GE2D 用户编程手册

日期	版本	发布者	内容
2011/02/21	V0.1	Jianfeng.wang	Ge2d 用户编程参考

GE2D 用户编程手册

[目录结构]

GE2D 用户编程手册

1. 【引言】

我们在下面探讨的 GE2D 代表 2D graphics engine 模块,本模块可以实现矩形填充（fill rectangle）, 位块移动（blit）缩放（stretchblit）以及混合操作（blend）。GE2D 设备支持多进程操作，也就是用户可以在不同的应用进程中多次打开此设备并发操作。

本文档的编写目的是为上层应用软件开发人员提供参考，所以对 GE2D 的内部实现原理不做探讨。力求通过事例代码，引导读者快速实现应用代码编写。

2. 【GE2D 相关的数据结构说明】

以下这些数据结构是 GE2D 操作需要使用的，读者可以首先浏览一下，以作后续章节参考使用。

◆ IOCTL 命令控制字

```
#define    GE2D_STRETCHBLIT_NOALPHA_NOBLOCK    0x4708
#define    GE2D_BLIT_NOALPHA_NOBLOCK           0x4707
#define    GE2D_BLEND_NOBLOCK                  0x4706
#define    GE2D_BLIT_NOBLOCK                   0x4705
#define    GE2D_STRETCHBLIT_NOBLOCK            0x4704
#define    GE2D_FILLRECTANGLE_NOBLOCK          0x4703

#define    GE2D_STRETCHBLIT_NOALPHA            0x4702
#define    GE2D_BLIT_NOALPHA                   0x4701
#define    GE2D_BLEND                          0x4700
#define    GE2D_BLIT                           0x46ff
#define    GE2D_STRETCHBLIT                    0x46fe
#define    GE2D_FILLRECTANGLE                  0x46fd
#define    GE2D_SRCCOLORKEY                    0x46fc
#define    GE2D_SET_COEF                       0x46fa
#define    GE2D_CONFIG                         0x46f9
#define    GE2D_ANTIFLICKER_ENABLE             0x46f8
```

- 上述带 **NOBLOCK** 后缀的命令与不带此后缀命令的区别：当发送命令结束后是否等待 GE2D 模块操作完成，如果带此后缀 IOCTL 操作将阻塞直到 GE2D 模块完成本次申请的操作，否则立即返回。
- **GE2D_ANTIFLICKER_ENABLE** ------ 在 CVBS 输出模式下 enable 或者 diable antiflicker,使用此命令打开和关闭防闪烁的功能。IOCTL 命令的参数为 1 :enable antiflicker 0: disable antiflicker.
- **GE2D_CONFIG** ---- 在每次的 GE2D 操作源和目的类型发生改变的时候都要通过此命令对 GE2D 模块重新配置。(本 IOCTL 命令的参数请参照**模块配置数据结构**)
- **GE2D_SET_COEF** --- 设定缩放模式下，水平方向和垂直方向数据平均的方式，有以下三种方式。

  ```
  #define FILTER_TYPE_BICUBIC      1
  #define FILTER_TYPE_BILINEAR     2
  #define FILTER_TYPE_TRIANGLE     3
  ```

- **GE2D_SRCCOLORKEY** --- 我们在 GE2D 的操作中，不仅可以设定 COLORKEY,而且可以设定屏蔽的是 COLORKEY 还是除 COLORKEY 以外的颜色。下面举例说明，当我们设定了一种颜色作为 COLORKEY，并指定为屏蔽 COLORKEY 操作方式的时候，如果我们做 GE2D 的 BLIT 操作，那么 SRC 数据中所有与 COLORKEY 相同（match）的数据都将不参与操作，也就是不被 COPY。具体参考 **COLORKEY 控制数据结构**，请注意我们在这里讨论的 COLORKEY 是 GE2D 模块专用的 COLORKEY 与 OSD 层指定的 COLORKEY 无关。

- **GE2D_FILLRECTANGLE** --- 填充矩形。具体的操作参数参考 **ge2d_op_para_t**
- **GE2D_STRETCHBLIT** --- 缩放操作，具体的操作参数参考 **ge2d_op_para_t**

- **GE2D_BLIT** --- 位块移动操作，具体的操作参数参考 **ge2d_op_para_t**
- **GE2D_BLEND** --- 混合操作，注意 BLEND 操作的操作码，请参考后续的 BLENDOP 码。
- 上述带 **NOALPHA** 后缀的操作，是标识在 GE2D 操作中忽略源当中的 alpha 位信息。

◆ **模块配置数据结构**

```
struct config_para_t {
int    src_dst_type;
int    alu_const_color;
unsigned int src_format;
unsigned int dst_format ;
config_planes_t src_planes[4];
config_planes_t dst_planes[4];
src_key_ctrl_t    src_key;
};
```

src_dst_type: 取值为下列之一，代表源和目的地址的类型。

```
OSD0_OSD0 = 0,        //src =osd0    dst=osd0
OSD0_OSD1=1,          // src =osd0    dst=osd1
OSD1_OSD1=2,
OSD1_OSD0=3,
ALLOC_OSD0=4,         // src =alloc   dst=osd0
ALLOC_OSD1=5,
ALLOC_ALLOC=6
```

alu_const_color：当 GE2D 操作为 OP_LOGIC 的时候，参与 LOGIC 操作的 const color 值。

src_format：ge2d 操作 source 颜色模式。

dst_format：ge2d 操作 destination 颜色模式

上述两个数据类型可能的取值范围。

```
GE2D_FORMAT_S16_RGB_655,
GE2D_FORMAT_S16_RGB_844,
GE2D_FORMAT_S16_RGBA_6442,
GE2D_FORMAT_S16_RGBA_4444,
GE2D_FORMAT_S16_RGBA_4642 ,
GE2D_FORMAT_S16_ARGB_1555,
GE2D_FORMAT_S16_ARGB_4444,
GE2D_FORMAT_S16_RGB_565,
GE2D_FORMAT_S24_ARGB_6666,
GE2D_FORMAT_S24_RGBA_6666,
GE2D_FORMAT_S24_ARGB_8565, ,
GE2D_FORMAT_S24_RGBA_5658,
GE2D_FORMAT_S24_BGR,
GE2D_FORMAT_S24_RGB,
GE2D_FORMAT_S32_BGRA,
GE2D_FORMAT_S32_ABGR,
GE2D_FORMAT_S32_RGBA,
```

