorkey 等运算或直接旋转镜像后拷贝到目标，此接口在 1.0 版本上使用可以旋转和缩放一起用，但是 2.0 版本以后，缩放和旋转不可以同时操作。

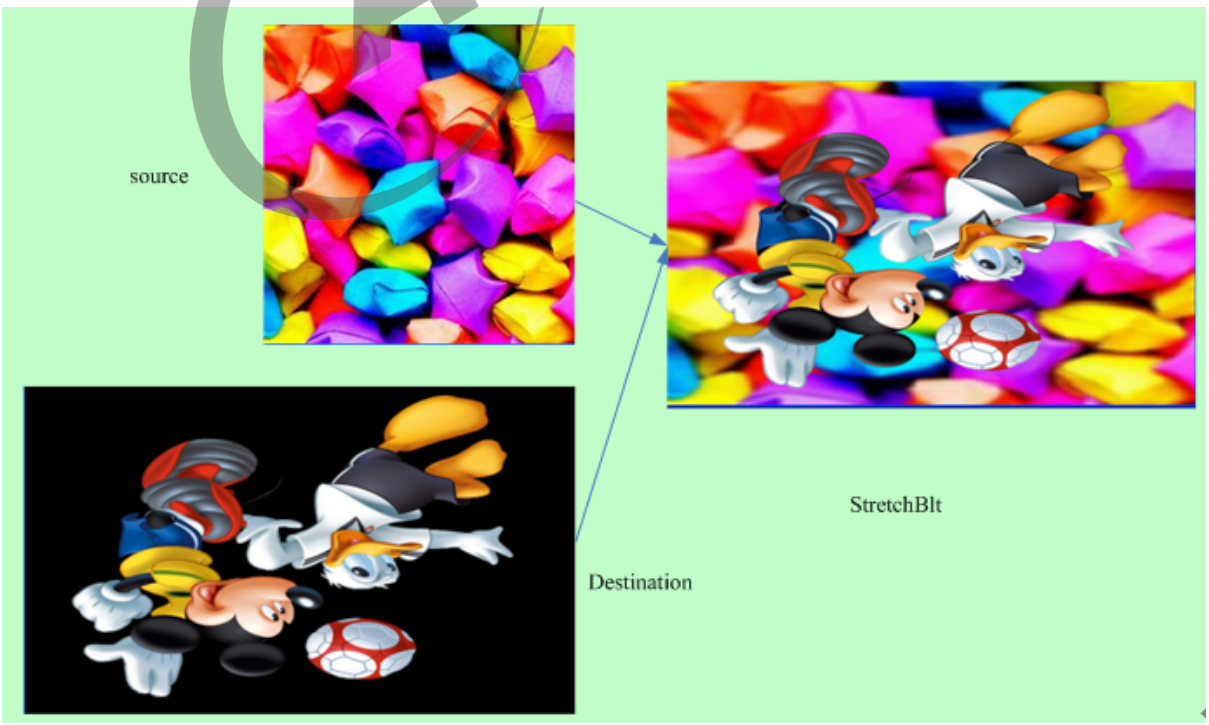

图 2-11: scale and alpha blending

2.5.6 二元光栅操作 (Rop2)

二元光栅操作（Binary Raster Operations）是图形处理中常见的一种光栅操作方式，使用两个输入值进行操作。在二元光栅操作中，通常表示为[src] op [dst] -> [dst]，其中包括源图像（src）、目标图像（dst）和结果保存在目标图像中。

二元光栅操作的基本原理是通过对源图像中的像素与目标图像中对应位置的像素进行某种操作来生成最终的输出。这种操作可以是简单的像素值覆盖、逻辑运算、位运算等，用于实现各种图形处理效果和算法。

G2D 目前支持的二元光栅操作和对应操作码见下面的表格

操作码	含义
G2D_BLT_BLACKNESS	dst = blackness
G2D_BLT_NOTMERGEPEN	dst = ~(dst + src)
G2D_BLT_MASKNOTPEN	dst = ~src & dst
G2D_BLT_NOTCOPYPEN	dst = ~src
G2D_BLT_MASKPENNOT	dst = src & ~dst
G2D_BLT_NOT	dst = ~dst
G2D_BLT_XORPEN	dst = src ^ dst
G2D_BLT_NOTMASKPEN	dst = ~(src & dst)
G2D_BLT_MASKPEN	dst = src & dst
G2D_BLT_NOTXORPEN	dst = ~(src ^ dst)
G2D_BLT_NOP	dst = dst
G2D_BLT_MERGENOTPEN	dst = ~src + dst
G2D_BLT_COPYPEN	dst = src
G2D_BLT_MERGEPENNOT	dst = src + ~dst
G2D_BLT_MERGEPEN	dst = src + dst
G2D_BLT_WHITENESS	dst = whiteness

2.5.7 三元光栅操作 (Rop3)

三元光栅操作（Ternary Raster Operations）是图形处理中使用三个输入值进行操作的一种光栅操作方式。在三元光栅操作中，通常表示为[src0] op [src1] op [src2] -> [dst]，其中包括两个源图像（src0和src1）、一个额外参数（src2），以及目标图像（dst）来保存操作结果。

三元光栅操作通常用于更复杂的图形处理任务，需要对多个输入进行组合、运算或控制的情况。这种操作方式可以实现更加灵活和多样化的图形处理效果，扩展了图形处理的功能和应用范围。

一些常见的三元光栅操作包括：

1. 混合模式：通过将两个源图像按照第三个参数指定的比例混合，实现不同程度的图像叠加效

果。

2. 遮罩操作：使用第三个参数作为掩码（mask），指定哪些像素需要参与操作，实现局部区域的处理。
3. 调色板操作：利用第三个参数作为调色板索引，实现颜色映射或特定颜色的替换。

下图展示了三元光栅操作中的遮罩操作，从左到右，从上到下，依次为 src0, src1, src2, dst。

图 2-12: maskblt rop3

G2D 支持的三元光栅操作和对应操作码见下面的表格

操作码	含义
G2D_ROP3_BLACKNESS	dst = BLACK
G2D_ROP3_NOTSRCERASE	dst = (NOT src) AND (NOT dst)
G2D_ROP3_NOTSRCCOPY	dst = (NOT src)
G2D_ROP3_SRCERASE	dst = src AND (NOT dst)
G2D_ROP3_DSTINVERT	dst = (NOT dst)
G2D_ROP3_PATINVERT	dst = pattern XOR dst
G2D_ROP3_SRCINVERT	dst = src XOR dst
G2D_ROP3_SRCAND	dst = src AND dst
G2D_ROP3_MERGEPAINT	dst = (NOT src) OR dst
G2D_ROP3_MERGECOPY	dst = (src AND pattern)
G2D_ROP3_SRCCOPY	dst = src
G2D_ROP3_SRCPAINT	dst = src OR dst
G2D_ROP3_PATCOPY	dst = pattern

操作码	含义
G2D_ROP3_PATPAINT	dst = (~src OR pattern) OR DST
G2D_ROP3_WHITENESS	dst = WHITE

