IMXGRAPHICUG

i.MX Graphics User's Guide

Rev. 8.4.1 — 15 December 2022 User guide

Document information

Information	Content
Keywords	i.MX, Linux, Android, Graphics
Abstract	The purpose of this document is to provide information on graphic APIs and driver support.

1 Introduction

The purpose of this document is to provide information on graphic APIs and driver support. Each chapter describes a specific set of APIs or driver integration as well as specific hardware acceleration customization. The target audiences for this document are developers writing graphics applications or video drivers.

1.1 i.MX full GPU line

The whole family of GPUs are listed in the following table. On i.MX 6 boards, only 6Quad and 6QuadPlus support OpenCL. The theoretical number of GFLOPS, the key performance indicator of OpenCL, is also shown in the table. Some benchmarks such as Clpeak, can be used to verify it.

i.MX 8QuadMax supports OpenVX, which will be introduced in next chapter.

Product	i.MX 6SoloX	i.MX 6Solo 6DualLite	i.MX 6Dual 6Quad	i.MX 6DualPlus 6QuadPlus	i.MX 7ULP	i.MX 8ULP	i.MX 8M Mini	i.MX 8M Nano	i.MX 8X 8DualXPlus 8QuadXPlus	i.MX 8M Quad, Dual QuadLite	i.MX 8M Plus	i.MX 8 8QuadPlus	i.MX 8 8QuadMax
GPU 2D	GC400T (2D)	GC320	GC355 (VG) GC320	GC355 (VG) GC328	GC328	GC520L	GC520L	N/A	High Perf 2D Blit Engine	N/A	GC520L	High Perf 2D Blit Engine	High Perf 2D Blit Engine
GPU 3D	GC400T (3D)	GC880	GC2000	GC2000+	GC700 NanoUltra	GC7000 NanoUltra31	GC7000 NanoUltra	GC7000 UltraLite	GC7000 Lite	GC7000 Lite	GC7000 UltraLite	x2 GC7000 XSVX	x2 GC7000 XSVX
# Shaders (Vec4)	1	1	4	4	1	1	1	2	4	4	2	8 + 8	8 + 8
Clock (MHz) Core [Shader]	360 [720]	264 [528]	528 [594]	594 [720]	400 [400]	317 [317]	1000	500 [600]	700 [850]	800 [800]	1000[1000]	625 [625]	800 [1000]
Pixel Rate (Mpix/s)	180	264	1056	1188	200	158	500	500	1400	1600	1000	1250 + 1250 (dual) 2500 (bridged)	1600 + 1600 (dual) 3200 (bridged)
Geom. Rate (MTri/s)	36	81	176	198	40	52	50	83	234	267	166	208 + 208 (dual) 208 (bridged)	267 + 267 (dual) 267 (bridged)
GFLOPS (Theoretical) Med/High Precision	2.9 (high)	4.2 (high)	19 (high)	46 / 23	3.2 (high)	2.5 (high)	8 (high)	9.6 (high)	55.2 / 27.6	51.2 / 25.6	16	160 / 80	256 / 128
2D API	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1 G2D	OpenVG 1.1 G2D	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1†	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D	OpenVG 1.1†, G2D
3D API	OGL ES 2.0	OGL ES 3.0	OGL ES 3.0	OGL ES 3.0	OGL ES 2.0	OGL ES 3.1 Vulkan	OGL ES 2.0	OGL ES 3.1, Vulkan	OGL ES 3.1, Vulkan	OGL ES 3.1, Vulkan	OGL ES 3.1, Vulkan	OGL ES 3.2, Vulkan	OGL ES 3.2, Vulkan
Compute	N/A	N/A	OCL 1.2 EP	OCL 1.2 FP	N/A	OCL 3.0	N/A	OCL 3.0	OCL 3.0	OCL 3.0	OCL 3.0	OCL 3.0	OCL 3.0
Other	2D / 3D Multithreaded	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	OpenVX 1.2 (NPU)	OpenVX 1.2	OpenVX 1.2

Figure 1. GPU Scalability across i.MX processors

Note: *† OpenVG on 3D GPU with software tessellation.*

2 i.MX G2D API

2.1 Overview

The G2D Application Programming Interface (API) is designed to be easy to understand and to use the 2D Bit blit (BLT) function. It allows the user to implement the customized applications with simple interfaces. It is hardware and platform independent for i.MX 2D Graphics.

G2D API supports the following features but is not limited to these:

- Simple BLT operation from source to destination
- 16/32bit RGB(alpha) and YUV color format conversions
- Alpha blending for source and destination with Porter-Duff rules
- High-performance memory copy from source to destination
- Up-scaling and down-scaling from source to destination
- 90/180/270 degree rotation from source to destination

- Horizontal and vertical flip from source to destination
- Enhanced visual quality with dither for pixel precision-loss (*)
- High performance memory clear for destination
- Pixel-level cropping for source surface
- Global alpha blending for source only
- Asynchronous mode and sync
- Contiguous memory allocator
- Support cacheable memory (*)
- Support VG engine (*)
- Multi source blit (*)

Note: The features with (*) are available on specific devices. Applications can query G2D for available features.

The G2D API document includes a detailed interface description and sample code for reference.

The API is designed with C-Style coding and can be used in both C and C++ applications.

2.2 Enumerations and structures

This chapter describes all enumerations and structure definitions in G2D.

2.2.1 g2d_format enumeration

This enumeration describes the pixel format for source and destination.

Table 1. g2d_format enumeration

Name	Numeric	Description
G2D_RGB565	0	RGB565 pixel format
G2D_RGBA8888	1	32-bit RGBA pixel format
G2D_RGBX8888	2	32-bit RGBX without alpha blending
G2D_BGRA8888	3	32-bit BGRA pixel format
G2D_BGRX8888	4	32-bit BGRX without alpha blending
G2D_BGR565	5	16-bit BGR565 pixel format
G2D_ARGB8888	6	32-bit ARGB pixel format
G2D_ABGR8888	7	32-bit ABGR pixel format
G2D_XRGB8888	8	32-bit XRGB without alpha
G2D_XBGR8888	9	32-bit XBGR without alpha
G2D_RGB888	10	24-bit RGB
G2D_BGR888	11	24-bit BGR
G2D_NV12	20	Y plane followed by interleaved U/V plane
G2D_I420	21	Y, U, V are within separate planes
G2D_YV12	22	Y, V, U are within separate planes
G2D_NV21	23	Y plane followed by interleaved V/U plane
G2D_YUYV	24	Interleaved Y/U/Y/V plane

Table 1. g2d_format enumeration*...continued*

Name	Numeric	Description
G2D_YVYU	25	Interleaved Y/V/Y/U plane
G2D_UYVY	26	Interleaved U/Y/V/Y plane
G2D_VYUY	27	Interleaved V/Y/U/Y plane
G2D_NV16	28	Y plane followed by interleaved U/V plane
G2D_NV61	29	Y plane followed by interleaved V/U plane

2.2.2 g2d_blend_func enumeration

This enumeration describes the blend factor for source and destination.

Table 2. g2d_blend_func enumeration

Name	Numeric	Description
G2D_ZERO	0	Blend factor with 0
G2D_ONE	1	Blend factor with 1
G2D_SRC_ALPHA	2	Blend factor with source alpha
G2D_ONE_MINUS_SRC_ALPHA	3	Blend factor with 1 - source alpha
G2D_DST_ALPHA	4	Blend factor with destination alpha
G2D_ONE_MINUS_DST_ALPHA	5	Blend factor with 1 - destination alpha
G2D_PRE_MULTIPLIED_ALPHA	0x10	Extensive blend as pre-multiplied alpha
G2D_DEMULTIPLY_OUT_ALPHA	0x20	Extensive blend as demultiply out alpha

2.2.3 g2d_cap_mode enumeration

This enumeration describes the alternative capability in 2D BLT.

Table 3. g2d_cap_mode enumeration

Name	Numeric	Description
G2D_BLEND	0	Enable alpha blend in 2D BLT
G2D_DITHER	1	Enable dither in 2D BLT
G2D_GLOBAL_ALPHA	2	Enable global alpha in blend
G2D_BLEND_DIM	3	Enable blend dim effect
G2D_BLUR	4	Enable blur effect
G2D_YUY_BT_601	5	Enable YUV BT.601 mode
G2D_YUY_BT_709	6	Enable YUV BT.709 mode
G2D_YUY_BT_601FR	7	Enable YUV BT.601 full range mode
G2D_YUY_BT_709FR	8	Enable YUV BT.709 full range mode

Note: *G2D_GLOBAL_ALPHA is only valid when G2D_BLEND is enabled.*

2.2.4 g2d_rotation enumeration

This enumeration describes the rotation mode in 2D BLT.

Table 4. g2d_rotation enumeration

Name	Numeric	Description
G2D_ROTATION_0	0	No rotation
G2D_ROTATION_90	1	Rotation with 90 degree
G2D_ROTATION_180	2	Rotation with 180 degree
G2D_ROTATION_270	3	Rotation with 270 degree
G2D_FLIP_H	4	Horizontal flip
G2D_FLIP_V	5	Vertical flip

2.2.5 g2d_cache_mode enumeration

This enumeration describes the cache operation mode.

Table 5. g2d_cache_mode enumeration

Name	Numeric	Description
G2D_CACHE_CLEAN	0	Clean the cacheable buffer
G2D_CACHE_FLUSH	1	Clean and invalidate cacheable buffer
G2D_CACHE_INVALIDATE	2	Invalidate the cacheable buffer

2.2.6 g2d_hardware_type enumeration

This enumeration describes the supported hardware type.

Table 6. g2d_hardware_type enumeration

Name	Numeric	Description
G2D_HARDWARE_2D	0	2D hardware type by default
G2D_HARDWARE_VG	1	VG hardware type

2.2.7 g2d_surface structure

This structure describes the surface with operation attributes.

Table 7. g2d_surface structure

g2d_surface Members	Type	Description
format	g2d_format	Pixel format of surface buffer
planes[3]	Int	Physical addresses of surface buffer
left	Int	Left offset in blit rectangle
top	Int	Top offset in blit rectangle
right	Int	Right offset in blit rectangle
bottom	Int	Left offset in blit rectangle
stride	Int	RGB/Y stride of surface buffer
width	Int	Surface width in pixel unit
height	Int	Surface height in pixel unit
blendfunc	g2d_blend_func	Alpha blend mode

Table 7. g2d_surface structure...continued

g2d_surface Members	Type	Description
global_alpha	Int	Global alpha value 0~255
clrcolor	Int	Clear color is 32bit RGBA
rot	g2d_rotation	Rotation mode

Note: *RGB and YUV formats conversion, Y(*) means feature available on i.MX 6Quad Plus, i.MX 7ULP and i.MX 8 family devices.*

SRC \ DST	G2D_RGBs	G2D_YV12	G2D_I420	G2D_NV12	G2D_NV21	G2D_YUYV	G2D_NV16	G2D_NV61
G2D_RGBs	Y	N	N	N	N	Y(*)	N	N
G2D_NV12	Y	N	N	N	N	Y(*)	N	N
G2D_I420	Y	N	N	N	N	Y(*)	N	N
G2D_YV12	Y	N	N	N	N	Y(*)	N	N
G2D_NV21	Y	N	N	N	N	Y(*)	N	N
G2D_YUYV	Y	N	N	Y(*)	Y(*)	Y(*)	Y(*)	Y(*)
G2D_YVYU	Y	N	N	N	N	Y(*)	N	N
G2D_UYVY	Y	N	N	N	N	Y(*)	N	N
G2D_VYUY	Y	N	N	N	N	Y(*)	N	N
G2D_NV16	Y	N	N	N	N	Y(*)	N	N
G2D_NV61	Y	N	N	N	N	Y(*)	N	N

- RGB pixel buffer only uses planes [0], buffer address is with 16 bytes alignment on i.MX 6 (except i.MX 6Quad Plus), 1 pixel alignment on i.MX 6Quad Plus, i.MX 7ULP and i.MX 8 family devices.
- NV12: Y in planes [0], UV in planes [1], with 64bytes alignment,
- I420: Y in planes [0], U in planes [1], U in planes [2], with 64 bytes alignment
- The cropped region in source surface is specified with left, top, right and bottom parameters.
- RGB stride alignment is 16 bytes on i.MX 6 (except i.MX 6Quad Plus), 1 pixel alignment on i.MX 6Quad Plus, i.MX 7ULP and i.MX 8 family devices, both for source and destination surface.
- NV12 stride alignment is 8 bytes for source surface, UV stride = Y stride,
- I420 stride alignment is 8 bytes for source surface, U stride=V stride = ½ Y stride.
- G2D_ROTATION_0/G2D_FLIP_H/G2D_FLIP_V shall be set in source surface, and the clockwise rotation degree shall be set in destination surface.
- Application should calculate the rotated position and set it for destination surface.
- The geometry definition of surface structure is described as follows.

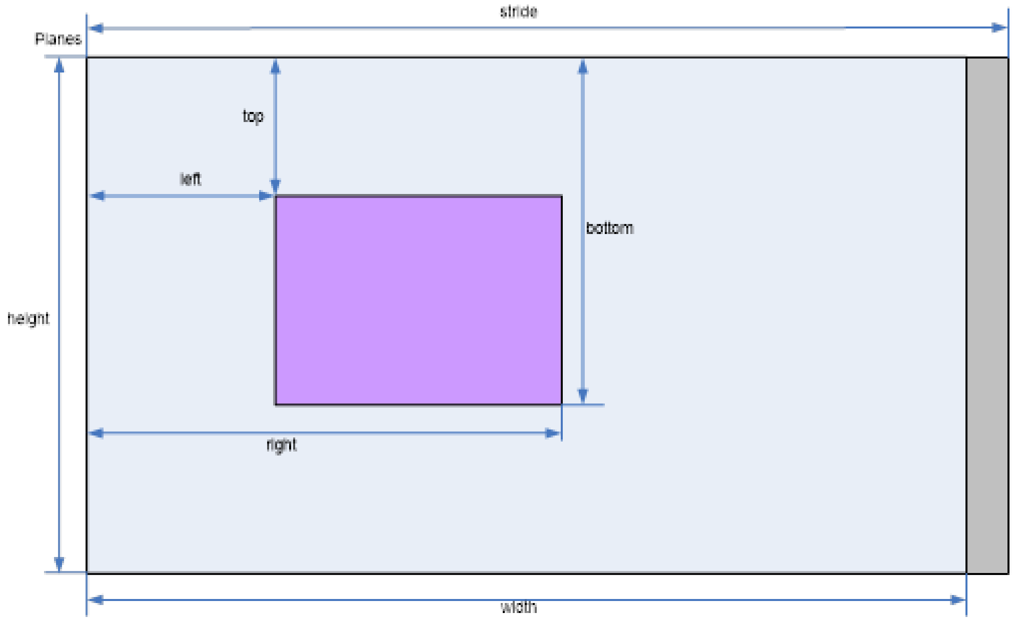


Figure 2. g2d_surface structure

2.2.8 g2d_buf structure

This structure describes the buffer used as G2D interfaces.

Table 8. g2d_buf structure

g2d_buf Members	Type	Description
buf_handle	void *	The handle associated with buffer
buf_vaddr	void *	Virtual address of the buffer
buf_paddr	int	Physical address of the buffer
buf_size	int	The actual size of the buffer

2.2.9 g2d_surface_pair structure

This structure binds one source g2d_surface and one destination g2d_surface as a pair. When doing multi-source blit, they are one-to-one correspondent.

Table 9. g2d_surface_pair structure

g2d_surface_pair Members	Type	Description
s	g2d_surface	Source g2d_surface
d	g2d_surface	Destination g2d_surface

2.2.10 g2d_feature enumeration

This enumeration describes the features in G2D BLT.

Table 10. g2d_feature enumeration

Name	Numeric	Description
G2D_SCALING	0	Scaling
G2D_ROTATION	1	Rotation
G2D_SRC_YUV	2	Source YUV format
G2D_DST_YUV	3	Destination YUV format
G2D_MULTI_SOURCE_BLT	4	Multisource blit
G2D_FAST_CLEAR	5	Support fast clear blit

2.3 G2D function description

2.3.1 g2d_open

Description	Open a G2D device and return a handle.
Syntax	`int g2d_open (void **handle);`
Parameters	`handle`: Pointer to receive G2D device handle
Returns	Success with 0, fail with -1

2.3.2 g2d_close

Description	Close G2D device with the handle.
Syntax	`int g2d_close (void *handle);`
Parameters	`handle`: G2D device handle
Returns	Success with 0, fail with -1

2.3.3 g2d_make_current

Description	Set the specific hardware type for current context, and the default is G2D_HARDWARE_2D.
Syntax	`int g2d_make_current (void *handle, enum g2d_hardware_type type);`
Parameters	`handle`: G2D device handle
Returns	Success with 0, fail with -1

2.3.4 g2d_clear

Description	Clear a specific area.
Syntax	`int g2d_clear (void *handle, struct g2d_surface *area);`
Parameters	`handle`: G2D device handle `area`: The area to be cleared
Returns	Success with 0, fail with -1

2.3.5 g2d_blit

Description	G2D blit from source to destination with alternative operation (Blend, Dither, etc.).
Syntax	`int g2d_blit (void *handle, struct g2d_surface *src, struct g2d_surface *dst);`
Parameters	`handle`: G2D device handle `src`: source surface `dst`: destination surface
Returns	Success with 0, fail with -1

2.3.6 g2d_copy

Description	G2D copy with specified size.
Syntax	`int g2d_copy (void *handle, struct g2d_buf *d, struct g2d_buf* s, int size);`
Parameters	`handle`: G2D device handle `d`: destination buffer `s`: source buffer `size`: copy bytes
Limitations	If the destination buffer is cacheable, it must be invalidated before g2d_copy due to the alignment limitation of G2D driver.
Returns	Success with 0, fail with -1

2.3.7 g2d_query_cap

Description	Query the alternative capability enablement.
Syntax	`int g2d_query_cap (void *handle, enum g2d_cap_mode cap, int *enable);`
Parameters	`handle`: G2D device handle `cap`: G2D capability to query `enable`: Pointer to receive G2D capability enablement
Returns	Success with 0, fail with -1

2.3.8 g2d_enable

Description	Enable G2D capability with the specific mode.
Syntax	`int g2d_enable (void *handle, enum g2d_cap_mode cap);`
Parameters	`handle`: G2D device handle `cap`: G2D capability to enable
Returns	Success with 0, fail with -1

2.3.9 g2d_disable

Description	Disable G2D capability with the specific mode.
Syntax	`int g2d_disable (void *handle, enum g2d_cap_mode cap);`
Parameters	`handle`: G2D device handle `cap`: G2D capability to disable
Returns	Success with 0, fail with -1

2.3.10 g2d_cache_op

Description	Perform cache operations for the cacheable buffer allocated through the G2D driver.
Syntax	`int g2d_cache_op (struct g2d_buf *buf, enum g2d_cache_mode op);`
Parameters	`buf`: the buffer to be handled with cache operations `op`: cache operation type
Returns	Success with 0, fail with -1

2.3.11 g2d_alloc

Description	Allocate a buffer through G2D device
Syntax	`struct g2d_buf *g2d_alloc (int size, int cacheable);`
Parameters	`size`: allocated bytes `cacheable`: 0, non-cacheable; 1, cacheable attribute defined by system
Returns	Success with valid G2D buffer pointer, fail with 0

2.3.12 g2d_free

Description	Free the buffer through G2D device.
Syntax	`int g2d_free (struct g2d_buf *buf);`
Parameters	`buf`: G2D buffer to free
Returns	Success with 0, fail with -1

2.3.13 g2d_flush

Description	Flush G2D command and return without completing pipeline.
Syntax	`int g2d_flush (void *handle);`
Parameters	`handle`: G2D device handle
Returns	Success with 0, fail with -1

2.3.14 g2d_finish

Description	Flush G2D command and then return when pipeline is finished.
Syntax	`int g2d_finish (void *handle);`
Parameters	`handle`: G2D device handle
Returns	Success with 0, fail with -1

2.3.15 g2d_multi_blit

Description	Blit multiple sources to one destination.
Syntax	`int g2d_multi_blit (void *handle, struct g2d_surface_pair *sp[], int layers);`
Parameters	`handle`: G2D device handle `sp`: array in which elements point to g2d_surface_pair

	`layers`: number of the source layers that need to be blited
Returns	Success with 0, fail with -1

Note:

There are some restrictions for this API that we should be aware of.

- *This API only works on the i.MX 6DualPlus/QuadPlus platform.*
- *The maximum number of the source layers that can be blited one time is 8.*
- *Although g2d_surface_pair binds one source g2d_surface and one destination g2d_surface as a pair, it only supports one destination surface. The relationship between the source and destination is many to one, but each source surface can be set separately and differently, and its dimension, stride, rotation, and format can differ with that of the destination surface.*
- *The rotation of the destination surface is set to 0 degree by defaut, and cannot be changed.*
- *The key restriction is that the destination rectangle cannot be set, which means that the destination rectangle must be the same as the source rectangle. Therefore, if the source rectangle is set to (l, t, r, b), the destination rectangle should also be set to (l, t, r, b) by hardware. In the chapter on multi source blit (Section 2.5.4), as it makes no sense to set the destination rectangles, we just set all of them to (0, 0, width, height) for future extension.*

2.3.16 g2d_query_hardware

Description	Query whether 2D and VG hardware are available in the current G2D.
Syntax	`int g2d_query_hardware (void *handle, enum g2d_hardware_type type, int *available);`
Parameters	`handle`: G2D device handle `type`: G2D hardware type `available`: Pointer to receive G2D hardware type availability
Returns	Success with 0, fail with -1

2.3.17 g2d_query_feature

Description	Query if the features are available in G2D BLT.
Syntax	`int g2d_query_feature (void *handle, enum g2d_feature feature, int *available);`
Parameters	`handle`: G2D device handle `feature`: G2D feature in g2d_blit `available`: Pointer to receive G2D feature availability
Returns	Success with 0, fail with -1

2.4 Support of new operating system in G2D

G2D code is independent on operating system (OS) except of buffer allocation. Allocating the memory for buffer is made by mechanism that is offered by each OS differently. The code for allocation is located in [G2D repository copy]/source/os/[OS name]. Therefore, supporting new OS includes the following steps:

1. Create a new folder in **[G2D repository copy]/source/os/** with the name of the new OS and update implementation in the included source code according to the new OS allocation mechanism.
2. When creating new makefiles for the OS, include the files from the new folder.
3. The test named **overlay_test** contains the OS dependent code. For supporting the new OS in this test, create new folder in **[G2D repository copy]/test/overlay_test/os** and update the code according to the new OS mechanism for display initialization. Also update makefiles to include code from the new folder.

2.5 Sample code for G2D API usage

This chapter provides the brief prototype code with G2D API.

2.5.1 Color space conversion from YUV to RGB

```
g2d_open(&handle);
src.planes[0] = buf_y;
src.planes[1] = buf_u;
src.planes[2] = buf_v;
src.left = crop.left;
src.top = crop.top;
src.right = crop.right;
src.bottom = crop.bottom;
src.stride = y_stride;
      src.width = y_width;
      src.height = y_height;
src.rot    = G2D_ROTATION_0;
src.format = G2D_I420;
dst.planes[0] = buf_rgba;
dst.left = 0;
dst.top = 0;
dst.right = disp_width;
dst.bottom = disp_height;
dst.stride = disp_width;
      dst.width = disp_width;
      dst.height = disp_height;
dst.rot    = G2D_ROTATION_0;
dst.format = G2D_RGBA8888;
g2d_blit(handle, &src, &dst);
      g2d_finish(handle);
g2d_close(handle);
```

2.5.2 Alpha blend in source over mode

```
g2d_open(&handle);
src.planes[0] = src_buf;
src.left = 0;
```

```
src.top = 0;
src.right = test_width;
src.bottom = test_height;
src.stride = test_width;
src.width = test_width;
src.height = test_height;
src.rot = G2D_ROTATION_0;
src.format = G2D_RGBA8888;
src.blendfunc = G2D_ONE;
dst.planes[0] = dst_buf;
dst.left = 0;
dst.top = 0;
dst.right = test_width;
dst.bottom = test_height;
dst.stride = test_width;
dst.width = test_width;
dst.height = test_height;
dst.format = G2D_RGBA8888;
dst.rot = G2D_ROTATION_0;
dst.blendfunc = G2D_ONE_MINUS_SRC_ALPHA;
g2d_enable(handle,G2D_BLEND);
g2d_blit(handle, &src, &dst);
g2d_finish(handle);
g2d_disable(handle,G2D_BLEND);
g2d_close(handle);
```

2.5.3 Source cropping and destination rotation

```
g2d_open(&handle);
src.planes[0] = src_buf;
src.left = crop.left;
src.top = crop.left;
src.right = crop.right;
src.bottom = crop.bottom;
src.stride = src_stride;
src.width = src_width;
src.height = src_height;
src.format = G2D_RGBA8888;
src.rot    = G2D_ROTATION_0;//G2D_FLIP_H or G2D_FLIP_V
dst.planes[0] = dst_buf;
dst.left = 0;
dst.top = 0;
dst.right = dst_width;
dst.bottom = dst_height;
dst.stride = dst_width;
dst.width = dst_width;
dst.height = dst_height;
dst.format = G2D_RGBA8888;
dst.rot    = G2D_ROTATION_90;
g2d_blit(handle, &src, &dst);
g2d_finish(handle);
g2d_close(handle)
```

2.5.4 Multi source blit

```
const int layers = 8;
struct g2d_buf *d_buf;
```

```
struct g2d_buf *mul_s_buf[layers];
struct g2d_surface_pair *sp[layers];
g2d_open(&handle)
for(n = 0; n < layers; n++) {
sp[n] = (struct g2d_surface_pair *)malloc(sizeof(struct
 g2d_surface_pair));
}
d_buf = g2d_alloc(test_width * test_height * 4, 0);
for(n = 0; n < layers; n++) {
  mul_s_buf[n] = g2d_alloc(test_width * test_height * 4, 0);
}
for(n = 0; n < layers; n++) {
sp[n]->s.left = img_info_ptr[n]->img_left;
          sp[n]->s.top = img_info_ptr[n]->img_top;
          sp[n]->s.right =  img_info_ptr[n]->img_right;
          sp[n]->s.bottom = img_info_ptr[n]->img_bottom;
          sp[n]->s.stride = img_info_ptr[n]->img_width;
          sp[n]->s.width =  img_info_ptr[n]->img_width;
          sp[n]->s.height = img_info_ptr[n]->img_height;
          sp[n]->s.rot =    img_info_ptr[n]->img_rot;
          sp[n]->s.format = img_info_ptr[n]->img_format;
  sp[n]->s.planes[0] = mul_s_buf[n]->buf_paddr;
}
sp[0]->d.left = 0;
sp[0]->d.top  = 0;
sp[0]->d.right = test_width;
sp[0]->d.bottom = test_height;
sp[0]->d.stride = test_width;
sp[0]->d.width  = test_width;
sp[0]->d.height = test_height;
sp[0]->d.format = G2D_RGBA8888;
sp[0]->d.rot    = G2D_ROTATION_0;
sp[0]->d.planes[0] = d_buf->buf_paddr;
for(n = 1; n < layers; n++) {
  sp[n]->d = sp[0]->d;
}
g2d_multi_blit(handle, sp, layers);
    g2d_finish(handle);
    for(n = 0; n < layers; n++)
      g2d_free(mul_s_buf[n]);
    g2d_free(d_buf);
    g2d_close(handle);
```

2.5.5 Sharing Buffers between APIs using G2D Buffers:

The G2D buffers can be used to avoid memory copies between APIs. Create a buffer using g2d_alloc and then map it as an OpenGL ES texture or as an OpenVX buffer or an OpenCV Mat:

Allocate your buffer with:

```
struct g2d_buf * buffer0;
buffer0 = g2d_alloc(WIDTH*HEIGHT*4, 0);
```

For OpenCV, you map the buffer to the data field of the cv::Mat

```
cv::Mat buffer0Mat;
buffer0Mat.create (WIDTH, HEIGHT, CV_8UC4);
```

```
buffer0Mat.data = (uchar *) ((unsigned long) buffer0-
>buf_vaddr);
```

For OpenGL ES, you can make use of the DirectVIV extensions:

```
glGenTextures(1, &textureHandle[0]);
glBindTexture(GL_TEXTURE_2D, textureHandle[0]);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
 GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
 GL_LINEAR);
glTexDirectVIVMap(GL_TEXTURE_2D, WIDTH, HEIGHT,  GL_RGBA,
                                   &buffer0->buf_vaddr, (uint
 *)&buffer0->buf_paddr);
glTexDirectInvalidateVIV (GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, 0);
```

For OpenVX you create vxImages from the buffer ranges:

```
vx_imagepatch_addressing_t patch0 = { (vx_uint32)WIDTH,
 (vx_uint32)HEIGHT,
(vx_int32)4, (vx_int32)HEIGHT*4, VX_SCALE_UNITY,
 VX_SCALE_UNITY, 1, 1 };
    void *ptr0 = buffer0->buf_vaddr;
vxInputImage = vxCreateImageFromHandle(contextVX,
VX_DF_IMAGE_RGBX, &patch0, (void **)&ptr0,
 VX_MEMORY_TYPE_HOST);
```

With this scheme you can create a multi API pipeline, where you can post-process your OpenGL ES render result with CV or VX without the need of copying data.

2.6 Feature list on multiple platforms

This user guide is for multiple platforms, such as i.MX 6 and i.MX 8, and the hardwares for the G2D implementation are different on those platforms, so some G2D features are also different.

For example, the G2D_YVYU and G2D_VYUY formats are not supported on the i.MX 8, and the g2d_multi_blit function only works on the i.MX 6DualPlus/QuadPlus. Therefore, we list those differences in the following feature table.

Table 11. Feature list on multiple platforms

Feature	i.MX 6		i.MX 7	i.MX 8	
	6Solo/6 Dual/ 6Quad	6DualPlus/ 6QuadPlus	7ULP	8M Mini/ 8M Plus	8Quad Max/8Quad XPlus
G2D_YVYU	Yes	Yes	Yes	Yes	No
G2D_VYUY	Yes	Yes	Yes	Yes	No
G2D_HARDWARE_VG	Yes	Yes	No	No	No
G2D_MULTI_ SOURCE_BLT	No	Yes	Yes	Yes	No
g2d_cache_op	Yes	Yes	Yes	Yes	No

3 i.MX EGL and OGL Extension Support

3.1 Introduction

The following tables list the level of support for EGL and OES extensions available with i.MX hardware and software. Support levels are current as of the date of the document and subject to change.

Two tables are provided. The first table lists the EGL interface extensions. The second table lists extensions for OpenGL ES 1.1, OpenGL ES 2.0, and OpenGL ES 3.0.

Key:

- **Extension Name and Number**: Each listed extension is derived from the relevant khronos.org webpage list and includes the extension number as well as a hyperlink to the khronos description of the extension.
- **Yes**: Support is currently available.
- **No**: Support is not available. (Reasons for lack of support may vary: the extension may be proprietary or obsolete, or not applicable to the specified OES version.)
- **N/A**: Support is not provided as the extension is not applicable in this and subsequent versions of the specification.

3.2 EGL extension support

The following table includes the list of all current EGL Extensions and indicates their support level.

(list from www.khronos.org/registry/egl/ as of 1/24/2020)

Table 12. EGL extension support

EGL Extension Number, Name and hyperlink (2020)	Linux	Android	QNX
1. EGL_KHR_config_attribs			
2. EGL_KHR_lock_surface	YES	YES	YES
3. EGL_KHR_image	YES	YES	YES
4. EGL_KHR_vg_parent_image			
5. EGL_KHR_gl_texture_2D_image	YES	YES	YES
EGL_KHR_gl_texture_cubemap_image	YES	YES	YES
EGL_KHR_gl_texture_3D_image			
EGL_KHR_gl_renderbuffer_image	YES	YES	YES
6. EGL_KHR_reusable_sync	YES	YES	YES
7. EGL_KHR_image_base	YES	YES	YES
8. EGL_KHR_image_pixmap	YES	YES	YES
9. EGL_IMG_context_priority	YES	YES	
10. EGL_NOK_texture_from_pixmap			
11. EGL_KHR_lock_surface2			
12. EGL_NV_coverage_sample			

Table 12. EGL extension support...*continued*

EGL Extension Number, Name and hyperlink (2020)	Linux	Android	QNX
13. EGL_NV_depth_nonlinear			
14. EGL_NV_sync			
15. EGL_KHR_fence_sync	YES	YES	YES
16. EGL_NOK_swap_region2			
17. EGL_HI_clientpixmap			
18. EGL_HI_colorformats			
19. EGL_MESA_drm_image			
20. EGL_NV_post_sub_buffer			
21. EGL_ANGLE_query_surface_pointer			
22. EGL_ANGLE_surface_d3d_texture_2d_share_handle			
23. EGL_NV_coverage_sample_resolve			
24. EGL_NV_system_time			
25. EGL_KHR_stream			
EGL_KHR_stream_attrib			
26. EGL_KHR_stream_consumer_gltexture			
27. EGL_KHR_stream_producer_eglsurface			
28. EGL_KHR_stream_producer_aldatalocator			
29. EGL_KHR_stream_fifo			
30. EGL_EXT_create_context_robustness			
31. EGL_ANGLE_d3d_share_handle_client_buffer			
32. EGL_KHR_create_context	YES	YES	YES
33. EGL_KHR_surfaceless_context	YES	YES	YES
34. EGL_KHR_stream_cross_process_fd			
35. EGL_EXT_multiview_window			
36. EGL_KHR_wait_sync	YES	YES	YES
37. EGL_NV_post_convert_rounding			
38. EGL_NV_native_query			
39. EGL_NV_3dvision_surface			
40. EGL_ANDROID_framebuffer_target		YES	
41. EGL_ANDROID_blob_cache		YES	
42. EGL_ANDROID_image_native_buffer		YES	
43. EGL_ANDROID_native_fence_sync		YES	
44. EGL_ANDROID_recordable		YES	
45. EGL_EXT_buffer_age	YES	YES	YES
46. EGL_EXT_image_dma_buf_import	YES	YES	
47. EGL_ARM_pixmap_multisample_discard			

Table 12. EGL extension support...*continued*

EGL Extension Number, Name and hyperlink (2020)	Linux	Android	QNX
48. EGL_EXT_swap_buffers_with_damage	YES	YES	YES
49. EGL_NV_stream_sync			
50. EGL_EXT_platform_base	YES	YES	YES
51. EGL_EXT_client_extensions	YES	YES	YES
52. EGL_EXT_platform_x11	YES	YES	YES
53. EGL_KHR_cl_event			
54. EGL_KHR_get_all_proc_addresses	YES	YES	YES
EGL_KHR_client_get_all_proc_addresses	YES	YES	YES
55. EGL_MESA_platform_gbm			
56. EGL_EXT_platform_wayland	YES		
57. EGL_KHR_lock_surface3			
58. EGL_KHR_cl_event2			
59. EGL_KHR_gl_colorspace			
60. EGL_EXT_protected_surface	YES	YES	YES
61. EGL_KHR_platform_android		YES	
62. EGL_KHR_platform_gbm	YES	YES	YES
63. EGL_KHR_platform_wayland	YES		
64. EGL_KHR_platform_x11	YES		
65. EGL_EXT_device_base			
66. EGL_EXT_platform_device			
67. EGL_NV_device_cuda			
68. EGL_NV_cuda_event			
69. EGL_TIZEN_image_native_buffer			
70. EGL_TIZEN_image_native_surface			
71. EGL_EXT_output_base			
72. EGL_EXT_device_drm			
EGL_EXT_output_drm			
73. EGL_EXT_device_openwf			
EGL_EXT_output_openwf			
74. EGL_EXT_stream_consumer_egloutput			
75. EGL_KHR_partial_update	YES	YES	YES
76. EGL_KHR_swap_buffers_with_damage	YES	YES	YES
77. EGL_ANGLE_window_fixed_size			
78. EGL_EXT_yuv_surface			
79. EGL_MESA_image_dma_buf_export			
80. EGL_EXT_device_enumeration			

Table 12. EGL extension support*...continued*

EGL Extension Number, Name and hyperlink (2020)	Linux	Android	QNX
81. EGL_EXT_device_query			
82. EGL_ANGLE_device_d3d			
83. EGL_KHR_create_context_no_error			
84. EGL_KHR_debug			
85. EGL_NV_stream_metadata			
86. EGL_NV_stream_consumer_gltexture_yuv			
87. EGL_IMG_image_plane_attribs			
88. EGL_KHR_mutable_render_buffer			
89. EGL_EXT_protected_content			
90. EGL_ANDROID_presentation_time			
91. EGL_ANDROID_create_native_client_buffer			
92. EGL_ANDROID_front_buffer_auto_refresh			
93. EGL_KHR_no_config_context	YES	YES	YES
94. EGL_KHR_context_flush_control			
95. EGL_ARM_implicit_external_sync			
96. EGL_MESA_platform_surfaceless			
97. EGL_EXT_image_dma_buf_import_modifiers	YES	YES	
98. EGL_EXT_pixel_format_float			
99. EGL_EXT_gl_colorspace_bt2020_linear			
EGL_EXT_gl_colorspace_bt2020_pq			
100. EGL_EXT_gl_colorspace_scrgb_linear			
101. EGL_EXT_surface_SMPTE2086_metadata			
102. EGL_NV_stream_fifo_next			
103. EGL_NV_stream_fifo_synchronous			
104. EGL_NV_stream_reset			
105. EGL_NV_stream_frame_limits			
106. EGL_NV_stream_remote			
EGL_NV_stream_cross_object			
EGL_NV_stream_cross_display			
EGL_NV_stream_cross_process			
EGL_NV_stream_cross_partition			
EGL_NV_stream_cross_system			
107. EGL_NV_stream_socket			
EGL_NV_stream_socket_unix			
EGL_NV_stream_socket_inet			
108. EGL_EXT_compositor			

Table 12. EGL extension support...*continued*

EGL Extension Number, Name and hyperlink (2020)	Linux	Android	QNX
109. EGL_EXT_surface_CTA861_3_metadata			
110. EGL_EXT_gl_colorspace_display_p3			
111. EGL_EXT_gl_colorspace_display_p3_linear			
112. EGL_EXT_gl_colorspace_scrgb (non-linear)			
113. EGL_EXT_image_implicit_sync_control			
114. EGL_EXT_bind_to_front			
115. EGL_ANDROID_get_frame_timestamps			
116. EGL_ANDROID_get_native_client_buffer			
117. EGL_NV_context_priority_realtime			
118. EGL_EXT_image_gl_colorspace			
119. EGL_KHR_display_reference			
120. EGL_NV_stream_flush			
121. EGL_EXT_sync_reuse			
122. EGL_EXT_client_sync			
123. EGL_EXT_gl_colorspace_display_p3_passthrough			
124. EGL_MESA_query_driver			
125. EGL_ANDROID_GLES_layers			
126. EGL_NV_n_buffer			
127. EGL_NV_stream_origin			
128. EGL_NV_stream_dma			
129. EGL_WL_bind_wayland_display	YES		
130. EGL_WL_create_wayland_buffer_from_image	YES		

3.3 OpenGL ES extension support

The following table includes the list of all current OpenGL ES Extensions and indicates their support level.

