Novatek HDAL Design Specification - hd_gfx

Table of Content

Difference Table (for IPC only)

Item	NT9668X	NT9852X
hd_gfx_add_job()	Not supported.	supported
hd_gfx_affine()	Not supported.	supported
palette of HD_GFX_IMG_BUF	Not supported.	supported
engine of HD_GFX_COPY	Not supported.	supported
engine of HD_GFX_SCALE	Not supported.	supported
engine of HD_GFX_ROTATE	Not supported.	supported
engine of HD_GFX_COLOR_TRANSFORM	Not supported.	supported
engine of HD_GFX_DRAW_LINE	Not supported.	supported
engine of HD_GFX_ARITHMETIC	Not supported.	supported

1 Introduction

hd_gfx aims to provide user-mode application with a CPU friendly manner to edit images. The editing operations are listed below:

1. copy
2. scale
3. rotate
4. color transform
5. draw line
6. draw rectangle

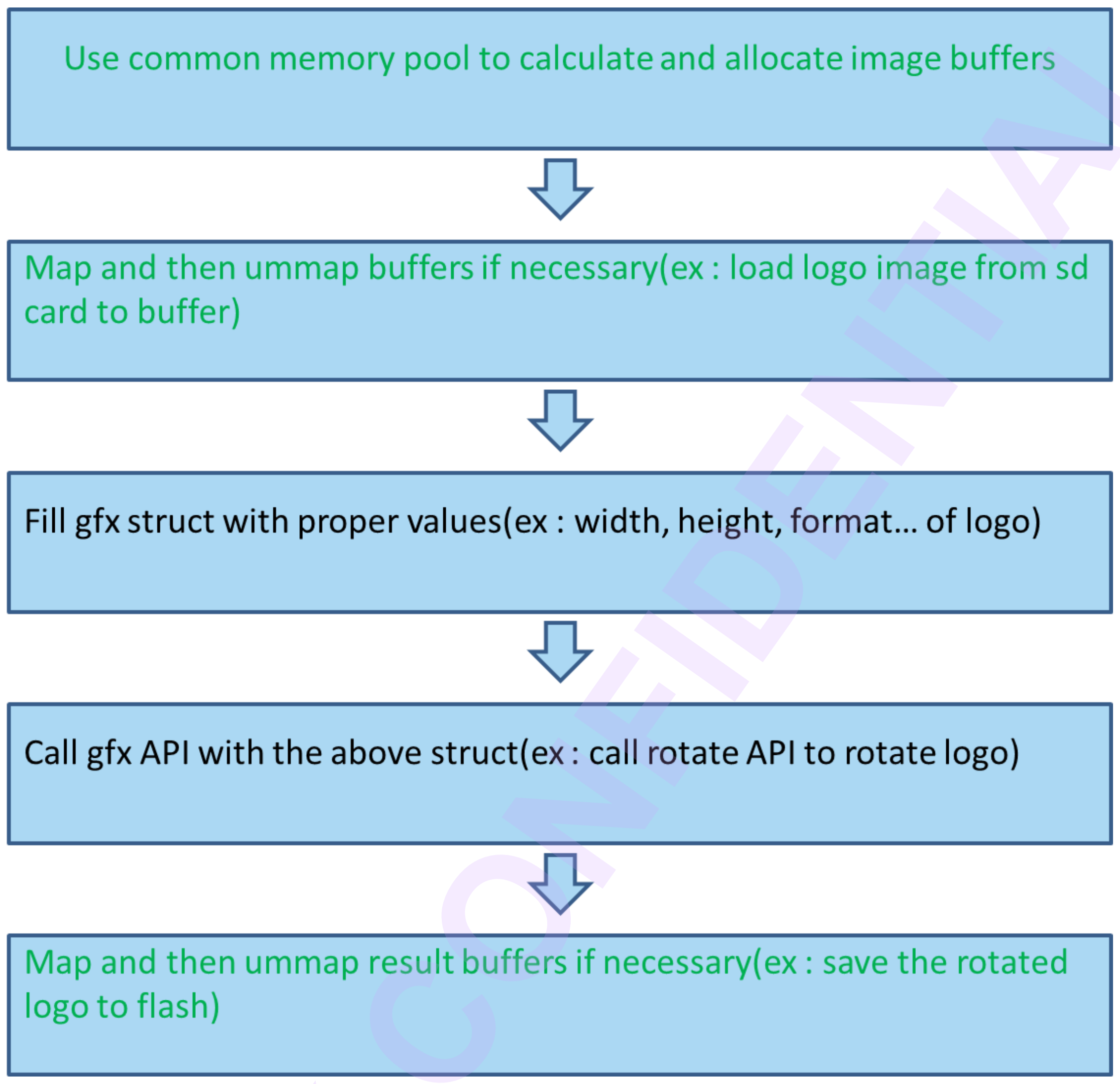
Draw line and draw rectangle have job/list counterparts which further reduces CPU loading by deliver a batch of operations to hardware engines in a single function call. This mechanism reduces number of switch between user-mode and kernel mode. Interrupt number is also reduced.