3 3. 模块配置介绍

3.1 驱动框架

G2D 代码框架如下图所示：

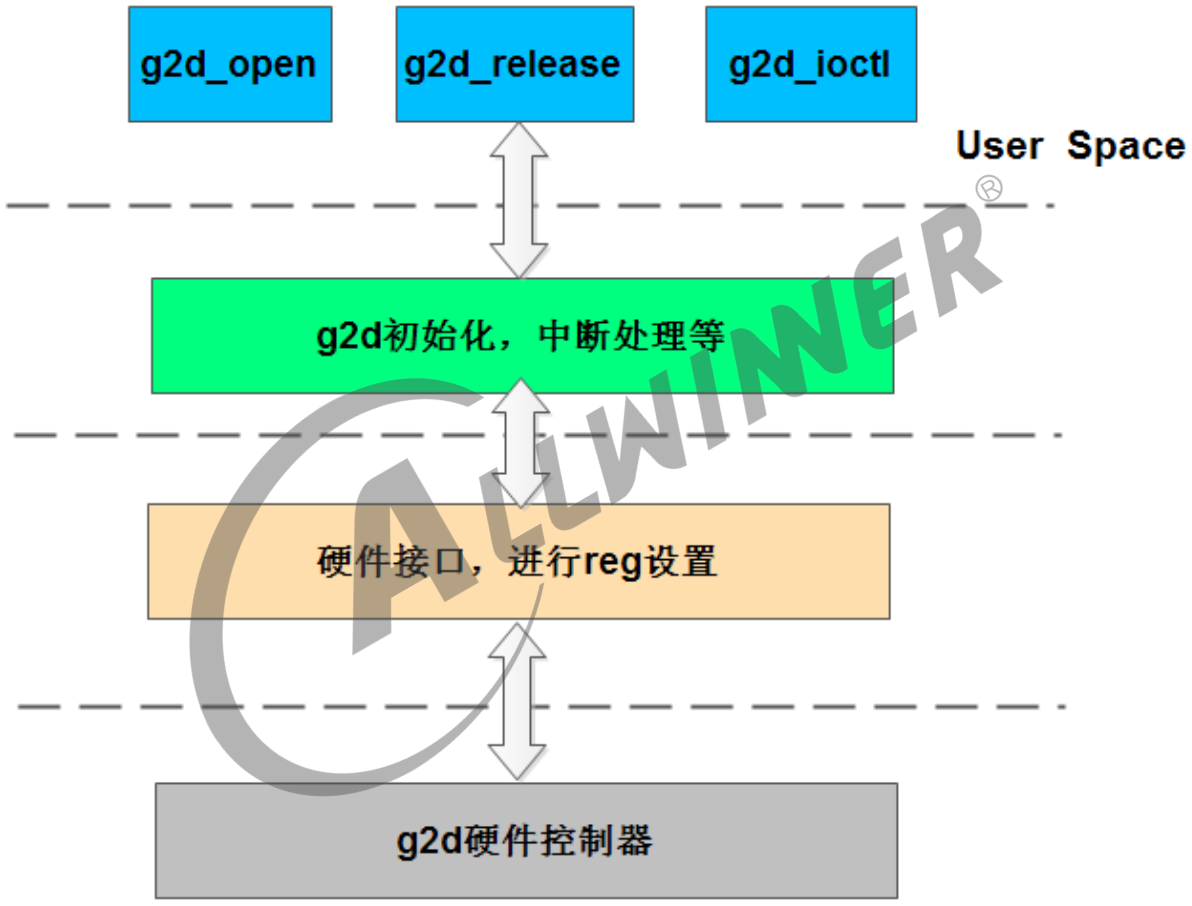

图 3-1: G2D 代码框架图

3.2 驱动版本

方案是否采用 BSP 独立仓库	g2d 源码位置
是	bsp/drivers/g2d
否	kernel/drivers/char/sunxi_g2d

G2D 驱动源码目前一共有两套，分别是 g2d_legacy 和 g2d_rcq。详情见下面表格。保留 g2d_legacy 主要为了支持之前的方案。推荐使用 g2d_rcq，g2d_rcq 兼容 g2d_legacy 的接口。

g2d 驱动版本	说明	编译配置
g2d_legacy	此源码主要为了支持过去的 MixerProcessor (G2D100) 方案	CONFIG_AW_G2D 和 CONFIG_G2D_LEGACY
g2d_rcq	除使用 MixerProcessor (G2D100) 外，其他建议升级驱动为 g2d_rcq，后续主要维护该版本	CONFIG_AW_G2D, CONFIG_G2D_RCQ, CONFIG_G2D_ROTATE, CONFIG_G2D_MIXER（后两者开关依赖于设计指标）

3.3 Device Tree 配置说明

```
g2d:g2d@01480000{
  compatible = "allwinner,sunxi-g2d";
  reg = <0x0 0x01480000 0x0 0xbffff>;
  interrupts = <GIC_SPI 21 0x0104>;
  clocks = <&clk_g2d>;
  iommus = <&mmu_aw 5 1>;
  status = "okay";
 };
```

注：上述配置仅供参考，具体配置跟芯片有关。

4 4. 功能使用 Demo

注：以下代码仅为伪代码。如不特别指出，不适用于 G2D100

4.1 旋转和镜像 (Rotate/Mirror)

4.1.1 关键参数

```
typedef struct {
    g2d_blt_flags_h flag_h;
    g2d_image_enh src_image_h;
    g2d_image_enh dst_image_h;
} g2d_blt_h;

typedef enum {
    G2D_BLT_NONE_H = 0x0,   /* single source operation */
    /* rop2 */
    /* Fill the target rectangular area with the color associated with index 0 of the physical color palette.
     * For the default physical color palette, this color is black. */
    G2D_BLT_BLACKNESS,  /* blackness */
    G2D_BLT_NOTMERGEPEN,    /* dst = ~(dst + src) */
    G2D_BLT_MASKNOTPEN, /* dst = ~src & dst */
    G2D_BLT_NOTCOPYPEN, /* dst = ~src */
    G2D_BLT_MASKPENNOT, /* dst = src & ~dst */
    G2D_BLT_NOT,    /* dst = ~dst */
    G2D_BLT_XORPEN, /* dst = src ^ dst */
    G2D_BLT_NOTMASKPEN, /* dst = ~(src & dst) */
    G2D_BLT_MASKPEN,    /* dst = src & dst */
    G2D_BLT_NOTXORPEN,  /* dst = ~(src ^ dst) */
    G2D_BLT_NOP,    /* dst = dst */
    G2D_BLT_MERGENOTPEN,    /* dst = ~src + dst */
    G2D_BLT_COPYPEN,    /* dst = src */
    G2D_BLT_MERGEPENNOT,    /* dst = src + ~dst */
    G2D_BLT_MERGEPEN,   /* dst = src + dst */
    G2D_BLT_WHITENESS,  /* whiteness */

    /* Fill the target rectangular area with the color associated with index 0 of the physical color palette.
     * For the default physical color palette, this color is white. */
    G2D_BLT_EXTRACT = 0x000000ff,

    G2D_ROT_90  = 0x00000100,
    G2D_ROT_180 = 0x00000200,
    G2D_ROT_270 = 0x00000300,
    G2D_ROT_0   = 0x00000400,
    G2D_ROT_H = 0x00001000,
    G2D_ROT_V = 0x00002000,
    G2D_ROT_90_H  = 0x00001100,
    G2D_ROT_90_V  = 0x00002100,
```

```
/*  G2D_SM_TDLR_1 =   0x10000000, */
   G2D_SM_DTLR_1 = 0x10000000,
/*  G2D_SM_TDRL_1 =   0x20000000, */
/*  G2D_SM_DTRL_1 =   0x30000000, */
} g2d_blt_flags_h;

/* image struct */
typedef struct {
   int     bbuff; /* determine color_filling or image buffer */
   __u32       color; /* color_filling */
   g2d_fmt_enh  format;   /* pixel format */
   __u32       laddr[3];  /* low address for three planes */
   __u32       haddr[3];  /* high address for three planes */
   __u32       width; /* image window width */
   __u32       height;    /* image window height */
   __u32       align[3];  /* image windows alignment, y/u/v planes*/

   g2d_rect    clip_rect; /* roi */
   g2d_size    resize;    /* image size after down_sampling */
   g2d_coor     coor;  /* top-left corner coordinates placed in the dst_image window
                * after src_image has been processed */

   g2d_color_gmt    gamut; /* color space */
/*  __u32       gamut; */
   int     bpremul;   /* alpha premutiled or not */
   __u8       alpha;
   g2d_alpha_mode_enh mode;    /* alpha mode */
   int     fd;
   __u32 use_phy_addr; /* use phy_addr or not */
   enum color_range color_range;
} g2d_image_enh;
```

4.1.2 使用 demo

注：目前支持的方向请参考模块功能概述里展示的 8 种旋转和镜像

```
/* 将格式为argb8888格式，分辨率为1920x1080分辨率图像旋转90度 */
int ret;
int g2d_fd;
g2d_blt_h blit;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blit, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_ROT_90;

#ifdef USE_PHY_ADDR
blit.src_image_h.use_phy_addr = 1;
blit.src_image_h.laddr[0] = src_image_addr[0];
#elif USE_DMA_BUFFERR_HEAP
blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd;
```

```
#else
...
#endif
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 1920;
blit.src_image_h.height = 1080;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1920;
blit.src_image_h.clip_rect.h = 1080;

#ifdef USE_PHY_ADDR
blit.dst_image_h.use_phy_addr = 1;
blit.dst_image_h.laddr[0] = dst_image_addr[0];
#elif USE_DMA_BUFFERR_HEAP
blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
#else
...
#endif
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 1080;
blit.dst_image_h.height = 1920;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1080;
blit.dst_image_h.clip_rect.h = 1920;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.2 图像裁剪和拼接 (Crop/Stiching)

注：各个硬件版本都有裁剪和拼接功能，但性能不同。如果使用的模块带有 rcq 更新，建议使用带 rcq 更新的接口。以 G2D101r1p0 系列为例，分为 G2D101r1p0 和 G2D101r1p0 (Rotate-Only)。后者的性能在需要大量执行 crop/stiching 时因为每个任务都要产生和等待 cpu 响应中断，所以远远不如前者。下面的代码展示了两种调用。

4.2.1 关键参数

```
/* 裁剪和拼接功能依赖于`overlay`取数时候和回写时候的参数设置，重点和输入输出图像的`width, height, clip_rect.x,
      clip_rect.y, clip_rect_w, clip_rect.h`参数有关,可以把`width`和`height`理解为图像处理领域中整个图像区域的`
      window_size`，`clip_rect`等价于ROI */