(list from www.khronos.org/registry/gles/ as of 6/14/2020)

Table 13. OpenGL ES extension support

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
1. GL_OES_blend_equation_separate	YES	
2. GL_OES_blend_func_separate	YES	
3. GL_OES_blend_subtract	YES	
4. GL_OES_byte_coordinates	YES	
5. GL_OES_compressed_ETC1_RGB8_texture	YES	YES
6. GL_OES_compressed_paletted_texture	YES	YES
7. GL_OES_draw_texture	YES	

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
8. GL_OES_extended_matrix_palette	YES	
9. GL_OES_fixed_point	YES	
10. GL_OES_framebuffer_object	YES	
11. GL_OES_matrix_get	YES	
12. GL_OES_matrix_palette	YES	
13. GL_OES_point_size_array	YES	
14. GL_OES_point_sprite	YES	
15. GL_OES_query_matrix	YES	
16. GL_OES_read_format	YES	
17. GL_OES_single_precision	YES	
18. GL_OES_stencil_wrap	YES	
19. GL_OES_texture_cube_map	YES	
20. GL_OES_texture_env_crossbar		
21. GL_OES_texture_mirrored_repeat	YES	
22. GL_OES_EGL_image	YES	YES
23. GL_OES_depth24	YES	YES
24. GL_OES_depth32		YES
25. GL_OES_element_index_uint	YES	YES
26. GL_OES_fbo_render_mipmap	YES	YES
27. GL_OES_fragment_precision_high		YES
28. GL_OES_mapbuffer	YES	YES
29. GL_OES_rgb8_rgba8	YES	YES
30. GL_OES_stencil1		
31. GL_OES_stencil4		
32. GL_OES_stencil8	YES	
33. GL_OES_texture_3D		
34. GL_OES_texture_float_linear		
GL_OES_texture_half_float_linear		CORE
35. GL_OES_texture_float		CORE
GL_OES_texture_half_float		CORE
36. GL_OES_texture_npot	YES	YES
37. GL_OES_vertex_half_float	YES	YES
38. GL_AMD_compressed_3DC_texture		
39. GL_AMD_compressed_ATC_texture		
40. GL_EXT_texture_filter_anisotropic	CORE	CORE
41. GL_EXT_texture_type_2_10_10_10_REV		CORE

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
42. GL_OES_depth_texture		YES
43. GL_OES_packed_depth_stencil	YES	YES
44. GL_OES_standard_derivatives		YES
45. GL_OES_vertex_type_10_10_10_2		CORE
46. GL_OES_get_program_binary		YES
47. GL_AMD_program_binary_Z400		
48. GL_EXT_texture_compression_dxt1		YES
49. GL_AMD_performance_monitor		
50. GL_EXT_texture_format_BGRA8888	YES	YES
51. GL_NV_fence		
52. GL_IMG_read_format		
53. GL_IMG_texture_compression_pvrtc		
54. GL_QCOM_driver_control		
55. GL_QCOM_performance_monitor_global_mode		
56. GL_IMG_user_clip_plane		
57. GL_IMG_texture_env_enhanced_fixed_function		
58. GL_APPLE_texture_2D_limited_npot		
59. GL_EXT_texture_lod_bias	YES	
60. GL_QCOM_writeonly_rendering		
61. GL_QCOM_extended_get		
62. GL_QCOM_extended_get2		
63. GL_EXT_discard_framebuffer		YES
64. GL_EXT_blend_minmax	YES	YES
65. GL_EXT_read_format_bgra	YES	YES
66. GL_IMG_program_binary		
67. GL_IMG_shader_binary		
68. GL_EXT_multi_draw_arrays	YES	YES
GL_SUN_multi_draw_arrays	NO	
69. GL_QCOM_tiled_rendering		
70. GL_OES_vertex_array_object		YES
71. GL_NV_coverage_sample		
72. GL_NV_depth_nonlinear		
73. GL_IMG_multisampled_render_to_texture		
74. GL_OES_EGL_sync	YES	YES
75. GL_APPLE_rgb_422		
76. GL_EXT_shader_texture_lod		

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
77. GL_APPLE_framebuffer_multisample		
78. GL_APPLE_texture_format_BGRA8888		
79. GL_APPLE_texture_max_level		
80. GL_ARM_mali_shader_binary		
81. GL_ARM_rgba8		
82. GL_ANGLE_framebuffer_blit		
83. GL_ANGLE_framebuffer_multisample		
84. GL_VIV_shader_binary		
85. GL_EXT_frag_depth		YES
86. GL_OES_EGL_image_external	YES	YES
87. GL_DMP_shader_binary		
88. GL_QCOM_alpha_test		
89. GL_EXT_unpack_subimage		
90. GL_NV_draw_buffers		
91. GL_NV_fbo_color_attachments		
92. GL_NV_read_buffer		
93. GL_NV_read_depth_stencil		
94. GL_NV_texture_compression_s3tc_update		
95. GL_NV_texture_npot_2D_mipmap		
96. GL_EXT_color_buffer_half_float		CORE
97. GL_EXT_debug_label		
98. GL_EXT_debug_marker		
99. GL_EXT_occlusion_query_boolean		
100. GL_EXT_separate_shader_objects		
101. GL_EXT_shadow_samplers		
102. GL_EXT_texture_rg		YES
103. GL_NV_EGL_stream_consumer_external		
104. GL_EXT_sRGB		YES
105. GL_EXT_multisampled_render_to_texture		YES
106. GL_EXT_robustness		YES
107. GL_EXT_texture_storage		
108. GL_ANGLE_instanced_arrays		
109. GL_ANGLE_pack_reverse_row_order		
110. GL_ANGLE_texture_compression_dxt3		
GL_ANGLE_texture_compression_dxt1		
GL_ANGLE_texture_compression_dxt5		

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
111. GL_ANGLE_texture_usage		
112. GL_ANGLE_translated_shader_source		
113. GL_FJ_shader_binary_GCCSO		
114. GL_OES_required_internalformat		YES
115. GL_OES_surfaceless_context		YES
116. GL_KHR_texture_compression_astc_hdr		
GL_KHR_texture_compression_astc_ldr		YES
117. GL_KHR_debug		YES
118. GL_QCOM_binning_control		
119. GL_ARM_mali_program_binary		
120. GL_EXT_map_buffer_range		
121. GL_EXT_shader_framebuffer_fetch		CORE
GL_EXT_shader_framebuffer_fetch_non_coherent		
122. GL_APPLE_copy_texture_levels		
123. GL_APPLE_sync		
124. GL_EXT_multiview_draw_buffers		
125. GL_NV_draw_texture		
126. GL_NV_packed_float		
127. GL_NV_texture_compression_s3tc		
128. GL_NV_3dvision_settings		
129. GL_NV_texture_compression_latc		
130. GL_NV_platform_binary		
131. GL_NV_pack_subimage		
132. GL_NV_texture_array		
133. GL_NV_pixel_buffer_object		
134. GL_NV_bgr		
135. GL_OES_depth_texture_cube_map		YES
136. GL_EXT_color_buffer_float		CORE
137. GL_ANGLE_depth_texture		
138. GL_ANGLE_program_binary		
139. GL_IMG_texture_compression_pvrtc2		
140. GL_NV_draw_instanced		
141. GL_NV_framebuffer_blit		
142. GL_NV_framebuffer_multisample		
143. GL_NV_generate_mipmap_sRGB		
144. GL_NV_instanced_arrays		

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
145. GL_NV_shadow_samplers_array		
146. GL_NV_shadow_samplers_cube		
147. GL_NV_sRGB_formats		
148. GL_NV_texture_border_clamp		
149. GL_EXT_disjoint_timer_query		
150. GL_EXT_draw_buffers		
151. GL_EXT_texture_sRGB_decode		YES
152. GL_EXT_sRGB_write_control		
153. GL_EXT_texture_compression_s3tc		YES
154. GL_EXT_pvrtc_sRGB		
155. GL_EXT_instanced_arrays		
156. GL_EXT_draw_instanced		
157. GL_NV_copy_buffer		
158. GL_NV_explicit_attrib_location		
159. GL_NV_non_square_matrices		
160. GL_EXT_shader_integer_mix		
161. GL_OES_texture_compression_astc		
162. GL_NV_blend_equation_advanced		
GL_NV_blend_equation_advanced_coherent		
163. GL_INTEL_performance_query		
164. GL_ARM_shader_framebuffer_fetch		
165. GL_ARM_shader_framebuffer_fetch_depth_stencil		
166. GL_EXT_shader_pixel_local_storage		
167. GL_KHR_blend_equation_advanced		CORE
GL_KHR_blend_equation_advanced_coherent		
168. GL_OES_sample_shading		CORE
169. GL_OES_sample_variables		CORE
170. GL_OES_shader_image_atomic		CORE
171. GL_OES_shader_multisample_interpolation		CORE
172. GL_OES_texture_stencil8		CORE
173. GL_OES_texture_storage_multisample_2d_array		CORE
174. GL_EXT_copy_image		CORE
175. GL_EXT_draw_buffers_indexed		CORE
176. GL_EXT_geometry_shader		CORE
GL_EXT_geometry_point_size		CORE
177. GL_EXT_gpu_shader5		CORE

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
178. GL_EXT_shader_implicit_conversions		CORE
179. GL_EXT_shader_io_blocks		CORE
180. GL_EXT_tessellation_shader		CORE
GL_EXT_tessellation_point_size		CORE
181. GL_EXT_texture_border_clamp		CORE
182. GL_EXT_texture_buffer		CORE
183. GL_EXT_texture_cube_map_array		CORE
184. GL_EXT_texture_view		
185. GL_EXT_primitive_bounding_box		CORE
186. GL_ANDROID_extension_pack_es31a		CORE
187. GL_EXT_compressed_ETC1_RGB8_sub_texture		
188. GL_KHR_robust_buffer_access_behavior		YES
189. GL_KHR_robustness		YES
190. GL_KHR_context_flush_control		
GLX_ARB_context_flush_control		
WGL_ARB_context_flush_control		
191. GL_DMP_program_binary		
192. GL_APPLE_clip_distance		
193. GL_APPLE_color_buffer_packed_float		
194. GL_APPLE_texture_packed_float		
195. GL_NV_internalformat_sample_query		
196. GL_NV_bindless_texture		
197. GL_NV_conditional_render		
198. GL_NV_path_rendering		
199. GL_NV_image_formats		
200. GL_NV_shader_noperspective_interpolation		
201. GL_NV_viewport_array		
202. GL_EXT_base_instance		
203. GL_EXT_draw_elements_base_vertex		CORE
204. GL_EXT_multi_draw_indirect		CORE
205. GL_EXT_render_snorm		
206. GL_EXT_texture_norm16		
207. GL_OES_copy_image		CORE
208. GL_OES_draw_buffers_indexed		CORE
209. GL_OES_geometry_shader		CORE
210. GL_OES_gpu_shader5		CORE

Table 13. OpenGL ES extension support*...continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
211. GL_OES_primitive_bounding_box		CORE
212. GL_OES_shader_io_blocks		CORE
213. GL_OES_tessellation_shader		CORE
GL_OES_tessellation_point_size		CORE
214. GL_OES_texture_border_clamp		CORE
215. GL_OES_texture_buffer		CORE
216. GL_OES_texture_cube_map_array		CORE
217. GL_OES_texture_view		CORE
218. GL_OES_draw_elements_base_vertex		CORE
219. GL_OES_EGL_image_external_essl3		CORE
220. GL_EXT_texture_sRGB_R8		
221. GL_EXT_YUV_target		
222. GL_EXT_texture_sRGB_RG8		
223. GL_EXT_float_blend		
224. GL_EXT_post_depth_coverage		
225. GL_EXT_raster_multisample		
226. GL_EXT_texture_filter_minmax		
227. GL_NV_conservative_raster		
228. GL_NV_fragment_coverage_to_color		
229. GL_NV_fragment_shader_interlock		
230. GL_NV_framebuffer_mixed_samples		
231. GL_NV_fill_rectangle		
232. GL_NV_geometry_shader_passthrough		
233. GL_NV_path_rendering_shared_edge		
234. GL_NV_sample_locations		
235. GL_NV_sample_mask_override_coverage		
236. GL_NV_viewport_array2		
237. GL_NV_polygon_mode		
238. GL_EXT_buffer_storage		
239. GL_EXT_sparse_texture		
240. GL_OVR_multiview		
241. GL_OVR_multiview2		
242. GL_KHR_no_error		
243. GL_INTEL_framebuffer_CMAA		
244. GL_EXT_blend_func_extended		
245. GL_EXT_multisample_compatibility		

Table 13. OpenGL ES extension support...*continued*

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0/3.1/3.2
246. GL_KHR_texture_compression_astc_sliced_3d		
247. GL_OVR_multiview_multisampled_render_to_texture		
248. GL_IMG_texture_filter_cubic		
249. GL_EXT_polygon_offset_clamp		
250. GL_EXT_shader_pixel_local_storage2		
251. GL_EXT_shader_group_vote		
252. GL_IMG_framebuffer_downsample		
253. GL_EXT_protected_textures		
254. GL_EXT_clip_cull_distance		
255. GL_NV_viewport_swizzle		
256. GL_EXT_sparse_texture2		
257. GL_NV_gpu_shader5		
258. GL_NV_shader_atomic_fp16_vector		
259. GL_NV_conservative_raster_pre_snap_triangles		
260. GL_EXT_window_rectangles		
261. GL_EXT_shader_non_constant_global_initializers		
262. GL_INTEL_conservative_rasterization		
263. GL_NVX_blend_equation_advanced_multi_draw_buffers		
264. GL_OES_viewport_array		
265. GL_EXT_conservative_depth		

3.4 Extension GL_VIV_direct_texture

Name	VIV_direct_texture
Name strings	GL_VIV_direct_texture
IPStatus	Contact NXP Semiconductor regarding any intellectual property questions associated with this extension.
Status	Implemented: July, 2011
Version	Last modified: 29 July, 2011 Revision: 2
Number	Unassigned
Dependencies	OpenGL ES 1.1 is required. OpenGL ES 2.0/3.x support is available.
Overview	Create a texture with direct access support. This is useful when an application desires to use the same texture over and over while frequently updating its content. It could also be used for mapping live video to a texture. A video decoder could write its result directly to the texture and then the texture could be directly rendered onto a 3D shape. glTexDirectVIVMap is similar to glTexDirectVIV. The only difference is that it has two inputs, “Logical” and “Physical,” which

	support mapping a user space memory or a physical address into the texture surface.

3.4.1 New Procedures and Functions

glTexDirectVIV

Syntax:

```
GL_API void GL_APIENTRY
glTexDirectVIV (
GLenum Target,
GLsizei Width,
GLsizei Height,
GLenum Format,
GLvoid ** Pixels
);
```

Parameters

Target	Target texture. Must be GL_TEXTURE_2D.
Width **Height**	Size of LOD 0. Width must be 16 pixel aligned. The width and height of LOD 0 of the texture is specified by the Width and Height parameters. The driver may auto-generate the rest of LODs if the hardware supports high quality scaling (for non-power of 2 textures) and LOD generation. If the hardware does not support high quality scaling and LOD generation, the texture remains a single-LOD texture.
Format	Choose the format of the pixel data from the following formats: GL_VIV_YV12, GL_VIV_NV12, GL_VIV_NV21, GL_VIV_YUY2, GL_VIV_UYVY, GL_RGBA, and GL_BGRA_EXT. • If the format is GL_VIV_YV12, glTexDirectVIV creates a planar YV12 4:2:0 texture and the format of the Pixels array is as follows: Yplane, Vplane, Uplane. • If the format is GL_VIV_NV12, glTexDirectVIV creates a planar NV12 4:2:0 texture and the format of the Pixels array is as follows: Yplane, UVplane. • If the format is GL_VIV_NV21, glTexDirectVIV creates a planar NV21 4:2:0 texture and the format of the Pixels array is as follows: Yplane, VUplane. • If the format is GL_VIV_YUY2 or GL_VIV_UYVY, glTexDirectVIV creates a packed 4:2:2 texture and the Pixels array contains only one pointer to the packed YUV texture. • If Format is GL_RGBA, glTexDirectVIV creates a pixel array with four GL_UNSIGNED_BYTE components: the first byte for red pixels, the second byte for green pixels, the third byte for blue, and the fourth byte for alpha. • If Format is GL_BGRA_EXT, glTexDirectVIV creates a pixel array with four GL_UNSIGNED_BYTE components: the first byte for blue pixels, the second byte for green pixels, the third byte for red, and the fourth byte for alpha.
Pixels	Stores the memory pointer created by the driver.

Output

If the function succeeds, it returns a pointer, or, for some YUV formats, it returns a set of pointers that directly point to the texture. The pointer(s) are returned in the user-allocated array pointed to by the Pixels parameter.

GlTexDirectVIVMap

Syntax:

```
GL_API void GL_APIENTRY
glTexDirectVIVMap (
GLenum Target,
GLsizei Width,
GLsizei Height,
GLenum Format,
GLvoid ** Logical,
const GLuint * Physical
);
```

Parameters

Target	Target texture. Must be GL_TEXTURE_2D.
Width **Height**	Size of LOD 0. Width must be 16 pixel aligned. See glTexDirectVIV.
Format	Same as glTexDirectVIV Format.
Logical	Pointer to the logical address of the application-defined texture buffer. Logical address must be 64 bit (8 byte) aligned.
Physical	Pointer to the physical address of the application-defined buffer to the texture, or ~0 if no physical address has been provided.

GlTexDirectInvalidateVIV

Syntax:

```
GL_API void GL_APIENTRY
glTexDirectInvalidateVIV (
GLenum Target
);
```

Parameters

Target	Target texture. Must be GL_TEXTURE_2D.

New Tokens

GL_VIV_YV12	0x8FC0
GL_VIV_NV12	0x8FC1
GL_VIV_YUY2	0x8FC2
GL_VIV_UYVY	0x8FC3
GL_VIV_NV21	0x8FC4

Error codes

GL_INVALID_ENUM	Target is not GL_TEXTURE_2D, or format is not a valid format.
GL_INVALID_VALUE	Width or Height parameter is less than 1.
GL_OUT_OF_MEMORY	A memory allocation error occurred.

GL_INVALID_OPERATION	Specified format is not supported by the hardware, or no texture is bound to the active texture unit, or some other error occurs during the call.

Example 1.

First, call `glTexDirectVIV` to get a pointer.

Second, copy the texture data to this memory address.

Then, call `glTexDirectInvalidateVIV` to apply the texture before drawing something with that texture.

```
… …
glTexDirectVIV(GL_TEXUTURE_2D, 512, 512, GL_VIV_YV12, &texels);
… …
GLTexDirectInvalidateVIV(GL_TEXTURE_2D);
…
glDrawArrays(…);
…
```

Example 2.

First, call `glTexDirectVIVMap` to map Logical and Physical address to the texture.

Second, modify Logical and Physical data.

Then, call `glTexDirectInvalidateVIV` to apply the texture before drawing something with that texture.

```
… …
char *Logical = (char*) malloc (sizeof(char)*size);
GLuint physical = ~0U;
glTexDirectVIVMap(GL_TEXUTURE_2D, 512, 512, GL_VIV_YV12,
 (void**)&Logical, &physical);
… …
GLTexDirectInvalidateVIV(GL_TEXTURE_2D);
…
glDrawArrays(…);
```

Issues

None

3.5 Extension GL_VIV_texture_border_clamp

Name

VIV_texture_border_clamp

Name Strings

GL_VIV_texture_border_clamp

Status

Implemented September 2012.

Version

Last modified: 27 September 2012

Vivante revision: 1

Number

Unassigned

Dependencies

This extension is implemented for use with OpenGL ES 1.1 and OpenGL ES 2.0.

This extension is based on OpenGL ARB Extension #13: GL_ARB_texture_border_clamp: www.opengl.org/registry/specs/ARB/texture_border_clamp.txt. See also vendor extension GL_SGIS_texture_border_clamp: www.opengl.org/registry/specs/SGIS/texture_border_clamp.txt.

Overview

This extension was adapted from the OpenGL extension for use with OpenGL ES implementations. The OpenGL ARB Extension 13 description applies here as well:

“The base OpenGL provides clamping such that the texture coordinates are limited to exactly the range [0,1]. When a texture coordinate is clamped using this algorithm, the texture sampling filter straddles the edge of the texture image, taking 1/2 its sample values from within the texture image, and the other 1/2 from the texture border. It is sometimes desirable for a texture to be clamped to the border color, rather than to an average of the border and edge colors.

This extension defines an additional texture clamping algorithm. CLAMP_TO_BORDER_[VIV] clamps texture coordinates at all mipmap levels such that NEAREST and LINEAR filters return only the color of the border texels.”

The color returned is derived only from border texels and cannot be configured.

Issues

None

New Tokens

Accepted by the <param> parameter of TexParameteri and TexParameterf, and by the <params> parameter of TexParameteriv and TexParameterfv, when their <pname> parameter is TEXTURE_WRAP_S, TEXTURE_WRAP_T, or TEXTURE_WRAP_R:

CLAMP_TO_BORDER_VIV	0x812D

Errors

None.

New State

Only the type information changes for these parameters.

See OES 2.0 Specification Section 3.7.4, page 75-76, Table 3.10, “Texture parameters and their values.”

4 i.MX Framebuffer API

4.1 Overview

The graphics software includes i.MX Framebuffer (FB) API which enables users to easily create and port their graphics applications by using a framebuffer device without the need to expend additional effort handling platform-related tasks. i.MX Framebuffer API focuses on providing mechanisms for controlling display, window, and pixmap render surfaces.

The EGL Native Platform Graphics Interface provides mechanisms for creating rendering surfaces onto which client APIs can draw, creating graphics contexts for client APIs, and synchronizing drawing by client APIs as well as native platform rendering APIs. This enables seamless rendering using Khronos APIs such as OpenGL ES and OpenVG for high-performance, accelerated, mixed-mode 2D, and 3D rendering. For further information on EGL, see www.khronos.org/registry/egl. The API described in this document is compatible with EGL version 1.4 of the specification.

The following platforms are supported:

- Linux OS/X11
- Android platform
- Windows Embedded Compact OS
- QNX

Note:

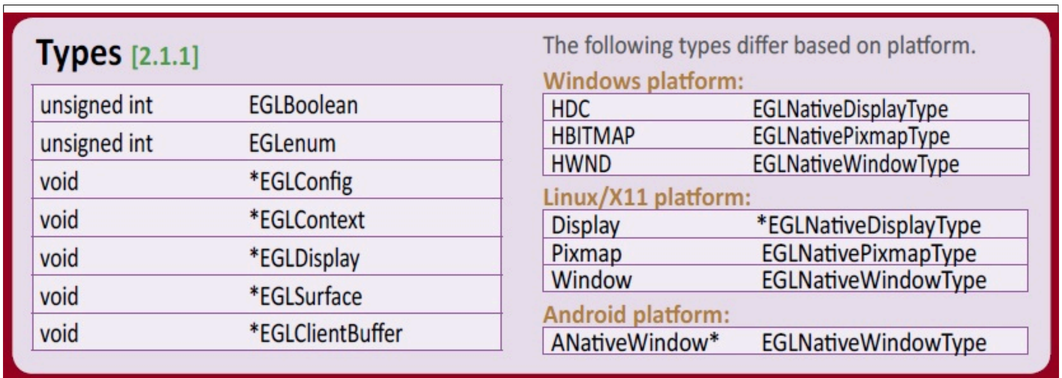
i.MX 8 on Linux OS supports Direct Rendering Manager (DRM) where the Linux framebuffer support is limited, recommended to Graphics Buffer Manager (GBM).

4.2 API data types and environment variables

4.2.1 Data types

The GPU software provides platform independent member definitions for the following EGL types:

```
typedef struct _FBDisplay   *  EGLNativeDisplayType;
typedef struct _FBWindow *  EGLNativeWindowType;
typedef struct _FBPixmap   *  EGLNativePixmapType;
```



Types [2.1.1]

unsigned int	EGLBoolean
unsigned int	EGLenum
void	*EGLConfig
void	*EGLContext
void	*EGLDisplay
void	*EGLSurface
void	*EGLClientBuffer

The following types differ based on platform.

Windows platform:

HDC	EGLNativeDisplayType
HBITMAP	EGLNativePixmapType
HWND	EGLNativeWindowType

Linux/X11 platform:

Display	*EGLNativeDisplayType
Pixmap	EGLNativePixmapType
Window	EGLNativeWindowType

Android platform:

ANativeWindow*	EGLNativeWindowType

Figure 3. Types as listed on EGL 1.4 API Quick Reference Card

(from www.khronos.org/files/egl-1-4-quick-reference-card.pdf)

4.2.2 Environment variables

Table 14. i.MX FB API environment variables

Environment Variables	Description
FB_MULTI_BUFFER	To use multiple-buffer rendering, set the environment variable FB_ MULTI_BUFFER to an unsigned integer value, which indicates the number of buffers required. The maximum is 8. **Recommended values: 4.** The FB_MULTI_BUFFER variable can be set to any positive integer value. • If set to 1, the multiple-buffer function is not enabled, and the VSYNC is also disabled, so there may be tearing on screen, but it is good for benchmark test. • If set to 2 or 3, VSYNC is enabled and there are double or trible frame buffer. Because of the hardware limitation of current IPU, there may be tearing on screen. • If set to 4 or more, VSYNC is enabled and no screen tearing appears. • If set to a value more than 8, the driver uses 8 as the buffer count.
FB_FRAMEBUFFER_0, FB_FRAMEBUFFER_1, FB_FRAMEBUFFER_2, FB_FRAMEBUFFER_n	To open a specified framebuffer device, set the environment variable **FB_FRAMEBUFFER_n** to a proper value (for example, FB_ FRAMEBUFFER_0 = /dev/fb0). **Allowed values for n: any positive integer.** Note: If there are no environment variables set, the driver tries to use the default framebuffer devices (fb0 for index 0, fb1 for index 1, fb2 for index 2, fb3 for index 3, and so on).
FB_IGNORE_DISPLAY_ SIZE	When set to a positive integer and a window's initial size request is greater than the display size, the window size is not reduced to fit within the display. Global. **Allowed values: any positive integer.** Note: The drivers read the value from this environment variable as a Boolean to check if the user wants to ignore the display size when creating a window. • If the variable is set to value, 0, or this environment variable is not set, when creating window, the driver uses display size to cut down the size of the window to ensure that the entire window area is inside the display screen. • If the user sets this variable to 1, or any positive integer value, then the window area can be partly or entirely outside of the display screen area (see the image below in which the ignore display size is equal to 1).
GPU_VIV_DISABLE_ CLEAR_FB	It turns off zero fill memory, so the content of FBDEV buffer is not cleared.
FB_LEGACY	If the board support drm-fb, the gpu will render though drm by default. If the user wants to render to framebuffer directly instead of through drm, sets this variable to 1.

Below are some usage syntax examples for environment variables:

To create a window with its size different from the display size, use the environment variable **FB_IGNORE_DISPLAY_SIZE**. Example usage syntax:

```
export FB_IGNORE_DISPLAY_SIZE=1
```

To let the driver use multiple buffers to do swap work, use the environment variable **FB_MULTI_BUFFER.** Example usage syntax:

```
export FB_MULTI_BUFFER=2
```

To specify the display device, use the environment variable **FB_FRAMEBUFFER_n**, where n = any positive integer. Example usage syntax:

```
export FB_FRAMEBUFFER_0=/dev/fb0
export FB_FRAMEBUFFER_1=/dev/fb1
export FB_FRAMEBUFFER_2=/dev/fb2
export FB_FRAMEBUFFER_3=/dev/fb3
```

4.3 API description and syntax

fbGetDisplay:

Description	This function is used to get the default display of the framebuffer device. To open the framebuffer device, set an environment variable `FB_FRAMEBUFFER_n` to the framebuffer location.
Syntax	`EGLNativeDisplayType` `fbGetDisplay (` `void *        context` `);`
Parameters	`context`: Pointer to the native display instance.
Return Values	The function returns a pointer to the EGL native display instance if successful; otherwise, it returns a NULL pointer.

fbGetDisplayByIndex:

Description	This function is used to get a specified display within a multiple framebuffer environment by providing an index number. To use multiple buffers when rendering, set the environment variable `FB_MULTI_BUFFER` to an unsigned integer value, which indicates the number of buffers. Maximum is **3**. To open a specific Framebuffer device, set environment variables to their proper values (e.g., set `FB_FRAMEBUFFER_0 = /dev/fb0`). If there are no environment variables set, the driver tries to use the default fb devices (fb0 for index 0, fb1 for index 1, fb2 for index 2, fb3 for index 3, and so on).
Syntax	`EGLNativeDisplayType` `fbGetDisplayByIndex (` `int           DisplayIndex` `);`

Parameters	`DisplayIndex`: An integer value where the integer is associated with one of the following environment variables for framebuffer devices: `FB_FRAMEBUFFER_0` `FB_FRAMEBUFFER_1` `FB_FRAMEBUFFER_2` `FB_FRAMEBUFFER_n`
Return Value	The function returns a pointer to the EGL native display instance if successful; otherwise, it returns a NULL pointer.

fbGetDisplayGeometry:

Description	This function is used to get display width and height information.
Syntax	`void` `fbGetDisplayGeometry (` `EGLNativeDisplayType   Display,` `int *                  Width,` `int *                  Height` `);`
Parameters	`Display`: [in] Pointer to EGL native display instance created by `fbGetDisplay`. `Width`: [out] Pointer that receives the width of the display. `Height`: [out] Pointer that receives the height of the display.

fbGetDisplayInfo:

Description	This function is used to get display information.
Syntax	`void` `fbGetDisplayInfo (` `EGLNativeDisplayType   Display,` `int *                  Width,` `int *                  Height,` `unsigned long *        Physical,` `int *                  Stride,` `int *                  BitsPerPixel` `);`
Parameters	`Display`: [in] A pointer to the EGL native display instance created by `fbGetDisplay`. `Width`: [out] A pointer to the location that contains the width of the display.

	`Height`: [out] A pointer to the location that contains the height of the display. `Physical`: [out] A pointer to the location that contains the physical start address of the display. `Stride`: [out] A pointer to the location that contains the stride of the display. `BitsPerPixel:` [out] A pointer to the location that contains the pixel depth of the display.

fbDestroyDisplay:

Description	This function is used to destroy a display.
Syntax	<pre>void fbDestroyDisplay (EGLNativeDisplayType Display);</pre>
Parameters	`Display`: [in] Pointer to EGL native display instance created by `fbGetDisplay`.

fbCreateWindow:

Description	This function is used to create a window for the framebuffer platform with the specified position and size. If width/height is 0, it uses the display width/height as its value. Note: When either window X + width or the Y + height is larger than the display's width or height respectively, the API reduces the window size to force the whole window inside the display screen limits. To avoid reducing the window size in this scenario, users can set a value of "1" to the environment variable `FB_IGNORE_DISPLAY_SIZE`.
Syntax	<pre>EGLNativeWindowType fbCreateWindow (EGLNativeDisplayType Display, int X, int Y, int Width, int Height);</pre>
Parameters	`Display`: [in] Pointer to EGL native display instance created by `fbGetDisplay`. `X`: [in] Specifies the initial horizontal position of the window. `Y`: [in] Specifies the initial vertical position of the window. `Width`: [in] Specifies the width of the window.

	`Height`: [in] Specifies the height of the window in device units.
Return Value	The function returns a pointer to the EGL native window instance if successful; otherwise, it returns a NULL pointer.

fbGetWindowGeometry:

Description	This function is used to get window position and size information.
Syntax	`void` `fbGetWindowGeometry (` `EGLNativeWindowType    Window,` `int *                  X,` `int *                  Y,` `int *                  Width,` `int *                  Height` `);`
Parameters	`Window`: [in] Pointer to EGL native window instance created by `fbCreateWindow`. `X`: [out] Pointer that receives the horizontal position value of the window. `Y`: [out] Pointer that receives the vertical position value of the window. `Width`: [out] Pointer that receives the width value of the window. `Height`: [out] Pointer that receives the height value of the window.

fbGetWindowInfo:

Description	This function is used to get window position and size and address information.
Syntax	`void` `fbGetWindowInfo (` `EGLNativeWindowType   Window,` `int *                 X,` `int *                 Y,` `int *                 Width,` `int *                 Height` `int *                 BitsPerPixel,` `unsigned int *        Offset` `);`
Parameters	`Window`: [in] A pointer to the EGL native window instance created by `fbCreateWindow`. `X`: [out] A pointer to the location that contains the horizontal position value of the window. `Y`: [out] A pointer to the location that contains the vertical position value of the window.

	Width: [out] A pointer to the location that contains the width of the window. Height: [out] A pointer to the location that contains the height of the window. BitsPerPixel: [out] A pointer to the location that contains the pixel depth of the window. Offset: [out] A pointer to the location that contains the offset of the window.

fbDestroyWindow:

Description	This function is used to destroy a window.
Syntax	`void` `fbDestroyWindow (` `EGLNativeWindowType    Window` `);`
Parameters	Window: [in] Pointer to EGL native window instance created by fbCreateWindow.

fbCreatePixmap:

Description	This function is used to create a pixmap of a specific size on the specified framebuffer device. If either the width or height is 0, the function fails to create a pixmap and return NULL.
Syntax	`EGLNativePixmapType` `fbCreatePixmap (` `EGLNativeDisplayType    Display,` `int                     Width,` `int                     Height` `);`
Parameters	Display: [in] Pointer to the EGL native display instance created by fbGetDisplay. Width: [in] Specifies the width of the pixmap. Height: [in] Specifies the height of the pixmap.
Return Value	The function returns a pointer to the EGL native pixmap instance if successful; otherwise, it returns a NULL pointer.

fbCreatePixmapWithBpp:

Description	This function is used to create a pixmap of a specific size and bit depth on the specified framebuffer device. If either the width or height is 0, the function fails to create a pixmap and return NULL.
Syntax	`EGLNativePixmapType` `fbCreatePixmapWithBpp (` `EGLNativeDisplayType    Display,` `int                     Width,` `int                     Height` `int                     BitsPerPixel` `);`
Parameters	`Display`: [in]A pointer to the EGL native display instance created by `fbGetDisplay`. `Width`: [in] Specifies the width of the pixmap. `Height`: [in] Specifies the height of the pixmap. `BitsPerPixel`: [in] Specifies the bit depth of the pixmap.
Return Value	The function returns a pointer to the EGL native pixmap instance if successful; otherwise, it returns a NULL pointer.

fbGetPixmapGeometry:

Description	This function is used to get pixmap size information.
Syntax	`void` `fbGetPixmapGeometry (` `EGLNativePixmapType    Pixmap,` `int *                  Width,` `int *                  Height` `);`
Parameters	`Pixmap`: [in] Pointer to the EGL native pixmap instance created by `fbCreatePixmap`. `Width`: [out] Pointer that receives a width value for pixmap. `Height`: [out] Pointer that receives a height value for pixmap.

fbGetPixmapInfo:

Description	This function is used to get pixmap size and depth information.
Syntax	`void` `fbGetPixmapInfo (` `EGLNativePixmapType    Pixmap,` `int *                  Width,` `int *                  Height`

	`` int * BitsPerPixel int * Stride, void ** Bits); ```
Parameters	`Pixmap`: [in] A pointer to the EGL native pixmap instance created by `fbCreatePixmap`. `Width`: [out] A pointer to the location that contains a width value for pixmap. `Height`: [out] A pointer to the location that contains a height value for pixmap. `BitsPerPixel`: [out] A pointer to the location that contains the pixel depth of the pixmap. `Stride`: [out] A pointer to the location that contains the stride of the pixmap. `Bits`: [out] A pointer to the location that contains the bit address of the pixmap.

fbDestroyPixmap:

Description	This function is used to destroy a pixmap.
Syntax	`void` `fbDestroyPixmap (` `EGLNativePixmapType    Pixmap` `);`
Parameters	`Pixmap`: [in] Pointer to the EGL native pixmap instance created by `fbCreatePixmap`.

5 OpenCL

5.1 Overview

5.1.1 General description

Open Computing Language (OpenCL) is an open industry standard application programming interface (API) used to program multiple devices including GPUs, CPUs, as well as other devices organized as part of a single computational platform. The OpenCL standard targets a wide range of devices from mobile phones, tablets, PCs, and consumer electronic (CE) devices, all the way to embedded applications such as automotive and image processing functions. The API takes advantage of all resources in a platform to fully utilize all compute capability and to efficiently process the growing complexity of incoming data streams from multiple I/O (input/output) sources. I/O streams can be camera inputs, images, scientific or mathematical data, and any other form of complex data that can make use of data or task parallelism.

OpenCL uses parallel execution SIMD (single instruction, multiple data) engines found in GPUs to enhance data computational density by performing massively parallel data processing on multiple data items, across multiple compute engines. Each compute unit has its own arithmetic logic units (ALUs), including pipelined floating point (FP), integer (INT) units and a special function unit (SFU) that can perform computations as well as transcendental operations. The parallel computations and associated series of operations are called a kernel, and the GPU cores can execute a kernel on thousands of work-items in parallel at any given time.

At a high level, OpenCL provides both a programming language and a framework to enable parallel programming. OpenCL includes APIs, libraries and a runtime system to assist and support software development. With OpenCL, it is possible to write general purpose programs that can execute directly on GPUs, without needing to know graphics architecture details or using 3D graphics APIs like OpenGL or DirectX. OpenCL also provides a low-level Hardware Abstraction Layer (HAL) as well as a framework that exposes many details of the underlying hardware layer and thus allows the programmer to take full advantage of the hardware.

For more details on all the capabilities of OpenCL, see the following specifications from the Khronos Group:

• OpenCL 1.2 Specification

www.khronos.org/registry/cl/specs/opencl-1.2.pdf

• OpenCL 1.2 C++ Bindings Specification

www.khronos.org/registry/cl/specs/opencl-cplusplus-1.2.pdf

5.1.2 OpenCL framework

The OpenCL framework has two principal parts, similar to OpenGL, the host C API and the device C-based language runtime. The host in OpenCL terminology corresponds to the client in OpenGL and the device corresponds to the server. Device programs are called kernels. Execution of an OpenCL program is preceded by a series of API calls that configure the system and GPGPU for execution.

OpenCL abstracts today's heterogeneous architectures using a hierarchical platform model. A host coordinates the execution and data transfers on, to and from one or several compute devices. Compute devices are comprised of compute units and each such unit contains an array of processing elements.

5.1.2.1 OpenCL execution model: kernels and work elements

The OpenCL execution model is defined by how the kernels are executed. When a kernel is submitted for execution by the host, an index space is defined. An instance of the kernel executes for each point in this index space. This kernel instance is called a **work-item**. Work-items are identified by their position in the index space that provides the global ID for the work-item. Each work-item executes the same code but the specific pathway through the code and the data operated upon varies by work-item.

Work-items are organized into **work-groups**. Work-groups provide a broader decomposition of the index space. Work-groups are each assigned a unique work-group ID with the same dimensionality as the index space used for the work-items. Work-items are assigned a unique local ID within a work-group so that a single work-item can be uniquely identified by its global ID or by a combination of its local ID and work-group ID. The work-items in a given work-group execute concurrently on the same compute device.

The index space supported in OpenCL is called an **NDRange**. An NDRange is an N-dimensional index space, where **N** is one (1), two (2) or three (3). An NDRange is defined by an integer array of length N specifying the extent of the index space in each dimension starting at an offset index **F** (zero by default). Each work-item’s global ID and local ID are N-dimensional tuples. The global ID components are values in the range from F, to F plus the number of elements in that dimension minus one.

Work-groups are assigned IDs using a similar approach to that used for work-item global IDs. An array of length N defines the number of work-groups in each dimension. Work-items are assigned to a work-group and given a local ID with components in the range from zero to the size of the work-group in that dimension minus one. Hence, the combination of a work-group ID and the local-ID within a work-group uniquely defines a work-item. Each work-item is identifiable in two ways; in terms of a global index, unique through the whole kernel index space, and in terms of a local index, unique within a work group.

5.1.2.2 OpenCL command queues

OpenCL provides both task and data parallelism. Data movements are coordinated via **command queues**, which provide a general means of specifying inter-task relationships and task execution orders that obey the dependencies in the computation. OpenCL may execute several tasks in parallel, if they are not order dependent. Tasks are composed of data-parallel kernels which, similarly to shaders, apply a single function to a range of elements in parallel. Only restricted synchronization and communication is allowed during kernel execution.

OpenCL kernels execute over a 1, 2 or 3 dimensional index space. All work-items execute the same program (kernel) but their execution may diverge, with branching dependent on the data or their index. For details regarding how many work groups are allowed within an index space see “Using clEnqueueNDRangeKernel”.

A kernel or a memory operation is first **enqueued** onto a command queue. Kernels are executed asynchronously and the host application execution may proceed right after the enqueue operation. The application may opt to wait for an operation to complete and an operation (kernel or memory) may be marked with a list of events that must occur before it executes.

Events are kernel completion and memory operations. OpenCL traverses the dependence graph between the kernels and memory transfers in a queue and ensures the correct execution order. Multiple command queues may be constructed, further enhancing parallelism control across platforms and multiple compute devices.

- **Command-queue barriers** are used to control the commands within the command queue. The command-queue barrier indicates which commands must be finished before proceeding. This allows for out-of-order command processing. The command queue barrier ensures that all previously enqueued commands finish execution before any following commands begin execution.

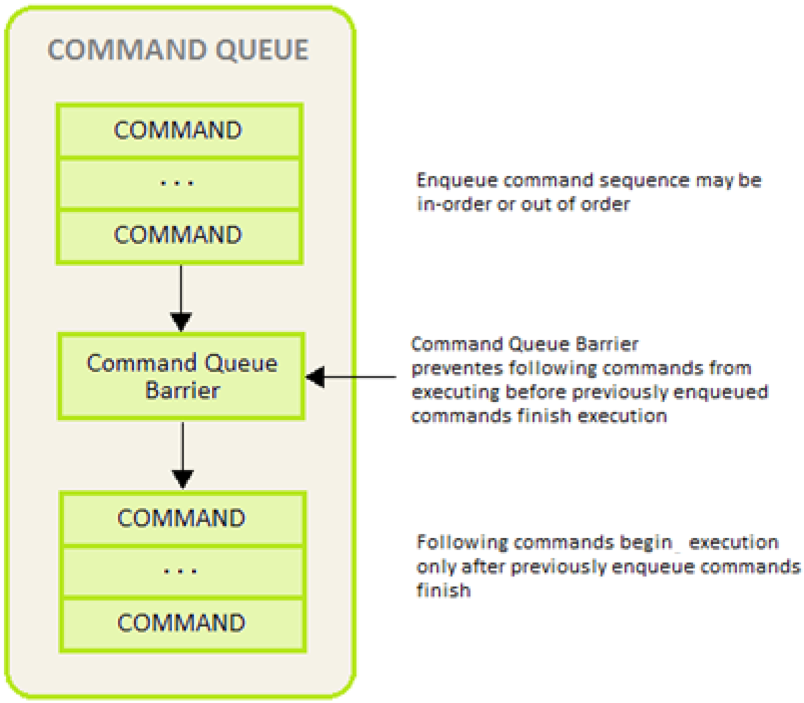


Figure 4. Command queue barrier

The work-group barrier built-in function provides control of the work-item flow within work-groups. All work-items must execute the barrier construct before any can continue execution beyond the barrier.

5.1.2.3 OpenCL memory model

The OpenCL memory model is divided into four different types of memory domains. These are:

• **Global Memory**: Each compute device has global memory space which can reside off-chip in system memory (DRAM) or inside the chip at the L1 or temporary register level. Global memory is accessible to all work-items executing in a context, as well as to the host (read, write, and map commands).

• **Constant Memory**: is also global memory, but it is read-only. Constant memory can be placed in any level of memory that the application programmer decides, making it an implementation dependent decision. This is the region for host-allocated and host-initialized objects that are not changed during kernel execution.

• **Local Memory**: Each compute unit has local memory which resides very near the processing elements. Access to local memory is very fast and the size of local memory is much smaller than global memory, making it a scarce resource that needs to be controlled for optimal communication of work-items inside a work-group. Local memory is specific to a work-group, and is accessible only by work-items belonging to that work group.

• **Private Memory**: Each processing element has another level of memory called private memory, which is only accessible to a single work-item. Private memory is specific to a work-item and is not visible to other work-items.

During run-time, each processing element is assigned a set of on-chip registers that are used for data storage of intermediate data. Data that cannot be stored in registers spills over to global memory which can be very costly in terms of performance and constant data movement to/from temporary registers. Software may emulate local and private memory using global memory. System Memory is often loaded to L1 cache, Temporary or Local Storage Registers and the GPGPU reads from those locations. At every level of the application program, the programmer must be aware of the size and hierarchy of storage elements.

Table 15. Vivante memory structures mapped to Khronos OpenCL memory types

Khronos OpenCL Memory Model Name	Vivante GPGPU OpenCL Memory Structures Utilized	Definition
Private Memory	Registers, System Memory	Accessible only to an individual work-item; not visible to any other work-items
Local Memory	Local Storage Registers, System Memory	Accessible to all work-items within a specific work-group; accessible only by work-items belonging to that work-group
Global Memory	System Memory	Accessible to all-work-items executing in a context, as well as to the host (read, write, and map commands).
Constant Memory	Constant Registers, System Memory	Read only global memory region for host-allocated and initialized objects that are not changed during kernel execution
Host (CPU) Memory	Host Memory	Region for a kernel application’s program data and structures

The OpenCL concurrent-read /concurrent-write (CRCW) memory model has so-called relaxed consistency which means that different work-items may see a different view of global memory as the computation proceeds. Within individual work-items reads and writes to all memory spaces are ordered. Synchronization between work-items in a work-group is necessary to ensure consistency. No mechanism for synchronization between work-groups is provided. Such a model assures parallel scalability by requiring explicit synchronization and communication.

For the highest throughput and computational speed, kernels should use high-speed on-chip memories and registers as much as possible. Instruction control flow and memory operations, including data gathering / scattering and direct memory access (DMA) should be automatically reorganized / re-ordered depending on data dependencies detected by the optimized compiler. The Vivante OpenCL compiler automatically maps dependencies and re-orders instructions for the best performance.

5.1.2.4 Host to GPGPU compute device data transfers

The application running on the host uses the OpenCL API to create memory objects in global memory, and to enqueue memory commands that operate on these memory objects. The host and OpenCL device memory models are, for the most part, independent of each other. This is by necessity as the host is defined outside of OpenCL. They do, however, at times need to interact. This interaction occurs in one of two ways: by explicitly copying data from the host to the GPU compute device memory, or implicitly, by mapping and unmapping regions of a memory object.

• **Explicit** using clEnqueueReadBuffer and clEnqueueWriteBuffer (clEnqueueReadImage, clEnqueueWriteImage.)

To copy data explicitly, the host enqueues commands to transfer data between the memory object and host memory. These memory transfer commands may be blocking or non-blocking. The OpenCL function call for a blocking memory transfer returns once the associated memory resources on the host can be safely reused. For a non-blocking memory transfer, the OpenCL function call returns as soon as the command is enqueued regardless of whether host memory is safe to use.

• **Implicit** using clEnqueueMapBuffer and clEnqueueUnMapMemObject.

The mapping/unmapping method of interaction between the host and OpenCL memory objects allows the host to map a region from the memory object into its address space. The memory map command may be blocking or non-blocking. Once a region from the memory object has been mapped, the host can read or write to this region. The host unmaps the region when accesses (reads and/or writes) to this mapped region by the host are complete.

The OpenCL specification does not explicitly state where each memory space will be mapped to on individual implementations. This provides great freedom for vendors on the one hand and some uncertainty for programmers on the other. Fortunately, kernels may be compiled just-in-time and possible differences may be tackled during run-time.

When using these interfaces, it is important to consider the amount of copying involved to/from system memory and the various levels within the compute device(s). There is a two-copy process: between host and AXI (or SoC internal bus), and between AXI (or SoC internal bus) and the Vivante GPGPU compute device. Double copying lowers overall system memory bandwidth and lowers performance. Because of variations in system architecture (both internal and external/memory), there is sometimes a large performance delta between the system or calculated GFLOPS and the kernel or GPGPU GFLOPS. GPGPU GFLOPS are based on the theoretical computational capability of the ALUs within the GPGPU, assuming the system architecture can deliver full data to the GPGPU. OpenCL APIs for buffers and images aid in avoiding double copy by allowing the mapping of host memory to device memory. With proper memory transfer management and the use of host/CPU memory remapped to the GPGPU memory space, copying between host memory and GPGPU memory can be skipped so data transfer becomes a one-copy process. The trade-off is that the programmer needs to be mindful of page boundaries and memory alignment issues.

5.1.3 OpenCL profiles

In addition to Full Profile, the OpenCL specification also includes an Embedded Profile, which relaxes the OpenCL compliance requirements for mobile and embedded devices. The main commons and differences between OpenCL 1.1/1.2 EP (Embedded Profile) and FP (Full Profile) come down to:

Commons:

- Both EP and FP significantly offload the CPU of parallel, multi-threaded tasks.
- For both EP and FP double precision and half-precision floating point are optional.

Difference:

- Full Profile is for highly complex, accurate, and real time computations, while Embedded Profile is a small subset targeting smaller devices (handheld, mobile, embedded) that perform GPGPU/OpenCL processing with relaxed data type and precision requirements (image processing, augmented reality, gesture recognition, and more).
- 64-bit integers are required for FP and optional for EP.

- EP requires either RTZ or RTE. FP requires both.
- Computational precision (units in the last place; i.e., ULP) requirements in EP are relaxed.
- Atomic instruction support is not required in EP.
- 3D Image support is not required in EP.
- Minimum requirements for constant buffer size, object allocation size, constant argument counts and local memory sizes are scaled down in EP.
- And more (in general EP is a scaled down version of FP).
- Die size and power increase with FP because of the higher requirements, features and memory sizes.

5.1.4 Vivante OpenCL embedded compatible IP

As of the date of this document, select Vivante GPGPU cores are compatible with OpenCL Embedded Profile version 1.1. The folllowing table lists the hardware capability deltas.