For IPCam, gfx competes for hardware resource with osg. Therefore, it's recommended not to use gfx intensively if osg is in effect. Gfx competes for hardware resource with below osg modules:

1. all ex osds of videoprocess
2. all ex osds of videoenc
3. all osds of videoout

1.1 Typical Flow

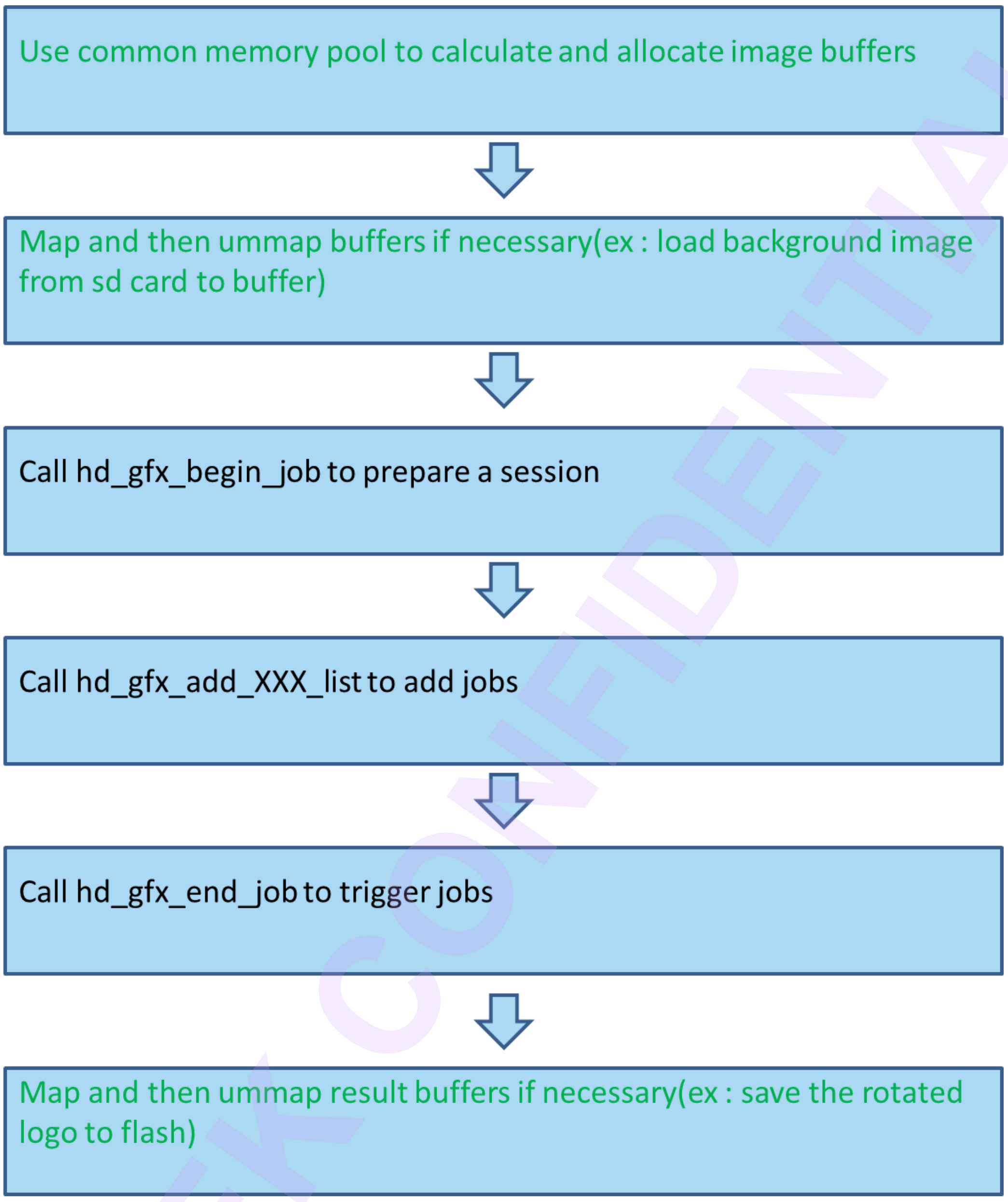
Texts in black are essential to gfx. Almost all gfx operations will need them. Texts in green are optional and typically related to other modules.


