MI GFX API

Version 2.05

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	04/12/2018
2.04	• Removed MI_GFX_RopCode_e and MI_GFX_Yuv422_e • Added MI_Gfx_DfbBlendFlags_e • Updated MI_GFX_Opt_t	08/01/2019
2.05	• Add interface MI_GFX_SetPalette	03/11/2020

TABLE OF CONTENTS

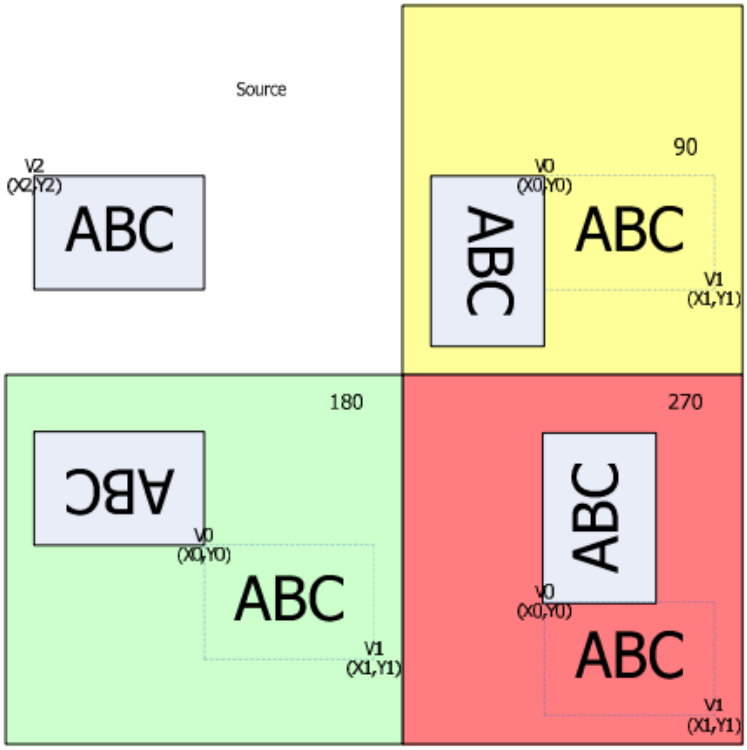
1. 概述

1.1. 模块说明

GFX（Graphic Engine）硬件为客户画 UI 提供快速的图形绘制功能，主要有矩形色彩填充、位图搬移（支持缩放、旋转、镜像翻转、格式转换、alpha 混合叠加、Color Key 等操作）

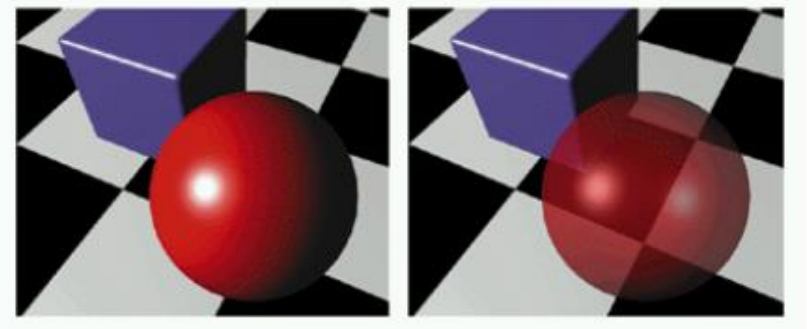
Rotate：HW 支持 90/180/270 度旋转

Rotate

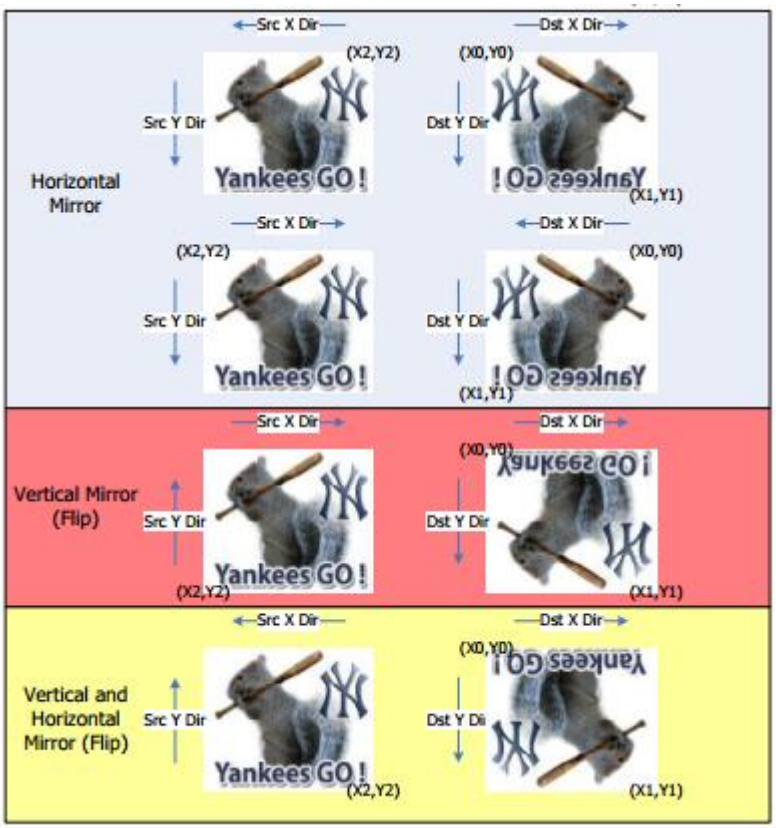


Alpha 混合叠加：支持正常的 2D alpha 操作

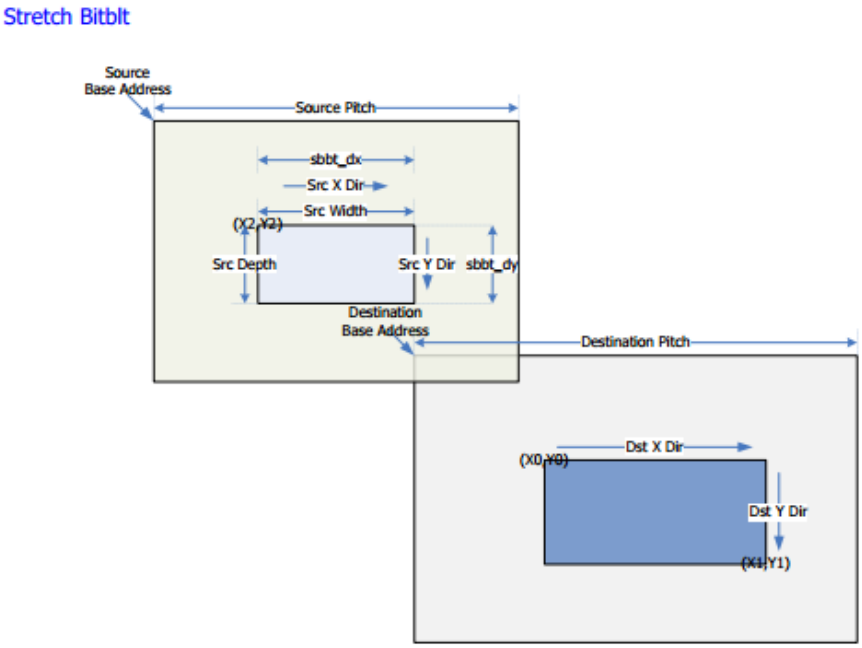
Alpha Blending



镜像：支持 H 镜像，V 镜像，HV 同时镜像

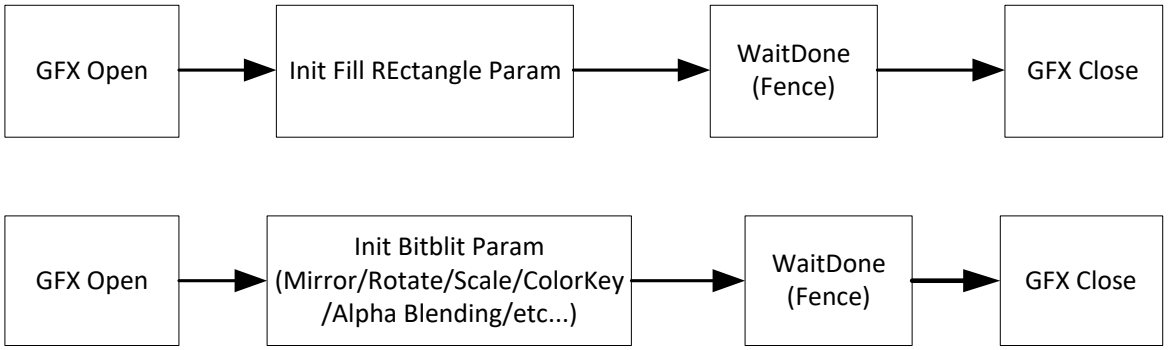


缩放：支持在 bitblit 的时候缩放

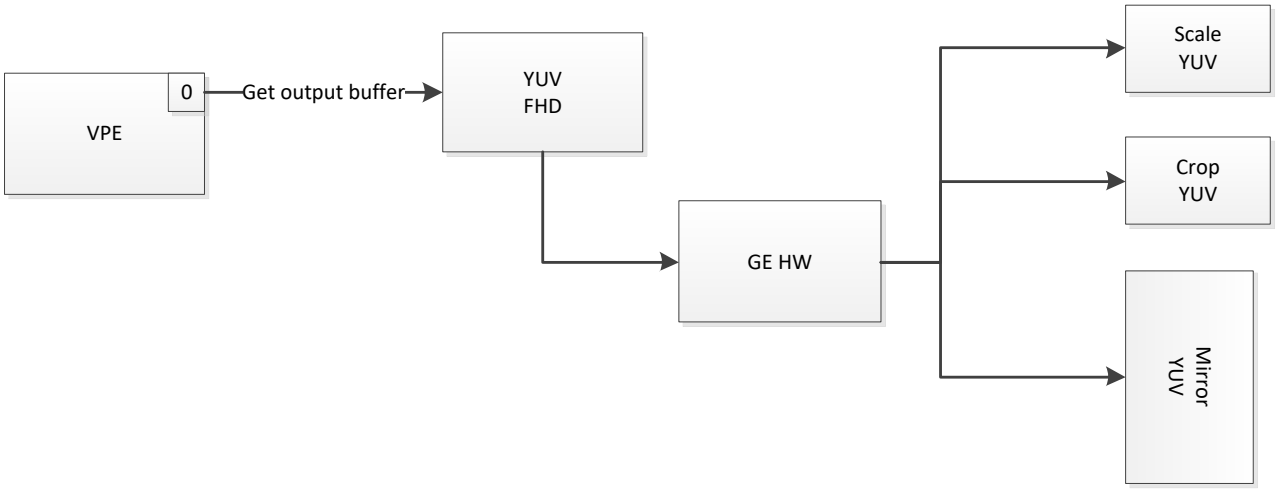


其他操作就不一一罗列，和标准的 2D 加速基本一致。

1.2. 调用流程框图



如果客户想用 GE 来处理视频或者 UI 数据都可以如下流程使用即可



1.3. 关键字说明

- Fence
  对应一个 GFX 的操作需要等待的状态，会阻塞等待，但是内部也会有超时机制用来防呆。
- Colorkey
  关键色，比如在做搬移的时候可以去掉位图中的某一个颜色，这就可以把这个颜色设置为关键色
- Pitch
  每行像素占的字节数,(像素位数/8) * Width，也对应常用的 Stride、LineLength