Table 16. Vivante OpenCL embedded profile hardware

Hardware and revision	GC2000
Feature	**5.1.0.rc8a**
Compute Devices (GPGPU cores)	1
Compute Units per device (Shader cores)	4
Processing Elements per compute unit	4
Profile	Embedded
Preferred work-group/thread group size	16
Max count global work-items each dim	64K
Max count of work-items each dim per work-group	1K
Local Storage Registers On-chip	64
Instruction Memory	512
Texture Samplers	8 PS + 4 VS
Texture Samplers available to OCL (HW, unlimited via SW)	4
L1 Cache Size	4 KB
L1 Cache Banks	1
L1 Cache Sets/Bank	4
L1 Cache Ways/Set	16
L1 Cache Line Size	64B
L1 Cache MC ports	1

5.1.5 Vivante OpenCL full profile hardware model

As of the date of this document, select Vivante GPGPU cores are compatible with OpenCL Full Profile version 1.2. Hardware capability deltas are subject to change and includes:

Table 17. Vivante OpenCL full profile hardware

Hardware and revision	GC2000+	GC7000XSVX	GC7000L	GC7000UL
i.MX SoC	**i.MX 6Quad Plus, i.MX 6DualPlus**	**i.MX 8 QuadMax**	**i.MX 8M Quad, i.MX 8Quad XPlus**	**i.MX 8M Nano i.MX 8M Plus**
Compute Devices (GPGPU cores)	1	1	1	1
Compute Units per device (for sub-device)	1	1	1	1
Processing Elements per device	16	32	16	8
Profile	Full-Lite*	Full	Full	Full
Preferred work-group/ thread group size	16	32	16	8
Max count global work-items each dim (if 3D only 1 dim can be up to 4G, the others 64K)	4 G/64 K	4 G/64 K	4G	4G
Max count of work-items each dim per work-group	1 K	1 K	1K	1K
Local Storage Registers On-chip	0	2048 (32 K)	16 (KB)	
Instruction Memory	I$:512/1 M	8K	8K	8K
Texture Samplers	32 undefined	32 undefined	32	32
Texture Samplers available to OCL	32	32	32	32
L1 Cache Size	4 KB	64 KB	16KB	8 KB
L1 Cache Banks	2	4	2	1
L1 Cache Sets/Bank	2	8	N/A	8
L1 Cache Ways/Set	16	8	8	8
L1 Cache Line Size	64 B	64 B	64 B	64 B
L1 Cache MC ports per GPGPU core	2	2	2	1

5.2 Vivante OpenCL implementation

5.2.1 OpenCL pipeline

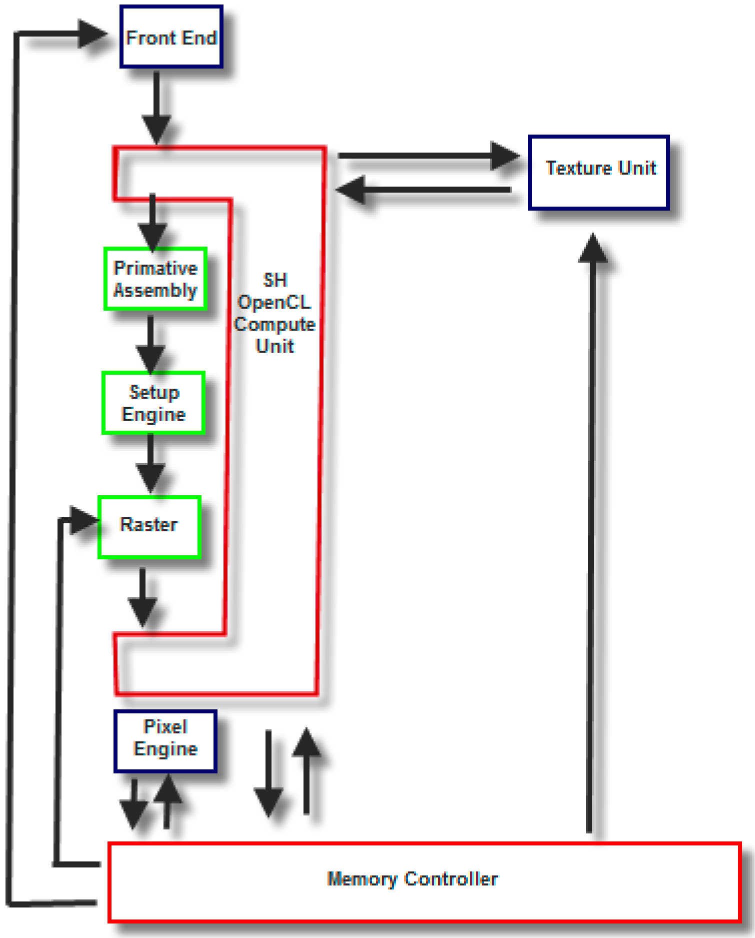


Figure 5. Vivante OpenCL data pipeline for an OpenCL compute device

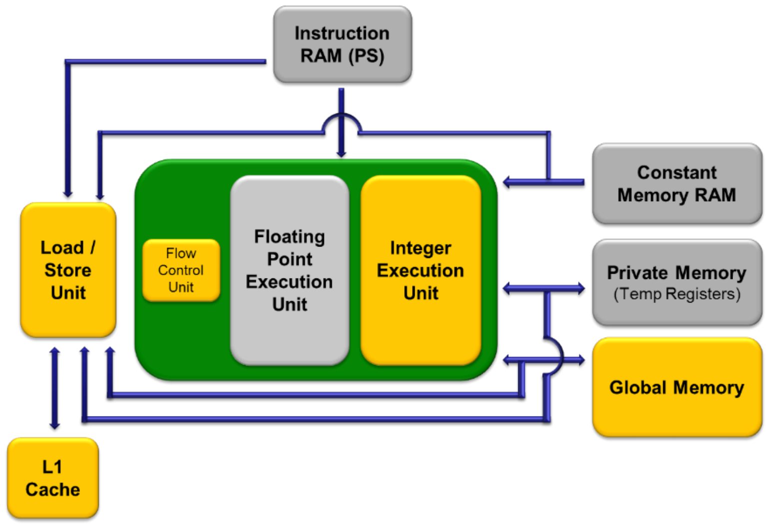


Figure 6. Vivante OpenCL compute device showing memory scheme

5.2.2 Front end

The front end passes the instructions and constant data as State Loads to the OpenCL Compute Unit (Shader) block. State Loads program instructions and constant data and work groups initiate execution on the instructions and the constants loaded.

5.2.3 OpenCL compute unit

All OpenCL executions occur in this block and all work-groups in a compute unit should belong to the same kernel. Threads from a work-group are grouped into internal “Thread-groups”. All the threads in a thread-group execute in parallel. Barrier instruction is supported to enforce synchronization within a work-group.

The compute unit contains Local Memory and the L1 Cache and is where the Load/Store instruction to access global memory originates. The compute unit can accommodate multiple work-groups (based on the temporary register and local memory usage) simultaneously.

5.2.4 Memory hierarchy

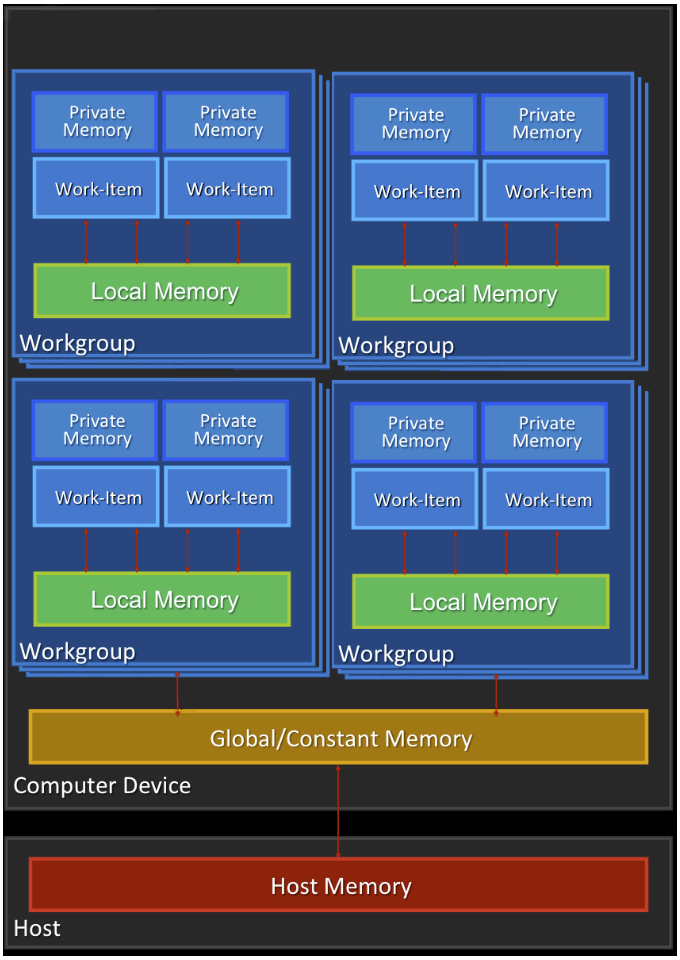


Figure 7. OpenCL memory hierarchy

5.2.5 CL Extension support

5.2.5.1 CL_DEVICE_EXTENSION support

The following table provides a list of CL_DEVICE_EXTENSIONs referenced in the OpenCL 1.2 specification (pp. 46-47). The support level for these device specific extensions is also indicated.

List from OpenCL 1.2 Specification https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf (version 1.2, document revision 19, revision date 11/14/12)

Table 18. Support level for these device specific extensions (1)

CL_DEVICE_EXTENSIONS **OpenCL C 1.2 Extensions which must be returned (p. 47)**	**SW 6.2.x/6.4.x**
cl_khr_byte_addressable_store	YES
cl_khr_global_int32_base_atomics	CORE
cl_khr_global_int32_extended_atomics	CORE
cl_khr_local_int32_base_atomics	CORE
cl_khr_local_int32_extended_atomics	CORE

Table 19. Support level for these device specific extensions (2)

CL_DEVICE_EXTENSIONS **Device specific support for Khronos approved extension names (p.46)** ***A number after the extension name indicates the extension is also listed in the numbered extensions on the Khronos website.***	**SW 6.2.x/6.4.x**
cl_khr_3d_image_writes	
cl_khr_context_abort	
cl_khr_d3d10_sharing (#6)	
cl_khr_d3d11_sharing	
cl_khr_depth_images	
cl_khr_dx9_media_sharing	
cl_khr_fp16	
cl_khr_gl_depth_images	
cl_khr_gl_event	
cl_khr_gl_msaa_sharing	
cl_khr_gl_sharing (#1)	YES
cl_khr_image2d_from_buffer	
cl_khr_initialize_memory	
cl_khr_int64_base_atomics	
cl_khr_int64_extended_atomics	
cl_khr_spir	

5.2.5.2 Vivante OpenCL extension support

The following table provides a list of all current OpenCL Extensions and indicates their support level in Vivante software.

Table 20. CL extensions supported by Vivante with 6.2.x SW

OpenCL Extension Number, Name and hyperlink	SW 6.2.x
cl_khr_gl_sharing	YES
cl_khr_icd	YES
VIV_bitfield_extension	**YES** *(from 6.2.2, revised in 6.2.3)*
VIV_cmplx_extension	**YES** *(from 6.2.3)*
VIV_uncached_host_mem	**YES** *(from 6.2.2)*
VIV_vx_extension	**YES**, for VX/VIP hw *(from 6.2.2)*

5.3 Optimization for OpenCL embedded profile

OpenCL EP (Embedded Profile) is basically a scaled down version of OpenCL FP(Full Profile) and thus may require extra optimization. The guidelines below help with the optimization of Vivante OpenCL Embedded Profile GPGPU cores.

When optimizing code on Vivante hardware, it is important to remember a few key points to get the best performance from the hardware:

• Take advantage of algorithm and data parallelism

• Choose the correct execution configuration (more details below)

• Overlap memory transfer from different levels of the OpenCL memory hierarchy with simultaneous thread execution

• Maximize memory bandwidth and minimize data transfers (large transfers are more beneficial than many smaller transfers because of the impact of latency)

• Maximize instruction throughput and minimize instruction count

5.3.1 Using preferred multiple of work-group size

The work-group size should be a multiple of the thread group size. Otherwise, some threads remain idle and the application does not fully utilize all the compute resources. For example, if the work-group size is 8 and the Vivante core supports 16, only half the compute resources are used. For example, in some early Vivante GPGPU revisions, the work-group size limit is 192 and the thread group size is 16. See the Overview section on OpenCL Compatible IP for IP-specific capabilities.

5.3.2 Using multiple work-groups of reduced size

Multiple work groups need to be set to reduce synchronization penalties. To prevent stalls at barriers, it is recommended to have at least four (4) work-groups to keep the cores busy or as long as the number of work-groups is greater than or equal to two (2). One work-group is very inefficient; four or more is preferred and helps avoid latency.

5.3.3 Packing work-item data

It is important to pack data to extract the optimal performance from the SIMD ALU hardware and align the data into a format supported by the hardware. Efficient use of the Vivante GPGPU core requires that the kernel contains enough parallelism to fill all four vector units. Work-items in the same thread group have the same program counter and execute the same instruction for each cycle. Whenever possible, pack together work-items that follow the same direction (e.g., on branches) since the granularity is very close and there may be less divergence and higher performance. If each work-item handles less than or equal to 8 bytes, it is better to combine two or more work-items into one to improve utilization of the SIMD ALU.

5.3.4 Improving locality

If the input data is an array-of-structs, and each work-item needs to access only a small part of the struct across many array elements at different stages, it may be better to convert and use a struct-of-arrays or several different arrays as input to improve data locality and avoid cache thrashing.

If each work-item needs to process a row of data without sharing any data with other work-items, it is better to check if the algorithm can be converted to make each work-item process a column of data so that data accessed by adjacent work-items can share the same cache lines.

5.3.5 Minimizing use of 1 KB local memory

The OpenCL Embedded Profile specification defines the minimum requirement for local memory to be 1KB to pass conformance testing. Based on algorithm analysis and profiling different image and computer vision algorithms, we found that a 1KB local memory size was too small to benefit those algorithms. In most instances, those algorithms actually slowed down when using 1KB local memory. To increase performance, we recommend not using local memory since it is more efficient to transfer larger chunks of data from system memory to keep the OpenCL pipeline full.

Note: If local memory type is CL_GLOBAL, the local memory is emulated using global memory, and the performance is the same as global memory. There is extra overhead on data copy from global to local, which slows down the performance.

5.3.6 Using 16 byte memory Read/Write size

When accessing memory, it is important to minimize the read/write count and to ensure L1 cache utilization is high to reduce outstanding read/write requests. Since the internal GPGPU read-write-request queue has a limit, if the queue and L1 cache are filled, then the GPGPU remains idle.

5.3.7 Using _RTZ rounding mode

Wherever possible, use _RTZ (round to zero) since it is natively supported in hardware with one instruction. Support for _RTE (round to nearest even) is optional in OpenCL EP and is only supported in Vivante GPGPU EP hardware from 2013. This function is handled in software for EP cores if necessary.

5.3.8 Using float4 for better performance on i.MX 8M Quad and i.MX 8QuadXPlus

Since both the i.MX 8M Quad and i.MX 8QuadXPlus boards have new RTL 6214, the CL kernel compiler generates GPU instructions using more registers on RTL6214. Float4 is recommended for real applications for better performance.

5.3.9 Using native functions

5.3.9.1 Using native_function() for increased performance

There are two types of runtime math libraries available to developers. Native_function() and regular function().

• Function(): slower, computationally expensive, higher instruction count, and greater accuracy

• Native_function(): faster, computationally inexpensive, lower instruction count (sometimes reduced to one instruction), and lower accuracy.

• If accuracy is not important but speed/performance is, use native math functions that map directly to the Vivante GPGPU hardware.

For image processing computations that do not require high accuracy, use native instructions to significantly lower the instruction count and speed up performance. Based on actual analysis and performance profiling with the Vivante GPGPU, we found that using native_function() instructions such as sin, cos, etc., reduces the instruction count from many instructions to one or two instructions. Use of native functions also sped performance by 3x-10x.

5.3.9.2 Using native_divide and native_reciprocal for faster floating point calculations

There are two use cases for floating point division which a user can select:

• Normal use of the division operator (/) in OpenCL has high precision and covers all corner use cases. This operator generates more instructions and runs slower.

• Native Divide: this use case uses the built-in function native_divide or native_reciprocal, which uses what the hardware supports. The Vivante OpenCL compiler generates one or two instructions for each native_divide or native_reciprocal instruction. If there are no corner use cases in applications, such as NaN, INF, or (2^127) / (2^127), it is better to use native_divide since it is faster.

5.3.9.3 Using compile option for native functions

Both the function() and native_function() methods are supported in the Vivante GPGPUs, so it is up to the developer to use whichever method makes sense for their application. If the OpenCL program uses the standard division operator and a developer wants to use native_divide or native_reciprocal without modifying their program, the Vivante OpenCL compiler has a simple option “-cl-fast-relaxed-math” that uses native built-in functions during compilation.

5.3.10 Using buffers instead of images

For the following image functions, it is better to use buffers instead of images.

• read_image{f/i/ui/h}

• write_image{f/i/ui/h}

Write_image* functions are implemented by software; it is better to use buffers to reduce the additional overhead involved in checking for size, format, etc. Since a few formats are not supported by Vivante GPGPU hardware, some built-in read_image() functions are implemented in software. The software implementation uses more instructions with many steps of “condition” checking. To improve performance, we recommend using buffers since it reduces instruction count.

5.4 OpenCL Debug messages

When writing OpenCL applications, it is important to check the code returned by the API. Since the return codes specified in the OpenCL specification may not be descriptive enough to isolate where the problem is located, the Vivante OpenCL driver provides an environment variable, VIV_DEBUG, to help debug problems. When VIV_DEBUG is set to -MSG_LEVEL:ERROR, the Vivante OpenCL driver prints onscreen error messages and returns the error code to the caller.

The following error code descriptions and suggested workarounds are provided.

5.4.1 OCL-007005: (clCreateKernel) cannot link kernel

One of the following “Not Enough” messages usually precedes this message. Issuer indicates the real reason for the problem which may be:

- Not Enough Register Memory (constant or temp)
- Not Enough Instruction Memory

5.4.2 Not enough register memory

Local variables, including arrays, are implemented using temp registers. If an array is larger than the number of available temp registers, a link-time failure occurs.

Workarounds:

1. If the array size is more than 64, use an array address to force the compiler to use private memory instead of temp registers.

2. If there are many variables, use variable addresses to force the compiler to use private memory to reduce register usage.

Note that there is performance degradation when using private memory instead of registers. It is better to change the algorithm to use a smaller array or less variables.

5.4.3 Not enough instruction memory

Workarounds:

1. Replace `sin/cos/tan/divide/powr/exp/exp2/exp10/log/log2/log10/sqrt/rsqrt/recip` with `native_sin/native_divide`, etc.

2. Convert unrolled-loops back to loops.

3. Use buffer instead of image for write, and for reads which are not linear-filtered.

4. If the program is too long, it should be split into two or more programs with intermediate data saved from one program to next.

5.4.4 GlobalWorkSize over hardware limit

WORKAROUND:

1. Split one clEnqueueNDRangeKernel into several instances. Change the kernel source to compute real global/local/group ID using offset as a parameter.

2. Convert one dimension to two dimensions, or two dimensions to three. For example, one dimension of 1M work-items can be converted to a GlobalWorkSize of 64K x16 work-items. The kernel function needs modification to reflect the change of dimension.

5.5 Zero copy

A buffer object can be created with clCreateBuffer(cl_context context, cl_mem_flags flags, size_t size, void* host_ptr, cl_int* error_code_ret). If memory flags contain CL_MEM_USE_HOST_PTR, GPU will map the memory pointed by host ptr for GPU to use to avoid copying data between CPU and GPU.

To make sure the results are correct, the size of buffer, the third parameter of clCreateBuffer(), needs to be aligned with 64-byte since Arm data cache operations are performed line by line, the unaligned bits will be cleared with cache line mask. A53, A57, A72 and A73 all have 64-byte cachaline size. If the size of the buffer doesn’t meet this, GPU will use copy method instead.

Besides, the host_ptr should be aligned with 64-bit to meet the ARM cacheline mechanism.

At last, need to call clEnqueueReadBuffer() to make sure the data has been read back to CPU.

5.6 Instruction cache availability for i.MX graphics

This section describes the instruction cache (iCache) available in the Vivante graphics IP included in the selected i.MX products.

There is hardware support for iCache available for i.MX 6QuadPlus and all later IP including that used in i.MX 8 products. There is no SH (Shader) instruction limit for these newer chips beyond the ISA limitation of 2*20.

Only the older chips have a SH instruction limit.

Table 21. i.MX products with graphics IP with iCache

i.MX Product	GPU IP & rev	Instruction Limit	Description
i.MX 8 Series and later	various *(from rev 5450)*	none	HW supports iCache
i.MX 6QuadPlus	GC2000 Plus rev FFFF5450	none	HW supports iCache
S32V234	GC3000 rev 5451	none	HW supports iCache

The SH limitation for i.MX products is listed in the following table.

Table 22. i.MX products with instruction limited graphics IP

i.MX Product	GPU IP & rev	Instruction Limit	Description
i.MX 6SoloX	GC400 rev 4645	256 for VS, 256 for PS	Separate Instruction buffers for Vertex Shader and for Pixel Shader
i.MX 7ULP	GCNanoUltra rev 4653a	256 for VS, 256 for PS	Separate Instruction buffers for Vertex Shader and for Pixel Shader
i.MX 6DualLite	GC880 rev 5106	512	Instruction buffer shared by Vertex and Pixel Shaders
i.MX 6Quad	GC2000 rev 5108	512	Instruction buffer shared by Vertex and Pixel Shaders

6 OpenVX Introduction

6.1 Overview

OpenVX is a low-level programming framework domain to enable software developers to efficiently access computer vision hardware acceleration with both functional and performance portability. OpenVX has been designed to support modern hardware architectures, such as mobile and embedded SoCs as well as desktop systems. Many of these systems are parallel and heterogeneous: containing multiple processor types including multi-core CPUs, DSP subsystems, GPUs, dedicated vision computing fabrics as well as hardwired functionality. Additionally, vision system memory hierarchies can often be complex, distributed, and not fully coherent. OpenVX is designed to maximize functional and performance portability across these diverse hardware platforms, providing a computer vision framework that efficiently addresses current and future hardware architectures with minimal impact on applications.

OpenVX defines a C Application Programming Interface (API) for building, verifying, and coordinating graph execution, as well as for accessing memory objects. The graph abstraction enables OpenVX implementers to optimize the execution of the graph for the underlying acceleration architecture.

OpenVX also defines the vxu utility library, which exposes each OpenVX predefined function as a directly callable C function, without the need for first creating a graph. Applications built using the vxu library do not benefit from the optimizations enabled by graphs; however, the vxu library can be useful as the simplest way to use OpenVX and as first step in porting existing vision applications.

For more details of programming with OpenVX, see the following specification from Khronos Group,

OpenVX specification (https://www.khronos.org/registry/vx).

6.2 OpenVX extension implementation

VeriSilicon's VX Extensions for Vision Imaging provide additional functionality for Vision Image processing beyond the functions provided through the Khronos Group OpenVX API. These enhancements take advantage of the enhanced Vision capabilities available in VeriSilicon's Vision-capable hardware. VeriSilicon software provides a set of extensions which interface with OpenCL 1.2 and support higher level C language programming of VeriSilicon's custom EVIS (Enhanced Vision Instruction Set).

The VeriSilicon VX extension and enhancements includes three major components:

- An API level interface to the EVIS (Enhanced Vision Instruction Set)
- Extended C language features for Vision Processing
- Supported for a subset of Vision-compatible OpenCL built-in functions

6.2.1 Hardware requirements

Vision Imaging hardware capabilities are required to support full OpenVX. The following configurations are supported:

- GC7000XSVX (i.MX 8QuadMax)
- VIP8000NanoSI (i.MX 8M Plus)

6.2.2 EVIS instruction interface

Vivante's Vision Imaging capable IP have an Enhanced Vision Instruction Set (EVIS), which enhances the ability of the GPU or VIP (Vision Image Processor) to process complex vision operations. A single EVIS instruction can do a task which may require tens or even hundreds of normal ISA instructions to finish.

The following table shows the instructions supported as Intrinsic calls.

6.2.3 Extended language features

Vivante's OpenVX C programming Language corresponds closely to the OpenCL C programming language.

- Vivante's C language extensions for OpenVX C share many language facilities with OpenCL C 1.2. However, it can be considered a subset of OpenCL C 1.2, as it does not include OCL features which are useless for OpenVX and other Vision Imaging applications.
- Vivante's OpenVX C includes specific language facilities like Vision built-ins and data types specific for OpenVX.

Table 23. OPCODE EVIS instructions supported as intrinsic calls

EVIS OP_CODE	Description	Supported by Vivante VX
ABS_DIFF	Absolute difference between two values	Y
IADD	Adds two or three integer values	Y
IACC_SQ	Squares a value and adds it to an accumulator	Y
LERP	Linear interpolation between two values	Y
FILTER	Performs a filter on a 3x3 block	Y
MAG_PHASE	Computes magnitude and phase of 2 packed data values	Y
MUL_SHIFT	Multiples two 8-or 16-bit integers and shifts	Y
DP16X1	1 Dot Product from 2 16 component values	Y
DP8X2	2 Dot Products from 2 8 component values	Y
DP4X4	4 Dot Products from 2 4 component values	Y
DP2X8	8 Dot Products from 2 2 component values	Y
CLAMP	Clamps up to 16 values to a max or min value	Y

Table 23. OPCODE EVIS instructions supported as intrinsic calls...*continued*

EVIS OP_CODE	Description	Supported by Vivante VX
BI_LINEAR	Computes a bi0linear interpolation of 4 pixel values	Y
SELECT_ADD	Adds a pixel value or increments a counter inside bins	Y
ATOMIC_ADD	Adds a valid atomically to an address	Y
BIT_EXTRACT	Extracts up to 8 bitfields from a packed stream	Y
BIT_REPLACE	Replaces up to 8 bitfields from a packed stream	Y
DP32X1	1 Dot Product from 2 32 component values	Y
DP16X2	2 Dot Products from 2 16 component values	Y
DP8X4	4 Dot Products from 2 8 component values	Y
DP4X8	8 Dot Products from 2 4 component values	Y
DP2X16	16 Dot Products from 2 2 component values	Y

6.2.4 Packed types

Vivante's OpenCL compiler implements OpenCL C signed and unsigned char and short types in an unpacked format, such that a normal char4 occupies 128 bits (4 32-bit registers). This is undesirable for Vision applications, where packed data is the "natural" layout for almost all operations. To fully utilize the computing power of EVIS instructions, Vivante VX includes additional packed types, which can be identified by their **vxc_** prefix.

```
/* packed char2/4/8/16 */
typedef _viv_char2_packed vxc_char2;
typedef _viv_char4_packed vxc_char4;
typedef _viv_char8_packed vxc_char8;
typedef _viv_char16_packed vxc_char16;
/* packed uchar2/4/8/16 */
typedef _viv_uchar2_packed vxc_uchar2;
typedef _viv_uchar4_packed vxc_uchar4;
typedef _viv_uchar8_packed vxc_uchar8;
typedef _viv_uchar16_packed vxc_uchar16;
/* packed short2/4/8 */
typedef _viv_short2_packed vxc_short2;
typedef _viv_short4_packed vxc_short4;
typedef _viv_short8_packed vxc_short8;
/* packed ushort2/4/8 */
typedef _viv_ushort2_packed vxc_ushort2;
typedef _viv_ushort4_packed vxc_ushort4;
typedef _viv_ushort8_packed vxc_ushort8;
```

6.2.5 Initializing constants on load

Constant data in OpenCL requires compile-time initialization. There is also a need to initialize the data when the kernel is loaded/run, so that the application can control the behavior of a program by changing its constants at load-time. The VeriSilicon VX extended keyword **_viv_uniform** can be used to define load-time initialization constant data,

```
_viv_uniform vxc_512bits u512;
```

An application using VeriSilicon VX needs to set the proper values for _viv_uniform before the kernel program is run.

6.2.6 Inline assembly

A packed type cannot be used as an unpacked type in expressions or built-in functions. The programmer needs to convert packed type data to unpacked type data in order to perform these operations. The conversion negatively impacts performance in terms of both instruction count and register usage, so it is desirable to perform operations directly on packed data whenever possible. The Vivante Vision compiler accepts inline assembly for a wide range of operations to speed up packed data calculations.

For example, to add two packed char16 data, the programmer can use following inline assembly:

```
vxc_uchar16 a, b, c;
vxc_short8 b;
_viv_uniform vxc_512bits u512;
...
_viv_asm(ADD, c, a, b); /* c = a + b; */
where the syntax of inline assembly is:
_viv_asm(
OP_CODE,
dest,
source0,
source1
);
```

The following table lists the standard shader instructions that operate on packed data and are supported through inline assembly, keyword **_viv_asm**.

Table 24. OPCODES IR instructions supported by inline assembly

IR OP_CODE Instruction	Description	Supported by Vivante VX
ABS	Absolute value	Y
ADD	Add	Y
ADD_SAT	Integer add with saturation	Y
AND_BITWISE	Bitwise AND	Y
BIT_REVERSAL	Integer bit-wise reversal	ES31
BITEXTRACT	Extract Bits from src to dest	ES31
BITINSERT	Bit replacement	ES31
BITSEL	Bitwise Select	Y
BYTE_REVERSAL	Integer byte-wise reversal	ES31
CLAMP0MAX	clamp0max dest, value, max	Y
CMP	Compare each component	Y
CONV	Convert	Y
DIV	Divide	Y
FINDLSB	Find least significant bit	ES31
FINDMSB	Find most significant bit	ES31

Table 24. OPCODES IR instructions supported by inline assembly*...continued*

IR OP_CODE Instruction	Description	Supported by Vivante VX
LEADZERO	Detect Leading Zero	Y
LSHIFT	Left Shifter	Y
MADSAT	Integer multiple and add with saturation	Y
MOD	Modulus	Y
MOV	Move	Y
MUL	Multiply	Y
MULHI	Integer only	Y
MULSAT	Integer multiply with saturation	Y
NEG	neg(a) is similar to (0 - (a))	Y
NOT_BITWISE	Bitwise NOT	Y
OR_BITWISE	Bitwise OR	Y
POPCOUNT	Population Count	ES31/OCL1.2
ROTATE	Rotate	Y
RSHIFT	Right Shifter	Y
SUB	Substract	Y
SUBSAT	Integer subtraction with saturation	Y
XOR_BITWISE	Bitwise XOR	Y

Note: **ES31 = Supported by VivanteVX, but may not be needed for Vision processing*

6.3 OpenCL functions compatible with Vivante vision

Vivante’s VX extensions for Vision Image processing support most of the OpenCL 1.2 built-in functions for normal OCL data types. Packed types are not supported in these built-in functions.

For image read/write functions, only sample-less 1D/1D array/2D image read/write functions are supported.

6.3.1 Read_Imagef,i,ui

```
/* OCL image builtins can be used in VX kernel */
float4 read_imagef (image2d_t image, int2 coord);
int4 read_imagei (image2d_t image, int2 coord);
uint4 read_imageui (image2d_t image, int2 coord);
float4 read_imagef (image1d_t image, int coord);
int4 read_imagei (image1d_t image, int coord);
uint4 read_imageui (image1d_t image, int coord);
float4 read_imagef (image1d_array_t image, int2 coord);
int4 read_imagei (image1d_array_t image, int2 coord);
uint4 read_imageui (image1d_array_t image, int2 coord);
```

6.3.2 Write_Imagef,i,ui

```
void write_imagef (image2d_t image, int2 coord, float4 color);
void write_imagei (image2d_t image, int2 coord, int4 color);
void write_imageui (image2d_t image, int2 coord, uint4 color);
void write_imagef (image1d_t image, int coord, float4 color);
void write_imagei (image1d_t image, int coord, int4 color);
void write_imageui (image1d_t image, int coord, uint4 color);
void write_imagef (image1d_array_t image, int2 coord, float4
 color);
void write_imagei (image1d_array_t image, int2 coord, int4
 color);
void write_imageui (image1d_array_t image, int2 coord, uint4
 color)
```

6.3.3 Query Image Dimensions

```
int2 get_image_dim (image2d_t image);
size_t get_image_array_size(image1d_array_t image);
/* Built-in Image Query Functions */
int get_image_width (image1d_t image);
int get_image_width (image2d_t image);
int get_image_width (image1d_array_t image);
int get_image_height (image2d_t image);
```

6.3.4 Channel Data Types Supported

```
/* Return the channel data type. Valid values are:
* CLK_SNORM_INT8
* CLK_SNORM_INT16
* CLK_UNORM_INT8
* CLK_UNORM_INT16
* CLK_UNORM_SHORT_565
* CLK_UNORM_SHORT_555
* CLK_UNORM_SHORT_101010
* CLK_SIGNED_INT8
* CLK_SIGNED_INT16
* CLK_SIGNED_INT32
* CLK_UNSIGNED_INT8
* CLK_UNSIGNED_INT16
* CLK_UNSIGNED_INT32
* CLK_HALF_FLOAT
* CLK_FLOAT
*/
int get_image_channel_data_type (image1d_t image);
int get_image_channel_data_type (image2d_t image);
int get_image_channel_data_type (image1d_array_t image);
```

6.3.5 Image Channel Orders Supported

```
/* Return the image channel order. Valid values are:
* CLK_A
* CLK_R
* CLK_Rx
* CLK_RG
```

```
* CLK_RGx
* CLK_RA
* CLK_RGB
* CLK_RGBx
* CLK_RGBA
* CLK_ARGB
* CLK_BGRA
* CLK_INTENSITY
* CLK_LUMINANCE
*/
int get_image_channel_order (image1d_t image);
int get_image_channel_order (image2d_t image);
int get_image_channel_order (image1d_array_t image);
```

7 Vulkan

7.1 Overview

Vulkan is a new generation graphics and compute API that provides high-efficiency, cross-platform access to modern GPUs used in a wide variety of devices from PCs and consoles to mobile phones and embedded platforms.

Vulkan defines as an API (Application Programming Interface) for graphics and compute hardware. The API consists of many commands that allow a programmer to specify shader programs, compute kernels, objects, and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

To the programmer, Vulkan is a set of commands that allow the specification of shader programs or shaders, kernels, data used by kernels or shaders, and state controlling aspects of Vulkan outside the scope of shaders. Typically, the data represents geometry in two or three dimensions and texture images, while the shaders and kernels control the processing of the data, rasterization of the geometry, and the lighting and shading of fragments generated by rasterization, resulting in the rendering of geometry into the framebuffer.

A typical Vulkan program begins with platform-specific calls to open a window or otherwise prepare a display device onto which the program will draw. Then, calls are made to open queues to which command buffers are submitted. The command buffers contain lists of commands which will be executed by the underlying hardware. The application can also allocate device memory, associate resources with memory and refer to these resources from within command buffers. Drawing commands cause application-defined shader programs to be invoked, which can then consume the data in the resources and use them to produce graphical images. To display the resulting images, further platform-specific commands are made to transfer the resulting image to a display device or window.

For more details of programming with Vulkan, refer to the following specification from Khronos Group.

https://www.khronos.org/registry/vulkan/

7.2 Vivante Extension Support for Vulkan

The following table includes a list of all current Vulkan extensions and indicates their support level in Vivante software.

All information provided in this document is subject to legal disclaimers.

(list from *https://www.khronos.org/registry/vulkan/* as of 6/1/2020)

Note: *This list does not include unsupported vendor specific extensions.*

Table 25. Vulkan extension

Supported Vulkan 1.1 Extension Names	SW 6.4.x for Vulkan 1.1
VK_KHR_16bit_storage	YES
VK_KHR_bind_memory2	YES
VK_KHR_descriptor_update_template	YES
VK_KHR_device_group	YES
VK_KHR_external_memory	YES
VK_KHR_get_memory_requirements2	YES
VK_KHR_maintenance1	YES
VK_KHR_maintenance2	YES
VK_KHR_maintenance3	YES
VK_KHR_variable_pointers	YES
VK_KHR_dedicated_allocation	YES
VK_EXT_queue_family_foreign	YES
VK_KHR_external_semaphore_fd	YES
VK_KHR_external_fence_fd	YES
VK_KHR_external_semaphore_win32	YES
VK_KHR_external_fence_win32	YES
VK_ANDROID_native_buffer	YES
VK_ANDROID_external_memory_android_hardware_buffer	YES
VK_KHR_swapchain	YES
VK_EXT_debug_report	YES
VK_KHR_device_group_creation	YES
VK_KHR_external_memory_capabilities	YES
VK_KHR_external_semaphore_capabilities	YES
VK_KHR_external_fence_capabilities	YES
VK_KHR_get_physical_device_properties2	YES
VK_KHR_win32_surface	YES
VK_KHR_android_surface	YES
VK_KHR_wayland_surface	YES
VK_KHR_surface	YES
VK_KHR_display	YES

7.3 Vulkan Validation Layers

Vulkan is an explicit API, enabling direct control over how GPUs actually work. By design, minimal error checking is done inside a Vulkan driver. Applications have full control and responsibility for correct operation. Any errors in how Vulkan is used can result in a crash. Vulkan validation layers that can be enabled to assist development by enabling developers to verify their applications correct use of the Vulkan API.

7.4 Window System Integration

Vulkan relies on a new mechanism to interact with the native Windowing System and present the rendered results to the user. This mechanism is called the Window System Integration and is provided via extensions outside of the core API.

In the i.MX BSPs where Vulkan is enabled, the default window manager is Weston, a Wayland compositor reference implementation.

When compiling a Vulkan application for Wayland make sure to define the VK_USE_PLATFORM_WAYLAND_KHR symbol, so all the proper includes and code paths are enabled.

GLFW and SDL can manage the surface creation and proper extension initializations, but when an application is newly developed without using any frameworks, require to enable the following instance extensions:

`VK_KHR_SURFACE_EXTENSION_NAME`

`VK_KHR_WAYLAND_SURFACE_EXTENSION_NAME`

Once there is a display connection to the Wayland server and a surface created, then start to use the wl_display and wl_surface pointers to populate the info structure required by vkCreateWaylandSurfaceKHR.

A word of advice, when querying the Physical Device Surface capabilities with vkGet PhysicalDeviceSurfaceCapabilitiesKHR before having created the Swapchain, the current extent width and height will return a value of 0xFFFFFFFF, make sure to add checks for this in the code, when this happens, set the swapchain extent to the actual size of the surface want to render to, or a fallback extent size.

8 Multiple GPUs and Virtualization

8.1 Overview

Vivante multi-GPU implementations provide a variety of capabilities which can be managed through hardware and software controls. This chapter intends to summarize the software controls used for Vivante multi-GPU IP implementations.

Multi-GPU feature can be enabled with dual GC7000XSVX on i.MX 8QuadMax and the derived devices.

8.2 Multi-GPU configurations

Vivante Multi-GPU IP may be configured into one of the following behavior model through software:

- Combined Mode where two (or more) GPU cores in the multi-GPU design behave in concert. Driver presents multi-GPU to SW application as a single logical GPU. The multiple GPUs work in the same virtual address space and share the same MMU page table. The multiple GPUs fetch and execute a shared Command Buffer.

- Independent Mode where each GPU in the multi-GPU design performs independently. The multiple GPUs work in different virtual address spaces but share the same MMU page table. Each GPU core fetches and executes its own Command Buffer. This enables different SW applications to run simultaneously on different GPU cores.
- OpenCL API allows application to handle the multi-GPU Independent Mode directly, as each GPU core in a multi-GPU design represents an independent OpenCL Compute Device.

8.3 GPU affinity configuration

In the multi-GPU Independent Mode, application can specify to run on a specific GPU among the multiple GPUs through an environment variable VIV_MGPU_AFFINITY. Once an application's GPU affinity is specified, the application will only run on the specified GPU and will not migrate to other GPUs even if those GPUs are idle.

VIV_MGPU_AFFINITY is the environment variable to control the application GPU affinity on multi-GPU platform. The client drivers will assume they are using a standalone GPU through a gcoHARDWARE object no matter how this variable is set. The possible values for the environment variable VIV_MGPU_AFFINITY include:

- Not defined or
- Defined as "0" gcoHARDWARE objects work in gcvMULTI_GPU_COMBINED mode (default)
  - "1:0" gcoHARDWARE objects work in gcvMULTI_GPU_INDEPENDENT mode and GPU0 is used
  - "1:1" gcoHARDWARE objects work in gcvMULTI_GPU_INDEPENDENT mode and GPU1 is used

On a single GPU device, setting VIV_MGPU_AFFINITY to 0 or 1 does not make any difference as all application processes/threads are bound to GPU0. But the application will fail the GPU context initialization if VIV_MGPU_AFFINITY is set to "1:1" (driver reports error).

8.4 OpenCL on multi-GPU device

OpenCL driver works in bridged mode as single logical compute device. In this configuration, multiple GPUs in the device operate as individual OpenCL Compute Devices. The OpenCL application is responsible to assign and dispatch the compute tasks to each GPU (Compute Device).

The following OpenCL APIs return the list of compute devices available on a platform, and the device information.

```
cl_int clGetDeviceIDs (cl_platform_id platform, cl_device_type
 device_type, cl_uint num_entries,
cl_device_id *devices, cl_uint *num_devices)
cl_int clGetDeviceInfo (cl_device_id device, cl_device_info
 param_name, size_t param_value_size,
void *param_value, size_t *param_value_size_ret)
```

8.5 GPU virtualization configuration

Multi-GPU also can be used on different OS systems as independent mode separately, this can be configured by overriding the irq availability n DTS entry for different OS implementation, in `arch/arm64/boot/dts/freescale/fsl-imx8qmxxx.dts`.

Guest OS 1 (GPU0 only)

```
&gpu_3d1 {
        status = "disable";
};
```

Guest OS 2 (GPU1 only)

```
&gpu_3d0 {
        status = "disable";
};
```

9 GBM - Generic Buffer Management

The GBM (Graphic Buffer Management) API is a thin layer over DRM KMS (Linux Direct Rendering Manager - Kernel ModeSetting API) that provides a mechanism for allocating buffers for graphics rendering. GBM is intended to be used as a native platform for EGL on DRM. The handle it creates can be used to initialize EGL and to create render target buffers. This can be resumed as a modern OpenGL ES FBDEV, because it permits full usage of the DRM KMS API with OpenGL ES acceleration.

Starting from i.MX 8, the DRM is supported and recommended to use GBM. GBM provides options of allocating modifier-abiding surfaces too, for Wayland compositors and the X11 server to render to.

9.1 Introduction to DRM Format Modifiers

A DRM format modifier is a 64-bit, vendor-prefixed, semi-opaque unsigned integer. Most modifiers represent a concrete, vendor-specific tiling format for images. Some exceptions are DRM_FORMAT_MOD_LINEAR (which is not vendor-specific); DRM_FORMAT_MOD_NONE (which is an alias of DRM_FORMAT_MOD_LINEAR due to historical accident); and DRM_FORMAT_MOD_INVALID (which does not represent a tiling format). The modifier’s vendor prefix consists of the 8 most significant bits. The canonical list of modifiers and vendor prefixes is found in drm_fourcc.h in the Linux kernel source.

One goal of modifiers in the Linux ecosystem is to enumerate for each vendor a reasonably sized set of tiling formats that are appropriate for images shared across processes, APIs, and/or devices, where each participating component may possibly be from different vendors. A non-goal is to enumerate all tiling formats supported by all vendors. Some tiling formats used internally by vendors are inappropriate for sharing; no modifiers should be assigned to such tiling formats.

Modifier values typically do not describe memory layouts. More precisely, a modifier's lower 56 bits usually have no structure. Instead, modifiers name memory layouts; they name a small set of vendor-preferred layouts for image sharing. As a consequence, in each vendor namespace the modifier values are often sequentially allocated starting at 1.

Each modifier is usually supported by a single vendor and its name matches the pattern {VENDOR}_FORMAT_MOD_* or DRM_FORMAT_MOD_{VENDOR}_*. Examples are DRM_FORMAT_MOD_VIVANTE_TILED and DRM_FORMAT_MOD_BROADCOM_VC4_T_TILED. An exception is DRM_FORMAT_MOD_LINEAR, which is supported by most vendors.

Many APIs in Linux use modifiers to negotiate and specify the memory layout of shared images. For example, a Wayland compositor and Wayland client may, by relaying modifiers over the Wayland protocol zwp_linux_dmabuf_v1, negotiate a vendor-specific tiling format for a shared wl_buffer. The client may allocate the underlying memory for the wl_buffer with GBM, providing the chosen modifier to gbm_bo_create_with_modifiers. The client may then import the wl_buffer into Vulkan for producing image content, providing the resource's dma_buf to VkImportMemoryFdInfoKHR and its modifier to VkImageDrmFormatModifierExplicitCreateInfoEXT. The compositor may then import the wl_buffer into OpenGL for sampling, providing the resource's dma_buf and modifier to eglCreateImage. The compositor may also bypass OpenGL and submit the wl_buffer directly to the kernel's display API, providing the dma_buf and modifier through drm_mode_fb_cmd2.

10 Wayland and Weston

10.1 Overview

Wayland is a protocol for a compositor to talk to its clients as well as a C library implementation of that protocol. Wayland is intended as a simpler replacement for X, easier to develop and maintain. The compositor can be a standalone display server running on Linux kernel modesetting and evdev input devices, an X application, or a Wayland client itself. The clients can be traditional applications, X servers (rootless or full screen) or other display servers.

10.2 Wayland EGL

Wayland-EGL is the client side implementation of the Wayland that binds the EGL stack and buffer sharing mechanism to the generic Wayland API. Frontend of the wayland-egl is now part of the wayland and i.MX graphics driver supports the implementation of buffer sharing mechanism.

10.3 Weston Compositor

Weston is reference implementation of a Wayland compositor. The Weston compositor is minimal and lightweight and is suitable for many embedded and mobile use cases. Weston support multiple renderers and backends which need to be chosen appropriately based on the processor configurations. This is usually preset in the i.MX image.

10.3.1 Weston Backends

Weston have implementation to support different display APIs, which is called backend. i.MX 8 support KMS/DRM hence uses DRM backend while the i.MX 6/7 uses FBDEV backend. i.MX graphics continues to support graphics acceleration with FBDEV backends.

10.3.2 Weston Renderer

10.3.2.1 GL Renderer

GL (GLES) renderer implementation is the default with Weston implementation. GL renderer takes the buffer passed from clone and maps as a texture. After the initial setup, the client only needs to tell the compositor which buffer to use and when and where it has rendered new content into it.