Use common memory pool to calculate and allocate image buffers
Map and then ummap buffers if necessary(ex : load logo image from sd card to buffer)
Fill gfx struct with proper values(ex : width, height, format… of logo)
Call gfx API with the above struct(ex : call rotate API to rotate logo)
Map and then ummap result buffers if necessary(ex : save the rotated logo to flash)

Below section presents details of all API and structure for text in black.

1.2 Job/list flow

Texts in black are essential to gfx. Almost all gfx operations will need them. Texts in green are optional and typically related to other modules.


Use common memory pool to calculate and allocate image buffers
Map and then ummap buffers if necessary(ex : load background image from sd card to buffer)
Call hd_gfx_begin_job to prepare a session
Call hd_gfx_add_XXX_list to add jobs
Call hd_gfx_end_job to trigger jobs
Map and then ummap result buffers if necessary(ex : save the rotated logo to flash)

Below section presents details of all API and structure for text in black.

2 Function and data structure definition

2.1 Function definition

2.1.1 hd_gfx_begin_job

[Description]
Prepare a job/list session

[Syntax]
HD_RESULT hd_gfx_begin_job (HD_GFX_HANDLE *p_handle)

[Parameter]

Value	Description
p_handle	output handle to identify a job

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.2 hd_gfx_add_job

[Description]
Add a job to a specified session. Adding a job wouldn't actually execute it. Instead, call hd_gfx_end_job to execute all added jobs in a session.

[Syntax]
HD_RESULT hd_gfx_add_job (HD_GFX_HANDLE handle, HD_GFX_OP op, void *p_param)

[Parameter]

Value	Description
handle	Which session to add job to
op	What kind of job to add (HD_GFX_OP_COPY, HD_GFX_OP_SCALE…)
p_param	pointer of parameters(HD_GFX_COPY for HD_GFX_OP_COPY, HD_GFX_SCALE for HD_GFX_OP_SCALE)

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.3 hd_gfx_cancel_job

[Description]
Remove all added jobs for a session

[Syntax]
HD_RESULT hd_gfx_cancel_job (HD_GFX_HANDLE handle)

[Parameter]

Value	Description
handle	handle to identify a job

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.4 hd_gfx_end_job

[Description]

Trigger all added jobs for a session. 8 jobs at top can be added in a trigger. This API blocks until all jobs finish.

[Syntax]

HD_RESULT hd_gfx_end_job (HD_GFX_HANDLE handle)

[Parameter]

Value	Description
handle	handle to identify a job

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.5 hd_gfx_add_draw_line_list

[Description]

Draw multiple lines on multiple images.

[Syntax]

HD_RESULT hd_gfx_add_draw_line_list (HD_GFX_HANDLE handle, HD_GFX_DRAW_LINE param[], UINT32 num)

[Parameter]

Value	Description
handle	handle to identify a job
param	array of parameters
num	number of array element

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.6 hd_gfx_add_draw_rect_list

[Description]
Draw multiple rectangles on multiple images

[Syntax]
HD_RESULT hd_gfx_add_draw_rect_list (HD_GFX_HANDLE handle, HD_GFX_DRAW_RECT param[], UINT32 num)

[Parameter]

Value	Description
handle	handle to identify a job
param	array of parameters
num	number of array element

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.7 hd_gfx_init

[Description]
Initialize all image engines, typically called during the init phase of HDAL

[Syntax]
HD_RESULT hd_gfx_init(void)

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.8 hd_gfx_uninit

[Description]
Uninitialize all image engines, typically called during the uninit phase of HDAL

[Syntax]
HD_RESULT hd_gfx_uninit (void)

[Parameter]

Value	Description
VOID	Not available

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure

2.1.9 hd_gfx_copy

[Description]
This API serves two purposes:

1. 2d memory copy
2. Image blending.

