Freescale Semiconductor

Document Number: IMX6GRAPHICUG
Rev. L3.10.31_1.1.0-beta, 08/2014

i.MX 6 Graphics User's Guide

Contents

Chapter 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.

Chapter 2 i.MX 6 G2D API

2.1 Overview

G2D API (Application Programming Interface) is designed for the purposes of easy to understand and use 2D BLT function. It allows the user to implement the customized applications with the simple interfaces. It is hardware and platform independent for i.MX 6 2D Graphics.
G2D API supports the following features but not limited:

- Simple BLT operation from source to destination
- Alpha blend 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 blend for source only
- Asynchronous mode and sync
- Contiguous memory allocator
- Support VG engine

G2D API document include the detailed interface description, and sample code for reference.
The API is designed with C-Style and can be used in both C and C++ application.

2.2 Enumerations and structures

This chapter describes all Enumeration and Structure definition in G2D.

2.2.1 g2d_format enumeration

The Enumeration describes the pixel format for source and destination

Name	Numeric	Description
G2D_RGB565	0	RGB565 pixel format
G2D_RGBA8888	1	32bit-RGBA pixel format
G2D_RGBX8888	2	32bit-RGBX without alpha
G2D_BGRA8888	3	32bit-BGRA pixel format
G2D_BGRX8888	4	32bit-BGRX without alpha

G2D_BGR565	5	16bit-BGR565 pixel format
G2D_ARGBA8888	6	32bit-ARGB pixel format
G2D_ABGR8888	7	32bit-ABGR pixel format
G2D_XRGB8888	8	32bit-XRGB without alpha
G2D_XBGR8888	9	32bit-XBGR without alpha
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
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

The Enumeration describes the blend factor for source and destination

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

2.2.3 g2d_cap_mode enumeration

The Enumeration describes the alternative capability in 2D BLT

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

Note: G2D_GLOBAL_ALPHA is only valid when G2D_BLEND is enabled.

2.2.4 g2d_rotation enumeration

The Enumeration describes the rotation mode in 2D BLT

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

The Enumeration describes the cache operation mode

Name	Numeric	Description
G2D_CACHE_CLEAN	0	Clean the cacheable buffer
G2D_CACHE_FLUSH	1	Clean and invalidate cacheable buffer
G2D_GLOBAL_INVALIDATE	2	Invalidate the cacheable buffer

2.2.6 g2d_hardware_type enumeration

The Enumeration describes the supported hardware type

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

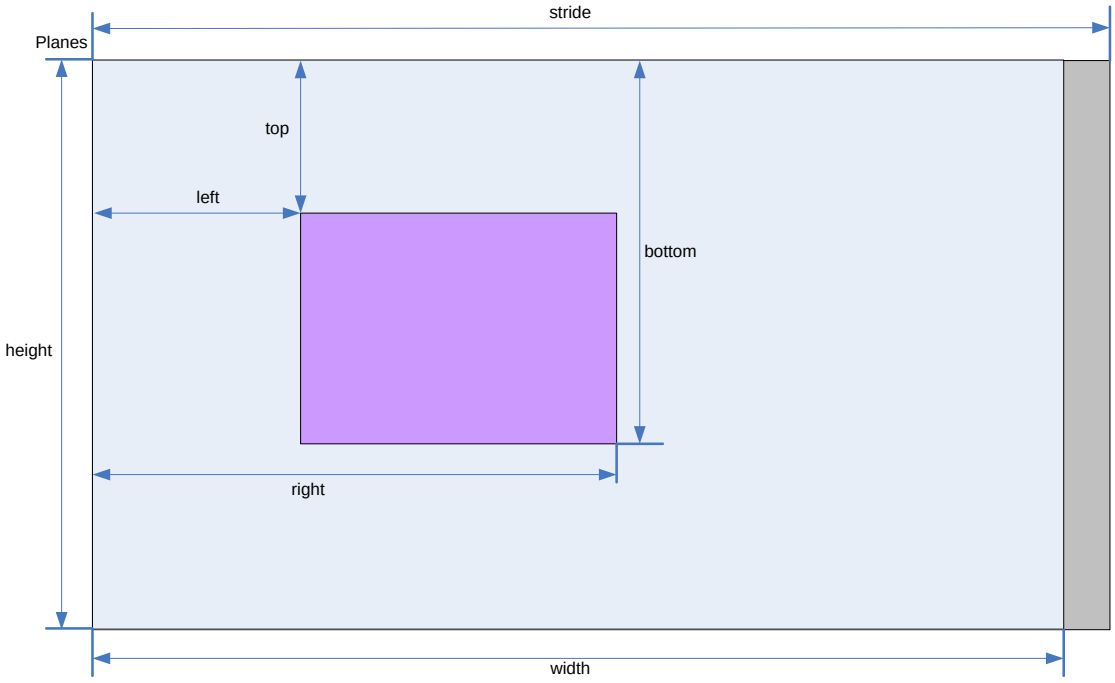
2.2.7 g2d_surface structure

The Structure describes the surface with operation attributes

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
global_alpha	int	Global alpha value 0~255
clrcolor	int	Clear color is 32bit RGBA
rot	g2d_rotation	Rotation mode

Notes:

- RGB and YUV formats can be set in source surface, but only RGB format can be set in destination surface.
- RGB pixel buffer only uses planes [0], buffer address is with 16bytes alignment,
- 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 16bytes for source and destination surface,
- NV12 stride alignment is 8bytes for source surface, UV stride = Y stride,
- I420 stride alignment is 8bytes 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 below:



2.2.8 g2d_buf structure

The Structure describes the buffer used as g2d interfaces.

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

2.3 G2D function descriptions

2.3.1 g2d_open

Description:
Open 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, default is G2D_HARDWARE_2D

Syntax:
int g2d_make_current(void *handle, enum g2d_hardware_type type);

Parameters:
handle g2d device handle
type g2d hardware type

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:
Enable 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 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

2.4 Sample codes for G2D API usage

This chapter provides the brief prototype codes with G2D API.

2.4.1 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);
```

2.4.2 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);
```

2.4.3 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);
```

Chapter 3 i.MX 6 EGL & OGL Extension Support

3.1 Introduction

The following tables list the level of support for EGL and OES extensions available with i.MX 6 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 http://www.khronos.org/registry/egl/ as of 1/24/2013)

EGL Extension Number, Name, and hyperlink	Supported?
1. EGL_KHR_config_attribs	No
2. EGL_KHR_lock_surface	Yes
3. EGL_KHR_image	Yes
4. EGL_KHR_vg_parent_image	No
5. EGL_KHR_gl_texture_2D_image	Yes
EGL_KHR_gl_texture_cubemap_image	Yes
EGL_KHR_gl_texture_3D_image	No
EGL_KHR_gl_renderbuffer_image	Yes
6. EGL_KHR_reusable_sync	Yes
8. EGL_KHR_image_base	Yes
9. EGL_KHR_image_pixmap	Yes
10. EGL_IMG_context_priority	No
16. EGL_KHR_lock_surface2	No
17. EGL_NV_coverage_sample	No
18. EGL_NV_depth_nonlinear	No
19. EGL_NV_sync	No
20. EGL_KHR_fence_sync	Yes
24. EGL_HI_clientpixmap	No
25. EGL_HI_colorformats	No
26. EGL_MESA_drm_image	No
27. EGL_NV_post_sub_buffer	No
28. EGL_ANGLE_query_surface_pointer	No
29. EGL_ANGLE_surface_d3d_texture_2d_share_handle	No
30. EGL_NV_coverage_sample_resolve	No
31. EGL_NV_system_time	No
32. EGL_KHR_stream	No
33. EGL_KHR_stream_consumer_gltexture	No
34. EGL_KHR_stream_producer_eglsurface	No
35. EGL_KHR_stream_producer_aldatalocator	No
36. EGL_KHR_stream_fifo	No
37. EGL_EXT_create_context_robustness	Yes
38. EGL_ANGLE_d3d_share_handle_client_buffer	No
39. EGL_KHR_create_context	Yes
40. EGL_KHR_surfaceless_context	No
41. EGL_KHR_stream_cross_process_fd	No
42. EGL_EXT_multiview_window	No
43. EGL_KHR_wait_sync	No
44. EGL_NV_post_convert_rounding	No
45. EGL_NV_native_query	No
46. EGL_NV_3dvision_surface	No
47. EGL_ANDROID_framebuffer_target	No

48. EGL_ANDROID_blob_cache	No
49. EGL_ANDROID_image_native_buffer	Yes
50. EGL_ANDROID_native_fence_sync	Yes
51. EGL_ANDROID_recordable	No
52. EGL_EXT_buffer_age	Yes
53. EGL_EXT_image_dma_buf_import	No
54. EGL_ARM_pixmap_multisample_discard	No
55. EGL_EXT_swap_buffers_with_damage	No
56. EGL_NV_stream_sync	No
57. EGL_EXT_platform_base	No
58. EGL_EXT_client_extensions	No
59. EGL_EXT_platform_x11	No
60. EGL_KHR_cl_event	No
61. EGL_KHR_get_all_proc_addresses	No
EGL_KHR_client_get_all_proc_addresses	No
62. EGL_MESA_platform_gbm	No
63. EGL_EXT_platform_wayland	No
64. EGL_KHR_lock_surface3	No
65. EGL_KHR_cl_event2	No
66. EGL_KHR_gl_colorspace	No
67. EGL_ANDROID_get_render_buffer	Yes
68. EGL_ANDROID_swap_rectangle	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 http://www.khronos.org/registry/gles/ as of 9/27/2012)

Extension Number, Name and hyperlink	**ES1.1**	**ES2.0/3.0**
1. GL_OES_blend_equation_separate	Yes	N/A
2. GL_OES_blend_func_separate	Yes	N/A
3. GL_OES_blend_subtract	Yes	N/A
4. GL_OES_byte_coordinates	Yes	N/A
5. GL_OES_compressed_ETC1_RGB8_texture	Yes	Yes
6. GL_OES_compressed_paletted_texture	Yes	Yes
7. GL_OES_draw_texture	Yes	N/A
8. GL_OES_extended_matrix_palette	Yes	No
9. GL_OES_fixed_point	Yes	No
10. GL_OES_framebuffer_object	Yes	N/A
11. GL_OES_matrix_get	Yes	N/A
12. GL_OES_matrix_palette	Yes	N/A
14. GL_OES_point_size_array	Yes	No
15. GL_OES_point_sprite	Yes	No
16. GL_OES_query_matrix	Yes	N/A
17. GL_OES_read_format	Yes	No

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0
18. GL_OES_single_precision	Yes	No
19. GL_OES_stencil_wrap	Yes	No
20. GL_OES_texture_cube_map	Yes	N/A
21. GL_OES_texture_env_crossbar	No	No
22. GL_OES_texture_mirrored_repeat	Yes	N/A
23. GL_OES_EGL_image	Yes	Yes
24. GL_OES_depth24	Yes	Yes
25. GL_OES_depth32	No	No
26. GL_OES_element_index_uint	Yes	Yes
27. GL_OES_fbo_render_mipmap	Yes	Yes
28. GL_OES_fragment_precision_high	No	Yes
29. GL_OES_mapbuffer	Yes	Yes
30. GL_OES_rgb8_rgba8	Yes	Yes
31. GL_OES_stencil1	Yes	Yes
32. GL_OES_stencil4	Yes	Yes
33. GL_OES_stencil8	Yes	N/A
34. GL_OES_texture_3D	No	No
35. GL_OES_texture_float_linear GL_OES_texture_half_float_linear	No No	No No
36. GL_OES_texture_float GL_OES_texture_half_float	No No	No No
37. GL_OES_texture_npot	Yes	Yes
38. GL_OES_vertex_half_float	Yes	Yes
39. GL_AMD_compressed_3DC_texture	No	No
40. GL_AMD_compressed_ATC_texture	No	No
41. GL_EXT_texture_filter_anisotropic	Yes	Yes
42. GL_EXT_texture_type_2_10_10_10_REV	No	Yes
43. GL_OES_depth_texture	No	Yes
44. GL_OES_packed_depth_stencil	Yes	Yes
45. GL_OES_standard_derivatives	No	Yes
46. GL_OES_vertex_type_10_10_10_2	No	Yes
47. GL_OES_get_program_binary	No	Yes
48. GL_AMD_program_binary_Z400	No	No
49. GL_EXT_texture_compression_dxt1		
50. GL_AMD_performance_monitor	No	No
51. GL_EXT_texture_format_BGRA8888	Yes	Yes
52. GL_NV_fence	No	No
53. GL_IMG_read_format	No	No
54. GL_IMG_texture_compression_pvrtc	No	No
55. GL_QCOM_driver_control	No	No
56. GL_QCOM_performance_monitor_global_mode	No	No
57. GL_IMG_user_clip_plane	No	No
58. GL_IMG_texture_env_enhanced_fixed_function	No	No
59. GL_APPLE_texture_2D_limited_npot	No	No
60. GL_EXT_texture_lod_bias	Yes	N/A
61. GL_QCOM_writeonly_rendering	No	No

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0
62. GL_QCOM_extended_get	No	No
63. GL_QCOM_extended_get2	No	No
64. GL_EXT_discard_framebuffer	No	Yes
65. GL_EXT_blend_minmax	Yes	Yes
66. GL_EXT_read_format_bgra	Yes	Yes
67. GL_IMG_program_binary	No	No
68. GL_IMG_shader_binary	No	No
69. GL_EXT_multi_draw_arrays GL_SUN_multi_draw_arrays	Yes No	Yes No
70. GL_QCOM_tiled_rendering	No	No
71. GL_OES_vertex_array_object	No	No
72. GL_NV_coverage_sample	No	No
73. GL_NV_depth_nonlinear	No	No
74. GL_IMG_multisampled_render_to_texture	No	No
75. GL_OES_EGL_sync	Yes	N/A
76. GL_APPLE_rgb_422	No	No
77. GL_EXT_shader_texture_lod	No	No
78. GL_APPLE_framebuffer_multisample	No	No
79. GL_APPLE_texture_format_BGRA8888	No	No
80. GL_APPLE_texture_max_level	No	No
81. GL_ARM_mali_shader_binary	No	No
82. GL_ARM_rgba8	No	No
83. GL_ANGLE_framebuffer_blit	No	No
84. GL_ANGLE_framebuffer_multisample	No	No
85. GL_VIV_shader_binary	No	Yes
86. GL_EXT_frag_depth	No	Yes
87. GL_OES_EGL_image_external	Yes	Yes
88. GL_DMP_shader_binary	No	No
89. GL_QCOM_alpha_test	No	No
90. GL_EXT_unpack_subimage	No	N/A
91. GL_NV_draw_buffers	No	No
92. GL_NV_fbo_color_attachments	No	No
93. GL_NV_read_buffer	No	No
94. GL_NV_read_depth_stencil	No	No
95. GL_NV_texture_compression_s3tc_update	No	No
96. GL_NV_texture_npot_2D_mipmap	No	No
97. GL_EXT_color_buffer_half_float	No	No
98. GL_EXT_debug_label	No	No
99. GL_EXT_debug_marker	No	No
100. GL_EXT_occlusion_query_boolean	No	No
101. GL_EXT_separate_shader_objects	No	No
102. GL_EXT_shadow_samplers	No	No
103. GL_EXT_texture_rg	No	No
104. GL_NV_EGL_stream_consumer_external	No	No
105. GL_EXT_sRGB	No	No
106. GL_EXT_multisampled_render_to_texture	No	Yes

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0
107. GL_EXT_robustness	No	Yes
108. GL_EXT_texture_storage	No	No
109. GL_ANGLE_instanced_arrays	No	No
110. GL_ANGLE_pack_reverse_row_order	No	No
111. GL_ANGLE_texture_compression_dxt3	No	No
GL_ANGLE_texture_compression_dxt5	No	No
112. GL_ANGLE_texture_usage	No	No
113. GL_ANGLE_translated_shader_source	No	No
114. GL_FJ_shader_binary_GCCSO	No	No
115. GL_OES_required_internalformat	No	No
116. GL_OES_surfaceless_context	No	No
117. GL_KHR_texture_compression_astc_ldr	No	No
118. GL_KHR_debug	No	No
119. GL_QCOM_binning_control	No	No
120. GL_ARM_mali_program_binary	No	No
121. GL_EXT_map_buffer_range	No	No
122. GL_EXT_shader_framebuffer_fetch	No	No
123. GL_APPLE_copy_texture_levels	No	No
124. GL_APPLE_sync	No	No
125. GL_EXT_multiview_draw_buffers	No	No
126. GL_NV_draw_texture	No	No
127. GL_NV_packed_float	No	No
128. GL_NV_texture_compression_s3tc	No	No
129. GL_NV_3dvision_settings	No	No
130. GL_NV_texture_compression_latc	No	No
131. GL_NV_platform_binary	No	No
132. GL_NV_pack_subimage	No	No
133. GL_NV_texture_array	No	No
134. GL_NV_pixel_buffer_object	No	No
135. GL_NV_bgr	No	No
136. GL_OES_depth_texture_cube_map	No	No
137. GL_EXT_color_buffer_float	No	No
138. GL_ANGLE_depth_texture	No	No
139. GL_ANGLE_program_binary	No	No
140. GL_IMG_texture_compression_pvrtc2	No	No
141. GL_NV_draw_instanced	No	No
142. GL_NV_framebuffer_blit	No	No
143. GL_NV_framebuffer_multisample	No	No
144. GL_NV_generate_mipmap_sRGB	No	No
145. GL_NV_instanced_arrays	No	No
146. GL_NV_shadow_samplers_array	No	No

Extension Number, Name and hyperlink	ES1.1	ES2.0/3.0
147. GL_NV_shadow_samplers_cube	No	No
148. GL_NV_sRGB_formats	No	No
149. GL_NV_texture_border_clamp	No	No
150. GL_VIV_direct_texture	Yes	Yes
151. GL_EXT_disjoint_timer_query	No	No
152. GL_EXT_draw_buffers	No	No
153. GL_EXT_texture_sRGB_decode	No	No
154. GL_EXT_sRGB_write_control	No	No
155. GL_EXT_texture_compression_s3tc	No	No
156. GL_EXT_pvrtc_sRGB	No	No
157. GL_EXT_instanced_arrays	No	No
158. GL_EXT_draw_instanced	No	No
159. GL_NV_copy_buffer	No	No
160. GL_NV_explicit_attrib_location	No	No
161. GL_NV_non_square_matrices	No	No
162. GL_EXT_shader_integer_mix	No	No
163. GL_OES_texture_compression_astc	No	No
164. GL_NV_blend_equation_advanced GL_NV_blend_equation_advanced_coherent	No No	No No
165. GL_INTEL_performance_query	No	No
166. GL_VIV_texture_border_clamp	No	No

3.4 Extension GL_VIV_direct_texture

Name

VIV_direct_texture

Name strings

GL_VIV_direct_texture

IPStatus

Contact Freescale 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 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.

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 will remain a single-LOD texture.

Format — You can 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) will be 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 you draw 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, you can modify Logical and Physical data.
Then, call `glTexDirectInvalidateVIV` to apply the texture before you draw 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, &24hysical);
… …
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: http://www.opengl.org/registry/specs/ARB/texture_border_clamp.txt. See also vendor extension GL_SGIS_texture_border_clamp, http://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."

Chapter 4 i.MX 6 Framebuffer API

4.1 Overview

The graphics software includes i.MX 6 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 6 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 http://www.khronos.org/registry/egl. The API described in this document is compatible with EGL version 1.4 of the specification.

This API document is current as of the software version specified in the document revision history. The following platforms are supported:

- Linux /X11
- Android
- Windows Embedded Compact

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;
```

Table 1. Types as listed on EGL 1.4 API Quick Reference Card

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

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:

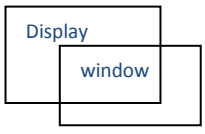
Display	*EGLNativeDisplayType
Pixmap	EGLNativePixmapType
Window	EGLNativeWindowType

Android platform:

ANativeWindow*	EGLNativeWindowType

4.2.2 Environment variables

Table 2. i.MX 6 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. Maximum is 3. **Recommended values: 2 or 3.** The FB_MULTI_BUFFER variable can be set to any positive integer value. • If set to 1, the multiple buffer function is not enabled. • If set to 2 or 3, the driver will run as users expect. • If set to a value more than 3, the driver will use 3 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 will try 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 will not be reduced to fit within the display. Global. **Allowed values: any positive integer.** Note: The drivers will 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, then the driver when creating window will use 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). 

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 will try 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 respectively 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:

```
void
fbDestroyDisplay (
      EGLNativeDisplayType      Display
);
```

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 will use 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 will reduce 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:

```
EGLNativeWindowType
fbCreateWindow (
      EGLNativeDisplayType   Display,
      int                    X,
      int                    Y,
      int                    Width,
      int                    Height
);
```

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 will fail 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 will fail 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**.

Chapter 5 Freescale XServer Video Driver

5.1 EXA driver

XServer video driver is designed to help XServer to render desktop onto a screen. It manages the display driver, and provides rendering acceleration and other display features, such as rotation and multiple display methods. Freescale video driver implements XServer's EXcellent Architecture (EXA).

5.1.1 EXA driver options

These options are used in the configuration file `/etc/X11/xorg.conf`:

```
Section "Device"
    Identifier  "i.MX Accelerated Framebuffer Device"
    Driver      "vivante"
    Option      "fbdev"     "/dev/fb1"
    Option      "vivante_fbdev" "/dev/fb1"
    Option      "SyncDraw"  "false"
EndSection
```

Option	Meaning	Default Value	Comment
ShadowFB	Whether to enable the shadow frame buffer (FB).	False	*Deprecated technology.* It rotates the FB. If it is enabled, acceleration is disabled.
Rotate	Rotation of FB.	<null>	*Deprecated technology.* It can be CW/CCW/UD. If it is set to one of these values, Shadow FB is automatically enabled. Rotation cannot change after XServer is started.
NoAccel	Disables EXA acceleration.	False	If it is set to **True**, the EXA functions are not accelerated by the GPU.
VivCacheMem	Pixmap created by GPU is generally cacheable.	True	Normal Pixmaps are created cacheable. Special Pixmaps used for EGL are still non-cacheable.
SyncDraw	Wait for the GPU to complete for every single drawing.	False	This will affect the performance if it is set to **True**.