10.3.2.2 G2D Renderer

G2D is the 2D API refer to Chapter 2 for full details of G2D APIs. G2D renderer provides mechanism to accelerate Weston with 2D GPU. The 2D Graphics Engine reduces the burden on 3D GPU and saves power as well as integrates nicely with the video capabilities of the SoC. G2D compositor can increase system bandwidth utilization, so the performance will be better than GL compositor in the complex usecase environment.

To enable the G2D compositor, open the file: `/etc/xdg/weston/weston.ini` in the Linux image.

```
use-g2d=1
```

10.3.3 Weston Shells

Weston supports multiple shells, each of these shells have its own public protocol interface for clients. This means that a client must be specifically written for a shell protocol. Otherwise, it will not work. Below are the currently supported shell.

10.3.3.1 Desktop shell

Desktop shell is like a typical X desktop environment, concentrating on traditional keyboard and mouse user interfaces and the familiar desktop-like window management. Desktop shell consists of the shell plugin desktop-shell.so and the special client weston-desktop-shell which provides the wallpaper, panel, and screen locking dialog.

10.3.3.2 Fullscreen shell

Fullscreen shell is intended for a client that needs to take over whole outputs, often all outputs. This is primarily intended for running another compositor on Weston. The other compositor does not need to handle any platform-specifics like DRM/KMS or evdev/libinput. The shell consists only of the shell plugin fullscreen-shell.so.

10.3.3.3 IVI-shell

In-vehicle infotainment shell is a special purpose shell that exposes a GENIVI Layer Manager compatible API to controller modules, and a very simple shell protocol towards clients. IVI-shell starts with loading ivi-shell.so, and then a controller module which may launch helper clients. This shell provides option of setting windowing position, which need to be programmed from the client application.

11 X Windowing Acceleration

X11 is accelerated on i.MX 8 through Xwayland. Support on i.MX 6 deprecated.

12 Advanced GPU Configuration

12.1 GPU Scaling Governor

i.MX 8QuadMax GPU design supports different running modes: overdrive, nominal, and underdrive. Nominal is the default, the overdrive is supposed to be performance/ benchmark mode, and underdrive mode is expected as energy saving mode.

Switch among the 3 modes using command line without needing to recompile the GPU driver.

```
$ echo "overdrive"   >  /sys/bus/platform/drivers/galcore/
gpu_govern
$ echo "nominal"     >  /sys/bus/platform/drivers/galcore/
gpu_govern
$ echo "underdrive"  >  /sys/bus/platform/drivers/galcore/
gpu_govern
```

Try to check the mode is running currently, use the command line as follows:

```
$ cat /sys/bus/platform/drivers/galcore/gpu_govern
```

12.2 GPU Device Cooling

i.MX device support the thermal driver, which could signal the overheat event to GPU driver, once GPU driver receive the event, it can enable GPU DFS feature to reduce GPU frequency as N/64 of the original designated clock.

The default N factor is 1 in the original BSP release, the end-user can reconfigure it through the following command:

```
echo N >/sys/bus/platform/drivers/galcore/gpu3DMinClock
```

The user also can check the existing configuration as follows:

```
cat /sys/bus/platform/drivers/galcore/gpu3DMinClock
```

13 Vivante IDE

13.1 VivanteIDE overview

The VivanteIDE provides a single interface to a set of applications designed to be used by graphics, compute, vision and neural network application developers to rapidly develop and port applications either stand alone or as part of an IDE. Vivante IDE is built on the top of Eclipse, CDT

VivanteIDE capabilities include the following key features.

- **Project Management**
  The Project Manager supports individual compile options for each file. In addition, workspace options define project dependencies, removing the need for manual management of file builds.
- **Source code smart editing and analysis**

The VivanteIDE Editor provides timesaving editing features such as type ahead for structures, word completion and automatic code indentation for a readable, formatted code view.

- **Automatic code generation**
  Kernel development wizard can automatically generate the kernel code basing on simple inputs.
- **Performance and bandwidth profiling**
  The Profile tabbed window provides profiler information. Every time the profiler is suspected accumulated statistical information is updated. For OGL applications the VPD Analyzer is provided.
- **Post-mortem performance analysis**
  VPD Analyzer visualized the hardware data recorded at GPU application runtime.
- **Texture browse and conversion**
  Texture browser and converter support texture file preview and format conversion.
- **Command line tools for OGL, OCL and OVX** compile.
- **Command line tools for Vulkan** application development.
- **Command line tools for Texture** compression/decompression and tile/de-tiling.

13.1.1 VivanteIDE component overview

VivanteIDE provides both command line tools and GUI “Perspective” views for performing different activities. Some functionality is available through both GUI and command line, while tools such as vCompiler and vcCompiler are available only using command line syntax.

Table 26. VivanteIDE tool overview

Perspective/Tool	Key Functionality	GUI	Command Line
Debug	Debug projects	Yes	
Profile	Configure projects	Yes	
vCompiler	Offline OGL compiler	No	Yes, vCompiler
vcCompiler	Offline OCL compiler	No	Yes, vcCompiler
VPD Analyzer	Performance analysis	Yes	No
vTexture, vTextureTools	Texture manipulations and viewing; Compress, Decompress, Tile, De-Tile	Yes Texture Viewer Texture Browser	Yes vTextureTools
SPIR-V Disassembly	Debug Vulkan apps	Yes	No
Shader Assistant	Shader programming	Yes	No

13.2 VivanteIDE Requirements

13.2.1 Operating system compatibility

VivanteIDE is available for both Linux and Windows environments. VivanteIDE has been verified to work in Windows 7, Windows 10, Ubuntu 18.04, and Ubuntu 16.04. It might work in other Windows or Linux systems but has not been verified for alternate environments.

Table 27. Operating System Tool Compatibility Summary

Components	Linux	Windows
VivanteIDE	GUI and command	GUI and command
Tools		
vCompiler, vcCompiler	command	command
vProfiler	Built part of i.MX unified driver (target)	Built part of i.MX unified in driver(target)
VPD Analyzer	GUI	GUI
Shader Assistant	GUI	GUI
Texture Viewer	GUI	GUI
Texture Browser	GUI	GUI
vTextureTools	GUI and command	GUI and command

13.2.2 Hardware requirements

VivanteIDE can be used in either a simulation environment or on i.MX processors supporting OpenGL ES, OpenCL, OpenVX, and Neural Networks capabilities in the tools assume compatible hardware capability in the running environment, which must be correctly profiled in the tool for accurate results.

13.2.3 VivanteIDE license

i.MX supported VivanteIDE release package contains with preloaded license and restricted only to use with NXP processors. For more information, read NXP EULA.

13.3 VivanteIDE installation

13.3.1 VivanteIDE package

Each release of VivanteIDE will be compatible with its companion driver version. Forward and backward compatibility is not tested and use of VivanteIDE with any different driver version other than its companion version is NOT RECOMMENDED.

The package is delivered as a compressed file from nxp.com as `Verisilicon_Tool__IDE_<version>.tgz`.

Table 28. VivanteIDE package contents

Top level Directory and exe file	Description
`VivanteIDE-<version>-Linux-x86_64-**-Install`	Installation wizard for Linux 64-bit.
`VivanteIDE-<version>-Windows-**-Setup.exe`	Installation wizard for Windows 64-bit/32-bit
`README`	README with basic installation notes

After installation the following directories will be created in the installation directory

Table 29. VivanteIDE tools directory

Files and Directories	Description
`ide/`	Directory containing IDE executables and plugins
`examples/`	Directory containing examples (just for Windows)
`cmdtools/`	Directory containing Vivante command line tools: vCompiler, vcCompiler, vTextureTools
`doc/`	Directory containing documents
`license/`	Directory containing license tools and license files
`jre/`	Directory containing JRE binaries
`mingw32/`	Directory containing MinGW (just for Windows)
`uninstall.exe`	Uninstaller of VivanteIDE

13.3.2 Installation

Install the package to run both the GUI and command line tools. You must install the package even if you are only going to use the command line tools.

13.3.2.1 Linux GUI

Run `Vivante-<version>-Linux-x86_64-**-Install` to launch the installation wizard. Follow the installation steps guided by the installation wizard to finish the installation.

13.3.2.2 Windows GUI

Run `Vivante-<version>-Windows-**-Setup.exe` to launch the installation wizard. Follow the installation steps guided by the installation wizard to finish the installation.

13.3.2.3 Installation from command line

The VivanteIDE installer can also be launched from the command line. Options can be specified as follows:

```
installer [option1] [option2] [option3]
```

Example Usage for Windows:

```
installer /mode silent /prefix destination_location /license
 license_file_path
```

Example Usage for Linux:

```
installer --mode silent --prefix destination_location --license
 license_file_path
```

Table 30. Command line installer options

Option for Windows	Option for Linux	Description
/mode silent	--mode silent	Silent mode (without GUI, without prompting)
/license license_file_path	--license license_file_path	Specify a license file to be installed

Table 30. Command line installer options*...continued*

Option for Windows	Option for Linux	Description
/prefix destination_ location	--prefix destination_ location	Specify the folder where VivanteIDE will be installed

13.3.3 VivanteIDE launch

13.3.3.1 Linux launch of GUI tool

To launch the GUI tool,

- Double-click the desktop shortcut **VivanteIDE<version>**.
- Run `installation_dir/ide/vivanteide<version>` in a BASH.

13.3.3.2 Windows launch of GUI tool

To launch the GUI tool:

- Click **Start Menu->VeriSilicon->VivanteIDE <version>->VivanteIDE <version>**.
- Double-click the desktop shortcut **VivanteIDE <version>**.
- Run `installation_dir/ide/vivanteide<version>.bat`.

13.3.3.3 Command line tool launch

To launch the command line tools, use the following paths. For Linux OS, launch in a BASH.

Run `installation_dir/cmdtools/vCompiler, vcCompiler, vTextureTools`.

13.3.3.4 Basic launch path summary

Table 31. Basic launch instruction summary

Tool	Linux Basic Launch Instuction	Windows Basic Launch Instuction
VivanteIDE GUI	Run `installation_dir/ide/vivanteide<version>` in a BASH.	Run `installation_dir/ide/vivanteide<version>.bat`
vcCompiler	Run `installation_dir/cmdtools/bin/vcCompiler` in a BASH.	Run `installation_dir/cmdtools/bin/vcCompiler.exe`
vCompiler	Run `installation_dir/cmdtools/bin/vcompiler` in a BASH.	Run `installation_dir/cmdtools/bin/vCompiler.exe`
vTexture Tools	Run `installation_dir/cmdtools/bin/vtexturetools` in a BASH.	Run `installation_dir/cmdtools/bin/vTextureTools.exe`

13.4 VivanteIDE GUI

The desktop development environment for VivanteIDE is referred to as the Workbench. The Workbench contains panes that may change depending on the current activity. Some key panes are indicated in the figure below.

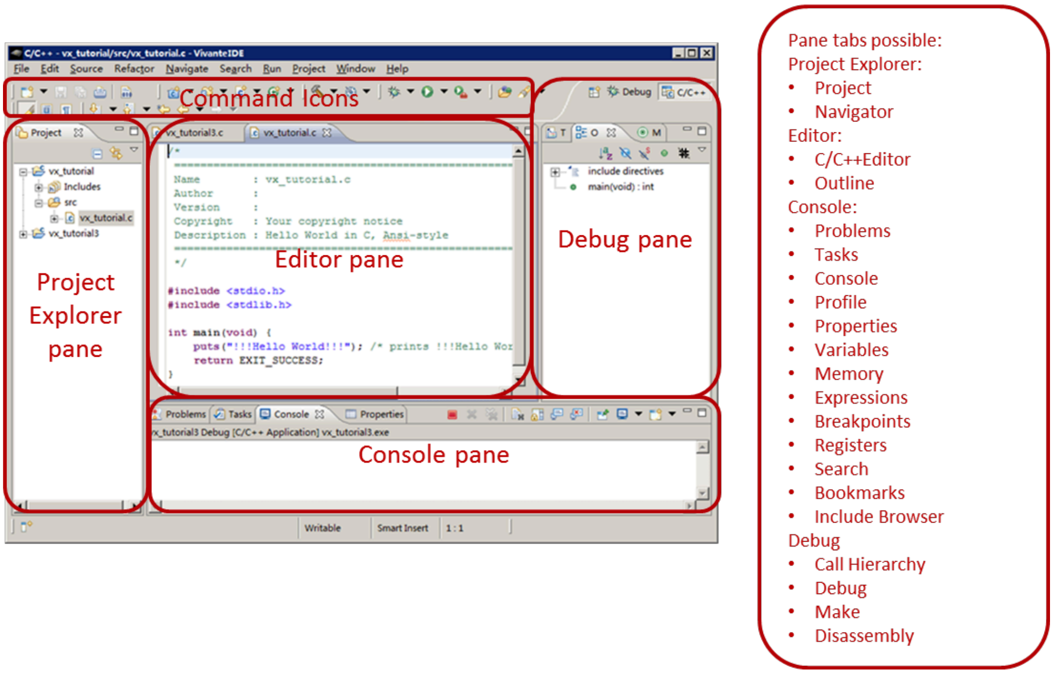


Figure 8. VivanteIDE Workbench Key Panes

The following examples provide users with basic simple steps to get started using VivanteIDE. The GUI is similar but not identical for each tool GUI: VPD Analyzer, Shader Assistant, Texture Browser, Texture Viewer.

13.4.1 Selecting a workspace

When VivanteIDE is opened, the **Workspace Launcher - Select a workspace** dialog box pops up by default.

Click the **OK** button.

If the workspace is a new empty workspace, the **Welcome** dialog box is displayed.

If the workspace is not a new empty workspace, the workbench is displayed.

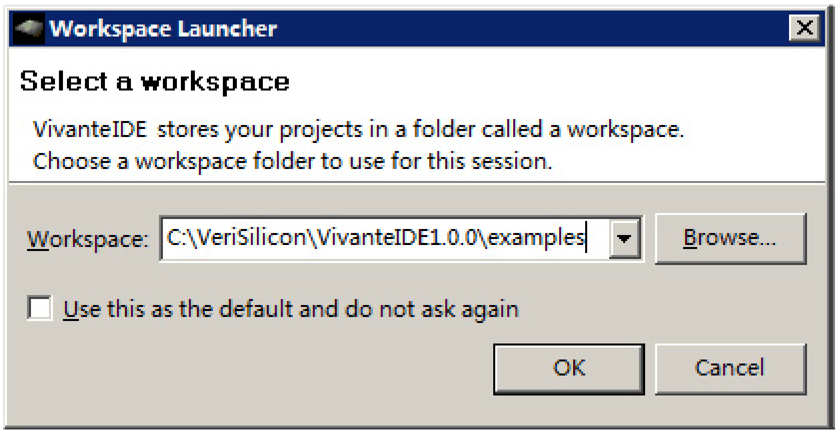


Figure 9. Figure 21. Workspace Launcher

13.4.2 Switching perspective

Click the pull-down menu items or click directly on the visible perspective name to switch perspective views.

Switch to the C/C++ perspective to manage projects and write source code. VivanteIDE will switch to the Debug perspective by default after a program is launched successfully in Debug mode.

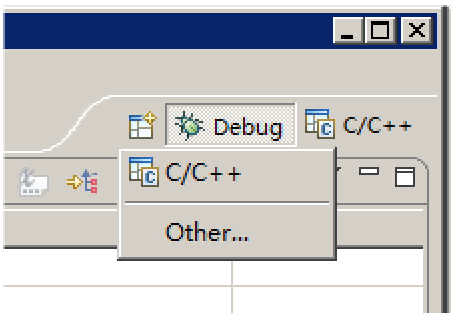


Figure 10. Switching perspective

13.4.3 Creating a new project

This section describes how to create an OpenVX project as an example.

New project creation is available from the main menu. Choose **File-->New-->Project...**

In the **New Project - Select a wizard** dialog box, open the **C/C++** folder in the **Wizards** list box and select **OpenVX C Project**.

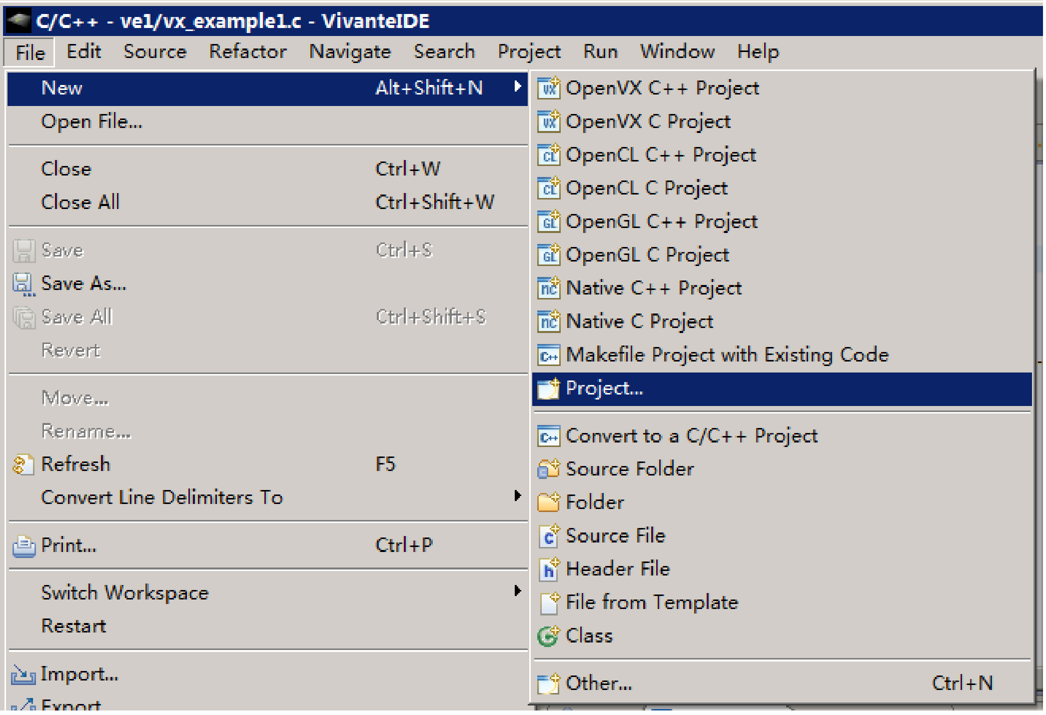


Figure 11. Creating a new project

13.4.4 Creating an OpenVX kernel wizard

1. To create an **OpenVX C(C++)** project, in the **OpenVX C(C++) Project** dialog box, enter the Project name, select **OpenVX Kernel Project(1.1)** under **Static Library** or **Shared Library**.

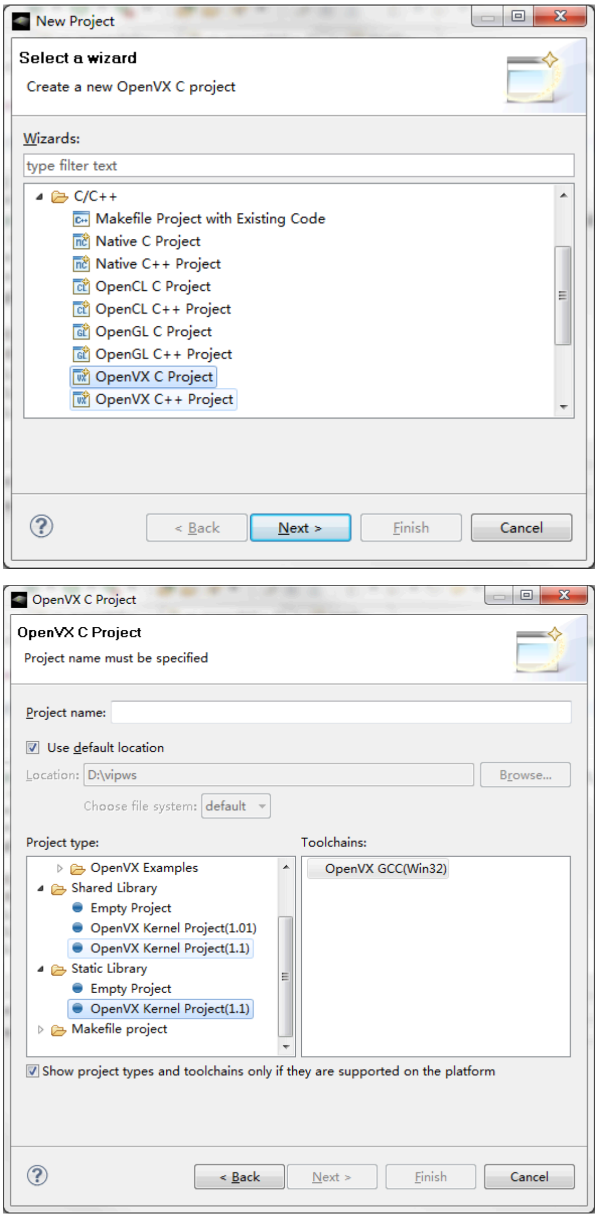


Figure 12. Creating a new project (1)

2. Press **Next** to input **Author** and **Copyright notice**, **Kernel ENUM offset** and **Kernel Name prefix** information in the following dialogs, and then add arguments for the kernel.

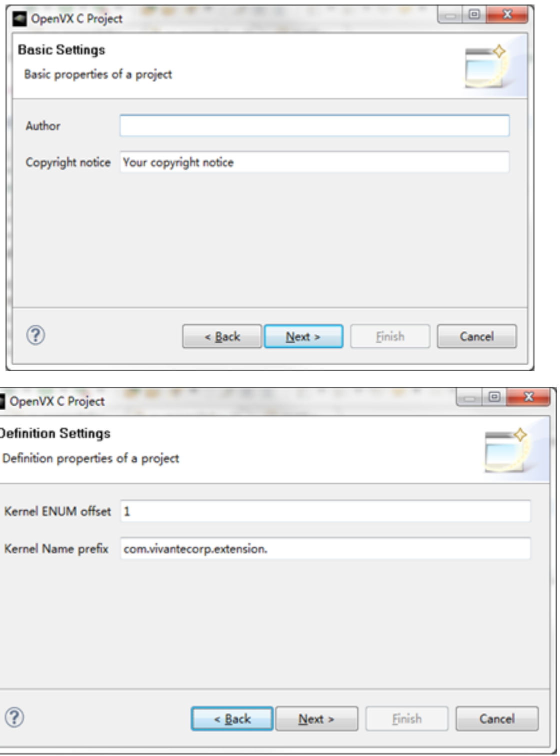
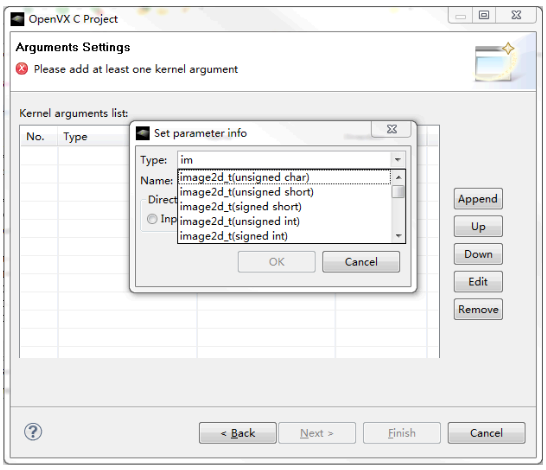


Figure 13. Creating a new project (2)

3. Click the **Finish** button, and the new kernel project will be created.
   Refer to the *VivanteIDE User Guide* for detailed information.



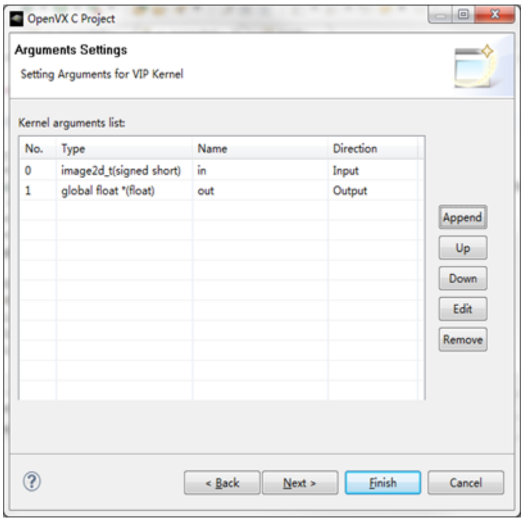


Figure 14. Creating a new project (3)

13.4.5 Source code smart editing for OpenVX and OpenCL

When a user edits a source file in VivanteIDE, the OpenVX/OpenCL keywords and predefined structure will be automatically highlighted. The Editor also supports keyword completion using keyboard combination "alt"+"/".

In addition, the **Outline** view tab will provide structured information and quick navigation for the source file currently being edited.

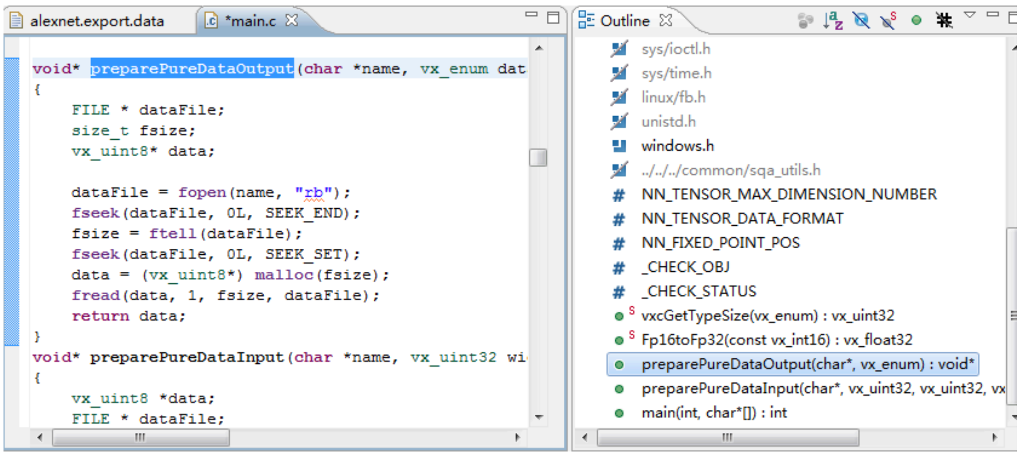


Figure 15. Source code smart editing for OpenVX and OpenCL (1)

All information provided in this document is subject to legal disclaimers.

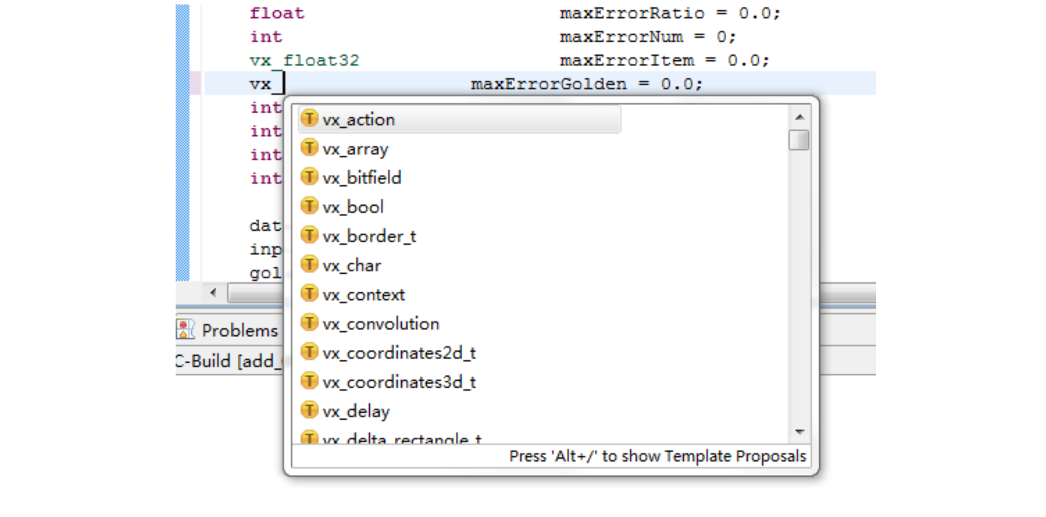


Figure 16. Source code smart editing for OpenVX and OpenCL (2)

13.4.6 Creating a Neural Network Inference Project from a model file

New project creation is available from the main menu.

1. Choose **File-->New-->Project**...

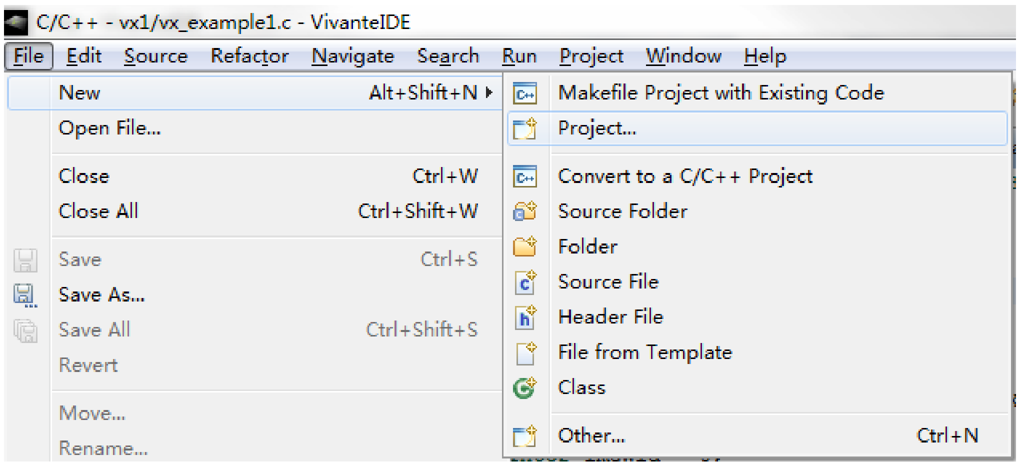


Figure 17. Creating a Neural Network Inference Project from a model file (1)

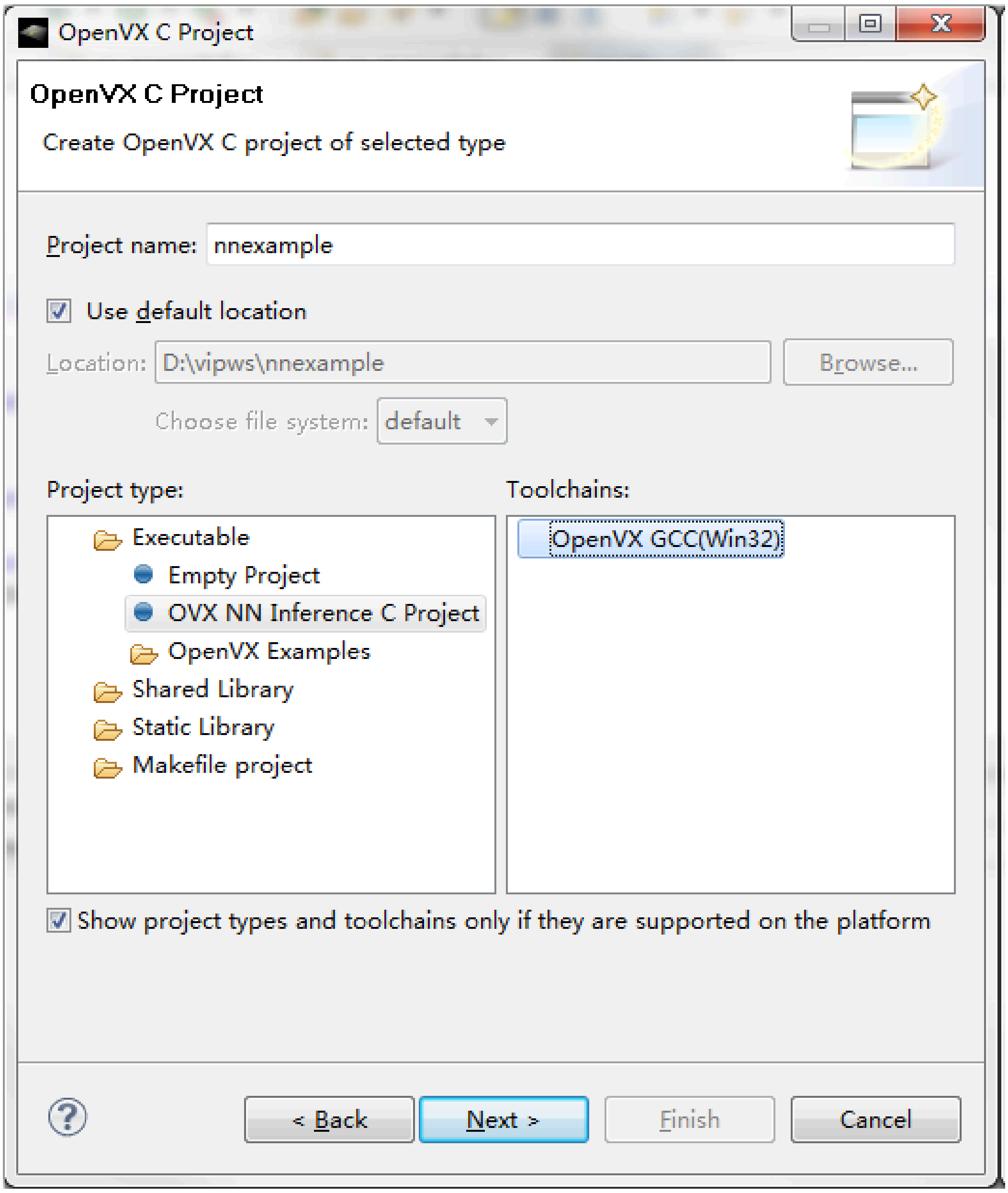


Figure 18. Creating a Neural Network Inference Project from a model file (2)

2. In the **New Project - Select a wizard** dialog box, open the **C/C++** folder in the **Wizards** list box and select **OpenVX C Project**.

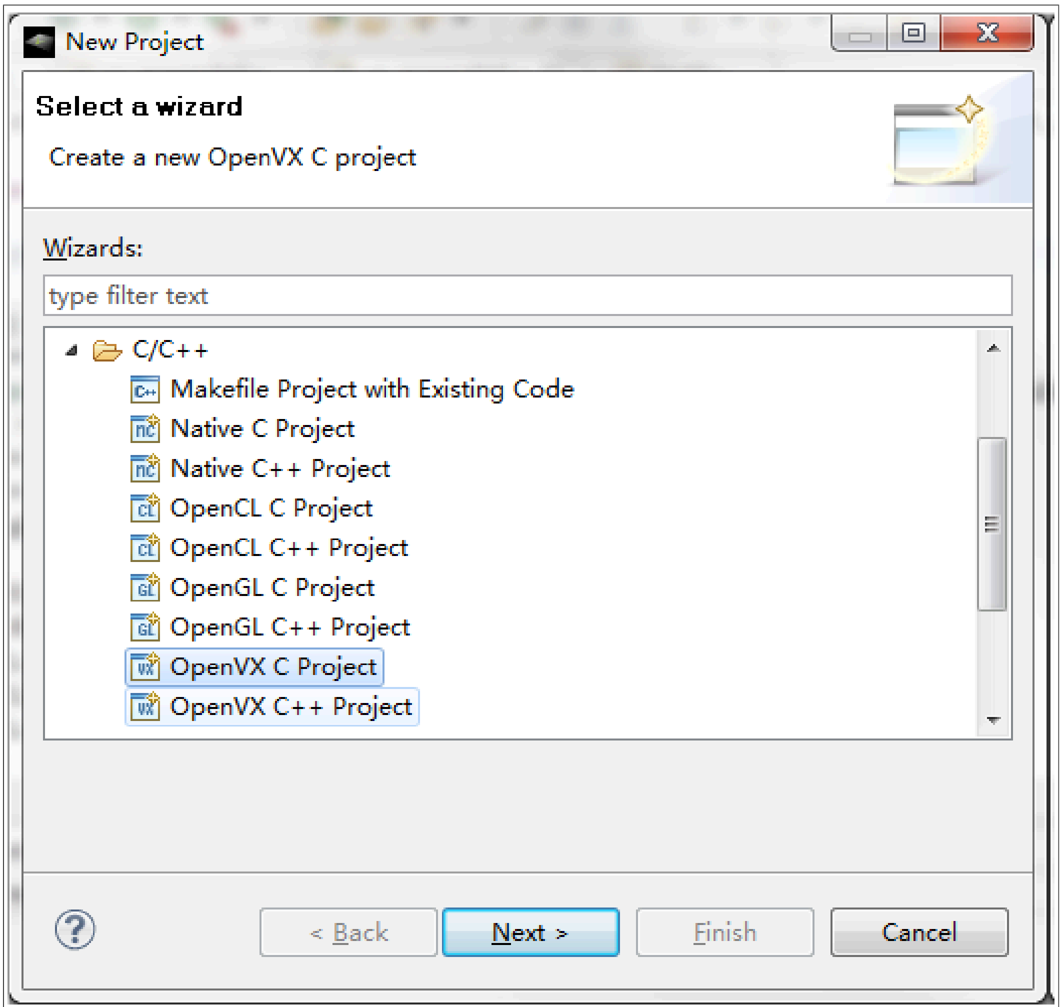


Figure 19. Creating a Neural Network Inference Project from a model file (3)

3. Click **Next** to continue.
4. In the **OpenVX C Project** dialog box, enter the Project name. Check the ***Use default location*** checkbox. This will cause our new directory to be created in our workspace. The directory path is displayed.
5. Select the Project type: **Executable -> OVX NN Inference C Project**.
6. Once the project name is entered, click **Next** to continue. The **OpenVX C Project - Basic Settings** dialog box is displayed.

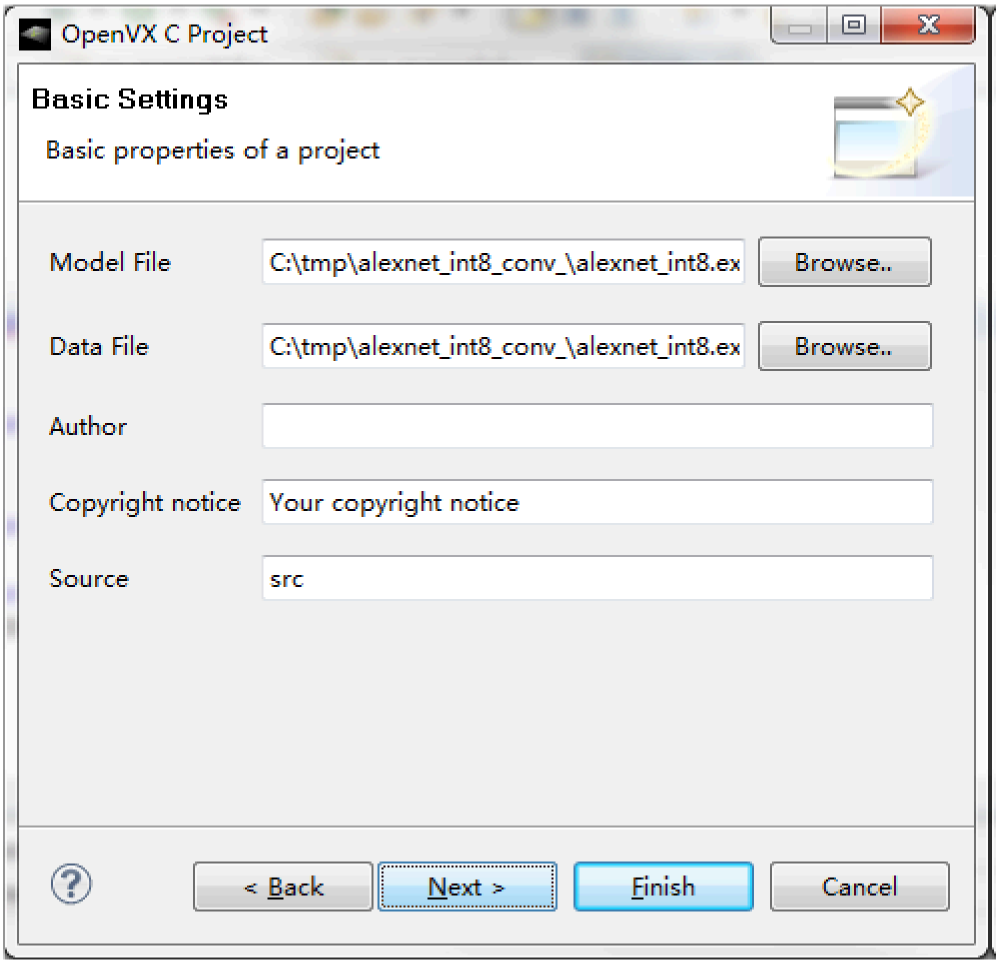


Figure 20. Figure 31. Creating a Neural Network Inference Project from a model file (4)

7. Browse or input the information to select a **Model file** and a **Data file**.
8. Click **Next** to continue. The **OpenVX C Project - Conversion Settings** dialog box is displayed. Make sure the **Add reference main.c** checkbox is checked.
   Note:
   *If **Add reference main.c** is checked, a main.c would be created by this wizard. If it is unchecked, main.c would not be created.*
   `Function main()` locates in `main.c`, which is just an application for testing the model.
   *Usually the NN model is a part of an OpenVX application, so writing function main to use the NN model is still neccessary to execute the project if **Add reference main.c** is not checked.*
9. Click **Next** to continue. The **OpenVX C Project - Select Configurations** dialog box is now displayed.

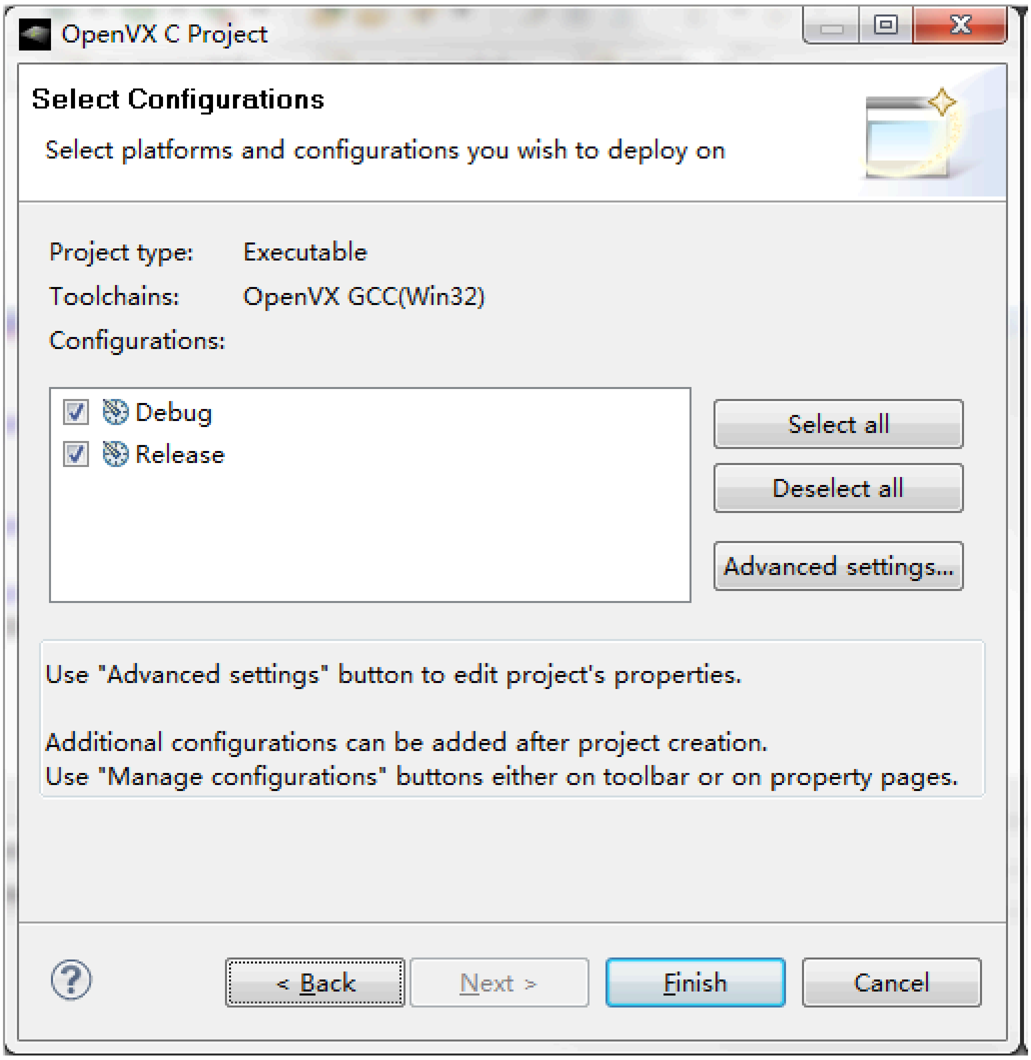


Figure 21. Creating a Neural Network Inference Project from a model file (5)

10. Click the **Finish** button. The new project is now created. The new Project is viewable in the **Project Explorer** pane.