[Syntax]
HD_RESULT hd_gfx_copy (HD_GFX_COPY *p_param)

[Parameter]

Value	Description

p_param	pointer of parameters

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.10 hd_gfx_scale

[Description]

Scale up/down an image.

[Syntax]

HD_RESULT hd_gfx_scale(HD_GFX_SCALE *p_param)

[Parameter]

Value	Description
p_param	pointer of parameters

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.11 hd_gfx_rotate

[Description]

Rotate an image

[Syntax]

HD_RESULT hd_gfx_rotate (HD_GFX_ROTATE *p_param)

[Parameter]

Value	Description
p_param	pointer of parameters

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.12 hd_gfx_color_transform

[Description]
Transform pixel format

[Syntax]
HD_RESULT hd_gfx_color_transform (HD_GFX_COLOR_TRANSFORM *p_param)

[Parameter]

Value	Description
p_param	pointer of parameters

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.13 hd_gfx_draw_line

[Description]
Draw a line on an image

[Syntax]

HD_RESULT hd_gfx_draw_line (HD_GFX_DRAW_LINE *p_param)

[Parameter]

Value	Description
p_param	pointer of parameter

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.14 hd_gfx_draw_rect

[Description]

Draw a rectangle on an image

[Syntax]

HD_RESULT hd_gfx_draw_rect (HD_GFX_DRAW_RECT *p_param)

[Parameter]

Value	Description
p_param	pointer of parameter

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.15 hd_gfx_memcpy

[Description]

Use specified hardware to copy memory content. Therefore, cpu loading is relieved.

[Syntax]
void *hd_gfx_memcpy(UINT32 dest, const UINT32 src, size_t n)

[Parameter]

Value	Description
dest	Physical buffer address of destination
src	Physical buffer address of source
n	Number of bytes to copy

[Return Value]
In success, hd_gfx_memcpy returns dest. Other returned values indicate failure.

2.1.16 hd_gfx_arithmetic

[Description]
Use hardware to do bulk arithmetic calculations.

[Syntax]
HD_RESULT hd_gfx_arithmetic(HD_GFX_ARITHMETIC *p_param)

[Parameter]

Value	Description
p_param	pointer of parameter

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.1.17 hd_gfx_affine

[Description]
Apply affine effects(shear, displacement, rotation) on images

[Syntax]
HD_RESULT hd_gfx_affine(HD_GFX_AFFINE *p_param)

[Parameter]

Value	Description
p_param	pointer of parameter

[Return Value]

Value	Description
HD_OK	Success
HD_ERR_NG	Failure
HD_ERR_NOT_SUPPORT	Not support this parameter

2.2 Data structure definition

This chapter contains all data structures in hd_gfx:
1. HD_GFX_IMG_BUF
2. HD_GFX_COPY
3. HD_GFX_SCALE
4. HD_GFX_ROTATE
5. HD_GFX_COLOR_TRANSFORM
6. HD_GFX_DRAW_LINE
7. HD_GFX_DRAW_RECT

2.2.1 HD_GFX_IMG_BUF

[Description]
Structure for general image and buffer

[Parameter]

Value	Description
dim	width and height

format	yuv or argb and bit depth
p_phy_addr	physical buffer address of each plane
lineoffset	lineoffset of each buffer plane
palette	if image format is palette, assign palette information in this fields

[Note]
IPCam supports these formats:
1. yuv420
2. yuv422
3. rgb565
4. argb1555
5. argb4444

NVR/DVR supports these formats:
1. yuv422
2. rgb565
3. argb1555
4. argb4444
5. rgb888
6. argb8888
7. palette is not supported

2.2.2 HD_GFX_COPY

[Description]
Used by hd_gfx_copy

[Parameter]

Value	Description
src_img	config of source image
dst_img	config of destination image
src_region	which area of source image is copied
dst_pos	which position of destination image is source image copied to

colorkey	which color of source image should be transparent
alpha	transparent value
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]

For IPCam:

1. If source and destination images are of the same width, height and format, this API applies copy operation. Otherwise, blending operation is applied.
2. Copy operation supports argb1555/argb4444/rgb565/yuv420
3. Blending operation supports overlay an argb1555/argb4444/rgb565/yuv420 source image to a yuv420 destination image.
4. Colorkey is only valid for blending a rgb565 source image.
5. Alpha is only valid for blending an argb1555/rgb565 source image.
   For argb1555, alpha bits[7…4] is for foreground(pixels having A=1) and alpha bits[3…0] is for background(pixels having A=0). 16 steps are available for argb1555's foreground and background pixels.
   For argb4444, alpha is completely determined by pixel's 4 A bits. Alpha field of HD_GFX_COPY is ignored.
   For rgb565, alpha ranges from 0 to 255

For NVR/DVR:

1. If source and destination images are of the same width, height and format, this API applies copy operation. Otherwise, blending operation is applied.
2. Copy operation supports
3. Blending operation supports overlay a source image to a destination image in same format.
4. Colorkey is only valid for blending a rgb source image
5. Alpha is only valid for blending an argb source image.