GE2D_FORMAT_S32_ARGB,

src_planes：参考下面 **config_planes_t** 的说明

dst_planes：参考下面 **config_planes_t** 的说明

src_key ：参考 COLORKEY 控制数据结构

- **源或者目的地址 CANVAS 属性定义**。

typedef struct {
unsigned long addr;
unsigned int w;
unsigned int h;
}config_planes_t;

addr: canvas 物理地址

w :canvas width

h :canvas height

- **COLORKEY 控制数据结构**

typedef struct{
int key_enable;
int key_color;
int key_mask;
int key_mode;
}src_key_ctrl_t;

key_enable: 1:enable colorkey 0:disable colorkey

key_color: ge2d color key value

key_mask: bitmask value

key_mode: 0:mask when match 1:mask when unmatch.

- **操作参数数据结构**

typedef struct {
unsigned int color ;
rectangle_t src1_rect;
rectangle_t src2_rect;
rectangle_tdst_rect;
int op;
}ge2d_op_para_t ;

color: 用于 FILLRECT 填充的颜色。

src1_rect: source1 的尺寸。

src2_rect:source2 的尺寸。

dst_rect: destination 的尺寸。

op:请参考后续的 **Blend 操作的操作码**

typedef struct {
int x; // X coordinate of its top-left point
int y; // Y coordinate of its top-left point
int w; // width of it
int h; // height of it
}rectangle_t;

◆ **Blend 操作的操作码。**

```
Blend_op_code= (color_blending_mode << 24) |
               (color_blending_src_factor << 20) |
               (color_blending_dst_factor << 16) |
               (alpha_blending_mode << 8) |
               (alpha_blending_src_factor << 4) |
               (alpha_blending_dst_factor << 0);
```

color_blending_mode：

alpha_blending_mode：可能的取值

```
#define BLENDOP_ADD                 0       //Cd = Cs*Fs+Cd*Fd
#define BLENDOP_SUB                 1       //Cd = Cs*Fs-Cd*Fd
#define BLENDOP_REVERSE_SUB         2       //Cd = Cd*Fd-Cs*Fs
#define BLENDOP_MIN                 3       //Cd = Min(Cd*Fd,Cs*Fs)
#define BLENDOP_MAX                 4       //Cd = Max(Cd*Fd,Cs*Fs)
#define BLENDOP_LOGIC               5       //Cd = Cs op Cd
#define BLENDOP_LOGIC_CLEAR         (BLENDOP_LOGIC+0 )
#define BLENDOP_LOGIC_COPY          (BLENDOP_LOGIC+1 )
#define BLENDOP_LOGIC_NOOP          (BLENDOP_LOGIC+2 )
#define BLENDOP_LOGIC_SET           (BLENDOP_LOGIC+3 )
#define BLENDOP_LOGIC_COPY_INVERT   (BLENDOP_LOGIC+4 )
#define BLENDOP_LOGIC_INVERT        (BLENDOP_LOGIC+5 )
#define BLENDOP_LOGIC_AND_REVERSE   (BLENDOP_LOGIC+6 )
#define BLENDOP_LOGIC_OR_REVERSE    (BLENDOP_LOGIC+7 )
#define BLENDOP_LOGIC_AND           (BLENDOP_LOGIC+8 )
#define BLENDOP_LOGIC_OR            (BLENDOP_LOGIC+9 )
#define BLENDOP_LOGIC_NAND          (BLENDOP_LOGIC+10)
#define BLENDOP_LOGIC_NOR           (BLENDOP_LOGIC+11)
#define BLENDOP_LOGIC_XOR           (BLENDOP_LOGIC+12)
#define BLENDOP_LOGIC_EQUIV         (BLENDOP_LOGIC+13)
#define BLENDOP_LOGIC_AND_INVERT    (BLENDOP_LOGIC+14)
#define BLENDOP_LOGIC_OR_INVERT     (BLENDOP_LOGIC+15)
```

color_blending_src_factor：

color_blending_dst_factor：可能的取值

```
#define COLOR_FACTOR_ZERO                       0
#define COLOR_FACTOR_ONE                        1
#define COLOR_FACTOR_SRC_COLOR                  2
#define COLOR_FACTOR_ONE_MINUS_SRC_COLOR        3
#define COLOR_FACTOR_DST_COLOR                  4
#define COLOR_FACTOR_ONE_MINUS_DST_COLOR        5
#define COLOR_FACTOR_SRC_ALPHA                  6
#define COLOR_FACTOR_ONE_MINUS_SRC_ALPHA        7
#define COLOR_FACTOR_DST_ALPHA                  8
#define COLOR_FACTOR_ONE_MINUS_DST_ALPHA        9
```

```
#define COLOR_FACTOR_CONST_COLOR                10
#define COLOR_FACTOR_ONE_MINUS_CONST_COLOR      11
#define COLOR_FACTOR_CONST_ALPHA                12
#define COLOR_FACTOR_ONE_MINUS_CONST_ALPHA      13
#define COLOR_FACTOR_SRC_ALPHA_SATURATE         14
```

alpha_blending_src_factor:

alpha_blending_dst_factor:可能的取值

```
#define ALPHA_FACTOR_ZERO                       0
#define ALPHA_FACTOR_ONE                        1
#define ALPHA_FACTOR_SRC_ALPHA                  2
#define ALPHA_FACTOR_ONE_MINUS_SRC_ALPHA        3
#define ALPHA_FACTOR_DST_ALPHA                  4
#define ALPHA_FACTOR_ONE_MINUS_DST_ALPHA        5
#define ALPHA_FACTOR_CONST_ALPHA                6
#define ALPHA_FACTOR_ONE_MINUS_CONST_ALPHA      7
```