/* image struct */
typedef struct {
  int    bbuff;
```

```
    __u32      color;
    g2d_fmt_enh  format;   /* pixel format */
    __u32      laddr[3];  /* low address for three planes */
    __u32      haddr[3];  /* high address for three planes */
    __u32      width; /* image window width */
    __u32      height;   /* image window height */
    __u32      align[3];  /* image windows alignment, y/u/v planes*/

    g2d_rect    clip_rect; /* roi */
    g2d_size    resize;   /* image size after down_sampling */
    g2d_coor    coor;  /* top-left corner coordinates placed in the dst_image window
                * after src_image has been processed */

    g2d_color_gmt   gamut; /* color space */
/*  __u32      gamut; */
    int     bpremul;   /* alpha premutiled or not */
    __u8       alpha;
    g2d_alpha_mode_enh mode;   /* alpha mode */
    int     fd;
    __u32 use_phy_addr; /* use phy_addr or not */
    enum color_range color_range;
} g2d_image_enh;
```

4.2.2 使用 demo

注：由于裁剪拼接任务往往常见于将一张大图切割为许多张小图，将多张小图拼接为一张大图，且每帧任务往往参数基本相同，故而如果使用的模块带 rcq 更新，推荐利用 rcq 特性在批处理工作模式下执行此类任务, 关于模块是否支持 rcq 请查询设计指标。

4.2.2.1 rcq

```
/* 将格式为argb8888格式，分辨率为1024x1024分辨率图像src_image_fd中ROI区域为(0，0，512，512)、(0，512，512，
      512)、(512，0，512，512)、(512，512，512，512)的区域裁剪下来，输出到四个512*512分辨率的图像dst_image_fd
      [4]上去
 * 以使用dmabufferheap内存句柄为例 */
#define FRAME_TO_BE_PROCESS 4
int g2d_fd;
unsigned long arg[2];
struct mixer_para info[FRAME_TO_BE_PROCESS];
int task_id
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

info[0].op_flag = OP_BITBLT;
info[0].flag_h = G2D_BLT_NONE_H;
info[0].src_image_h.use_phy_addr = 0;
info[0].src_image_h.fd = src_image_fd;
```

```
info[0].src_image_h.bbuff = 1;
info[0].src_image_h.mode = G2D_PIXEL_ALPHA;
info[0].src_image_h.alpha = 0xff;
info[0].src_image_h.format = G2D_FORMAT_ARGB8888;
info[0].src_image_h.width = 1024;
info[0].src_image_h.height = 1024;
info[0].src_image_h.clip_rect.x = 0;
info[0].src_image_h.clip_rect.y = 0;
info[0].src_image_h.clip_rect.w = 512;
info[0].src_image_h.clip_rect.h = 512;
info[0].dst_image_h.use_phy_addr = 0;
info[0].dst_image_h.fd = dst_image_fd[0];
info[0].dst_image_h.bbuff = 1;
info[0].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[0].dst_image_h.alpha = 0xff;
info[0].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[0].dst_image_h.width = 512;
info[0].dst_image_h.height = 512;
info[0].dst_image_h.clip_rect.x = 0;
info[0].dst_image_h.clip_rect.y = 0;
info[0].dst_image_h.clip_rect.w = 512;
info[0].dst_image_h.clip_rect.h = 512;

info[1].op_flag = OP_BITBLT;
info[1].flag_h = G2D_BLT_NONE_H;
info[1].src_image_h.use_phy_addr = 0;
info[1].src_image_h.fd = src_image_fd;
info[1].src_image_h.bbuff = 1;
info[1].src_image_h.mode = G2D_PIXEL_ALPHA;
info[1].src_image_h.alpha = 0xff;
info[1].src_image_h.format = G2D_FORMAT_ARGB8888;
info[1].src_image_h.width = 1024;
info[1].src_image_h.height = 1024;
info[1].src_image_h.clip_rect.x = 0;
info[1].src_image_h.clip_rect.y = 512;
info[1].src_image_h.clip_rect.w = 512;
info[1].src_image_h.clip_rect.h = 512;
info[1].dst_image_h.use_phy_addr = 0;
info[1].dst_image_h.fd = dst_image_fd[1];
info[1].dst_image_h.bbuff = 1;
info[1].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[1].dst_image_h.alpha = 0xff;
info[1].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[1].dst_image_h.width = 512;
info[1].dst_image_h.height = 512;
info[1].dst_image_h.clip_rect.x = 0;
info[1].dst_image_h.clip_rect.y = 0;
info[1].dst_image_h.clip_rect.w = 512;
info[1].dst_image_h.clip_rect.h = 512;

info[2].op_flag = OP_BITBLT;
info[2].flag_h = G2D_BLT_NONE_H;
info[2].src_image_h.use_phy_addr = 0;
info[2].src_image_h.fd = src_image_fd;
info[2].src_image_h.bbuff = 1;
info[2].src_image_h.mode = G2D_PIXEL_ALPHA;
info[2].src_image_h.alpha = 0xff;
info[2].src_image_h.format = G2D_FORMAT_ARGB8888;
info[2].src_image_h.width = 1024;
info[2].src_image_h.height = 1024;
```

```
info[2].src_image_h.clip_rect.x = 512;
info[2].src_image_h.clip_rect.y = 0;
info[2].src_image_h.clip_rect.w = 512;
info[2].src_image_h.clip_rect.h = 512;
info[2].dst_image_h.use_phy_addr = 0;
info[2].dst_image_h.fd = dst_image_fd[2];
info[2].dst_image_h.bbuff = 1;
info[2].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[2].dst_image_h.alpha = 0xff;
info[2].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[2].dst_image_h.width = 512;
info[2].dst_image_h.height = 512;
info[2].dst_image_h.clip_rect.x = 0;
info[2].dst_image_h.clip_rect.y = 0;
info[2].dst_image_h.clip_rect.w = 512;
info[2].dst_image_h.clip_rect.h = 512;

info[3].op_flag = OP_BITBLT;
info[3].flag_h = G2D_BLT_NONE_H;
info[3].src_image_h.use_phy_addr = 0;
info[3].src_image_h.fd = src_image_fd;
info[3].src_image_h.bbuff = 1;
info[3].src_image_h.mode = G2D_PIXEL_ALPHA;
info[3].src_image_h.alpha = 0xff;
info[3].src_image_h.format = G2D_FORMAT_ARGB8888;
info[3].src_image_h.width = 1024;
info[3].src_image_h.height = 1024;
info[3].src_image_h.clip_rect.x = 512;
info[3].src_image_h.clip_rect.y = 512;
info[3].src_image_h.clip_rect.w = 512;
info[3].src_image_h.clip_rect.h = 512;
info[3].dst_image_h.use_phy_addr = 0;
info[3].dst_image_h.fd = dst_image_fd[3];
info[3].dst_image_h.bbuff = 1;
info[3].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[3].dst_image_h.alpha = 0xff;
info[3].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[3].dst_image_h.width = 512;
info[3].dst_image_h.height = 512;
info[3].dst_image_h.clip_rect.x = 0;
info[3].dst_image_h.clip_rect.y = 0;
info[3].dst_image_h.clip_rect.w = 512;
info[3].dst_image_h.clip_rect.h = 512;

arg[0] = (unsigned long) info;
arg[1] =  FRAME_TO_BE_PROCESS;
task_id = ioctl(g2d_fd, G2D_CMD_CREATE_TASK ,(unsigned long)(arg))
if (task_id < 0)
   printf("failed to create g2d task\n")
memset(arg, 0, sizeof(arg));
arg[0] = task_id;
ret = ioctl(g2d_fd, G2D_CMD_TASK_APPLY ,(unsigned long)(arg))
ret = ioctl(g2d_fd, G2D_CMD_TASK_DESTROY ,(unsigned long)(arg))
```

```
/* 将4个格式为argb8888格式，分辨率为512x512的图像src_image_fd[4]拼接为一个1024x1024分辨率的图像dst_image_fd
 * 以使用dmabufferheap内存句柄为例 */
#define FRAME_TO_BE_PROCESS 4
int g2d_fd;
unsigned long arg[2];
struct mixer_para info[FRAME_TO_BE_PROCESS];
```