2.2.3 HD_GFX_SCALE

[Description]

Used by hd_gfx_scale

[Parameter]

Value	Description
src_img	config of source image
dst_img	config of destination image
src_region	which area of source image is scaled
dst_region	which area of destination image is source image scaled to
quality	by which algorithm is the scale performed
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]

For IPCam:

1. argb1555/argb4444/rgb565/yuv420 are supported
2. Src_region and dst_region are not supported. Which means this API simply performs whole source image scale.
3. Max scale up/down factor is 16.
4. src_region is extracted before scale. So src_region's x, y, w and h are against src_img's coordinate.

For NVR/DVR:

1. Max scale up/down factor is 16.

2.2.4 HD_GFX_ROTATE

[Description]

Used by hd_gfx_rotate

[Parameter]

Value	Description
src_img	config of source image
dst_img	config of destination image
src_region	which area of source image is scaled

dst_pos	which position of destination image is source image scaled to
angle	rotation angle
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]
For IPCam:
1. Only yuv420 is supported
2. 90/180/270/mirror x/mirror y is supported.
3.

For NVR/DVR:
1. 90/180/270/mirror x/mirror y is supported.

2.2.5 HD_GFX_COLOR_TRANSFORM

[Description]
Used by hd_gfx_color_transform

[Parameter]

Value	Description
src_img	config of source image
dst_img	config of destination image
p_tmp_buf	temporary working buffer
tmp_buf_size	size of p_tmp_buf
p_lookup_table	color map between gray format to argb
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]
For IPCam:
1. Only transform between rgb888 and yuv420 is supported
2. tmp_buf_size is [image width * ALIGN_8(image height) * 7]

3. p_lookup_table is not supported. Which means transformation between gray format and argb format is not supported.

For NVR/DVR:

1. Supports input graphic formats of RGB-565, ARGB-1555, ARGB-8888, and YUV (UYVY)
2. Supports output graphic formats of RGB-565, ARGB-1555, ARGB-8888

2.2.6 HD_GFX_DRAW_LINE

[Description]
Used by hd_gfx_draw_line

[Parameter]

Value	Description
dst_img	config of destination image
color	line color in rgb
start	start position
end	end position
thickness	line's width
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]
For IPCam:

1. Only yuv420 is supported
2. Only vertical and horizontal lines are supported
3. thickness should be multiple of 2

For NVR/DRV:

1. Supports output graphic formats of RGB-565, ARGB-1555, ARGB-8888

2.2.7 HD_GFX_DRAW_RECT

[Description]
Used by hd_gfx_draw_rect

[Parameter]

Value	Description
dst_img	config of destination image
color	line color in rgb
rect	rectangle's position and size
type	solid or hollow
thickness	if type is hollow, this fields is edge’s width
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]
For IPCam:

1. Only yuv420 is supported
2. Hollow rectangle is not supported

2.2.8 HD_GFX_ARITHMETIC

[Description]
Used by hd_gfx_arithmetic

[Parameter]

Value	Description
p_op1	array of 1st operand
p_op2	array of 2nd operand
p_out	array of output
size	element number of an array
operation	plus, minus or multiply
bits	operand is 8bits or 16bits
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]
For IPCam:
1. 8bits and 16bits operands are supported. 32bits is not supported
2. Operand is unsigned but max value is half of the highest theoretical value(e.g. max value of 8bits operand is 127)
3. If the minus result is mathematically negative, 0 is returned instead(e.g. 1-3=0)
4. If the result of plus and multiply overflows, highest value is returned(e.g. 8bits 127*127=255)
5. For 16bits multiply, the 2nd operand array should be 8bits(e.g. p_op1 and p_out are 16bits arrays. p_op2 is an 8bits array)

For NVR/DRV:
This feature is not supported

2.2.9 HD_GFX_AFFINE

[Description]

Used by hd_gfx_affine

[Parameter]

Value	Description
src_img	config of source image
dst_img	config of destination image
coeff_a ~ coeff_f	affine's 6 coefficient values
engine	if multiple hardware are available, choose the hardware. This value is typically 0 in most platform.