- **模块配置扩展数据结构**

```
typedef    struct    {
        int    canvas_index;
        int    top;
        int    left;
        int    width;
        int    height;
        int    format;
        int    mem_type;
        int    color;
        unsigned char x_rev;
        unsigned char y_rev;
        unsigned char fill_color_en;
        unsigned char fill_mode;
}src_dst_para_ex_t ;
```

canvas_index: canvas 索引号

top,left,width,height: canvas 尺寸

format: ge2d 颜色模式，参照**模块配置数据结构**中 src_format ,dst_format**.**

mem_type: 可能的取值范围

CANVAS_OSD0 =0,

CANVAS_OSD1 =1,

CANVAS_ALLOC=2

color: src 的 default color.比如 fill rect 就是使用的这个 default color.

x_rev: x 方向是否反转。

y_rev: y 方向是否反转。

fill_color_en: 1: 使能 default color 作为填充色

0：关闭上述功能。

fill_mode: **0:**重复最后的数据

1:使用 default color.

```
typedef       struct {
src_dst_para_ex_t src_para;
src_dst_para_ex_t src2_para;
src_dst_para_ex_t dst_para;

//key mask
src_key_ctrl_t    src_key;
src_key_ctrl_t    src2_key;

int alu_const_color;
unsigned src1_gb_alpha;
unsigned op_mode;
unsigned char bitmask_en;
unsigned char bytemask_only;
unsigned int    bitmask;
unsigned char dst_xy_swap;

// scaler and phase releated
unsigned hf_init_phase;
int hf_rpt_num;
unsigned hsc_start_phase_step;
int hsc_phase_slope;
unsigned vf_init_phase;
int vf_rpt_num;
unsigned vsc_start_phase_step;
int vsc_phase_slope;
unsigned char src1_vsc_phase0_always_en;
unsigned char src1_hsc_phase0_always_en;
unsigned char src1_hsc_rpt_ctrl;    //1bit, 0: using minus, 1: using repeat data
unsigned char src1_vsc_rpt_ctrl;    //1bit, 0: using minus    1: using repeat data

//canvas info
config_planes_t src_planes[4];
config_planes_t src2_planes[4];
config_planes_t dst_planes[4];
}config_para_ex_t;
```

src_para, src2_para, dst_para: 参照**src_dst_para_ex_t**

src_key,src2_key: 参照**COLORKEY 控制数据结构**

alu_const_color: 参与算数运算的 const color.

例如： Cs*Fs + Cd*Fd

Cs 代表源的颜色数据

Fs:代表源的系数

Cd:目的的颜色数据。

Fd:目的的系数。

其中系数有多种选择，可以选择 0,1 以及 const color 等等，当选择 const color 的时候，用的就是这个值。

src1_gb_alpha:	src1 的 global alpha
bitmask_en:	destination bit mask enable : 0:disable 1:enable
bitmask:	destination bit mask value
bytemask_only:	destination bytemask only if destination bitmask is enable
dst_xy_swap:	destination x y swap.
hf_init_phase :	horizontal scaler initial phase
hf_rpt_num :	horizontal repeat line0 number
hsc_start_phase_step:	horizontal scaler phase step
hsc_phase_slope:	horizontal phase slope(相位斜率)
vf_init_phase:	vertical scaler initial phase
vf_rpt_num:	vertical repeat line0 number
vsc_start_phase_step:	vertical scaler phase step
vsc_phase_slope:	vertical phase slope
src1_vsc_phase0_always_en :	if true,src1 always use phase0 in vertical scaler(0 or 1)
src1_hsc_phase0_always_en:	if true, always use phase0 in horizontal scaler(0 or 1)
src1_hsc_rpt_ctrl:	1bit, 0: using minus, 1: using repeat data
src1_vsc_rpt_ctrl:	1bit, 0: using minus 1: using repeat data
src_planes:	**参照**config_planes_t
src2_planes:	**参照**config_planes_t
dst_planes:	**参照**config_planes_t

3. 【GE2D 模块的工作模式介绍】

根据 GE2D 操作源和目的类型，我们把 GE2D 的操作分为以下几个工作模式。并分别说明。

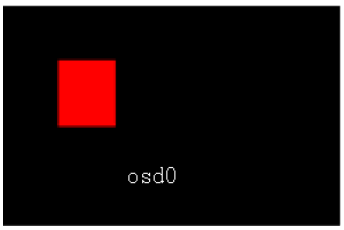
- 单一的 source 到 destination 的操作。
  - Fill rectangle:用指定的颜色填充指定的矩形区域。

    Sample codes:

    ```
    config_para_t config;
    ge2d_op_para_t op_para;
    int    ge2d_fd;

    ge2d_fd=open("/dev/ge2d",O_RDWR);
    config. src_dst_type=OSD0_OSD0;
    ioctl(ge2d_fd, GE2D_CONFIG, &config) ;
    op_para.src1_rect.x=100;
    op_para.src1_rect.y=100;
    op_para.src1_rect.w=100;
    op_para.src1_rect.h=100;

    op_para.color=0xff0000ff; //color order :RGBA
    ioctl(ge2d_fd, GE2D_FILLRECTANGLE, &op_para) ;
    ```

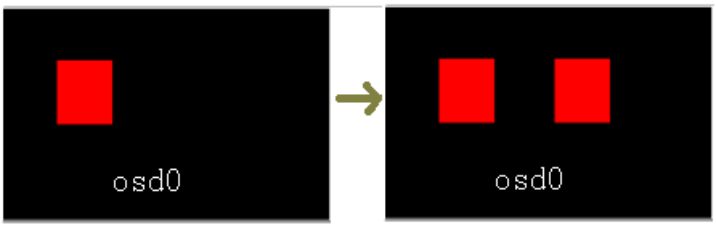
    

  - Blit: 源到目的的位块的移动。下面的事例代码描述的是 OSD 之间的操作，包括 OSD0_OSD0,OSD0_OSD1,OSD1_OSD0,OSD1_OSD1，对于使用 CMEM 以及其他的内存分配器分配的内存空间之间，以及这些内存空间与 OSD 之间的操作，在后续章节会加以解释和演示说明。