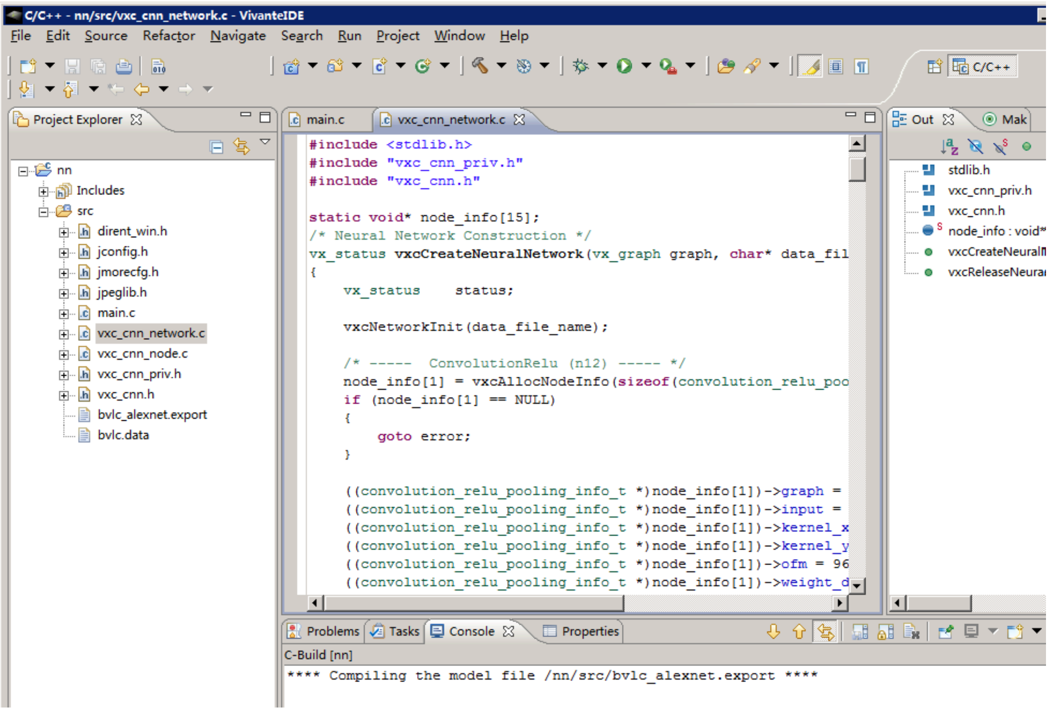


Figure 22. Creating a Neural Network Inference Project from a model file (6)

13.4.7 Building a sample project

1. On the **Project** tab, select **Properties** to open the **Properties Setting** dialog to modify the build settings.

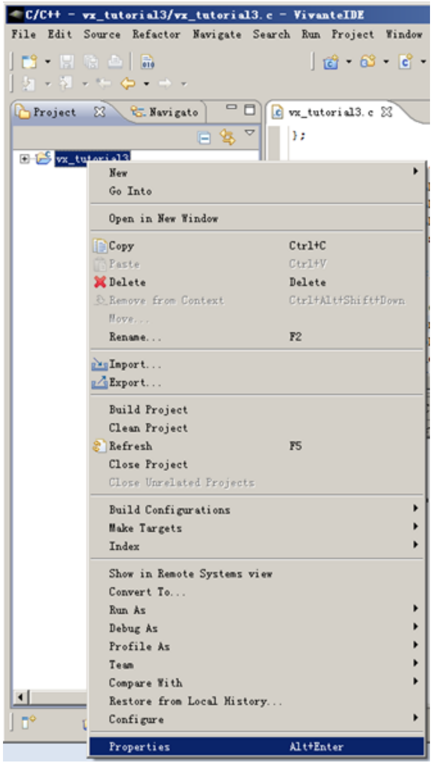


Figure 23. Building a sample project (1)

2. There are build tools available that can be set for C or C++ projects.

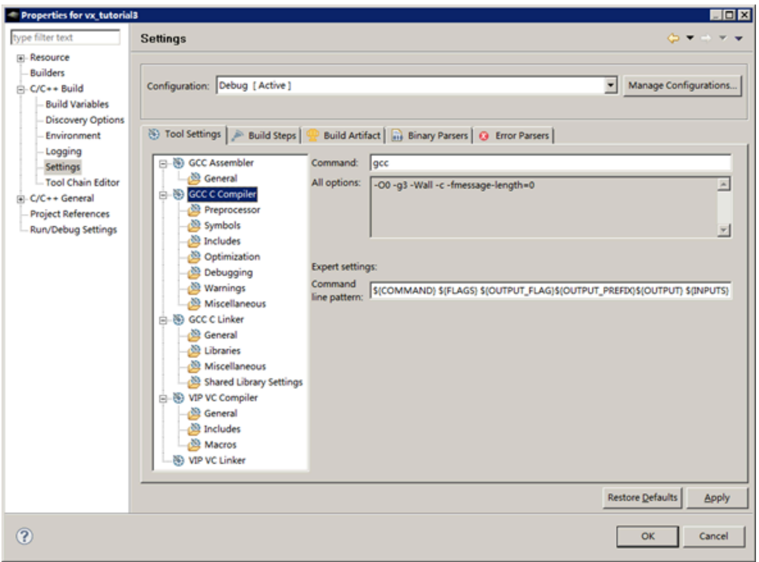


Figure 24. Building a sample project (2)

3. The sample project **'vx_tutorial3'** is ready to build after the build settings are saved. You can build the **'vx_tutorial3'** project by using one of following two methods, with the target project selected in the left pane:
   - Choose from the main menu **Project->Build Project**.
   - Right-click the target project and select **Build Project**.

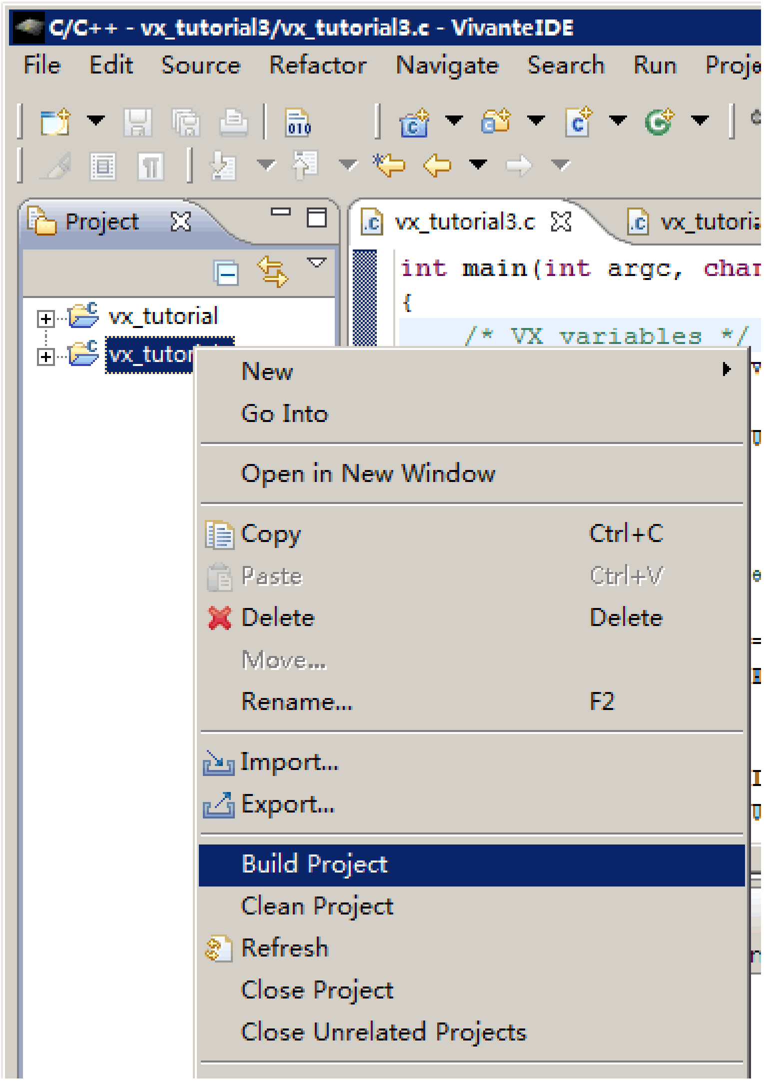


Figure 25. Building a sample project (3)

4. The build results are displayed on the **Console** and **Problems** tabs of the lower right pane of the application.

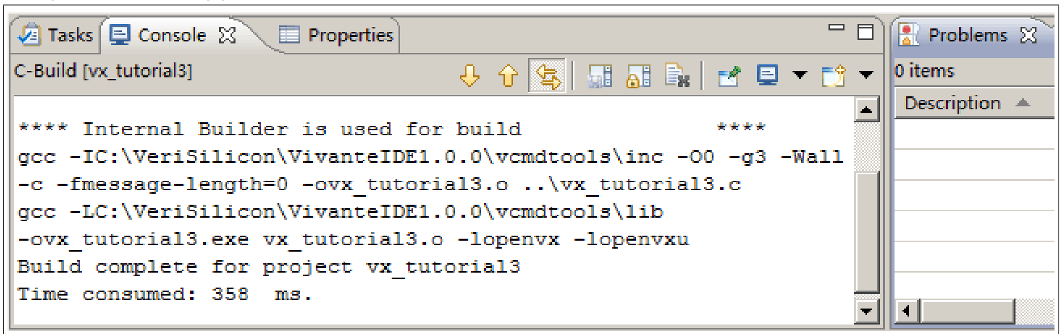


Figure 26. Building a sample project (4)

5. If **No error occurs. build was successful,** the executable file is displayed in the **Project Explorer** pane.

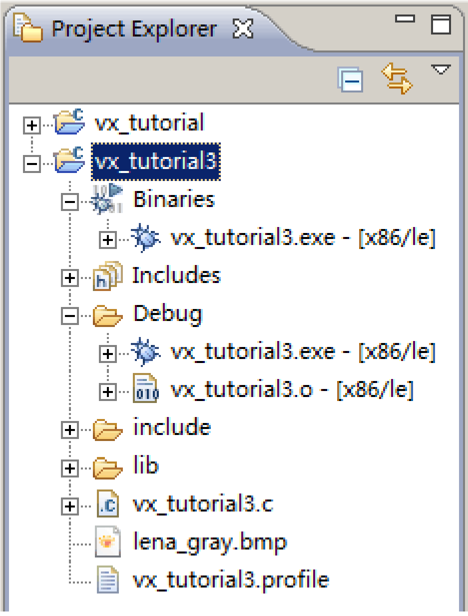


Figure 27. Building a sample project (5)

6. Use the **Build Steps** tab on the **Properties > C/C++ Build > Settings** dialog to customize the selected build configuration allowing for the specification of user defined build command steps, as well to enable displaying of descriptive messages in the build output, immediately before and after, normal build processing.

13.4.8 Debugging and profiling a project

1. To open the **Debug Configurations** dialog box, select **Run->Debug Configurations...** from the main menu.
2. Set the dialog options, and then click **Debug** to debug your project.

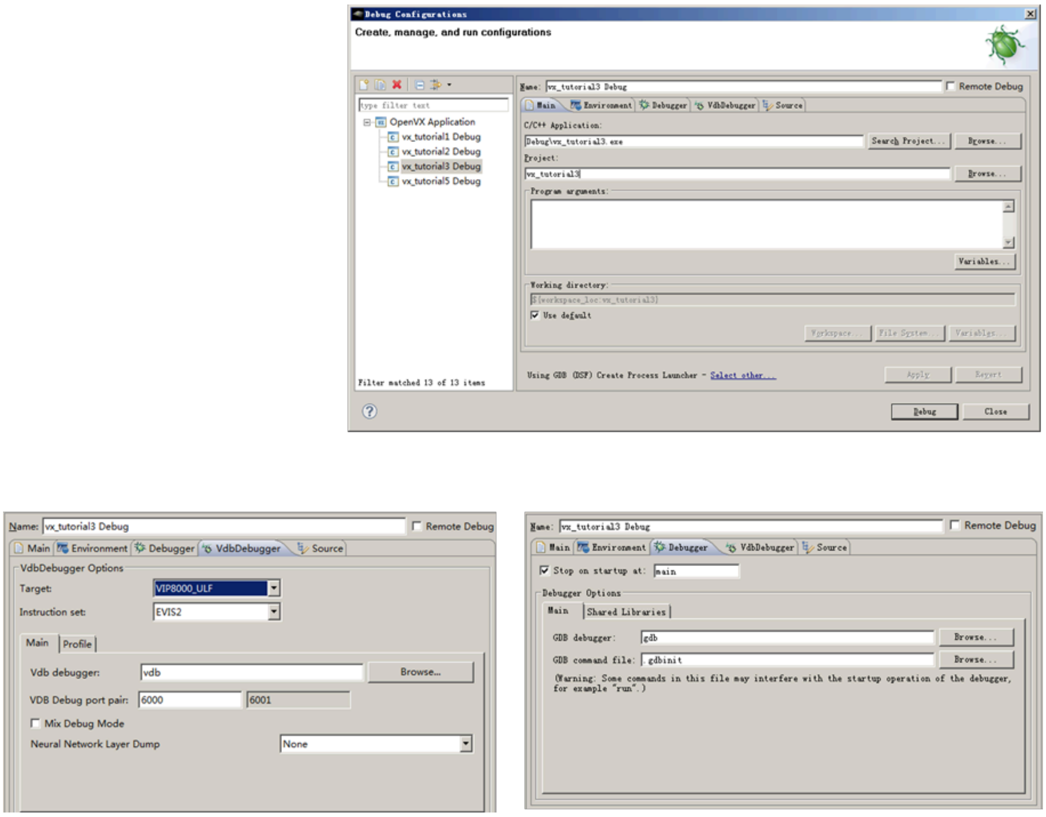


Figure 28. Debugging and profiling a project

13.5 VivanteIDE – Debug and Profiling

13.5.1 Fundamentals of performance optimization

Whenever an application runs on a computer, it makes use of one or more of the available resources. These

compute resources include the CPU, the graphics processor, caches and memory, hard disks, and possibly even the

network. Viewed simplistically, it is always true that one of these resources is the limiting factor in how quickly the

application can finish its tasks. This limiting resource is the performance bottleneck. Remove this bottleneck, and

application performance should be improved. Note, however, that removing one limiting factor always promotes

something else to become the new performance bottleneck.

The goal of optimizing, or tuning, application performance is to balance the use of resources so that none of them

holds back the application more than any of the others. In practice there is no single, simple way to tune an application. The whole system needs to be considered, including the size and speed of individual components as

well as interactions and dependencies among components.

vProfiler collects information on GPU usage and on calls to Vivante functions within the graphics pipeline. As such

it provides an excellent view into what is happening on the GCCORE graphics processor at any point in time, down

to the individual frame. When the application performance is GPU-bound, vProfiler and VPD Analyser are the right

tools to help determine why.

Note that the initial determination regarding which component of the computer system is the performance

bottleneck—CPU, GPU, memory, etc.—is the domain of system performance analyzers and is outside the scope of

the GPU tools. A list of such performance analysis tools can be found at Wikipedia:

en.wikipedia.org/wiki/List_of_performance_analysis_tools

13.5.2 VPD Analyzer for Analyzing Performance Data

vProfiler is a run-time environment for collecting performance statistics of an application and the graphics pipeline. The VPD Analyzer perspective view is provided to facilitate graphically displaying the data gathered by vProfiler and aiding in visual analysis of graphics performance. Used together, these tools assist software developers in optimizing application performance on Vivante enabled platforms.

13.5.3 vProfiler

When building Vivante Graphics Drivers, the driver is built with vProfiler capability. vProfiler gathers data from these counters during runtime and can track data for a range of frames or a single frame from any graphics, compute application. vProfiler outputs performance data to binary files with a `.vpd` extension. These files can be using the VivanteIDE VPD Analyzer both in text lists and as line graphs. VPD Analyzer gives the user several ways to inspect any frame in a captured animation sequence.

13.5.4 Enabling vProfiler on Linux OS

When building Vivante Graphics Drivers in a Linux OS environment, the driver is built with vProfiler capability.

- vProfiler functionality can be enabled by `export VIV_PROFILE=1`.
- To enable OpenVX profile, use `export VIV_VX_PROFILE=1`.
- To enable OpenCL profile, use `export VIV_CL_PROFILE=1`.

Kernel module driver arguments are no longer needed.

13.5.4.1 Setting vProfiler property options for OpenGL ES

vProfiler property options are set using environment variables on Linux. The following table summarizes the environment variables that vProfiler supports.

Table 32. vProfiler property options

Environment Variable	Description
VIV_PROFILE	[0] Disable vProfiler (default), [1] Enable vProfiler, [2] Control via application call, [3] Allows control over which frames to profile with v Profiler
VP_OUTPUT	Specify the output file name of vProfiler (default is vprofiler.vpd)
VP_FRAME_NUM	When VIV_PROFILE=1, specify the number of frames dumped by v Profiler.
VP_FRAME_START	When VIV_PROFILE=3, specify the frame to start profiling with v Profiler.
VP_FRAME_END	When VIV_PROFILE=3, specify the frame to end profiling with v Profiler.
VP_USE_GLFINISH	Enable [1] or disable [0] the use of glFinish()/glFlush() APIs as the frame delimiter in addition to eglSwapBuffers() (default 0). This variable enables application thread which does not use eglSwap Buffers() to generate useful GPU profiling data for analysis.
VP_PERDRAW_MODE	Enable [1] or disable [0] (default). When enabled, vProfiler will collect a counter for each draw call.
VP_DISABLE_PROBE	Disables PROBE mode and makes vProfiler to use AHB counters for profiling.
VP_ENABLE_PRINT	Enable vProfiler to print out the counter information to the console.

13.5.5 Setting vProfiler property options for Vision, OpenVX Profiling

vProfiler for OpenVX Profiling (for use with Vision/VIP/VX IP) is similar to vProfiler for OpenGL, except that fewer environment variables and fewer supported values for those variables are available.

Table 33. vProfiler property options

Environment Variable	Description
VIV_VX_PROFILE	[0] Disable vProfiler for OpenVX(default), [1] Enable vProfiler for OpenVX
VIV_CL_PROFILE	[0] Disable vProfiler for OpenCL(default), [1] Enable vProfiler for OpenCL
VP_OUTPUT	Specify the output file name of vProfiler (default is vprofiler.vpd)

13.5.6 Enabling vProfiler Option for Android OS

i.MX Android release GPU drivers are built with vProfiler capability. To enable the vProfiler feature, boot the Android image, and then stop U-Boot by pressing a key on the serial terminal.

Perform the following steps to capture the VPD file using vProfiler on Android OS.

1. Set application name to be profiled, for example, nenamark2 application.
```
setprop VP_PROCESS_NAME se.nena.nenamark2
```
2. Set the profile output file path, for example, nenamark2 application.
```
setprop VP_OUTPUT /data/data/se.nena.nenamark2/
```

3. Start profiling.

```
setprop VIV_PROFILE 1
```

4. Run application and check if the `*.vpd` file is generated in the path indicated by VP_OUTPUT, for example, nenamark2 application.

```
ls -l /data/data/se.nena.nenamark2/*.vpd
```

5. Stop profiling.

```
setprop VIV_PROFILE 0
```

13.5.7 Setting vProfiler property options for OpenGL ES Profiling with Android

The following table summarizes the property options that vProfiler supports through running the command adb shell setprop [OPTIONS]. These options are similar to the environment variables available for Linux.

Table 34. vProfiler property options

adb shell setprop OPTIONS	Description
setprop VIV_PROFILE 0	Run this command in adb shell to disable vProfiler in the drivers
setprop VIV_PROFILE 1	Run this command in adb shell to enable vProfiler in the drivers
setprop VIV_PROFILE 2	Run this command in adb shell to have vProfiler enable/ disable controlled in the application by glEnable(GL_ PROFILE_VIV) and glDisable(GL_PROFILE_VIV) calls.
setprop VIV_PROFILE 3 setprop VIV_FRAME_START setprop VIV_FRAME_START	Run these commands in adb shell to have vProfiler start-stop at frames specified in VP_FRAME_START and VP_FRAME_ END.
setprop VP_PROCESS_NAME appname	Run this command in adb shell to specify the application you need to profile. Change the app name as needed to profile another application. NOTES: There may be different sub-case names used by an app. Be sure to accurately specify a case name to match the name that you saw on the command line when using ps command. This option is only available for Android, not available for Linux.
setprop VP_OUTPUT newpath	Run this command in adb shell to specify a new location for vProfiler output. By default, the vpd file will created under / sdcard/. If an application has no access to the SD card, you can specify another path where the application does have write permission. NOTE: For applications which initialize during Android system boot startup, such as launcher, you need to kill the process after you change to a new path. When the application automatically restarts, then your vpd will be accessible where you want it.
setprop VP_FRAME_NUM xxx	Run this command in adb shell to limit the number of frames to analyze. For example, to make vProfiler dump performance data for the first 100 frames: setprop VP_FRAME_NUM 100. NOTES: Only use when VIV_PROFILER is set to 1. **When this option is not used, the profile file generated when running an application for a long time can be very large. This takes up a large amount of disk space and also makes it hard to view the data in vAnalyzer.**

Table 34. vProfiler property options *...continued*

adb shell setprop OPTIONS	Description
setprop VP_USE_GLFINISH 0 setprop VP_USE_GLFINISH 1	Run this command in adb shell to enable or disable use of glFinish()/glFlush() as the frame delimiter in addition to egl SwapBuffers() (default 0). By default eglSwapBuffers() is used as the frame delimiter. This command will make application thread which does not use eglSwapBuffers() to generate useful GPU profiling data for analysis.
setprop VP_PERDRAW_MODE 0 setprop VP_PERDRAW_ MODE 1	Run this command in adb shell to enable or disable per draw mode. When enabled, vProfiler will collect a counter for each draw call.
setprop VP_DISABLE_PROBE	Run this command in adb shell to disable PROBE mode and make vProfiler use AHB counters for profiling.
setprop VP_ENABLE_PRINT	Run this command in adb shell to enable vProfiler to print out the counter information to the console.

13.5.8 vProfiler Set Property Options for Vision/OVX Profiling with Android

vProfiler for Vision Profiling (for use with Vision/VIP/VX IP) is similar to vProfiler for OpenGL, except that fewer property options and fewer supported values are available.

Table 35. vProfiler Set Property Options

adb shell setprop OPTIONS for VIP/VX/OVX	Description
`setprop VIV_VX_PROFILE 0`	Run this command in adb shell to disable vProfiler in the drivers
`setprop VIV_VX_PROFILE 1`	Run this command in adb shell to enable vProfiler in the drivers
`setprop VP_PROCESS_NAME appname`	Run this command in adb shell to specify the application you need to profile. Change the app name as needed to profile another application. NOTES: There may be different sub-case names used by an app. Be sure to accurately specify a case name to match the name that you saw on the command line when using ps command. This option is only available for Android, not available for Linux.
`setprop VP_OUTPUT newpath`	Run this command in adb shell to specify a new location for v Profiler output. By default, the vpd file will be created under / sdcard/. If an application has no access to the SD card, you can specify another path where the application does have write permission. NOTE: For applications that initialize during Android system boot startup, such as launcher, you need to kill the process after you change to a new path. When the application automatically restarts, then your vpd will be accessible where you want it.

13.5.9 Enabling vProfiler Option for QNX

When building the Vivante Graphics Drivers for QNX environment, build the driver with the vProfiler capability.

The `graphics.conf` file contains the configuration information for Screen and is found under the following directory:

`SCREEN-DIR/usr/lib/graphics/TARGET-SPECIFIC`

To activate the vProfiler functionality, add the **gpu-gpuProfiler=1** option into the *khronos* section of the

corresponding `graphics.conf` file:

```
begin khronos
...
begin wfd device 1
...
gpu-gpuProfiler=1
...
end wfd device
...
end khronos
```

13.5.9.1 Setting vProfiler Environment Variables for OGL/OES Profiling

The following table summarizes the environment variables that vProfiler supports.

Table 36. vProfiler Environment Variables

Environment Variable	Description
VIV_PROFILE	[0] Disable vProfiler (default), [1] Enable vProfiler, [2] Control via application call, [3] Allows control over which frames to profile with v Profiler
VP_OUTPUT	Specify the output file name of vProfiler (default is vprofiler.vpd)
VP_FRAME_NUM	When VIV_PROFILE=1, specify the number of frames dumped by v Profiler.
VP_FRAME_START	When VIV_PROFILE=3, specify the frame to start profiling with v Profiler.
VP_FRAME_END	When VIV_PROFILE=3, specify the frame to end profiling with v Profiler.
VP_USE_GLFINISH	Enable [1] or disable [0] the use of glFinish()/glFlush() APIs as the frame delimiter in addition to eglSwapBuffers() (default 0). This variable enables application thread which does not use eglSwap Buffers() to generate useful GPU profiling data for analysis.
VP_PERDRAW_MODE	Enable [1] or disable [0] (default). When enabled, vProfiler will collect a counter for each draw call.
VP_DISABLE_PROBE	Disables PROBE mode and makes vProfiler to use AHB counters for profiling.
VP_ENABLE_PRINT	Enable vProfiler to print out the counter information to the console.

13.5.9.2 Setting vProfiler Environment Variables for Vision, OpenVX Profiling

vProfiler for OpenVX Profiling (for use with Vision/VIP/VX IP) is similar to vProfiler for OpenGL, except that fewer environment variables and fewer supported values for those variables are available.

Table 37. vProfiler Environment Variables

Environment Variable	Description
VIV_VX_PROFILE	[0] Disable vProfiler for OpenVX(default), [1] Enable vProfiler for OpenVX
VIV_CL_PROFILE	[0] Disable vProfiler for OpenCL(default), [1] Enable vProfiler for OpenCL
VP_OUTPUT	Specify the output file name of vProfiler (default is vprofiler.vpd)

13.5.10 Environment Variable Details

13.5.10.1 VIV_PROFILE

The environment variable VIV_PROFILE can be used to control enable/disable and set profiling modes for vProfiler.

- **VIV_PROFILE=0**
  By default, vProfiler is disabled in the driver. If vProfiler has been enabled and you wish to disable it, set VIV_PROFILE to 0:

```
export VIV_PROFILE=0
```

- **VIV_PROFILE=1**
  To enable vProfiler, set VIV_PROFILE to 1:

```
export VIV_PROFILE=1
```

  To limit the number of frames to analyze, use the environment variable VP_FRAME_NUM. (This option is available only when VIV_PROFILE=1.) For example, this setting will make vProfiler dump performance data for the first 100 frames.

```
export VP_FRAME_NUM=100
```

- **VIV_PROFILE=2**
  Mode VIV_PROFILE=2 provides support for glEnable(GL_PROFILE_VIV) and glDisable(GL_PROFILE_VIV), which are used to choose which frames are to be profiled. In this mode, vProfiler is disabled by default. It begins to do profiling only after a glEnable(GL_PROFILE_VIV) call from the application. And it will stop profiling when glDisable (GL_PROFILE_VIV) is called. Note that the flag is only checked at every frame end, i.e., in eglSwapBuffers(). To use this mode, set VIV_PROFILE to 2:

```
export VIV_PROFILE=2
```

- **VIV_PROFILE=3**
  Setting VIV_PROFILE to 3 provides support for two environment variables VP_FRAME_START and VP_FRAME_END, which are used to choose which frames are to be profiled. In this mode, vProfiler is disabled by default. It begins to do profiling starting at the frame number specified by VP_FRAME_START, and it ends the profiling after the frame number specified by VP_FRAME_END. For example to use this mode, set VIV_PROFILE to 3:

```
export VIV_PROFILE=3 export VP_FRAME_START=10 export
 VP_FRAME_END=90
```

  Note:
  *To get precise profiling data, the IP's Power Management (PM) functions need to be disabled. When kernel module **galcore** is inserted with gpuProfiler=1, the PM functions*

in the driver are not disabled. The PM functions are disabled when VIV_PROFILE is set to 1, 2, or 3, and the application starts. The PM functions are enabled when VIV_PROFILE is set to 0, and the application starts again.

13.5.10.2 VP_OUTPUT

The output file of vProfiler is `vprofiler.vpd` by default. To specify an alternate filename use the environment variable `VP_OUTPUT`. For example,

```
export VP_OUTPUT=sample.vpd
```

13.5.10.3 VP_USE_GLFINISH

glFinish()/glFlush() will be treated as the frame delimiter in addition to eglSwapBuffers(). By default, vProfiler only uses eglSwapBuffers() as the delimiter to check hardware counters. The command below will enable vProfiler to use glFinish()/glFlush() as additional delimiters so an application thread which does not use eglSwapBuffers() can generate useful profiling data for analysis.

```
export VP_USE_GLFINISH=1
```

13.5.10.4 VP_DISABLE_PROBE

This variable only applies to IP with the PROBE feature support. It disables PROBE mode and makes vProfiler use AHB counters for profiling. This variable has no affect on hardware that only supports the AHB counter. The default value is off.

13.5.10.5 VP_ENABLE_PRINT

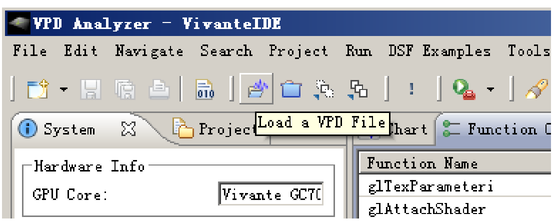
This variable provides a convenient way to check some critical profiling information without using the off-line vAnalyzer to open a VPD file. Once it is enabled, vProfiler prints out the counter information to the console. For the OpenVX and OpenCL drivers, the default value is on; for GLES and GL drivers, the default value is off.

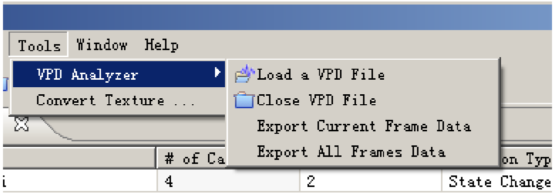
13.6 VPD Analyzer

VPD Analyzer provides graphic displays of the data gathered by vProfiler and aids in the visual analysis of graphics, compute and vision performance. vProfiler outputs performance data to binary files with a `.vpd` extension. These files can be opened using the VivanteIDE VPD Analyzer both in text lists and as line graphs. VPD Analyzer gives the user several ways to inspect any frame in a captured animation sequence.

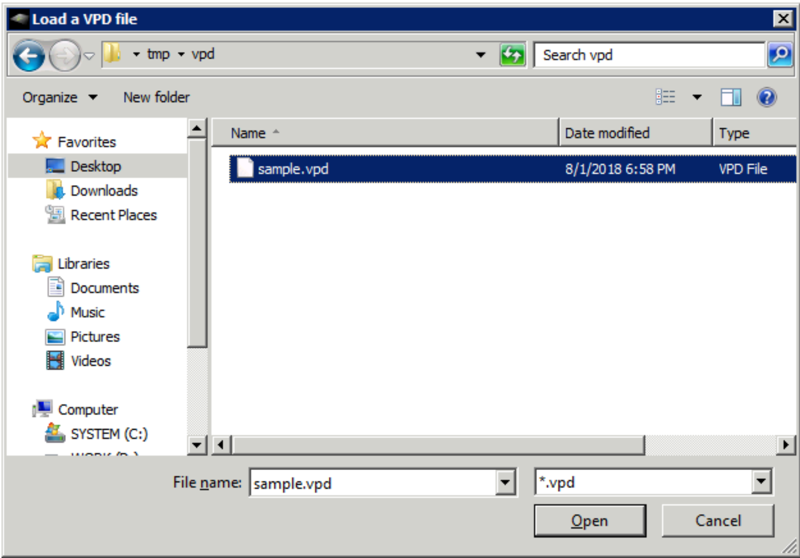
13.6.1 Loading a VPD File

To open the VPD Analyzer perspective based on a VPD file, click the icon from the toolbar or select **Tools->VPD Analyzer->Load VPD File ...**

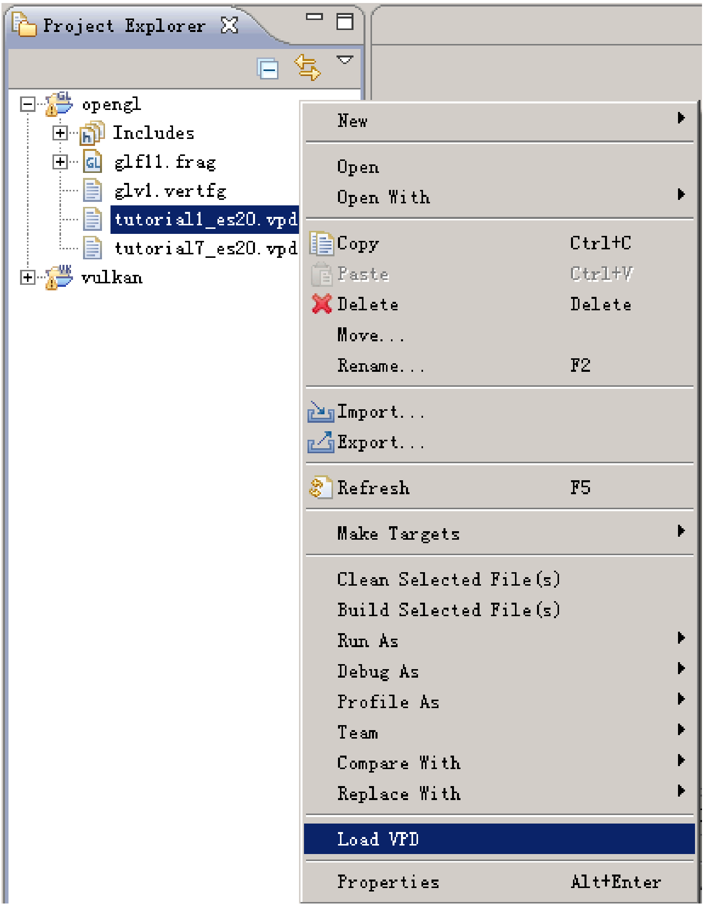




The **Load a VPD file** dialog box appears. Select a VPD (.vpd) file, and click **Open**.



Or, in the Project Explorer view, right-click on a VPD file and select Load VPD.



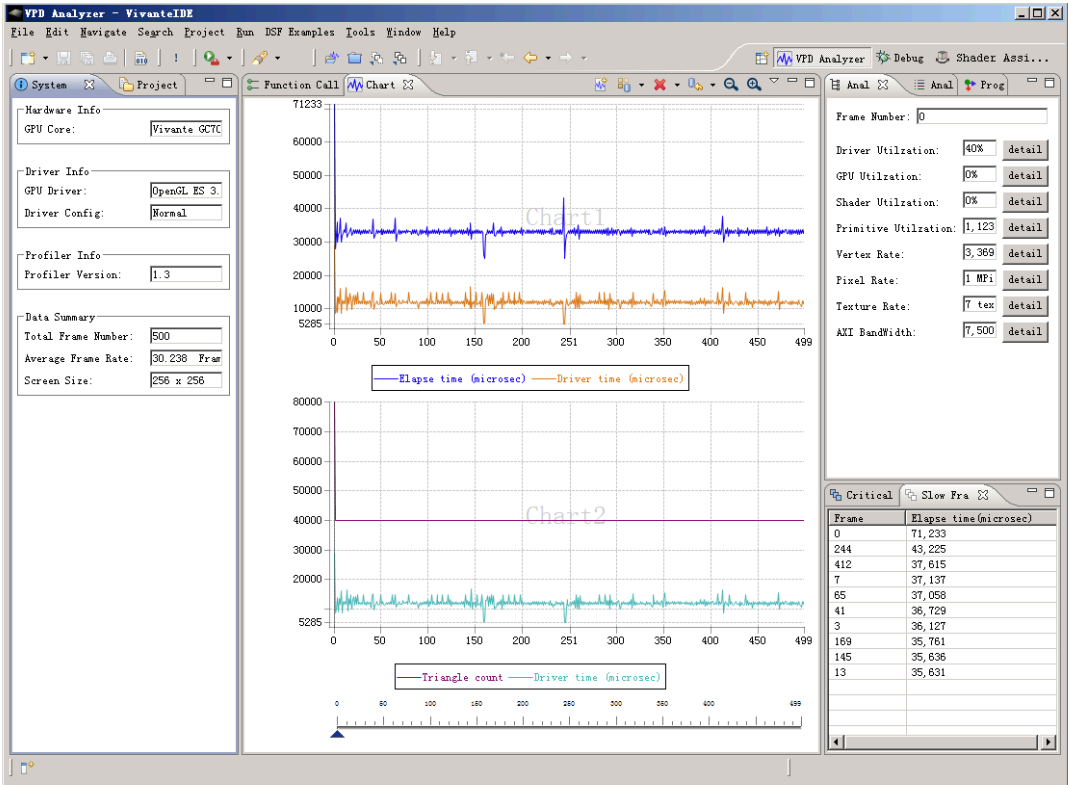
13.6.2 VPD Analyzer Perspective

Once the VPD file is loaded, the VivanteIDE workbench switches to the VPD Analyzer perspective view, and analyze data from the selected VPD file will be displayed on a series of tabs in chart or text format.

Available tabs (left to right) are:

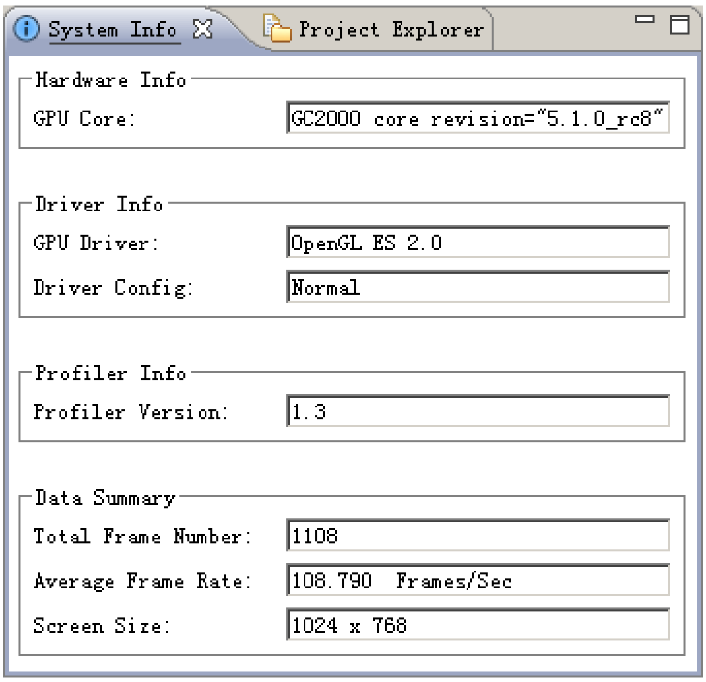
Table 38. Available tabs

VPD Analyzer Tab	Description
System Info	Shows hardware and software version information and Average Frame Rate
Project Explorer	Shows project files
Chart	Shows customizable graph views of various counters
Function Call	Three panes shows a table of functions called, a graph of Top 5 calls and properties of the selected call.
Analysis Summary	Shows data for the current frame
Analysis Detail	Shows analysis detail for the current frame
Program	Shows program counters and their value



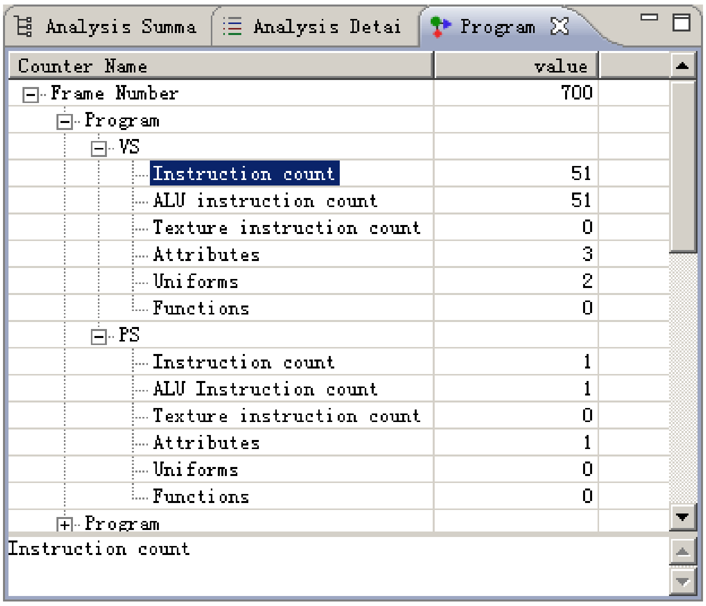
13.6.3 System Info View

The left most System Info tab shows the system information related to the VPD data under analysis, such as hardware, driver and vProfiler versions. The Average Frame Rate is also reported on this tab.



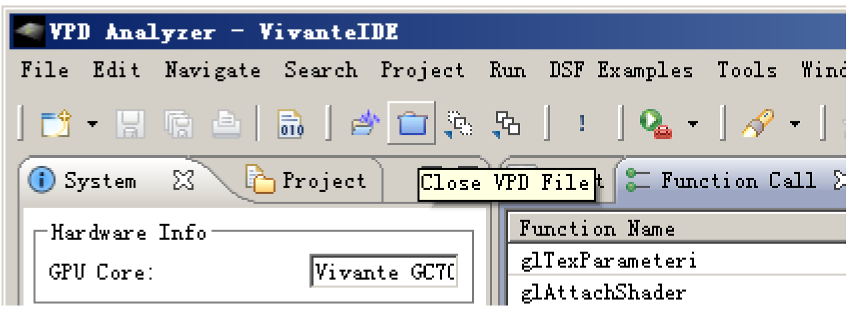
13.6.4 Program Counters View

The rightmost tab in the rightmost pane is the Program tab which shows program counter information, such as Instruction counts and attribute counts.



13.6.5 Closing the VPD File

Click the icon  from the toolbar or select **Tools->VPD Analyzer->Close VPD File** to close the current VPD file. The analysis data associated with the closed file will be cleared from all views.



13.7 SPIR-V Disassembler

A SPIR-V Disassembler tool is provided as an aid in debugging Vulkan applications. If a SPIR_V file is already located in a project, simply double click on it to disassemble. Otherwise use the main menu **File -> Open File…** to locate the SPIR-V. Options can be set via the **Window->Preferences** dialog box.

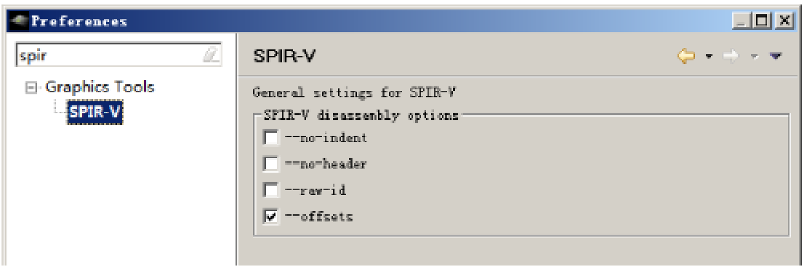


Figure 29. SPIR-V Disassembler

13.7.1 Shader Assistant

Shader Assistant perspective is provided for Shader program development for OpenGL, OpenCL and Vulkan projects. Shader Assistant provides an environment for editing, previewing, analyzing, and optimizing shader programs. Shader Assistant includes samples of shader programs, a number of standard meshes (sphere, cube, tea pot, pyramid, etc.) and a text editor. These extra features will help programmers get a quick start on creating their shader programs.

There are two ways to switch to the Shader Assistant perspective view. From the main menu, choose Window **->** Open Perspective **->** Shader Assistant, or in the C/C++ **Project Explorer** pane, right click and select **Develop Shader**. Using the table in the left pane **Preview Settings** tab, select items in the *Setting* column and configure project as well as header, shaders, attributes, etc.

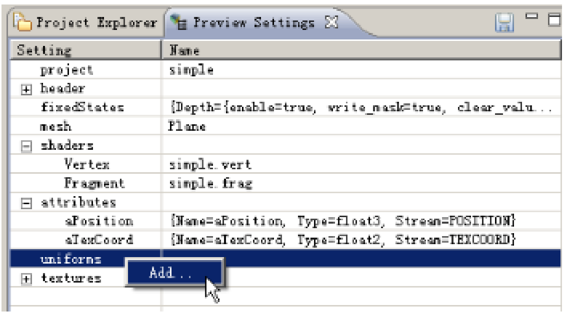


Figure 30. Shader Assistant

13.7.2 vTexture

Texture manipulation and viewing is available in four different areas of VivanteIDE:

- **Texture Editor** dialog boxes accessible from the Shader Assistant Preview Settings tab provides for texture customization, q.v. preceding Section 13.7.1 for launching Shader Assistant.
- **vTexture** Browser and Viewer panes are available from the main menu **Window -> Open Perspective -> VTexture**. It provides thumbnail and detail view of textures as well as the basic properties of the textures, such as image size and color depth.