[Note]

For IPCam:

1. Only yuv420 packed is supported
2. Config of dst_img should be equal to src_img. Only p_phy_addr can be different
3. Max rotate angle is +/- 15 degree
4. Both width and height should be multiple of 16
5. Uncovered area has undetermined value

For NVR/DRV:

1. This feature is not supported

3 Debug Menu

3.1 IPC

IPC provides history information of all gfx operations. Cat /proc/hdal/gfx/info to show history information. PID, action, execution time and extra information are logged for every operation. Job lists are split into drawing rectangles and lines to better understand blocking status. Below is an example after executing all test commands from hd_gfx_only:

```
cat /proc/hdal/gfx/info
pid     action    time(us)      info
```

```
639     scale     7546          sw(1920) sh(1080) dw(480) dh(270)
637     ct        1823          sw(1920) sh(1080) yuv4203p=>rgb8883p
634     ct        5325          sw(1920) sh(1080) yuv4203p=>yuv420
632     rect      226           sw(1920) sh(1080) rw(500) rh(200)
632     rect      221           sw(1920) sh(1080) rw(500) rh(200)
632     rect      221           sw(1920) sh(1080) rw(500) rh(200)
632     rect      226           sw(1920) sh(1080) rw(500) rh(200)
632     line      359           w(1920) h(1080) area(15000)
632     line      78            w(1920) h(1080) area(15000)
632     line      281           w(1920) h(1080) area(15000)
632     line      253           w(1920) h(1080) area(15000)
630     -(16)     96            length(1024)
628     dmacopy   1075          len(1048576)
626     rect      399           sw(1920) sh(1080) rw(500) rh(250)
624     line      432           w(1920) h(1080) area(15000)
622     ct        739           sw(1000) sh(200) argb4444=>yuv420
620     rotate    3368           w(1920) h(1080) mirrory
618     scale     23935         sw(1920) sh(1080) dw(3840) dh(2160)
615     copy      646           sw(1000) sh(200) dw(1920) dh(1080)
```

Meanings of each column are listed below:

Value	Description
pid	which process issues that operation
action	which gfx action is executed. Job lists are split into drawing rectangles and lines
time	how much time this operation takes in kernel
info	extra information. for example: resolution and format of input and output image etc.

4 Sample Codes

Since hd_gfx is a simple, self-contained and stateless module, there is no need for debug information.

4.1 Scale Image

```
#include "hdal.h"
```

```
void scale_img(MMAP_BUF *src_buf, HD_DIM src_dim, MMAP_BUF *dst_buf, HD_DIM dst_dim, char
*filename)
{
     HD_GFX_SCALE scale_img;

     scale_img.src_region.x = 0;
     scale_img.src_region.y = 0;
     scale_img.src_region.w = src_dim.w;
     scale_img.src_region.h = src_dim.h;
     scale_img.src_img.dim.w = src_dim.w;
     scale_img.src_img.dim.h = src_dim.h;
     scale_img.src_img.format = HD_VIDEO_PXLFMT_ARGB1555;
     scale_img.src_img.p_phy_addr[0] = src_buf->pa_addr;

     //scale img
     scale_img.dst_img.dim.w = dst_dim.w;
     scale_img.dst_img.dim.h = dst_dim.h;
     scale_img.dst_img.format = HD_VIDEO_PXLFMT_ARGB1555;
     scale_img.dst_img.p_phy_addr[0] = dst_buf->pa_addr;
     scale_img.dst_region.x = 0;
     scale_img.dst_region.y = 0;
     scale_img.dst_region.w = scale_img.dst_img.dim.w;
     scale_img.dst_region.h = scale_img.dst_img.dim.h;

     hd_gfx_scale(&scale_img);

save_output(filename,dst_buf->va_addr,
dst_buf->blk_size);

}
```