5.1.2 24 bpp pixmap

The GPU can only accelerate a 16 bpp or 32 bpp pixmap. For a 24 bpp screen, a 32 bpp buffer is actually reserved.

5.1.3 Shared pixmap extension

The Shared Pixmap Extension (SHM) pixmap will be described in next release.

5.1.4 How to disable XRandR

For an embedded device that does not support XRandR (for which the memory can be reduced), set "gEnableXRandR" to **False** in vivante_fbdev_driver.c.

5.1.5 Cursor

Freescale hardware IPU does not provide a hardware cursor.

5.1.6 DRI

DRI is designed to accelerate OpenGL rendering. It enables the GPU direct render to the on-screen buffer. Due to the lack of hard cursor support, and because often the window location is not well aligned, the GPU cannot render to screen directly. Therefore, DRI is not fully used.
DRI is supported in this video driver. DRI2 or DRI3 is not supported.

5.1.7 Tearing

XServer (and early Microsoft Windows) does not support double buffering for the screen. There will be a copy from off-screen buffer to target on-screen area (or direct rendering to on-screen). The operation cannot be completed in the blank time of the display, and the IPU cannot provide an ideal VSYNC signal. Therefore, there will be tearing. To remove tearing, a GLES compositor is needed. This tearing free feature will be described in next release.

5.2 XRandR

This video driver supports XRandR.
The X Resize, Rotate and Reflect Extension (RandR) is an X Window System extension, which allows clients to dynamically resize, rotate, and reflect the root window of a screen (http://en.wikipedia.org/wiki/Xrandr).

5.2.1 Useful commands

If the display supports multiple resolution types, use the following commands for a query:

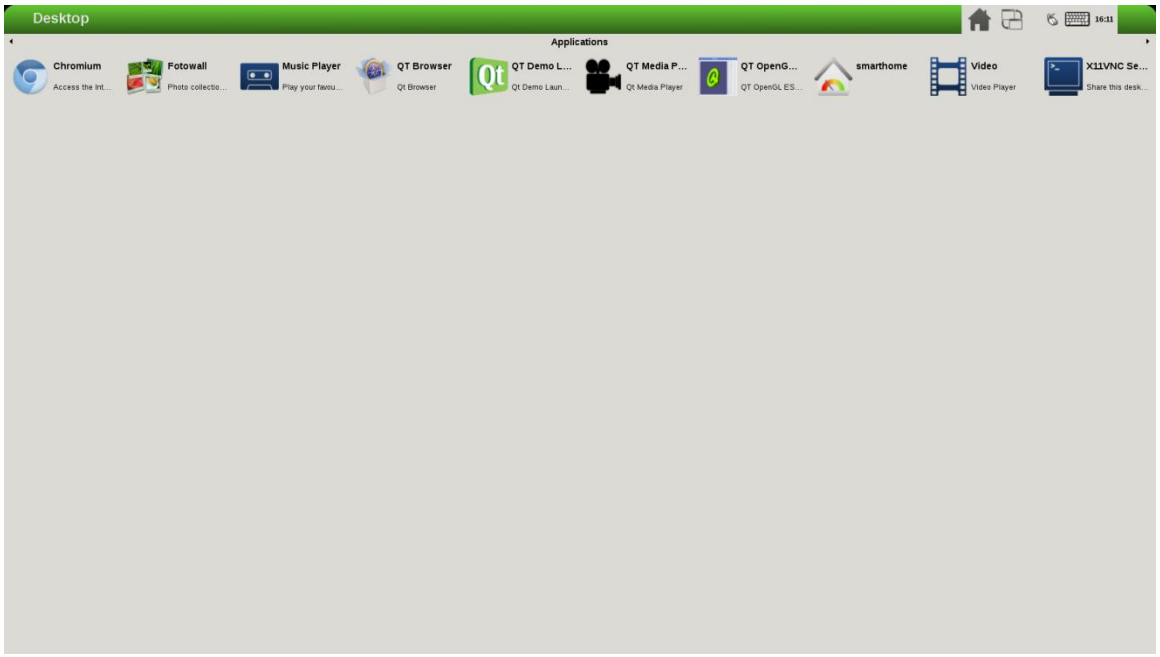
```
root@imx6qsabresd:~# export DISPLAY=:0.0
root@imx6qsabresd:~# xrandr
Screen 0: minimum 240 x 240, current 1920 x 1080, maximum 8192 x 8192
DISP3 BG connected 1920x1080+0+0 (normal left inverted right x axis y axis) 0mm x 0mm
   S:1920x1080p-50   50.0*
   S:1920x1080p-60   60.0
   S:1280x720p-50   50.0
   S:1280x720p-60   60.0
   S:720x576p-50   50.0
   S:720x480p-60   59.9
   V:640x480p-60   60.0
   S:640x480p-60   59.9
```

If using the console serial port for the command line interface, the DISPLAY environment variable is not configured by default and the xrandr command will fail. The solution is to set the DISPLAY environment variable. (Reference: see manpage for X)

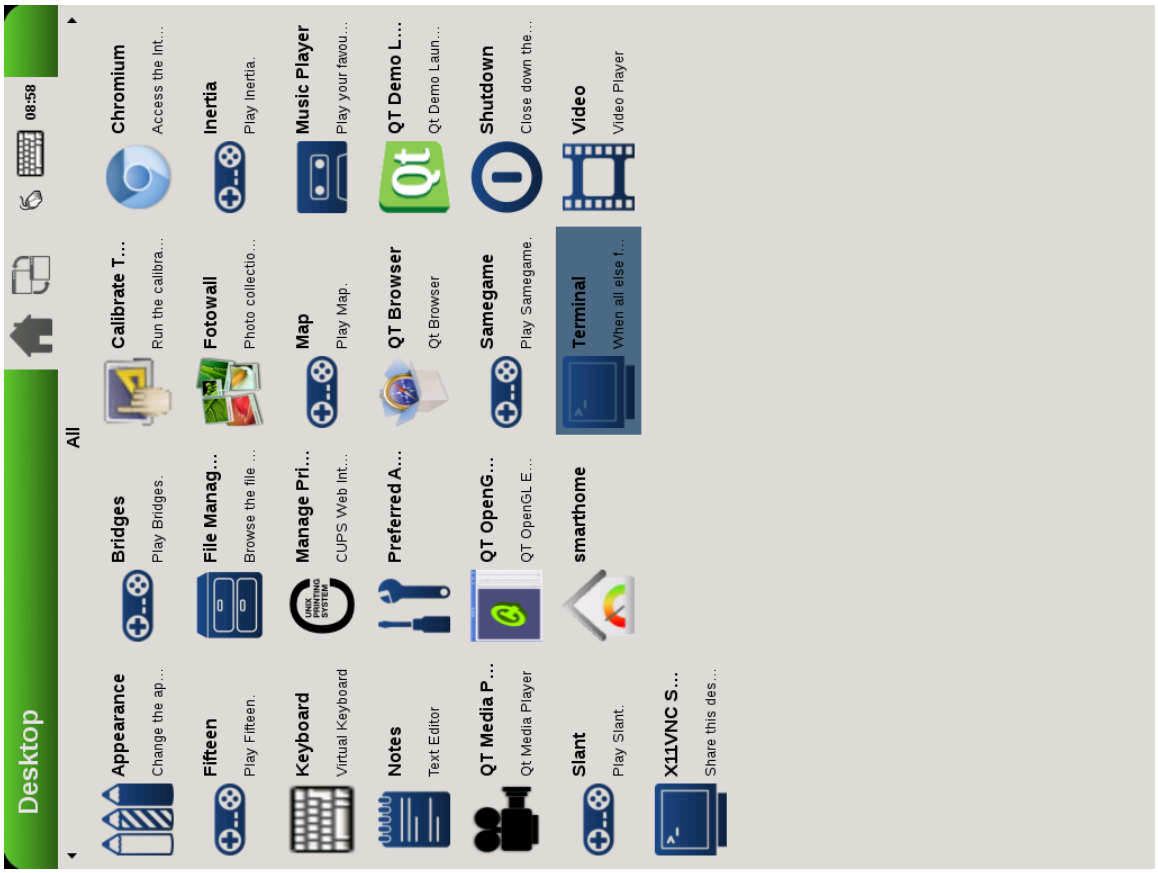
```
root@imx6qsabresd:~# xrandr
Can't open display
root@imx6qsabresd:~# echo $DISPLAY
```

```
root@imx6qsabresd:~# export DISPLAY=:0.0
root@imx6qsabresd:~# xrandr
Screen 0: minimum 240 x 240, current 1024 x 768, maximum 8192 x 8192
DISP4 BG - DI1 connected 1024x768+0+0 (normal left inverted right x axis y axis) 0mm x
0mm
   U:1024x768p-60   60.0*+
```

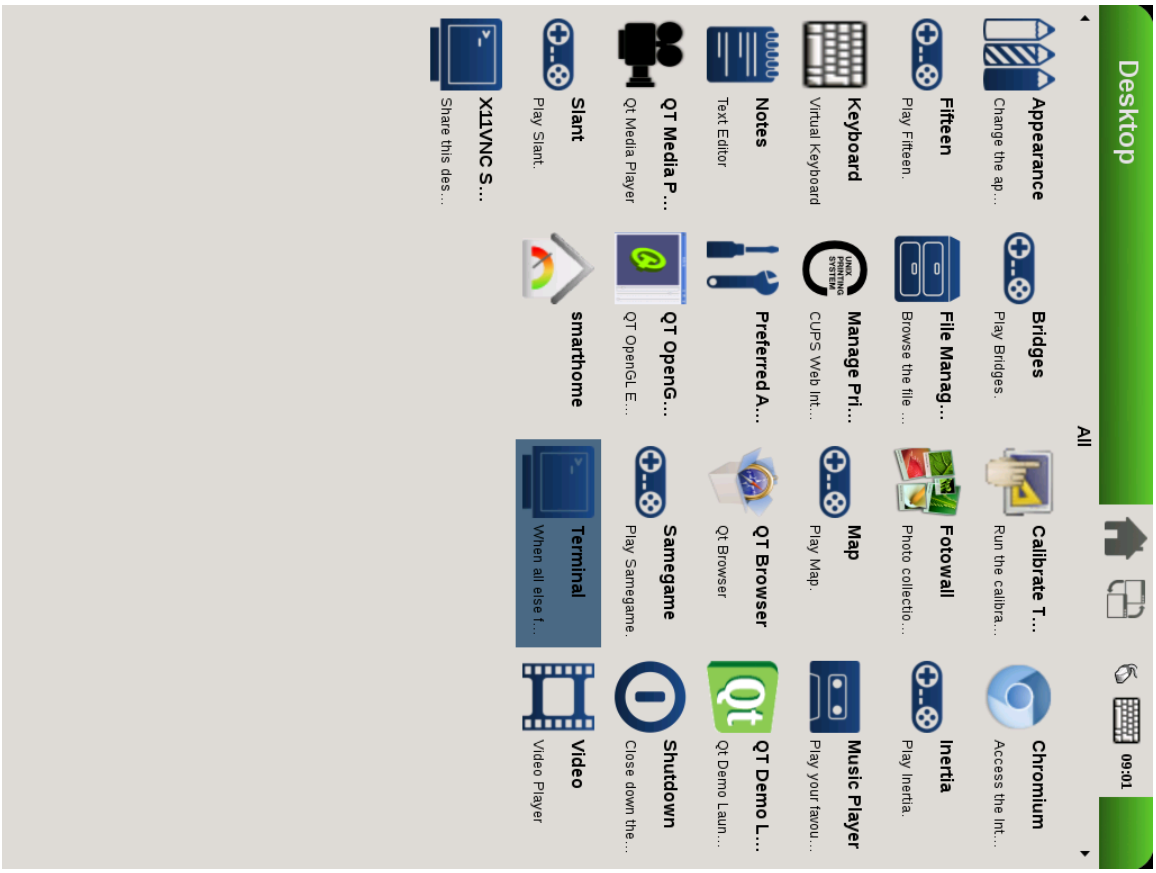
- Change the resolution:
  root@imx6qsabresd:~# xrandr -s 1920x1080



- Rotate the screen:
  root@imx6qsabresd:~# xrandr -o left:

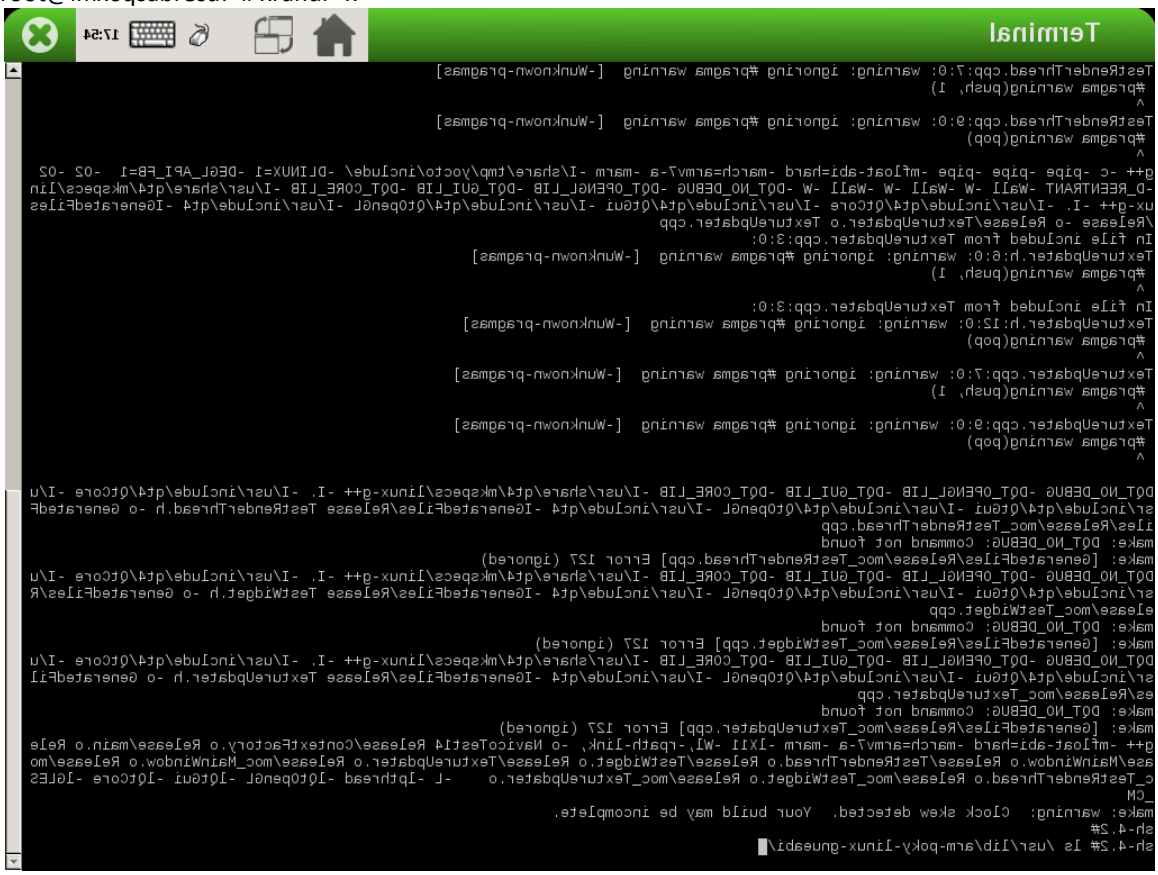


root@imx6qsabresd:~# xrandr -o right:

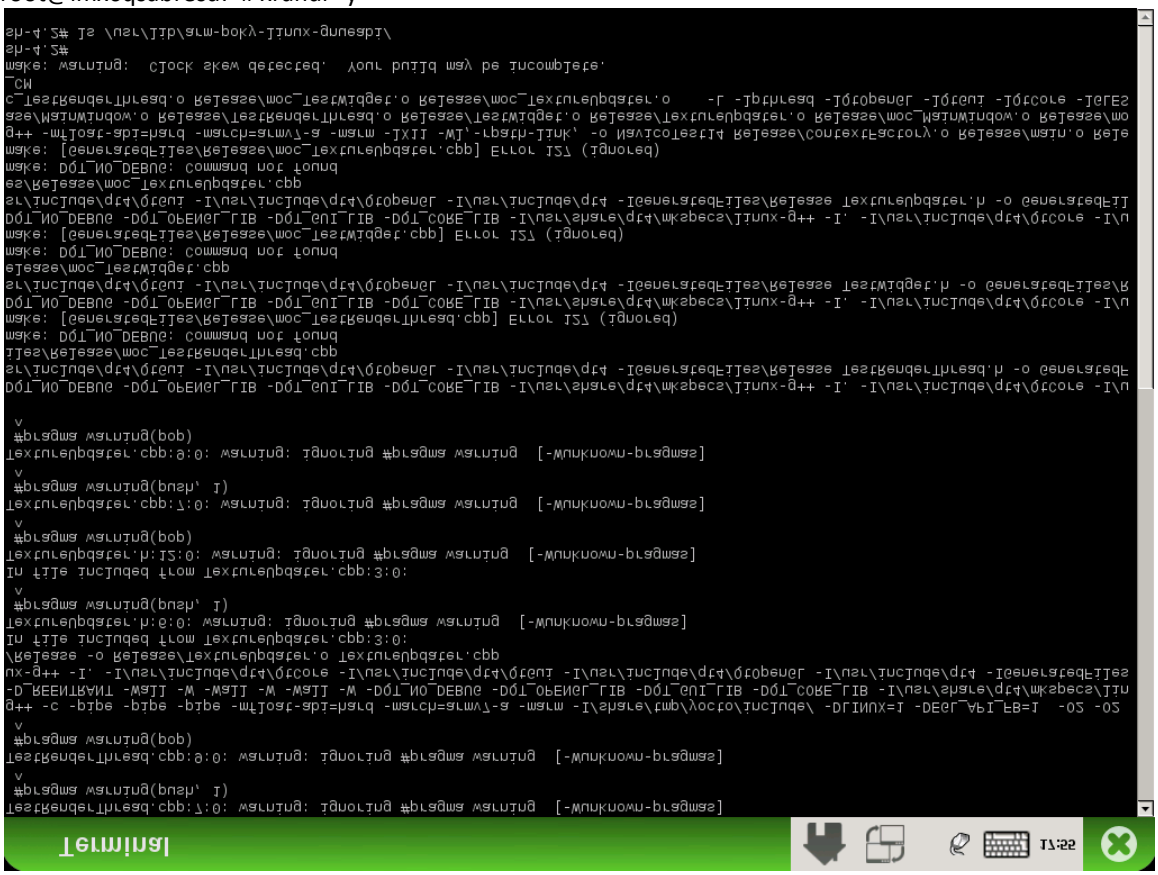


root@imx6qsabresd:~# xrandr -o inverted:

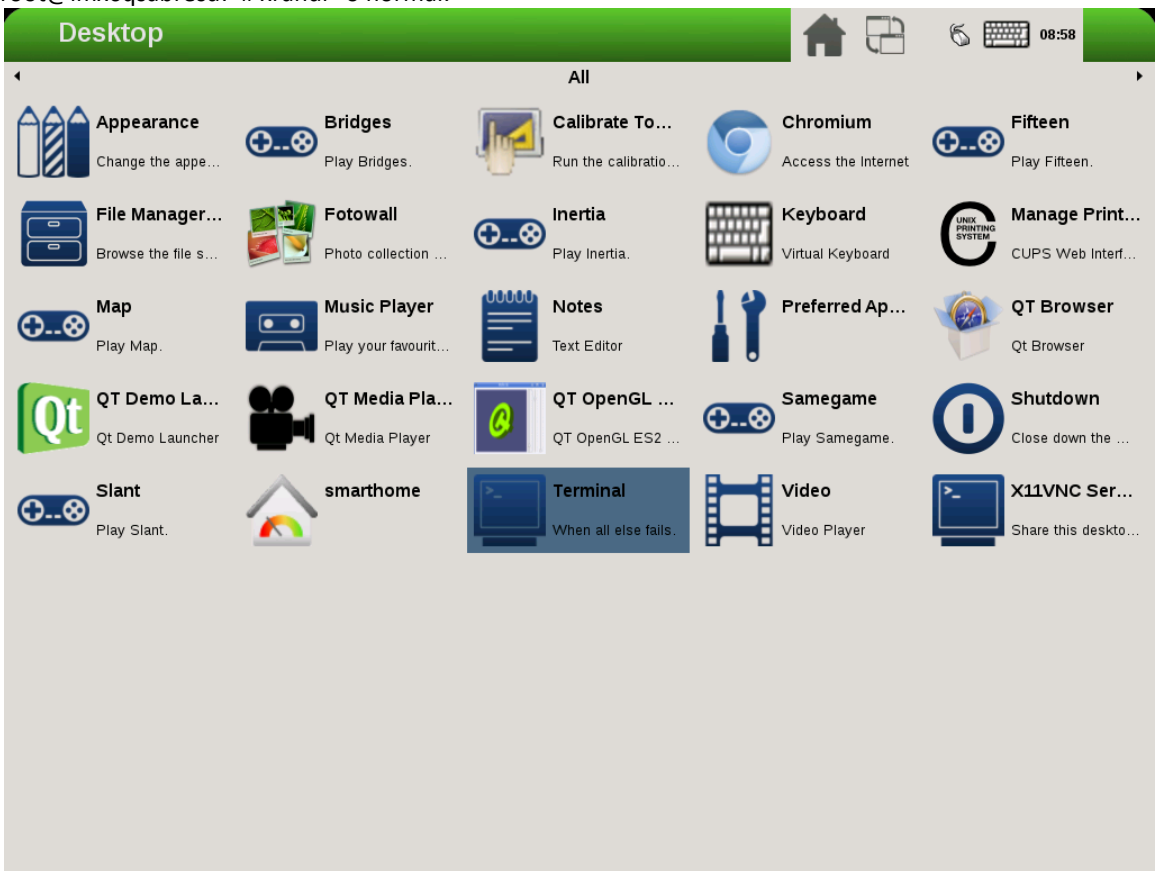
- Reflect the screen:
  root@imx6qsabresd:~# xrandr -x



root@imx6qsabresd:~# xrandr -y



- Restore to normal state:
  root@imx6qsabresd:~# xrandr -o normal:

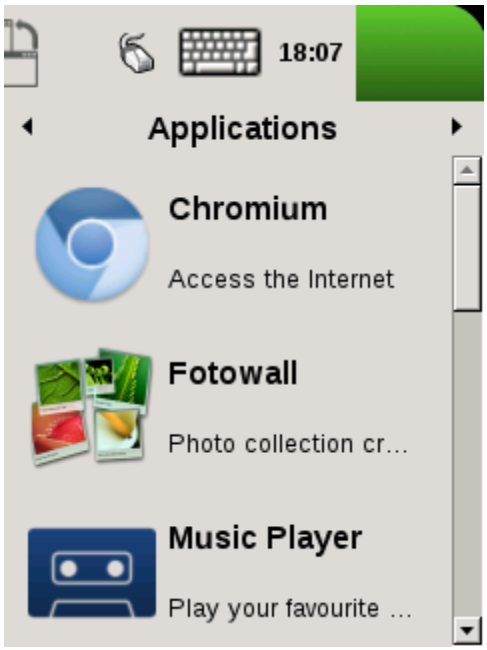


5.2.2 Rendering the desktop on overlay

`/dev/fb1` is the overlay device on the same screen as `/dev/fb0`; and `/dev/fb3` is the overlay of `/dev/fb2`. Use xorg.conf to specify fb1 or fb3:

```
Section "Device"
    Identifier  "i.MX Accelerated Framebuffer Device"
    Driver      "vivante"
    Option      "fbdev"     "/dev/fb1"
    Option      "vivante_fbdev" "/dev/fb1"
EndSection
```

After rebooting the system, the desktop will be rendered on the overlay:



If the size is too small (240x240), XRandR can be used to define a new mode.

1. Get the output name:

```
root@imx6qsabresd:~# xrandr
Screen 0: minimum 240 x 240, current 240 x 320, maximum 8192 x 8192
DISP4 FG connected 240x320+0+0 (normal left inverted right x axis y axis) 0mm x 0mm
   U:240x320p-60    60.0*
```

2. Define a new mode:

   root@imx6qsabresd:~# xrandr --newmode "640x480R" 23.50 640 688 720 800 480 483 487 494 +hsync -vsync

3. Add the newly created mode:

   root@imx6qsabresd:~# xrandr --addmode "DISP4 FG" 640x480R

4. Check the modes:

   root@imx6qsabresd:~# xrandr

```
Screen 0: minimum 240 x 240, current 240 x 320, maximum 8192 x 8192
DISP4 FG connected 240x320+0+0 (normal left inverted right x axis y axis) 0mm x 0mm
   U:240x320p-60    60.0*
   640x480R        59.5
```

5. Switch to a new mode:

   root@imx6qsabresd:~# xrandr -s 640x480

Note:

- The overlay size cannot exceed the display size. For example, if LVDS is 1024x768, the overlay size cannot be larger than this.
- Timings for overlay are meaningless, but wrong timings may damage your display, so be careful when creating a new display mode for your display.
- If fb3 is used, fb2 must be enabled. Otherwise, fb3 is invisible.

5.2.3 Process of selecting the HDMI default resolution

The process of selecting the HDMI default resolution is as follows:

1. Set the user preferred mode (must be within the initial size).
2. Set the display preferred mode (must be within the initial size).
3. Check the aspect (if not found, use 4:3. Find the biggest resolution within the initial size for the aspect ratio).
4. Check the first mode.

Initial size: initial FB virtual size or configured maximum size.
To specify the user preferred mode, add the option “PreferredMode” or “modes”.

5.2.4 Performance

The performance will decrease in the case of screen rotation or mirroring.

5.2.5 Memory consumption

The video driver supports a maximum of 1920x1080@32bpp. To support rotation, a shadow buffer is reserved, so the total memory consumption is 16 MB (1920x1080x4x2).

Chapter 6 Vivante Software Tool Kit

6.1 Vivante Tool Kit Overview

The Vivante Tool Kit (VTK) is a set of applications designed to be used by graphics application developers to rapidly develop and port graphics applications either stand alone, or as part of an IDE targeting a system-on-chip(SoC) platform containing an embedded GPU.

6.1.1 VTK Component Overview

The VTK includes a graphics and OpenCL emulator (vEmulator) to enable embedded graphics and compute application development on a PC platform, a driver and hardware performance profiling utility (vProfiler), and a visual analyzer (vAnalyzer) for graphing the performance metrics. Also provided are pre-processing utilities for stand-alone development of optimized shader programs (vShader) and for compiling shader code (vCompiler) into binary executeables targeting Vivante accelerated hardware platforms. An image transfer utility (vTexture) provides compression and decompression options.

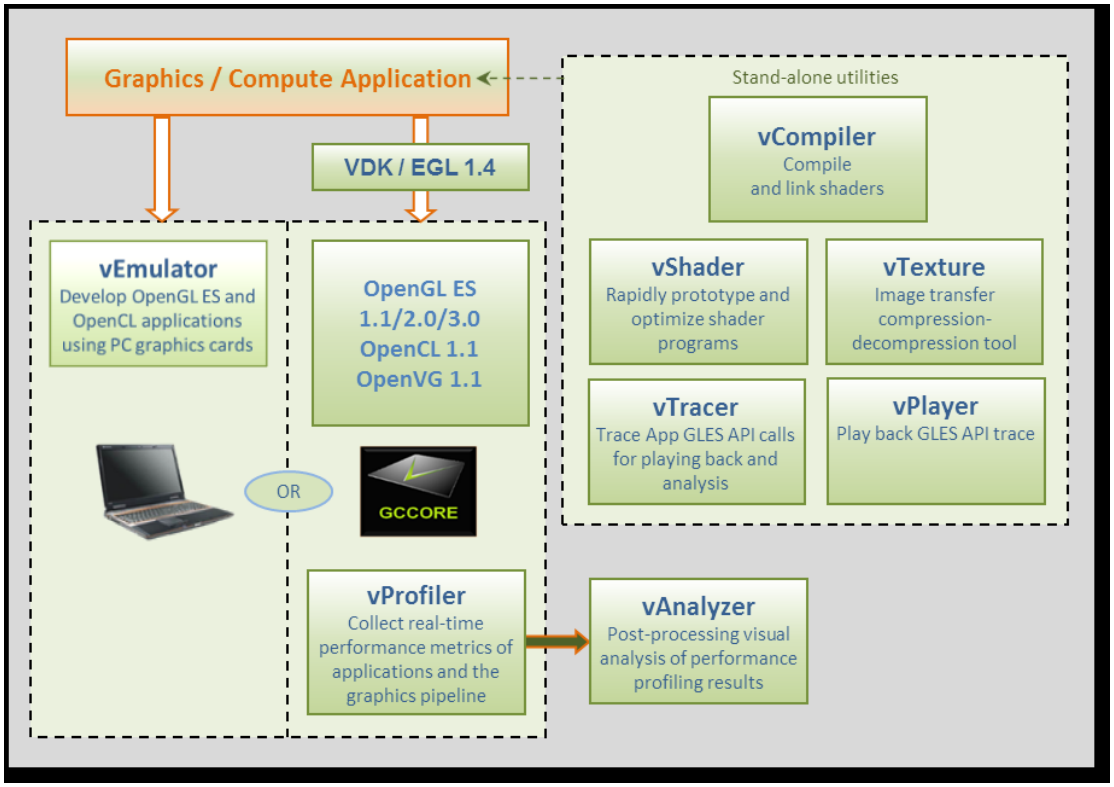


Figure 1. Vivante Tool Kit vTools Components

6.1.2 VTK Operating System Requirements

Most VTK vTools applications are designed to run on Microsoft Windows operating systems. The following systems are compatible with current releases of vTools:

- Microsoft Windows XP Professional, with Service Pack 2 or later
- Microsoft Windows Vista with Service Pack 2 or later
- Microsoft Windows 7 Professional

Some components, such as the vProfiler, are run on other platforms. Refer to the individual vTools component detail description.

6.1.3 VTK Installation

The vProfiler tool is not included in the VTK. This tool can be built by setting a build command option when making the Vivante Graphics Drivers.
The VTK package contains a **vtools** folder. Inside this folder are six .zip packages which can be individually extracted. As an example, if you have a system with WinRAR installed, right click and select Extract Here. A folder will be created with the same name as the .zip file.

- **vAnalyze.zip**
- **vCompiler.zip**
- **vEmulator.zip**
- **vShader.zip**
- **vTexture.zip**
- **vTracer.zip**

Each vTools extracted folder will contain a **SETUP.exe** and a ***vToolName.msi*** file. The tool can be installed independently by running the **SETUP.exe** located in that tool's folder. Typical licensing and folder placement options may appear as part of the installation prompts.

vAnalyzer and vShader have a Windows GUI. vEmulator is a library. vCompiler and vTexture are utilities run from the command line.
NOTES:
• The default installation location for the VTK is usually a folder named something like **C:\Program Files\Vivante*vToolName*,** where *vToolName* is the name of the tool being installed. Some systems may install to a Program Files (x86) folder.
• Windows navigation instructions such as Control Panel navigation will vary with the different Windows operating systems.
• Administrator rights may be required to install the tool.
• If you are installing an update version, use Windows Add/Remove programs to remove the installed version of the tool, before installing the update version.

6.2 vEmulator

Vivante's vEmulator duplicates the graphics and compute functionality of the Khronos APIs—namely, OpenGL ES 3.0, 2.0, 1.1 and OpenCL 1.1—in a desktop PC environment. This enables developers to write and test applications for Vivante embedded GPU cores prior to their availability, using the graphics cards on Windows XP or Windows Vista™ or Windows 7 PC platforms.

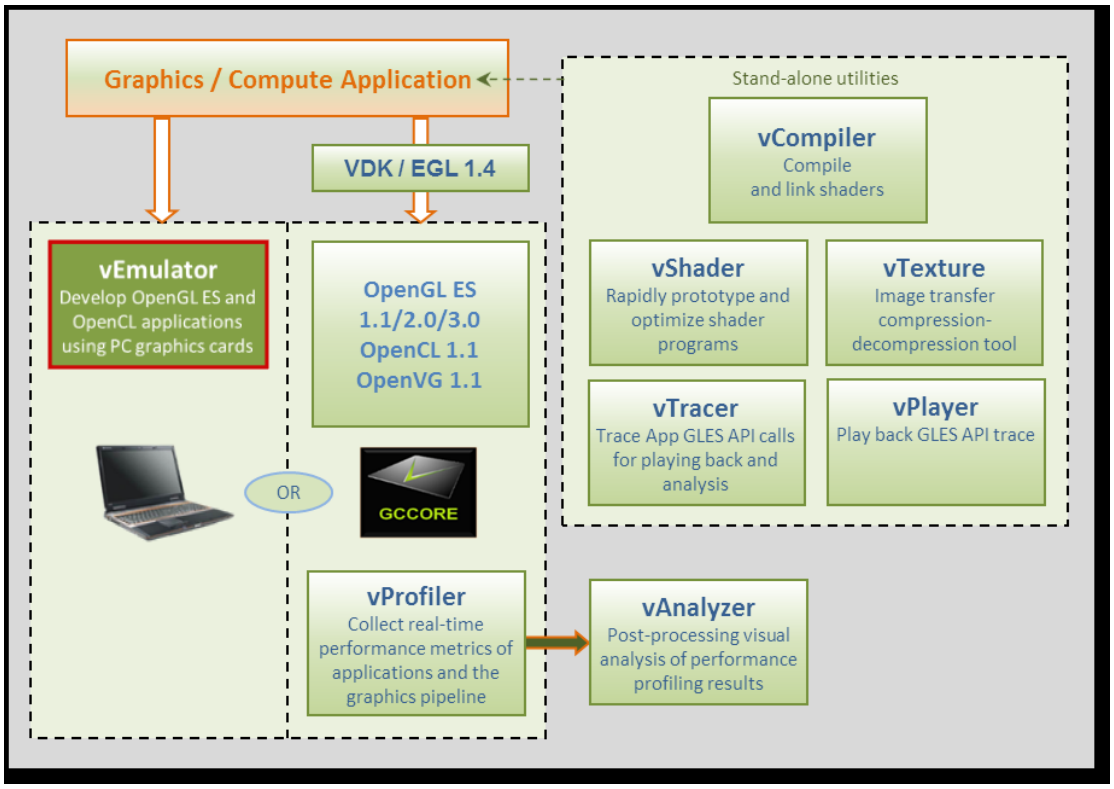


Figure 2. vEmulator embedded graphics emulator

vEmulator is not an application, but rather a set of libraries that convert Khronos mobile API function calls into OpenGL desktop or OpenCL function calls. These libraries can be accessed directly by the graphics / compute application.

6.2.1 Supported Operating Systems and Graphics Hardware

vEmulator libraries are available for Microsoft Windows XP, Windows Vista and Windows 7 operating systems:

- Microsoft Windows XP Professional, with Service Pack 2 or later
- Microsoft Windows Vista with Service Pack 2 or later
- Microsoft Windows 7 Professional

vEmulator has been tested on popular graphics cards, including:

- NVIDIA GeForce GTX 200 series with driver version 182.05or later
- NVIDIA GeForce 9000 and 8000 series with driver version 182.05or later
- NVIDIA GeForce 8400 GS with ForceWare driver version 176.44 or later
- ATI Radeon HD 3000 and 4000 series with driver version Catalyst 9.1 or later

vEmulator for OpenGL ES 3 has been tested on the nVidia GeForce GT430 card with driver version 310.90. Additional graphics cards will be added as testing is confirmed.

6.2.2 vEmulatorComponents

vEmulator libraries are packaged with the Vivante VTK installer. Once installed the libraries will reside in a folder vEmulator in the VTK installation path which can be specified by the user at time of installation. The default location of the Vivante VTK is:

`C:\Program Files\Vivante`

The vEmulator folder contains everything that is needed for emulation. The vEmulator directory structure and its files are described in the following table.

Table 3. vEmulator Directory Contents

vEmulator subdirectory	Filename	Description
bin	libEGL.dll	Dynamic library for invoking EGL at runtime
	libGLESv1_CM.dll	Dynamic library for OpenGL ES 1.1 emulation
	libGLESv2x.dll	Dynamic library for OpenGL ES 2.0 emulation
	libGLESv3.dll	Dynamic library for OpenGL ES 3.0 emulation
	libOpenCL.dll	Dynamic library for OpenCL 1.1 emulation
	libVEmulatorVDK.dll	Dynamic library for vEmulator VDK functions
inc	gc_vdk.h	Vivante VDK declarations
	gc_vdk_types.h	Vivante VDK type declarations
inc/EGL	egl.h	EGL declarations
	eglext.h	EGL extension declarations
	eglplatform.h	Platform specific EGL declarations
	eglrename.h	Rename for building static link driver
	eglunname.h	For mixed usage of ES11, ES20
	eglvivante.h	Vivante EGL declarations
inc/GLES	egl.h	EGL declarations
	gl.h	OpenGL 1.1 declarations
	glext.h	OpenGL1.1 extension declarations
	glplatform.h	Platform specific OpenGL 1.1 declarations
	glrename.h	Rename for building static link driver
	glunname.h	For mixed usage of ES11, ES20
inc/GLES2	gl2.h	OpenGL 2.0 declarations
	gl2ext.h	OpenGL 2.0 extension declarations
	gl2platform.h	Platform specific OpenGL 2.0 declarations
	gl2rename.h	Rename for building static link driver
	gl2unname.h	Unified name definitions
inc/GLES3	gl3.h	OpenGL 3.0 declarations
	gl3ext.h	OpenGL 3.0 extension declarations
	gl3platform.h	Platform specific OpenGL 3.0 declarations
inc/hal	gc_hal_eglplatform_type.h	Vivante HAL platform specific struct declarations
inc/KHR	khrplatform.h	Platform specific Khronos declarations
	khrvivante.h	Vivante specific Khronos declarations
lib	libEGL.lib	Static library for linking EGL functions
	libGLESv1_CM.lib	Static library for linking OpenGL ES 1.1 functions
	libGLESv2x.lib	Static library for linking OpenGL ES 2.0 functions

	libGLESv3x.lib	Static library for linking OpenGL ES 3.0 functions
	libVEmulatorVDK.lib	Static library for linking vEmulator VDK functions
	libOpenCL.lib	Static library for linking OpenCL functions
samples/es11, /es20	tutorials.sln	Microsoft Visual Studio project solution file for samples
samples/es11/tutorialN	-- Varies with N --	Sample OpenGL ES 1.1 applications
samples/es20/tutorialN	-- Varies with N --	Sample OpenGL ES 2.0 applications

6.2.3 vEmulator for OpenCL

If your edition of vEmulator includes support for OpenCL, additional files may be present. For OpenCL emulation using vEmulator on your PC, please refer to the OpenCL emulator readme file (OCL_Readme.txt) in the vEmulator folder for additional installation instruction.
Note: An additional environment variable **CL_ON_GC2100** needs to be set for simulation for GC2100. The value can be any characters, as long as it is not null. This variable does not need to be set for other OCL cores.

Table 4. vEmulator Files for OpenCL 1.1

vEmulator subdirectory	Filename	Description
	OCL_Readme.txt	Readme file for OpenCL 1.1
bin	libOpenCL.dll	Dynamic library for invoking OCL at runtime
	cl.h	OpenCL 1.1 core API header file
	cl.hpp	OpenCL 1.1 C++ binding header file
	cl_d3d10.h	OpenCL 1.1KhronosOCL/Direct3D extensions header file
	cl_ext.h	OpenCL 1.1 extensions header file
	cl_gl.h	OpenCL 1.1Khronos OCL/OpenGL extensions header file
	cl_gl_ext.h	OpenCL 1.1Vivante OCL/OpenGL extensions header file
	cl_platform.h	Platform specific OCL declarations
	opencl.h	Vivante HAL version
lib	libOpenCL.lib	Dynamic library for linking OpenCL functions
samples/cl11	cl_sample.cpp	Sample OpenCL 1.1 source code
samples/cl11	cl_sample.sln	Sample OpenCL 1.1 Visual Studio solution file
samples/cl11	cl_sample.vcproj	Sample OpenCL 1.1 Visual Studio solution project file
samples/cl11	square.cl	Sample OpenCL 1.1 kernel file

6.2.4 Supported Extensions

Refer to the document **EGL & OES Extensions Support** for a list of supported and custom extensions available for EGL and OpenGL ES.
Software extensions have not been added to vEmulator for OpenGL ES 2.0. vEmulator relies on the extensions available with the installed version of native OpenGL.

6.2.5 vEmulator Environment Variable Setup

There are two steps to running an OpenGL ES or OpenCL application with vEmulator:
Step 1. Link to the vEmulator **`*.lib`** static libraries at build time when creating an application executable image.
Step 2. Provide a path to the vEmulator **`*.dll`** dynamic libraries during run-time.
These steps require a one-time setup in which the location of the vEmulator libraries is added to the Microsoft Windows system environment variable named "Path." In our example, the following string would be added to the system "Path" variable: `C:\Program Files\Vivante\vEmulator\bin.`
To add vEmulator DLL files to the Windows XP system path:

a. Click **Start** then click **Control Panel** then double-click **System**

- Vista: then click **Advanced system settings** from the Tasks list in the upper-left corner of the window.
- Windows 7: in the System and Security window, click System, then on the left menu column click **Advanced system settings**.

b. Select the **Advanced** tab, then click on the **Environment Variables…** button.

- An Environment Variables dialogue box will pop up, with two panes for variables.

c. Select **Path**, and then click on the **Edit…** button.

d. In the **Variable value:** field type the following environment variables in the order you want them found. For instance:

`C:\Program Files\vivante\vEmulator\bin;<current path>`

Note: The system parses a path string in left-to-right order when looking for a file. Whatever it finds first is what will be used.

e. If the Vivante Core is GC2100, an additional variable **`CL_ON_GC2100`** should be set to any non-null value.

f. Click **OK**.

- Click **OK** to close the Environment Variables dialogue window.
- Click **OK** to close the System Properties dialogue window.
- Close the Control Panel > System window.

6.2.6 Sample Code Overview

In the discussions that follow about the various sample programs included with the vEmulator distribution, we assume that vEmulator has been installed in the default location within the vivante/VTK folder:

`C:\Program Files\Vivante\vEmulator`

Relative to this path:

- run-time dlls are located at **`…\bin`**
- include-files are found at **`…\inc`**
- library files are located at **`…\lib\<API>`**
- examples are located at **`…\samples\<API>\tutorial*`**

where API is one of: **`es11`** or **`es20`**

The code examples are distributed with working *.exe executable images so that the VTK user can see how the results should look.
They are presented in a tutorial fashion, progressing from simpler programs to more complex as the tutorial number increases.

6.2.7 Building and Running the Code Examples

The steps to build and run are identical for all code examples, regardless of the API (es11 or es20). There are two general guidelines to keep in mind.