2. API 参考

该功能模块提供以下 API：

API名	功能
MI_GFX_Open	打开 GFX 设备
MI_GFX_Close	关闭 GFX 设备
MI_GFX_WaitAllDone	等待 GFX 的所有任务完成
MI_GFX_QuickFill	向任务中添加快速填充操作
MI_GFX_GetAlphaThresholdValue	获取 alpha 判决阈值
MI_GFX_SetAlphaThresholdValue	设置 alpha 判决阈值
MI_GFX_Bitblit	向任务中添加对光栅位图进行有附加功能的搬移操作
MI_GFX_SetPalette	设置索引颜色格式的调色板

2.1. MI_GFX_Open

- 功能

  调用此接口打开GFX 设备并初始化硬件设备。

- 语法

  MI_ S32 MI_GFX_Open(void);

- 形参

  无。

- 返回值

  返回值：
  - MI_SUCCESS 打开设备成功。
  - 初始化失败，详情参照错误码。

- 依赖

  头文件：mi_gfx.h
  库文件：libmi_gfx.a/libmi_gfx.so

※ 注意

1. 在进行 GFX相关操作前应该首先调用此接口，保证 GFX 设备处于打开状态。
2. GFX设备初始化一次后，再次调用将返回已初始化，可重复调用。

- 举例

```
1.  MI_S32 s32Ret = 0;
2.
3.  /* open GFX device*/
4.  s32Ret = MI_GFX_Open();
5.  if (MI_SUCCESS != s32Ret)
6.  {
7.          return -1;
8.  }
9.  /* close GFX device*/
10. MI_GFX_Close();
```

- 相关主题

  无

2.2. MI_GFX_Close

- 功能

  调用此接口关闭 GFX设备。

- 语法

  MI_ S32 MI_GFX_Close(void);

- 形参

  无。

- 返回值

  返回值：
  - MI_SUCCESS 成功销毁GFX模块。
  - 销毁GFX模块失败，详情参照错误码。

- 依赖

  头文件：mi_gfx.h
  库文件：libmi_gfx.a/libmi_gfx.so

※ 注意

- 调用 MI_GFX_Open 与 MI_GFX_Close 的次数需要对应，可以重复调用，Open的次数和Close的次数一致时，GFX的api将不再能调用。

- 举例

  无。

- 相关主题

2.3. MI_GFX_WaitAllDone

- 功能

  调用此接口等待 GFX 的所有任务完成或者等待指定的任务完成。

- 语法

  MI_S32 MI_GFX_WaitAllDone(MI_BOOL bWaitAllDone, MI_penU16 u16TargetFence);

- 形参

参数名称	描述	输入/输出
bWaitAllDone	是否等待所有GFX任务完成	输入
u16TargetFence	等待指定的fence完成，在bWaitAllDone为FALSE时才有效	输入

- 返回值

  返回值：
  - MI_SUCCESS 等待的GFX任务都已经完成。
  - 等待GFX任务完成失败，详情参照错误码。

- 依赖

  头文件：mi_gfx.h
  库文件：libmi_gfx.a/libmi_gfx.so

- 注意

  - 此接口为阻塞接口，会阻塞等待所有的 GFX 任务完成。

- 举例

  无。

2.4. MI_GFX_QuickFill

- 功能

  将数据值u32ColorVal填充到以pstDst 为目的地址、pstDstRect 为输出区域的内存中，可实现颜色填充的功能。

- 语法

```
MI_S32 MI_GFX_QuickFill(MI_GFX_Surface_t *pstDst, MI_GFX_Rect_t *pstDstRect,MI_U32 u32ColorVal, MI_U16 *pu16Fence);
```

- 形参

参数名称	描述	输入/输出
pstDst	目标位图。	输入
pstDstRect	目标位图操作区域。	输入
u32ColorVal	填充颜色值。	输入
pu16Fence	返回的Fence指针	输出

- 返回值

  返回值：
  - MI_SUCCESS 成功添加QuickFill行为的任务到HW队列。
  - 填充色块失败，详情参照错误码。

- 依赖

  头文件：mi_gfx.h
  库文件：libmi_gfx.a/libmi_gfx.so

- 注意

  - 由于该操作直接将u32ColorVal填充在位图的指定区域内，调用者欲填充蓝色到指定位图，应按照位图格式指定相应的蓝色填充值。
  - 如位图格式为 ARGB1555，欲填充为蓝色，则应指定 u32ColorVal为 0x801F（其中alpha 位为 1）。
  - u32ColorVal从高位到低位，依次是A8：R8：G8：B8，如果Dst的格式为YUV，则A8则无效，R对应Y，G对应U，B对应V。

➢ 举例

```
1.      ExecFunc(MI_SYS_Init(), MI_SUCCESS);
2.      FILE *fp = NULL;
3.      fp = fopen(SRC_FILE_NAME, "wb");
4.      ExecFunc(MI_SYS_MMA_Alloc("mma_heap_name0", SRC_WIDTH*SRC_HEIGHT*2, &phyAddr), MI_SUCCES
    S);
5.
6.      //MI_U32 phySrcAddr, phyDstAddr;
7.      ExecFunc(MI_GFX_Open(), MI_SUCCESS);
8.      ExecFunc(MI_SYS_Mmap(phyAddr, SRC_WIDTH*SRC_HEIGHT*2, &u64VirAddr , FALSE), MI_SUCCESS);

9.      memset(u64VirAddr, 0x22, SRC_WIDTH*SRC_HEIGHT*2);
10.
11.     //fillrect
12.     stSrc.eColorFmt = E_MI_GFX_FMT_ARGB1555;
13.     stSrc.u32Width = SRC_WIDTH;
14.     stSrc.u32Height = SRC_HEIGHT;
15.     stSrc.u32Stride = SRC_WIDTH * 2;
16.     stSrc.phyAddr = phyAddr;
17.
18.     stSrcRect.s32Xpos = 100;
19.     stSrcRect.s32Ypos = 100;
20.     stSrcRect.u32Width = 100;
21.     stSrcRect.u32Height = 100;
22.     ExecFunc(MI_GFX_QuickFill(&stSrc, &stSrcRect, u32ColorVal, &u16TargetFence), MI_SUCCESS)
    ;
23.     ExecFunc(MI_GFX_WaitAllDone(FALSE, u16TargetFence), MI_SUCCESS);
24.     if (NULL != fp)
25.     {
26.         fwrite(u64VirAddr, 1, SRC_WIDTH*SRC_HEIGHT*2, fp);
27.         fclose(fp);
28.         fp = NULL;
29.     }
30.     ExecFunc(MI_SYS_Munmap(u64VirAddr, SRC_WIDTH*SRC_HEIGHT*2), MI_SUCCESS);
31.     ExecFunc(MI_SYS_MMA_Free(phyAddr), MI_SUCCESS);
32.
33. ExecFunc(MI_GFX_Close(), MI_SUCCESS);
34. ExecFunc(MI_SYS_Exit(), MI_SUCCESS);
```