4.2 Blend Image

```
#include "hdal.h"


void blend_img(MMAP_BUF *src_buf, HD_DIM src_dim, MMAP_BUF *dst_buf, HD_DIM dst_dim, char
*filename)
{
	HD_GFX_COPY copy_param;
	//set dst pot
	copy_param.dst_pos.x = 100;
	copy_param.dst_pos.y = 200;

	copy_param.src_region.x = 0;
	copy_param.src_region.y = 0;
	copy_param.src_region.w = src_dim.w;
	copy_param.src_region.h = src_dim.h;

	copy_param.src_img.dim.w = src_dim.w;
	copy_param.src_img.dim.h = src_dim.h;
	copy_param.src_img.format = HD_VIDEO_PXLFMT_ARGB1555;
	copy_param.src_img.p_phy_addr[0] = src_buf->pa_addr;

	copy_param.dst_img.dim.w = dst_dim.w;
	copy_param.dst_img.dim.h = dst_dim.h;
	copy_param.dst_img.format = HD_VIDEO_PXLFMT_ARGB1555;
	copy_param.dst_img.p_phy_addr[0] = dst_buf->pa_addr;

	hd_gfx_copy(&copy_param);
	save_output(filename, dst_buf->va_addr, dst_buf->blk_size);
}
```

4.3 Draw Rectangle and Line

```
#include "hdal.h"
void draw_line_rect(void)
{
     //prepare dst buf
  img_w = 512;
  img_h = 256;
  prepare_buf(img_w, img_h, 4, &mmap_buf[0]);
  gfx_rect.dst_img.dim.w = img_w;
  gfx_rect.dst_img.dim.h = img_h;
  gfx_rect.dst_img.format = HD_VIDEO_PXLFMT_ARGB8888;
  gfx_rect.dst_img.p_phy_addr[0] = mmap_buf[0].pa_addr;

  //draw fisrt rect to dst buf
  gfx_rect.rect.x = 0;
  gfx_rect.rect.y = 0;
  gfx_rect.rect.w = img_w;
  gfx_rect.rect.h = img_h;
  gfx_rect.color = COLOUR8888(0xFF, 0, 0, 0xFF);
  gfx_rect.type = HD_GFX_RECT_SOLID;
  hd_gfx_draw_rect(&gfx_rect);

  //draw second rect to dst buf
  gfx_rect.rect.x = 100;
  gfx_rect.rect.y = 100;
  gfx_rect.rect.w = 64;
  gfx_rect.rect.h = 64;
  gfx_rect.color = COLOUR8888(0, 0xFF, 0xFF, 0xFF);
  gfx_rect.type = HD_GFX_RECT_HOLLOW;
  hd_gfx_draw_rect(&gfx_rect);

  //draw line to dst buf
  gfx_line.dst_img.dim.w = img_w;
  gfx_line.dst_img.dim.h = img_h;
  gfx_line.dst_img.format = HD_VIDEO_PXLFMT_ARGB8888;
  gfx_line.dst_img.p_phy_addr[0] = mmap_buf[0].pa_addr;
  gfx_line.color = COLOUR8888(0, 0, 0xFF, 0xFF);
  gfx_line.start.x = 50;
  gfx_line.start.y = 50;
  gfx_line.end.x = 150;
  gfx_line.end.y = 150;
  hd_gfx_draw_line(&gfx_line);

  return HD_OK;
}
```