    Sample codes:

    ```
    config.src_dst_type=OSD0_OSD0;
    ioctl(ge2d_fd, GE2D_CONFIG, &config) ;
    op_para.src1_rect.x=100;
    op_para.src1_rect.y=100;
    op_para.src1_rect.w=100;
    op_para.src1_rect.h=100;

    op_para.dst_rect.x=400;
    op_para.dst_rect.y=100;
    ioctl(ge2d_fd, GE2D_BLIT, &op_para) ;
    ```



config.src_dst_type=OSD0_OSD1;

ioctl(ge2d_fd, GE2D_CONFIG, &config) ;

op_para.src1_rect.x=100;

op_para.src1_rect.y=100;

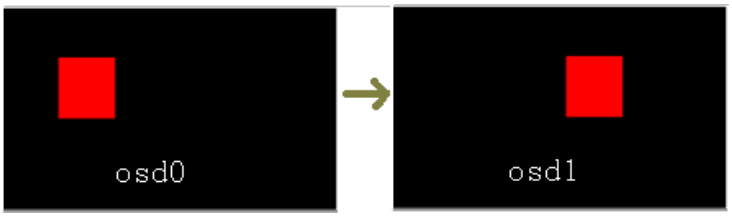
op_para.src1_rect.w=100;

op_para.src1_rect.h=100;

op_para.dst_rect.x=400;

op_para.dst_rect.y=100;

ioctl(ge2d_fd, GE2D_BLIT, &op_para) ;



- Stretchblit:位块的缩放移动。

  Sample codes:

  config.src_dst_type=OSD0_OSD0;

  ioctl(ge2d_fd, GE2D_CONFIG, &config) ;

  op_para.src1_rect.x=100;

  op_para.src1_rect.y=100;

  op_para.src1_rect.w=100;

  op_para.src1_rect.h=100;

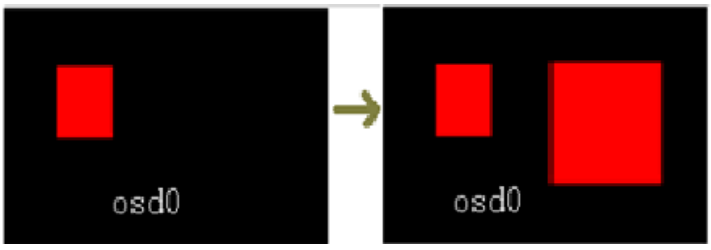
  op_para.dst_rect.x=400;

  op_para.dst_rect.y=100;

  op_para.dst_rect.w=200;

  op_para.dst_rect.h=200;

  ioctl(ge2d_fd, GE2D_BLIT, &op_para) ;



config.src_dst_type=OSD0_OSD1;

ioctl(ge2d_fd, GE2D_CONFIG, &config) ;

op_para.src1_rect.x=100;

op_para.src1_rect.y=100;

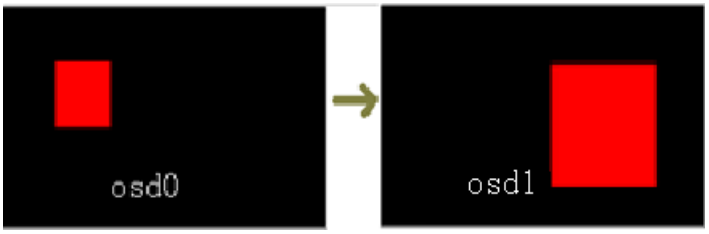
op_para.src1_rect.w=100;

op_para.src1_rect.h=100;

op_para.dst_rect.x=400;

op_para.dst_rect.y=100;

ioctl(ge2d_fd, GE2D_BLIT, &op_para) ;



- 两个 source（src1,src2） 到 destination(dst)的操作。
  - Blend 操作,src2 同时作为 dst.

    下面的事例代码演示，src1 OSD0 中的数据与 src2 OSD1 中的数据进行相加。而后相加的结果显示在 OSD1 上

    config.src_dst_type=OSD0_OSD1;

    ioctl(ge2d_fd, GE2D_CONFIG, & config) ;

    op_para.src1_rect.x=100;

    op_para.src1_rect.y=100;

    op_para.src1_rect.w=100;

    op_para.src1_rect.h=100;

    op_para.src2_rect.x=400;

    op_para.src2_rect.y=100;

    op_para.src2_rect.w=100;

    op_para.src2_rect.h=100;

    op_para.dst_rect.x=100;

    op_para.dst_rect.y=100;

    op_para.dst_rect.w=100;

    op_para.dst_rect.h=100;