➢ 相关主题

无。

2.5. MI_GFX_GetAlphaThresholdValue

➢ 功能

获取 alpha 判决阈值。用于结果图片像素格式为 ARGB1555 的情况。若前景位图和背景位图的 alpha 运算结果小于此阈值，结果像素的 alpha 位取 0；大于或等于此阈值，像素的 alpha 位取 1。

➢ 语法

MI_S32 MI_GFX_GetAlphaThresholdValue(MI_U8 *pu8ThresholdValue);

➢ 形参

参数名称	描述	输入/输出
pu8ThresholdValue	指向 alpha 判决阈值的指针	输出

- 返回值

返回值
- MI_SUCCESS 成功获取到alpha阈值。
- 获取alpha阈值失败，详情参照错误码。

- 依赖

头文件：mi_gfx.h
库文件：libmi_gfx.a/libmi_gfx.so

- 注意

暂不支持。

- 举例

无。

- 相关主题

无。

2.6. MI_GFX_SetAlphaThresholdValue

- 功能

设置 alpha 判决阈值。当前景和背景做 bitblit 操作时，不管前景位图和背景位图的格式为什么，硬件都会生成 ARGB8888 的中间位图格式，若目标图片象素格式为 ARGB1555 的情况，则若前景位图和背景位图的 alpha 运算结果小于此阈值，结果象素的 alpha 位取0；大于或等于此阈值，象素的 alpha 位取 1。

- 语法

MI_S32 MI_GFX_SetAlphaThresholdValue(MI_U8 u8ThresholdValue);

- 形参

数名称	描述	输入/输出
u8ThresholdValue	alpha 判决阈值	输入

- 返回值

返回值
- MI_SUCCESS 成功设置alpha阈值。
- 设置alpha阈值失败，详情参照错误码。

- 依赖

头文件：mi_gfx.h
库文件：libmi_gfx.a/libmi_gfx.so

➢ 注意

暂不支持。

➢ 举例

无。

2.7. MI_GFX_BitBlit

➢ 功能

- 将前景位图（pstSrc）与目标位图（pstDst）的指定区域（pstSrcRect、pstDstRect）进行运算，将运算后的位图拷贝到目标位图（pstDst）的指定区域（pstDstRect）中。

- MI_GFX_Opt_t 结构中存放有 GFX 运算功能的配置信息，如：是否作色键（colorkey）及 colorkey 的配置值；是否作区域裁减（clip 操作）及指定 clip 区域；是否镜像、是否进行 alpha 混合等信息。上述的操作可以同时使能。

➢ 语法

MI_S32 MI_GFX_BitBlit(MI_GFX_Surface_t *pstSrc, MI_GFX_Rect_t *pstSrcRect, MI_GFX_Surface_t *pstDst, MI_GFX_Rect_t *pstDstRect, MI_GFX_Opt_t *pstOpt, MI_U16 *pu16Fence);

➢ 形参

参数名称	描述	输入/输出
pstSrc	源位图。	输入
pstSrcRect	源位图操作区域。	输入
pstDst	目标位图。	输入
pstDstRect	目标位图操作区域。	输入
pstOpt	运算参数设置结构。	输入
pu16Fence	返回的Fence指针	输出

➢ 返回值

返回值：
- MI_SUCCESS 初始化成功。
- 返回失败，详情参照错误码。

➢ 依赖

头文件：mi_gfx.h
库文件：libmi_gfx.a/libmi_gfx.so

➢ 注意

- 在调用此接口前应保证调用 MI_GFX_Open 打开 GFX 设备，然后调用对应Fill或者Bitblit操作。
- 目标位图必须与背景位图的颜色空间一致，前景位图的颜色空间可以与背景/目标位图不一致，这种情况下会进行颜色空间转换功能。
- 当前景源位图与目标位图尺寸不一致时，如果设置了缩放则按照设定的区域进行缩放，否则按照设置公共区域的最小值进行裁减搬移。

- 当只需要使用单源搬移操作时（比如只对源位图进行 ROP 取非操作），可以将背景或背景位图的结构信息和操作区域结构指针设置为空。
- Mirror/Flip的同时禁止进行缩放。
- clip 操作时若为区域内 clip，则裁减区域必须与操作区域有公共交集，否则会返回错误。
- 若为区域外 clip，则裁减区域不可完全覆盖操作区域，否则会返回错误码。也就是说，实际更新区域不能为空。