1) A Visual Studio project has environment variables that allow the specification of additional paths to "include" and "library" files when a source module from that project is being built. The Visual Studio projects that are part of the vEmulator distribution package are configured out-of-the-box for building all of the sample code executables, relative to the location where vEmulator is installed. Specifically the additional paths are set as "$(SolutionDir)..\..\inc" and "$(SolutionDir)..\..\lib".

If \samples is moved, or if the VTK user begins with the provided projects as templates for developing applications in a directory that is not directly under the \vEmulator installation, then the project path variables must be adjusted accordingly. For example:
To access these path variables for tutorial1, first launch the tutorials.sln

- Right-click on **tutorial1**, then select **Properties** (at the bottom of the pop-up menu)

- Under "Configuration Properties" > "C/C++" > "General", edit the **Additional Include Directories** entry
  - E.g., change **`..\..\..\inc`** to **`C:\Program Files\Vivante\vEmulator\inc`**

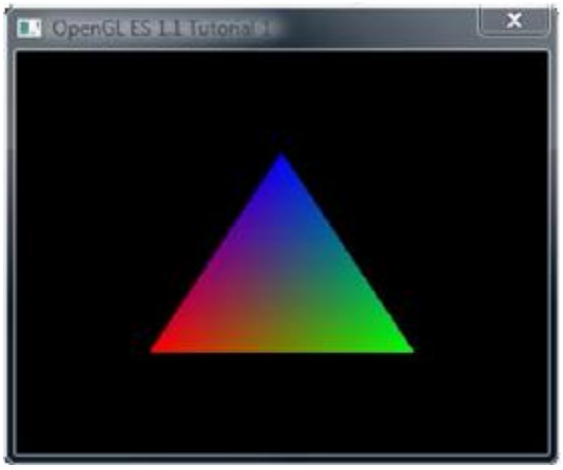
- Under "Configuration Properties" >"Linker" > "General", edit the **`Additional Library Directories`** entry
  - E.g., change **`..\..\..\lib`** to **`C:\Program Files\Vivante\vEmulator\lib`**

2) Make sure that the system environment variable **PATH** contains a path to the vEmulator DLL files. (See above section on vEmulatorEnvironment Variable Setup, above.) Remember that the path is order-dependent; whatever the system finds first will be used. If there is more than one DLL with the same name, double-check that the path to the desired one is listed first in the **PATH** string.

6.2.8 OpenGL ES 1.1 Examples

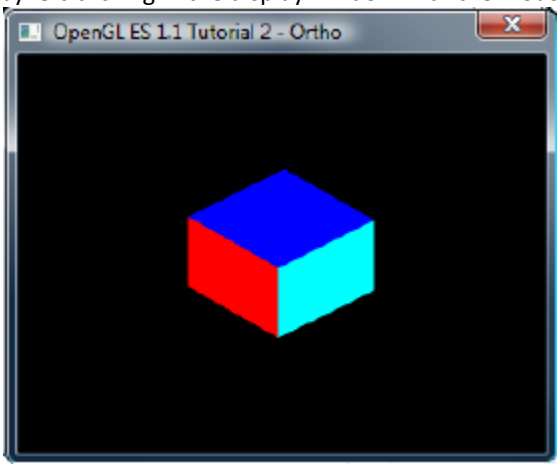
6.2.8.1 tutorial1: Rotating Three Color Triangle

Renders a cube centered at the origin with a different color on each face. Flat shading is used. The cube rotates about the vertical axis. The default projection is ORTHO, which can be toggled between ORTHO and PERSPECTIVE by left-clicking in the display window with the mouse or pressing Enter.



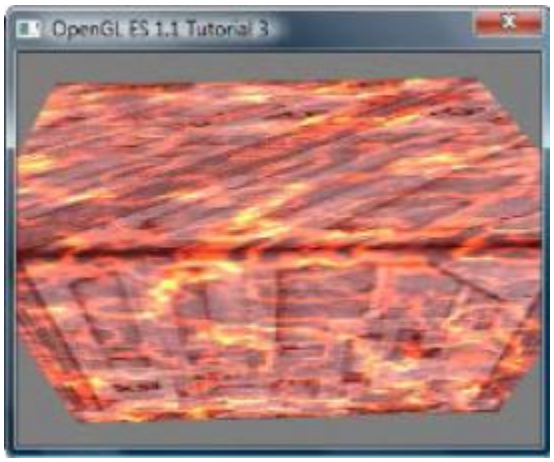
6.2.8.2 tutorial2: Rotating Six-color Cube

Renders a cube centered at the origin with a different color on each face. Flat shading is used. The cube rotates about the vertical axis. The default projection is ORTHO, which can be toggled between ORTHO and PERSPECTIVE by left-clicking in the display window with the mouse or pressing Enter.



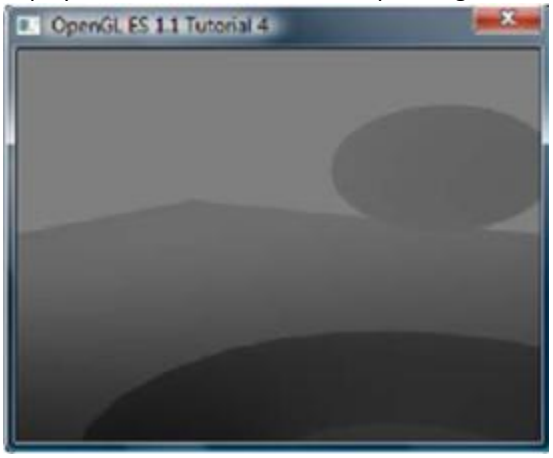
6.2.8.3 tutorial3: Rotating Multi-Textured Cube

This example takes the cube of the previous example with PERSPECTIVE projection, loads two textures from file and combines them using GL_ADD blending mode, and applies the resulting texture to the cube faces.



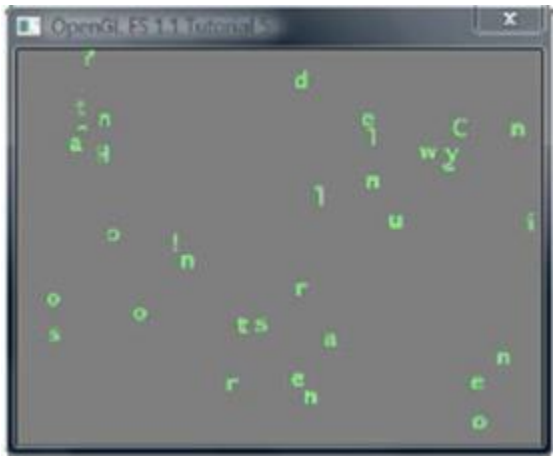
6.2.8.4 tutorial4: Lighting and Fog

What appears to be a torus, a cone, and an oblate spheroid orbiting about the center of a plane is actually a single mesh being lit by a single rotating, diffuse light source. Green fog is added to the scene by left-clicking on the display window with the mouse or pressing Enter.



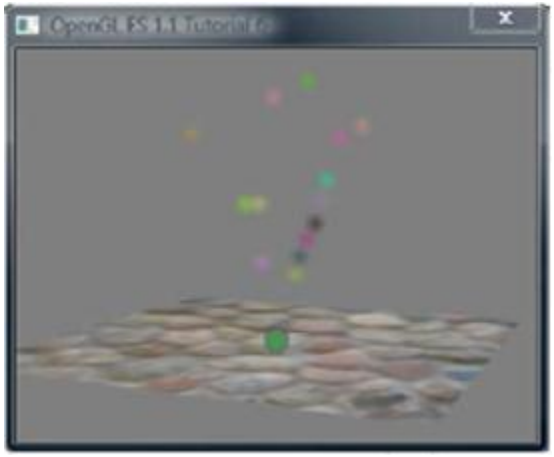
6.2.8.5 tutorial5: Blending and Bit-mapped Fonts

This example makes use of alpha blending to animate sprites across the display, and it also instructs how to create a bit-mapped font from a texture. Jumbled letters iteratively print and move across the display as they unscramble into a text message.



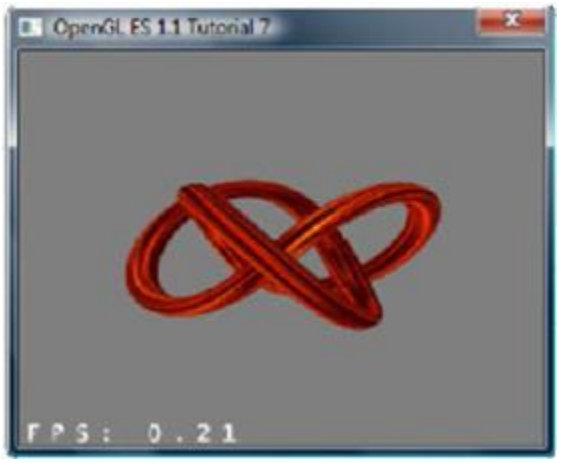
6.2.8.6 tutorial6: Particles Using Point Sprites

This example reuses the bit-mapped font technique from the previous tutorial, but it adds a particle generator to simulate and animate particles being emitted from the textured plane. All computation is performed in fixed-point arithmetic.



6.2.8.7 tutorial7: Vertex Buffer Objects

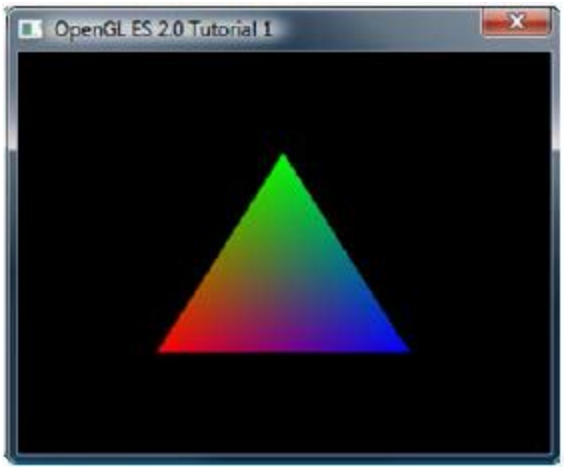
Using Vertex Buffer Objects (VBO) can substantially increase performance by reducing the bandwidth required to transmit geometry data. Information such vertex, normal vector, color, and so on is sent once to locate device video memory and then bound and used as needed, rather than being read from system memory every time. This example illustrates how to create and use vertex buffer objects.



6.2.9 OpenGL ES 2.0 Examples

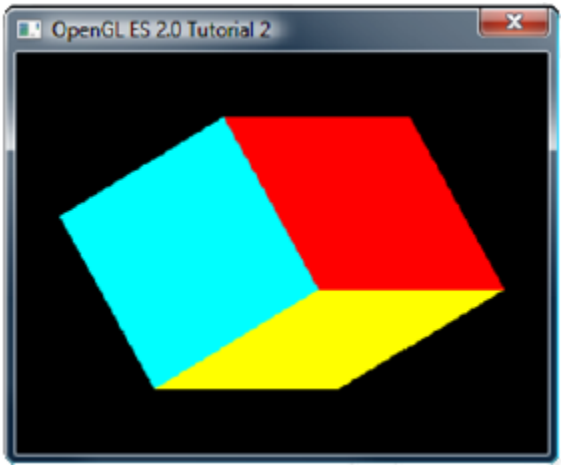
6.2.9.1 tutorial1: Rotating Three-color Triangle

A single triangle is rendered with a different color at each vertex, Gouraud shading for blending, rotational animation in the final display. This is the same example as es11/tutorial1, only implemented in OpenGL ES 2.0.



6.2.9.2 tutorial2: Rotating Six-color Cube

Renders a cube centered at the origin with a different color on each face, and rotates it about the vertical axis. Similar to the es11/tutorial2 example, the default projection is ORTHO. But there is no toggle for PERSPECTIVE.



6.2.9.3 tutorial3: Rotating Reflecting Ball

A ball made of a mirroring material and centered at the origin spins about its Y-axis and reflects the scene surrounding it.
Note: if the program cannot be executed and print “GL error” in the console, please remove the line “return” before the line of “DeleteCubeTexture(cubeTexData);”

6.2.9.4 tutorial4: Rotating Refracting Ball

This example is the same as the previous one, except that the ball is made of clear glass which refracts the surrounding environment.
Note: if the program cannot be executed and print “GL error” in the console, please remove the line “return” before the line of “DeleteCubeTexture(cubeTexData);”

6.3 vShader

vShader is a complete off-line environment for editing, previewing, analyzing, and optimizing shader programs.

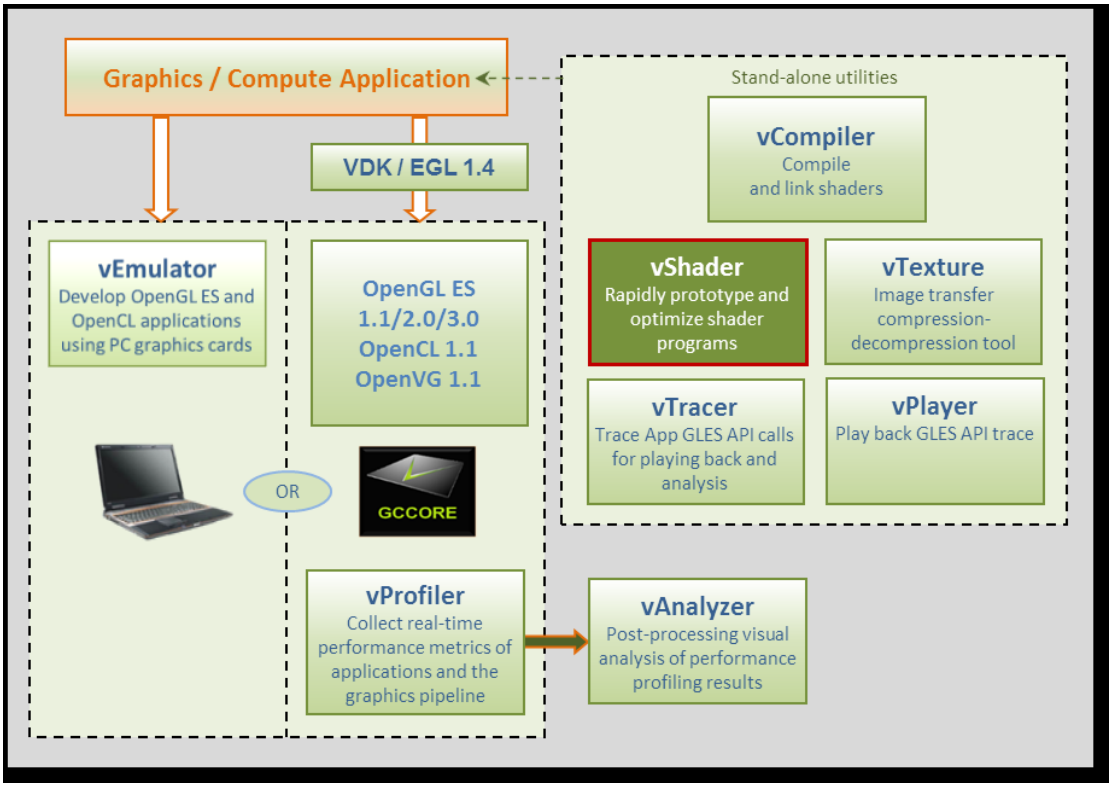


Figure 3. vShader shader editor

vShader allows users to:

- Map any texture onto shaders
- Import user-defined meshes
- Bind mesh attributes to shaders
- Set uniforms in shaders

- View shader compiler output for optimization hints
- Predict hardware performance

6.3.1 vShader Components

By default, the vShader executable installs in the following location within the Vivante Toolkit directories:
`C:\Program Files\Vivante\vShade.`
The vShader package 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.
By combining vertex shaders and fragment shaders into a single shader program, an application can produce a shader effect. A project can make use of many shader effects, which can share vertex and fragment shaders, mixing and matching to achieve the desired results.
The scope of this guide is to cover the vShader user interface. The tutorials provided with the vShader package are there to help the reader learn about shaders, if needed.

6.3.2 Getting Started with vShader

Once the vShader utility is launched by clicking on a shortcut or directly on the executable vShader.exe projects can be created, developed and saved. Project files have an extension **`.vsp`**.

6.3.2.1 Creating a new project

To create a new project, locate the main menu bar: Select **File** then **New Project…**
Depending on the current project status, one of three things will happen:
1. If this is the first time vShader is launched, then there is no project already open and selecting "File > New Project…" will appear to have no effect.
2. If there have been no changes to the current project since the last save, then the current project will close and a new, empty project will be opened.
3. If the current project has been modified, then a dialog box will pop up to ask if you want to save the changes. Choosing **Yes** will commit the changes to the current project, which will then be closed, and a new, empty project will be opened.

6.3.2.2 Opening an existing project

To open an existing project, locate the main menu bar:

1. Select **File** then **Open Project…**
2. Double-click on the desired project from the list that pops up, or single-click on the project name and click OK.
   The project will load into vShader and appear in the state it was last saved.

6.3.2.3 Saving a project

To save a project, locate the main menu bar:

1. Select **File** then **Save Project…**
2. In the resulting dialog box indicate where to save the project, then click **OK.**

6.3.3 vShaderNavigation

The vShader application runs on the Windows XP, Windows Vista and Windows 7 platforms and is driven from a graphical user interface as shown in the figure below.
Main components of the GUI include:

- on upper portion of window: a Menu Bar, Menu Icons,
- on left: Preview pane, Project Explorer pane
- on right: Shader Editor pane
- on lower portion of window: InfoLog pane.

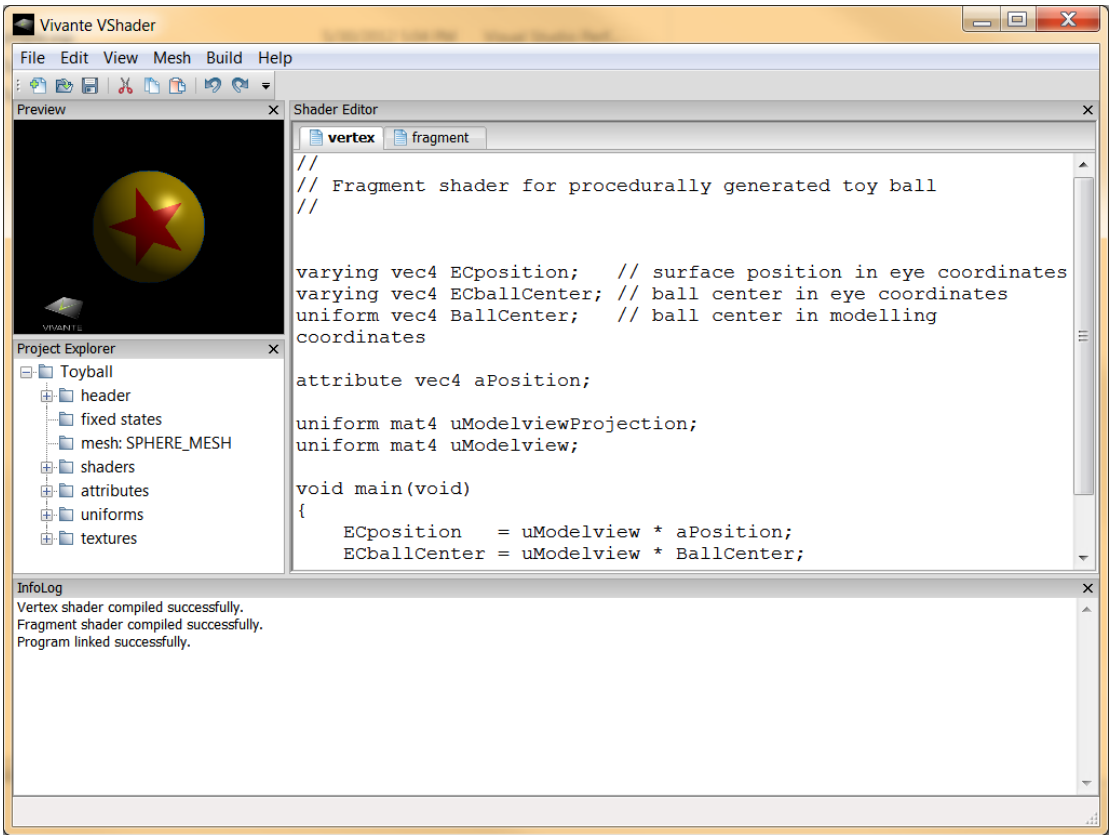


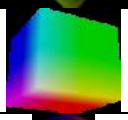
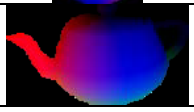
Figure 4. vShader GUI main window

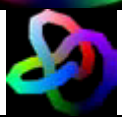
6.3.3.1 vShader Menu Bar

The main window opens when a user launches vShader. The main menu bar contains drop-down menus for File, Edit, View, Mesh, Build, and Help.

Table 5. vShader Menu Commands

Menu Name	Menu Command	Description
File		
	New Project…	Create a new project file; if a project is currently open, then the user is prompted to choose whether to save it first.
	Open Project…	Browse for and load a .vsp VShader project.
	Save Project…	Save the current project; if this is the first time saving this project, then the user is prompted to choose where to save it.
	Load Vertex…	Browse for and load a vertex shader from an existing text file.

	Load Fragment…	Browse for and load a fragment shader from an existing text file.	
	Save VertexShader As…	Prompts for filename and location to save the active vertex shader.	
	Save FragmentShader As…	Prompts for filename and location to save the active fragment shader.	
	Exit	Close all open files and exit VShader.	
Edit			
	Undo [Ctrl-z]	Revert to a previous edit state (Note: Undo is only 1-level deep)	
	Redo [Ctrl-y]	Re-apply the last “undone” edit command (Note: Redo is only 1-level deep)	
	Cut [Ctrl-x]	Delete the selected item(s) and save a copy in the paste buffer	
	Copy [Ctrl-c]	Save a copy of the selected item(s) item in the paste buffer	
	Paste [Ctrl-v]	Insert the contents of the paste buffer	
	Delete [Del or Bkspc]	Remove the selected item(s)	
	Select All [Ctrl-a]	Highlight all items in the current view	
View			
	Reset Preview	Reset Preview window.	
	Snapshot	Save current preview image to bitmap bmp file. A dialog will display to let user choose where to save the bmp.	
	Perspective	Use perspective projection in the Shader Preview pane	
	Ortho	Use orthographic projection in the Shader Preview pane	
	Tool Bar	Show or hide toolbar icons	
	Preview Window	Show or hide Preview window	
	Project Explorer	Show or hide Project Explorer window	
	Shader Editor	Show or hide Shader Editor window	
	InfoLog	Show or hide InfoLog window	
Mesh			
	Conic	Looks like a spiral horn.	
	Cube	A 3D cube.	
	Klein	The Klein bottle.	
	Plane	A 2D square.	
	Sphere	A ball.	
	Teapot	The Utah teapot.	

	Torus	Looks like a donut.	
	Trefoil	A trefoil knot.	
	Custom Mesh…	Browse for and open a 3DS mesh file.	
Build			
	Compile	Compile the active shader.	
	Link	Link the vertex and fragment shaders into a shader program, and apply it to the mesh showing in the Shader Preview window pane.	
	Clear InfoLog	Remove all text currently showing in the InfoLog window pane.	
Help			
	About	Information about the version of VShader being used.	

6.3.3.2 vShader Window Panes

There are four window panes in the vShader GUI: Preview, Project Explorer, Shader Editor, and InfoLog. Each pane can be resized by left-mouse-dragging the pane edge. A pane can be hidden by clicking the **X** in the upper-right corner of the pane, or by un-checking the box next to its name in the View pull-down of the main menu. Restoring a hidden window pane is done by checking the appropriate box in the View pull-down menu.
Individual panes in the vShader application can be resized, relocated or converted to detached windows, as in the example to the right.

Note: Changes made to pane arrangement are not restored on application or project relaunch.

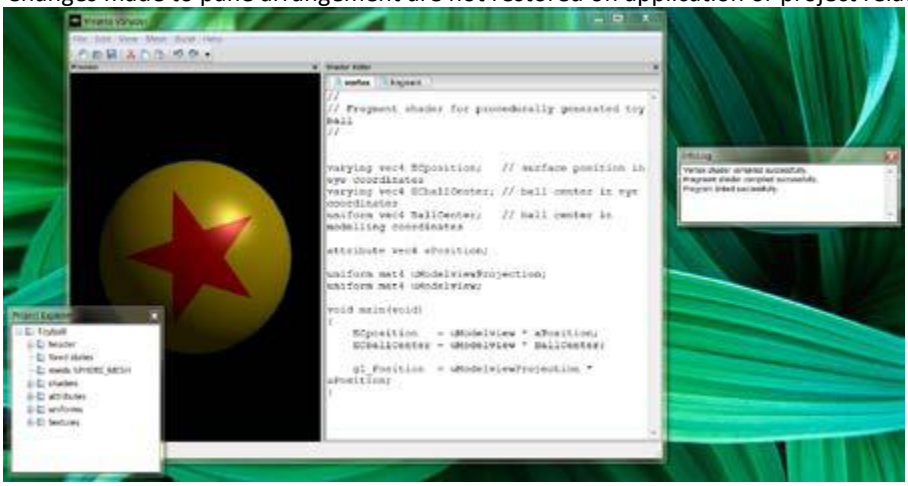


Figure 5. vShader Moveable Panes

6.3.3.2.1 Preview

The shader Preview pane shows the current effect of the shaders on the chosen mesh geometry. A different mesh may be chosen either via the Mesh pull-down menu in the menu bar near the top of the vShader main window or by right-mouse clicking in the Preview pane.

When using the right-click method, the user also can choose between perspective and orthographic views of the mesh, can reset the view orientation to the default, or can save the current view in the Preview window as a bitmap file by selecting **Snapshot**.
The object in the Preview window can be rotated, translated, and scaled. Rotation is controlled by left-mouse-drag; translation is done by holding the Ctrl key plus left-mouse-drag; scaling the image is seen by holding the Alt key while applying left-mouse-drag.
When shader variables are changed, the shader preview updates automatically. When shader programs are changed they must be recompiled and relinked by the user, through the Build menu. The Preview display will automatically update to reflect the new Build.

6.3.3.3 Project Explorer

The Project Explorer displays all of the project resources in a familiar tree structure. The root of the tree is the project name, and the branches and leaves classify the resources. Folders can be expanded by clicking on the plus sign next to them, and they can be collapsed by choosing the minus sign. By right-mouse clicking on any resource name, the user can view and usually edit that resource.

6.3.3.3.1 Shader Editor

The Shader Editor is a work area for entering and modifying shader programs. There are two tabs: one for vertex shader, and one for fragment shader. Changes made to a shader must be compiled and linked in order for their effect to appear in the Shader Preview.
Compiling can be done by selecting **Build** then **Compile** from the main menu bar. Likewise, linking and applying the shaders is performed by choosing **Build** then **Link**.

6.3.3.3.2 Info Log

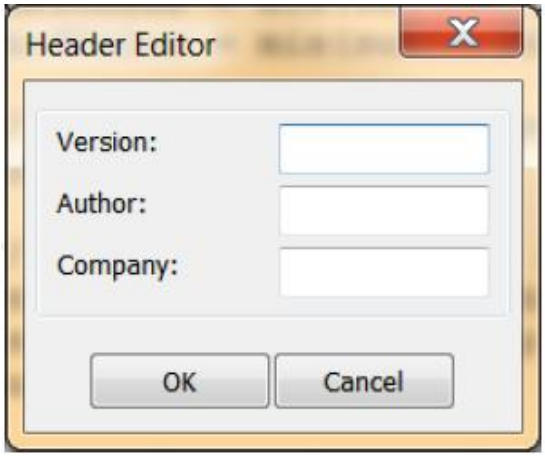
The Info Log window pane receives diagnostic messages from the compiler and linker, so that the user can see if the current shaders have built without errors. This pane can be cleared of text by selecting the **Build** then **Clear InfoLog** entry in the main menu.

6.3.4 vShader Project Resources

Project resources are accessible from the Project Explorer pane. Click on the item and an Editor pop-up dialog will appear where the user can enter alternate values. Resources include: header, fixed states, mesh, shaders, attributes, uniforms, and textures.

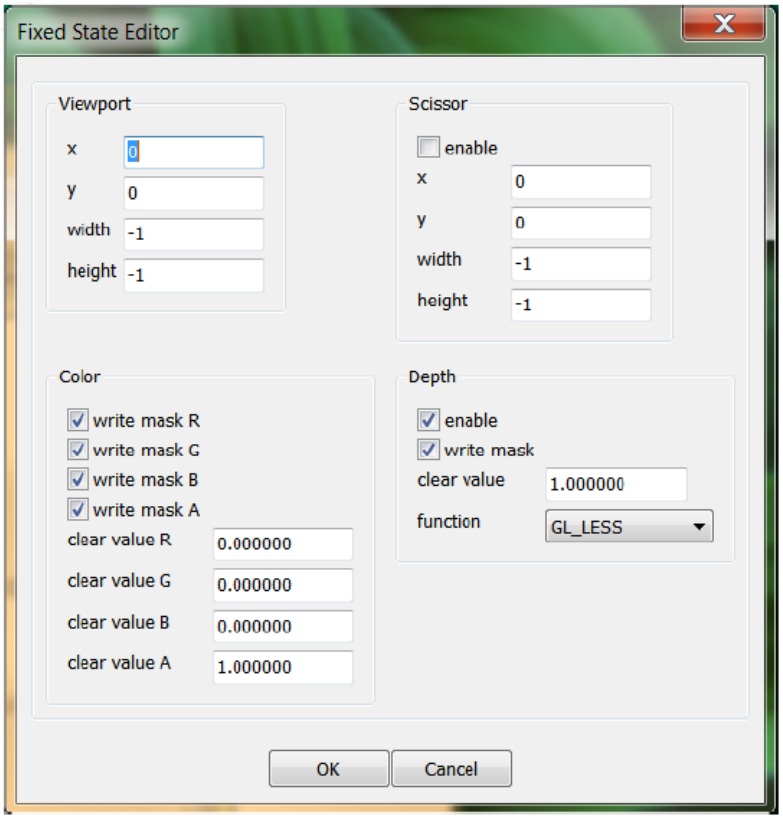
6.3.4.1 Header

Some project identifying information, namely version, author, and company. Expand the folder to see the settings, or right-click (or double-click) the folder to edit them.



6.3.4.2 Fixed States

The Fixed State Editor is a list of OpenGL ES 2.0 fixed states settings, such as depth test enable/disable, etc. It allows the user to set all fixed states manually. Right-click or double click to display an edit dialog.



6.3.4.3 Mesh

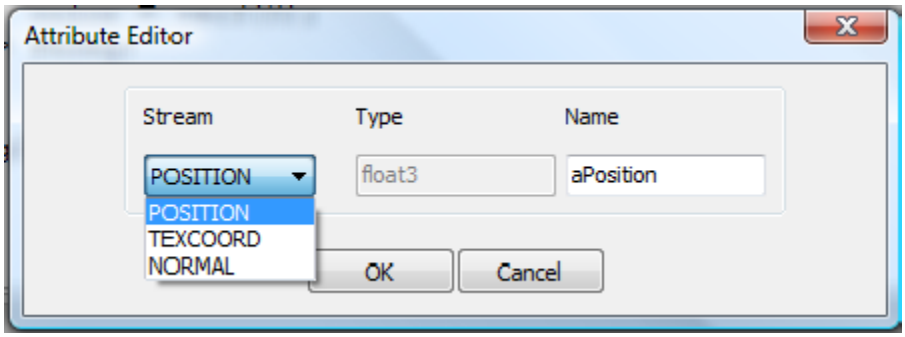
This resource shows the name of the mesh which is currently being displayed in the Preview pane. It does not have a pop-up window. Right-click on the mesh name to select a different mesh can be selected from the resulting pull-down menu.

6.3.4.4 Shaders

Left-click on the plus sign next to the “shaders” folder to reveal the two sub nodes in this section, which are vertex and fragment. Double-click (or right-click and then choose **Active**) on either shader to bring it forward in the Shader Editor for editing.

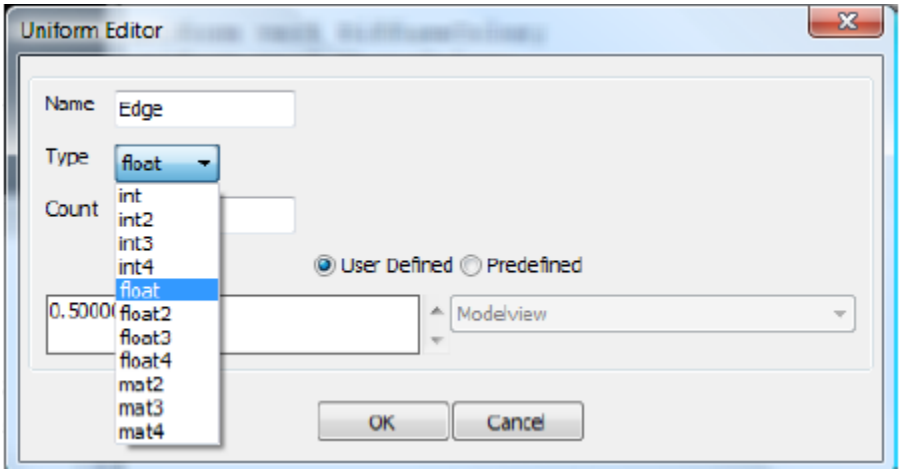
6.3.4.5 Attributes

The Attribute Editor dialog displays all attributes bound to the current project. It allows the user to add new attributes, and edit or remove existing attributes. Right click on **Attributes** to add a new one. Click on the plus sign to expand the attributes list, and then double-click to edit a particular attribute. Also, by right-clicking on an attribute, you can edit or remove that attribute or add a new one. Up to 12 attributes are allowed.



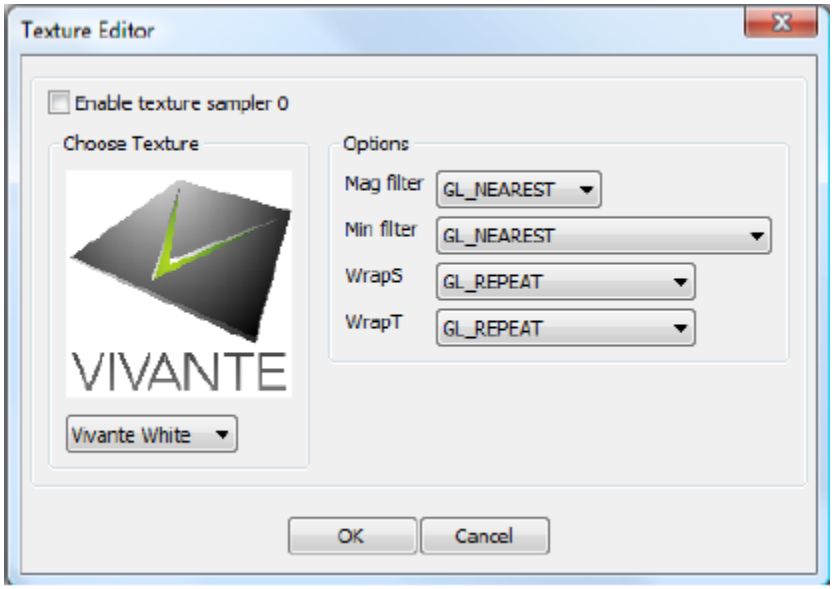
6.3.4.6 Uniforms

This displays all uniforms bound to the current project. Right click on **Uniforms** to add a new one, or expand the list and double-click on a given uniform to bring up the Uniform Editor dialog. When a uniform is right-clicked, the user can add new uniforms, or edit or remove existing uniforms. Up to 160 uniforms are allowed.



6.3.4.7 Textures

The Texture Editor dialog allows the user to select a texture for each of up to 8 texture units. The effect of applying each texture is seen immediately in the Shader Preview pane.
The texture selection option list is created from the texture files located in the “textures” subfolder of the project. The list can be expanded by adding textures to the textures folder, formatted as bitmap files.



6.4 vCompiler

vCompiler is an off-line compiler and linker for translating vertex and fragment shaders written in OpenGL ES Shading Language (ESSL) into binary executables targeting Vivante accelerated hardware platforms. vCompiler is driven by a simple command-line interface.

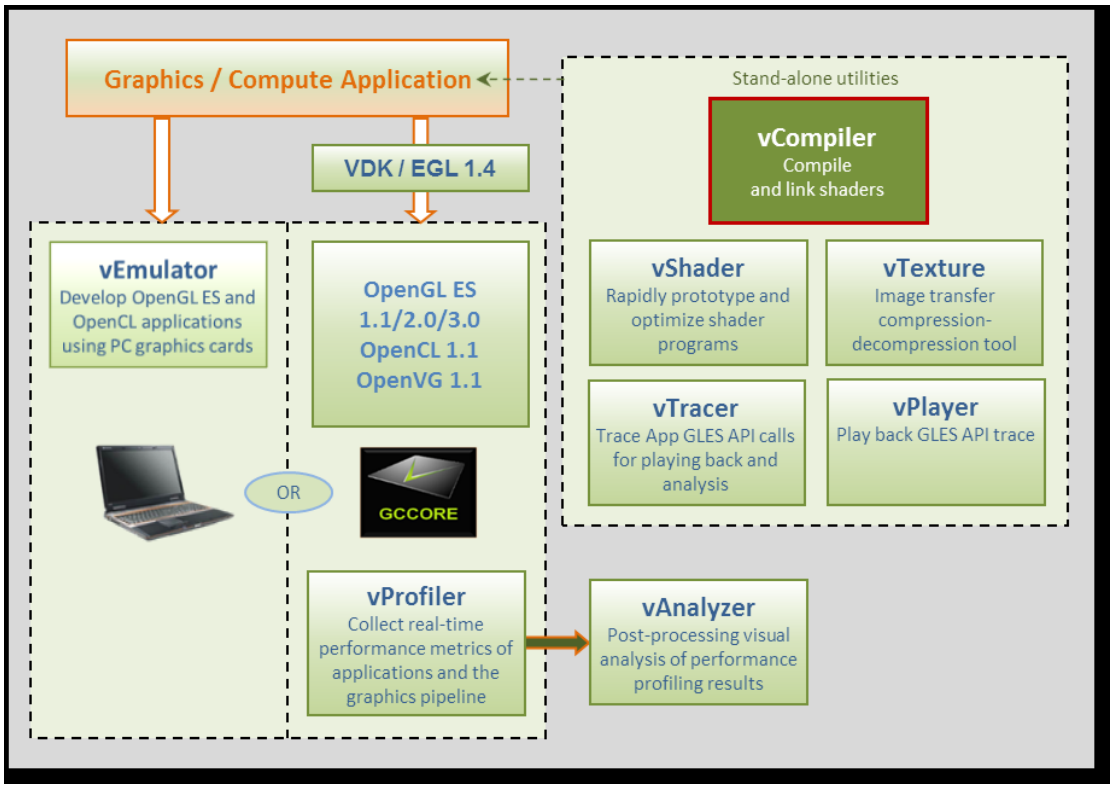


Figure 6. vCompiler compiler/linker

6.4.1 vCompiler Command Line Syntax

6.4.1.1 Syntax:

Optional inputs are indicated by italic font.

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

6.4.1.2 Input parameters (required):

`shaderInputFileName` 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

6.4.1.3 Input parameters (optional):

`shaderInputFileName_2` 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 shaderInputFileName. 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 pixel shader. Note: pre-compiled and compiled shaders may be mixed, as long as one is a vertex shader and the other a fragment shader.
-c	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: a)When only one shader is specified, that shader will be compiled into a .[v/p]gcSL file. b) 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.
-h	Shows a help message on all the command options.
-l	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., **myvert.vert => myvert.log** **inputfrag => inputfrag.log**
-o <outputFileName>	Specify the output file name. If the path is other than the current directory, it must also be specified. Any extension can be specified. If the extension is not specified, the following are **outputFileName** supported default types: 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/pixelshader into a single output file
-O*n*	Optimization level. Default is **-O1:** **-O0** — Disable optimizations **-O1- -O9** — Indicates on which level optimization should be done. The default is level 1. Note: Optimization is actually implemented in the compiler, not vCompiler.
-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.

6.4.1.4 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.

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

6.5 vTexture

vTexture is a command line tool which provides compression and decompression functions to help developers transfer image formats.

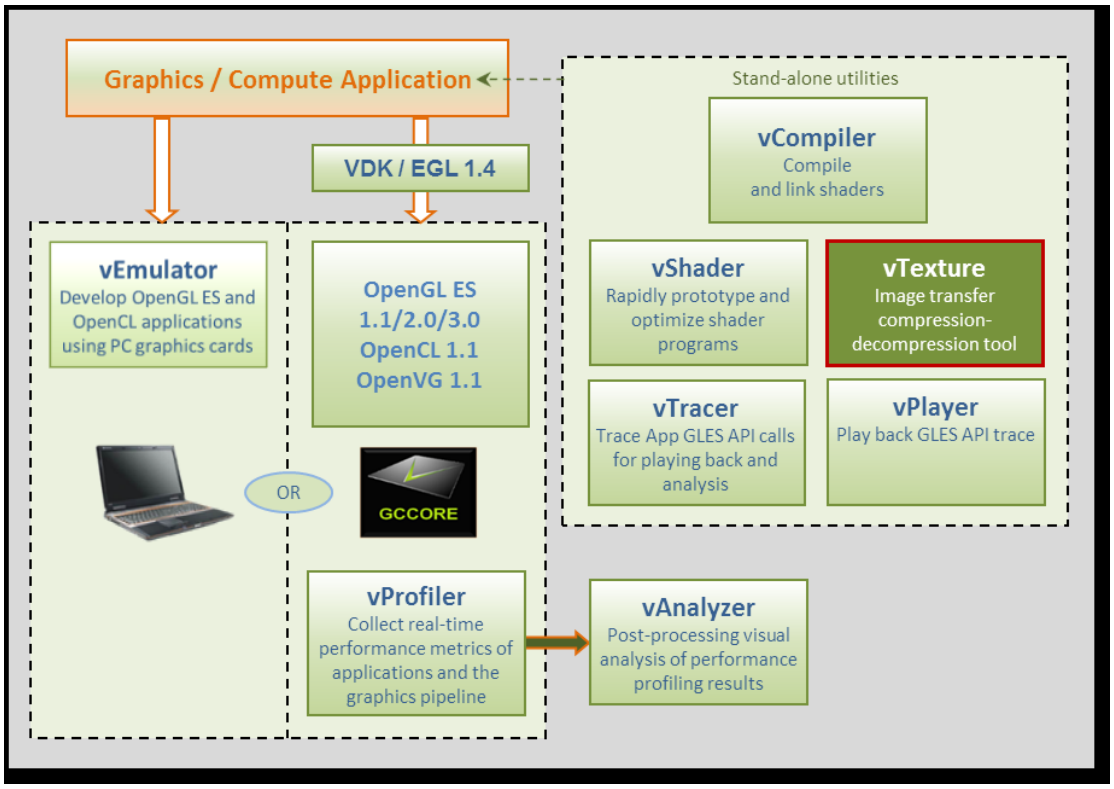


Figure 7. vTexture Image Transfer Tool

6.5.1 Formats

6.5.1.1 Supported Formats

The vTexture tool supports:

- compression of uncompressed TGA format files to any of the following formats:
  - DXT1
  - DXT3
  - DXT5
  - ETC1
  - ETC2
- decompression to uncompressed TGA format of the following compressed format file types:
  - DXT1
  - DXT3
  - DXT5
  - ETC1
  - ETC2

The compressed DXTn format image file will be stored as a DDS file, and the ETCn format image will be stored as a PKM or KTX file.
The TGA format either the RGBA or RGB color model and ETCn format provides an image following the RGB color model RGB888. Note that compressing a TGA image of RGBA format to an ETCn format will result in a loss of alpha values.

6.5.1.2 Supported Formats for Tile and De-Tile Conversions

vTexture supports conversions between linear textures and the tile configurations supported in Vivante hardware:

- Linear no tiling
- Tile 4x4 tile
- Supertile 64x64 tile

The following two tile configurations are supported by hardware, but not routinely utilized in Vivante software:

- Multi-tile A split-tile (possible, but rarely used).
- Multi-supertile A split or multi-supertile surface can occur with GC2000 and above, where, each pixel engine of the multi-pipe renders into a different render buffer and each render buffer is supertiled.

Formats supported for tile format conversions include the following:

- source data
  - BMP
  - TGA
- output data
  - BMP
  - raw data of a specified type. Supported formats are: RGBA8888 BGRA8888 RGB888 BGR888 RGB565 BGR565 ARGB1555

6.5.1.3 vTexture Output Formats

Output from the compress option:

- DXTn format image file will be stored as a DDS file,
- ETC1 and ETC2 format images will be stored as a PKM or KTM file.

Output from the decompress option:

- all supported formats will be decompressed to an uncompressed TGA file.

Output from tile / de-tile options:

- BMP if `-r` not specified
- RAW if `-r` specified.

6.5.1.4 vTexture RAW Output File Format Definition

The Vivante vTexture Tools RAW file is a Vivante-defined file. The file extension is .RAW.
The format consists of the following:

Table 6. Vivante RAW File Header and Pixel Data Definition

Vivante RAW File Header and Pixel Data Definition	Size 16 bytes	Data type	Detail
Width in pixels	4 bytes	INT	Number of pixels
Height in pixels	4 bytes	INT	Number of pixels
Pixel format	4 bytes	INT	Integer value of numeric for a supported format, as defined in gceSURF_FORMAT enumeration:

Supported Format	Numeric
ARGB_1555	208

			<table><tr><td>RGB_565</td><td>209</td></tr><tr><td>BGR_888</td><td>210</td></tr><tr><td>BGRX_8888</td><td>211</td></tr><tr><td>BGRA_8888</td><td>212</td></tr><tr><td>BGR_565</td><td>302</td></tr><tr><td>RGB_888</td><td>303</td></tr><tr><td>RGBX_8888</td><td>305</td></tr><tr><td>RGBA_8888</td><td>306</td></tr></table>These value can also be found in samples(Named TiledTexture).
Tile format	1 byte	BOOL	bit 0: tile bit 1 supertile bit 5: flag for multi- other bits reserved
Supertile format	1 byte	INT	Integer value: 0 = supertile layout mode 0 1 = supertile layout mode 1 2 = supertile layout mode 2
Reserved	2 bytes		not used

6.5.2 Set vTexture Environment Variable

The following table summarizes the only environment variable that vTexture currently expects.

Table 7. vTexture Environment Variables

Environment Variable	Description
PATH	set PATH=%PATH%;"C:\Program Files\Vivante\vTexture\"

6.5.3 Command Line Syntax

Open a Command prompt.
Navigate to the folder which contains the vTexture files (for example, **C:\Program Files (x86)\Vivante\vTexture)**.
Launch the **vTexture** or **vTextureTools** application using the command line syntax described below.

6.5.4 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]
```

6.5.4.1 General Parameters

General parameters:

-h	show help
-src [FILE]	source file - input image path and filename.
	Note:

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.
Note: 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.

6.5.4.2 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.
Note: [TYPE] supported tga. namely, we can only use -d tga

-s compression **[SPEED]** mode for ETCn images:
slow
medium
fast (default)

6.5.4.3 Tile / De-Tile Parameters

These parameters are used for tiling and de-tiling between linear and tiled formats:

-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 (7) are:
rgba8888, bgra8888, rgb888, bgr888, rgb565, bgr565, argb1555.

6.5.4.4 vTexture Syntax Examples

COMPRESS:

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

DECOMPRESS:

```
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 texture

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

Tile linear texture to multi-supertiled texture and output rgb565

```
vTextureTools.exe -st -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
```

6.6 vProfiler and vAnalyzer

vProfiler is a run-time environment for collecting performance statistics of an application and the graphics pipeline. vAnalyzer is a utility for graphically displaying the data gathered by vProfiler and aiding in visual analysis of graphics performance. Used together, these tools can assist software developers in optimizing application performance on Vivante enabled platforms. The GPU includes performance counters that track a variety of GPU

functions. vProfiler gathers data from these counters during runtime and can track data for a range of frames or a single frame from any application. Appendix A contains a partial list of the data gathered by the hardware performance counters. Additional counters are present in the software drivers and hardware access layer.

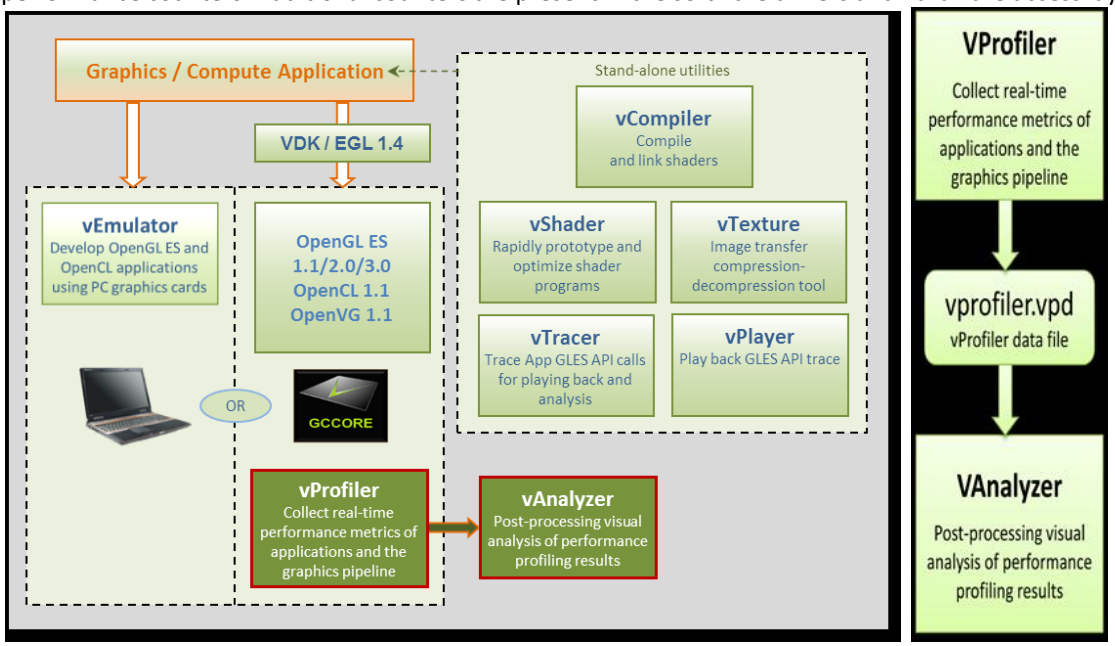


Figure 8. vProfiler performance profiling save data for review in the vAnalyzer visual analyzer

6.6.1 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 will always be the case that one of these resources will be 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 will always promote 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. So when your application performance is GPU-bound, vProfiler and vAnalyzer are the right tools to help you determine why.

Please note that the initial determination regarding which component of your 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: http://en.wikipedia.org/wiki/List_of_performance_analysis_tools.

6.6.2 vProfiler Setup for Linux

The VTK Windows package includes vAnalyzer for the Windows environment. The vProfiler tool can be compiled for Linux, as per the instructions below.
vProfiler stores software and hardware counters captured per frame in the **vprofiler.vpd** file. vAnalyzer reads the .vpd file and allows the user to browse all counters, visualize application performance bottlenecks, and measure system utilization of that application run. Presently, vProfiler does not store frame buffer images due to excessive overhead that changes the behavior of applications.

6.6.2.1 Enable vProfiler Option in Kernel

When building Vivante Graphics Drivers in a Linux environment, the driver is built with vProfiler capability.
To activate vProfiler functionality, there are two ways to do it.
(1) Vivante Graphics Driver is compiled into the the Linux image. We only need to add a sentence "galcore.gpuProfiler=1" in the attribute of bootargs in your boot configuration. For example, if use the nfs to boot the linux, we can configuration boot like: setenv bootargs_nfs 'setenv bootargs ${bootargs} root=/dev/nfs ip=dhcp nfsroot=${serverip}:${nfsroot},v3,tcp video=mxcepdcfb:E060SCM,bpp=16 max17135:pass=2,vcom=-2030000 galcore.gpuProfiler=1'.
(2) Vivante Graphics Driver is built as a Linux Loadable Kernel Module (LKM). While using insmod to insert the GAL kernel driver, use the command line to add the **gpuProfiler=1** option, or add the option into an existing **.sh** script similar to the following:

```
#!/bin/bash
#
insmod /lib/modules/galcore.ko gpuProfiler=1 [OPTIONS]
chmod 777 /dev/galcore
```

Note: The path before file galcore.ko need not to reference this path, it is the path where your .ko file puts.

6.6.2.2 Set vProfiler Environment Variables

The following table summarizes the environment variables that vProfiler supports.

Table 8. vProfiler Environment Variables for Linux

Environment Variable	Description
VIV_PROFILE	Enable (1) or disable (0) vProfiler. (default is disable)
VP_OUTPUT	Specify the output file name of vProfiler
VP_FRAME_NUM	Specify the number of frames dumped by vProfiler
VP_SYNC_MODE	Enable [1] or disable [0] the synchronous mode of vProfiler (default is synchronous enabled)

6.6.2.2.1 VIV_PROFILE

The environment variable **VIV_PROFILE** can be used to enable or disable vProfiler. By default, vProfiler is disabled in the driver. The command below will enable vProfiler. Set **VIV_PROFILE** equal to 0 to disable vProfiler.
export VIV_PROFILE=1

6.6.2.2.2 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
Then the output will be sample_<pid>_<tid>.vpd.

6.6.2.2.3 VP_FRAME_NUM

This takes up a large amount of disk space and also makes it hard to view the data in vAnalyzer. To limit the number of frames to analyze, use the environment variable **VP_FRAME_NUM**. For example, this example setting will make vProfiler dump performance data for the first 100 frames.

```
export VP_FRAME_NUM=100
```

6.6.2.2.4 VP_SYNC_MODE

To get accurate values from the GPU counters, vProfiler needs to commit the GPU commands at the end of every frame; this is so-called synchronous mode. The environment variable **VP_SYNC_MODE** can be used to enable or disable synchronous mode. By default, vProfiler works in synchronous mode. The command below will make vProfiler work in asynchronous mode.

```
export VP_SYNC_MODE=0
```

6.6.3 vProfiler Setup for Android

The vProfiler tool can be set up for use with Android, as per the instructions below.

6.6.3.1 Enable vProfiler Option in Kernel

we only need to add a sentence "galcore.gpuProfiler=1" in the attribute of bootargs in your boot configuration. For example, we can configuration bootargs like: setenv bootargs 'console=ttymxc0,115200 init=/init video=mxcfb0:dev=ldb,bpp=32 video=mxcfb1:off video=mxcfb2:off video=mxcfb3:off vmalloc=400M androidboot.console=ttymxc0 consoleblank=0 androidboot.hardware=freescale cma=512M androidboot.serialno=1e09a1d4d72d78a9 galcore.gpuProfiler=1'.

6.6.3.2 Set Property Options for vProfiler

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

Table 9. vProfiler Set Property Options for Android

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 VP_PROCESS_NAME appname	Run this command in **adb shell** to specify the application you need to profile. Change the appname as needed to profile another application.For example: setprop VP_PROCESS_NAME com.android.gallery3d 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 sdcard, 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** NOTE: 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.
setprop VP_SYNC_MODE 0 **setprop VP_SYNC_MODE 1**	Run this command in **adb shell** to enable or disable synchronous mode. By default, vProfiler works in synchronous mode (=1). To get accurate values from the GPU counters, vProfiler needs to commit the GPU commands at the end of every frame; this is so-called synchronous mode. This example command will make vProfiler work in asynchronous mode: **setprop VP_SYNC_MODE 0**

6.6.4 vProfiler Collects Performance Data

vProfiler is implemented by utilizing hardware counters and a group of instrumentations inserted into drivers that are controlled by compilation flags.

Note: If you execute a GPU program which the number of frame is bigger than variable VP_FRAME_NUM, the program should be stopped automatic, otherwise if you stop the program by the combination key Ctrl + C, then load the .vpd file into vAnalyzer, it will pop-up a window: "The data file may be incomplete".

6.6.4.1 Performance Counters

vProfiler counters are divided into four sets: HAL (Vivante Graphics driver), (shader) program, OpenGL and OpenVG. The counters provide detailed per frame runtime information about the application that can help the developer monitor and tune an application's resource usage. The following table briefly lists the various profile counter sets. For further information, see Appendix A at the end of this document.

Table 10. Performance Counter Types

Counter Type	Description
HALCounters	Driver memory usage
Program	Statistics of the shaders loaded in the GPU (Note: Available only for OpenGL ES 2.0 applications.)
OGLCounters	Various OpenGL (OpenGLES 20 or 11) counters, such as API usage and primitives drawn.
OVGCounters	Various OpenVG counters, such as API usage and primitives drawn.

6.6.5 vAnalyzer Viewing and Analyzing a Run-time Profile

vAnalyzer is a GUI-based tool whose purpose is to help the user view and analyze GPU performance data that was collected using counters during an application run. The performance data from a binary file (*.vpd) written by vProfiler is displayed by vAnalyzer both in text lists and as line graphs. vAnalyzer features a multi-tab, multi-pane, graphical user interface that gives the user several ways to inspect any frame in a captured animation sequence.

6.6.5.1 Loading Profile Files

vAnalyzer accepts a profile for input, which is a **.vpd** file of performance data created by the Vivante vProfiler during a run. For example, the saved file may have a name such as **sample.vpd**.
A **.vpd** file can be selected using the **File/Load Profile Data** menu option.
When a performance profile is loaded, vAnalyzer populates the title bar with information about the GPU and the CPU.
The vAnalyzer screen shot below shows the vAnalyzer GUI immediately after loading a .vpd performance file, and moving the frame number slider to frame 700. By default, the main pane of the vAnalyzer window will display the Charts tab which provides charts for frame time, driver time and GPU idle cycles. Additional charts can be added in the graph window by selecting from the list of variables on the right. Different combinations of counters can be displayed in graphical and list form to illustrate resource utilization for any portion of the profiled application. A second tab contains system information.

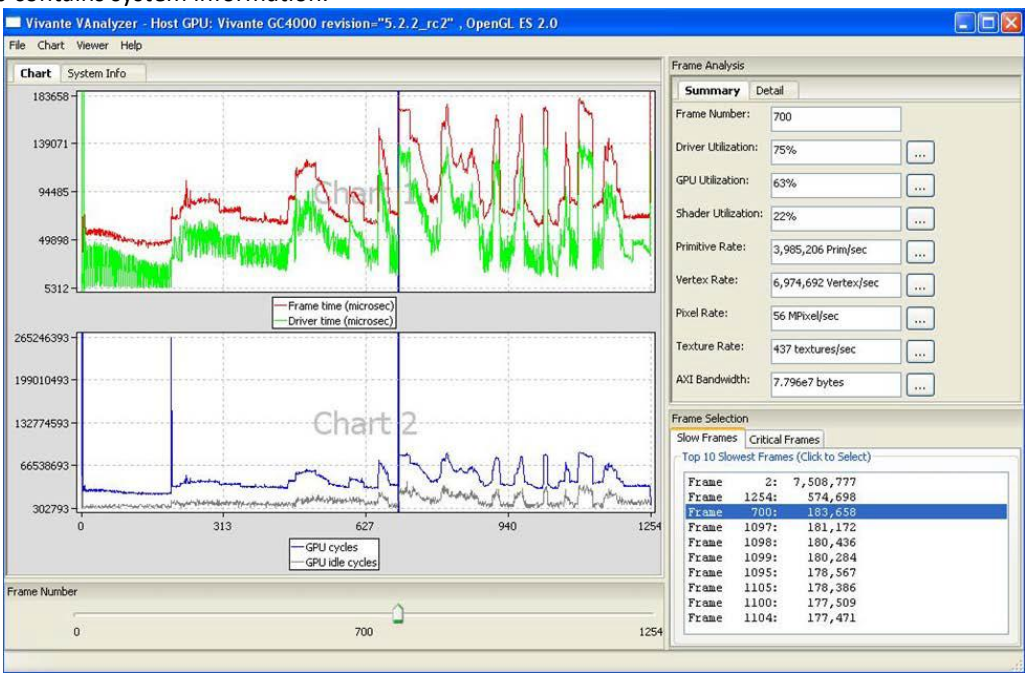


Figure 9. vAnalyzer GUI Main Window

6.6.5.2 vAnalyzer Menu Bar

The vAnalyzer main window opens when a user launches vAnalyzer. The main menu bar contains drop-down menus for **File**, **Chart**, **Viewer** and **Help**. Menu options include the following:
File
– **Load Profile Data**: load a .vpd profile file
– **Export Current Frame Data**: dump all the counters for the frame being viewed to a .cvs file
– **Exit**: exit vAnalyzer

Chart
– **Create New Chart**: create a new chart
– **Customize**: add or delete counters in an existing chart
– **Remove**: delete a chart
– **Export Data From**: dump the counters in a chart to a .csv file
– **Save Chart to PNG**: dump the chart to a .png file
– **View**: zoom in, zoom out or fit the chart

Viewer
– **Function Call Viewer**: display the OpenGL function call statistics
– **Program Viewer**: display the shader program statistics

Help
– **About**: gives version information for vAnalyzer

6.6.6 vAnalyzer Charts

6.6.6.1 vAnalyzer Upper Left Pane: Chart Tab and Menu Options

On the Chart tab in the vAnalyzer main window two default line graphs are displayed.

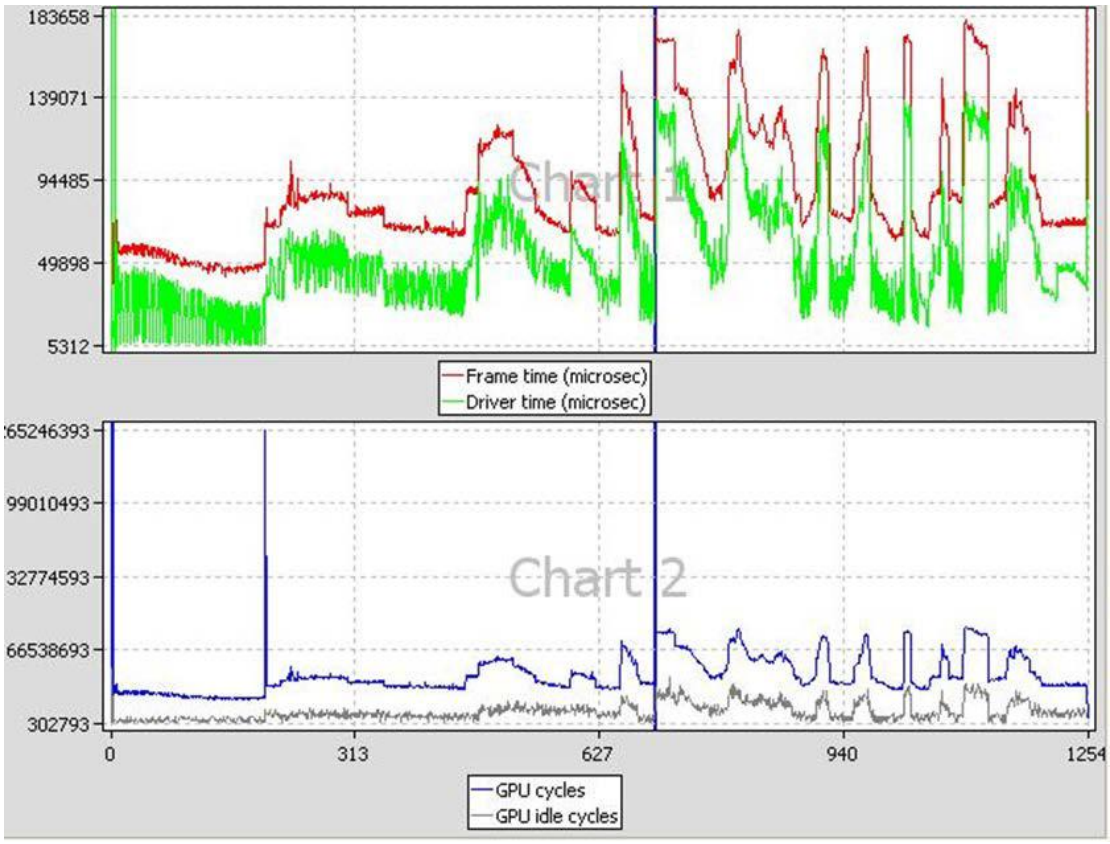


Figure 10. vAnalyzer Performance Counter Charts

6.6.6.2 Chart Customization

Chart / Customize: Additional performance counters can be added to existing chart using the **Customize Chart** dialog window, which can be invoked from the drop menu **Chart/Customize**, or from a pop-up menu which can be invoked by right clicking in the Chart tab area.

Create New Chart: A new chart can be added in a similar way. A single chart can display up to four (4) counters, and the Chart pane can hold up to eight (8) charts. Thus a maximum of thirty-two (32) counters can be graphed at the same time.

Remove Chart: Any chart can be removed from the display using the drop menu **Chart / Remove Chart**.

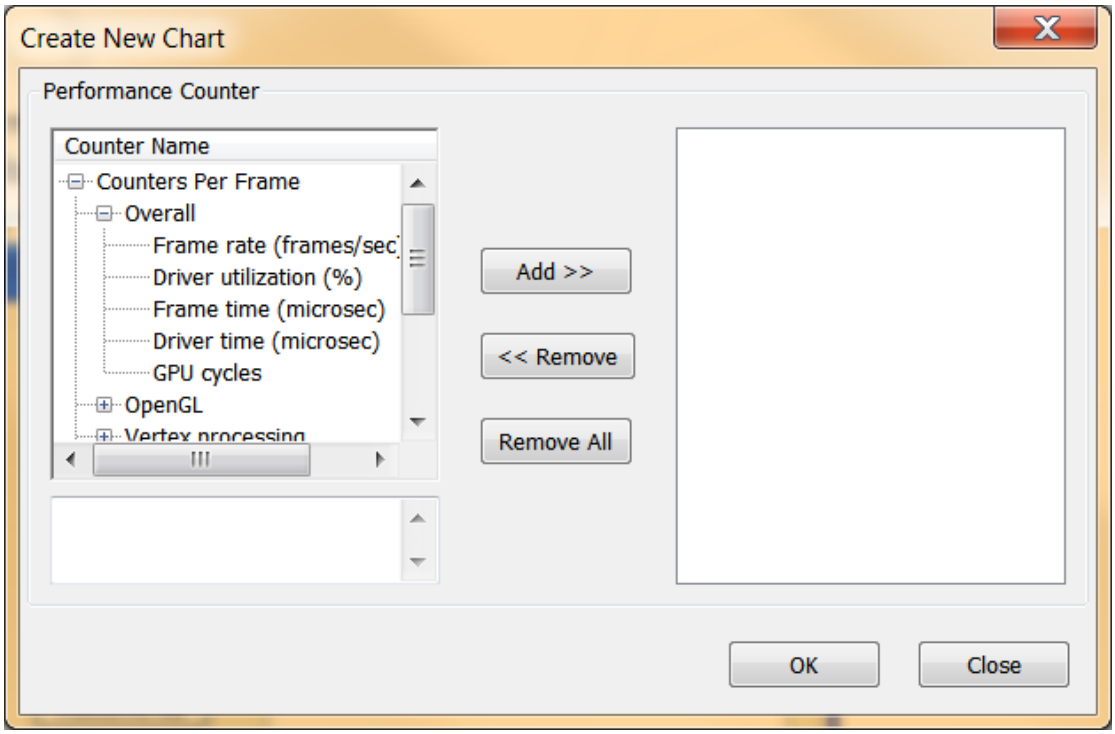


Figure 11. vAnalyzer Create New Chart Dialog

6.6.6.2.1 Chart Components and Navigation

Frame Marker: On the plots displayed in the chart example above there is a blue, vertical frame marker. This marks the current frame position in the timeline.

Zoom:

Zooming in on a set of frames can be achieved in one of two ways.

- One method is to hold down the left mouse button and then sweep a selection box across a range of frame numbers, either on a plot itself or in the common X-axis (frame numbers) in the "Chart" pane, before releasing the mouse button. All charts in the "Chart" pane will zoom in to the same range of frames.
- Alternatively, if your mouse has a scroll wheel, you may also zoom in by rolling the wheel forward--toward the screen.

To zoom out move the scroll wheel backward, toward you.

To reset zoom to the default, which will show the entire timeline, press the escape key (ESC) on the keyboard. The chart view will change to include all frames, from start to end.

6.6.6.2.2 Data Export

The performance counters in a chart can be dumped to a `.csv` file by selecting from the dropdown menu **Chart / Export Data From**. The `.csv` file can be viewed using Excel or another text viewer.

The chart can also be dumped to a **.png** file by selecting from the main menu **Chart / Save chart to PNG**.

6.6.6.3 vAnalyzer Lower Left Pane: Frame Number Slider Bar

In the lower left pane of the vAnalyzer window, there is a Frame Number gauge in the form of a slider bar. Numbers at each end of the bar indicate the initial frame (0) and the last frame available in the loaded sample. By left-clicking and holding the slider, the user can change which frame is selected for analysis. When the frame number is changed, the blue vertical line which indicates the current frame will move, and the reported Frame Number will change in the upper right pane Frame Analysis Summary.

Figure 12. vAnalyzer Frame Number Slider Bar

6.6.6.4 vAnalyzer Left Pane: System Info Tab

When a **.vpd** profile is loaded, system information about the profiled machine populates the fields on the System Info pane. Some information is repeated in the title bar of the main GUI for quick reference.

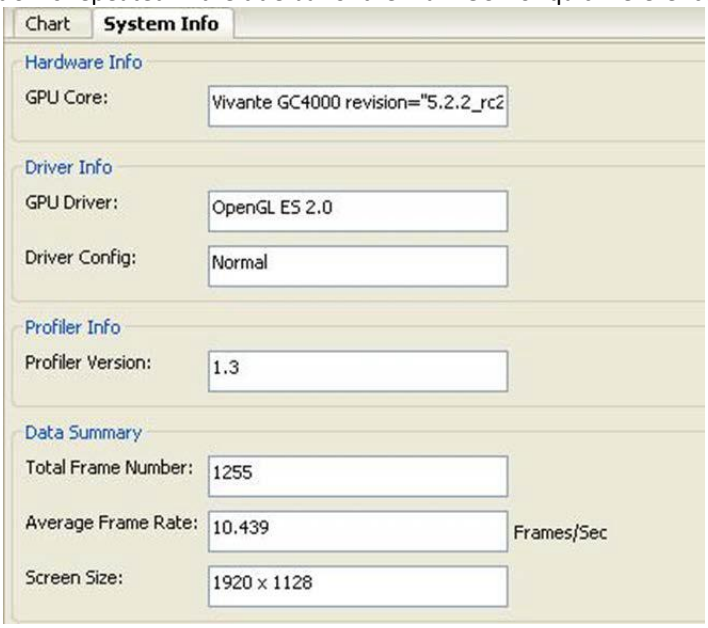


Figure 13. vAnalyzer System Info Tab

6.6.6.5 vAnalyzer Upper Right Pane: Frame Analysis

A selection of performance counters for the frame being viewed are displayed on the right side of the vAnalyzer main GUI. The user can convert this pane to a pop-up window by dragging the pane outside the application window. Drag it back to the right pane area of the application window to reintegrate the pane.

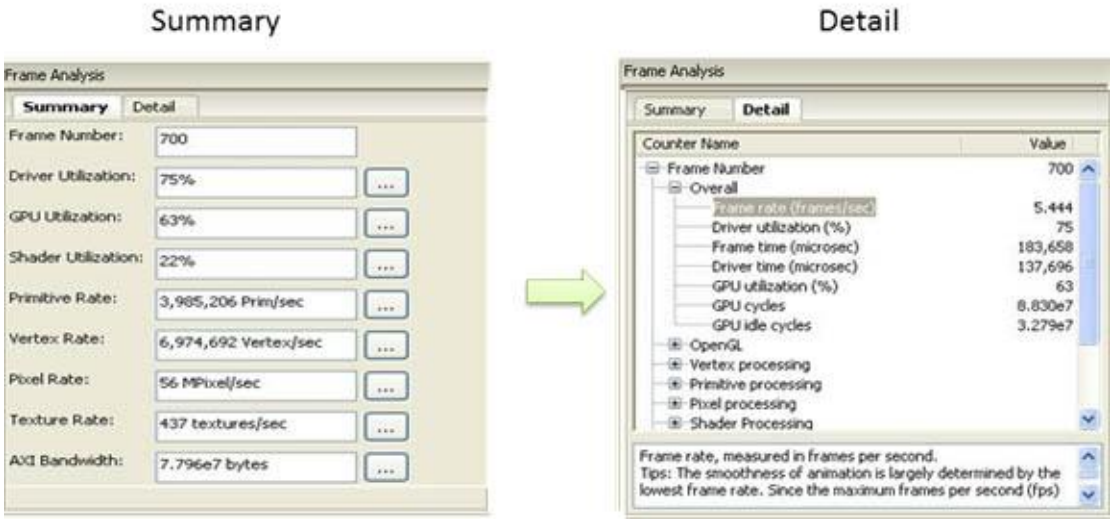


Figure 14. vAnalyzer Frame Analysis Summary and Detail Tabs

6.6.6.5.1 Summary Tab

The **Summary** tab displays summary information for the frame being viewed.
The Selected Frame Number can be changed by entering a new frame number in the text box at the top of the list. The user must press Enter after the input to activate the change. Then Summary values, sliders, and charts all change to reflect the newly entered frame number.
The Summary values below frame number are not directly changeable. They change only when the frame number is changed, either in the Summary tab, by moving the Frame Number slider, or by selecting a frame from the Frame Selection pane. Clicking the "•••" button to the right of a **Summary** item will bring up the corresponding counters in the **Detail** tab. For example, clicking the "•••" button to the right of **Primitive Rate:** switches the view to the **Detail** tab and expands the**Primitive processing** catogory. Clicking the "•••" button for**Driver Utilization:** brings up the pop-p window **Function Call Viewer**.

6.6.6.5.2 Detail Tab

The **Detail** tab reports values for overall performance evaluation, such as Frame Rate, Driver Utilization, and GPU cycles. Additionally counter detail is accessible on this tab. The categories of available counters in the **Detail** tabare: Overall, OpenGL, Vertex processing, Primitive processing, Pixel processing, Shader Processing, Texturing and AXI Bandwidth. Appendix A lists performance as well as hardware counters.

6.6.6.6 vAnalyzer Lower Right Pane: Frame Selection

As with the Frame Analysis pane, this pane can be dragged to display as an independent popup window.

6.6.6.6.1 Slow Frames Tab

The "Slow Frames" tab lists the ten (10) slowest frames in the animation sequence, by time in ascending order from slowest to tenth slowest.

The user can left click on any entry, or can use the arrow keys to move up and down the list, and the display in each of the other GUI panes will change to match that frame.

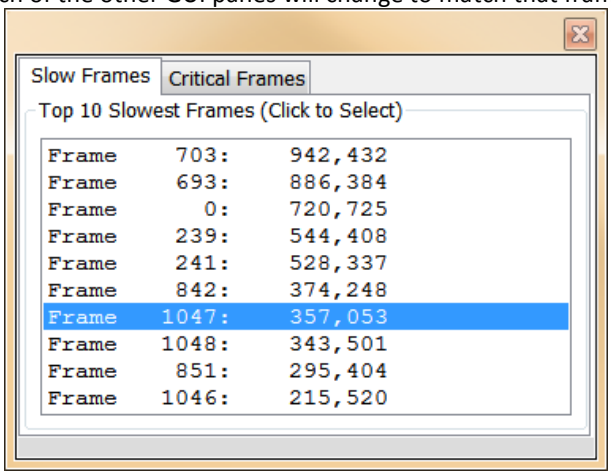


Figure 15. vAnalyzer Frame Selection Slow Frames Tab

6.6.6.6.2 Critical Frames Tab

Select the "Critical Frames" tab to customize the criteria by which a frame is chosen for inspection. One or more of the performance counters can be specified in building the query, which also allows for AND and OR logic. Queries should follow a pattern such as:

"counter name" condition('<','>','==') values.

Users can identify counter names from those in the Frame Analysis pane Detail tab. An example is provided just below the Query input text box.

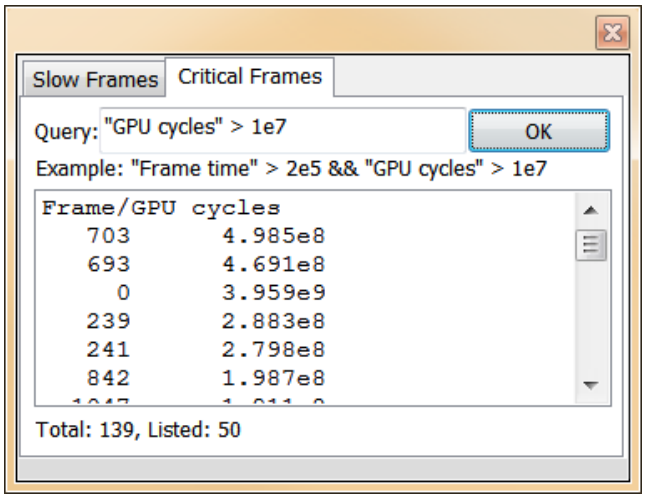
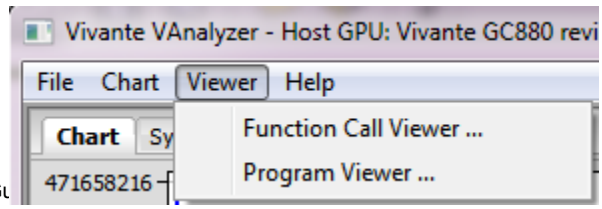


Figure 16. vAnalyzer Frame Selection Critical Frames Tab

6.6.7 vAnalyzer Viewers

The Viewer information pop-up windows can be launched by selecting **Viewer/Function Call Viewer** or **Viewer/Program Viewer** from the Main



menu. The selected Viewer will appear in a pop-up window.

6.6.7.1 OpenGL Function Call Viewer

The OpenGL function call viewer includes three information areas.

• The **OGL Function Name** area contains a table which lists the available OpenGL ES functions by Function Name and Function Type, the run time and the number of times each has been called for this frame. Functions can be sorted by clicking in the column heading area. For example, you can sort the functions by call count or run time by clicking the title bar of "# of Call" or "Time (microsec)".

• The **Top 5 Functions** area contains a histogram which shows the top 5 call count of the listed OpenGL functions.

• The **Property view** area shows the summary when no function is selected; while it shows performance hints for the function when one is selected.

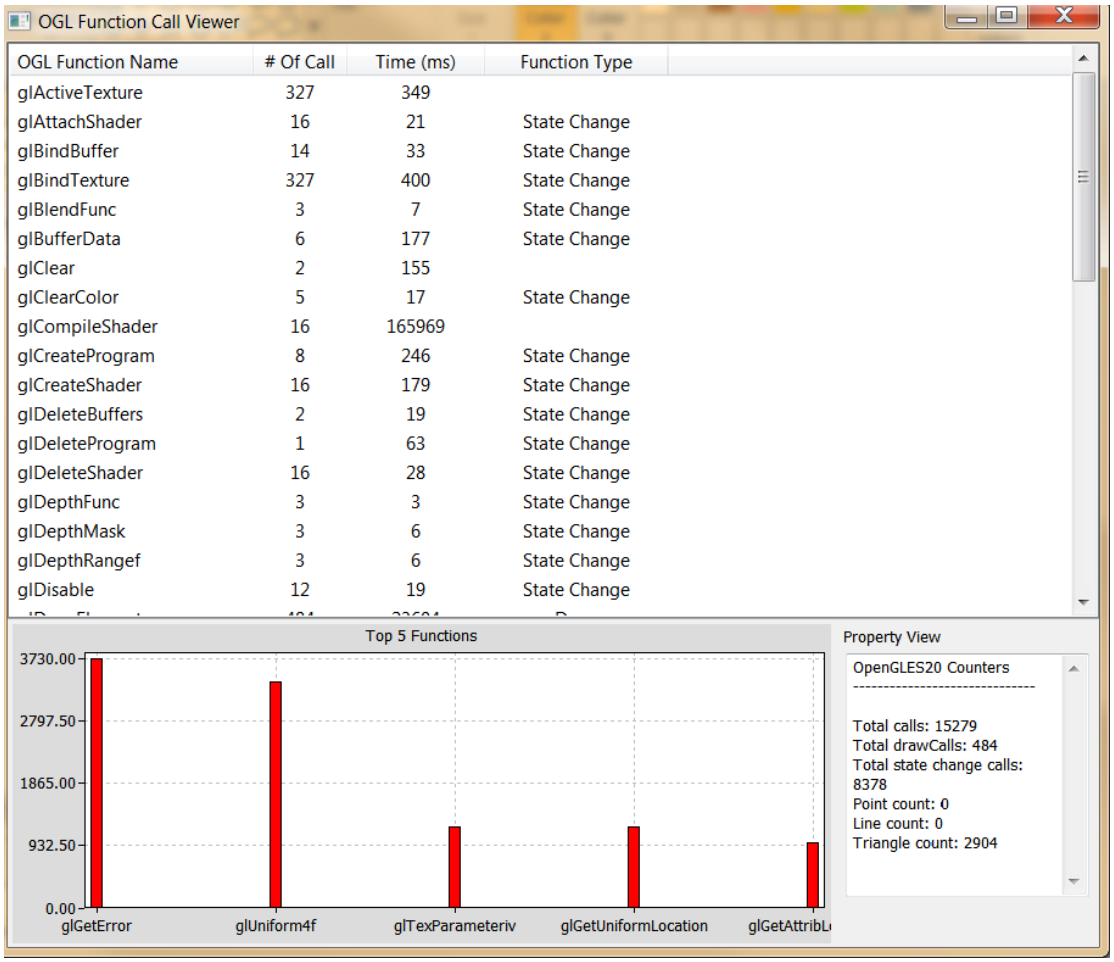


Figure 17. vAnalyzer OpenGL function call viewer window

6.6.7.2 Program Viewer

For a given Frame Number, the Program Viewer gives the statistics for shader programs: uniforms, attributes, and the number of instructions in the shader. This is only for OpenGLES2, ES3 profile data. The description of the item will be displayed in the lower text window when selecting the item. Expand by clicking on **VS** or **PS** submenu to expand the detail for that shader's source code.

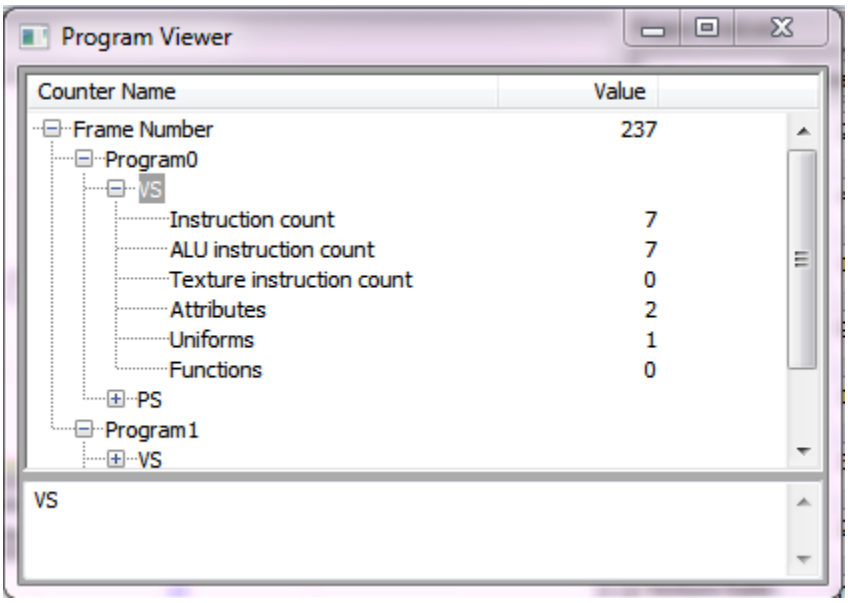


Figure 18. vAnalyzer Program Viewer

6.7 vTracer

The Vivante vTracer package includes two components, vTracer and vPlayer. vTracer is a tool which can record an application's EGL 1.4 and OpenGL ES 1.1/2.0/3.0 API traces on Linux or Android operating systems. vTracer supports two trace modes for Linux: Intercept Trace Mode and Direct Trace Mode. Intercept Trace Mode is the mode used with Android operating systems.
The package also includes vPlayer, a tool for playing back the recorded API trace files.
This section will discuss setup and use of vTracer. The following section will discuss vPlayer.

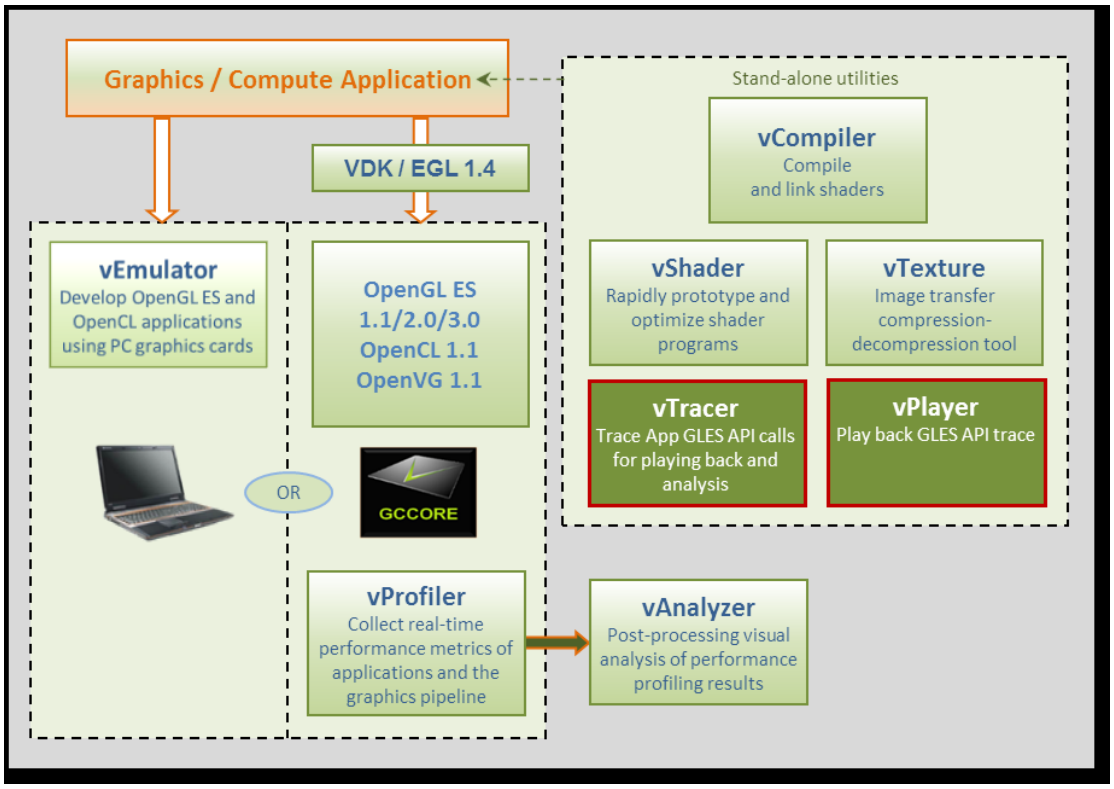


Figure 19. vTracer API trace and vPlayer for trace playback

6.7.1 vTracer Setup

6.7.1.1 Prerequisites

For Linux and Linux Embedded OS, you need to check whether these .so file are exist in /usr/lib:
libEGL.so
libGLESv1_CM.so
libGLESv2.so
For Android, you need to check whether these .so file are exist in /system/lib/egl:
libEGL_VIVANTE.so
libGLESv1_CM_VIVANTE.so
libGLESv2_VIVANTE.so
NOTE: Different versions of vTracer and vPlayer are prepared for Linux backends FBDev.