    op_para.op=blendop(

    OPERATION_ADD,

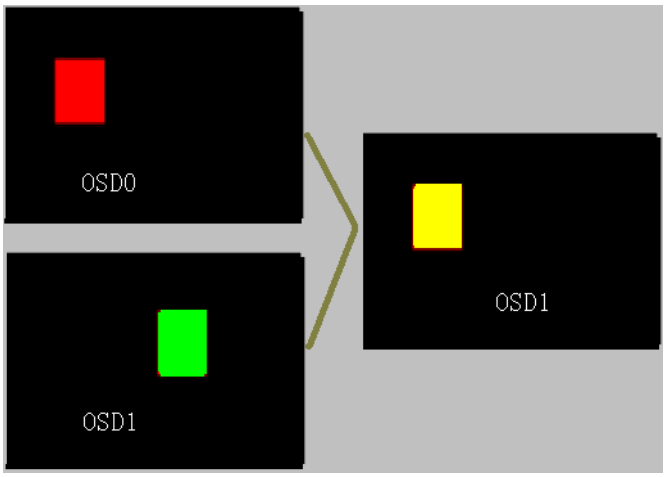
    / color blending src factor */*

    COLOR_FACTOR_ONE,

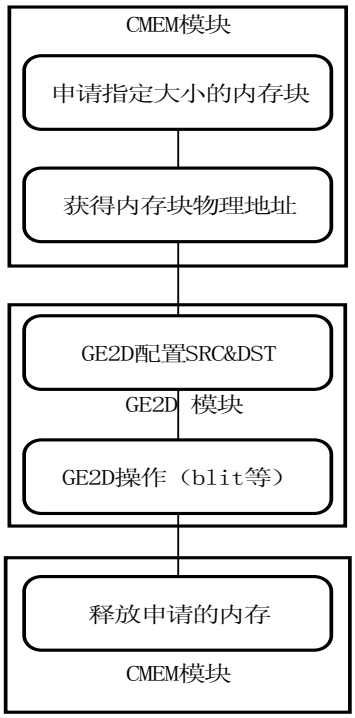
    / color blending dst factor */*

    COLOR_FACTOR_ONE,

```
        /* alpha blending_mode */
        OPERATION_MAX,
        /* alpha blending src factor */
        ALPHA_FACTOR_ONE,
        /* color blending dst factor */
        ALPHA_FACTOR_ONE);
ioctl( osd0->ge2d_fd, GE2D_BLEND, &op_para) ;
```



◆ 有 CMEM 以及其他的内存分配器所分配的物理地址参与的 GE2D 操作。
我们下面以 CMEM 为例来进行说明，其他的内存分配器同理。
CMEM 等内存分配器的作用是:实现对从系统内存中申请的或者在启动之初预留的连续的物理内存进行管理，包括申请，释放，内存块链表管理，查找等等。所以针对这种类型内存的操作流程是。



● 单一的源到目的的操作。包括 ALLOC_OSD0, ALLOC_OSD1,ALLOC_ALLOC，几种形式。

➢ **ALLOC_OSD0**

Sample codes:

```
//这段示例代码描述把一幅解码得到的 JPEG 图片搬移到 OSD0
#include "cmemlib.h"
CMEM_AllocParams cmemParm = {CMEM_HEAP, CMEM_NONCACHED, 8};
unsigned char *planes[4] = {NULL, NULL, NULL,NULL};

CMEM_init();
//首先申请 YUV 地址空间。对应于 ALLOC_OSD0 中的 ALLOC
planes[0] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width) * info.height, &cmemParm);
planes[1] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width/2) * info.height/2, &cmemParm);
planes[2] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width/2) * info.height/2, &cmemParm);

config.addr_y = CMEM_getPhys(planes[0]);
config.addr_u = CMEM_getPhys(planes[1]);
```

```
config.addr_v = CMEM_getPhys(planes[2]);
```

//decoder 把 JPEG 图片解码得到的 YUV 数据送到啊 addr_y,addr_u addr_v 指定的地址空间。

```
……
```

//进行 GE2D 的位块移动操作。

```
ge2d_config.src_dst_type = ALLOC_OSD0;
ge2d_config.alu_const_color=0xff0000ff;
ge2d_config.src_format = GE2D_FORMAT_M24_YUV420;
ge2d_config.dst_format = GE2D_FORMAT_S32_ARGB;

ge2d_config.src_planes[0].addr = config.addr_y;
ge2d_config.src_planes[0].w = config.canvas_width;
ge2d_config.src_planes[0].h = config.dec_h;

ge2d_config.src_planes[1].addr = config.addr_u;
ge2d_config.src_planes[1].w = config.canvas_width/2;
ge2d_config.src_planes[1].h = config.dec_h / 2;

ge2d_config.src_planes[2].addr = config.addr_v;
ge2d_config.src_planes[2].w = config.canvas_width/2;
ge2d_config.src_planes[2].h = config.dec_h / 2;

ge2d_config.dst_planes[0].addr=CMEM_getPhys(planes[3]);
ge2d_config.dst_planes[0].w=config.canvas_width;
ge2d_config.dst_planes[0].h = config.dec_h;
ioctl(fd_ge2d,    GE2D_CONFIG, &ge2d_config);

op_para.src1_rect.x = image_start_x;
op_para.src1_rect.y = image_start_y;
op_para.src1_rect.w = image_w;
op_para.src1_rect.h = image_h;

op_para.dst_rect.x =    screen_pos_start_x;
op_para.dst_rect.y = screen_pos_start_y;
op_para.dst_rect.w = screen_pos_end_x - screen_pos_start_x ;
op_para.dst_rect.h = screen_pos_end_y - screen_pos_start_y;
ioctl(fd_ge2d, GE2D_STRETCHBLIT_NOALPHA, &op_para);
```

//释放从 CMEM 申请的内存资源。

```
if (planes[0])
        CMEM_free(planes[0], &cmemParm);
if (planes[1])
        CMEM_free(planes[1], &cmemParm);
if (planes[2])
        CMEM_free(planes[2], &cmemParm);
```

ALLOC 是不可见部分。并且在本例中是 multi plane 的方式存储。

- **ALLOC_ALLOC**

上面展示了 ALLOC_OSD0 的操作过程，对于 ALLOC_ALLOC 的操作过程类似，仅仅是需要两次从 CMEM 模块申请内存空间，而后在 GE2D config 的时候进行正确的配置即可。下面的例子展示把 JPEG 解码得到的 YUV 数据搬移到从 CMEM 申请的 RGB 空间

Sample codes :

```
//首先申请 YUV 地址空间。对应于 ALLOC_OSD0 中的 ALLOC
planes[0] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width) * info.height, &cmemParm);
planes[1] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width/2) * info.height/2, &cmemParm);
planes[2] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width/2) * info.height/2, &cmemParm);
//下面是申请的 RGB 空间
planes[3]= (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width) * info.height*4, &cmemParm);
config.addr_y = CMEM_getPhys(planes[0]);
config.addr_u = CMEM_getPhys(planes[1]);
config.addr_v = CMEM_getPhys(planes[2]);

//decoder 把 JPEG 图片解码得到的 YUV 数据送到啊 addr_y,addr_u addr_v 指定的地址空间。
……
//进行 GE2D 的位块移动操作。
ge2d_config.src_dst_type = ALLOC_ALLOC
ge2d_config.alu_const_color=0xff0000ff;
ge2d_config.src_format = GE2D_FORMAT_M24_YUV420;
ge2d_config.dst_format = GE2D_FORMAT_S32_ARGB;

ge2d_config.src_planes[0].addr = config.addr_y;
ge2d_config.src_planes[0].w = config.canvas_width;
ge2d_config.src_planes[0].h = config.dec_h;

ge2d_config.src_planes[1].addr = config.addr_u;
ge2d_config.src_planes[1].w = config.canvas_width/2;
ge2d_config.src_planes[1].h = config.dec_h / 2;
```

```
ge2d_config.src_planes[2].addr = config.addr_v;
ge2d_config.src_planes[2].w = config.canvas_width/2;
ge2d_config.src_planes[2].h = config.dec_h / 2;

ge2d_config.dst_planes[0].addr=CMEM_getPhys(planes[3]);
ge2d_config.dst_planes[0].w=config.canvas_width;
ge2d_config.dst_planes[0].h = config.dec_h;
ioctl(fd_ge2d,    GE2D_CONFIG, &ge2d_config);

op_para.src1_rect.x = image_start_x;
op_para.src1_rect.y = image_start_y;
op_para.src1_rect.w = image_w;
op_para.src1_rect.h = image_h;

op_para.dst_rect.x =    canvas_pos_start_x;
op_para.dst_rect.y = canvas_pos_start_y;
op_para.dst_rect.w = canvas_pos_end_x - canvas_pos_start_x ;
op_para.dst_rect.h = canvas_pos_end_y - canvas_pos_start_y;
ioctl(fd_ge2d, GE2D_STRETCHBLIT_NOALPHA, &op_para);
//释放从 CMEM 申请的内存资源。
if (planes[0])
        CMEM_free(planes[0], &cmemParm);
if (planes[1])
        CMEM_free(planes[1], &cmemParm);
if (planes[2])
        CMEM_free(planes[2], &cmemParm);
if (planes[3])
        CMEM_free(planes[3], &cmemParm);
```

在本例中左边的 ALLOC 是采用 multi plane 的 YUV 地址空间，而右边部分是 Single plane 的 RGB32 地址空间，同时两个都是不可见的。

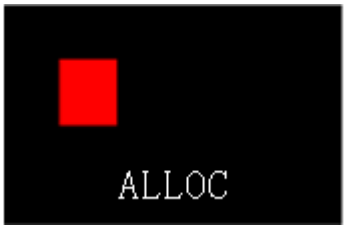
对于 ALLOC_ALLOC 也可以实现 ALLOC 空间的 FILL_RECT 操作，我们可以把 SRC 和 DST 指向同一个 ALLOC 空间，而后使用 ALLOC_ALLOC 进行配置,并执行 FILL_RECT 的动作。

Sample codes:

```
//首先申请 RGB ALLOC 空间
planes[0] = (unsigned char *)CMEM_alloc(0,
                CANVAS_ALIGNED(info.width) * info.height*4, &cmemParm);
```

```
ge2d_config.src_planes[0].addr = CMEM_getPhys(planes[0]);
ge2d_config.src_planes[0].w = info.width;
ge2d_config.src_planes[0].h = info.height;
ge2d_config.dst_planes[0].addr= ge2d_config.src_planes[0].addr;
ge2d_config.dst_planes[0].w=info.width;
ge2d_config.dst_planes[0].h = info.height;

ge2d_config.src_dst_type = ALLOC_ALLOC
ge2d_config.alu_const_color=0xff0000ff;
ge2d_config.src_format = GE2D_FORMAT_S32_ARGB;
ge2d_config.dst_format = GE2D_FORMAT_S32_ARGB;
ioctl(fd_ge2d,    GE2D_CONFIG, &ge2d_config);
op_para.src1_rect.x =0;
op_para.src1_rect.y = 0;
op_para.src1_rect.w = rect_w;
op_para.src1_rect.h = rect_h;
op_para.color=0xff0000ff; //color order :RGBA
ioctl(fd_ge2d, GE2D_FILLRECTANGLE, &op_para);
//释放从 CMEM 申请的内存资源。
if (planes[0])
        CMEM_free(planes[0], &cmemParm);
```



- **颜色空间的转换**

从 CMEM 申请的内存之间，以及申请的内存与 OSD 之间借助 BLIT 操作可以实现 YUV 与 RGB 之间的变换。包括 YUV → RGB，以及 RGB →YUV，上面 ALLOC_OSD0 已经演示了 YUV420 到 RGB 的变换，下面的代码将演示 RGB 到 YUV 420 的变换。最大的不同就是需要三次 BLIT 操作 。