```
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

info[0].op_flag = OP_BITBLT;
info[0].flag_h = G2D_BLT_NONE_H;
info[0].src_image_h.use_phy_addr = 0;
info[0].src_image_h.fd = src_image_fd[0];
info[0].src_image_h.bbuff = 1;
info[0].src_image_h.mode = G2D_PIXEL_ALPHA;
info[0].src_image_h.alpha = 0xff;
info[0].src_image_h.format = G2D_FORMAT_ARGB8888;
info[0].src_image_h.width = 0;
info[0].src_image_h.height = 0;
info[0].src_image_h.clip_rect.x = 0;
info[0].src_image_h.clip_rect.y = 0;
info[0].src_image_h.clip_rect.w = 512;
info[0].src_image_h.clip_rect.h = 512;
info[0].dst_image_h.use_phy_addr = 0;
info[0].dst_image_h.fd = dst_image_fd;
info[0].dst_image_h.bbuff = 1;
info[0].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[0].dst_image_h.alpha = 0xff;
info[0].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[0].dst_image_h.width = 1024;
info[0].dst_image_h.height = 1024;
info[0].dst_image_h.clip_rect.x = 0;
info[0].dst_image_h.clip_rect.y = 0;
info[0].dst_image_h.clip_rect.w = 512;
info[0].dst_image_h.clip_rect.h = 512;

info[1].op_flag = OP_BITBLT;
info[1].flag_h = G2D_BLT_NONE_H;
info[1].src_image_h.use_phy_addr = 0;
info[1].src_image_h.fd = src_image_fd[1];
info[1].src_image_h.bbuff = 1;
info[1].src_image_h.mode = G2D_PIXEL_ALPHA;
info[1].src_image_h.alpha = 0xff;
info[1].src_image_h.format = G2D_FORMAT_ARGB8888;
info[1].src_image_h.width = 512;
info[1].src_image_h.height = 512;
info[1].src_image_h.clip_rect.x = 0;
info[1].src_image_h.clip_rect.y = 0;
info[1].src_image_h.clip_rect.w = 512;
info[1].src_image_h.clip_rect.h = 512;
info[1].dst_image_h.use_phy_addr = 0;
info[1].dst_image_h.fd = dst_image_fd;
info[1].dst_image_h.bbuff = 1;
info[1].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[1].dst_image_h.alpha = 0xff;
info[1].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[1].dst_image_h.width = 1024;
info[1].dst_image_h.height = 1024;
info[1].dst_image_h.clip_rect.x = 0;
info[1].dst_image_h.clip_rect.y = 512;
```

```
info[1].dst_image_h.clip_rect.w = 512;
info[1].dst_image_h.clip_rect.h = 512;

info[2].op_flag = OP_BITBLT;
info[2].flag_h = G2D_BLT_NONE_H;
info[2].src_image_h.use_phy_addr = 0;
info[2].src_image_h.fd = src_image_fd[2];
info[2].src_image_h.bbuff = 1;
info[2].src_image_h.mode = G2D_PIXEL_ALPHA;
info[2].src_image_h.alpha = 0xff;
info[2].src_image_h.format = G2D_FORMAT_ARGB8888;
info[2].src_image_h.width = 512;
info[2].src_image_h.height = 512;
info[2].src_image_h.clip_rect.x = 0;
info[2].src_image_h.clip_rect.y = 0;
info[2].src_image_h.clip_rect.w = 512;
info[2].src_image_h.clip_rect.h = 512;
info[2].dst_image_h.use_phy_addr = 0;
info[2].dst_image_h.fd = dst_image_fd;
info[2].dst_image_h.bbuff = 1;
info[2].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[2].dst_image_h.alpha = 0xff;
info[2].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[2].dst_image_h.width = 1024;
info[2].dst_image_h.height = 1024;
info[2].dst_image_h.clip_rect.x = 512;
info[2].dst_image_h.clip_rect.y = 0;
info[2].dst_image_h.clip_rect.w = 512;
info[2].dst_image_h.clip_rect.h = 512;

info[3].op_flag = OP_BITBLT;
info[3].flag_h = G2D_BLT_NONE_H;
info[3].src_image_h.use_phy_addr = 0;
info[3].src_image_h.fd = src_image_fd[3];
info[3].src_image_h.bbuff = 1;
info[3].src_image_h.mode = G2D_PIXEL_ALPHA;
info[3].src_image_h.alpha = 0xff;
info[3].src_image_h.format = G2D_FORMAT_ARGB8888;
info[3].src_image_h.width = 512;
info[3].src_image_h.height = 512;
info[3].src_image_h.clip_rect.x = 0;
info[3].src_image_h.clip_rect.y = 0;
info[3].src_image_h.clip_rect.w = 512;
info[3].src_image_h.clip_rect.h = 512;
info[3].dst_image_h.use_phy_addr = 0;
info[3].dst_image_h.fd = dst_image_fd;
info[3].dst_image_h.bbuff = 1;
info[3].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[3].dst_image_h.alpha = 0xff;
info[3].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[3].dst_image_h.width = 1024;
info[3].dst_image_h.height = 1024;
info[3].dst_image_h.clip_rect.x = 512;
info[3].dst_image_h.clip_rect.y = 512;
info[3].dst_image_h.clip_rect.w = 512;
info[3].dst_image_h.clip_rect.h = 512;

arg[0] = (unsigned long) info;
arg[1] =  FRAME_TO_BE_PROCESS;
task_id = ioctl(g2d_fd, G2D_CMD_CREATE_TASK ,(unsigned long)(arg))
```

```
if (task_id < 0)
   printf("failed to create g2d task\n")
memset(arg, 0, sizeof(arg));
arg[0] = task_id;
ret = ioctl(g2d_fd, G2D_CMD_TASK_APPLY ,(unsigned long)(arg))
ret = ioctl(g2d_fd, G2D_CMD_TASK_DESTROY ,(unsigned long)(arg))
```

4.2.2.2 非 rcq

```
/* 将格式为argb8888格式，分辨率为1024x1024分辨率图像中ROI区域为(0，512，512，512)的区域裁剪下来，输出到512
       *512分辨率的图像上去
 * 以使用dmabufferheap内存句柄为例 */
int g2d_fd;
g2d_blt_h blt;
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_ROT_0;

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd[0];
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 1024;
blit.src_image_h.height = 1024;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 512;
blit.src_image_h.clip_rect.w = 512;
blit.src_image_h.clip_rect.h = 512;

blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 512;
blit.dst_image_h.height = 512;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 512;
blit.dst_image_h.clip_rect.h = 512;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

```
/* 将4个格式为argb8888格式，分辨率为512x512的图像拼接为一个1024x1024分辨率的图像
 * 以使用dmabufferheap内存句柄为例 */
int ret;
int g2d_fd;
```

```
g2d_blt_h blt;
unsigned int src_image_fd[4];
unsigned int dst_image_fd;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_ROT_0;

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd[0];
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 512;
blit.src_image_h.height = 512;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 512;
blit.src_image_h.clip_rect.h = 512;

blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 1024;
blit.dst_image_h.height = 1024;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 512;
blit.dst_image_h.clip_rect.h = 512;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))

blit.src_image_h.fd = src_image_fd[1];
blit.dst_image_h.clip_rect.x = 512;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 512;
blit.dst_image_h.clip_rect.h = 512;
ret |= ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))

blit.src_image_h.fd = src_image_fd[1];
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 512;
blit.dst_image_h.clip_rect.w = 512;
blit.dst_image_h.clip_rect.h = 512;
ret |= ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))

blit.src_image_h.fd = src_image_fd[1];
blit.dst_image_h.clip_rect.x = 512;
blit.dst_image_h.clip_rect.y = 512;
blit.dst_image_h.clip_rect.w = 512;
blit.dst_image_h.clip_rect.h = 512;
```

```
ret |= ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.3 图像缩放或下采样 (Scaler/Down_Sampling)

注：Scaler 和 Down_Sampling 的不同之处在于前者既可以放大又可以缩小，同时可以实现无极缩放；而后者仅仅可以用来将图像按照按照 1/2 的整数次幂进行抽样，从而实现缩小。下采样和缩放可以同时使用，比如可以先将 8k 图像先下采样到宽度为 2048，再利用 Scaler 缩小到 1080p。

4.3.1 关键参数

```
/* Down_Sampling主要和 resize 参数有关，Scaler主要和参数 clip_rect 有关 */
/* image struct */
typedef struct {
    int     bbuff;
    __u32       color;
    g2d_fmt_enh  format;    /* pixel format */
    __u32       laddr[3];  /* low address for three planes */
    __u32       haddr[3];  /* high address for three planes */
    __u32       width; /* image window width */
    __u32       height;    /* image window height */
    __u32       align[3];  /* image windows alignment, y/u/v planes*/

    g2d_rect    clip_rect; /* roi */
    g2d_size    resize;    /* image size after down_sampling */
    g2d_coor    coor;  /* top-left corner coordinates placed in the dst_image window
                 * after src_image has been processed */

    g2d_color_gmt   gamut; /* color space */
/*  __u32       gamut; */
    int     bpremul;   /* alpha premutiled or not */
    __u8        alpha;
    g2d_alpha_mode_enh mode;    /* alpha mode */
    int     fd;
    __u32 use_phy_addr; /* use phy_addr or not */
    enum color_range color_range;
} g2d_image_enh;
```

4.3.2 使用 demo

```
/* 将格式为argb8888格式，分辨率为8192x8192分辨率的图像下采样到2048x8192，再缩放为1920x1080
 * 以使用dmabufferheap内存句柄为例 */
int g2d_fd;
g2d_blt_h blit;
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
    printf("failed to open g2d device\n");
    return;
```

```
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_BLT_NONE_H;

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd;
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 8192;
blit.src_image_h.height = 8192;
/* 如果不需要下采样，不需要设置resize参数 */
blit.src_image_h.resize.w = 2048;
blit.src_image_h.resize.h = 8192;

blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 2048;
blit.src_image_h.clip_rect.h = 8192;

blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.4 图像格式转换（Format_Conversion）

4.4.1 关键参数

```
/* 格式转换主要和输入输出图像的format参数有关 */
/* image struct */
typedef struct {
  int     bbuff;
  __u32      color;
  g2d_fmt_enh  format;   /* pixel format */
  __u32      laddr[3];  /* low address for three planes */
  __u32      haddr[3];  /* high address for three planes */
  __u32      width; /* image window width */
  __u32      height;   /* image window height */
  __u32      align[3];  /* image windows alignment, y/u/v planes*/

  g2d_rect    clip_rect; /* roi */
  g2d_size    resize;   /* image size after down_sampling */
```

```
    g2d_coor    coor;  /* top-left corner coordinates placed in the dst_image window
                    * after src_image has been processed */

    g2d_color_gmt    gamut; /* color space */
/*  __u32       gamut; */
    int     bpremul;   /* alpha premutiled or not */
    __u8        alpha;
    g2d_alpha_mode_enh mode;    /* alpha mode */
    int     fd;
    __u32 use_phy_addr; /* use phy_addr or not */
    enum color_range color_range;
} g2d_image_enh;
```

4.4.2 使用 demo

```
/* 将格式为argb8888格式，分辨率为1920x1080分辨率的图像转换为nv12格式
 * 以使用dmabufferheap内存句柄为例 */
int g2d_fd;
g2d_blt_h blt;
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
    printf("failed to open g2d device\n");
    return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_BLT_NONE_H;

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd;
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 1920;
blit.src_image_h.height = 1080;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1920;
blit.src_image_h.clip_rect.h = 1080;

blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_YUV420UVC_U1V1U0V0;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.5 图层混合 (Alpha Blending)

4.5.1 关键参数

```
typedef struct {
   g2d_bld_cmd_flag bld_cmd;
   g2d_image_enh dst_image;
   g2d_image_enh src_image[4];/* now only ch0 and ch3 */
   g2d_ck ck_para;
} g2d_bld;       /* blending enhance */
```

4.5.2 使用 demo

```
/* 将两张1280x800的图片进行混合
 * 注：这里的src_image[0]、src_image[1]和destination 相当于struct g2d_bld_cmd_flag 中的source、destination和result
       */
int ret;
g2d_bld blend;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_bld));

blend.bld_cmd = G2D_BLD_COPY;  /* 根据具体想要的混合效果选择,详情见头文件g2d_bld_cmd_flag */

blend.src_image[0].use_phy_addr = 0;
blend.src_image[0].fd = src_image_fd;
blend.src_image[0].bbuff = 1;
blend.src_image[0].mode = G2D_PIXEL_ALPHA;
blend.src_image[0].format = G2D_FORMAT_ARGB8888;
blend.src_image[0].alpha = 128;
blend.src_image[0].width = 1280;
blend.src_image[0].height = 800;
blend.src_image[0].clip_rect.x = 0;
blend.src_image[0].clip_rect.y = 0;
blend.src_image[0].clip_rect.w = 1280;
blend.src_image[0].clip_rect.h = 800;

blend.src_image[1].use_phy_addr = 0;
blend.src_image[1].fd = src1_image_fd;
blend.src_image[1].bbuff = 1;
blend.src_image[1].mode = G2D_PIXEL_ALPHA;
blend.src_image[1].format = G2D_FORMAT_ARGB8888;
blend.src_image[1].alpha = 128;
blend.src_image[1].width = 1280;
blend.src_image[1].height = 800;
blend.src_image[1].clip_rect.x = 0;
blend.src_image[1].clip_rect.y = 0;
blend.src_image[1].clip_rect.w = 1280;
blend.src_image[1].clip_rect.h = 800;
```

```
blend.dst_image.use_phy_addr = 0;
blend.dst_image.fd = dst_image_fd;
blend.dst_image.bbuff = 1;
blend.dst_image.mode = G2D_PIXEL_ALPHA;
blend.dst_image.format = G2D_FORMAT_ARGB8888;
blend.dst_image.alpha = 128;
blend.dst_image.width = 1280;
blend.dst_image.height = 800;
blend.dst_image.clip_rect.x = 0;
blend.dst_image.clip_rect.y = 0;
blend.dst_image.clip_rect.w = 1280;
blend.dst_image.clip_rect.h = 800;

ret = ioctl(g2d_fd, G2D_CMD_BLD_H ,(unsigned long)(&blend))
```

4.6 画点/画线/矩形填充 (Point/Line drawing, Rect_ColorFilling)

4.6.1 关键参数

```
/* 其中g2d_image_enh的color成员用于传递填充的颜色参数，各个分量：A[31:24] R[23:16] G[15:8] B[7:0] */
typedef struct {
   g2d_image_enh dst_image_h;
} g2d_fillrect_h;

typedef struct {
   int     bbuff; /* determine color_filling or image buffer */
   __u32       color; /* color_filling */
   g2d_fmt_enh  format;    /* pixel format */
   __u32       laddr[3];  /* low address for three planes */
   __u32       haddr[3];  /* high address for three planes */
   __u32       width; /* image window width */
   __u32       height;    /* image window height */
   __u32       align[3];  /* image windows alignment, y/u/v planes*/

   g2d_rect    clip_rect; /* roi */
   g2d_size    resize;    /* image size after down_sampling */
   g2d_coor     coor;  /* top-left corner coordinates placed in the dst_image window
                * after src_image has been processed */

   g2d_color_gmt    gamut; /* color space */
/*  __u32       gamut; */
   int     bpremul;   /* alpha premutiled or not */
   __u8       alpha;
   g2d_alpha_mode_enh mode;    /* alpha mode */
   int     fd;
   __u32 use_phy_addr; /* use phy_addr or not */
   enum color_range color_range;
} g2d_image_enh;
```

4.6.2 使用 demo

注：更改下述 demo 的 width, height,clip_rect 即可实现画点画线

```
int g2d_fd;
g2d_blt_h blt;
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_BLT_NONE_H;

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd;
blit.src_image_h.bbuff = 0;
blit.src_image_h.color = 0xffff0000; /* alpha(bit31-bit24) Y/R(bit23-bit16) U/G(bit15-bit8) V/B(bit7-bit0) */
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 1920;
blit.src_image_h.height = 1080;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1920;
blit.src_image_h.clip_rect.h = 1080;

blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 0;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.7 色键 (ColorKey)

4.7.1 关键结构

```
typedef struct {
  int match_rule; /* 当match_rule为假时，Color Min=<Color<=Color Max表示满足匹配条件;当match_rule为真时，Color>
      Color Max or Color <Color Min表示满足匹配条件 */
  __u32 max_color;  /* Color Max */
  __u32 min_color;  /* Color Min */
}g2d_ck;
```

4.7.2 使用 demo

```
/* 将两张1280x800的图片进行混合,同时对src_image[0]设置色键
 * 注：这里的src_image[0]、src_image[1]和destination 相当于struct g2d_bld_cmd_flag 中的source、destination和result
      */
int ret;
g2d_bld blend;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
  printf("failed to open g2d device\n");
  return;
}
memset(&blt, 0, sizeof(g2d_bld));

blend.bld_cmd = G2D_BLD_COPY | G2D_CK_SRC;
blend.ck_para.match_rule = ture;
blend.ck_para.min_color = 35;
blend.ck_para.max_color = 235;

blend.src_image[0].mode = G2D_PIXEL_ALPHA;
blend.src_image[0].format = G2D_FORMAT_ARGB8888;
blend.src_image[0].alpha = 128;
blend.src_image[0].width = 1280;
blend.src_image[0].height = 800;
blend.src_image[0].clip_rect.x = 0;
blend.src_image[0].clip_rect.y = 0;
blend.src_image[0].clip_rect.w = 1280;
blend.src_image[0].clip_rect.h = 800;

blend.src_image[1].mode = G2D_PIXEL_ALPHA;
blend.src_image[1].format = G2D_FORMAT_ARGB8888;
blend.src_image[1].alpha = 128;
blend.src_image[1].width = 1280;
blend.src_image[1].height = 800;
blend.src_image[1].clip_rect.x = 0;
blend.src_image[1].clip_rect.y = 0;
blend.src_image[1].clip_rect.w = 1280;
blend.src_image[1].clip_rect.h = 800;

blend.dst_image.mode = G2D_PIXEL_ALPHA;
blend.dst_image.format = G2D_FORMAT_ARGB8888;
```

```
blend.dst_image.alpha = 128;
blend.dst_image.width = 1280;
blend.dst_image.height = 800;
blend.dst_image.clip_rect.x = 0;
blend.dst_image.clip_rect.y = 0;
blend.dst_image.clip_rect.w = 1280;
blend.dst_image.clip_rect.h = 800;

ret = ioctl(g2d_fd, G2D_CMD_BLD_H ,(unsigned long)(&blend))
```

4.8 二元光栅操作 (Rop2)

4.8.1 关键参数

```
/* 主要跟struct g2d_blt_h的flag_h成员有关，G2D目前支持的ROP2操作见模块功能概述 */
typedef struct {
    g2d_blt_flags_h flag_h;
    g2d_image_enh src_image_h;
    g2d_image_enh dst_image_h;
} g2d_blt_h;

typedef enum {
    G2D_BLT_NONE_H = 0x0,   /* single source operation */
    /* rop2 */
    /* Fill the target rectangular area with the color associated with index 0 of the physical color palette.
     * For the default physical color palette, this color is black. */
    G2D_BLT_BLACKNESS,  /* blackness */
    G2D_BLT_NOTMERGEPEN,    /* dst = ~(dst + src) */
    G2D_BLT_MASKNOTPEN, /* dst = ~src & dst */
    G2D_BLT_NOTCOPYPEN, /* dst = ~src */
    G2D_BLT_MASKPENNOT, /* dst = src & ~dst */
    G2D_BLT_NOT,    /* dst = ~dst */
    G2D_BLT_XORPEN, /* dst = src ^ dst */
    G2D_BLT_NOTMASKPEN, /* dst = ~(src & dst) */
    G2D_BLT_MASKPEN,    /* dst = src & dst */
    G2D_BLT_NOTXORPEN,  /* dst = ~(src ^ dst) */
    G2D_BLT_NOP,    /* dst = dst */
    G2D_BLT_MERGENOTPEN,    /* dst = ~src + dst */
    G2D_BLT_COPYPEN,    /* dst = src */
    G2D_BLT_MERGEPENNOT,    /* dst = src + ~dst */
    G2D_BLT_MERGEPEN,   /* dst = src + dst */
    G2D_BLT_WHITENESS,  /* whiteness */

    /* Fill the target rectangular area with the color associated with index 0 of the physical color palette.
     * For the default physical color palette, this color is white. */
    G2D_BLT_EXTRACT = 0x000000ff,

    G2D_ROT_90  = 0x00000100,
    G2D_ROT_180 = 0x00000200,
    G2D_ROT_270 = 0x00000300,
    G2D_ROT_0   = 0x00000400,
    G2D_ROT_H = 0x00001000,
    G2D_ROT_V = 0x00002000,
    G2D_ROT_90_H  = 0x00001100,
    G2D_ROT_90_V  = 0x00002100,
```

```
/* G2D_SM_TDLR_1 =  0x10000000, */
   G2D_SM_DTLR_1 = 0x10000000,
/* G2D_SM_TDRL_1 =  0x20000000, */
/* G2D_SM_DTRL_1 =  0x30000000, */
} g2d_blt_flags_h;
```

4.8.2 使用 demo

```
/* 二元光栅操作 */
int ret;
int g2d_fd;
g2d_blt_h blt;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&blt, 0, sizeof(g2d_blt_h));

blit.flag_h = G2D_BLT_NOTCOPYPEN;  /* 将原来的图像取反 */

blit.src_image_h.use_phy_addr = 0;
blit.src_image_h.fd = src_image_fd;
blit.src_image_h.bbuff = 1;
blit.src_image_h.mode = G2D_PIXEL_ALPHA;
blit.src_image_h.alpha = 0xff;
blit.src_image_h.format = G2D_FORMAT_ARGB8888;
blit.src_image_h.width = 1920;
blit.src_image_h.height = 1080;
blit.src_image_h.clip_rect.x = 0;
blit.src_image_h.clip_rect.y = 0;
blit.src_image_h.clip_rect.w = 1920;
blit.src_image_h.clip_rect.h = 1080;


blit.dst_image_h.use_phy_addr = 0;
blit.dst_image_h.fd = dst_image_fd;
blit.dst_image_h.bbuff = 1;
blit.dst_image_h.mode = G2D_PIXEL_ALPHA;
blit.dst_image_h.alpha = 0xff;
blit.dst_image_h.format = G2D_FORMAT_ARGB8888;
blit.dst_image_h.width = 1920;
blit.dst_image_h.height = 1080;
blit.dst_image_h.clip_rect.x = 0;
blit.dst_image_h.clip_rect.y = 0;
blit.dst_image_h.clip_rect.w = 1920;
blit.dst_image_h.clip_rect.h = 1080;

ret = ioctl(g2d_fd, G2D_CMD_BITBLT_H ,(unsigned long)(&blit))
```

4.9 三元光栅操作 (Rop3)

4.9.1 关键参数

```
typedef struct {
   g2d_rop3_cmd_flag back_flag;
   g2d_rop3_cmd_flag fore_flag;

   g2d_image_enh dst_image_h;
   g2d_image_enh src_image_h;
   g2d_image_enh ptn_image_h;
   g2d_image_enh mask_image_h;

} g2d_maskblt;

/* ROP3 command */
typedef enum {
   G2D_ROP3_BLACKNESS = 0x00,  /* dst = BLACK */
   G2D_ROP3_NOTSRCERASE = 0x11,   /* dst = (NOT src) AND (NOT dst) */
   G2D_ROP3_NOTSRCCOPY = 0x33, /* dst = (NOT src) */
   G2D_ROP3_SRCERASE = 0x44,   /* dst = src AND (NOT dst ) */
   G2D_ROP3_DSTINVERT = 0x55,  /* dst = (NOT dst) */
   G2D_ROP3_PATINVERT = 0x5A,  /* dst = pattern XOR dst */
   G2D_ROP3_SRCINVERT = 0x66,  /* dst = src XOR dst */
   G2D_ROP3_SRCAND = 0x88, /* dst = src AND dst */
   G2D_ROP3_MERGEPAINT = 0xBB, /* dst = (NOT src) OR dst */
   G2D_ROP3_MERGECOPY = 0xC0,  /* dst = (src AND pattern) */
   G2D_ROP3_SRCCOPY = 0xCC,    /* dst = src */
   G2D_ROP3_SRCPAINT = 0xEE,   /* dst = src OR dst */
   G2D_ROP3_PATCOPY = 0xF0,    /* dst = pattern */
   G2D_ROP3_PATPAINT = 0xFB,   /* dst = (~src OR pattern) OR DST */
   G2D_ROP3_WHITENESS = 0xFF,  /* dst = WHITE */
} g2d_rop3_cmd_flag;
```

4.9.2 使用 demo

G2D 目前支持的 ROP3 操作见模块功能概述

```
/* 三元光栅操作：遮罩操作 */
int ret;
g2d_maskblt mask;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
   printf("failed to open g2d device\n");
   return;
}
memset(&g2d_maskblt, 0, sizeof(g2d_maskblt));

mask.back_flag = G2D_ROP3_NOTSRCCOPY;
mask.fore_flag = G2D_ROP3_SRCINVERT;
mask.src_image_h.clip_rect.x = 0;
mask.src_image_h.clip_rect.y = 0;
```

```
mask.src_image_h.clip_rect.w = 1280;
mask.src_image_h.clip_rect.h = 800;
mask.src_image_h.width = 1280;
mask.src_image_h.height = 800;
mask.src_image_h.mode = G2D_PIXEL_ALPHA;
mask.dst_image_h.clip_rect.x = 0;
mask.dst_image_h.clip_rect.y = 0;
mask.dst_image_h.clip_rect.w = 1280;
mask.dst_image_h.clip_rect.h = 800;
mask.dst_image_h.width = 1280;
mask.dst_image_h.height = 800;
mask.dst_image_h.mode = G2D_PIXEL_ALPHA;
mask.mask_image_h.clip_rect.x = 0;
mask.mask_image_h.clip_rect.y = 0;
mask.mask_image_h.clip_rect.w = 1280;
mask.mask_image_h.clip_rect.h = 800;
mask.mask_image_h.width = 1280;
mask.mask_image_h.height = 800;
mask.mask_image_h.mode = G2D_PIXEL_ALPHA;
mask.ptn_image_h.clip_rect.x = 0;
mask.ptn_image_h.clip_rect.y = 0;
mask.ptn_image_h.clip_rect.w = 1280;
mask.ptn_image_h.clip_rect.h = 800;
mask.ptn_image_h.width = 1280;
mask.ptn_image_h.height = 800;
mask.ptn_image_h.mode = G2D_PIXEL_ALPHA;
mask.src_image_h.alpha = 0xff;
mask.mask_image_h.alpha = 0xff;
mask.ptn_image_h.alpha = 0xff;
mask.dst_image_h.alpha = 0xff;
mask.src_image_h.format = G2D_FORMAT_ARGB8888;
mask.mask_image_h.format = G2D_FORMAT_ARGB8888;
mask.ptn_image_h.format = G2D_FORMAT_ARGB8888;
mask.dst_image_h.format = G2D_FORMAT_ARGB8888;

ret = ioctl(int fd, G2D_CMD_MASK_H ,(unsigned long)(&mask))
```

4.10 批处理工作模式

4.10.1 关键参数

```
struct mixer_para {
   g2d_operation_flag op_flag;
   g2d_blt_flags_h flag_h;
   g2d_rop3_cmd_flag back_flag;
   g2d_rop3_cmd_flag fore_flag;
   g2d_bld_cmd_flag    bld_cmd;
   g2d_image_enh src_image_h;
   g2d_image_enh dst_image_h;
   g2d_image_enh ptn_image_h;
   g2d_image_enh mask_image_h;
   g2d_ck ck_para;
};

typedef enum {
```

```
    OP_FILLRECT = 0x1,
    OP_BITBLT = 0x2,
    OP_BLEND = 0x4,
    OP_MASK = 0x8,
    OP_SPLIT_MEM = 0x10,
} g2d_operation_flag;
```

struct mixer_para 是 RCQ 批处理的核心结构体。可以看到除了第一个成员，其它成员的类型都是旧驱动里面有的。struct mixer_para 是之前驱动接口结构体的一个合集，如图 3-1 所示：

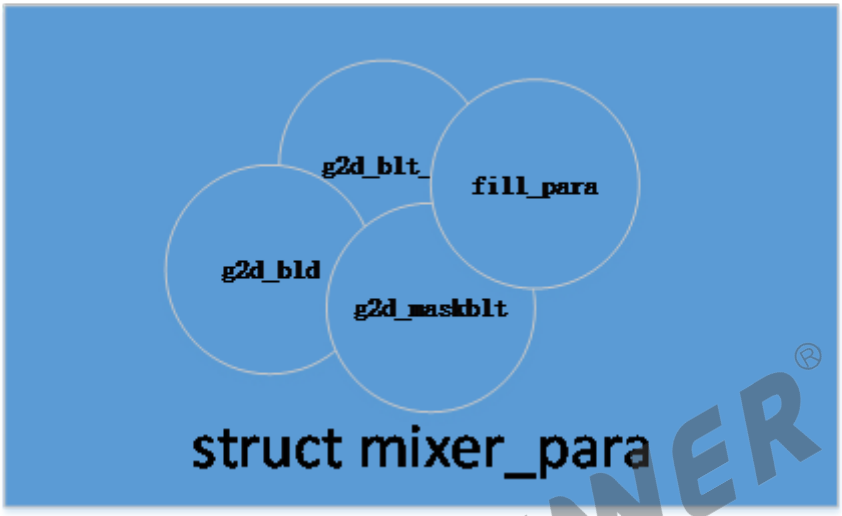

图 4-1: mixerpara

所以可以用批处理接口完成上面其它接口的功能，只要设置好对应的成员和 g2d_operation_flag 即可。

G2D 的批处理工作模式主要有同步和异步两种。

4.10.2 批处理工作同步模式使用 demo

4.10.2.1 G2D_CMD_MIXER_TASK

```
/* 将格式为argb8888格式，分辨率为1024x1024分辨率图像src_image_fd中ROI区域为(0，0，512，512)、(0，512，512，
      512)、(512，0，512，512)、(512，512，512，512)的区域裁剪下来，输出到四个512*512分辨率的图像dst_image_fd
      [4]上去
 * 以使用dmabufferheap内存句柄为例 */
#define FRAME_TO_BE_PROCESS 4
int g2d_fd;
unsigned long arg[2];
struct mixer_para info[FRAME_TO_BE_PROCESS];
int ret;

g2d_fd = open("dev/g2d", O_RDWR);
if (g2d_fd < 0) {
    printf("failed to open g2d device\n");
```

```
    return;
}
memset(&info, 0, FRAME_TO_BE_PROCESS * sizeof(info));

info[0].op_flag = OP_BITBLT;
info[0].flag_h = G2D_BLT_NONE_H;
info[0].src_image_h.use_phy_addr = 0;
info[0].src_image_h.fd = src_image_fd;
info[0].src_image_h.bbuff = 1;
info[0].src_image_h.mode = G2D_PIXEL_ALPHA;
info[0].src_image_h.alpha = 0xff;
info[0].src_image_h.format = G2D_FORMAT_ARGB8888;
info[0].src_image_h.width = 1024;
info[0].src_image_h.height = 1024;
info[0].src_image_h.clip_rect.x = 0;
info[0].src_image_h.clip_rect.y = 0;
info[0].src_image_h.clip_rect.w = 512;
info[0].src_image_h.clip_rect.h = 512;
info[0].dst_image_h.use_phy_addr = 0;
info[0].dst_image_h.fd = dst_image_fd[0];
info[0].dst_image_h.bbuff = 1;
info[0].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[0].dst_image_h.alpha = 0xff;
info[0].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[0].dst_image_h.width = 512;
info[0].dst_image_h.height = 512;
info[0].dst_image_h.clip_rect.x = 0;
info[0].dst_image_h.clip_rect.y = 0;
info[0].dst_image_h.clip_rect.w = 512;
info[0].dst_image_h.clip_rect.h = 512;

info[1].op_flag = OP_BITBLT;
info[1].flag_h = G2D_BLT_NONE_H;
info[1].src_image_h.use_phy_addr = 0;
info[1].src_image_h.fd = src_image_fd;
info[1].src_image_h.bbuff = 1;
info[1].src_image_h.mode = G2D_PIXEL_ALPHA;
info[1].src_image_h.alpha = 0xff;
info[1].src_image_h.format = G2D_FORMAT_ARGB8888;
info[1].src_image_h.width = 1024;
info[1].src_image_h.height = 1024;
info[1].src_image_h.clip_rect.x = 0;
info[1].src_image_h.clip_rect.y = 512;
info[1].src_image_h.clip_rect.w = 512;
info[1].src_image_h.clip_rect.h = 512;
info[1].dst_image_h.use_phy_addr = 0;
info[1].dst_image_h.fd = dst_image_fd[1];
info[1].dst_image_h.bbuff = 1;
info[1].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[1].dst_image_h.alpha = 0xff;
info[1].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[1].dst_image_h.width = 512;
info[1].dst_image_h.height = 512;
info[1].dst_image_h.clip_rect.x = 0;
info[1].dst_image_h.clip_rect.y = 0;
info[1].dst_image_h.clip_rect.w = 512;
info[1].dst_image_h.clip_rect.h = 512;

info[2].op_flag = OP_BITBLT;
info[2].flag_h = G2D_BLT_NONE_H;
```

```
info[2].src_image_h.use_phy_addr = 0;
info[2].src_image_h.fd = src_image_fd;
info[2].src_image_h.bbuff = 1;
info[2].src_image_h.mode = G2D_PIXEL_ALPHA;
info[2].src_image_h.alpha = 0xff;
info[2].src_image_h.format = G2D_FORMAT_ARGB8888;
info[2].src_image_h.width = 1024;
info[2].src_image_h.height = 1024;
info[2].src_image_h.clip_rect.x = 512;
info[2].src_image_h.clip_rect.y = 0;
info[2].src_image_h.clip_rect.w = 512;
info[2].src_image_h.clip_rect.h = 512;
info[2].dst_image_h.use_phy_addr = 0;
info[2].dst_image_h.fd = dst_image_fd[2];
info[2].dst_image_h.bbuff = 1;
info[2].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[2].dst_image_h.alpha = 0xff;
info[2].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[2].dst_image_h.width = 512;
info[2].dst_image_h.height = 512;
info[2].dst_image_h.clip_rect.x = 0;
info[2].dst_image_h.clip_rect.y = 0;
info[2].dst_image_h.clip_rect.w = 512;
info[2].dst_image_h.clip_rect.h = 512;

info[3].op_flag = OP_BITBLT;
info[3].flag_h = G2D_BLT_NONE_H;
info[3].src_image_h.use_phy_addr = 0;
info[3].src_image_h.fd = src_image_fd;
info[3].src_image_h.bbuff = 1;
info[3].src_image_h.mode = G2D_PIXEL_ALPHA;
info[3].src_image_h.alpha = 0xff;
info[3].src_image_h.format = G2D_FORMAT_ARGB8888;
info[3].src_image_h.width = 1024;
info[3].src_image_h.height = 1024;
info[3].src_image_h.clip_rect.x = 512;
info[3].src_image_h.clip_rect.y = 512;
info[3].src_image_h.clip_rect.w = 512;
info[3].src_image_h.clip_rect.h = 512;
info[3].dst_image_h.use_phy_addr = 0;
info[3].dst_image_h.fd = dst_image_fd[3];
info[3].dst_image_h.bbuff = 1;
info[3].dst_image_h.mode = G2D_PIXEL_ALPHA;
info[3].dst_image_h.alpha = 0xff;
info[3].dst_image_h.format = G2D_FORMAT_ARGB8888;
info[3].dst_image_h.width = 512;
info[3].dst_image_h.height = 512;
info[3].dst_image_h.clip_rect.x = 0;
info[3].dst_image_h.clip_rect.y = 0;
info[3].dst_image_h.clip_rect.w = 512;
info[3].dst_image_h.clip_rect.h = 512;

arg[0] = (unsigned long) info;
arg[1] =  FRAME_TO_BE_PROCESS;
ret = ioctl(g2d_fd, G2D_CMD_MIXER_TASK ,(unsigned long)(arg))
```

4.10.3 批处理工作异步模式使用 demo

4.10.3.1 G2D_CMD_CREATE_TASK

- PROTOTYPE

```
int ioctl(int fd, int cmd, void *arg)
```

- ARGUMENTS

```
cmd        G2D_CMD_CREATE_TASK

arg[0]     arg指向mixer_para指针，批处理的话就是数组指针。

arg[1]     需要处理的帧的数量，大于等于1。
```

- RETURN

```
成功：task id，大于等于1，其它情况则为失败。

arg[0]对应的指针所指向的mixer_para内容会被更新。
```

该 ioctl 命令用于创建新的批处理实例，但不做硬件处理, 只是准备好软件。

这个过程会构造对应帧数的 rcq 队列内存以及进行输入输出图像的 dma map 和 dma umap 操作，构造完毕之后会更新 mixer_para 回应用层。task_id 是唯一的，只要不销毁批处理实例，会一直占据这个 id。根据这个 id 用户可以进一步操作，比如设置，销毁，获取当前 mixer_para。

如下例子，会创建两个不同帧数和输入输出格式的批处理实例，最终得到两个不同的 task id，task0 和 task1。mixer_para 如何构造参考 G2D_CMD_MIXER_TASK 的例子。

```
config_mixer_para(&test_info.info);
config_mixer_para(&test_info2.info);
arg[0] = (unsigned long)test_info.info;
arg[1] = FRAME_TO_BE_PROCESS;
task0 = ioctl(g2d_fd, G2D_CMD_CREATE_TASK, (arg));
if (task0 < 1) {
   printf("[%d][%s][%s]G2D_CMD_CREATE_TASK failure!\n", __LINE__,
       __FILE__, __FUNCTION__);
}

arg[0] = (unsigned long)test_info2.info;
arg[1] = FRAME_TO_BE_PROCESS2;
task1 = ioctl(g2d_fd, G2D_CMD_CREATE_TASK, (arg));
if (task1 < 1) {
   printf("[%d][%s][%s]G2D_CMD_CREATE_TASK failure!\n", __LINE__,
       __FILE__, __FUNCTION__);
}
```

4.10.3.2 G2D_CMD_TASK_APPLY

- PROTOTYPE

```
int ioctl(int fd, int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_TASK_APPLY

arg[0]      task id(由G2D_CMD_CREATE_TASK命令获得)
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是执行批处理的硬件操作。

值得注意 arg[1] 中的 mixer_para，必须是 G2D_CMD_CREATE_TASK 之后返回的 mixer_para 或者是通过另外一个 ioctl 命令 G2D_CMD_TASK_GET_PARA 才行，这里不需要制定帧数的原因是前面的 G2D_CMD_CREATE_TASK 已经指定好帧数，而 G2D_CMD_TASK_APPLY 是基于 task id 来执行的。

```
arg[0] = task0;
if(ioctl(g2d_fd, G2D_CMD_TASK_APPLY, (arg)) < 0) {
   printf("[%d][%s][%s]G2D_CMD_TASK_APPLY failure!\n", __LINE__,
          __FILE__, __FUNCTION__);
   goto FREE_SRC;
}

arg[0] = task1;
if(ioctl(g2d_fd, G2D_CMD_TASK_APPLY, (arg)) < 0) {
   printf("[%d][%s][%s]G2D_CMD_TASK_APPLY failure!\n", __LINE__,
          __FILE__, __FUNCTION__);
   goto FREE_SRC;
}
```

4.10.3.3 G2D_CMD_TASK_DESTROY

- PROTOTYPE

```
int ioctl(int fd, int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_TASK_DESTROY

arg[0]      task id
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是销毁指定 task id 的批处理实例。

```
arg[0] = task0;
ret = ioctl(g2d_fd, G2D_CMD_TASK_DESTROY ,(unsigned long)(arg))
```

4.10.3.4 G2D_CMD_TASK_GET_PARA

- PROTOTYPE

```
int ioctl(int fd, int cmd, void *arg)
```

- ARGUMENTS

```
cmd         G2D_CMD_TASK_DESTROY

arg[0]      task id

arg[1]      指向mixer_para指针，多帧的话就是数组指针。
```

- RETURN

```
成功：0，失败：失败号
```

该 ioctl 命令的作用是获取指定 task id 的 mixer para。

用户必须自行保证传入的指针所指向的内存足够存放这么多帧的参数。

4.11 操作混合

rotate 功能仅可以和 crop 操作混合，不支持和其他操作一起做。

图像裁剪拼接、缩放或下采样、图像格式转换、图层混合等操作可以混合起来一起做，但注意缩放或下采样、图像格式转换仅支持对 vchn 的输入源进行处理。比如可以对 src0 进行裁剪、缩放和下采样后，再和经过裁剪操作的 src1 进行图层混合，最后输出一个指定格式。具体请参考各功能 demo 涉及到的参数。

5 FAQ

5.1 G2D 理论性能计算公式

G2D 在执行拷贝时，可以通过以下公式进行计算理论耗时（该功能仅支持数据的拷贝评估），G2D 每个时钟周期能够处理的像素数量和 G2D 频率可以查询设计指标获得：

单次拷贝图像耗时 = 图像宽 × 图像高 / G2D 每秒能处理的像素数量

= 图像宽 × 图像高 /（G2D 每个时钟周期能够处理的像素数量 × G2D 频率）

例如：一幅 1920 × 1080 大小的图像用 G2D（频率设定为 300M）做旋转的理论耗时是：

G2D101： 1920 × 1080 /（4 × 300000000）= 0.001728s

而实际的耗时与使用的内存类型是相关的，耗时上，物理地址 ~= dma_fd + map/unmap。

此外，还和 cpu 占用率、带宽等信息相关。

5.2 G2D 的处理帧率不能满足我们产品的需求，有什么办法可以提升么

- 建议考虑能使用 rcq 批处理优使用 rcq 批处理工作模式
- 需要查询具体 IP 的时钟信息，确定 G2D 是否可以提频。如果可以，修改 dtsi 节点下 g2d 频率信息，重新编译固件
- 如果每次传的 buffer 都是固定的几个，建议将 dma_map 后的物理地址保存下来，每次直接传物理地址。

5.3 打印 G2D irq pending flag timeout

出现此错误的可能很多，需要一一排查

1. 检查 g2d 的时钟、供电是否正常。

```
# 检查时钟
mount -t debugfs nodev /sys/kernel/debug
cat /sys/kernel/debug/clk/clk_summary | grep g2d

# 检查供电
mount -t debugfs none /sys/kernel/debug/
cat /sys/kernel/debug/pm_genpd/pm_genpd_summary
```

执行任务后，等待一会后检查 g2d 中断数量统计值前后是否增加。如果没有，一般说明设置的参数有问题。如果产生了，调大 G2D 驱动中 WAIT_CMD_TIME_MS 宏，直到不打印超时 log，抓取输出 buffer 查看结果是否正确。

```
cat /proc/interrupts | grep g2d
```

2. 打开 g2d 调试节点，重新执行任务，抓取打印的调试信息，参考 demo 分析参数是否正确

```
# 参数信息
echo 1 > /sys/class/g2d/g2d/attr/debug
# 时间信息
echo 1 > /sys/class/g2d/g2d/attr/func_time
```

5.4 每次测试结果都不一样，出图上有黑条带

同样参数如果每次出图都不一样，建议增加刷新 cache 操作。

如果每次出图都一样，但还是有固定的条带，有可能是 IC 不良品，建议联系原厂确认和更换

著作权声明

商标声明

免责声明