➢ 举例

```
1.  ExecFunc(MI_SYS_Init(), MI_SUCCESS);
2.      FILE *fp = NULL;
3.      FILE *dstfp = NULL;
4.      fp = fopen(SRC_FILE_NAME, "wb");
5.      dstfp = fopen(DST_FILE_NAME, "wb");
6.      ExecFunc(MI_SYS_MMA_Alloc("mma_heap_name0", SRC_WIDTH*SRC_HEIGHT*2, &phyAddr), MI_SUCCES
    S);
7.      ExecFunc(MI_SYS_MMA_Alloc("mma_heap_name0", DST_WIDTH*360*2, &phyAddr2), MI_SUCCESS);
8.
9.      //MI_U32 phySrcAddr, phyDstAddr;
10.     ExecFunc(MI_GFX_Open(), MI_SUCCESS);
11.     ExecFunc(MI_SYS_Mmap(phyAddr, SRC_WIDTH*SRC_HEIGHT*2, &u64VirAddr , FALSE), MI_SUCCESS);

12.     memset(u64VirAddr, 0x22, SRC_WIDTH*SRC_HEIGHT*2);
13.
14.     ExecFunc(MI_SYS_Mmap(phyAddr2, DST_WIDTH*DST_HEIGHT*2, &u64VirAddr2 , FALSE), MI_SUCCESS
    );
15.     memset(u64VirAddr2, 0x0F, DST_WIDTH*DST_HEIGHT*2);
16.
17.     //bitblit
18.     stSrc.eColorFmt = E_MI_GFX_FMT_ARGB1555;
19.     stSrc.u32Width = SRC_WIDTH;
20.     stSrc.u32Height = SRC_HEIGHT;
21.     stSrc.u32Stride = SRC_WIDTH * 2;
22.     stSrc.phyAddr = phyAddr;
23.
24.     stSrcRect.s32Xpos = 100;
25.     stSrcRect.s32Ypos = 100;
26.     stSrcRect.u32Width = 300;
27.     stSrcRect.u32Height = 300;
28.
29.     stDst.eColorFmt = E_MI_GFX_FMT_ARGB1555;
30.     stDst.u32Width = DST_WIDTH;
31.     stDst.u32Height = DST_HEIGHT;
32.     stDst.u32Stride = DST_WIDTH * 2;
33.     stDst.phyAddr = phyAddr2;
34.
35.     stDstRect.s32Xpos = 200;
36.     stDstRect.s32Ypos = 100;
37.     stDstRect.u32Width = 200;
38.     stDstRect.u32Height = 100;
39.     ExecFunc(MI_GFX_BitBlit(&stSrc, &stSrcRect, &stDst, &stDstRect, NULL, &u16TargetFence),
    MI_SUCCESS);
40.     ExecFunc(MI_GFX_WaitAllDone(FALSE, u16TargetFence), MI_SUCCESS);
41.     if (NULL != dstfp)
42.     {
43.         fwrite(u64VirAddr2, 1, DST_WIDTH*DST_HEIGHT*2, dstfp);
44.         fclose(dstfp);
45.         dstfp = NULL;
46.     }
47.
48.     ExecFunc(MI_SYS_Munmap(u64VirAddr, SRC_WIDTH*SRC_HEIGHT*2), MI_SUCCESS);
```

```
49.     ExecFunc(MI_SYS_MMA_Free(phyAddr), MI_SUCCESS);
50.
51.     ExecFunc(MI_SYS_Munmap(u64VirAddr2, DST_WIDTH*DST_HEIGHT*2), MI_SUCCESS);
52.     ExecFunc(MI_SYS_MMA_Free(phyAddr2), MI_SUCCESS);
53.
54.     ExecFunc(MI_GFX_Close(), MI_SUCCESS);
```

2.8. MI_GFX_SetPalette

- 功能

  为索引颜色格式(I2/I4/I8)设置调色板

- 语法

  MI_S32 MI_GFX_SetPalette(MI_GFX_ColorFmt_e eColorFmt, MI_GFX_Palette_t* pstPalette);

- 形参

参数名称	描述	输入/输出
eColorFmt	调色板对应的颜色格式: MI_GFX_ColorFmt_e。	输入
pstPalette	调色板数据。	输入

- 返回值

  返回值：
  - MI_SUCCESS 初始化成功。
  - 返回失败，详情参照错误码。

- 依赖

  头文件：mi_gfx.h
  库文件：libmi_gfx.a/libmi_gfx.so

- 注意

  pstPalette 是一个容量为 256 的 MI_GFX_PaletteEntry_t 数组,每一个 MI_GFX_PaletteEntry_t 代表一种颜色,对应在数组中的下标代表了颜色的索引。

➢ 举例

```
MI_GFX_Palette_t myPal;
MI_GFX_PaletteEntry_t index_0;
MI_GFX_PaletteEntry_t index_1;
index_0.RGB.u8A = 0x40;
index_0.RGB.u8R = 0xFF;
index_0.RGB.u8G = 0;
index_0.RGB.u8B = 0;
index_1.RGB.u8A = 0x40;
index_1.RGB.u8R = 0;
index_1.RGB.u8G = 0xFF;
index_1.RGB.u8B = 0;

myPal.aunPalette[0] = index_0;
myPal.aunPalette[1] = index_1;
myPal.u16PalStart = 0
myPal.u16PalEnd = 1;
MI_GFX_SetPalette(E_MI_GFX_FMT_I8,&myPal);
```

3. GFX 数据类型

GFX 相关数据类型：

数据类型	说明
MI_GFX_ColorFmt_e	GFX对应surface color format
MI_GFX_Rect_t	操作区域属性
MI_GFX_RopCode_e	GFX 支持的 ROP 操作类型
MI_GFX_ColorKeyOp_e	Colorkey 模式
MI_GFX_ColorKeyValue_t	Colorkey关键色属性
MI_GFX_DfbBldOp_e	用户自定义 alpha 混合模式
MI_GFX_Mirror_e	图像镜像属性
MI_GFX_Surface_t	位图 surface 结构体
MI_GFX_Opt_t	GFX 操作附加属性结构体
MI_GFX_ColorKeyValue_t	Colorkey的颜色结构体
MI_GFX_ColorKeyInfo_t	Colorkey的操作结构体（针对位图）
MI_GFX_PaletteEntry_t	调色板单个颜色定义联合体
MI_GFX_Palette_t	调色板定义结构体