6.7.1.2 Extract the vTracer Package

Extract the vTracer release package (*.tar.gz). For example:

```
tar -zxvf VivanteVTK-v1.5.6.tgz
unzip vTracer.zip
```

For Linux, the files included in the package which will be used for vTracer setup on Linux include:

```
vTracer/bin/linux/libGLES_vtracer.so
vTracer/bin/linux/libGLES_vlogger.so
vTracer/bin/linux/vtracer.conf
```

For Android, the files included in the package which will be used for vTracer setup on Android include:
`vTracer/bin/android/libGLES_vtracer.so`
`vTracer/bin/android/vtracer.conf`
Files included in the package which will be used for vPlayer include:
`vTracer/bin/linux/vplayer` *(for Linux)*
`vTracer/bin/android/vplayer.apk` *(for Android)*
`vTracer/bin/windows/vplayer.exe` *(for Windows)*

6.7.1.3 Install the vTracer Library

For Linux: Copy the tracer libraries **`libGLES_vtracer.so`** and **`libGLES_vlogger.so`** to the Linux OS **/usr/lib** directory.
For Android: The pre-built vTracer library **libGLES_vtracer.so** for ARM Android devices needs to be uploaded to the Android device directly through adb:

```
adb push vTracer/bin/android/libGLES_vtracer.so /system/lib/egl
```

6.7.1.4 Set Linux Environment Variables for Trace Mode

This step is not required for Android systems.
On Linux, vTracer supports two trace modes: Intercept Trace Mode and Direct Trace Mode.
`libGLES_vtracer.so` `supports` the intercept trace mode and **`libGLES_vlogger.so`** supports the direct trace mode. Either trace mode can be used to generate the application's API trace file, but the two trace modes should not be used simultaneously. Configuration of trace mode is controlled through environment variables.

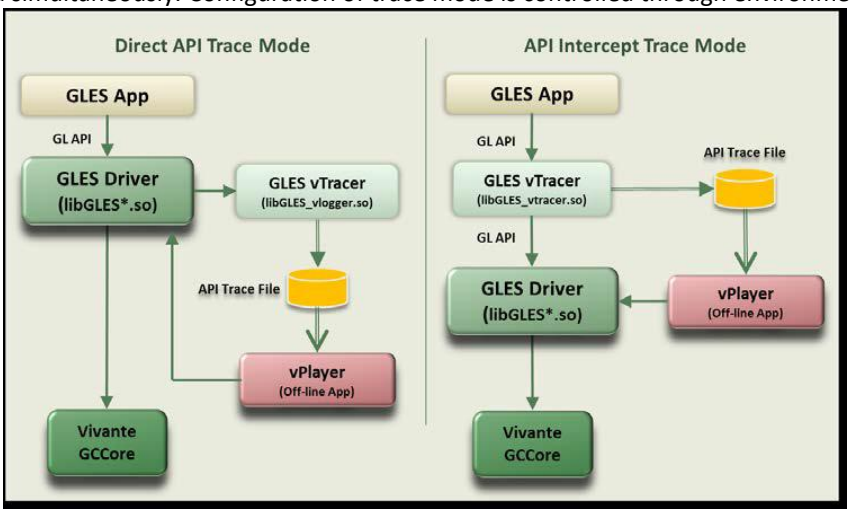


Figure 20. vTracer Has Two API Trace Modes for Linux

6.7.1.4.1 Set vTracer Environment Variable for Intercept Trace Mode

Set the environment variable **LD_PRELOAD** to specify **libGLES_vtracer.so** as the pre-load dynamic linked library:
`export LD_PRELOAD=/usr/lib/libGLES_vtracer.so`
NOTE: Clear the environment variable **LD_PRELOAD** before running vPlayer. Otherwise, vPlayer play back will be traced again!
`export LD_PRELOAD=`

6.7.1.4.2 Set vTracer Environment Variable for Direct Trace Mode

Set the environment variable **VIV_TRACE** to **2** to enable direct trace mode:

```
export VIV_TRACE=2
```

Refer to the table below for other values for VIV_TRACE. By default, vPlayer will reset the environment variable **VIV_TRACE** to **0** to avoid tracing the vPlayer playback again.

You can use the vPlayer option "**--enable-trace**" to enable the **VIV_TRACE** environment variable. When this option is selected, vPlayer honors the current setting for VIV_TRACE. If this option is not used, vPlayer will disable tracing and overwrite the current VIV_TRACE setting.

The following table summarizes the Linux environment variables that vTracer supports.

Table 11. vTracer Environment Variables for Linux

Environment Variable	Description
LD_PRELOAD	**= libGLES_vtracer.so** (specifies the name of the pre-load dynamic linked library **=** NOTE: Clear this variable before running vPlayer.
VIV_TRACE	**=0** Disables all levels of API trace functions. **=1** Enables simple API dump on the console display. This is performed by the Vivante GLES3.0 driver itself (without the need for the external vTracer library **libGLES_vlogger.so**). **=2** Enables Direct Trace Mode

6.7.2 vTracer Configuration

vTracer configuration options can be controlled through the **vtracer.conf** file. Most options are similar for Linux and Android environments.

6.7.2.1 vTracer Optional Configuration on Linux

This step is optional for both trace modes on Linux.
Copy the vTracer configuration file **bin/linux/vtracer.conf** to the local directory (put vtracer.conf file together with your GLES program).
Edit the **vtracer.conf** file which contains multiple settings to control vTracer behavior. Refer to the file for the default settings provided in your specific release.

Table 12. vTracer Configuration Options for Linux

vTracer option	Description
cmdline	Specifies the token used to identify the App process that should be traced. The token is read from **/proc/self/cmdline**. default: **cmdline=?**
codec	Specifies the API trace file format: **binary** or **ascii**. default: **codec=binary**
device	Must be set as **device=stdio** (default)
egl_library	path and filename location of egl library default is **/usr/lib/libEGL.so**
end_frame	Specifies the frame number at which the vTracer trace will end. default for Linux: **end_frame=0** (indicates there is no end limit).
extension	Specifies the API trace file extension. default: **extension=bin**
gles1_library	path and filename location of OpenGL ES 1.0 library default is **/usr/lib/libGLESv1_CM.so**
gles2_library	path and filename location of OpenGL ES 2.0/3.0 library default is **/usr/lib/libGLESv2.so**

overwrite	When **overwrite=1** vTracer will overwrite an existing trace file of the same filename. **default: overwrite=1**.
path	Specifies the path where the API trace files will be placed. default: **path=./**
sync	Set **sync=1** to specify to sync data to disk after every API call. default: **sync=0**
type	Must be set as **type=output** (default)
verbose	Specifies if vTracer prints out verbose messages while tracing. default: **verbose=0**

6.7.2.2 vTracer Configuration on Android

vTracer for Android requires a configuration file to work properly. Edit the vTracer configuration file **bin/android/vtracer.conf** to specify which application process on Android is to be traced.
Only the **cmdline** setting which names the application to be traces is required. Other configuration settings in the **vtracer.conf** file are optional. Default values are usually appropriate.
After any changes to the **vtracer.conf** file are saved, upload the file to the Android device using adb:

```
adb remount
adb push vTracer/bin/android/vtracer.conf /system/lib/egl
```

Table 13. vTracer Configuration Options for Android

vTracer option for Android	Description
REQUIRED SETTING	
cmdline=application_name	Specifies the token used to identify the App process that should be traced. The token is read from **/proc/self/cmdline**. A valid application name must be provided. For example: **cmdline=se.nena.nenamark2** Note that when launching native applications, the application needs to be launched with a full path if the full path is specified here, i.e. if **cmdline=/system/bin/demo** you should start the application with **/system/bin/demo**. Note: Android application names can be looked up with the following command: **adb shell ps**
OPTIONAL SETTINGS	
codec	Specifies the API trace file format: **binary** or **ascii**. default: **codec=binary**.
device	Must be set as **device=stdio** (default)
egl_library	path and filename location of egl library: **/system/lib/egl/libEGL_VIVANTE.so***
end_frame	Specifies the frame number at which the vTracer trace will end. Default value may vary per OS. For Android, to trace only the first 100 frames: **end_frame=100**. Default **end_frame=0** means not set.
extension	Specifies the API trace file extension. default for Android: **extension=ARM.bin**
gles1_library	path and filename location of OpenGL ES 1.0 library **/system/lib/egl/libGLESv1_CM_VIVANTE.so***
gles2_library	path and filename location of OpenGL ES 2.0/3.0 library **/system/lib/egl/libGLESv2_VIVANTE.so***
overwrite	When **overwrite=1** vTracer will overwrite an existing trace file of the

	same file name. default: **overwrite=1**.
path	Specifies the path where the API trace files will be placed. default **path=/sdcard/traces/** Note: the path must have write permission for any users on Android!
sync	Set **sync=1** to specify to sync data to disk after every API call. default: **sync=0**
type	Must be set as **type=output** (default)
verbose	Specifies if vTracer prints out verbose messages while tracing. default: **verbose=0**

*vTracer works on other vendors Android platforms as well. File location values can be set to other vendor's EGL/GLES libraries to trace applications on other platforms.

6.7.3 Enable vTracer

6.7.3.1 Enable vTracer on Linux

vTracer is available for use as soon as the libraries and environment variables are in place.

6.7.3.2 Enable vTracer on Android

For Android versions from 4.4:
For Android 4.4, no **egl.cfg** file is required in **/system/lib/egl**. Once the library **libGLES_vtracer.so** is pushed into the **/system/lib/egl/** directory, vTracer is enabled.
As long as the application specified in the **vtracer.conf** file is launched after the library push, the binary trace file will be generated. If you want to trace an application which was running before that operation, you need to kill the process of that application, and then restart the application.

6.7.4 Using vTracer to Generate API Trace Files for Applications

6.7.4.1 Launch vTracer

On Linux simply launch the application to be traced.
On Android launch the application that is specified in the **vtracer.conf cmdline** option. When launching native applications, the application needs to be launched with its full path if the full path is specified in **vtracer.conf** cmdline, i.e. if `cmdline=/system/bin/demo` you should start the application with `/system/bin/demo`.

6.7.4.2 Android Output Messages

Android output messages can be checked with `logcat`:

```
adb logcat | grep "vtracer"
```

The following output messages indicate vTracer is working properly:

```
D/libEGL ( 2825): loaded /system/lib/egl/libGLES_vtracer.so
I/vtracer ( 2825): Initializing VIVANTE vTracer.
I/vtracer ( 2825): reading configuration file at /system/lib/egl/tracer.conf for pid: 2825
I/vtracer ( 2825): Opened stdio binary trace file
/sdcard/traces/tracer_se.nena.nenamark1.ARM.bin
```

6.7.4.3 Generated Trace Files

A set of API trace files (one trace file per thread) will be generated at the local directory specified by the **vtracer.conf path** option (by default this is **path=./** for Linux or **path=/sdcard/traces/** for Android). The trace files can be generated to a different location by changing the value for **path** in **vtracer.conf**.

The application rendering threads' API trace files can be played back using the Vivante vPlayer tool on Linux, Android or Windows platforms. Note that non-rendering threads' trace files cannot be played back as they don't have EGL/GLES API traces.
On Android the command **`adb pull filename`** can be used to retrieve a generated API trace file from the Android device. For example:

```
adb pull /sdcard/traces/tracer_se.nena.nenamark1.ARM.bin
```

6.8 vPlayer

The vPlayer software tool facilitates play back of the recorded EGL/GLES API trace file in a Linux, Android or Windows environment. To facilitate debugging, the vPlayer application is capable of running trace files created on another platform.
vPlayer for Linux and Windows includes hot key controls, while vPlayer for Android uses touchscreen controls and inputs.

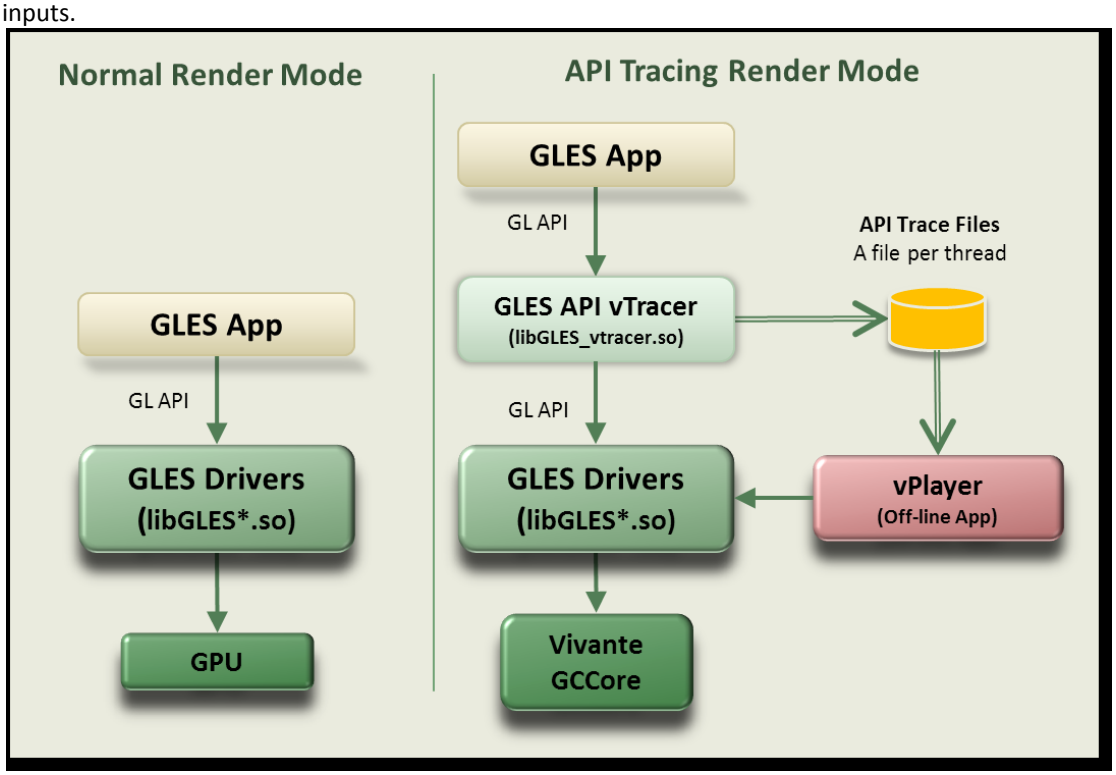


Figure 21. Rendering with vPlayer

6.8.1 vPlayer Setup

The instructions below assume you have already downloaded and expanded the Vivante Toolkit Package and extracted the vPlayer executables from the vTracer zip file. For example:

```
tar -zxvf VivanteVTK-v1.5.5.tgz
unzip vTracer.zip
```

should provide the following vPlayer application files:
`vTracer/bin/linux/vplayer` *(for Linux)*
`vTracer/bin/android/vplayer.apk` *(for Android)*
`vTracer/bin/windows/vplayer.exe` *(for Windows)*

6.8.2 vPlayer Command Line Syntax for Linux and Windows

6.8.2.1 Syntax

On Linux and Windows, the syntax to launch vPlayer from the command line is:
`vplayer [options] tracefilename`
NOTE: If you use Intercept Trace Mode on Linux (by setting a value for the environment variable **LD_PRELOAD**, be sure to clear the environment variable **LD_PRELOAD** to disable pre-loading the **libGLES_vtracer.so** library before running vPlayer. If you fail to do this step, Linux vPlayer play back will be traced again!

6.8.2.2 Input parameters [required]

`tracefilename` `full` path and file name of the API trace file (previously generated with vTracer, usually a .BIN file) to play back. For example,
on Linux: **`./vplayer /workspaces/glb2.7.bin`**
on Android in **vplayer.conf** file: **/sdcard/traces/glb2.7.ARM.bin**

6.8.2.3 Input parameters [optional]

The first two characters in each option are "**`--`**". Options are available for Linux, Android and Windows vPlayer unless indicated otherwise.

`--config N`	Use the specified display EGL config
`--default-config`	Force use of a hard coded default display config
`--default-context`	Use default window surface, config and context
`--dumparray`	Dump array contents in hex
`--dumpbmp`	Dump each frame into a bmp file
`--dumpshader`	Dump GLSL shader source code
`--enable-trace`	Enable **VIV_TRACE** environment variable. If not set, the **VIV_TRACE** environment variable setting value will be ignored. If this option is used, then the current value of the **VIV_TRACE** environment variable will be used.
`--end E`	End frame for rendering (vPlayer will pause at frame E and skip rendering the frames after frame E)
`--error`	Insert eglGetError/glGetError after each API call
`--finish`	Insert glFinish after each API call
`--fps Interval`	Output average FPS and interval FPS information
`--help`	Print/output to console a help list of options for vPlayer
`--repeat N`	Repeat play back N times from --start to --end.
`--showframenumber`	Print the current frame number that vPlayer is playing.
`--start F`	Start frame for rendering (vPlayer will skip rendering the frames before frame F)
`--swapbuf`	Call eglSwapBuffers after each GL draw call (glDrawArrays, glDrawElements, etc.)
`--targetkey`	Reads inputs from target device. By default, vPlayer reads the key presses from the console terminal on the host device. The option `--targetkey` enables vPlayer to read key presses from the target device. On Linux and Windows: allows key inputs to pause, resume, etc. see Hot Keys, below). On Android: By default, touch screen input is enabled on Android for trace playback control (single frame, pause/resume. The option `--targetkey` disables touch screen input to vPlayer on Android.

`--verbose`	Print data parsed from the API trace file.
`--zoom`	Enable zoom to fit the target device screen.

6.8.2.4 vPlayer Hot Keys for Linux and Windows

vPlayer playback can be controlled by hot keys on the keyboard when using Linux and Windows operating systems. By default, vPlayer reads key strokes from the console terminal on the host device. Using the option **--targetkey** enables vPlayer to read key strokes from the target device.
The following Hot Keys are available for control of playback in vPlayer:
Esc Exit application
M Pause / Resume the playback
Q Exit application
SPACE Render a single frame and then pause
When the options **--start F --end E** are used for playback, vPlayer will skip rendering frames before frame F and pause after rendering frame E. Use the **SPACE** key or **M** key to control playback of the remaining frames.

6.8.3 Running vPlayer on Android

6.8.3.1 Launching vPlayer

Touch the vPlayer icon on the Android desktop to launch vPlayer and play back the API trace file specified in the **/sdcard/vivante/vplayer.conf** file.

6.8.3.2 vPlayer Options for Android

Playback options will be those as specified in the **vplayer.conf** configuration file. These match the options available as command line input parameters for Linux and Windows, as described above.
Here is an example vplayer.conf file:

```
    --start 10 --end 100 /sdcard/traces/glb2.7.ARM.bin
```

The vPlayer will skip rendering frames 1 ~ 9 and then pause after rendering frame 10. Then you can use touch to control the playback of frames 11 ~ 100.

6.8.3.3 vPlayer Touch Controls for Rendering on Android

vPlayer supports the following touch controls during API trace file play back:

Single Frame Rendering	Touch the left-half of the screen on the Android device, and vPlayer will render a single frame then pause with every touch.
Pause/Resume Rendering	Touch the right-half of the screen on the Android device, and vPlayer will pause or resume the rendering with every touch.

6.9 Debug and Performance Counters

Availability of some counters will vary depending on core capabilities and software source tree.

6.9.1 AXI Bandwidth

- Read bandwidth (byte)
- Write bandwidth (byte)
- Total bandwidth (byte)
- AXI cycles when read request stalled
- AXI cycles when write request stalled

- AXI cycles when write data stalled

6.9.2 Overall

- Frame rate (frames/sec)
- Driver utilization (%)
- Frame time (microsec)
- Driver time (microsec)
- GPU utilization (%)
- GPU cycles
- GPU idle cycles

6.9.3 OpenGL

- Total calls
- Total draw calls
- Total state change calls
- Point count
- Line count
- Triangle count

6.9.4 Pixel Processing

- Valid pixel count
- % alpha test fail
- % depth&stencil test fail
- Overdraw

6.9.5 Shader Processing

- VS instruction count
- VS branch instruction count
- VS texture fetch count
- Rendered vertex count
- PS instruction count
- PS branch instruction count
- PS texture fetch count
- Rendered pixel count

6.9.6 Texturing

- Total bilinear requests
- Total trilinear requests
- Total texture requests
- Total discarded texture requests

6.9.7 Vertex Processing

- Input vertex count
- Vertics per batch

- Vertics per primitive

6.9.8 Primitive processing

- Input primitive count
- Depth clipped count
- Trivial rejected count
- Culled count
- Output primitive count

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: IMX6GRAPHICUG
Rev. L3.10.31_1.1.0-beta
08/2014