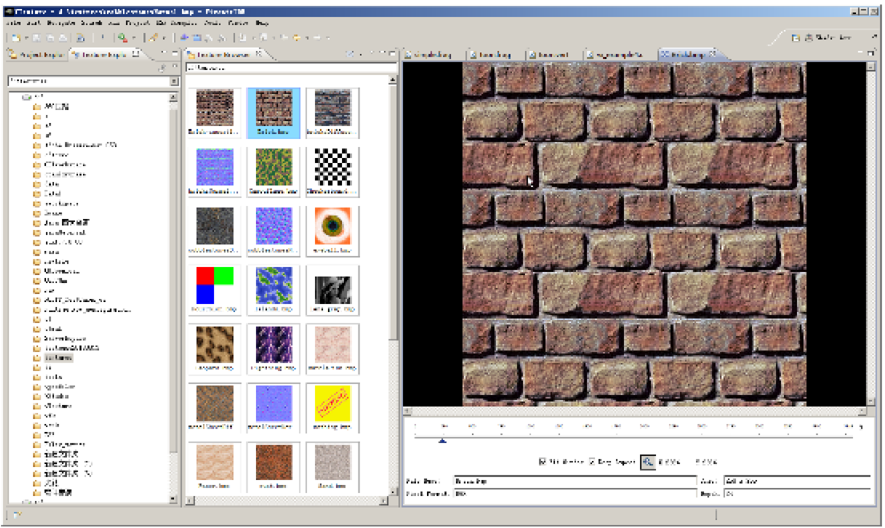


Figure 31. vTexture (1)

- Convert Texture provides a GUI for texture compression/decompression and tiling/de-tiling. It is accessible by clicking on the main menu **Tools->Convert Texture**. Note that **vTextureTools** is the command line tool version of this tool. Refer to Section 13.8.4 for details.

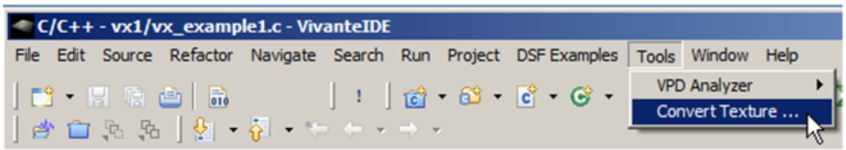


Figure 32. vTexture (2)

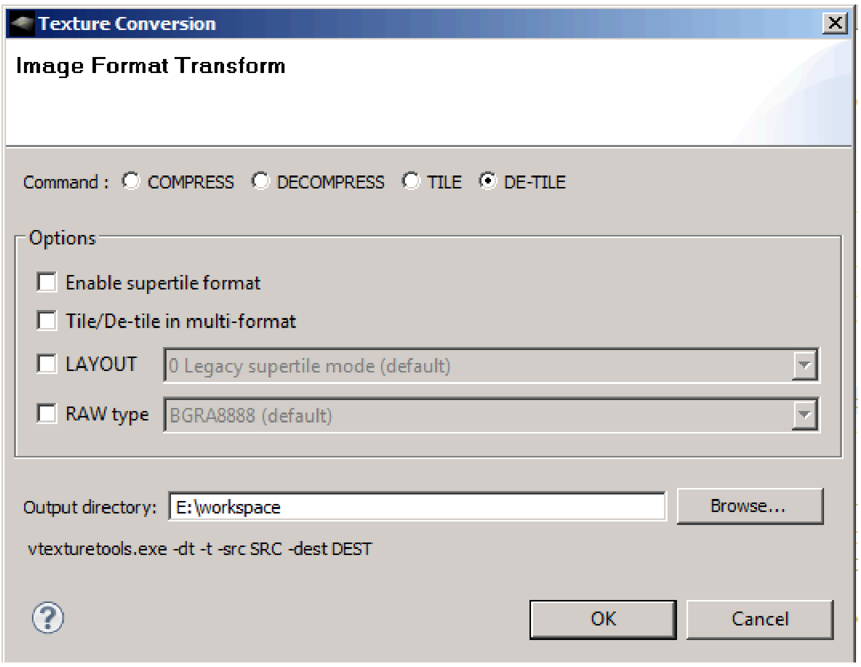


Figure 33. vTexture (3)

13.8 VivanteIDE command line tools

For easy reference, the syntax for the VivanteIDE command line tools are provided on the following pages. You can also refer to the *VivanteIDE User Guide* or inline `-h` (help) for syntax for these command line tools.

13.8.1 Preparing the environment

Before running command line tools, prepare the environment as in the examples below.

For Linux OS

- Launch a BASH
- `$ source installation_dir/ide/setenv-vivanteide<version>` # initialize the environment

For Windows OS

- Launch a Command Shell
- `> installation_dir/ide/setenv-vivanteide<version>.bat` # initialize the environment

13.8.2 vCompiler Command Line Syntax for OGL and OGLES

Open a Command prompt. Navigate to the folder, which contains the vTextureTools files (for example, `installation_dir/cmdtools/vCompiler`, and launch the `vCompiler` application executable using the command line syntax described below.

Make sure the configuration file is customized for your target environment.

13.8.2.1 Syntax

Windows and Linux command line syntax is the same.

Optional inputs are indicated by brackets. A fixed order for options in the command is not required.

```
vCompiler [-f <gpuConfigurationFile>] <shaderInputFileName>
 [shaderInputFileName_2]
[ -c ] [-h] [ -l ] [-o <outputFileName> ] [-On ] [ -v ] [-x
 <shaderType> ]
```

13.8.2.2 Input parameters (required)

shaderInoutFileName	shader input file name, which **must** contain one of the following file extensions: • vert: vertex shader source file • frag: fragment shader source file • vgcSL: previously compiled vertex shader input/output file • pgcSL: previously compiled pixel shader input/output file

13.8.2.3 Input parameters (optional)

<table>
<tr><td>shaderInputFileName_2</td><td>Up to two shader files can be specified. The second shader file is optional but must have one of the file extensions described above for shader InputFileName. If the first shader is a vertex shader, this second shader should be a fragment shader; conversely if the first shader is a fragment shader, the second should be a vertex shader.

Note: Pre-compiled and compiled shaders may be mixed, as long as one is a vertex shader and the other a fragment shader.</td></tr>
<tr><td>-c</td><td>Compile each vertex .vert file into a vgcSL file and/or fragment shader .frag file into a pgcSL only, with no merged result file of type .gcPGM.

If the –c option is not specified:
• When only one shader is specified, that shader will be compiled into a <code>.[v/p]gcSL</code> file.
• When two shaders are specified, one is assumed to be a vertex shader and the other a fragment shader. Each shader can be either a previously compiled .vgcSL or .pgcSL. file or a .vert or .frag still to be compiled. The two will be merged into a .gcPGM file after successful compilation.</td></tr>
<tr><td>-f <gpuConfigurationFile></td><td>Specifies a configuration file (from VTK 1.6.2). If –f is not specified, the file viv_gpu.config in the vCompiler working directory will be used as the default configuration file. Example syntax:

<code>vCompiler –f viv_gpu_880.config foo.vert bar.frag</code>

Note: vCompiler will not work correctly if the GPU configuration file cannot be found or contains incorrect content.</td></tr>
<tr><td>-h</td><td>Shows a help message on all the command options.</td></tr>
<tr><td>-l</td><td>Create a log file. The log file name is created by taking the first input file name, then replacing its file extension with “.log”. If the input file name does not have a file extension, .log is appended, e.g.,

<code>myvert.vert   => myvert.log</code>
<code>inputfrag   => inputfrag.log</code></td></tr>
<tr><td>-o <outputFileName></td><td>Specify the output file name. If the path is other than the current directory, it must also be specified. Any extension</td></tr>
</table>

	can be specified. If the extension is not specified, the **outputFileName** supported default types are as follows: • vgcSL: compiled vertex shader output file, usually compiled from a .vert input source file (default result for single file compile) • pgcSL: compiled pixel shader output file, usually compiled from a .frag source input file. • gcPGM: compiled file merging vertex shader and fragment/ pixel shader into a single output file
-O<*n*>	Optimization level. Default is **–O2**: • **-O0**: Disable optimizations • **-O1**: Some optimizations are enabled. • **-O2** All optimization levels are on (default).
-v	Verbose; prints compiler version and diagnostic messages to STDOUT.
-x<shaderType>	Explicitly specifies the type of shader instead of relying on the file extension. This option applies to all following input files until the next **-x** option. **ShaderType**: supported values for Shader type include: • **vert**: vertex shader source file • **frag**: fragment shader source file • **vgcSL**: compiled vertex shader input/output file • **pgcSL**: compiled pixel shader input/output file
-x none	Revert back to recognizing shader type according to the file name extension.

13.8.2.4 vCompilerOutput

Output files are placed in the current directory, unless another directory is specified with the `-o` option. The files can be of the three types described above under outputFileName value of the `-o` option.

13.8.2.5 vCompiler Syntax examples

`vCompiler foo.vert` produces `foo. vgcSL`.

`vCompiler bar.frag` produces `bar.pgcSL`.

`vCompiler foo.vert bar.frag` produces `foo.gcPGM`.

`vCompiler -v -l -O1 foo.ver tbar.frag` produces `foo.gcPGM` and `foo.log`.

`vCompiler -v -l -O1 -o foo_bar foo.vert bar.frag` produces `foo_bar.gcPGM` and `foo_bar.log`.

13.8.3 vcCompiler Command Line Syntax for OCL

Open a Command prompt. Navigate to the folder which contains the vTextureTools files (for example, `installation_dir/cmdtools/vCompiler`, and launch the `vCompiler` application executable using the command line syntax described below.

Make sure the configuration file is customized for your target environment.

13.8.3.1 Syntax

Windows and Linux command line syntax is the same.

Optional inputs are indicated by brackets. A fixed order for options in the command is not required.

```
vcCompiler [-f <gpuConfigurationFile>] [-v] [-l] [-O0] [-D
 <MacroDefinition>] [-I <IncludeDirectory>]
[-K <KernelName>] [-M] [-B] <OpenCLOrOpenVXFileName>
 <OpenCLOrOpenVXFileName_2> . . . [-allkernel]
```

13.8.3.2 Input parameters (required)

OpenCLOrOpenVXFileName	Input file name, which **must** contain one of the following file extensions: • cl: OpenCL source file • vx: OpenVX Vision source file If an input file extension is not specified, vcCompiler will report a “wrong file extension” error.

13.8.3.3 Input parameters (optional)

OpenCLOrOpenVXFileName_2, _n	Multiple input files can be specified. The second and additional files are optional but must have the appropriate file extension as described above. All files must be of the same type (.cl or .vx).
-allkernel	Allows VX applications to create all kernels in one program and save them into one package.
-B	Support source level intrinsic built-in functions.
-D <MacroDefinition>	Predefined inline macro, as referenced in the input file.
-f <gpuConfigurationFile>	Specifies a configuration file. If **–f** is not specified, the file **viv_gpu.config** in the vcCompiler working directory will be used as the default configuration file. Syntax example: `vcCompiler –f viv_gpu_gc7000.config foo.cl`

	Note: *vcCompiler will not work correctly if the GPU configuration file cannot be found or contains incorrect content.*
-h	Shows a help message on all the command options.
-I <IncludeDirectory>	Specify the directory path for include files.
-K <KernelName>	Link with kernel name. Default is **main**.
-l	Create a log file. The log file name is created by taking the input file name, then replacing its file extension with ".log". If there are multiple input files, the filename of the first input file will be used, `inputcl.cl      => inputcl.log` `myvx1.vx myvx2.vx    => myvx1.log`
-M	Merge all compiled output from each file into one file. The combined output will have the name of the last input file combined with the output extension .gcPGM.
-O<*n*>	Optimization level. Default is **–O2**: • **-O0**: Disable optimizations • **-O1**: Some optimizations are enabled. • **-O2** All optimization levels are on (default).
-v	Verbose; prints compiler version and diagnostic messages to STDOUT

13.8.3.4 vcCompiler Output

Output files are placed in the current directory. When compiled successfully, the supported output file extensions for vcCompiler are:

- `.clgcSL`: compiled CL output file, compiled from a `.cl` input source file.
- `.vxgcSL`: compiled VX output file, compiled from a `.vx` input source file.

13.8.3.5 vcCompiler Syntax Examples

```
vcCompiler [-f <gpuConfigurationFile>] [-v] [-l] [-O0] [-D
 <MacroDefinition>] [-I <IncludeDirectory>]
[-K <KernelName>] [-M] [-B] <OpenCLOrOpenVXFileName>
 <OpenCLOrOpenVXFileName_2> [-allkernel] . . .
```

`vcCompiler -v -O1 foo.cl`: produces `foo.clgcSL`.

`vcCompiler -v -l foo.vx`: produces `foo.vxgcSL` and `foo.log`.

13.8.4 vTextureTools command line tool

Open a Command prompt. Navigate to the folder which contains the vTextureTools files, for example, `installation_dir/cmdtools/vTextureTools`, and launch the `vTextureTools` application executable using the command line syntax described below.

13.8.4.1 Syntax

The usage of the command line tool is as follows for compression/decompression:

```
vTextureTools -c TYPE [-s SPEED] –src FILE [-dest FILE]
```

or

```
vTextureTools -d TYPE –src FILE [-dest FILE]
```

The usage of the command line tool is as follows for tiling/de-tiling:

```
vTextureTools -t|-st [-2] [-r|--raw=FORMAT] [-m LAYOUT] –src
 FILE [-dest FILE]
```

or

```
vTextureTools -dt -t|-st [-2] [-r|--raw=FORMAT] [-m LAYOUT] –
src FILE [-dest FILE]
```

13.8.4.2 General parameters

General parameters:

- `-h` show help
- `-src [FILE]` source file - input image path and filename. vTexture will use the file extension type as image type.
  - For option `–c` compress, the application expects an input filename with a .TGA extension.
  - For `–d` decompression, the application expects .DDS, .KTX or .PKM.
  - For `–t` tile, the application expects .BMP or .TGA.
  - For `–dt` detile, the application expects .BMP or .TGA.
- `-dest [FILE]` destination file - image path and filename.
  - The application expects a filename with a .TGA, .DDS, .KTX or .PKM extension for compress/uncompress or .BMP or .RAW for tile/detile.
  - If the `–dest` parameter is not set, vTexture will auto generate a name for the newly generated file, using the source file name as the prefix appending critical parameters and file type information.

13.8.4.3 Compression/Decompression parameters

These parameters are used for compression and decompression:

- **-c** compress a source image of format uncompressed TGA
- **[TYPE]** specify the target output compression format:
- **-DXT1** compress image to DXT1 format (default format).
- **-DXT3** compress image to DXT3 format.

- **-DXT5** compress image to DXT5 format.
- **-ETC1** compress image to ETC1 format
- **-ETC2** compress image to ETC2 format
- **-d** decompress a source image of format specified by the value **[TYPE]**.
  The resulting file type will be uncompressed TGA.
  This option decompresses DXT1, DXT3, DXT5, ECT1 or ETC2 format image to TGA format.
- **-s** compression **[SPEED]** mode for ETCn images:
  – slow
  – medium
  – fast (default)

13.8.4.4 Tile/De-Tile parameters

The parameters listed in the following table are used for tiling and de-tiling between linear and tiled formats.

Table 39. Tile/De-Tile parameters

`-t`	Convert linear data to tiled texture output.
`-st`	Enable supertile format. This option is an alternate to `-t`. If `-st` and `-t` are used together, `-st` will be set.
`-dt`	De-tile: Convert tiled texture to linear texture output.
`-2`	Tile/de-tile in multi-format. Tile format is multi-tiled (when used with `-t`) or multi-supertiled (with `-st`).
`-m`	`[LAYOUT]`: layout mode for supertiled or multi-supertiled textures: • `0`: Legacy supertile mode (default). • `1`: Supertile mode when hardware has HZ. • `2`: Supertile mode when hardware has NEW_HZ or FAST_MSAA.
`-r`	Specify output data as raw pixel output instead of BMP. Use `--raw=rgb565` to specify raw pixel `[FORMAT]`. Supported raw formats (8) are: `rgba8888, bgra8888, rgb888, bgr888, rgb565, bgr565, argb1555, yuy2`

13.8.4.5 vTexture Syntax Examples

COMPRESS:

```
vTextureTools -c dxt1 -src d:\myfile.png -dest c:\compress.dds
vTextureTools -c dxt1 -src d:\myfile.tga -dest c:\compress.dds
vTextureTools -c etc1 -s slow -src d:\myfile.png -dest c:
\compress.pkm
vTextureTools -c etc1 -s slow -src d:\myfile.tga -dest c:
\compress.pkm
vTextureTools -c etc2 -s slow -src d:\myfile.bmp -dest c:
\compress.ktx
vTextureTools -c etc2 -s slow -src d:\myfile.tga -dest c:
\compress.ktx
vTextureTools -c etc2 -src d:\myfile.bmp -dest c:\compress.ktx
vTextureTools -c etc2 -src d:\myfile.tga -dest c:\compress.ktx
vTextureTools -c etc2 -src d:\myfile.tga -dest c:\compress.pkm
```

DECOMPRESS:

```
vTextureTools -d etc1 –src c:/vtexin/myfile2.pkm –dest c:/
vtextout/myfile2.tga
vTextureTools -d –src c:/vtexin/myfile3.dds –dest c:/vtextout/
myfile3.tga (assumes DXT1)
vTextureTools -d tga -src d:\myfile.dds -dest c:\decompress.tga
vTextureTools –d tga -src d:\myfile.ktx -dest c:\decompress.tga
```

TILE: LINEAR TO TILE CONVERSION:

- Tile linear texture to standard tile texturev

```
TextureTools.exe -t -src 123.bmp
```

- Tile linear texture to multi-tiled texture

```
vTextureTools.exe -t -2 -src 123.bmp
```

- Tile linear texture to supertiled texture

```
vTextureTools.exe -st -src 123.bmp
```

- Tile linear texture to multi-supertiled texture

```
vTextureTools.exe -2 –st -src 123.bmp
```

- Tile linear texture to multi-supertiled texture and output rgb565

```
vTextureTools.exe -2 --raw=rgb565 -src 123.bmp
```

- Tile linear texture to multi-supertiled texture with layout mode 2

```
vTextureTools.exe -st -2 -m 2 -src 123.bmp
```

DE-TILE: TILED TO LINEAR CONVERSION:

- De-tile tiled texture to linear texture

```
vTextureTools.exe –dt -t -src 123-tiled.bmp
```

- De-tile supertiled texture to linear texture

```
vTextureTools.exe -dt -st -src 123-supertiled.bmp
```

- De-tile multi-supertiled texture to linear texture

```
vTextureTools.exe –dt -t -2 -src 123-tiled-multi-tiled.bmp
```

- De-tile multi-Super-tiled texture with layout mode 2 to linear texture

```
vTextureTools.exe -dt -st -2 -m 2 -src 123-multi-
supertiled-2.bmp
```

14 GPU Tools

14.1 gpuinfo tool

14.1.1 Introduction

gpuinfo is a script to gather GPU runtime status through debugfs interface. It exports the following information:

- GPU hardware information.
- GPU total memory usage.
- GPU memory usage of certain process or all processes (user space only).
- GPU idle percentage.

14.1.2 Usage

The script is located at Yocto `rootfs /unit_tests/`. There are three ways to run it.

- Normal run to get all GPU-related processes information:

```
>/unit_tests/GPU/gpuinfo.sh
```

- Get GPU information for certain process by clarifying the process id.
  The process id (pid) can be got by command ps or top. Take the process 1035 as example.

```
>/unit_tests/GPU/gpuinfo.sh 1035
```

- Get the GPU information for certain process by clarifying part of process name.
  Take the process sample_test_fbo as an example.

```
>/unit_tests/GPU/gpuinfo.sh sample_test_fbo
```

or

```
>/unit_tests/GPU/gpuinfo.sh sample
```

or

```
>/unit_tests/GPU/gpuinfo.sh test
```

14.1.3 Sample log information

14.1.3.1 GPU hardware information

This section shows all GPU cores model name and revision information with index in the SoC.

The sample information:

```
GPU Info
gpu      : 0
model    : 2000
revision : 5108
gpu      : 1
model    :  320
revision : 5007
gpu      : 2
model    :  355
```

14.1.3.2 Total memory information

This part shows total GPU memory information.

Table 40. Total memory information

gcvPOOL_SYSTEM:	GPU reserved system memory.
gcvPOOL_CONTIGUOUS:	contiguous memory allocated from CMA pool, low memory zone and high memory zone.
gcvPOOL_VIRTUAL:	non-contigous memory allocated from low memory zone and high memory zone.
NON PAGED MEMORY:	Allocated from CMA pool(mainly for command buffer)

The sample information:

```
VIDEO MEMORY:
    gcvPOOL_SYSTEM:
        Free  :  124170474 B
        Used  :   10047254 B
        Total :  134217728 B
    gcvPOOL_CONTIGUOUS:
        Used  :          0 B
    gcvPOOL_VIRTUAL:
        Used  :          0 B
NON PAGED MEMORY:
    Used  :          0 B
Paged memory Info
low:  892928 bytes
high: 0 bytes
CMA memory info
cma:  0 bytes
```

14.1.3.3 Process user space GPU memory usage information

This part shows detail user space GPU memory usage per process.

Table 41. User space GPU memory usage

Index	memory for index buffer.
Vertex	memory for vertex data buffer.
Texture	memory for texture buffer.
RT	memory for render target buffer.
Depth	memory for depth buffer.
Bitmap	memory for bitmap buffer.
TS	memory for tile status buffer.
Image	memory for vg image buffer.
Mask	memory for vg mask buffer.
Scissor	memory for vg scissor buffer.
HZDepth	memory for hierarchical Z depth buffer.

The sample information:

```
VidMem Usage (Process 1106):
Counter: vidMem (for each surface type)
All      Index    Vertex   Texture      RT      Depth    Bitmap
   TS    Image   Mask  Scissor  HZDepth
```

```
Current  10047254  489362  1213248   435200   3866624   3727360
    0   36352    0      0    0   245760
Maximum  10047254  489362  1213248   435200   3866624   3727360
    0   36352    0      0    0   245760
Total    10047254  489362  1213248   435200   3866624   3727360
    0   36352    0      0    0   245760
Counter: vidMem (for each pool) All   1      2     3      4
  5      6      7        8        9
Current  10047254        0         0        0         0
  0  10047254   0        0         0
Maximum  10047254        0         0        0         0
  0  10047254   0        0         0
Total    10047254        0         0        0         0
  0  10047254   0        0         0
Counter: nonPaged
                All
Current           0
Maximum           0
Total             0
Counter: contiguous
                All
Current           0
Maximum           0
Total             0
Counter: mapUserMemory
                All
Current           0
Maximum           0
Total             0
Counter: mapMemory
                All
Current   134217728
Maximum   134217728
Total     134217728
```

14.1.3.4 GPU idle percentage

This part shows GPU idle percentage in past 1s.

The sample information:

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Idle percentage:0.00%
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

14.2 gputop tool

gputop -- Monitor GPU clients memory, hardware counters, occupancy state load on DMA engines, video memory and and DDR memory bandwidth (only under Linux).

- The gputop tool is developed to trace the overall memory utilization in classification of memory pools.
- The available memory size is reported for the reserved pool.
- GPU idle time is reported from the last capture.

14.2.1 Synopsis

gputop [options]

gputop -m [mode] -- Where mode can be: mem, counter_1, counter_2, occupancy, dma, vidmem and ddr (under Linux/Android). Use this option to start gputop directly in a mode that you're interested on. For counter_1 and counter_2 a context will be needed. See *NOTES* section why this is necessary.

gputop -c ctx_no -- specify a context to attach when display context-aware hardware counters.

gputop -b -- display in batch mode. For other modes than memory, this will only take an instantaneous sample. See -f

gputop -f -- Use this when using gputop from a script.

gputop -x -- useful to display contexts when used with ``-b''

gputop -i -- ignore warnings about kernel mismatch

gputop -h -- display usage and help

14.2.2 Interactive mode

Normally, when starting up, gputop, starts in interactive mode. The following are a list of useful commands:

- 'h' -- display help page
- '0-6'/Left-Right arrows -- switch between viewing pages
- 'x' -- display application contexts
- 'SPACE' -- select a context that you want to track. Useful for reading counter_1 and counter_2 values.
- 'r' -- useful for hardware-counter pages to display different viewing modes (switches between different modes of aggregation: MIN/MAX/AVERAGE/TIME)
- 'q'/ESC -- exits gputop.
- 'p' -- stops reading counter values and displays only current values. Useful to get a instantaneous values of the counters.

14.2.3 Description

gputop can be used to determine the memory usage your application is using, or to read the hardware counters exposed by the GPU in real-time. Additionally, DMA engines and Occupancy states are displayed. gputop has multiple viewing pages: a memory usage page, two hardware counter pages, a DMA engine page and an Occupancy page. When normally started, gputop will be in interactive mode. Type 'h' to get a list of the current keybindings.

14.2.4 Requirements

14.2.4.1 Linux OS

gputop requires access to debugfs sub-system on Linux to display memory usage, used by clients submitting commands to the GPU. gputop will try to mount the debugfs pseudo-filesystem if it is not already mounted. In order to read hardware counters the profiler must be activated in the driver. Usually this can be set by setting the environment variable export VIV_PROFILE=1.

14.2.4.2 QNX

Just like in Linux OS, to read the hardware counter values, gpu-gpuProfiler has to be set to 1 in `graphics.conf` file under `$GRAPHICS_ROOT` directory. Other views like occupancy and DMA will require gpu-powerManagement to be set to 0 (disabled).

14.2.5 Notes

14.2.5.1 Sampling hardware-counters

gputop samples the driver for hardware counter values. Internally the driver will update the values of the counters whenever the application submits a special type of command to the GPU. Depending on how fast that happens gputop can't foresee/adjust the values of the counters. So tweaking the amount of sample taken or the delay time doesn't really help. For dealing with situations where the application will submit either to fast or to low commands to the GPU, several modes of viewing counters has been added. Cycle between them to understand or get a bird-eye view of the counter values. Empirically MAX/AVERAGE displays the closes values to the truth.

14.2.5.2 Context-aware counters

counter_1 and counter_2 are context-aware counters (i.e.: tied to an application).

Internally the driver assigns various context IDs to the application submitting commands to the GPU. These contexts IDs are currently required to read those hardware counter values. Either use -x on the command line (together with -b option and choosing -m mem viewing mode), or for interactive mode use 'x' and then 'SPACE' to show and select a context ID.

In case you are getting zero'ed out values for counter_1 and/or counter_2 values, cycle through the available counter IDs.

Due to the way the driver is built, single-GPU core applications will have two context-ids. Empirically the largest integer values holds the real context-id.

14.2.5.3 Unsupported GPUs

For GCV600 (i.MX 7ULP and i.MX 8M Mini), the IDLE/LOAD register is not available, so gputop will display incorrect (inversed) values.

14.2.6 Pages

14.2.6.1 Client attached page

When viewing client attached page, the following head columns are displayed:

PID RES (KB) CONT (KB) VIRT (KB) Non-PGD (KB) Total (KB) CMD

- PID -- process ID
- RES -- reserved memory
- CONT -- contiguous memory
- VIRT -- virtual memory

- Non-PGD -- Non-paged memory
- Total -- the sum of all above
- CMD -- the name of the application (trimmed)

These memory items correspond to memory pools in the driver.

14.2.6.2 Vidmem page

When viewing vidmem page, the following head columns are displayed for each process.

PID IN VE TE RT DE BM TS IM MA SC HZ IC TD FE TFB

- IN -- index
- VE -- vertex
- TE -- texture
- RT -- render target
- DE -- depth
- BM -- bitmap
- TS -- tile status
- IM -- image
- MA -- mask
- SC -- scissor
- HZ -- hz
- IC -- i_cache
- TD -- tx_desc
- FE -- fence
- TFB -- tfb header

14.2.7 Examples

When using `-b` option, gputop will start in interactive mode and execute just once its main loop. This is useful for various reason, either to get an instantaneous view of a different viewing page, or scripting.

- Get a list of processes attached to the GPU.

```
$ gputop -m mem -b
```

- Get a list of processes attached to the GPU, but also display the contexts IDs.

```
$ gputop -m mem -bx
```

- Display counters (counter_1) using context_id.

```
$ gputop -m counter_1 -b -c <context_id>
```

- Display counters (counter_2) using context_id.

```
$ gputop -m counter_2 -b -c <context_id>
```

- Get IDLE/USAGE

```
$ gputop -m occupancy -b | grep IDLE
```

14.2.8 See Also

- Under QNX, see `graphics.conf` for disabling powerManagement and enabling gpuProfiler.
- Under Linux, see `/sys/modules/galcore/paramenters/gpuProfiler` and `/sys/modules/galcore/parameters/PowerManagement`.

14.3 GPU clock information and debugging

GPU driver supports dynamic frequency scaling. Users can perform the following steps to query and update the GPU clock information, which is useful for GPU debugging.

1. Get the GPU clock. This is affected by the system RTC timer. Sometimes it varies between different boards.

```
root@imx8mpevk:/# mount -t debugfs none /sys/kernel/debug
 (optional, exec it only if there is no gc dir)
root@imx8mpevk:/# cat /sys/kernel/debug/gc/clk
gpu0 mc clock: 1000018036 HZ.
gpu0 sh clock: 1000021374 HZ.
gpu1 mc clock: 1000002214 HZ.
gpu1 sh clock: 999986723 HZ.
gpu8 mc clock: 499991523 HZ.
```

2. Change the GPU clock.
Read the `gpu3DClockScale` as the denominator using the following command:

```
root@imx8mpevk:/# cat /sys/bus/platform/drivers/galcore/
gpu3DClockScale
64
```

The GPU frequency can be changed to `numerator/gpu3DClockScale * clock` for different GPU instances. For example, the gpu0's mc and sh clock can be change to 1/2 and 1/4 of the original frequency.

```
root@imx8mpevk:/# echo 0 32 16 > /sys/kernel/debug/gc/clk
[ 2625.977856] Change core:0 MC scale:32 SH scale:16
[ 2625.982610] Warning: Power management status will be
 changed forever!
root@imx8mpevk:/# cat /sys/kernel/debug/gc/clk
gpu0 mc clock: 499997481 HZ.
gpu0 sh clock: 249997541 HZ.
gpu1 mc clock: 999995540 HZ.
gpu1 sh clock: 999992141 HZ.
gpu8 mc clock: 499998453 HZ.
```

14.4 Apitrace user guide

14.4.1 Introduction

Apitrace is a set of tools enhanced from open source project apitrace, supported by i.MX 6, i.MX 7, and i.MX 8 with Vivante GPU IP. This tool can dump OpenGL/GLES1.1/GLES2.0/GLES3.0 API calls and replay on a wide range of other devices.

For more information, see apitrace.github.io/.

14.4.2 Install

14.4.2.1 Yocto

Apitrace source code release is part of the i.MX Yocto Project Linux BSP release. The source code have more patches added on top of official Apitrace release. The Yocto Project recipes pull the Apitrace source package and install as needed for supported backend.

14.4.2.2 PC

Apitrace have set of PC tools. Prebuilt binary packages can be directly downloaded from the Apitrace website.

Currently supports Ubuntu 14.04 LTS, 64-bit.

```
sudo apt-get install libgles1-mesa libgles2-mesa libqt4-dev
```

14.4.3 Usage

14.4.3.1 Trace OpenGL ES1.1/2.0/3.0 application

```
apitrace trace --api=egl <app name and arguments>
```

e.g., `apitrace trace --api=egl es2gears_x11`

It generates trace file (.trace) under the current directory. To specify a new path, use `--output=<path_name>`.

14.4.3.2 Trace OpenGL ES 1.1/2.0/3.0 Java application on the Android platform

On the Android platform, a GLES application can be native (e.g., frameworks/native/opengl/angeles). This type of application can be traced as normal Linux application. Some other applications involving the Java virtual machine cannot run in this way. A script `apitrace_dalvik.sh` is provided to run this type of application. This is an example to trace `com.android.settings`:

```
sh /data/apitrace/bin/apitrace_dalvik.sh com.android.settings
 start
```

To stop tracing, run:

```
sh /data/apitrace/bin/apitrace_dalvik.sh com.android.settings
 stop
```

Because there is no “current” directory for a Java application, the trace file is stored under `/sdcard/`.

If Apitrace is installed in a different directory, update `apitrace_dalvik.sh` manually.

14.4.3.3 Trace OpenGL application

```
apitrace trace --api=glx <app name and arguments>
```

Only the X11 backend supports this feature.

14.4.3.4 Replay

This utility is also called retrace. It reads in the trace file and executes OpenGL (ES) APIs one by one. Each OpenGL (ES) API call is processed by a callback function. In that callback function, a hook can be inserted for debug or analysis purposes.

Figure 34. Replay

OpenGL ES 1.1/2.0/3.0 applications can be replayed with eglretrace; OpenGL applications can be replayed with **glretrace**:

```
eglretrace <trace file>
glretrace <trace file>
```

14.4.3.4.1 Analysis

qapitrace provides a detailed look at the trace file. It can only run on a PC. It was verified on Ubuntu 14.04 LTS 64-bit. The command is:

```
qapitrace <trace file name>
```

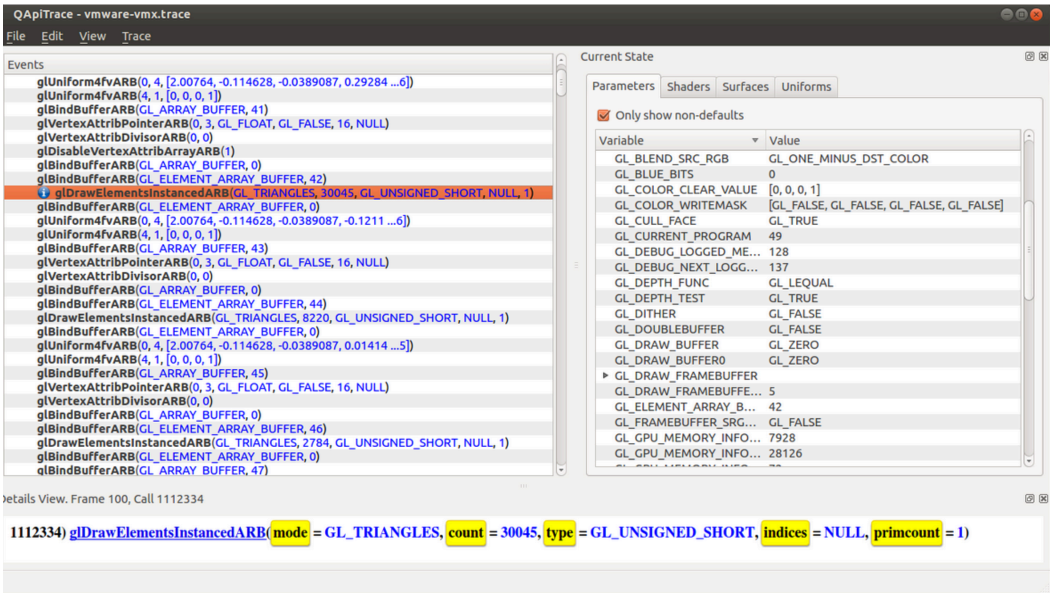


Figure 35. Checking state of every API call

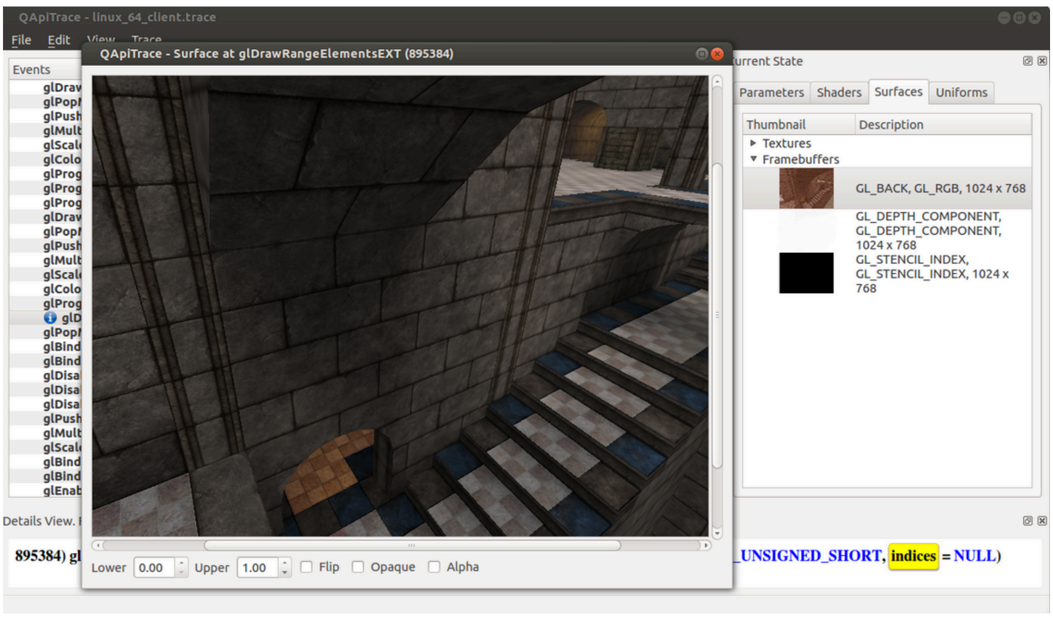


Figure 36. Checking Framebuffer

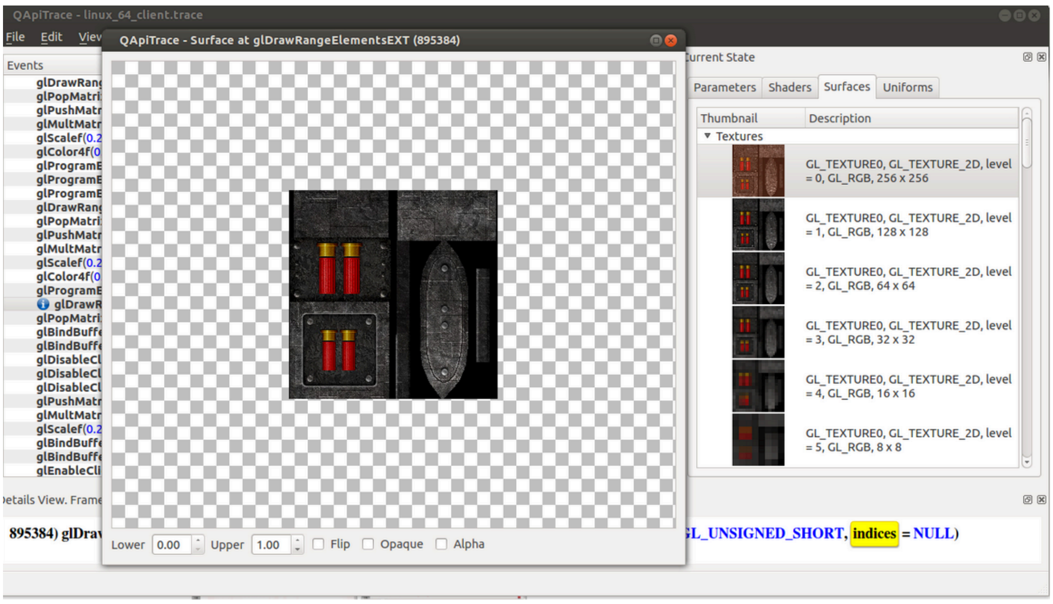


Figure 37. Checking Texture

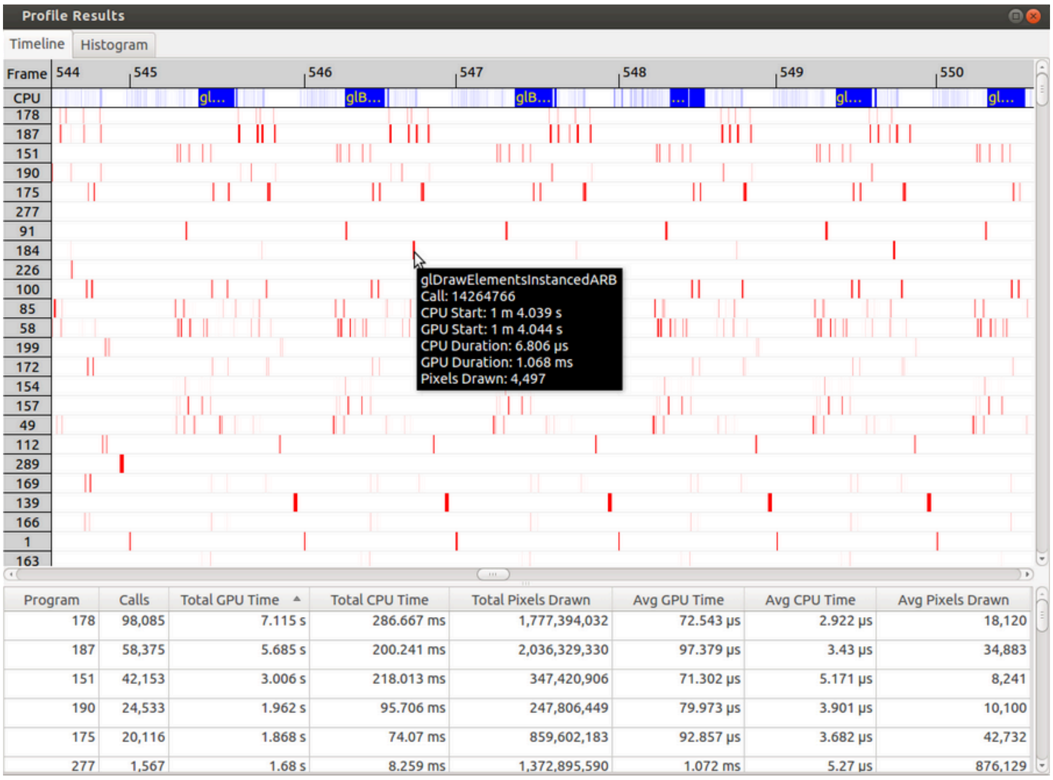


Figure 38. Checking performance

14.4.4 Reference

1. Apitrace introduction: apitrace.github.io/
2. More uses: github.com/apitrace/apitrace/blob/master/README.markdown

14.5 Renderdoc

Renderdoc™ is a frame-capture based graphics debugger, generally support for Vulkan, D3D11, D3D12, OpenGL, and OpenGL ES development. On i.MX, support available only for Vulkan. RenderDoc provides tools for deep analysis and graphics inspection, as well as detailed examination of API usage - allowing developers to locate bugs and problems in their programs.

14.5.1 Renderdoc components

Renderdoc source code release is part of the i.MX Yocto Project Linux BSP release. The source code has more patches added on top of the official Renderdoc release. The Yocto Project recipes pull the renderdoccmd tool source package and install it as needed for the supported backend. The version of renderdoccmd currently available for the user is 1.7.

Renderdoc has a set of PC tools. Prebuilt binary packages can be directly downloaded from Renderdoc website.

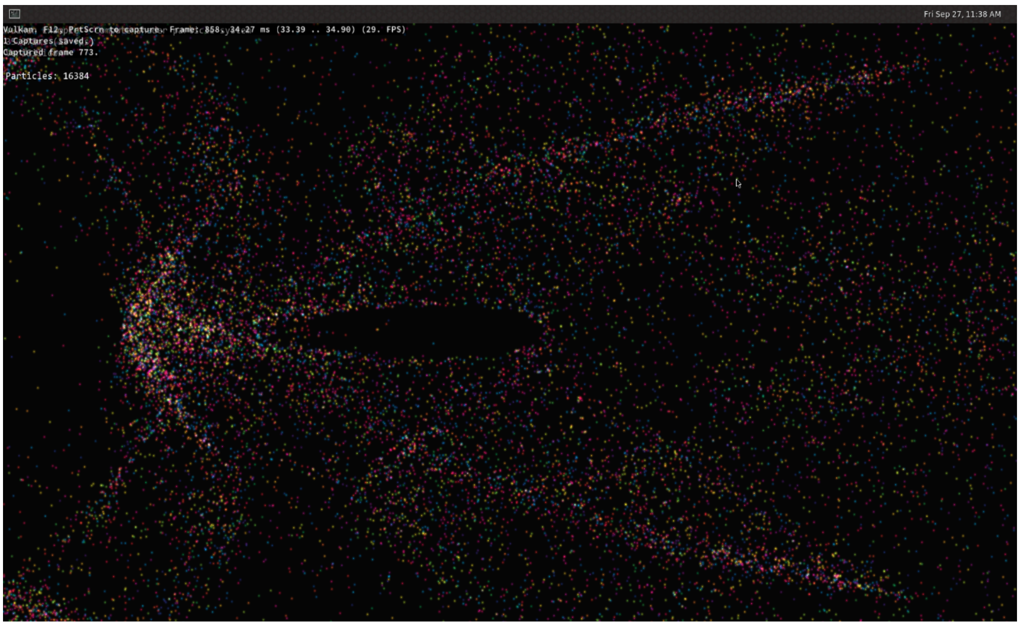
The renderdoccmd tool will be available on the i.MX board for capturing frames and replaying locally, as for debugging purposes qrenderdoc needs to be used remotely on a host machine.