3.1. MI_GFX_ColorFmt_e

➢ 说明

GFX 支持的颜色格式

➢ 定义

```
typedef enum
{
    E_MI_GFX_FMT_I1 = 0, /* ColorFormat */
    E_MI_GFX_FMT_I2,
    E_MI_GFX_FMT_I4,
    E_MI_GFX_FMT_I8,
    E_MI_GFX_FMT_FABAFGBG2266,
    E_MI_GFX_FMT_1ABFGBG12355,
    E_MI_GFX_FMT_RGB565,
    E_MI_GFX_FMT_ARGB1555,
    E_MI_GFX_FMT_ARGB4444,
    E_MI_GFX_FMT_ARGB1555_DST,
    E_MI_GFX_FMT_YUV422,
    E_MI_GFX_FMT_ARGB8888,
    E_MI_GFX_FMT_RGBA5551,
    E_MI_GFX_FMT_RGBA4444,
    E_MI_GFX_FMT_ABGR8888,
    E_MI_GFX_FMT_BGRA5551,
    E_MI_GFX_FMT_ABGR1555,
    E_MI_GFX_FMT_ABGR4444,
    E_MI_GFX_FMT_BGRA4444,
    E_MI_GFX_FMT_BGR565,
    E_MI_GFX_FMT_RGBA8888,
    E_MI_GFX_FMT_BGRA8888,
    E_MI_GFX_FMT_MAX
} MI_GFX_ColorFmt_e;
```

【注】

对应的 surface 支持的 color format，在 bitblt 操作时，对应的 SrcSurface、DstSurface 的颜色格式。

3.2. MI_GFX_Rect_t

➢ 说明

GFX 操作区域属性。

➢ 定义

```
typedef struct MI_GFX_Rect_s
{
    MI_S32 s32Xpos;
    MI_S32 s32Ypos;
    MI_U32 u32Width;
    MI_U32 u32Height;
} MI_GFX_Rect_t;
```

➢ 成员

成员	描述
s32Xpos	操作区域的起始横坐标，以像素数为单位。有效范围[0, 位图宽度）
s32Ypos	操作区域的起始纵坐标，以像素数为单位。有效范围[0, 位图高度）
u32Width	操作区域的宽度，以像素数为单位。有效范围(0,0xFFF]
u32Height	操作区域的高度，以像素数为单位。有效范围(0,0xFFF]

※ 注意事项

- 若操作区域与位图部分相交，则取相交部分为实际参与操作的区域；若操作区域与位图不相交，则返回相应错误码。
- 操作范围必须在SurFace范围内，否则将无任何操作。

➢ 相关数据类型及接口

无。

3.3. MI_GFX_ColorKeyOp_e

➢ 说明

GFX colorkey 操作模式。

➢ 定义

```
typedef enum
{
    E_MI_GFX_RGB_OP_EQUAL = 0,
    E_MI_GFX_RGB_OP_NOT_EQUAL,
    E_MI_GFX_ALPHA_OP_EQUAL,
    E_MI_GFX_ALPHA_OP_NOT_EQUAL,
    E_MI_GFX_ARGB_OP_EQUAL,
    E_MI_GFX_ARGB_OP_NOT_EQUAL,
    E_MI_GFX_CKEY_OP_BUTT,
} MI_GFX_ColorKey_e;
```

➢ 成员

成员	描述
E_MI_GFX_RGB_OP_EQUAL	RGB 相等做 colorkey 操作
E_MI_GFX_RGB_OP_NOT_EQUAL	RGB 不相等做 colorkey 操作
E_MI_GFX_ ALPHA_OP_EQUAL	ALPHA 相等做 colorkey 操作
E_MI_GFX_ ALPHA_OP_NOT_EQUAL	ALPHA 不相等做 colorkey 操作
E_MI_GFX_ ARGB_OP_EQUAL	RGB 相等做 colorkey 操作
E_MI_GFX_ARGB_OP_NOT_EQUAL	RGB 相等做 colorkey 操作
E_MI_GFX_CKEY_OP_BUTT	无效的 colorkey 操作

※ 注意事项

- 对位图进行 colorkey 时，可选择前景、背景然后可以选择key值相等或者不等。

➢ 相关数据类型及接口

无。

3.4. MI_GFX_Rotate_e

➢ 说明

支持旋转角度操作

➢ 定义

```
typedef enum
{
    E_MI_GFX_ROTATE_0   = 0，
    E_MI_GFX_ROTATE_90，
    E_MI_GFX_ROTATE_180，
    E_MI_GFX_ROTATE_270
} MI_GFX_Rotate_e;
```

➢ 成员

无

※ 注意事项

无

➢ 相关数据类型及接口

3.5. MI_GFX_ColorKeyValue_t

➢ 说明

对应设置的 colorkey 的颜色值

➢ 定义

```
typedef struct MI_GFX_ColorKey_s
{
        MI_U32 u32ColorStart;
        MI_U32 u32ColorEnd;
} MI_GFX_ColorKeyValue_t;
```

➢ 成员

成员	描述
u32ColorStart	ColorKey颜色范围起始值
u32ColorEnd	ColorKey颜色范围结束值

※ 注意事项

当仅针对一个 color 值做 colorkey 操作时，将 u32ColorStart = u32ColorEnd = colorVal。

➢ 相关数据类型及接口

无。

3.6. MI_GFX_ColorKeyInfo_t

➢ 说明

对应设置的 colorkey 的颜色值

➢ 定义

```
typedef struct MI_GFX_ColorKeyInfo_s
{
    MI_BOOL bEnColorKey;
    MI_GFX_ColorKeyOp_e eCKeyOp;
    MI_GFX_ColorFmt_e    eCKeyFmt;
    MI_GFX_ColorKeyValue_t    stCKeyVal;
} MI_GFX_ColorKeyInfo_t;
```

➢ 成员

成员	描述
bEnColorKey	是否使能ColorKey操作
eCKeyOp	ColorKey操作模式
stCKeyVal	ColorKey颜色范围
eCKeyFmt	ColorKey的颜色格式

※ 注意事项

- Colorkey操作模式对Surface有效。

➢ 相关数据类型及接口

无。

3.7. MI_GFX_DfbBldOp_e

➢ 说明

alpha 混合模式操作。

➢ 定义

```
typedef enum
{
    E_MI_GFX_DFB_BLD_ZERO = 0,
    E_MI_GFX_DFB_BLD_ONE,
    E_MI_GFX_DFB_BLD_SRCCOLOR,
    E_MI_GFX_DFB_BLD_INVSRCCOLOR,
    E_MI_GFX_DFB_BLD_SRCALPHA,
    E_MI_GFX_DFB_BLD_INVSRCALPHA,
    E_MI_GFX_DFB_BLD_DESTALPHA,
    E_MI_GFX_DFB_BLD_INVDESTALPHA,
    E_MI_GFX_DFB_BLD_DESTCOLOR,
    E_MI_GFX_DFB_BLD_INVDESTCOLOR,
    E_MI_GFX_DFB_BLD_SRCALPHASAT,
    E_MI_GFX_DFB_BLD_NONE,
    E_MI_GFX_DFB_BLD_MAX,
} MI_GFX_DfbBldOp_e;
```

➢ 成员

成员	描述
E_MI_GFX_DFB_BLD_OP_ZERO	0
E_MI_GFX_DFB_BLD_OP_ONE	1
E_MI_GFX_DFB_BLD_OP_SRCCOLOR	取 sc
E_MI_GFX_DFB_BLD_OP_INVSRCCOLOR	取 1-sc
E_MI_GFX_DFB_BLD_OP_SRCALPHA	sa
E_MI_GFX_DFB_BLD_OP_INVSRCALPHA	1-sa
E_MI_GFX_DFB_BLD_OP_DESTCOLOR	dc
E_MI_GFX_DFB_BLD_OP_INVDESTCOLOR	1-dc
E_MI_GFX_DFB_BLD_OP_DESTALPHA	da
E_MI_GFX_DFB_BLD_OP_INVDESTALPHA	1-da
E_MI_GFX_DFB_BLD_OP_SRCALPHASAT	min(1-da, sa)＋1
E_MI_GFX_BLD_MAX	无效的 alpha 混合模式

※ 注意事项

- 在前景位图和背景位图作叠加运算时，可以分别设置 Src1 通道和 Src2 通道的叠加模式。现在支持 11 种叠加模式。用 MI_GFX_DfbBldOp_e 设置模式。

➢ **相关数据类型及接口**

无

3.8. MI_GFX_Mirror_e

➢ 说明

图像镜像属性。

➢ 定义

```
typedef enum
{
    E_MI_GFX_MIRROR_NONE = 0,
    E_MI_GFX_MIRROR_HORIZONTAL,
    E_MI_GFX_MIRROR_VERTICAL,
    E_MI_GFX_MIRROR_BOTH,
    E_MI_GFX_MIRROR_MAX
} MI_GFX_Mirror_e;
```

➢ 成员

成员	描述
E_MI_GFX_MIRROR_NONE	输出图像不进行镜像操作
E_MI_GFX_MIRROR_HORIZONTAL	输出图像水平镜像
E_MI_GFX_MIRROR_VERTICAL	输出图像垂直镜像
E_MI_GFX_MIRROR_BOTH	输出图像水平+垂直镜像
E_MI_GFX_MIRROR_MAX	无效的镜像类型

※ 注意事项

无。

➢ 相关数据类型及接口

无。

3.9. MI_GFX_Surface_t

➢ 说明

位图 surface 结构体。

➢ 定义

```
typedef struct MI_GFX_Surface_s
{
    MI_PHY phyAddr;
    MI_GFX_ColorFmt_e eColorFmt;
    MI_U32 u32Width;
    MI_U32 u32Height;
    MI_U32 u32Stride;
} MI_GFX_Surface_t;
```

➢ 成员

成员	描述
u32PhyAddr	位图首地址。
eColorFmt	位图格式。
u32Height	位图高度。
u32Width	位图宽度。
u32Stride	位图跨度。

※ 注意事项

- 像素格式大于等于 Byte 的位图格式的位图首地址和 Stride 必须按照像素格式对齐，像素格式不足 Byte 的位图格式的位图首地址和 Stride 需要按照 Byte 对齐。
- 像素格式不足 Byte 的位图格式的水平起始位置和宽度必须按照像素对齐。
- YCbCr422 格式的位图的水平起始位置和宽度必须为偶数。

➢ 相关数据类型及接口

无。

3.10. MI_Gfx_DfbBlendFlags_e

➢ 说明

GFX blending 操作属性结构体。

➢ 定义

```
typedef enum
{
    E_MI_GFX_DFB_BLEND_NOFX = 0x00000000,
    E_MI_GFX_DFB_BLEND_COLORALPHA = 0x00000001,
    E_MI_GFX_DFB_BLEND_ALPHACHANNEL = 0x00000002,
    E_MI_GFX_DFB_BLEND_COLORIZE = 0x00000004,
    E_MI_GFX_DFB_BLEND_SRC_PREMULTIPLY = 0x00000008,
    E_MI_GFX_DFB_BLEND_SRC_PREMULTCOLOR = 0x00000010,
    E_MI_GFX_DFB_BLEND_DST_PREMULTIPLY = 0x00000020,
    E_MI_GFX_DFB_BLEND_XOR = 0x00000040,
    E_MI_GFX_DFB_BLEND_DEMULTIPLY = 0x00000080,
    E_MI_GFX_DFB_BLEND_SRC_COLORKEY = 0x00000100,
    E_MI_GFX_DFB_BLEND_DST_COLORKEY = 0x00000200,
    E_MI_GFX_DFB_BLEND_MAX = 0x3FF
} MI_Gfx_DfbBlendFlags_e;
```

➢ 成员

成员	描述
E_MI_GFX_DFB_BLEND_NOFX	没有任何BLD操作
E_MI_GFX_DFB_BLEND_COLORALPHA	src alpha会根据const color value运算 src.a = const color.a;
E_MI_GFX_DFB_BLEND_ALPHACHANNEL	src alpha会根据const color value运算 src.a = src.a * const color.a;
E_MI_GFX_DFB_BLEND_COLORIZE	将const color作为全局颜色，调整源颜色
E_MI_GFX_DFB_BLEND_SRC_PREMULTIPLY	使用源alpha预乘源颜色 src.r = src.r * src.a;等
E_MI_GFX_DFB_BLEND_SRC_PREMULTCOLOR	使用const alpha预乘源颜色 src.r = src.r * const.a;等
E_MI_GFX_DFB_BLEND_DST_PREMULTIPLY	使用目的alpha预乘目的颜色 dst.r = dst.r * dst.a;等
E_MI_GFX_DFB_BLEND_XOR	调节和预乘alpha src.a ^= dst.a;
E_MI_GFX_DFB_BLEND_DEMULTIPLY	预乘到非预乘的转换，最好不要使用这样的操作 x.r = ((int)x.r << 8) / ((int)x.a + 1);等
E_MI_GFX_DFB_BLEND_SRC_COLORKEY	如果src的颜色等于const color，则会被key掉
E_MI_GFX_DFB_BLEND_DST_COLORKEY	如果dst的颜色等于const color，则会被key掉
E_MI_GFX_DFB_BLEND_MAX	上面所有的action都会被使能

3.11. MI_GFX_Opt_t

➢ 说明

GFX 操作属性结构体。

➢ 定义

```
ty  typedef struct MI_GFX_Opt_s
{
        MI_GFX_Rect_t stClipRect;
        MI_GFX_ColorKeyInfo_t stSrcColorKeyInfo;
        MI_GFX_ColorKeyInfo_t stDstColorKeyInfo;
        MI_GFX_DfbBldOp_e eSrcDfbBldOp;
        MI_GFX_DfbBldOp_e eDstDfbBldOp;
        MI_GFX_Mirror_e eMirror;
        MI_GFX_Rotate_e eRotate;
 MI_Gfx_DfbBlendFlags_e eDFBBlendFlag;
        MI_U32 u32GlobalSrcConstColor;
        MI_U32 u32GlobalDstConstColor;
} MI_GFX_Opt_t;
```

➢ 成员

成员	描述
stSrcColorKeyInfo	Src surface colorkey 操作
stDstColorKeyInfo	Dst surface colorkey操作
stClipRect	clip 区域定义
eMirror	镜像类型
eSrcDfbBldOp	Src Surface BLEND操作类型
eDstDfbBldOp	Dst Surface BLEND操作类型
eRotate	支持旋转角度，0、90、180、270度旋转
eDFBBlendFlag	Blending的相关操作，如XOR、AlphaBlending、Colorize等操作
u32GlobalSrcConstColor	针对eDFBBlendFlag 操作对应的const参数，eDFBBlendFlag 操作有对应的mutli等等
u32GlobalDstConstColor	针对eDFBBlendFlag 操作对应的const参数，eDFBBlendFlag 操作有对应的mutli等等

※ 注意事项

目前仅有 u32GlobalSrcConstColor 有效。

➢ 相关数据类型及接口

无。

3.12. MI_GFX_PaletteEntry_t

➢ 说明

索引颜色格式对应的 RGB 颜色定义

➢ 定义

```
typedef union
{
    /// ARGB8888 byte order
    struct
    {
        MI_U8 u8A;
        MI_U8 u8R;
        MI_U8 u8G;
        MI_U8 u8B;
    } RGB;
    // u8Data[0] = u8A
    // u8Data[1] = u8R
    // u8Data[2] = u8G
    // u8Data[3] = u8B
    MI_U8 u8Data[4];
} MI_GFX_PaletteEntry_t;
```

➢ 成员

成员	描述
RGB	索引颜色对应的RGB颜色的分量结构体表示
u8Data	索引颜色对应的RGB颜色的字节序表示

※ 注意事项

注意定义中注释对颜色定义的描述，以便正确填写调色板数据。

➢ 相关数据类型及结果

无

3.13. MI_GFX_Palette_t

➢ 说明

调色板定义结构体

➢ 定义

```
typedef struct MI_GFX_Palette_s
{
    MI_GFX_PaletteEntry_t aunPalette[256];
    MI_U16 u16PalStart;
    MI_U16 u16PalEnd;
}MI_GFX_Palette_t;
```

➢ 成员

成员	描述
aunPalette	调色板索引对应的RGB颜色数组
u16PalStart	需要配置RGB颜色的索引集起始索引
u16PalEnd	需要配置RGB颜色的索引集结束索引

※ 注意事项

无

➢ 相关数据类型及结果

MI_GFX_PaletteEntry_t

4. GFX 错误码

GFX API 错误码如下所示

错误代码	宏定义	描述
0xA00D2003	MI_ERR_GFX_INVALID_PARAM	非法参数
0xA00D2002	MI_ERR_GFX_DEV_BUSY	GFX设备忙
0xA00D2220	MI_ERR_GFX_NOT_INIT	设备未初始化
0xA00D2221	MI_ERR_GFX_DRV_NOT_SUPPORT	不支持的操作类型
0xA00D2222	MI_ERR_GFX_DRV_FAIL_FORMAT	不支持的格式类型
0xA00D2223	MI_ERR_GFX_NON_ALIGN_ADDRESS	地址未对齐
0xA00D2224	MI_ERR_GFX_NON_ALIGN_PITCH	Pitch对齐错误
0xA00D2225	MI_ERR_GFX_DRV_FAIL_OVERLAP	Option内存有叠加区域
0xA00D2226	MI_ERR_GFX_DRV_FAIL_STRETCH	缩放失败
0xA00D2228	MI_ERR_GFX_DRV_FAIL_LOCKED	设备被其他使用者锁住
0xA00D2229	MI_ERR_GFX_DRV_FAIL_BLTADDR	Bitblit起始地址错误