4.4 Allocate Memory from Common Memory Pool

```
#include "hdal.h"

//Register memory requiremnt of gfx to common memory pool and get accessable address
HD_RESULT mem_init(HD_COMMON_MEM_VB_BLK *buf_blk, UINT32 buf_size, UINT32 *buf_pa)
{
     HD_RESULT                  ret;
     HD_COMMON_MEM_INIT_CONFIG    mem_cfg = {0};

     mem_cfg.pool_info[0].type     = HD_COMMON_MEM_COMMON_POOL;
     mem_cfg.pool_info[0].blk_size = buf_size;
     mem_cfg.pool_info[0].blk_cnt  = 1;
     mem_cfg.pool_info[0].ddr_id   = DDR_ID0;

     //mem_cfg should include all modules' memory requirement
     //but in this example, only register gfx's buffer to common memory pool
     ret = hd_common_mem_init(&mem_cfg);
     if(ret != HD_OK){
          printf("fail to allocate %d bytes from common pool\n", buf_size);
          return ret;
     }
```

```
	//get gfx's buffer block
	*buf_blk = hd_common_mem_get_block(HD_COMMON_MEM_COMMON_POOL, buf_size, DDR_ID0);
	if (*buf_blk == HD_COMMON_MEM_VB_INVALID_BLK) {
		printf("get block fail\r\n", *buf_blk);
		return HD_ERR_NOMEM;
	}

	//translate gfx's buffer block to physical address
	*buf_pa = hd_common_mem_blk2pa(*buf_blk);
	if (*buf_pa == 0) {
		printf("blk2pa fail, buf_blk = 0x%x\r\n", *buf_blk);
		return HD_ERR_NOMEM;
	}

	return HD_OK;
}

//return the buffer size of a 640x480 argb1555 image
int calc_buf_size()
{
	HD_VIDEO_FRAME frame;
	frame.sign = MAKEFOURCC('V','F','R','M');
	frame.pxlfmt  = HD_VIDEO_PXLFMT_ARGB1555;
	frame.dim.h   = 480;
	frame.loff[0] = 640;
	//frame.loff[1] = 640;//Some 2 planes format needs this field(e.g. yuv420)
	//frame.loff[2] = 640; //Some 3 planes format needs this field(e.g. yuv444)
	return hd_common_mem_calc_buf_size(&frame);
}

typedef struct _GFX {

	// (1)
	HD_COMMON_MEM_VB_BLK buf_blk;
	UINT32               buf_size;
	UINT32               buf_pa;
```

```
} GFX;

int main(void)
{
    HD_RESULT    ret;
      GFX         gfx;

      //gfx.buf_size should be the sum of all source/destination images' buffer size
      //suppose app uses max 16MB in this example
      //If programmers don't know how large a buffer should be, refer to calc_buf_size()
      gfx.buf_blk = 0;
      gfx.buf_size = 16 * 1024 * 1024;
      gfx.buf_pa = 0;

      ret = hd_common_init(0);
    if(ret != HD_OK) {
        printf("common fail=%d\n", ret);
        goto exit;
    }

      //allocate and map gfx memory from common memory pool
      ret = mem_init(&gfx.buf_blk, gfx.buf_size, &gfx.buf_pa);
    if(ret != HD_OK) {
        printf("mem fail=%d\n", ret);
        goto exit;
    }

    ret = hd_gfx_init();
    if(ret != HD_OK) {
        printf("init fail=%d\n", ret);
        goto exit;
    }

      //Do gfx task here. For example, call test_copy() from test app hd_gfx_only
      //In test_copy(), you will see how this API divides 16MB buffer for source and destination
```

```
images’ buffer
     ret = test_copy(gfx.buf_pa, gfx.buf_size);
     ……

exit:

     //undo mem_init()
     if(gfx.buf_blk)
          hd_common_mem_release_block(gfx.buf_blk);

     ret = hd_gfx_uninit();
     if(ret != HD_OK)
          printf("uninit fail=%d\n", ret);

     hd_common_mem_uninit();

     ret = hd_common_uninit();
     if(ret != HD_OK)
          printf("common fail=%d\n", ret);

     return 0;
}
```

5 Q&A

1. Can gfx API operates on virtual memory?
   - No.
   - Because memory from user space (e.g. malloc) is physically incontinuous and no gfx hardware supports scatter memory access, gfx API can’t operate on virtual memory.
2. What’s the difference between hd_gfx_memcpy and hd_gfx_copy?
   - hd_gfx_memcpy works on 1-dimension arrays and presents no special limitation .
   - hd_gfx_copy works on 2-dimension arrays and presents special limitation(e.g. width of arrays is best to be 4 bytes aligned).

3. Can hd_gfx_init and hd_gfx_uninit be called with other gfx API concurrently?
   - No. Other gfx API must be called between init and uninit API.
   - This is because gfx API protect themselves against concurrency by a lock. This lock is init and uninit in hd_gfx_init and hd_gfx_uninit. A lock can't be used to avoid concurrency if it is being init and uninit.
4. Some gfx API seems to be a subset of other API? Why not integrate them?
   - E.g. using hd_gfx_copy to copy an argb1555 source to a yuv420 destination of the same dimension is identical to hd_gfx_color_transform.
   - This is because gfx hardware on different platforms provides different function. For example, some platforms support above format transform. Some platforms don't. It's best stick to API semantics
5. Can I call gfx API in real-time streaming and expect 30fps?
   - Probably not. Certain platforms intentionally allocate less system resource to gfx hardware.
   - If gfx API is called in every frame, call it only when system loading is not heavy.
6. If gfx a single hardware?
   - Not always.
   - In some platforms, it's consisted of a couple of hardware.
   - In some platforms, it's a dedicate hardware.
7. Does gfx support multiple process?
   - No. gfx can only be called in multiple thread, not multiple process. Trying to do so may lead to system crash.