14.5.2 Running renderdoccmd on i.MX

```
renderdoccmd capture <options> <app_name> <arguments>
```

Renderdoccmd usage example:

- For capturing a frame from a graphics application available in the SDK, run
  ```
  renderdoccmd capture /opt/imx-gpu-sdk/Vulkan/Some_example/
  Some_example_Wayland
  ```
- Press F12 to capture frames:

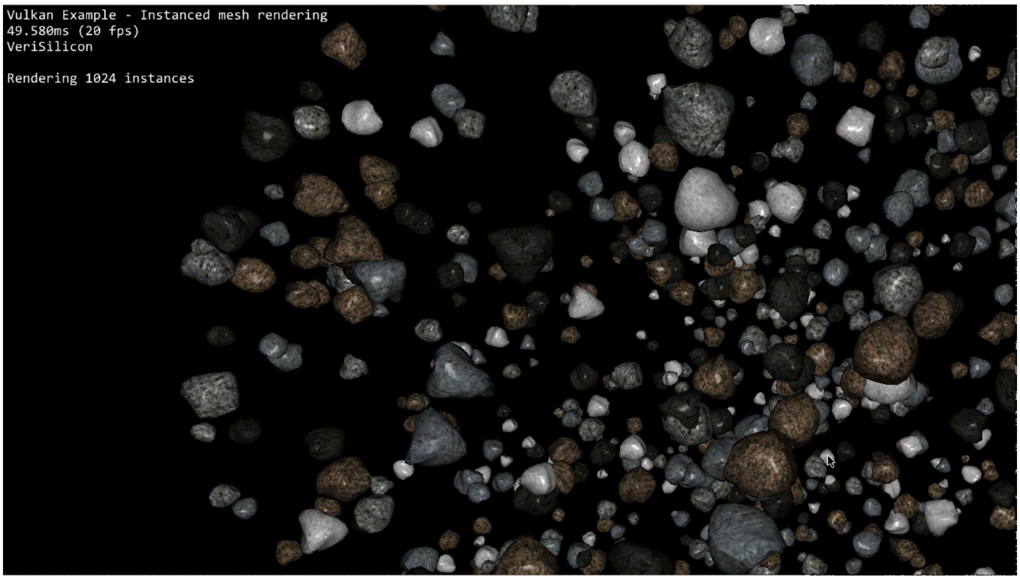


Frames will be written in `/tmp/Renderdoc`/ (run `renderdoccmd capture` to see all the options)

- For replaying a capture run

```
renderdoccmd replay /path/to/capture/file
```

(Run `renderdoccmd replay` for more options).

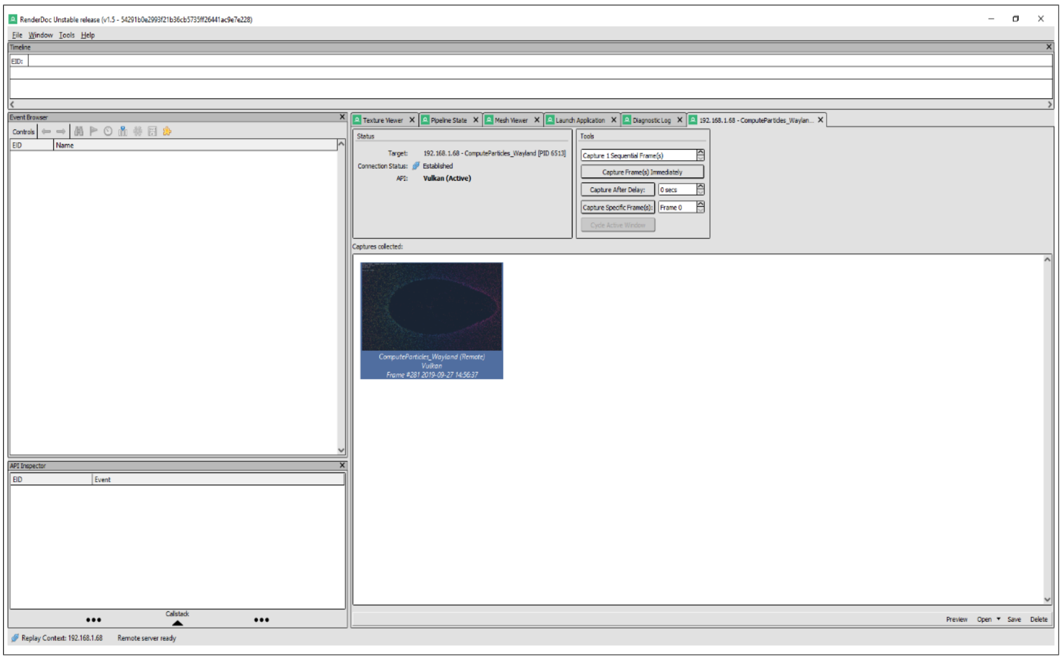


- Press F for full screen. Press F again to come back to the default window dimensions. Press ESC to quit replaying.

14.5.3 Capturing and replaying remotely

Usage:

1. Download a Renderdoc build from the website on your Windows/Linux host machine.
2. Set up a connection between the host and the board.
3. On the i.MX board, run `renderdoccmd remoteserver`.
4. On your machine, run `qrenderdoc`. Go to **File -> Attach to running instance**.
5. In the **Remote Host Manager Window**, add the target's IP address. Then qrenderdoc on your local machine should establish a connection with the renderdoccmd server instance.
6. In the left down corner of the screen, select **Replay Context** and change it from Local to the target's IP address.
7. Select **File -> Launch Application**. On **Executable Path**, insert the path of your Vulkan example from the target: /opt/imx-gpu-sdk/Vulkan/Some_example/Some_example_Wayland.
8. Press **Launch** and then capture. A new capture preview should appear.
9. You can save it by right clicking **Save** on the preview.



10. If you close the Vulkan application from the board, qrenderdoc will open the capture file.
11. To debug the capture, check the documentation available on the Renderdoc site.
12. To replay remotely, just use renderdoccmd on your local machine.
    Run `renderdoccmd replay --remote-host <target ip> <capture_file_on_you_local_machine>` and you should see exactly the same thing as when running on the target locally.

Notes for Android:

- Before starting the remote server and Vukan application, Android HWUI renderer must be set to Vulkan renderer. In Android console: `setprop debug.hwui.renderer skiavk`.
- Remote server on the Android platform is started from qrenderdoc application. Connect the board to PC through the USB-C port. In qrenderdoc, go to **Tools -> Manages Remote Servers**, and select the connected board. For example, “nxp MEK-MX8Q”, and press the **Run Server** button.
- On the Android platform, add permission "Allow access to manage all files" to RenderDocCmd when it is launched for the first time.
- Launch an application from qrenderdoc. Be sure the correct Replay Context is selected in the left bottom corner. Select a Vulkan application in the **Executable path** field from the **Launch Application** tab. Click the **Launch** button.
- Capture frame from qrenderdoc.
- Capture is replayed automatically on the Android platform when the Vulkan application is closed.

14.5.4 Reference

https://renderdoc.org/

https://github.com/baldurk/renderdoc/blob/v1.x/README.md

15 GPU Memory Introduction

15.1 GPU memory overview

- OpenGL-ES
  - Texture buffer
  - Vertex buffer
  - Index buffer
  - PBuffer surface
  - Color buffer
  - Z/Stencil buffer
  - HZ depth buffer
  - Tiled status buffer
  - 3D Command buffer
  - 3D Context buffer
- OpenVG
  - Image buffer
  - Tessellation buffer
  - VG command buffer
  - VG context buffer
- 2D buffers
  - 2D command buffer
  - 2D temporary buffer

15.2 GPU memory pools

- Reserved memory
  In the Linux 3.10.y kernel, the memory is reserved from CMA implemented in the GPU kernel driver, the size can be changed through U-Boot args with "galcore.contiguoussize =xxx"
  The memory allocation and lock very fast, but cannot support cacheable attribute.
- Contiguous memory
  The contiguous memory is from CMA or Normal or Highmem with alloc_pages_exact.
  The GPU driver tries the CMA allocator for non-cacheable request first. If CMA memory is used up, it goes to system allocator.
  The CMA allocator does not support the cacheable attribute, the system allocator supports cacheable attribute, but the memory performance is slow with the additional cache flush operations.
- Virtual memory pool
  The virtual memory is from Normal or Highmem with multiple page_alloc.
  The memory support cacheable attribute, but slow with GPU MMU and cache flush.
  The GPU virtual command buffer is allocated from virtual memory pool directly.
- Nonpaged memory pool
  In the 5.x GPU driver, this pool is not used any more.

15.3 GPU memory allocators

Two kinds of allocators are implemented in i.MX GPU kernel driver, see `drivers/mxc/gpu-viv/`.

- The video memory allocator implementation is very complicated. The memory is from the reserved pool, system contiguous pool (supports CMA), or system virtual pool (enables GPU MMU).
- The CMA allocator supports non-cacheable contiguous memory. It is implemented as a part of contiguous pool. When the system requests contiguous memory, the allocator tries CMA first. If CMA is used up, it goes to allocate the system contiguous pages.
- GPU memory-killer is implemented for special requirement of force contiguous GPU memory.

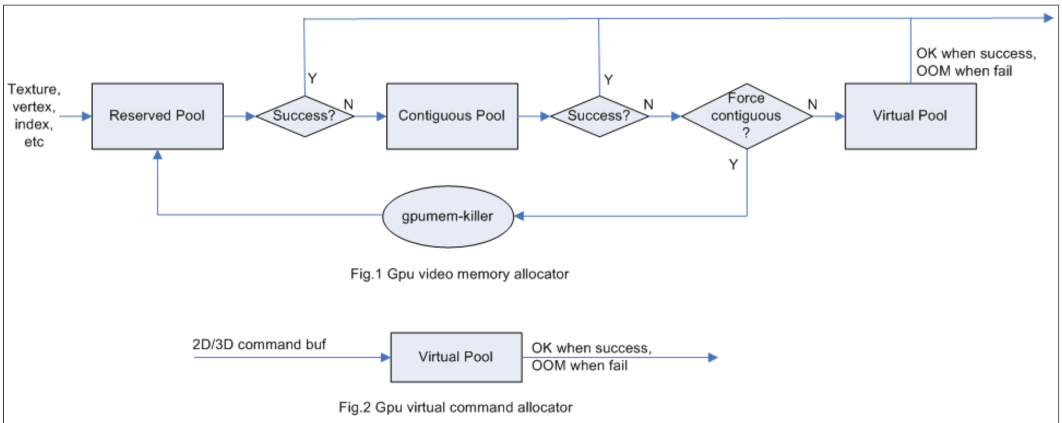


Figure 39. GPU memory allocators

15.4 GPU reserved memory

- The reserved memory is managed by two dual linked lists, one is free list, and another is node list.
- When allocate the reserved memory, the free list is scanned from head to tail until a available node is selected, it is very fast but makes more memory fragments, under test, 10~20M of 128M is not available to use after a lot of allocate/free operations.
- When the available node is selected, it is removed from the free list, but it always keeps the dual linked nodes to merge the conjoint available memory when freed.
- The reserved memory is mapped once when application process is attached, during 3D application running, the memory map/un-map operations are very fast, the virtual address is just calculated with logical base and offset.

15.5 GPU memory base address

- GPU support contiguous physical memory within (0-2G) address directly:
  – GPU address = CPU Physical address – GPU BaseAddress
- GPU MMU is enabled for two kinds of memory type as below:
  – Separated page memory from Virtual memory pool
  – Contiguous page memory with address out of (0-2G)
- BaseAddress should be set to RAM start address to achieve the better performance by reducing GPU MMU mapping.

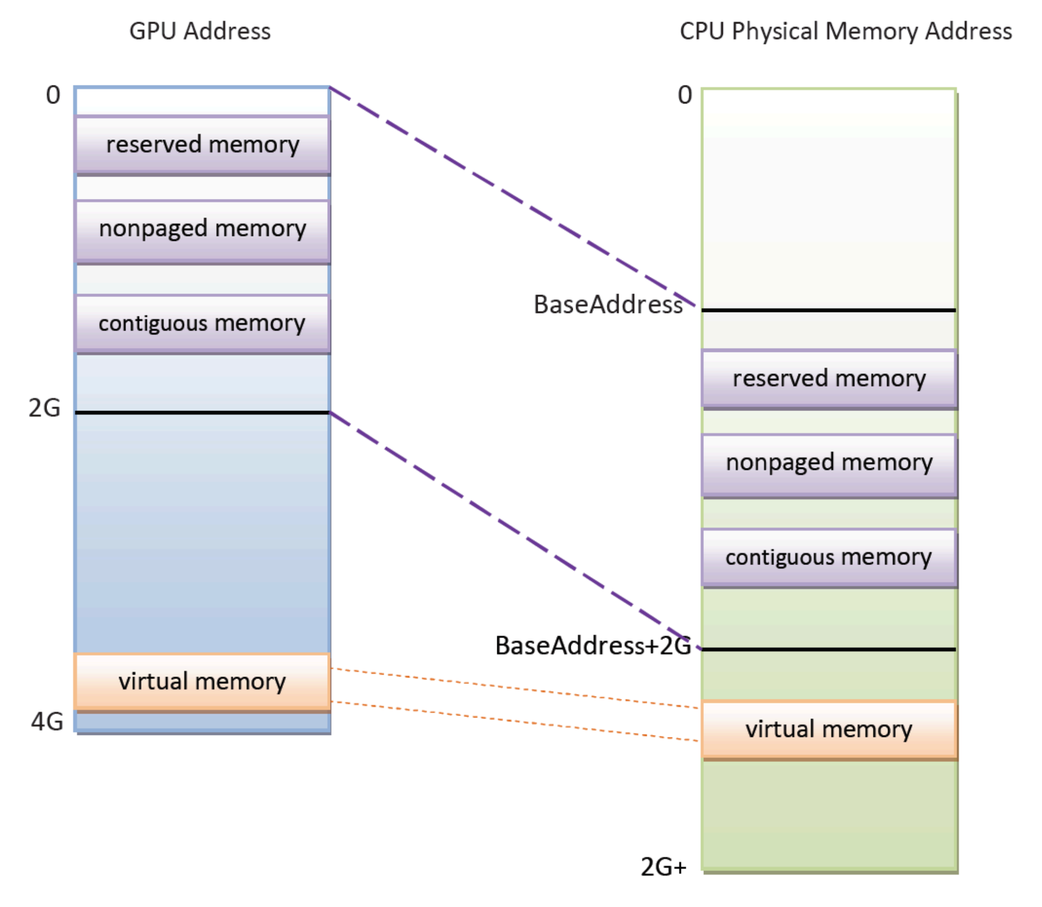


Figure 40. GPU memory base address

16 Application Programming Recommendations

The recommendations listed below take a holistic approach centered on overall system level optimizations that balance graphics and system resources.

16.1 Understanding the system configuration and target application

Knowing details about the application and use case allows developers to correctly utilize the hardware resources in an ideal access pattern. For example, an implementation for a 2D or 3D GUI could be rendered in a single pass instead of multiple passes if the draw call sequence is correctly ordered. In addition, knowing the most common graphics function calls allow developers to parallelize rendering to maximize performance.

Using Vivante and vendor-specific SoC profiling tools, you can determine bottlenecks in the GPU and CPU and make changes as needed. For example, in a 3D game, most CPU cycles may be spent on audio processing, AI, and physics and less on rendering or scene setup for the GPU. In this instance, the application is CPU-bound and configurations dealing with non-graphics tasks need to be reviewed and modified. If the system is GPU-bound, the profiler can point out where the GPU programming code bottlenecks are located and which sections to optimize to remove restrictions.

16.2 Optimizing off-chip data transfer such as accessing off-chip DDR memory/mobile DDR memory

Any data transfer off-chip takes bandwidth and resources from other functional blocks in the SoC, increases power, and causes additional cycles of latency and delay as the GPU pipeline needs to wait for data to return from memory. Using on-chip cache and writing the application to better take advantage of cache locality and coherency increase performance. In addition, accessing the GPU frame buffer from the CPU (not recommended) cause the driver to flush all queued render commands in the command buffer, slowing down performance as the GPU has to wait since the command queue is partially empty (inefficient use of resources) and CPU-GPU synchronization is not parallelized.

16.3 Avoiding W-clipping issue in the application program

The w-clipping overflow issue typically occurs with these three factors:

- **Objects with very large primitives.**
  In a 3D scene, this is usually the sky, the outer world or a long road that expands far behind the camera and far in front of the camera. At the same time, the object may also expand far in either the x or y direction.
- **Near-plane with a very small value**
  Usually this value is very close to zero. An example would be 10^{-4}.
- **Large screen resolution**

These three factors can cause the final window coordinate to overflow the 24-bit mantissa precision in IEEE single precision floating point format.

The following are suggested ways to modify an application to avoid overflow:

1. For draw calls with very large primitives such as sky or world, set the near-plane to 0.99 as an initial value.
2. If this removes the rendering error and the entire scene is rendered correctly, the issue can be considered resolved.
3. If the rendering error is still there and no desired objects are being culled (or there are no missing objects), increase the near-plane value until the rendering error disappears.
4. If the near-plane value is large (>10.0) already, the issue persists and some desired objects are being culled, reduce the near-plane value until the desired objects appear again then go to the next step.
5. Tessellate the large objects into smaller primitives until the rendering error disappears.

Please note that the suggested near plane adjustment can be done on a per draw call basis, and only needs to be modified for objects with very large primitives. Some applications scale the object by reduce the w value in vertex shader, as change w value will finally affect the near plane, this is not recommended, a better way to scale the object is scale the x, y, z coordinate, not w.

16.4 Avoiding GPU hanging and data corruption when using occlusion query

Description:

On i.MX 6Dual/Quad GPU IP, both Hierarchical Depth (Hz) write and Occlusion Query (OQ) write share the same port. If HZ Fast Clear (FC) is enabled, and OQ uses the HZ port to perform a write, the HZ FC data may become corrupted, even lead to GPU hang unexpectedly.

Software Workaround:

A software workaround is recommended for this issue and is available from L4.9 bsp release. Because the issue occurs very infrequently, a per-application work around is most efficient. Software will disable HZ with a per-app detection and also provide a new environment variable control (VIV_DISABLE_HZ).

16.5 Avoiding random cache or memory access

Cache thrashing, misses, and the need to access data in external memory causes performance hits. An example would be random texture cache access since it is expensive when performing per-pixel texture reads if the texture units need to access the cache randomly and go off-chip if there is a cache miss.

16.6 Optimizing your use of system memory

Memory is a valuable resource that needs to be shared between the GPU (frame buffer), CPU, system, and other applications. If you allocate too much memory for your OpenGL ES application, less memory is available for the rest of the system, which may impact system performance. Claim enough memory as needed for your application then deallocate it as soon as your application no longer needs it. For example, you can allocate a depth buffer only when needed or if your application only needs partial resources, load the necessary items initially and load the rest later.

16.7 Targeting a fixed frame rate that is visibly smooth

Smooth frame rate is achieved from a combination of a constant FPS and the lowest FPS (frames per second) that is visually acceptable. There is a trade-off between power and frame rates since the graphics engine loading increases with higher FPS. If the application is smooth at 30 FPS and no visual differences for the application are perceived at 50 FPS, then the developer should cap the FPS at 30 since the extra 20 FPS do not make a visual difference. The FPS limit also guarantees an achievable frame rate at all times. The savings in FPS help lower GPU and system power consumption.

16.8 Minimizing GL state changes

Setting up state values between draw calls adds significant overhead to application performance so they must be minimized. Most of these call setups are redundant since you are saving / restoring states prior to drawing. Try to avoid setting up multiple state calls between draw calls or setting the same values for multiple calls. Sometimes when a specific texture is used, it is better to sort draw calls around that texture to avoid texture thrashing which inhibits performance. Application developers should also try to group state changes.

16.9 Batch primitives to minimize the number of draw calls

When your application submits primitives to be processed by OpenGL ES, the CPU spends time preparing commands for the GPU hardware to execute. If you batch your draw calls into fewer calls, you reduce the CPU overhead and increase draw call efficiency. Batch processing allows a group of draw calls to be quickly executed without any intervention from the CPU (driver or application) in a fire-and-forget method.

Some examples of batching primitives are:

- Branching in shaders may allow better batching since each branch can be grouped together for execution.
- For primitives like triangle strips, the developer can combine multiple strips that share the same state to save successive draw calls (and state changes) into a single batch call that uses the same state (single setup) for many triangles.
- Developers can also consolidate primitives that are drawn in close proximity to take advantage of spatial relationships. If the batched primitives are too far apart, it is more difficult for the application to effectively cull if they are not visible in the frame.

16.10 Performing calculations per vertex instead of per fragment/pixel

Since the number of vertices is usually much less than the number of fragments/pixels, it is cheaper to do per vertex calculations to save processing power.

16.11 Enabling early-Z, hierarchical-Z, and back face culling

Hardware support of depth testing to determine if objects are in the user’s field of view are used to save workload and processing on vertex and pixel processing. If the object is in view, then the vertices are sent down the pipeline for processing. If the object is hidden or not viewable, the triangles are culled and not sent to the pipeline. This improves graphics performance since computations are only spent on visible objects. If the application already knows details about the contents and relative position of objects in the scene or screen, the developer can use that information to automatically bound areas that never need to be touched (for example an automotive application that has multiple layers of dials where parts of the underlying dials are occluded can have the application avoid occluded areas from the beginning). Another optimization is to perform basic culling on the CPU since the CPU has first-hand information about the scene details and object positions so it knows what scene data to send to the GPU.

16.12 Using branching carefully

Static branches perform well since states are known but they tend to use many general purpose registers. An example is a long shader that combines multiple shaders into a single, large shader that reduces state changes and batch draw calls. Dynamic branching has non-constant overhead since it processes multiple pixels as one and everything executes whether a branch is taken or not. In other words, dynamic branching goes through different permutations/branches in parallel to reach the correct results. If all pixels take the same path, then performance is good. The more pixels processed translates to higher overhead and lower performance. For dynamic branching, smaller pixel sizes/groups are optimal for throughput. Developers need to be aware of branching in their code to make sure excessive calculations and branches are efficient. Profiling tools can help determine if certain parts of code are optimized or not.

16.13 Using VBOs instead of static or stack data as vertex data

A vertex buffer object (VBO) is a buffer object that provides the benefits of vertex array and display list and allows a substantial performance gain for uploading data (vertex position, color, normals, and texture coordinates) to the GPU. VBOs create buffer objects in memory and allow the GPU to directly access memory without CPU intervention

(DMA). The memory manager can optimize buffer placement using feedback from the application. VBOs can also handle static and dynamic data sets and are managed by the Vivante driver. The benefits of each are:

- A vertex array reduces the number of function calls and allows redundant data to be shared between related vertices, instead of re-sending all the data each time. Access to data can be referenced by the array index.
- The display list allows commands to be stored for later execution and can be used repeatedly over multiple frames without re-transmitting data, thus minimizing CPU cycles to transfer data. The display list can also be shared by multiple OpenGL / OpenGL ES clients so they can access the same buffer with the corresponding identifier. If you put computationally expensive operations (ex. lighting or material calculations) inside display lists, then these computations are processed once when the list is created and the final result can be re-used multiple times without needing to re-calculate again.

If you combine the benefits of both by using VBO, the performance is enhanced over static or stack data sets.

16.14 Using dynamic VBO when the data is changing frame by frame

Locking a static vertex buffer while the GPU is using it can create a performance penalty since the GPU needs to finish reading the vertex data from the buffer before it can return to the calling application. Locking and rendering from a static buffer many times per frame also prevents the GPU buffering render commands since it mush finish commands before returning the lock pointer. Without buffered commands the GPU remains idle until the application finishes filling the vertex buffer and issues the draw commands.

If the scene data never changes from frame to frame then a static buffer may be sufficient. With newer applications (ex. games, maps) that have dynamic viewports where vertex data changes multiple times per frame or frame-to-frame, then a dynamic VBO is required to ensure performance is still met. If the *current* buffer is being used by the GPU when a lock is called, a pointer to a *new* buffer location is returned to the application to ensure updated data is written to the *new* buffer. The GPU can still access the old data (current buffer) while the application puts updated data into the new buffer. The Vivante memory management unit and driver automatically take care of allocating, re-allocating, or destroying buffers.

You can implement dynamic VBO depending on your preference, but one recommendation is to allocate a 1 MB dynamic VBO block and upload data to using different offsets for each dynamic buffer. If the buffer overflows you can loop back and use location offset 0 again.

16.15 Tessellating your data to make Hierarchical Z (HZ) work

We can break this into how OpenGL and OpenGL ES handle this use case.

OpenGL only renders simple convex polygons (edges only intersect at vertices with no duplicate vertices and only two edges meet at any vertex), in addition to points, lines, and triangles. If the application requires concave polygons (polygons with holes or intersecting edges), those polygons need to be subdivided into simple convex polygons, which is called tessellation (subdividing a polygon mesh into a bunch of smaller meshes). Once you have all the meshes in place our HZ hardware can automatically cull hidden polygons to efficiently process the frame, effectively breaking the frame into smaller chunks that can be processed very fast.

OpenGL ES only renders triangles, lines, and points. The same concepts apply as in OpenGL, which is to avoid very large polygons by breaking them down into smaller polygons where our internal GPU scheduler can distribute them into multiple threads to fully parallelize the process and remove hidden polygons.

16.16 Using dynamic textures as a texture cache (texture atlas)

The main reason for using dynamic textures as a cache is the application developer can create one larger texture that is subdivided into different regions (texture atlas). The application can upload data into each region and use an application side texture atlas to access the data. Each dynamic texture and sub-region can be locked, written to, and unlocked each frame, as needed. This method of allocating once is more efficient than using multiple smaller textures that need to be allocated, generated, and then destroyed each time.

16.17 Stiching small triangle strips together

It is better to combine several small, spatially related triangle strips together into a larger triangle stip to minimize overhead and increase performance. For each triangle strip, there are overhead and start up costs that are required by the CPU and GPU, including state loads. If there are too many small triangle strips that need to be loaded, this impacts performance. An application developer can combine multiple triangle strips by adding a degenerate triangle to join the strips together. The overhead to restart multiple new strips is much higher than adding the degenerate triangle.

16.18 Specifying EGL configuration attributes precisely

To obtain a 16 bit/pixel window buffer for rendering, the EGL config attributes need to be specified precisely according to the EGL spec. Specifying inaccurate EGL attributes may result in getting a 32-bit bit/pixel window buffer which doubles the bandwidth requirement for rendering which in turn leads to lower performance.

16.19 Using aligned texture/render buffers

The GPUs work on buffers with hardware-specific width/height alignment for better efficiency. Use the available API to query the GPU buffer alignment and allocate the texture / render buffers to satisfy these requirements, to avoid the cost of copies to aligned shadow memory.

16.20 Disabling MSAA rendering unless high quality is needed

Although MSAA rendering can achieve higher image quality with smoother lines and triangle edges, it requires much higher (4x, 8x) bandwidth because it has to rendering a single pixel 4x/8x times. So, if high rendering quality is not required, MSAA should be disabled.

16.21 Avoiding partial clears

Most GPUs have special hardware logic to do a fast clear of an entire buffer. So it is better to utilize the fast clear function to clear the entire buffer then render graphics again, instead of doing a partial clear to preserve a graphics region. If a partial clear is required by the application, make sure the clear area is aligned according to the GPU-specific requirements. Unaligned partial clears are expensive and should be avoided.

16.22 Avoiding mask operations

Do not use mask unless the mask is 0 (other than when you need a specific render quality). Clearing a surface with mask (color/depth stencil mask) could have a performance penalty.Pixel mask operations are normally pretty expensive on some GPUs as the mask operation has to be done on every single pixel.

16.23 Using MIPMAP textures

MIPMAP textures enable the application to sample a lower resolution texture image (1/2, 1/4, 1/8, 1/16, ... size of the original texture image) when the triangle is rendering further away from the view point. Thus, the bandwidth required to read the texture image is reduced which leads to better performance.

16.24 Using compressed textures if constricted by RAM/ROM budget

Compressed textures are normally only a fraction (up to 1/8) of the original texture size. Using compressed textures reduces the storage requirements in memory and can also reduce the required texture upload bandwidth, when using a format that is supported natively by the hardware.

Compressed textures should not be chosen, if only for the purposes of reducing the memory bandwidth required for sampling of the texture during rendering. This is because due to a fixed read request size from the GPU, the memory controller load is the same as for an uncompressed texture.

16.25 Drawing objects from near to far if possible

Drawing objects from near to far normally has better performance because the objects in the near foreground can block entire or partial objects in the background. Most GPUs have early Z rejection logic to reject the pixels that fail a Z compare. The GPU can skip fragment shader computations on these rejected pixels.

16.26 Avoiding indexed triangle strips

Index triangle strips can usually maximize the vertex cache utilization as each set of vertex data can be used in two triangles. There is however an errata in the GC2000 and GC880 GPUs which requires a SW conversion of indexed triangle strips to triangle lists in the driver. For small strips the conversion overhead is negligible, but for large geometries a different primitive type should be used.

16.27 Limiting vertex attribute stride within 256 bytes

Most Vivante GPUs provide native support for a 256 byte vertex attribute stride. If the vertex attribute stride is larger than 256 bytes, then the driver has to copy the vertex data around. Hardware versions v55 and higher (such as the GC7000L v55) support a 2048 byte vertex attribute stride as required in the OES3.1 spec.

16.28 Avoiding binding buffers to mixed index/vertex array

Most of Vivante GPUs do not natively support mixed index/vertex arrays. So the Vivante driver must copy the index and vertex data around to form separate vertex data streams for the GPU. Avoid mixing index and vertex data so the driver does not have to incur a performance hit while performing this task.

16.29 Avoiding using CPU to update texture/buffer contexts during render

Do not use the CPU to update texture/buffer contexts in the middle of rendering. Using the CPU to update texture/buffer causes the rendering pipeline to flush and stall, so that CPU can safely update the buffer contents. The pipeline flush/stall/resume causes significant performance impact.

16.30 Avoiding frequent context switching

Context switch is an inherently expensive operation as many GPU states need to be reset to start a new rendering context. Thus, frequent context switching has a negative impact on application performance.

16.31 Optimizing resources within a shader

Most GPUs have optimal support for a limited amount of resources (uniforms, varying, etc.). Using resources beyond the optimal working set causes the GPU to fetch/store resources from a lower performance memory pool and shader performance is negatively impacted.

16.32 Avoiding using glScissor Clear for small regions

glScissor Clear for small regions (less than 16x8 aligned window) fall back to CPU so the performance is not optimal.

16.33 Using PRE to accelerate data transfer

PRE is an optimized hardware that can transform tiled format image to linear framebuffer. With PRE, GPU can onlyoutput tiled render target and has no need to resolve it. To enable the PRE feature, set the environment GPU_VIV_EXT_RESOLVE variable to 1; otherwise set it to 0. Its default value on the FB backend is 1, which means PRE is enabled by default on FB.

Warning:

VG use cases can only output the linear format image. It is impossible to render linear and tiled format target to the same framebuffer at the same time. Therefor, when running 3D use cases with PRE and VG use cases together, there is garbage on the display. Besides, when running 3D use cases with PRE, the framebuffer format is changed from linear to tiled. It is the user’s responsibility to convert the format back after the use cases end, or the display is abnormal when showing the FB console.

16.34 i.MX 8QuadMax dual-GPU performance

For some legacy applications with small texture/rendering size and less shader complex, dual-GPU performance may become worse than single GPU mode, because the driver needs to take more CPU effort for dual-GPU programming, and the driver overhead is more significant than GPU load in the hardware pipeline.

For such a kind of legacy case, the users can single-GPU to achieve better performance on the i.MX 8QuadMax.

17 Demo Framework

17.1 Overview

This document describes the NXP Demo Framework, targeted at platform agnostic development of graphical demos. It covers the goals, architecture and instructions of how to use it across platforms, examples and best practices.

17.1.1 Executive summary

- Write a demo application once.
- Run it on Android, Yocto Linux, Ubuntu and MS Windows.
- Easily portable to additional platforms.
- Supports: OpenGL ES2, OpenGL ES3, OpenVG and experimental G2D support.

17.1.2 Technical overview

- Written in a limited subset of C++11 and uses RAII to manage resources.
- Uses a limited subset of STL to make it easier to port.
- No copyleft restrictions from GPL / L-GPL licenses.[1]
- Allows for direct access to the expected API's (EGL, ES2, ES3, VG).
- Provides optional helper classes for commonly used tasks.
  - Matrix, Vector3, GLShader, GLTexture, etc.
- Services:
  - Keyboard and mouse.
  - Persistent data manager.
  - Assets management (models, textures).
- Defines a standard way for handling.
  - Initialization, shutdown, and window resize.
  - Program input arguments.
  - Input events like keyboard, mouse and touch.
  - Fixed time-step and variable time-step demo implementations.
  - Logging functionality.

17.2 Introduction

The Demo Framework is a multi-platform framework that enables demos to run on various platforms without any changes. The framework abstracts away all the boilerplate & OS specific code of allocating surfaces, creating the context, model loading, texture loading, shader compilation, render loop, animation ticks, benchmarking graph overlays etc. This allows the demo/benchmark developer to focus on writing rendering code. It also enables them to develop demos on PC or Android where the tool chain and debug facilities allows for faster turnaround time and then take the working code and deploy without code changes to the supported platforms. The platforms we currently support are Windows (for development via emulated backends), Android NDK and Linux with various windowing systems. The framework allows us to provide 'real' comparative benchmarks between the different OS and windowing systems we support, since we can run the exact same demo/benchmark code on them all.

1 We don't use GPL or LGPL.

The long term plans for the framework include extending it with support for other relevant API’s.

17.3 Design overview

The framework is written in C++ and uses RAII[2] to manage resources. The resource management code focuses on ‘ease of use’ over raw performance, since it’s mainly run on construction and destruction of the demo.

To allow the demo framework to be easily portable to new platforms, its functionality is split into two parts: ‘core’ and ‘services’. The core framework depends on a limited subset of STL to make it easier to port. Framework services come with their own set of library requirements. The model importer Assimp[3] requires boost to be available on the platform.

Besides the demo framework core and demo framework services, there is a set of helper classes for commonly used functionality, which makes it easier to write demo’s for the API’s we support. The helper classes do not depend on the demo framework and can be used in any program for the given API. For example, for OpenGL ES, there is a GLShader and GLProgram class that hides away the complexities of compiling the shader object and linking the program object and since they are RAII objects, they also clean up after themselves once you are done with them.

Since our primarily supported BSPs are based on Linux OS, we decided to use an input argument framework that is compatible with the standard Unix parameter format, like the one exposed by getopt[4].

2 http://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization
3 http://www.assimp.org/
4 We do however not utilize getopt to remain GPL free across platforms.

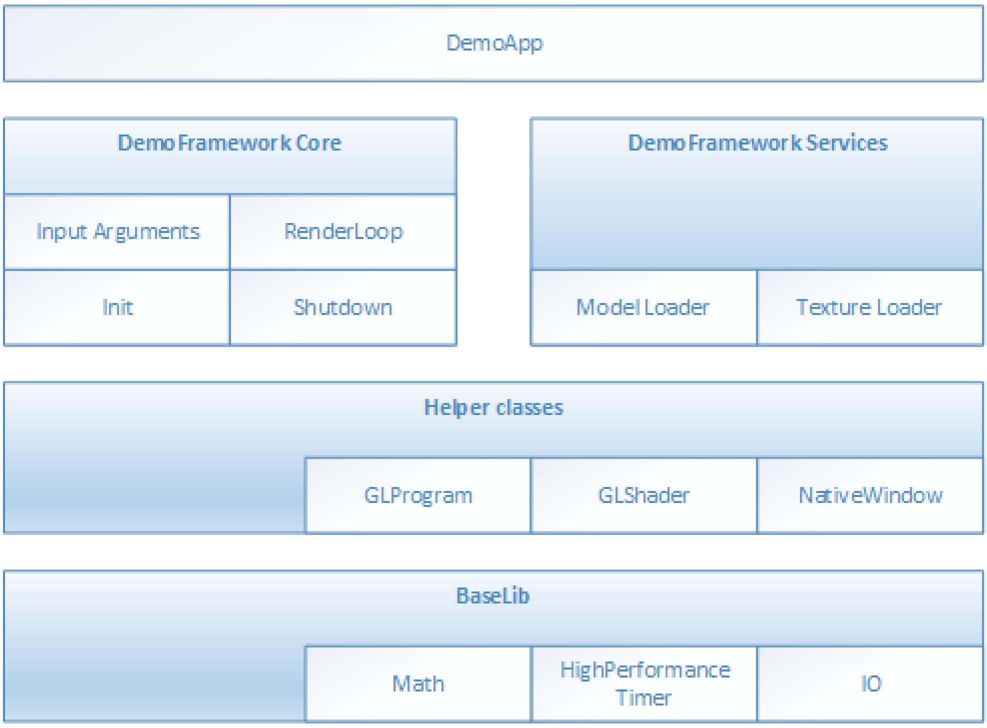


Figure 41. Demo framework

17.4 High level overview

The framework consists of three high level domains.

17.4.1 DemoMain

All the code that binds everything together and it is platform independent.

- It gets the current demo setup:
  - Which demo host to utilize for the demo.
  - Which demo application that needs to be run.
- It parses the input arguments.
- It launches the demo host.
- It logs any errors that might occur.

Figure 42. DemoMain

17.4.2 DemoHost

The demo-host is responsible for init & shutdown of the host environment and running the main loop.

The main loop utilizes the DemoAppManager to control the life of the DemoApp.

In other words, the DemoHost is the graphics API specific code needed to initialize and shutdown a given API and some code to run a render loop. All the API and platform independent code of the render loop resides inside the DemoAppManager class.

The exact capabilities of a DemoHost are also platform dependent. For example, some EGL implementations support running OpenVG and OpenGL ES, allowing a demo app to utilize both API's at once. This is not something that is supported by most windows emulation layers.

17.4.3 DemoApp

A demo application written for one or more specific APIs, which are supported by a specific DemoHost. The demo is usually platform independent. The exception to the rule is if it depends on specific features that only exist on certain platforms.

17.5 Demo application details

The following description of the demo application details uses a GLES2 demo named 'S01_SimpleTriangle' as example. It lists the default methods that a demo should implement, the way it can provide customized parameters to the windowing system and how asset management is made platform agnostic.

17.5.1 Demo method overview

This is a list of the methods that every Demo App is most likely to override[5].

```
// Init
S01_SimpleTriangle(const DemoAppConfig& config)
// Shutdown
~S01_SimpleTriangle()
// OPTIONAL: Custom resize logic (if the app requested it). The
 default logic is to
// restart the app.
void Resized(const Point2& size)
// OPTIONAL: Fixed time step update method that will be called
 the set number of times
// per second. The fixed time step update is often used for
 physics.
void FixedUpdate(const DemoTime& demoTime)
// OPTIONAL: Variable time step update method.
void Update(const DemoTime& demoTime)
// Put the rendering calls here
void Draw(const DemoTime& demoTime)
```

When the constructor is invoked, the Demo Host API will already be setup and ready for use, the demo framework will use EGL to configure things as requested by your EGL config and API version.

It is recommended that you do all your setup in the constructor.

This also means that you should never try to shutdown EGL in the destructor since the framework will do it at the appropriate time. The destructor should only worry about resources that your demo app actually allocated by itself.

5 See DemoFramework\FslDemoApp\include\FslDemoApp\ADemoApp.hpp for a complete list.

17.5.1.1 Resized

The resized method will be called if the screen resolution changes (if your app never changes resolution this will never be called)[6].

17.5.1.2 FixedUpdate

Is a fixed time-step update method that will be called the set number of times per second. The fixed time step update is often used for physics[7].

17.5.1.3 Update

Will be called once before every draw call and you will normally update your animation using delta time.

For example, if you need to move your object 10 units horizontally per second, you would do as follows:

```
m_positionX += 10 * demoTime.DeltaTime;
```

17.5.1.4 Draw

Should be used to render graphics.

17.5.2 Fixed or variable timestep update

Depending on what your demo is doing, you might use one or the other - or both. It's actually a very complex topic once you start to dig into it, but in general anything that need precision and predictable/repeatable calculations, like for example physics, often benefits from using fixed time steps. It really depends on your algorithm and it's recommended to do a couple of google searches on fixed vs variable, since there are lots of arguments for both. It's also worth noting that game engines like Unity3D[8] support both methods.

17.5.3 Execution order of methods during a frame

The methods will be called in this order

- Events (if any occurred)[9]
- Resized[10]
- FixedUpdate (0-N calls. The first frame will always have a FixedUpdate call)
- Update
- Draw

After the draw call, a swap will occur.

6 This version of the framework always restart the app, so this will never be called.
7 This version uses a fixed update frequency of 60 ticks per second. This will be configurable in the future.
8 https://unity.com/
9 For an example of event handling see the “DemoApps\GLES2\InputEvents” sample.
10 In this version of the framework this is never called as the app will be recreated on screen size changes (future versions will allow demo apps to handle resize events if they so desire)

17.5.4 Exit

The demo application can request an exit to occur, or it can be terminated through an external request.

In both cases, one of the following things occurs.

- If the application has been constructed and has received a FixedUpdate, it will finish its FixedUpdate, Update, Draw, swap sequence before its shutdown.
- If the application requests a shutdown during construction, the application will be destroyed before calling any other method on the object (and no swap will occur).

The application can request an exit to occur by calling:

```
GetDemoAppControl()->RequestExit(1);
```

17.5.5 Dealing with screen resolution changes

By default, the application is destroyed and recreated when a resolution change occurs. Future versions will allow demo apps to handle resize events if they so desire.

It is left up to the DemoApp to save and restore demo specific state.

17.5.6 Content loading

The framework supports loading files from the Content folder on all platforms.

Given a content folder like this:

```
Content/Texture1.bmp
```

```
Content/Stuff/Readme.txt
```

You can load the files via the *IContentManager* service that can be accessed by calling

```
std::shared_ptr<IContentManager> contentManager =
GetContentManager();
```

You can then load files like this:

Binary file:

```
std::vector<uint8_t> content;
```

```
contentManager->ReadAllBytes(content, "MyData.bin");
```

Text file:

```
const std::string content = contentManager->ReadAllText("Stuff/
Readme.txt");
```

Bitmap file[11]:

```
Bitmap bitmap;
```

```
contentManager->Read(bitmap, "Texture1.bmp",
PixelFormat::R8G8B8_UINT);
```

If you prefer to control the loading yourself, you can retrieve the path to the files like this:

```
IO::Path contentPath = contentManager->GetContentPath();
```

11 The current framework only png, bmp and jpeg images on all platforms but a few platforms has access to all formats supported by the DevIL library.

```
IO::Path myData = IO::Path::Combine(contentPath, "MyData.bin");
```

```
IO::Path readmePath = IO::Path::Combine(contentPath, "Stuff/
Readme.txt");
```

```
IO::Path texture1Path = IO::Path::Combine(contentPath,
"Texture1.bmp");
```

You can then open the files with any method you prefer.

Both methods work for all supported platforms.

For detailed information about how the content is handled on each platform, see the build guide appendixes.

17.5.7 Demo registration

This is done in the `S01_SimpleTriangle_Register.cpp` file.

```
namespace Fsl
{
  namespace
  {
     // Custom EGL config (these will per default overwrite the
 custom settings.
     // However, an exact EGL config can be used)
     static const EGLint g_eglConfigAttribs[] =
     {
        EGL_SAMPLES, 0,
        EGL_RED_SIZE, 8,
        EGL_GREEN_SIZE, 8,
        EGL_BLUE_SIZE, 8,
        EGL_ALPHA_SIZE, 0, // buffers with the smallest alpha
 component size are preferred
        EGL_DEPTH_SIZE, 24,
        EGL_SURFACE_TYPE, EGL_WINDOW_BIT,
        EGL_NONE,
     };
  }
  // Configure the demo environment to run this demo app in a
 OpenGLES2 host environment
  void ConfigureDemoAppEnvironment(HostDemoAppSetup& rSetup)
  {
    DemoAppHostConfigEGL config(g_eglConfigAttribs);

 DemoAppRegister::GLES2::Register<S01_SimpleTriangle>(rSetup,
 "GLES2.S01_SimpleTriangle", config);
  }
}
```

Since the demo framework is controlling the main method, you need to register your application with the Demo Host specific registration call (in this case the OpenGL ES2 host), for the framework to register your demo class.

17.5.7.1 OpenGLES 3.X registration

To register a demo for OpenGLES 3.X you would use the GLES3 register method:

```
DemoAppRegister::GLES3::Register<S01_SimpleTriangle>(rSetup,
 "GLES3.S01_SimpleTriangle", config);
```

17.6 Demo playback

17.6.1 Command line arguments

All demos support various command line arguments.

Table 42. Command line arguments

Key	Function
-h	Show the command line argument help.
--Stats	Show a performance graph.
--LogStats	Log various stats to the console.
--ScreenshotFrequency	Create a screenshot at the given frame frequency.
--ExitAfterFrame	Exit after the given number of frames has been rendered
--ContentMonitor	Monitor the Content directory for changes and restart the app on changes. WARNING: Might not work on all platforms and it might impact app performance (experimental).

Use `-h` on a demo for a complete list.

17.6.2 Demo single stepping/pause

In Windows OS, all samples support time stepping, which can be useful for debugging. It might also be available on under platforms that support the given keys.

Table 43. Demo single stepping/pause

Key	Function
Pause	Pause the sample.
PageDown	Move forward one timestep.
Delete	Toggle between normal and Slow 2x playback
End	Toggle between normal and Slow 4x playback
Insert	Toggle between normal and fast 2x playback.
Home	Toggle between normal and fast 4x playback.

17.7 Helper class overview

17.7.1 FslBase

Provides basic functionality missing from C++ standard libraries.

17.7.1.1 Bits

BitsUtil	Utility methods for working with bits.
ByteArrayUtil	Utility methods for reading and writing values from byte arrays in a specific endian format. This functionality is useful when working on platform independent load and save methods.

17.7.1.2 IO

Platform independent IO.