Sample codes:

```
memset((char*)&ge2d_config_ex,0,sizeof(config_para_ex_t));
ge2d_config_ex.src_para.mem_type=CANVAS_OSD0;
ge2d_config_ex.src_para.format    = GE2D_FORMAT_S32_ARGB;

ge2d_config_ex.src_para.top = 0;
ge2d_config_ex.src_para.left = 0;
ge2d_config_ex.src_para.width=osd[0]->vinfo->xres;
ge2d_config_ex.src_para.height=osd[0]->vinfo->yres;
```

```
//src2
ge2d_config_ex.src2_para.mem_type=CANVAS_TYPE_INVALID;
ge2d_config_ex.dst_para.mem_type=CANVAS_ALLOC;

ge2d_config_ex.dst_para.format=GE2D_FORMAT_YUV|GE2D_FORMAT_S8_Y;
//clip region
ge2d_config_ex.dst_para.width=PLANE_WIDTH;
ge2d_config_ex.dst_para.height=PLANE_HEIGHT;
ge2d_config_ex.dst_para.left=0;
ge2d_config_ex.dst_para.top=0;
//plane
ge2d_config_ex.dst_planes[0].addr=addr_y;
ge2d_config_ex.dst_planes[0].w=PLANE_WIDTH;
ge2d_config_ex.dst_planes[0].h=PLANE_HEIGHT;

ioctl(osd[0]->ge2d_fd, GE2D_CONFIG_EX, &ge2d_config_ex);
memset((char*)&op_para,0,sizeof(ge2d_op_para_t));
op_para.src1_rect.x = 0;
op_para.src1_rect.y = 0;
op_para.src1_rect.w =    PLANE_WIDTH ;
op_para.src1_rect.h = PLANE_HEIGHT ;

op_para.dst_rect.x = 0;
op_para.dst_rect.y = 0;
op_para.dst_rect.w =PLANE_WIDTH;
op_para.dst_rect.h = PLANE_HEIGHT;
ioctl(osd[0]->ge2d_fd,GE2D_STRETCHBLIT_NOALPHA, &op_para);

ge2d_config_ex.dst_para.format=GE2D_FORMAT_YUV|GE2D_FORMAT_S8_CB;
//plane
ge2d_config_ex.dst_planes[0].addr=addr_u;
//对于YUV420    x_ratio=0.5    y_ratio=0.5
//        YUV422    x_ratio=0.5    y_ratio=1
//        YUV444    x_ratio=1      y_ratio=1
ge2d_config_ex.dst_planes[0].w=PLANE_WIDTH*x_ratio;
ge2d_config_ex.dst_planes[0].h=PLANE_HEIGHT*y_ratio;
ge2d_config_ex.dst_para.width=PLANE_WIDTH*x_ratio;
ge2d_config_ex.dst_para.height=PLANE_HEIGHT*y_ratio;
ioctl(osd[0]->ge2d_fd, GE2D_CONFIG_EX, &ge2d_config_ex);
memset((char*)&op_para,0,sizeof(ge2d_op_para_t));
op_para.src1_rect.x = 0;
op_para.src1_rect.y = 0;
op_para.src1_rect.w =    PLANE_WIDTH ;
op_para.src1_rect.h = PLANE_HEIGHT ;
```

```
op_para.dst_rect.x = 0;
op_para.dst_rect.y = 0;
op_para.dst_rect.w =PLANE_WIDTH*x_ratio;
op_para.dst_rect.h = PLANE_HEIGHT*y_ratio;
ioctl(osd[0]->ge2d_fd,GE2D_STRETCHBLIT_NOALPHA, &op_para);

ge2d_config_ex.dst_para.format=GE2D_FORMAT_YUV|GE2D_FORMAT_S8_CR;
//plane
ge2d_config_ex.dst_planes[0].addr=addr_v;
ge2d_config_ex.dst_planes[0].w=PLANE_WIDTH*x_ratio;
ge2d_config_ex.dst_planes[0].h=PLANE_HEIGHT*y_ratio;
ge2d_config_ex.dst_para.width=PLANE_WIDTH*x_ratio;
ge2d_config_ex.dst_para.height=PLANE_HEIGHT*y_ratio;
ioctl(osd[0]->ge2d_fd, GE2D_CONFIG_EX, &ge2d_config_ex);
memset((char*)&op_para,0,sizeof(ge2d_op_para_t));
op_para.src1_rect.x = 0;
op_para.src1_rect.y = 0;
op_para.src1_rect.w =    PLANE_WIDTH ;
op_para.src1_rect.h = PLANE_HEIGHT ;

op_para.dst_rect.x = 0;
op_para.dst_rect.y = 0;
op_para.dst_rect.w =PLANE_WIDTH*x_ratio;
op_para.dst_rect.h = PLANE_HEIGHT*y_ratio;
ioctl(osd[0]->ge2d_fd,GE2D_STRETCHBLIT_NOALPHA, &op_para);
```

- 两个源到目的的操作，

  只有BLEND操作需要两个源，blend 操作通过操作码可以实现各种复杂的效果，可以进行各种算术运算和逻辑操作，请参照Blend操作的操作码

  下面简述操作的方式：

  Cs source color

  Fs source color factor

  Cd destination color

  Fd destination color factor

  As source alpha

  AFs source alpha factor

  Ad destination alpha

  AFd destination alpha factor

  Cc **const color,参照基本配置alu_const_color和扩展配置alu_const_color**

  BLENDOP_ADD Cs*Fs + Cd*Fd

BLENDOP_SUB	Cs*Fs - Cd*Fd
REVERSE SUBTRACT	Cd*Fd - Cs*Fs
BLENDOP_MIN	min(Cs*Fs, Cd*Fd)
BLENDOP_MAX	max(Cs*Fs, Cd*Fd)
BLENDOP_LOGIC	Cs op Cd

Fs 可能的取值

COLOR_FACTOR_ZERO	**0**
COLOR_FACTOR_ONE	**1**
COLOR_FACTOR_SRC_COLOR	**Cs**
COLOR_FACTOR _ONE_MINUS_SRC_COLOR	**1 - Cs**
COLOR_FACTOR _DST_COLOR	**Cd**
COLOR_FACTOR _ONE_MINUS_DST_COLOR	**1 - Cd**
COLOR_FACTOR _RC_ALPHA	**As**
COLOR_FACTOR _ONE_MINUS_SRC_ALPHA	**1 – As**
COLOR_FACTOR _DST_ALPHA	**Ad**
COLOR_FACTOR _ONE_MINUS_DST_ALPHA	**1 – Ad**
COLOR_FACTOR _CONST_COLOR	**Cc**
COLOR_FACTOR _ONE_MINUS_CONST_COLOR	**1 - Cc**
COLOR_FACTOR _CONST_ALPHA	**Ac**
COLOR_FACTOR _ONE_MINUS_CONST_ALPHA	**1 – Ac**
COLOR_FACTOR _SRC_ALPHA_SATURATE	**min(As,1-Ad)**

Fd 可能的取值

COLOR_FACTOR_ZERO	**0**
COLOR_FACTOR_ONE	**1**
COLOR_FACTOR_SRC_COLOR	**Cs(RGBs)**
COLOR_FACTOR_ONE_MINUS_SRC_COLOR	**1 - Cs(RGBs)**
COLOR_FACTOR_DST_COLOR	**Cd(RGBd)**
COLOR_FACTOR_ONE_MINUS_DST_COLOR	**1 - Cd(RGBd)**
COLOR_FACTOR_SRC_ALPHA	**As**
COLOR_FACTOR_ONE_MINUS_SRC_ALPHA	**1 – As**
COLOR_FACTOR_ DST_ALPHA	**Ad**
COLOR_FACTOR_ONE_MINUS_DST_ALPHA	**1 – Ad**
COLOR_FACTOR_CONST_COLOR	**Cc(RGBc)**
COLOR_FACTOR_ONE_MINUS_CONST_COLOR	**1 - Cc(RGBc)**
COLOR_FACTOR_CONST_ALPHA	**Ac**
COLOR_FACTOR_ONE_MINUS_CONST_ALPHA	**1 – Ac**
COLOR_FACTOR_SRC_ALPHA_SATURATE	**min(As,1-Ad)**

AFs，AFd 可能的取值

ALPHA_FACTOR_ZERO	**0**
ALPHA_FACTOR_ONE	**1**

ALPHA_FACTOR_SRC_ALPHA	**As**
ALPHA_FACTOR_ONE_MINUS_SRC_ALPHA	**1 - As**
ALPHA_FACTOR_DST_ALPHA	**Ad**
ALPHA_FACTOR_ONE_MINUS_DST_ALPHA	**1 – Ad**
ALPHA_FACTOR_CONST_ALPHA	**Ac**
ALPHA_FACTOR_ONE_MINUS_CONST_ALPHA	**1 – Ac**

下面的例子演示如下的效果

Cdst= Csrc1 *1 + Csrc2*0

Adst= Asrc1*0 + Asrc2*1

Sample codes:

```
memset((char*)&ge2d_config_ex,0,sizeof(config_para_ex_t));
ge2d_config_ex.src_para.mem_type=CANVAS_ALLOC;
ge2d_config_ex.src_para.format=ge2d_yuv_format;
//clip region
ge2d_config_ex.src_para.width=PLANE_WIDTH;
ge2d_config_ex.src_para.height=PLANE_HEIGHT;
ge2d_config_ex.src_para.left=0;
ge2d_config_ex.src_para.top=0;
//plane
ge2d_config_ex.src_planes[0].addr=addr_y;
ge2d_config_ex.src_planes[0].w=PLANE_WIDTH;
ge2d_config_ex.src_planes[0].h=PLANE_HEIGHT;
ge2d_config_ex.src_planes[1].addr=addr_u;
ge2d_config_ex.src_planes[1].w=PLANE_WIDTH*x_ratio;
ge2d_config_ex.src_planes[1].h=PLANE_HEIGHT*y_ratio;

ge2d_config_ex.src_planes[2].addr=addr_v;
ge2d_config_ex.src_planes[2].w=PLANE_WIDTH*x_ratio;
ge2d_config_ex.src_planes[2].h=PLANE_HEIGHT*y_ratio;

ge2d_config_ex.src2_para.mem_type=CANVAS_ALLOC;
ge2d_config_ex.src2_para.format=GE2D_FORMAT_S32_ARGB;
ge2d_config_ex.src2_planes[0].addr=addr_rgb;
ge2d_config_ex.src2_planes[0].w=PLANE_WIDTH;
ge2d_config_ex.src2_planes[0].h=PLANE_HEIGHT;
ge2d_config_ex.src2_para.width=PLANE_WIDTH;
ge2d_config_ex.src2_para.height=PLANE_HEIGHT;
ge2d_config_ex.src2_para.left=0;
ge2d_config_ex.src2_para.top=0;

ge2d_config_ex.dst_para.mem_type=CANVAS_OSD0;
ge2d_config_ex.dst_para.left=0;
```

```
ge2d_config_ex.dst_para.top=0;
ge2d_config_ex.dst_para.width=osd[0]->vinfo->xres;
ge2d_config_ex.dst_para.height=osd[0]->vinfo->yres;
ge2d_config_ex.dst_xy_swap=0;
ge2d_config_ex.src_para.x_rev = 0;
ge2d_config_ex.src_para.y_rev = 0;

ioctl(osd[0]->ge2d_fd, GE2D_CONFIG_EX, &ge2d_config_ex);
memset((char*)&op_para,0,sizeof(ge2d_op_para_t));
op_para.src1_rect.x = 0;
op_para.src1_rect.y = 0;
op_para.src1_rect.w =    PLANE_WIDTH;
op_para.src1_rect.h = PLANE_HEIGHT;

op_para.src2_rect.x = 0;
op_para.src2_rect.y = 0;
op_para.src2_rect.w =    PLANE_WIDTH;
op_para.src2_rect.h = PLANE_HEIGHT;

op_para.dst_rect.x = 0;
op_para.dst_rect.y = 0;
op_para.dst_rect.w =PLANE_WIDTH;
op_para.dst_rect.h = PLANE_HEIGHT;
op_para.op=blendop(
        OPERATION_ADD,
        /* color blending src factor */
        COLOR_FACTOR_ONE,
        /* color blending dst factor */
        COLOR_FACTOR_ZERO,
         /* alpha blending_mode */
         OPERATION_ADD,
         /* alpha blending src factor */
         ALPHA_FACTOR_ZERO,
        /* color blending dst factor */
         ALPHA_FACTOR_ONE);
ioctl(osd[0]->ge2d_fd,GE2D_BLEND, &op_para);
```