Table 44. IO

Directory	Helper methods for working on directories. • GetCurrentWorkingDirectory.
File	Helper methods for working with files • Checking if file exists. • File length. • Read all content from a file.
Path	A UTF8 path class and helper methods for working on it. • Combing paths. • Extracting directory or filename. • Getting the full path from a relative path.

17.7.1.3 Log

Platform independent logging.

Instead of using printf or std::cout to log information it's better to utilize the provided logging macro's since work across all supported platforms.

Table 45. Log

Log	Various logging macros • FSLLOG • FSLLOG_IF • FSLLOG_WARNING • FSLLOG_WARNING_IF • FSLLOG_ERROR • FSLLOG_ERROR_IF

17.7.1.4 Math

Mainly focused on math functionality useful for working with graphics. It focuses on ease of use instead of raw performance.

Table 46. Math

MathHelper	Various commonly used helper methods and constants like • PI • Clamping • Lerp • Conversions between radians and angles • PowerOfTwo

Table 46. Math*...continued*

Matrix	Matrix helper methods like • Perspective • Rotate • Translate • Scale • Multiply
Point2	A 2D integer point.
Rectangle	A integer based rectangle with helper methods like • Union • Intersection
Vector2	A 2d float point with helper methods like • Dot • Length • Lerp • Min, max • Normalize • Reflect
Vector3	A 3d float point with helper methods like • Cross • Dot • Length • Lerp • Min, max • Normalize • Reflect • Transform by matrix
Vector4	A 4d float point with helper methods like • Dot • Length • Lerp • Min, max • Normalize • Reflect • Transform by matrix
Quaternion	Basic Quaternion operations.

17.7.1.5 String

Table 47. Various string functionality

StringParseUtil	Various utility method for converting a string to a number.
UTF8String	A UTF8 string representation.

17.7.1.6 System

Table 48. System

HighResolutionTimer	A platform independent high resolution timer.

17.7.2 FslGraphics

Table 49. FslGraphics

Bitmap	A RAII class to manage bitmap data.
BitmapUtil	Contains various helper methods that works on the bitmap class. • Horizontal flip • Pixel format conversion
Color	RGBA color utility class.
PixelFormat	Various standardized pixel formats supported by the bitmap classes.
RawBitmap	Read only bitmap information.
RawBitmapEx	Writeable access to bitmap information
RawBitmapUtil	Low level helper methods that work on RawBitmap's • Horizontal flip • Padding clear • Swizzle

17.7.2.1 Font

Table 50. Font

BasicFontKerning	Contains basic kerning information for a font.
BinaryFontBasicKerningLoader	Load basic kerning information from "fbk" files.
FontDesc	A very basic font description.
FontGlyphBasicKerning	Basic kerning for one glyph.
FontGlyphPosition	Position information for one glyph
FontGlyphRange	Font glyph range information.
IFontBasicKerning	Interface for extracting basic font kerning information.
TextureAtlasBitmapFont	Describes a bitmap font stored in a texture atlas.
TextureAtlasGlyphInfo	Texture atlas glyph information.

17.7.2.2 IO

Table 51. IO

BMPUtil	A simple helper class for loading and saving BMP images. It's not recommended to utilize it directly. Instead utilize the framework for loading images[1].

[1] A future version will also add saving to the ContentManager.

17.7.2.3 Render

Table 52. Render

AtlasFont	An atlas based bitmap font using an API independent texture.
AtlasTexture2D	An atlas based API independent texture.
BlendState	API independent blend states.

Table 52. Render...continued

GenericBatch2D	An API independent 2D quad batcher.
Texture2D	An API independent texture representation.

17.7.2.4 TextureAtlas

Table 53. TextureAtlas

AtlasTextureInfo	Represents information about one texture that is stored in a texture atlas.
BasicTextureAtlas	A simple manager for looking up AtlasTextureInfo.
BinaryTextureAtlasLoader	A “BTA” basic texture atlas loader.
ITextureAtlas	Simple interface for accessing texture information.
NamedAtlasTexture	A named atlas texture.
TextureAtlasHelper	A simple way to extract AtlasTextureInfo from a texture atlas.
TextureAtlasMap	A more performance efficient way to extract AtlasTextureInfo from a texture atlas.

17.7.2.5 Vertices

API independent vertex helper classes.

Table 54. Vertices

IndexConverter	Simple utility class to convert between index formats. It might not be efficient but it gets the job done.
VertexConverter	Simple utility class to convert between vertex formats. It might not be efficient but it gets the job done.
VertexDeclaration	Defines how a vertex is constructed in an API independent way.
VertexElementEx	Defines a vertex element
VertexPositionColor	A vertex comprised of • position • color.
VertexPositionColorNormal Texture	A vertex comprised of • position • color • normal • texture coordinates
VertexPositionColorTexture	A vertex comprised of • position • color • texture coordinates
VertexPositionNormalTexture	A vertex comprised of • position • normal • texture coordinates

Table 54. Vertices...*continued*

VertexPositionTexture	A vertex comprised of • position • texture coordinates

17.7.2.6 Window

Table 55. Window

INativeWindow	An abstract from native windows.

17.7.3 FslUtil.OpenGLES2

RAII based helper classes for common GLES2 operations.

Table 56. FslUtil.OpenGLES2

GLBatch2D	A specialization of GenericBatch2D GLES2.
GLBatch2DQuadRenderer	The GenericBatch2D backend for rendering quads.
GLCheck	Various helper macro's for checking and transforming OpenGL ES errors to exception.
GLFrameBuffer	A RAII based frame buffer encapsulation.
GLIndexBuffer	A RAII based index buffer. • uint8_t & uint16_t based index buffers. • Easy creation and update.
GlIndexBufferArray	A RAII based index buffer array. • Improved efficiency when allocating many index buffers of the same format.
GLProgram	A RAII based GL program encapsulation. • Vertex and fragment shader combination.
GLRenderBuffer	A RAII based GL render buffer encapsulation.
GLShader	A RAII based GL shader encapsulation. • Compilation and logging.
GLTexture	A RAII based GL texture encapsulation. • Can be created from either FslGraphics RawBitmap's or Bitmaps. • Easy content update. • Supports both normal and cubemap textures.
GLUtil	Contains various utility methods for OpenGL ES2 • Capture screenshots
GLVertexBuffer	A RAII based vertex buffer. • Easy creation and updating from Custom or Fsl Graphics.Vertices. • Helper methods for quickly enabling/disabling Attribs
GLVertexBufferArray	A RAII based vertex buffer array. • Improved efficiency when allocating many vertex buffers of the same vertex format.
NativeBatch2D	Extends GenericBatch2D with direct support for GLES2 native textures.

Table 56. FslUtil.OpenGLES2*...continued*

NativeTexture2D	Implements the INativeTexture2D for GLES2. This is used by the Batch2D system.

17.7.4 FslUtil.OpenGLES3

RAII based helper classes for common GLES3 operations.

GLES3 has the exact same helper classes as GLES2 and the following additions:

Table 57. FslUtil.OpenGLES3

GLVertexArray	A RAII based vertex array. • Easy creation

17.7.5 FslUtil.OpenGLES3v1

RAII based helper classes for common GLES3.1 operation's.

Table 58. FslUtil.OpenGLES3v1

GLProgramPipeline	A RAII based program pipeline encapsulation.
GLShaderProgram	A RAII based shader program encapsulation.

17.7.6 FslUtil.OpenVG

RAII based helper classes for common OpenVG operations.

Table 59. FslUtil.OpenVG

VGPathBuffer	A RAII based path buffer • Easy creation
VGUtil	Contains various utility methods for OpenVG • Capture screenshots
VGCheck	Various helper macro's for checking and transforming OpenVG errors to exception.

17.7.7 FslGraphics3D

API independent descriptions of common 3D classes. This library is in development.

See the ModelLoaderBasics and ModelViewer samples for examples of how to use it.

Table 60. FslGraphics3D

Mesh	A basic mesh
Scene	A basic scene
SceneNode	A basic node in the scene

17.7.8 FslAssimp

The demo framework's Assimp integration provides various helper classes that make it easier to work with assimp in the framework.

Table 61. FslAssimp

MeshHelper	Helps to extract information from some assimp structures.
MeshImporter	Helps convert Assimp mesh structures to the FslGraphics3D ones.
SceneHelper	Extract basic information from a assimp scene.
SceneImporter	Helps convert Assimp scene structures to the FslGraphics3D ones.

17.7.9 FslGraphics3D.SceneFormat

Code to load and save a very basic portable scene format.

Table 62. FslGraphics3D.SceneFormat

BasicSceneFormat	Load/save scene functionality.

17.7.10 FslSimpleUI

A new experimental UI framework that makes it easy to get a basic UI up and running. The main code is API independent. It is not a show case of how to render a UI fast but only intended to allow you to quickly get a UI ready that is good enough for a demo.

You can look at:

- DFSimpleUI100
- DFSimpleUI101
- TessellationSample

To see how it's used.

The next release of the framework should make it even easier to work with.

When working with the UI system its recommended to store all or at least the most used bitmaps in the same texture atlas. One commercially available texture packer is Texture Packer which can output a json file that we can convert to a binary format that can be loaded by the demo framework.

If you look at the DFSimpleUI100 sample, there is "OriginalContent/TextureAtlas" directory which contain a "MainAtlas.tps" file that can be loaded into texture packer. Pressing publish in texture packer produces a "MainAtlas.png" and "MainAtlas.json" file based on the files under "Main". The "MainAtlas.png" can be copied directly to the samples "Content" directory but the json file needs to be converted to a binary file. For this we included the TPConvert python script that can be run like this:

```
TPConvert MainAtlas.json -f bta1
```

This will then produce a "MainAtlas.bta" file that can be copied to the "Content" directory which contains all the needed atlas meta data.

Please beware that the default atlas is required to contain the default font as well. The documentation for creating the "MainAtlas.fbk" file has not been completed yet. The fbk file contains some basic font kerning information.

17.8 FslBuild scripts

To read a text version of this document, look in the demo framework "Doc/FslBuild_toolchain_readme.md", which contains more detailed information.

17.8.1 FslBuildGen.py

Is a cross-platform build-file generator. Its main purpose is to keep all build files consistent, in synchronized and up to date. See `FslBuildGen.docx` for details.

17.8.2 FslBuild.py

Extends the technology behind FslBuildGen with additional knowledge about how to execute the build system for a given platform.

So basically, FslBuild works like this

1. Invoke the build-file generator that updates all build files if necessary.
2. Filter the builds request based on the provided feature list.
3. Build all necessary build files in the correct order.

17.8.2.1 Useful arguments

FslBuild comes with a few useful arguments

Table 63. Useful arguments

--ListFeatures	List all features required by the build
--UseFeatures	Allows you to limit what’s build based on a provided feature list. For example [EGL,OpenGLES2]. This parameter defaults to all features.
-t 'sdk'	Build all demo framework projects
-v	Set verbosity level
--	All arguments written after this is send directly to the native build system.

17.8.2.2 Important notes

- Do not modify the auto-generated files.
  The FslBuild scripts are responsible for creating all the build files for a platform and verifying dependencies. Since all build files are auto generated you can never modify them directly as the next build will overwrite your changes.
  Instead add your changes to the `Fsl.gen` files as they control the build file generation!
- The `Fsl.gen` file is the real build file.
- All include and source files in the respective folders are automatically added to the build files.

17.8.3 Build system per platform

Android	gradle
Qnx	Make
Ubuntu	Make
Windows	Visual studio (IDE or nmake)
Yocto	make

17.9 Android SDK+NDK on Windows build guide

For an easy-to-read text version of this document, see the demo framework “Doc/ Setup_guide_android.md”, which contains more detailed information.

17.9.1 Prerequisites

- Read Section 17.8 to know about the custom build system.
- **IMPORTANT:** The way Gradle currently handles CMake builds on windows place some serious limits on the path length, so its recommended to either place the DemoFramework folder close to the root of the drive or to set the environment variable FSL_GRAPHICS_SDK_ANDROID_PROJECT_DIR to a directory close to the root of the drive.
- JDK (64 bit)
  IMPORTANT: Make sure to configure JAVA_HOME to point to the JDK directory.
- Android SDK
  Once it is installed, run `SDK Manager.exe` and make sure everything is up to date.
  IMPORTANT: Android studio must be at least 3.1.
  IMPORTANT: Get the Android studio full package and enable the default packages.
  Configure the SDK manager
  - "SDK Platforms" added if necessary
    - Android 7.0 (Nougat)
  - "SDK Tools" added if necessary
    - CMake, LLDB, NDK, Android Support Repository
      IMPORTANT: Make sure to configure ANDROID_HOME to point to the android sdk directory
      IMPORTANT: Make sure to configure ANDROID_NDK to point to the android ndk directory
      IMPORTANT: Make sure you have at least android-ndk-r16b
- Python 3.4.x or better. We highly recommend at least 3.5+
  - For 64-bit Windows

17.9.2 Environment setup

Android projects are generated to the path specified in the environment variable `FSL_GRAPHICS_SDK_ANDROID_PROJECT_DIR`. If it is not defined the 'prepare' script, sets it to a default location.

1. Start a Windows console (`cmd.exe`) in the Demo Framework's root folder.
2. Run the `prepare.bat` file located in the root of the framework folder to configure the necessary environment variables and paths. Please beware that the `prepare.bat` file requires the current working directory to be the root of your demoframework folder to function (which is also the folder it resides in).

17.9.3 To Compile and run an existing sample application

In this example, we use the `GLES2 S06_Texturing` application.

1. Make sure that you performed the environment setup.
2. Change directory to the sample directory:
   `cd DemoApps\GLES2\S06_Texturing`
3. Build an application for Android using gradle + CMake:
   `FslBuild.py -p android`

If you just want to regenerate the CMake build files then you can just run:

```
FslBuildGen.py -p android
```

If you want to save a bit of compilation time you can build for the ANDROID ABI you need by adding:

```
FslBuildGen.py -p android --Variants [ANDROID_ABI=arm64-v8a]
```

or

```
FslBuild.py -p android --Variants [ANDROID_ABI=arm64-v8a]
```

17.9.4 To create a new GLES2 demo project named 'CoolNewDemo'

1. Make sure that you performed the environment setup.
2. Change directory to the GLES2 sample directory:
   `cd DemoApps/GLES2`
3. Create the project template using the FslBuildNew.py script
   `FslBuildNew.py GLES2 CoolNewDemo`
4. Change directory to the newly created project folder 'CoolNewDemo'
   `cd CoolNewDemo`
5. Build a app for Android using gradle + cmake

```
FslBuild.py -p android
```

If you just want to regenerate the cmake build files then you can just run

```
FslBuildGen.py -p android
```

If you want to save a bit of compilation time you can build for the ANDROID ABI you need by adding

```
FslBuildGen.py --Variants [ANDROID_ABI=armeabi-v7a]
```

or

```
FslBuild.py --Variants [ANDROID_ABI=armeabi-v7a]
```

17.9.5 Using Android studio

1. Follow the instructions for "creating a new project" or "building an existing project".
2. As projects are generated to the path specified by the FSL_GRAPHICS_SDK_ ANDROID_PROJECT_DIR environment variable you can locate the project there and open it with android studio. Be sure to open Android studio in a correctly configured environment. Here it could be a good idea to create a script for launching android studio with the right environment.

17.9.6 Linux notes

- Install for private user and unzip android studio like this:

```
sudo unzip android-studio-ide_FILENAME.zip -d ~/sdk
cd ~/sdk/android-studio/bin
./studio.sh
```

- In the ui make sure to install the sdk in a directory you have access to for example :

```
~/sdk/android-sdk-linux
```

17.9.7 Notes

17.9.7.1 Command line app building via Ant

http://developer.android.com/tools/building/building-cmdline.html

17.10 Ubuntu build guide

To read the text version of this document, look in the demo framework `Doc/Setup_guide_ubuntu16.04.md`, which contains more detailed information.

17.10.1 Prerequisites

- Read Section 17.8 to know about the custom build system.
- Ubuntu 16.04 64 bit
- Build tools and xrand
  `sudo apt-get install build-essential libxrandr-dev`
- Python 3.4+
  It should be part of the default Ubuntu16.04 install.
- An OpenGL ES 2+ emulator
  - Mesa OpenGL ES 2
    `sudo apt-get install libgles2-mesa-dev`
  - Arm Mali OpenGL ES 3.0 Emulator V3.0.2 (64 bit)
    ```
    wget https://armkeil.blob.core.windows.net/developer/Files/
    downloads/open-gl-es-emulator/3.0.2/Mali_OpenGL_ES_Emulator-
    v3.0.2.g694a9-Linux-64bit.deb
    sudo dpkg -i Mali_OpenGL_ES_Emulator-v3.0.2.g694a9-Li
    nux-64bit.deb
    ```
- DevIL
  - Developer's Image Library (DevIL)
    `sudo apt-get install libdevil-dev`
- Assimp
  Is now downloaded and build from source when needed. So its no longer necessary to run `sudo apt-get install libassimp-dev`.

17.10.2 Environment setup

1. Start a terminal (ctrl+alt t) in the DemoFramework folder
2. Run the `prepare.sh` file located in the root of the framework folder to configure the necessary environment variables and paths. Please beware that the `prepare.sh` file requires the current working directory to be the root of your demoframework folder to function (which is also the folder it resides in).
   `source prepare.sh`

17.10.3 Compiling all samples

1. Make sure that you have performed the environment setup
2. Compile `FslBuild.py -t sdk --BuildThreads 2`.

17.10.4 Compiling and running an existing sample application

In this example, we use the GLES2 S06_Texturing application.

1. Make sure that you have performed the environment setup.

2. Change directory to the sample directory: `cd DemoApps/GLES2/S06_Texturing`.
3. Compile the project (a good rule of thumb for '--BuildThreads N' is number of cpu cores * 2).
   If you run `FslBuild` without the `--BuildThreads` argument, it will be set to 'auto', which uses your CPU core count.
   `FslBuild.py --BuildThreads 2`

17.10.5 Creating a new GLES2 demo project named 'CoolNewDemo'

1. Make sure that you performed the environment setup.
2. Change the directory to the GLES2 sample directory: `cd DemoApps/GLES2`.
3. Create the project template using the FslBuildNew.py.py script: `FslBuildNew.py GLES2 CoolNewDemo`.
4. Change the directory to the newly created project folder 'CoolNewDemo': `cd CoolNewDemo`.
5. Compile the project: `FslBuild.py` .

Note:

Once a build has been done, you can just invoke the make file directly. However, this requires that you didn't change any dependencies or add files.

To do this, run `make -j 2`.

If you add source files to a project or change the `Fsl.gen` file, run the `FslBuildGen.py` script in the project root folder to regenerate the various build files or just make sure you always use the `FslBuild.py` script as it automatically adds files and regenerate build files as needed.

17.10.6 Notes

17.10.6.1 Manual environment setup

1. Configure your FSL_GRAPHICS_SDK to point to the downloaded SDK without the ending backslash:

```
export FSL_GRAPHICS_SDK=~/fsl/YourDemoFrameworkFolder
```

2. For easy access to the python scripts (not required for building):

```
PATH=$PATH:$FSL_GRAPHICS_SDK/.Config
```

17.10.6.2 Override platform auto-detection

To override the platform auto detection code set the following variable:

```
export FSL_PLATFORM_NAME=Ubuntu
```

17.10.6.3 Executable location

The final executable will be placed in the root of the demo application folder. If it is moved the content folder (if it exist) needs to be copied to the same location.

17.11 Windows build guide

To read text version of this document, see the demo framework `Doc\Setup_guide_windows.md` for more detailed information.

17.11.1 Prerequisites

- Read Section 17.8 to know about the custom build system.
- Visual Studio 2017 (community edition or better)
- Python 3.5.x or newer
  - For 64-bit Windows OS
- An OpenGL ES 2+ emulator
  - Arm Mali OpenGL ES Emulator 3.0.2.g694a9 (64 bit)
    - Use the exact version (64 bit) and use the installer to install it to the default location!
  - Vivante OpenGL ES Emulator

To get started its recommended to utilize the Arm Mali OpenGL ES 3.0.2 emulator (64 bit) which this guide will assume you are using.

17.11.2 Environment setup

1. Start a Windows console (cmd.exe) in the DemoFramework folder.
2. Run the visual studio ```vcvarsall.bat x64``` to prepare your command line compiler environment for x64 compilation.
   For VS2017, it is often located here:
   ```
   "C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build\vcvarsall.bat" x64
   ```
3. Run the `prepare.bat` file located in the root of the framework folder to configure the necessary environment variables and paths. Please beware that the `prepare.bat` file requires the current working directory to be the root of your demoframework folder to function (which is also the folder it resides in).

17.11.3 Compiling and running an existing sample application

In this example, we use the GLES2 S06_Texturing application.

1. Make sure that you have performed the environment setup.
2. Change directory to the sample directory: `cd DemoApps\GLES2\S06_Texturing`
3. Generate the build files: `FslBuildGen.py`.
4. Launch visual studio using the Arm Mali Emulator: `.StartProject.bat arm`.
5. Compile and run the project (The default is to press F5).

To use a different emulator, the `.StartProject.bat` file can be launched with the following arguments

arm	Arm mali emulator
powervr	Powervr emulator
qualcomm	Qualcomm andreno adreno emulator (expects its installed in "c:\AdrenoSDK”)
vivante	Vivante emulator

If it is launched without an argument, it defaults to the Arm emulator.

17.11.4 Creating a new GLES2 demo project named 'CoolNewDemo'

1. Make sure that you have performed the environment setup.
2. Change directory to the GLES2 sample directory: `cd DemoApps/GLES2`.
3. Create the project template using the `FslBuildNew.py` script: `FslBuildNew.py GLES2 CoolNewDemo`.
4. Change directory to the newly created project folder 'CoolNewDemo': `cd CoolNewDemo`.
5. Generate build files for Android, Ubuntu, and Yocto (this step will be simplified soon): `FslBuildGen.py`.
6. Launch visual studio using the Arm Mali Emulator: `.StartProject.bat arm`.
7. Compile and run the project (the default is to press F5) or start creating your new demo.

If you add source files to a project or change the `Fsl.gen` file, run the `FslBuildGen.py` script in the project root folder to regenerate the various build files.

17.11.5 Notes

17.11.5.1 Switching between emulators

The visual studio projects have been configured so that emulator builds can co-exist without interfering with each other. Furthermore, the only the emulator dependent parts will be rebuild when changing emulator.

Therefore, it is very fast to switch between emulators.

17.11.5.2 Executable location

The executable location is based on the build type release/debug and which emulator you are using. Therefore, the executable for a demo called S06_Texturing build as debug and using the Arm emulator will be located under

```
bin\S06_Texturing\Debug_ARM\
```

The content folder is located at:

`Content`

To move them, make sure that both the `S06_Texturing.exe` and `Content` folder is moved to the same location like this:

```
S06_Texturing.exe
Content
```

17.12 Yocto build guide

First you need to decide how you are going to be building for Yocto.

- Building using a prebuild Yocto SDK
- Building using a full Yocto build

To read text version of this document, see the demo framework `Doc/Setup_guide_yocto.md` for more detailed information.

17.12.1 Building using a prebuild Yocto SDK

Building using a prebuild Yocto SDK and a prebuild SD card image.

This is the fastest way to get started.

17.12.1.1 Prerequisites

- Read Appendix 2 to know about the custom build system.
- Ubuntu 16.04
- Python 3.5 (this is standard in Ubuntu 16.04)
- A prebuild SDK for your board typically called something like `toolchain.sh`.
- A prebuild sd-card image for your board typically called `BoardName.rootfs.sdcard.bz2`.
- Git: sudo apt-get install git

For this guide we will assume you are using a FB image.

- Download the DemoFramework source using git.

Read the introduction to the FslBuild toolchain in `Doc/FslBuild_toolchain_readme.md`.

17.12.1.2 Preparing a Yocto SDK build

1. Start a terminal (Ctrl+Alt t).

2. Install the SDK:

```
./fsl-imx-internal-xwayland-glibc-x86_64-fsl-image-gui-aarch64-
toolchain-4.9.51-mx8-beta.sh
```

Chose where to install it, you can use the default location or a location of your choice.

For this example, we use `~/sdk/4.9.51-mx8-beta`.

When the setup is complete it will list the configuration script you need to run to configure the sdk environment.

Something like this

```
$ . ~/sdk/4.9.51-mx8-beta/environment-setup-aarch64-poky-linux
```

Each time you wish to use the SDK in a new shell session, you need to source the environment setup script.

3. Your SDK is now installed.

17.12.1.3 Yocto SDK environment setup

1. Start a terminal (Ctrl+Alt t).

2. Prepare the yocto build environment by running the configuration command you got during the SDK installation.

```
. ~/sdk/4.9.51-mx8-beta/environment-setup-aarch64-poky-linux
```

3. You should now be ready to build using the demo framework. However, if you experience issues with the `prepare.sh` script you can help it out by defining the platform name and the location of the root fs.

```
export FSL_PLATFORM_NAME=Yocto
export ROOTFS=~/sdk/4.9.51-mx8-beta/sysroots/aarch64-poky-linux
```

Another possible error you can encounter is that the `FslBuild.py` scripts fail to include the 'typing' library.

This can happen because the SDK comes with a too old Python3 version or an incomplete Python3.5 version.

As a workaround for that you could delete the Python3 binaries from the SDK which will cause it to use the system Python3 version instead.

17.12.1.4 Ready to build

You are now ready to start building Yocto applications using the demo framework.

Continue the guide at "Using the demo framework".

17.12.2 Building using a full Yocto build

Building using a full manually Yocto build.

This process provides the most flexible solution, but it also takes significantly longer to build the initial Yocto SD card and toolchain.

17.12.2.1 Prerequisites

- Read Section 17.8 to know about the custom build system.
- The Ubuntu version required by the BSP release.
- Python 3.4 or newer
  It should be part of the default Ubuntu install.
  If you use 3.4 you need to install the 'typing' library manually so we highly recommended using 3.5 or newer.
  To install the typing library in Python **3.4** run:
  `sudo apt-get install python3-pip`
  `sudo pip3 install typing`
- A working yocto build
  For example, follow one of these:
  – https://github.com/nxp-imx/imx-manifest
  – https://community.nxp.com/docs/DOC-94866

For this guide we will assume you are using a FB image.

- Download the DemoFramework source using git.
- Read the introduction to the FslBuild toolchain at `Doc/FslBuild_toolchain_readme.md`.

17.12.2.2 Preparing a Yocto build

Before you build one of these yocto images you need to do the following:

1. Run the yocto build setup (X11 example).

```
MACHINE=imx6qpsabresd source fsl-setup-release.sh -b build-x11
 -e x11
```

3. Bake

```
bitbake fsl-image-gui
bitbake meta-toolchain
bitbake meta-ide-support
```

You can now build one of the images below (or a custom one).

- x11 yocto image
  Example:

  ```
  MACHINE=imx6qpsabresd source fsl-setup-release.sh -b build-x11
   -e x11
  bitbake fsl-image-gui
  bitbake meta-toolchain
  bbitbake meta-ide-support
  Extracted rootfs
  ```

  We assume your yocto build dir is located at `~/fsl-release-bsp/build-x11` and that the rootfs will be unpacked to `~/unpacked-rootfs/build-x11` and the image is called `fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2` (you will need to locate your image name).

  ```
  runqemu-extract-sdk
  ~/fsl-release-bsp/build-x11/tmp/deploy/images/imx6qpsabresd/
  fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2
  ~/unpacked-rootfs/build-x11
  ```

- FB yocto image
  Example:

  ```
  MACHINE=imx6qpsabresd source fsl-setup-release.sh -b build-fb
   -e fb
  bitbake fsl-image-gui
  bitbake meta-toolchain
  bitbake meta-ide-support
  Extracted rootfs
  ```

  We assume your yocto build dir is located at `~/fsl-release-bsp/build-fb` and that the rootfs will be unpacked to `~/unpacked-rootfs/build-fb` and the image is called `fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2` (you will need to locate your image name).

  ```
  runqemu-extract-sdk
  ~/fsl-release-bsp/build-fb/tmp/deploy/images/imx6qpsabresd/
  fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2
  ~/unpacked-rootfs/build-fb
  ```

- Wayland yocto image

Example:

```
MACHINE=imx6qpsabresd source fsl-setup-release.sh -b build-
wayland -e wayland
bitbake fsl-image-gui
bitbake meta-toolchain
bitbake meta-ide-support
Extracted rootfs
```

We assume your yocto build dir is located at `~/fsl-release-bsp/build-wayland` and that the rootfs will be unpacked to `~/unpacked-rootfs/build-wayland` and the image is called `fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2` (you will need to locate your image name).

```
runqemu-extract-sdk
~/fsl-release-bsp/build-wayland/tmp/deploy/images/
imx6qpsabresd/fsl-image-gui-imx6qpsabresd.rootfs.tar.bz2
~/unpacked-rootfs/build-wayland
```

For this guide we will assume you are using an FB image.

17.12.2.3 Yocto environment setup

Prepare the yocto build environment:

```
pushd ~/fsl-release-bsp/build-fb/tmp
source environment-setup-cortexa9hf-neon-poky-linux-gnueabi
export ROOTFS=~/unpacked-rootfs/build-fb
export FSL_PLATFORM_NAME=Yocto
popd
```

17.12.2.4 Ready to build

You are now ready to start building Yocto applications using the demo framework.

Continue the guide at “Using the demo framework”.

17.12.3 Using the demo framework

1. Make sure that you performed the Yocto environment setup for your chosen Yocto environment.
   - SDK build [Yocto SDK environment setup]
   - Custom build [Yocto environment setup].cd to the demoframework folder
2. cd to the demoframework folder.
3. Run the `prepare.sh` file located in the root of the framework folder to configure the necessary environment variables and paths. Please beware that the `prepare.sh` file requires the current working directory to be the root of your demoframework folder to function (which is also the folder it resides in).
   `source prepare.sh`
   also verify that the script detect that you are doing a Yocto build by outputting
   `PlatformName: Yocto`
   If it doesn't you can override the platform auto detection by setting the environment variable
   `export FSL_PLATFORM_NAME=Yocto`
   Before running the `prepare.sh` script.

17.12.4 Compiling all samples

1. Make sure that you have performed the demo framework environment setup.
2. Compile everything `FslBuild.py --Variants [WindowSystem=FB] -t sdk`. `WindowSystem` can be set to either: FB, Wayland, or X11

17.12.5 Compiling and running an existing sample application

In this example, we use the GLES2 S06_Texturing application.

1. Make sure that you have performed the demo framework environment setup.
2. Change directory to the sample directory: `cd DemoApps/GLES2/S06_Texturing`
3. Compile the project: `FslBuild.py --Variants [WindowSystem=FB]`. `WindowSystem` can be set to either: FB, Wayland, or X11.

17.12.6 Creating a new GLES2 demo project named 'CoolNewDemo'

1. Make sure that you have performed the demo framework environment setup.
2. Change directory to the GLES2 sample directory: `cd DemoApps/GLES2`.
3. Create the project template using the `FslBuildNew.py` script: `FslBuildNew.py GLES2 CoolNewDemo`.
4. Change the directory to the newly created project folder 'CoolNewDemo': `cd CoolNewDemo`.
5. Compile the project: `FslBuild.py --Variants [WindowSystem=FB]`. `WindowSystem` can be set to either: FB, Wayland, or X11.

Note:

Once a build has been done, you can just invoke the make file directly. However, this requires that you didn't change any dependencies or add files. To do this, run `make -f GNUmakefile_Yocto -j 2 WindowSystem=FB`.

If you add source files to a project or change the `Fsl.gen` file, run the `FslBuildGen.py` script in the project root folder to regenerate the various build files or just make sure you always use the `FslBuild.py` script as it automatically adds files and regenerate build files as needed.

17.12.7 Notes

17.12.7.1 Manual environment setup

1. Configure your FSL_GRAPHICS_SDK to point to the downloaded SDK without the ending backslash:
   ```
   export FSL_GRAPHICS_SDK=~/fsl/YourDemoFrameworkFolder
   ```
2. For easy access to the python scripts:
   ```
   PATH=$PATH:$FSL_GRAPHICS_SDK/.Config
   ```

17.12.7.2 Override platform auto-detection

To override the platform auto detection code set the following variable:

```
export FSL_PLATFORM_NAME=Yocto
```

17.12.7.3 Building for multiple backends

The makefiles have been configured so that the builds for all backends can co-exist without interfering with each other. In additoin, the only backend dependent parts will be rebuild when changing backend. Therefore, it should be very fast to switch between backends.

The demo application executables will be post fixed with the backend it builds for to ensure no conflict occurs.

17.12.7.4 Executable location

The final executable will be placed in the root of the demo application folder. If it is moved the content folder (if it exist) needs to be copied to the same location.

The executables follows this naming scheme:

```
<DemoAppName>_<BackendName>[<TargetPostFix>]
```

So a debug build of S06_Texturing for the FB backend will be called

```
S06_Texturing_FB_d
```

A release build of S06_Texturing for the X11 backend will be called

```
S06_Texturing_X11
```

17.13 FslContentSync.py notes

- Does not copy files that start with a '.' in its file or directory name.
- Does not allow files to contain ".." in its name.
- Do **not** utilize file names that only differ by casing like this:
  - Shader.txt
  - shader.txt
- Due to the android asset packer it’s not recommended to use Unicode file names as they are unsupported by the android tool at the moment.

17.14 Known limitations

17.14.1 General

- Android, Ubuntu, and Windows OpenVG support is considered experimental for this release.
- G2D support is experimental and it’s not recommended to use it yet.

17.14.2 Android OS

Android OS does not handle Unicode file names inside the 'content' folder. So do not utilize Unicode for filenames stored in Content. The culprit is the android assets folder which we utilize for content files.

17.14.3 Ubuntu

- OpenGLES3 is currently unsupported on Ubuntu, as we rely on the Mesa 3D graphics library for OpenGLES emulation.
- OpenVG is emulated via the Mesa 3D graphics library and it might contain unsupported features.

17.14.4 Windows

- OpenVG is emulated via the Mesa 3D graphics library and it might contain unsupported features.

17.15 Upgrading samples from earlier SDKs

To convert a sample to the latest SDK, start at the SDK version you are using and upgrade the application one step at a time. So a 2.0 application needs to be updated to 2.1 before it can be updated to 2.2.

17.15.1 From 2.0 to 2.1

Since version 2.1 contains minor incompatibilities with 2.0, any existing application will have to be upgraded. The easiest way to upgrade a sample is to rename the old directory, then run

- `FslNewDemoProject.py all -t <type> <name>`
- `cd <name>`
- `FslBuildGen.py`

Then do a two way merge of the old source directory and the new one. If any dependencies were manually added to Fsl.gen in the sample, they will have to be re-added to the new one.

Then run

- `FslBuildGen.py`

The project should now be converted.

17.15.2 From 2.1 to 2.2

V2.1 can easily be upgraded to 2.2. Just run `FslBuildGen.py` to update it.

17.15.3 From 2.2 to 2.3

V2.2 can easily be upgraded to 2.3. Just run `FslBuildGen.py` to update it.

17.16 What’s new

Version 5.1

- All ThirdParty code is now downloaded as needed instead of being included in the repo.
- Windows builds now default to Visual Studio 2017 instead of 2015.
- Basic support for changing the color-space via EGL.
- Examples of how to setup SRGB and HDR framebuffers.

- HDR to LDR display rendering examples with various basic tone-mapping algorithms.
- Vulkan enabled for the Yocto Wayland backend.
- Assimp upgraded to 4.1 on most platforms.
- GLES3.ColorspaceInfo
- GLES3.EquirectangularToCubemap
- GLES3.GammaCorrection demo.
- GLES3.HDR01_BasicToneMapping
- GLES3.HDR02_FBBasicToneMapping
- GLES3.HDR03_SkyboxTonemapping
- GLES3.HDR04_HDRFramebuffer
- GLES3.MultipleViewportsFractalShader demo.
- GLES3.Scissor101
- GLES3.Skybox
- GLES3.SRGBFramebuffer
- GLES3.TextureCompression demo.
- Vulkan.VulkanInfo demo.
- Android build now requires Android Studio 3.1 and the Android NDK16b or newer.

Version 5.0.1

- OpenVX.SoftISP demo.
- OpenCL.SoftISP demo.

Version 5.0

- Tools now require Python 3.4+ instead of python 2.7
- FslBuildNew script that can help you create a new project fast.
- Vulkan support is much closer to its final state.
- The application registration method has been changed so it’s more future proof and allow for greater customization.
- Prebuild binaries have been removed.
  - FslImageConvert.exe was removed as we now support saving screenshots directly in jpg.
  - Prebuild windows libraries removed as we now download and build them on demand instead.
- The directory structure was updated to make it simpler.
- Some tags in Fsl.gen xml files were deprecated.
- Gamepad support.
- New libraries
  - Stb, xinput, perfcounters.

Version 4.0

- First public release on github.
- Early access support for Vulkan, OpenCL, OpenCV and OpenVX.
  - Vulkan samples.
  - OpenVX samples.
  - OpenCL samples.
  - OpenCV samples.
- New libraries
  - GLI 0.8.10, GLM 0.9.7.6
- PixelFormats are now compatible with the vulkan pixel formats.

- FslBuild.py script introduced as a simple unified way to build on all platforms if so desired. It’s still possible to build using the native platform method.
- FslBuild scripts now support limited feature based filtering.
- Introduced a content pipeline to help build vulkan shaders.
- Windows builds
  - Visual Studio 2015 is now the default environment instead of 2013
  - We now use the OpenVG reference implementation to emulate OpenVG.

Version 2.3

- OpenGLES 3.1 support.
- A new ContentMonitor can reload your sample when it detects changes to the content folder (this does not work on Android). This allows for rapid prototyping on most platforms.
- New samples:
  - DFSimpleUI101, ModelLoaderBasics, ModelLoaderViewer, Tessellation101, TessellationSample.
- New libraries:
  - FslAssimp, FslGraphics3D, FslSceneFormat, FslSimpleUI, FslGraphicsGLES3v1
- New experimental UI framework intended to quickly create a UI for your sample app.
- Assimp support on most platforms. It is not supported on Android here we recommend using the FslSceneFormat instead. In general, it will be much more efficient to preprocess your model on a fast platform like a PC and save it in the FslSceneFormat instead of doing it on relatively slow target platform.
- Experimental support for generating Visual Studio 2015 projects (see the FslBuildgen documentation for details).
- Content loader for Binary texture and basic font kerning information.
- Windows PowerVR OpenGLES emulation support.

Version 2.2

- Demo content can now be stored in bmp, png and jpeg format on all platforms.
  - Some platforms support extra formats via the DevIL image library.
- Onscreen performance graph support that can be augmented with custom data.
- Pause and single stepping during demo playback.
- Added infrastructure that allows samples to share a library. See DemoApps/Shared for example libraries.
- Lots of new samples.
  - The Blur, FractalShader, FurShellRendering and DirectMultiSamplingVideoYUV are functional but experimental.
- Experimental G2D support.
- Experimental NativeBatch2D support under 3D api’s. See the DFNativeBatch2D samples for an example of how it works.
- Experimental –mmdc parameter for Yocto builds. If it shows the incorrect information then run mmdc2 before running the sample as it will reset things correctly.

Version 2.1

- OpenVG support.
- OpenVG examples.
- Examples: T3DstressTest for GLES2 + GLES3.

- Most samples were upgraded to use the Content system to load their shaders and graphics.
- All samples now support the following arguments:
  – –LogStats = Log basic rendering stats
  – –ScreenshotFrequency <frequency> = Create a screenshot at the given frame frequency (Not supported for OpenVG).

18 Environment Variables Summary

The table below lists the environment variables (ENV) available in the GPU drivers.

The use of most environment variables remains static from driver version to driver version, but sometimes these variables need refinements to meet new, advanced conditions not present with the ENV initially introduced.

18.1 Environment variable for drivers and HAL

Table 64. Environment variables for drivers and HAL

ENV name	Backends supported	Note
FB_IGNORE_ DISPLAY_SIZE	FB/WLD	0: Clip window to device display size. 1: Do not clip window to the device limits for width and height.
FB_MULTI_BUFFER	FB/WLD	Number of backend buffers of the framebuffer device. For WLD, define the multibuffer number of Weston.
FB_FRAMEBUFFER_ N	FB/WLD	Define the Nth framebuffer device.
FB_LEGACY	FB	If board doesn’t support drm-fb, ignore this variable. 0: GPU render through drm 1: GPU directly render to framebuffer.
VG_APITIME	FB/WLD/X11	Enable VG API function execution time print.
VIV_MGPU_AFFINITY	FB/WLD/X11	Control the multiple GPUs affinity configuration. Possible value: • Not defined or defined as "0" GPUs work in GPU_COMBINED mode. • 1:0 GPUs work in GPU_INDEPEDNENT mode, GPU0 is used. • 1:1 GPUs work in GPU_INDEPEDNENT mode, GPU1 is used.
VIV_DEBUG	FB/WLD/X11	Define the user debug message level (-MSG_LEVEL: ERROR/WARNING).
VIV_FBO_PREFER_ MEM	FB/WLD/X11	Renderbuffer is not freed after colorbuffer detaches from FBO (GL ES 2.0)
VIV_DISABLE_HZ	FB/WLD/X11	This variable can be specifically enabled for i. mx6d/q to avoid gpu hang with occlusion query in ES30, because of gpu hardware problem HBN1246

Table 64. Environment variables for drivers and HAL*...continued*

ENV name	Backends supported	Note
GPU_VIV_EXT_ RESOLVE	FB/WLD/X11	Enable the external resolve mode (1 by default for FB).
GPU_VIV_DISABLE_ SUPERTILED_ TEXTURE	FB/WLD/X11	Disable supertiled texture (64x64 tiled texture is not used).
GPU_VIV_DISABLE_ CLEAR_FB	FB/WLD/X11	Enable clear buffer when a new Window surface is created.
GPU_VIV_WL_ MULTI_BUFFER	WLD	Define the client multibuffer number.
WL_EGL_SYNC_ SWAP	WLD	0: Use asynchronous swap for better performance by default. 1: Enable synchronous swap with some performance impact.
DRI_IGNORE_ DISPLAY_SIZE/ X_IGNORE_ DISPLAY_SIZE	X11	0: Clip window to device display size. 1: Do not clip window to the device limits for width and height.
__GL_DEV_FB	X11	Set the path for framebuffer device like /dev/fb0.
LIBGL_ALWAYS_ INDIRECT	X11	Make OGL go into indirect mode. All rendering is done by XserverSet.
LIBGL_DEBUG	X11	Print error messages to stderr if LIBGL_DEBUG env var is set. Print information messages to stderr if LIBGL_DEBUG env var is set to “verbose”.
VIV_PROFILE	vProfiler	Enable profiler. Different level results generate different results.
VP_COUNTER_ FILTER	vProfiler	Used to control profile different system resource like memory/CPU time usage.
VP_FRAME_END	vProfiler	When VIV_PROFILE=3, specify the frame to end profiling with vProfiler.
VP_FRAME_NUM	vProfiler	When VIV_PROFILE=1, used to specify the number of frames dumped by vProfiler.
VP_FRAME_START	vProfiler	When VIV_PROFILE=3, specify the frame to start profiling with vProfiler.
VP_OUTPUT	vProfiler	Specify the output file name of vProfiler (default is vprofiler.vpd).
VP_PROCESS_NAME	vProfiler	Choose profiler enable process (This option is only available for Android platform, not available for Linux OS).
VP_SYNC_MODE	vProfiler	Enable [1] or disable [0] the synchronous mode of vProfiler (default is synchronous enabled).
VP_USE_GLFINISH	vProfiler	Use glFinish as the frameEnd.
VIV_TRACE	vTracer	Enable tracer. Different levels could generate different logs.

18.2 Environment variable for compiler

Table 65. Environment variables for compiler

ENV NAME	Compiler	Note
VC_DUMP_SHADER_ SOURCE	GLSLC/ VSC	Enable dumping the shader source code.

19 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2019 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision History

Revision history

Revision number	Date	Substantive changes
1	11/2018	Updated Chapter "OpenCL" with more precise information and also covered latest i.MX products.
2	06/2019	Made some grammatical updates.
3	08/2019	Added the i.MX 8M Nano information.
4	11/2019	Updated the Vivante IDE information.
5	04/2020	Updated for the Linux L5.4.3_2.0.0 and android-10.0.0_ 2.1.0 releases.
6	06/2020	Updated for the Linux L5.4.24-2.1.0 and later release.

Revision history*...continued*

Revision number	Date	Substantive changes
7	12/2020	Updated for the Linux L5.4.70_2.3.0, android-11.0.0_1.0.0, and later release.
7.1	03/2021	Updated Section 13.5.4 “Enabling vProfiler on Linux” as v Profiler no longer requires kernel module parameter, and made abundant changes to context description.
8	06/2021	Updated for the Linux LF5.10.35_2.0.0 and android-11.0.0_ 1.2.1 releases.
8.1	09/2021	Removed the Section "Designing framework of OpenVX", and made minor updates for the Linux LF5.10.52_2.1.0 release.
8.2	10/2021	Added the i.MX 8ULP information to Section 1.1.
8.2	03/2022	Updated the back page (Legal information).
8.3	09/2022	Updated Figure 1 and published the document in the new template.
8.4	10/2022	Some minor updates for the android-12.1.0_1.0.0 release.
8.4.1	12/2022	Updated the VivanteIDE package name in Section 13.3.1.

20 Legal information

20.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

20.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

20.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents