NXP Semiconductors
Reference Manual

Document identifier: IMXRTVGLITEAPIRM
Rev. 0, 22 February 2021

i.MX RT VGLite API Reference Manual

Contents

Chapter 1 Introduction..... 5

Chapter 2 Vivante VGLite Graphics API..... 6
2.1 API Partitions..... 6
2.2 API Files..... 6

Chapter 3 Common Parameters and Error Values..... 7
3.1 Common Parameter Types..... 7
3.2 Enumeration used for Error Reporting..... 7
3.2.1 vg_lite_error_t Enumeration..... 7

Chapter 4 Hardware Product and Feature Information..... 9
4.1 Enumerations for Product and Feature Queries..... 9
4.1.1 vg_lite_feature_t Enumeration..... 9
4.2 Structures for Product and Feature Queries..... 9
4.2.1 vg_lite_info_t Structure..... 9
4.3 Functions for Product and Feature Queries..... 10
4.3.1 vg_lite_get_product_info..... 10
4.3.2 vg_lite_get_info..... 10
4.3.3 vg_lite_get_register..... 10
4.3.4 vg_lite_query_feature..... 11
4.3.5 vg_lite_mem_avail..... 11

Chapter 5 API Control..... 13
5.1 Context Initialization and Control Functions..... 13
5.1.1 vg_lite_set_command_buffer_size..... 13
5.1.2 vg_lite_init..... 13
5.1.3 vg_lite_close..... 14
5.1.4 vg_lite_finish..... 14
5.1.5 vg_lite_flush..... 14

Chapter 6 Pixel Buffers..... 15
6.1 Pixel Buffer Alignment..... 15
6.2 Pixel Cache..... 15
6.3 Internal Representation..... 15
6.4 Pixel Buffer Enumerations..... 15
6.4.1 vg_lite_buffer_format_t Enumeration..... 15
6.4.1.1 Alignment Notes..... 22
6.4.2 vg_lite_buffer_image_mode_t Enumeration..... 23
6.4.3 vg_lite_buffer_layout_t Enumeration..... 23
6.4.4 vg_lite_buffer_transparency_mode_t Enumeration..... 23
6.4.5 vg_lite_swizzle_t Enumeration..... 24
6.4.6 vg_lite_yuv2rgb_t Enumeration..... 24
6.5 Pixel Buffer Structures..... 24
6.5.1 vg_lite_buffer_t Structure..... 24
6.5.2 vg_lite_yuvinfo_t Structure..... 25

6.6 Pixel Buffer Functions....26
6.6.1 vg_lite_allocate....26
6.6.2 vg_lite_free....26
6.6.3 vg_lite_buffer_upload....27
6.6.4 vg_lite_map....27
6.6.5 vg_lite_unmap....27
6.6.6 vg_lite_set_CLUT....28

Chapter 7 Matrices....29
7.1 Matrix Control Float Parameter Type....29
7.2 Matrix Control Structures....29
7.2.1 vg_lite_matrix_t Structure....29
7.3 Matrix Control Functions....29
7.3.1 vg_lite_identity....29
7.3.2 vg_lite_perspective....30
7.3.3 vg_lite_rotate....30
7.3.4 vg_lite_scale....30
7.3.5 vg_lite_translate....31

Chapter 8 BLITs for Compositing and Blending....32
8.1 BLIT Enumerations....32
8.1.1 vg_lite_blend_t Enumeration....32
8.1.2 vg_lite_color_t Parameter....36
8.1.3 vg_lite_filter_t Enumeration....36
8.2 BLIT Structures....36
8.2.1 vg_lite_buffer_t Structure....37
8.2.2 vg_lite_matrix_t Structure....37
8.2.3 vg_lite_path_t Structure....37
8.2.4 vg_lite_rectangle_t Structure....37
8.3 BLIT Functions....37
8.3.1 vg_lite_blit....37
8.3.2 vg_lite_blit_rect....38
8.3.3 vg_lite_clear....39
8.4 Premultiply and Scissor Functions....40
8.4.1 vg_lite_enable_premultiply....40
8.4.2 vg_lite_disable_premultiply....40
8.4.3 vg_lite_enable_scissor....40
8.4.4 vg_lite_disable_scissor....41
8.4.5 vg_lite_set_scissor....41

Chapter 9 Vector Path Control....42
9.1 Vector Path Enumerations....42
9.1.1 vg_lite_format_t Enumeration....42
9.1.2 vg_lite_quality_t Enumeration....42
9.2 Vector Path Structures....42
9.2.1 vg_lite_hw_memory Structure....42
9.2.2 vg_lite_path_t Structure....43
9.3 Vector Path Functions....44
9.3.1 vg_lite_path_calc_length....44
9.3.2 vg_lite_path_append....45
9.3.3 vg_lite_init_path....45
9.3.4 vg_lite_upload_path....46

9.3.5 vg_lite_clear_path....47
9.4 Vector Path Opcodes for Plotting Paths....47

Chapter 10 Vector Based Draw Operations....50
10.1 Draw and Gradient Enumerations....50
10.1.1 vg_lite_blend_t Enumeration....50
10.1.2 vg_lite_color_t Parameter....50
10.1.3 vg_lite_fill_t Enumeration....50
10.1.4 vg_lite_filter_t Enumeration....50
10.1.5 vg_lite_pattern_mode_t Enumeration....50
10.1.6 vg_lite_radial_gradient_spreadmode_t Enumeration....51
10.2 Draw and Gradient Structures....51
10.2.1 vg_lite_buffer_t Structure....51
10.2.2 vg_lite_color_ramp_t Structure....51
10.2.3 vg_lite_linear_gradient_t Structure....52
10.2.4 vg_lite_matrix_t Structure....52
10.2.5 vg_lite_path_t Structure....52
10.2.6 vg_lite_radial_gradient_parameter_t Structure....52
10.2.7 vg_lite_radial_gradient_t Structure....53
10.3 Draw Functions....53
10.3.1 vg_lite_draw....54
10.3.2 vg_lite_draw_gradient....54
10.3.3 vg_lite_draw_radial_gradient....55
10.3.4 vg_lite_draw_pattern....56
10.4 Linear Gradient Initialization and Control Functions....57
10.4.1 vg_lite_init_grad....57
10.4.2 vg_lite_set_grad....58
10.4.3 vg_lite_update_grad....58
10.4.4 vg_lite_get_grad_matrix....59
10.4.5 vg_lite_clear_grad....59
10.5 Radial Gradient Functions....59
10.5.1 vg_lite_set_rad_grad....59
10.5.2 vg_lite_update_rad_grad....60
10.5.3 vg_lite_get_rad_grad_matrix....61
10.5.4 vg_lite_clear_rad_grad....61

Chapter 11 VGLite API Programming Examples....62
11.1 vg_lite_clear Example....62
11.2 vg_lite_blit Example....62
11.3 vg_lite_draw Example....63
11.4 vg_lite_draw_gradient Example....64
11.5 vg_lite_draw_pattern Example....64
11.6 Vector-based Font Rendering Example....65

Chapter 12 Revision history....68

Chapter 1
Introduction

Vivante's platform independent VGLite Graphics API (Application Programming Interface) is designed to support 2D vector and 2D raster-based operations for rendering interactive user interface that may include menus, fonts, curves, and images. Its goal is to provide the maximum 2D vector/raster rendering performance, while keeping the memory footprint to the minimum. The Vivante VGLite Graphics API is a common interface to i.MX RT500 and i.MX RT1170 2D GPUs.

Chapter 2
Vivante VGLite Graphics API

The Vivante VGLite Graphics API is used to control the i.MX RT500 and i.MX RT1170 2D GPUs vector graphics hardware unit, which provides accelerated vector and raster operations.

The Vivante VGLite API is intended for use with i.MX RT500 and i.MX RT1170 2D GPUs hardware. Supported features include: Porter-Duff Blending, Gradient Controls, Fast Clear, Arbitrary Rotations, Path Filling rules, Path painting, and Pattern Path Filling, The API supports only one implicit global application context.

2.1 API Partitions

The Vivante VGLite Graphics API has been designed to allow for fine granularity in memory usage. It is appropriate for those cases where the user wants to use only one of the available rendering classes. The API is partitioned into these independent parts:

- **Initialization** – Used for initializing hardware and software structures.
- **Blit API** – Used for the raster part of rendering.
- **Draw API** – Used for 2D vector-based draw operations.

2.2 API Files

The Vivante VGLite Graphics API functions are defined in the header file **VGLite/inc/vg_lite.h**.

All VGLite enumerations and data types are also defined in **VGLite/inc/vg_lite.h**.

Chapter 3
Common Parameters and Error Values

This section provides an overview of the common parameter types and the enumeration ised for error reporting.

3.1 Common Parameter Types

Vivante VGLite Graphics API uses a naming convention scheme wherein definitions are preceded by ‘vg_lite’.

Below is the list of most proprietary defined types and structures in the drivers:

These are the types currently used by the API.

Table 1. Common parameter types

Name	Typedef	Value
int32_t	int	A signed 32 bit integer
uint32_t	Unsigned int	A unsigned 32 bit integer
VG_LITE_S8	enum vg_lite_format_t	A signed 8 bit integer coordinate
VG_LITE_S16	enum vg_lite_format_t	A signed 16 bit integer coordinate
VG_LITE_S32	enum vg_lite_format_t	A signed 32 bit integer coordinate
vg_lite_float_t	float	A single precision floating point number
vg_lite_color_t	uint32_t	A 32 bit color value The color value specifies the color used in various functions. The color is formed using 8-bit RGBA channels. The red channel is in the lower 8-bit of the color value, followed by the green and blue channels. The alpha channel is in the upper 8-bit of the color value. For L8 target formats, the RGB color is converted to L8 by using the default ITU-R BT.709 conversion rules.

3.2 Enumeration used for Error Reporting

This section provides an overview of the enumeration used for error reporting.

3.2.1 vg_lite_error_t Enumeration

Most functions in the API include an error status via the **vg_lite_error_t** enumeration. API functions return the status of the command and will report **VG_LITE_SUCCESS** if successful with no errors. Possible error values include the values in the table below.

Used in many functions, including initialization, flush, blit, draw, gradient and pattern functions.

Table 2. vg_lite_error_t enumeration

vg_lite_error_t String Values	Description
VG_LITE_GENERIC_IO	Cannot communicate with the kernel driver
VG_LITE_INVALID_ARGUMENT	Invalid argument specified
VG_LITE_MULTI_THREAD_FAIL	Multi-thread/tasks fail
VG_LITE_NO_CONTEXT	No context specified
VG_LITE_NOT_SUPPORT	Function call not supported
VG_LITE_OUT_OF_MEMORY	Out of memory (driver heap)
VG_LITE_OUT_OF_RESOURCES	Out of resources (OS heap)
VG_LITE_SUCCESS	Successful with no errors
VG_LITE_TIMEOUT	Timeout

Chapter 4
Hardware Product and Feature Information

These query functions can be used to identify the product and its key features and to get VGLite driver information.

4.1 Enumerations for Product and Feature Queries

This section provides and overview of the enumerations for product and feature queries.

4.1.1 vg_lite_feature_t Enumeration

The following feature values may be queried for availability in compatible hardware.

Used in information function: vg_lite_query_feature.

Table 3. vg_lite_feature_t enumeration

vg_lite_feature_t String Values	Description
gcFEATURE_BIT_VG_IM_INDEX_FORMAT	Index format support (not supported on i.MX RT1170)
gcFEATURE_BIT_VG_PE_PREMULTIPLY	Premultiply alpha support for image (not supported on i.MX RT500)
gcFEATURE_BIT_VG_SCISSOR	Scissor support
gcFEATURE_BIT_VG_RADIAL_GRADIENT	Radial gradient support (not supported on i.MX RT500)
gcFEATURE_BIT_VG_BORDER_CULLING	Border_culling support
gcFEATURE_BIT_VG_RGBA2_FORMAT	RGBA2222 format support

4.2 Structures for Product and Feature Queries

This section provides an overview of the structures for product and feature queries.

4.2.1 vg_lite_info_t Structure

This structure is used to query VGLite driver information.

Used in function: vg_lite_get_info.

vg_lite_info_t Members	Type	Description
api_version	uint32_t	VGLite API version
header_version	uint32_t	VGLite header version
release_version	uint32_t	VGLite driver release version
reserved	uint32_t	Reserved for future use

4.3 Functions for Product and Feature Queries

This section provides an overview of the functions for product and feature queries.

4.3.1 vg_lite_get_product_info

Description:

This function is used to identify the VGLite compatible product.

Syntax:

```
uint32_t vg_lite_get_product_info (
char                  *name
uint32_t              *chip_id
uint32_t              *chip_rev
);
```

Parameters:

*name	Character array to store the name of the chip.
*chip_id	Stores an ID number for the chip.
*chip_rev	Stores a revision number for the chip.

Returns:

The length of the name string, including the ending '\0'.

4.3.2 vg_lite_get_info

Description:

This function is used to query the VGLite driver information.

Syntax:

```
void vg_lite_get_info (
vg_lite_info_t        *info
);
```

Parameters:

*info	Points to the VGLite driver information structure which includes the API version, header version, and release version.

4.3.3 vg_lite_get_register

Description:

This function can be used to read a GCNanoLiteV AHB register value given the AHB Byte address of a register. Refer to Vivante GCNanoLiteV AHB Register specification documents for register descriptions. The value range of AHB accessible addresses for VGLite cores is usually 0x0 to 0x1FF and 0xA00 to 0xA7F.

Syntax:

```
vg_lite_error_t vg_lite_get_register (
uint32_t              address
uint32_t              *result
);
```

Parameters:

address	Address of the register whose value you want.
*result	The registers value.

Returns:

VG_LITE_SUCCESS. The behavior is undefined if register is outside the range of VGLite core accessible addresses.

4.3.4 vg_lite_query_feature

Description:

This function is used to query if a specific feature is available.

Syntax:

```
uint32_t vg_lite_query_feature (
vg_lite_feature_t     feature
);
```

Parameters:

feature	Feature being queried, as detailed in enum vg_lite_feature_t.

Returns:

Either the feature is not supported (0) or supported (1).

4.3.5 vg_lite_mem_avail

Description:

This function queries whether or not there is any remaining allocated contiguous video memory.

Syntax:

```
vg_lite_error_t vg_lite_mem_avail (
uint32_t           *size
);
```

Parameters:

size	Pointer to the remaining allocated contiguous video memory.

Returns:

Returns VG_LITE_SUCCESS if the query is successful and memory is available. Returns VG_LITE_NO_CONTEXT if the driver is not initialized, or there is no available memory.

Chapter 5
API Control

Before calling any VGLite API function, the application must initialize the VGLite implicit (global) context by calling *vg_lite_init()*, which will fill in a features table, reset the fast-clear buffer, reset the compositing target buffer, as well as allocate the command and tessellation buffers.

NOTE

The vg_lite_init() function does not initialize clocks. Driver users are responsible for ensuring that all necessary clocks are running and attached before calling this function.

The VGLite driver only supports one current context and one thread to issue commands to GCNanoLiteV hardware. The VGLite driver does not support multiple concurrent contexts running simultaneously in multiple threads/processes, as the VGLite kernel driver does not support context switching. A VGLite application can only use a single context at any time to issue commands to GCNanoLiteV hardware. If a VGLite application needs to switch contexts, **vg_lite_close()** should be called to close the current context in the current thread, then **vg_lite_init()** can be called to initialize a new context either in the current thread or from another thread/process.

5.1 Context Initialization and Control Functions

This section provides an overview of the context initialization and control functions.

5.1.1 vg_lite_set_command_buffer_size

Description:

This function is optional. If used, call it before **vg_lite_init** if you want to change the command buffer size.

This function is useful for devices where memory is limited and less than the default. The VGLite Command buffer is set to 64K by default, so that VGLite applications can render more complex paths with better performance. This function can be used to adjust the command buffer size to fit specific application and system/device requirements.

Syntax:

```
vg_lite_error_t vg_lite_set_command_buffer_size (
uint32_t           size
);
```

Parameters:

size	Size of the VGLite Command buffer. Default is 64K.

5.1.2 vg_lite_init

Description:

This function initializes the memory and data structures needed for VGLite draw/blit functions, by allocating memory for the command buffer and a tessellation buffer of the specified size. The tessellation buffer width & height must be a multiple of 16. The tessellation window can be specified based on the amount of memory available in the system and the desired performance. A smaller window can have a lower memory footprint but may result in lower performance. The minimum window that can be used for tessellation is 16x16. If the height or width is less than 0, then no tessellation buffer is created, thus it can be used in a blit-only case.

If this would be the first context that accesses the hardware, the hardware will be turned on and initialized. If a new context needs to be initialized**, vg_lite_close** must be called to close the current context. Otherwise, **vg_lite_init** will return an error.

Syntax:

```
vg_lite_error_t vg_lite_init(
int32_t            tessellation_width,
int32_t            tessellation_height
);
```

Parameters:

tessellation_width	Width of tessellation window. Value should be a multiple of 16; minimum width is 16 pixels, maximum cannot be greater than frame width. If less than or equal to 0, then no tessellation buffer is created, in which case the function is used for a blit init.
tessellation_height	Height of tessellation window. Value should be a multiple of 16; minimum height is 16 pixels; maximum cannot be greater than frame height. If less than or equal to 0, then no tessellation buffer is created, in which case the function is used for a blit init.

5.1.3 vg_lite_close

Description:

This function will deallocate all the resource and free up all the memory that was initialized earlier by the **vg_lite_init** function. It will also turn OFF the hardware automatically if this was the only active context.

Syntax:

```
vg_lite_error_t vg_lite_close (
     void
);
```

5.1.4 vg_lite_finish

Description:

This function explicitly submits the command buffer to the GPU and waits for it to complete.

Syntax:

```
vg_lite_error_t vg_lite_finish (
void
);
```

5.1.5 vg_lite_flush

Description:

This function explicitly submits the command buffer to the GPU without waiting for it to complete.

Syntax:

```
vg_lite_error_t vg_lite_flush (
void
);
```

Returns:

Returns VG_LITE_SUCCESS if the flush is successful. See vg_lite_error_t enum for other return codes.

Chapter 6
Pixel Buffers

This topic provides an overview of the pixel buffer alignment, cache, internal representation, enumerations, structures and functions.

6.1 Pixel Buffer Alignment

The VGLite hardware requires the pixel buffer width to be a multiple of 16 pixels. This requirement applies to all image formats. So the user needs to pad an arbitrary pixel buffer width to a multiple of 16 pixels for VGLite hardware to work correctly. The Byte alignment requirement for a pixel depends on the specific pixel format. See Alignment Notes Table 1. Image Source Alignment Summary later in this document.

The pixel buffer start address alignment requirement varies depending on whether or not the buffer layout format is tiled or linear (vg_lite_buffer_layout_t enum):

- If the format is tiled (4x4 tiled), the start address and stride need to be 64 byte aligned.
- If the format is linear, the start address and stride do not have an alignment requirement.

6.2 Pixel Cache

The Vivante Imaging Engine (IM) includes two fully associative caches. Each cache has 8 lines, each line has 64 bytes. In this case, one cache line can hold either a 4x4-pixel Tile or a 16x1-pixel row.

6.3 Internal Representation

For non 32-bit color formats, each pixel will be extended to 32-bits as such:

If the source and destination formats have the same color format, but differ in the number of bits per color channel, the source channel is multiplied by$(2^{d}- 1)/(2^{s}– 1)$ and rounded to the nearest integer, where:

- **d** is the number of bits in the destination channel
- **s** is the number of bits in the source channel

Example: a b11111 5-bit source channel gets converted to an 8-bit destination b11111000.

The YUV formats are internally converted to RGB. Pixel selection is unified for all formats by using the LSB of the coordinate.

6.4 Pixel Buffer Enumerations

This section provides an overview of the pixel buffer enumerations.

6.4.1 vg_lite_buffer_format_t Enumeration

This enumeration specifies the color format to use for a buffer. This applies to both Image and Render Target. Formats include supported swizzles for RGB. For YUV swizzles, use the related values and parameters in vg_lite_swizzle_t.

NOTE

See Alignment Notes following the value descriptions for alignment requirements summary for the image formats.

Used in structure: vg_lite_buffer_t.

See also vg_lite_blit, vg_lite_clear, vg_lite_draw.

Table 4. vg_lite_buffer_format_t enumeration

<table>
<tr><th>vg_lite_buffer_format_t String Value</th><th>Description</th><th>Supported as Source</th><th>Supported as Dest</th><th>Align (Bytes)</th></tr>
<tr><td>VG_LITE_ABGR8888</td><td>32-bit ABGR format with 8 bits per color channel.
Alpha is in bits 7:0, blue in bits 15:8, green in bits 23:16, and the red channel is in bits 31:24.
<table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>ABGR8888</td><td>R</td><td>G</td><td>B</td><td>A</td></tr></table></td><td>Yes</td><td>Yes</td><td>64</td></tr>
<tr><td>VG_LITE_ARGB8888</td><td>32-bit ARGB format with 8 bits per color channel.
Alpha is in bits 7:0, red in bits 15:8, green in bits 23:16, and the blue channel is in bits 31:24.
<table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>ARGB8888</td><td>B</td><td>G</td><td>R</td><td>A</td></tr></table></td><td>Yes</td><td>Yes</td><td>64</td></tr>
<tr><td>VG_LITE_BGRA8888</td><td>32-bit BGRA format with 8 bits per color channel.
Blue in bits 7:0, green in bits 15:8, red is in bits 23:16, and the alpha channel is in bits 31:24.
<table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>BGRA8888</td><td>A</td><td>R</td><td>G</td><td>B</td></tr></table></td><td>Yes</td><td>Yes</td><td>64</td></tr>
<tr><td>VG_LITE_RGBA8888</td><td>32-bit RGBA format with 8 bits per color channel.
Red is in bits 7:0, green in bits 15:8, blue in bits 23:16, and the alpha channel is in bits 31:24.
<table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>RGBA8888</td><td>A</td><td>B</td><td>G</td><td>R</td></tr></table></td><td>Yes</td><td>Yes</td><td>64</td></tr>
<tr><td>VG_LITE_BGRX8888</td><td>32-bit BGRX format with 8 bits per color channel.
Blue in bits 7:0, green in bits 15:8, red is in bits 23:16, and the X channel is in bits 31:24.</td><td>Yes</td><td>Yes</td><td>64</td></tr>
</table>

Table continues on the next page...

Table 4. vg_lite_buffer_format_t enumeration (continued)

vg_lite_buffer_format_t String Value	Description	Supported as Source	Supported as Dest	Align (Bytes)
	<table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>**BGRX8888**</td><td>X</td><td>R</td><td>G</td><td>B</td></tr></table>			
VG_LITE_RGBX8888	32-bit RGBX format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16, and the X channel is in bits 31:24. <table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>**RGBX8888**</td><td>X</td><td>B</td><td>G</td><td>R</td></tr></table>	Yes	Yes	64
VG_LITE_XBGR8888	32-bit XBGR format with 8 bits per color channel. X channel is in bits 7:0, blue in bits 15:8, green in bits 23:16, and the red channel is in bits 31:24. <table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>**XBGR8888**</td><td>R</td><td>G</td><td>B</td><td>X</td></tr></table>	Yes	Yes	64
VG_LITE_XRGB8888	32-bit XRGB format with 8 bits per color channel. X channel is in bits 7:0, red in bits 15:8, green in bits 23:16, and the blue channel is in bits 31:24. <table><tr><td></td><td>31:24</td><td>23:16</td><td>15:8</td><td>7:0</td></tr><tr><td>**XRGB8888**</td><td>B</td><td>G</td><td>R</td><td>X</td></tr></table>	Yes	Yes	64
VG_LITE_ABGR1555	16-bit ABGR format with 5 bits per color channel and one bit alpha. Alpha channel is in bit 0:0, blue in bits 5:1, green in bits 10:6 and the red channel is in bits 15:11. <table><tr><td></td><td>15:11</td><td>10:6</td><td>5:1</td><td>0:0</td></tr><tr><td>**ABGR5551**</td><td>R</td><td>G</td><td>B</td><td>A</td></tr></table>	Yes	Yes	32

Table continues on the next page...

Table 4. vg_lite_buffer_format_t enumeration (continued)

<table>
<tr><th>vg_lite_buffer_format_t String Value</th><th>Description</th><th>Supported as Source</th><th>Supported as Dest</th><th>Align (Bytes)</th></tr>
<tr><td>VG_LITE_ARGB1555</td><td>16-bit ARGB format with 5 bits per color channel and one bit alpha.
The alpha channel is bit 0:0, red in bits 5:1, green in bits 10:6 and the blue channel is in bits 15:11.
<table><tr><td></td><td>15:11</td><td>10:6</td><td>5:1</td><td>0:0</td></tr><tr><td>ARGB5551</td><td>B</td><td>G</td><td>R</td><td>A</td></tr></table></td><td>Yes</td><td>Yes</td><td>32</td></tr>
<tr><td>VG_LITE_BGRA5551</td><td>16-bit BGRA format with 5 bits per color channel and one bit alpha.
Blue is in bit 4:0, green in bits 9:5,red in bits 14:0 and the alpha channel is bit 15:15.
<table><tr><td></td><td>15:15</td><td>14:10</td><td>9:5</td><td>4:0</td></tr><tr><td>BGRA5551</td><td>A</td><td>R</td><td>G</td><td>B</td></tr></table></td><td>Yes</td><td>Yes</td><td>32</td></tr>
<tr><td>VG_LITE_RGBA5551</td><td>16-bit RGBA format with 5 bits per color channel and one bit alpha.
Red is in bit 4:0, green in bits 9:5, blue in bits 14:0 and the alpha channel is bit 15:15.
<table><tr><td></td><td>15:15</td><td>14:10</td><td>9:5</td><td>4:0</td></tr><tr><td>RGBA5551</td><td>A</td><td>B</td><td>G</td><td>R</td></tr></table></td><td>Yes</td><td>Yes</td><td>32</td></tr>
<tr><td>VG_LITE_BGR565</td><td>16-bit BGR format with 5 and 6 bits per color channel.
Blue is in bits 4:0, green in bits 10:5 and the red channel is in bits 15:11.
<table><tr><td></td><td>15:11</td><td>10:5</td><td>4:0</td></tr><tr><td>BGR565</td><td>R</td><td>G</td><td>B</td></tr></table></td><td>Yes</td><td>Yes</td><td>32</td></tr>
<tr><td>VG_LITE_RGB565</td><td>16-bit RGB format with 5 and 6 bits per color channel.
Red is in bits 4:0, green in bits 10:5 and the blue channel is in bits 15:11.</td><td>Yes</td><td>Yes</td><td>32</td></tr>
</table>

Table continues on the next page...

Table 4. vg_lite_buffer_format_t enumeration (continued)

vg_lite_buffer_format_t String Value	Description	Supported as Source	Supported as Dest	Align (Bytes)
	<table><tr><td></td><td>15:11</td><td>10:5</td><td>4:0</td></tr><tr><td>**RGB565**</td><td>B</td><td>G</td><td>R</td></tr></table>			
VG_LITE_ABGR4444	16-bit ABGR format with 4 bits per color channel. Alpha is in bits 3:0, blue in bits 7:4, green in bits 11:8 and the red channel is in bits 15:12. <table><tr><td></td><td>15:12</td><td>11:8</td><td>7:4</td><td>3:0</td></tr><tr><td>**ABGR4444**</td><td>R</td><td>G</td><td>B</td><td>A</td></tr></table>	Yes	Yes	32
VG_LITE_ARGB4444	16-bit ARGB format with 4 bits per color channel. Alpha is in bits 3:0, red in bits 7:4, green in bits 11:8 and the blue channel is in bits 15:12. <table><tr><td></td><td>15:12</td><td>11:8</td><td>7:4</td><td>3:0</td></tr><tr><td>**ARGB4444**</td><td>B</td><td>G</td><td>R</td><td>A</td></tr></table>	Yes	Yes	32
VG_LITE_BGRA4444	16-bit BGRA format with 4 bits per color channel. Red is in bits 11:8, green in bits 7:4, blue in bits 3:0 and the alpha channel is in bits 15:12. <table><tr><td></td><td>15:12</td><td>11:8</td><td>7:4</td><td>3:0</td></tr><tr><td>**BGRA4444**</td><td>A</td><td>R</td><td>G</td><td>B</td></tr></table>	Yes	Yes	32
VG_LITE_RGBA4444	16-bit RGBA format with 4 bits per color channel. Red is in bits 3:0, green in bits 7:4, blue in bits 11:8 and the alpha channel is in bits 15:12. <table><tr><td></td><td>15:12</td><td>11:8</td><td>7:4</td><td>3:0</td></tr><tr><td>**RGBA4444**</td><td>A</td><td>B</td><td>G</td><td>R</td></tr></table>	Yes	Yes	32

Table continues on the next page...

Table 4. vg_lite_buffer_format_t enumeration (continued)

<table>
<tr><th>vg_lite_buffer_format_t String Value</th><th>Description</th><th>Supported as Source</th><th>Supported as Dest</th><th>Align (Bytes)</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>VG_LITE_ABGR2222</td><td>8-bit BGRA format with 2 bits per color channel.
Alpha is in bits 1:0,blue in bits 3:2,green in bits 5:4 and the red channel is in bits 7:6.
NOTE
Not available for i.MX RT1170.
<table><tr><td></td><td>7:6</td><td>5:4</td><td>3:2</td><td>1:0</td></tr><tr><td>ABGR2222</td><td>R</td><td>G</td><td>B</td><td>A</td></tr></table></td><td>Yes</td><td>Yes</td><td>16</td></tr>
<tr><td>VG_LITE_ARGB2222</td><td>8-bit BGRA format with 2 bits per color channel.
Alpha is in bits 1:0,red in bits 3:2,green in bits 5:4 and the blue channel is in bits 7:6.
NOTE
Not available for i.MX RT1170.
Table 5.
<table><tr><td></td><td>7:6</td><td>5:4</td><td>3:2</td><td>1:0</td></tr><tr><td>ARGB2222</td><td>B</td><td>G</td><td>R</td><td>A</td></tr></table></td><td>Yes</td><td>Yes</td><td>16</td></tr>
<tr><td>VG_LITE_BGRA2222</td><td>8-bit BGRA format with 2 bits per color channel.
Blue is in bits 1:0, green in bits 3:2, red in bits 5:4 and the alpha channel is in bits 7:6.
NOTE
Not available for i.MX RT1170.
<table><tr><td></td><td>7:6</td><td>5:4</td><td>3:2</td><td>1:0</td></tr><tr><td>BGRA2222</td><td>A</td><td>R</td><td>G</td><td>B</td></tr></table></td><td>Yes</td><td>Yes</td><td>16</td></tr>
<tr><td>VG_LITE_RGBA2222</td><td>8-bit RGBA format with 2 bits per color channel.
Red is in bits 1:0, green in bits 3:2, blue in bits 5:4 and the alpha channel is in bits 7:6.</td><td>Yes</td><td>Yes</td><td>16</td></tr>
</table>

Table continues on the next page...

Table 4. vg_lite_buffer_format_t enumeration (continued)

vg_lite_buffer_format_t String Value	Description	Supported as Source	Supported as Dest	Align (Bytes)
	NOTE Not available for RT1170. | | 7:6 | 5:4 | 3:2 | 1:0 | | **RGBA2222** | A | B | G | R |			
VG_LITE_L8	8-bit luminance value. There is no alpha value.	Yes	Yes	16
VG_LITE_YUYV	Packed YUV format, 32-bit for 2 pixels. Y0 is in bits 7:0 and V is in bits 31:23. (available for Source IMAGE only). | | 31:24 | 23:16 | 15:8 | 7:0 | | **YUYV** | V0 | Y1 | U0 | Y0 |	Yes	No	32
VG_LITE_A4	4-bit alpha format. There are no RGB values. **NOTE** Premultiply support available with i.MX RT1170. | | 3:0 | | | **A4** | A | |	Yes	No	8
VG_LITE_A8	8-bit alpha format. There are no RGB values. **NOTE** Premultiply support available with RT1170. . | | 7:0 | | **A8** | A |	Yes	Yes	16

Table 6. Hardware dependent formats

Hardware dependent formats for vg_lite_buffer_format_t	Description	Supported as Source	Supported as Dest	Align (Bytes)
VG_LITE_INDEX_1	1-bit index format.	Yes	No	8
VG_LITE_INDEX_2	2-bit index format.	Yes	No	8
VG_LITE_INDEX_4	4-bit index format.	Yes	No	8
VG_LITE_INDEX_8	8-bit index format.	Yes	No	8

6.4.1.1 Alignment Notes

Source Image Alignment Requirement

The VGLite hardware requires the raster image width to be a multiple of 16 pixels. This requirement applies to all image formats. So the user needs to pad an arbitrary image width to a multiple of 16 pixels for VGLite hardware to work correctly.

The Byte alignment requirement for a pixel depends on the specific pixel format.

Table 7. Image Source Alignment Summary

Image Format	Bits per pixel	Alignment Requirement in Bytes	Supported for Source IMAGE	Supported for Destination
VG_LITE_INDEX1	1	8B	Yes	
VG_LITE_INDEX2	2	8B	Yes	
VG_LITE_INDEX4	4	8B	Yes	
VG_LITE_INDEX8	8	16B	Yes	
VG_LITE_A4	4	8B	Yes	
VG_LITE_A8	8	16B	Yes	Yes
VG_LITE_L8	8	16B	Yes	Yes
VG_LITE_ARGB2222 group	8	16B	Yes	Yes
VG_LITE_RGB565 group	16	32B	Yes	Yes
VG_LITE_ARGB1555 group	16	32B	Yes	Yes
VG_LITE_ARGB4444 group	16	32B	Yes	Yes
VG_LITE_YUY2/UYVY	16	32B	Yes	
VG_LITE_ARGB8888/XRGB8888 group	32	64B	Yes	Yes

Destination Alignment Requirement

- For Pixel Engine (PE) destination, the alignment should be 64B for all tiled (4x4) formats. There is no alignment requirements for linear buffer formats.
- Alignment may also be limited by the alignment requirements of backend modules such as DC (Display Controller).

6.4.2 vg_lite_buffer_image_mode_t Enumeration

Specifies how an image is rendered onto a buffer.

Used in structure: vg_lite_buffer_t.

Table 8. vg_lite_buffer_image_mode_t enumeration

vg_lite_buffer_image_mode_t String Value	Description
VG_LITE_NORMAL_IMAGE_MODE	Image drawn with blending mode
VG_LITE_NONE_IMAGE_MODE	Image input is ignored
VG_LITE_MULTIPLY_IMAGE_MODE	Image is multiplied with paint color

6.4.3 vg_lite_buffer_layout_t Enumeration

Specifies the buffer data layout in memory.

Used in structure: vg_lite_buffer_t.

Table 9. vg_lite_buffer_layout_t enumeration

vg_lite_buffer_layout_t String Value	Description
VG_LITE_LINEAR	linear (scanline) layout. Note: this layout does not have an alignment requirement for the buffer.
VG_LITE_TILED	data is organized in 4x4 pixel tiles. Note: for this layout, the buffer start address and stride need to be 64 byte aligned.

6.4.4 vg_lite_buffer_transparency_mode_t Enumeration

Specifies the transparency mode for a buffer.

Used in structure: vg_lite_buffer_t.

Table 10. vg_lite_buffer_transparency_mode_t enumeration

vg_lite_buffer_transparency_mode_t String Value	Description
VG_LITE_IMAGE_OPAQUE	Opaque image: all image pixels are copied to the VG PE for rasterization
VG_LITE_IMAGE_TRANSPARENT	Transparent image: only the non-transparent image pixels are copied to the VG PE. **NOTE** This mode is only valid when IMAGE_MODE (vg_lite_buffer_image_mode_t) is either VG_LITE_NORMAL_IMAGE_MODE or VG_LITE_MULTIPLY_IMAGE_MODE.

6.4.5 vg_lite_swizzle_t Enumeration

This enumeration specifies the swizzle for the UV components of YUV data.

Used in structure: vg_lite_yuvinfo_t.

Table 11. vg_lite_swizzle_t enumeration

vg_lite_swizzle_t String Value	Description
VG_LITE_SWIZZLE_UV	U in lower bits, V in upper bits
VG_LITE_SWIZZLE_VU	V in lower bits, U in upper bits

6.4.6 vg_lite_yuv2rgb_t Enumeration

This enumeration specifies the standard for conversion of YUV data to RGB data.

Used in structure: vg_lite_yuvinfo_t.

Table 12. vg_lite_yuv2rgb_t enumeration

vg_lite_yuv2rgb_t String Value	Description
VG_LITE_YUV601	YUV Converting with ITC.BT-601 standard
VG_LITE_YUV709	YUV Converting with ITC.BT-709 standard

6.5 Pixel Buffer Structures

This section provides an overview on the pixel buffer structures.

6.5.1 vg_lite_buffer_t Structure

This structure defines the buffer layout for a VGLite image or memory data.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in init functions: vg_lite_allocate, vg_lite_free, vg_lite_buffer_upload, vg_lite_map, vg_lite_unmap.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect, vg_lite_clear.

Used in draw functions: vg_lite_draw, vg_lite_draw_pattern, vg_lite_draw_gradient, vg_lite_draw_radial_gradient.

Table 13. vg_lite_buffer_t structure

vg_lite_buffer_t Members	Type	Description
width	int32_t	Width of buffer in pixels
height	int32_t	Height of buffer in pixels
stride	int32_t	Stride in bytes
tiled	vg_lite_buffer_layout_t	Linear or tiled format for buffer enum
format	vg_lite_buffer_format_t	color format enum

Table continues on the next page...

Table 13. vg_lite_buffer_t structure (continued)

vg_lite_buffer_t Members	Type	Description
handle	void *	memory handle
memory	void *	pointer to the start address of the memory
address	uint32_t	GPU address
yuv	vg_lite_yuvinfo_t	YUV format info struct
image_mode	vg_lite_buffer_image_mode_t	Blit image mode enum
transparency_mode	vg_lite_buffer_transparency_mode_t	Image transparency mode enum

6.5.2 vg_lite_yuvinfo_t Structure

This structure defines the organization of VGLite YUV data.

Used in structure: vg_lite_buffer_t.

Table 14. vg_lite_yuvinfo_t Structure

vg_lite_yuvinfo_t Members	Type	Description
swizzle	vg_lite_swizzle_t	UV swizzle enum
yuv2rgb	vg_lite_yuv2rgb_t	YUV conversion standard enum
uv_planar	uint32_t	UV (U) planar address for GPU, generated by driver
v_planar	uint32_t	V planar address for GPU, generated by driver
alpha_planar	uint32_t	Alpha planar address for GPU, generated by driver
uv_stride	uint32_t	UV (U) stride in bytes
v_stride	uint32_t	V planar stride in bytes
alpha_stride	uint32_t	Alpha stride in bytes
uv_height	uint32_t	UV (U) height in pixels
v_height	uint32_t	V stride in bytes
uv_memory	void *	Logical pointer to the UV (U) planar memory
v_memory	void *	Logical pointer to the V planar memory
uv_handle	void *	Memory handle of the UV (U) planar, generated by driver
v_handle	void *	Memory handle of the V planar, generated by driver

6.6 Pixel Buffer Functions

This section provides an overview of the pixel buffer functions.

6.6.1 vg_lite_allocate

Description:

This function is used to allocate a buffer before using it in either blit or draw functions.

In order for the hardware to access some memory, like a source image or a target buffer, it needs to be allocated first. The supplied vg_lite_buffer_t structure needs to be initialized with the size (width and height) and format of the requested buffer. If the stride is set to zero, this function will fill it in. The only input parameter to this function is the pointer to the buffer structure. If the structure has all the information needed, appropriate memory will be allocated for the buffer.

This function will call the kernel to actually allocate the memory.The memory handle, logical address, and hardware addresses in the vg_lite_buffer_t structure will be filled in by the kernel.

Alignment Note: Though GCNanoLiteV have an alignment requirement of 64 bytes, the VGLite Driver sets alignment to 128 bytes for render target buffer to conform to the alignment requirement of Vivante Display Controller. For source image buffer alignment requirement, see Alignment Notes Table 3 following the vg_lite_buffer_format_t value descriptions.

Syntax:

```
vg_lite_error_t vg_lite_allocate (
vg_lite_buffer_t      *buffer
);
```

Parameters:

*buffer	Pointer to the buffer that holds the size and format of the buffer being allocated. Either the memory or address field needs to be set to a non-zero value to map either a logical or physical address into hardware accessible memory.

Returns:

VG_LITE_SUCCESS if the allocating contiguous buffer is allocated successfully.

VG_LITE_OUT_OF_RESOURCES if there is insufficient memory in the host OS heap for the buffer

VG_LITE_OUT_OF_MEMORY if allocation of a contiguous buffer failed.

6.6.2 vg_lite_free

Description:

This function is used to deallocate the buffer that was previously allocated. This will free up the memory for that buffer.

Syntax:

```
vg_lite_error_t vg_lite_free (
vg_lite_buffer_t      *buffer
);
```

Parameters:

*buffer	Pointer to a buffer structure that was filled in by vg_lite_allocate.

6.6.3 vg_lite_buffer_upload

Description:

The function uploads the pixel data to a GPU memory buffer object. Note that the format of the data (pixel) to be uploaded must be the same as described in the buffer object. The input data memory buffer should contain enough data to be uploaded to the GPU buffer pointed by the input parameter “buffer”.

NOTE

i.MX RT500 only uses data[0] and stride[0] as i.MX RT500 does not support planar YUV formats.

Syntax:

```
vg_lite_error_t vg_lite_buffer_upload (
vg_lite_buffer_t      *buffer,
uint8_t               *data[3],
uint32_t              stride[3]
);
```

Parameters:

*buffer	Pointer to a buffer structure that was filled in by vg_lite_allocate.
*data[3]	Pointer to pixel data. For YUV format, there may be up to 3 pointers.
stride[3]	Stride for the pixel data.

6.6.4 vg_lite_map

Description:

This function is used to map the memory appropriately for a particular buffer. For some operating systems, it will be used to get proper translation to physical or logical address of the buffer needed by the GPU.

If you want the use a frame buffer directly as a target buffer, you need to wrap a vg_lite_buffer_t structure around it and call the kernel to map the supplied logical or physical address into hardware accessible memory. For example, if you know the logical address of the frame buffer, set the memory field of the vg_lite_buffer_t structure with that address and call this function. If you know the physical address, set the memory field to NULL and program the address field with the physical address.

Syntax:

```
vg_lite_error_t vg_lite_map (
vg_lite_buffer_t      *buffer
);
```

Parameters:

*buffer	Pointer to a buffer structure that was filled in by vg_lite_allocate.

6.6.5 vg_lite_unmap

Description:

This function unmaps the buffer and frees any memory resources allocated by a previous call to vg_lite_map.

Syntax:

```
vg_lite_error_t vg_lite_unmap (
vg_lite_buffer_t      *buffer
);
```

Parameters:

*buffer	Pointer to a buffer structure that was filled in by vg_lite_map.

6.6.6 vg_lite_set_CLUT

Description:

This function sets a global context state for Index color image. Once it is set (Not NULL), when an indexed format image is rendered, the image color is obtained from the Color Lookup Table (CLUT)according to the image's pixel indices.

NOTE

Only available on i.MX RT500.

Syntax:

```
vg_lite_error_t vg_lite_set_CLUT (
uint32_t               count,
uint32_t               *colors
);
```

Parameters:

count	This is the count of the colors in the color look up table. For INDEX_1, there can be up to 2 colors in the table; For INDEX_2, there can be up to 4 colors in the table; For INDEX_4, there can be up to 16 colors in the table; For INDEX_8, there can be up to 256 colors in the table.
*colors	This pointer is directly programmed to the command buffer. It will not take effect unless the command buffer is submitted. The color is in ARGB format with A located at the high bits. Note: The VGLite driver does not validate the CLUT data.

Returns:

VG_LITE_SUCCESS since no checking is done.

Chapter 7
Matrices

This part of the API provides matrix controls.

Note: All the transformations in the driver/API are actually the final plane/surface coordinate system. There is no transformation of different coordinate systems with VGLite.

7.1 Matrix Control Float Parameter Type

Name	Typedef	Value
vg_lite_float_t	float	A single precision floating point number

7.2 Matrix Control Structures

This section provides an overview of the matrix control structures.

7.2.1 vg_lite_matrix_t Structure

This structure defines a 3x3 float matrix.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern, vg_lite_identity, vg_lite_scale, vg_lite_translate.

Table 15. vg_lite_matrix_t structure

vg_lite_matrix_t Members	Type	Description
m[3][3]	vg_lite_float_t	3x3 matrix, in [row] [column] order

7.3 Matrix Control Functions

This section provides an overview of the matrix control functions.

7.3.1 vg_lite_identity

Description:

This function loads an identity matrix into a matrix variable.

Syntax:

```
void vg_lite_identity (
vg_lite_matrix_t        *matrix,
);
```

Parameters:

***matrix**	Pointer to the vg_lite_matrix_t structure that will be loaded with an identity matrix

7.3.2 vg_lite_perspective

Description:

This function sets a matrix for perspective transformation.

Syntax:

```
void vg_lite_perspective (
vg_lite_float_t       px,
vg_lite_float_t       py,
vg_lite_matrix_t      *matrix
);
```

Parameters:

px	w0 of the perspective transformation matrix.
py	w1 of the perspective transformation matrix.
***matrix**	Pointer to the vg_lite_matrix_t structure for the perspective transformation.

7.3.3 vg_lite_rotate

Description:

This function rotates a matrix a specified number of degrees.

Syntax:

```
void vg_lite_rotate (
vg_lite_float_t       degrees,
vg_lite_matrix_t      *matrix
);
```

Parameters:

degrees	Number of degrees to rotate the matrix. Positive numbers rotate clockwise. The coordinates for the transformation are given in the surface coordinate system (top-to-bottom orientation). Rotations with positive angles are in the clockwise direction
***matrix**	Pointer to the vg_lite_matrix_t structure that will be rotated.

7.3.4 vg_lite_scale

Description:

This function scales a matrix in both horizontal and vertical directions.

Syntax:

```
void vg_lite_scale (
vg_lite_float_t       scale_x,
vg_lite_float_t       scale_y,
vg_lite_matrix_t      *matrix
);
```

Parameters:

scale_x	Horizontal scale.
scale_y	Vertical scale.
***matrix**	Pointer to the vg_lite_matrix_t structure that will be scaled.

7.3.5 vg_lite_translate

Description:

This function translates a matrix to a new location.

Syntax:

```
void vg_lite_translate (
vg_lite_float_t        x,
vg_lite_float_t        y,
vg_lite_matrix_t       *matrix
);
```

Parameters:

x	X location of the transformation.
y	Y location of the transformation.
***matrix**	Pointer to the vg_lite_matrix_t structure that will be translated.

Chapter 8
BLITs for Compositing and Blending

This part of the API performs the hardware accelerated blit operations.

Compositing rules describes how two areas are combined to form a single area. Blending rules describes how combining the colors of the overlapping areas are combined. VGLite supports two blending operations and a subset of the Porter-Duff operations [PD84]. The Porter-Duff operators assume the pixels have the alpha associated (premultiplied), that is, pixels are premultiplied prior to the blending operation.

NOTE

Be sure to use the vg_lite_query_feature function to determine if your product supports premultiplication.

The source image is copied to the destination window with a specified matrix that can include translation, rotation,scaling, and perspective correction.

- The blit function can be used with or without the blend mode.
- The blit function can be used with or without specifying any color value.
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case do not specify blend mode and color value.

8.1 BLIT Enumerations

8.1.1 vg_lite_blend_t Enumeration

This enumeration defines the blending modes supported by some VGLite API functions. S and D represent source and destination color channels and Sa and Da represent the source and destination alpha channels.

Reference: *Thomas Porter and Tom Duff. Compositing digital images. SIGGRAPH Comput. Graph., 18(3):253–259, January 1984.*

Table 16. Porter Duff Operators and Related vg_lite_blend_t enum Values

Sf/Df	0	1	Sa	1 - Sa
0	clear (n/a)	dst (n/a)	dst-in VG_LITE_BLEND_DST_IN	dst-out VG_LITE_BLEND_SUBTRACT
1	src VG_LITE_BLEND_NONE	plus VG_LITE_BLEND_ADDITIVE	…	src-over VG_LITE_BLEND_SRC_OVER
Da	src-in VG_LITE_BLEND_SRC_IN	…	…	src-atop(n/a)
1 - Da	src-out (n/a)	dst-over VG_LITE_BLEND_DST_OVER	dst-atop (n/a)	xor (n/a)

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

Colors are shown at 100% and 50% opacity.

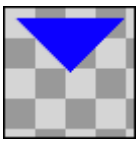
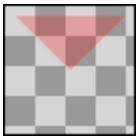
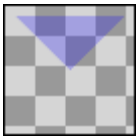
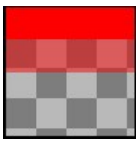
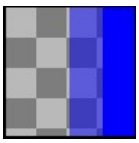
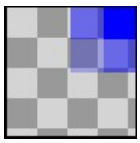
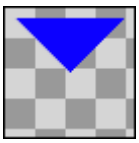
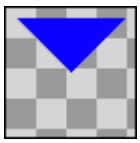
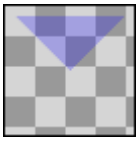
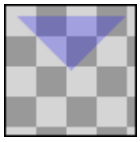
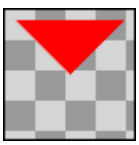
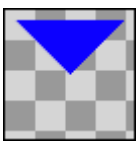
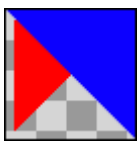
<table>
<tr><th>vg_lite_blend_t String Values</th><th>Description</th></tr>
<tr><td>VG_LITE_BLEND_ADDITIVE</td><td>
Table 17. Porter Duff Compositing Mode: plus
<table>
<tr><th>S + D</th><th></th><th></th><th>= Result</th></tr>
<tr><td>  </td><td>Plus</td><td>  </td><td>  </td></tr>
<tr><td>  </td><td>50%</td><td>  </td><td>  </td></tr>
</table>
</td></tr>
<tr><td>VG_LITE_BLEND_DST_IN</td><td>
Table 18. Porter Duff Compositing Mode: dst-in
<table>
<tr><th>Sa * D</th><th></th><th></th><th>= Result</th></tr>
<tr><td>  </td><td>DstIn</td><td>  </td><td>  </td></tr>
<tr><td>  </td><td>DstIn</td><td>  </td><td>  </td></tr>
<tr><td>  </td><td>50%</td><td>  </td><td>  </td></tr>
</table>
</td></tr>
<tr><td>VG_LITE_BLEND_DST_OVER</td><td>
Table 19. Porter Duff Compositing Mode: dst-over
<table>
<tr><th>(1 – Da) * S + D</th><th></th><th></th><th>= Result</th></tr>
<tr><td>  </td><td>DstOver</td><td>  </td><td>  </td></tr>
</table>
</td></tr>
</table>

Table 19. Porter Duff Compositing Mode: dst-over (continued)

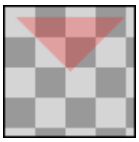
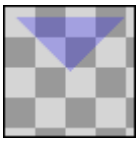
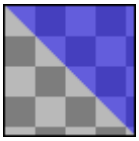
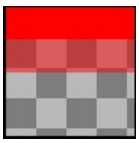
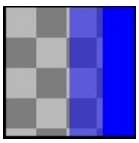
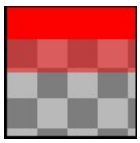
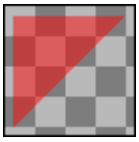
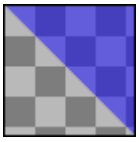
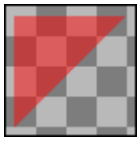
vg_lite_blend_t String Values	Description
	Table 19. Porter Duff Compositing Mode: dst-over (continued)
	 50%  
VG_LITE_BLEND_MULTIPLY	Table 20. Blending Mode: mathematical multiply
	S * (1 – Da) + D * (1 – Sa) + S * D = Result
	 Multiply  
	 50%  
	See https://www.w3.org/TR/compositing-1/#blendingmultiply) make white transparent for diagrams/text.
VG_LITE_BLEND_NONE	Table 21. Porter Duff Compositing Mode: src
	S = Result
	 Src  
	 Src  
	 50%  

Table continues on the next page...

Table continued from the previous page...

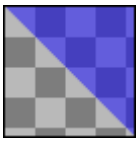
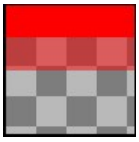
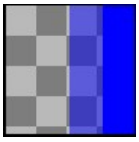
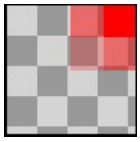
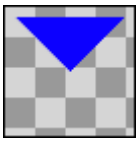
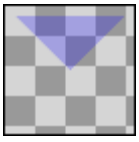
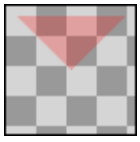
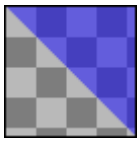
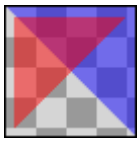
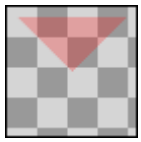
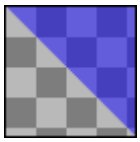
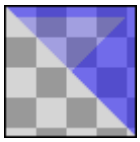
<table>
<tr><th>vg_lite_blend_t String Values</th><th>Description</th></tr>
<tr><td>VG_LITE_BLEND_SCREEN</td><td>
Table 22. Blending Mode: mathematical screen
<table>
<tr><th colspan="3">S + D – S * D</th><th>= Result</th></tr>
<tr><td>  </td><td>Screen</td><td>  </td><td>  </td></tr>
<tr><td>  </td><td>50%</td><td>  </td><td>  </td></tr>
</table>
See https://www.w3.org/TR/compositing-1/#blendingscreen) make black transparent for diagrams/text.
</td></tr>
<tr><td>VG_LITE_BLEND_SRC_IN</td><td>
Table 23. Porter Duff Compositing Mode: src-in, also known as clipping
<table>
<tr><th colspan="3">Da * S</th><th>= Result</th></tr>
<tr><td>  </td><td>SrcIn</td><td>  </td><td>  </td></tr>
<tr><td>  </td><td></td><td>  </td><td>  </td></tr>
<tr><td>  </td><td>50%</td><td>  </td><td>  </td></tr>
</table>
</td></tr>
<tr><td>VG_LITE_BLEND_SRC_OVER</td><td>
Table 24. Porter Duff Compositing Mode: src-over
<table>
<tr><th colspan="3">S + (1 - Sa) * D</th><th>= Result</th></tr>
<tr><td>  </td><td>SrcOver</td><td>  </td><td>  </td></tr>
</table>
</td></tr>
</table>

Table continues on the next page...

Table continued from the previous page...

vg_lite_blend_t String Values	Description
	Table 24. Porter Duff Compositing Mode: src-over (continued)  50%  
VG_LITE_BLEND_SUBTRACT	Table 25. Porter Duff Compositing Mode: dst-out D * (1 – Sa) = Result  DestOut    50%  

8.1.2 vg_lite_color_t Parameter

The common parameter vg_lite_color_t is described in Section 1.4. LINK to Common Parameter Types.

8.1.3 vg_lite_filter_t Enumeration

Specifies the sample filtering mode in VGLite blit and draw APIs.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect,

Used in draw functions: vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

Table 26. vg_lite_filter_t Enumeration

vg_lite_filter_t String Values	Description
VG_LITE_FILTER_POINT	Fetch only the nearest image pixel.
VG_LITE_FILTER_LINEAR	Use linear interpolation along horizontal line.
VG_LITE_FILTER_BI_LINEAR	Use a 2x2 box around the image pixel and perform an interpolation.

8.2 BLIT Structures

8.2.1 vg_lite_buffer_t Structure

Defined under Pixel Buffer Structures. LINK to vg_lite_buffer_t structure.

8.2.2 vg_lite_matrix_t Structure

Defined under Matrix Control Structures LINK to vg_lite_matrix_t structure

8.2.3 vg_lite_path_t Structure

Defined under Vector Path Structures. LINK to vg_lite_path_t structure

8.2.4 vg_lite_rectangle_t Structure

This structure defines the organization of a rectangle of VGLite data.

Used in blit function: vg_lite_clear.

Table 27. vg_lite_rectangle_t Structure

vg_lite_rectangle_t Members	Type	Description
x	int32_t	X Origin of rectangle, left coordinate in pixels
y	int32_t	Y Origin of rectangle, top coordinate in pixels
width	int32_t	X Width of rectangle in pixels
height	int32_t	Y Height of rectangle in pixels

8.3 BLIT Functions

This section provides an overview on BLIT functions.

8.3.1 vg_lite_blit

Description:

This is the blit function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_blit does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth especially with a rotation matrix. VGLite path rendering can be used to achieve high quality coverage sample anti-aliasing (16X, 4X) rendering effect.

NOTE

- The blit function can be used with or without the blend function (vg_lite_blend_t).
- The blit function can be used with or without specifying any color value(vg_lite_color_t).
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case do not specify blend mode and color value.

Syntax:

```
vg_lite_error_t vg_lite_blit (
vg_lite_buffer_t        *target,
vg_lite_buffer_t        *source,
vg_lite_matrix_t        *matrix,
vg_lite_blend_t         blend,
vg_lite_color_t         color
```

```
vg_lite_filter_t        filter
);
```

Parameters:

*target	Points to the vg_lite_buffer_t structure which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*source	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
*matrix	Points to a vg_lite_matrix_t structure that defines the x3 transformation matrix of source pixels into the target. If matrix is NULL, an identity matrix is assumed, meaning the source will be be directly copied on the target at 0,0 location.
blend	Specifies one of the hardware supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the “matrix” parameter is specified with rotation or perspective, and the “blend" parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver will overwrite the application’s setting for the BLIT operation as follows: • If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, the transparency mode will always be set to TRANSPARENT. • If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mix color gets multiplied with each source pixel before blending happens.If you don't need a mix color, set the color parameter to 0.
filter	Specifies the filter type. All formats available in the vg_lite_filter_tenum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

8.3.2 vg_lite_blit_rect

Description:

This is the blit rectangle function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_blit_rect does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth especially with a rotation matrix. VGLite path rendering can be used to achieve high quality coverage sample anti-aliasing (16X, 4X) rendering effect.

NOTE

- The blit_rect function can be used with or without the blend function (vg_lite_blend_t).
- The blit_rect function can be used with or without specifying any color value(vg_lite_color_t).
- The blit_rect function can be used for color conversion with an identity matrix and appropriate formats specified for the source and destination buffers. In this case do not specify blend mode and color value.

Syntax:

```
vg_lite_error_t vg_lite_blit_rect (
vg_lite_buffer_t        *target,
vg_lite_buffer_t        *source,
uint32_t                *rect,
vg_lite_matrix_t        *matrix,
vg_lite_blend_t         *blend,
vg_lite_color_t         color,
vg_lite_filter_t        filter
);
```

Parameters:

*target	Points to the vg_lite_buffer_t structure which defines the destination buffer. See Source Image Alignment Requirement for valid destination color formats for the blit_rect functions.
*source	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit_rect function.
*rect	Specifies the rectangle area of the source image to blit. rect[0]/[1]/[2]/[3] are x, y, width and height of the source rectangle respectively.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of source pixels into the target. If matrix is NULL, an identity matrix is assumed, meaning the source will be copied directly on the target at 0,0 location.
blend	Specifies one of the hardware supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the “matrix” parameter is specified with rotation or perspective, and the “blend" parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver will overwrite the application’s setting for the BLIT operation as follows: • If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, the transparency mode will always be set to TRANSPARENT. • If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mix color gets multiplied with each source pixel before blending happens. If you don't need a mix color, set the color parameter to 0.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

8.3.3 vg_lite_clear

Description:

This function performs the clear operation, clearing/filling the specified buffer (entire buffer or partial rectangle in a buffer) with an explicit color.

Syntax:

```
vg_lite_error_t vg_lite_clear (
vg_lite_buffer_t        *target,
vg_lite_rectangle_t     *rectangle,
vg_lite_color_t         color
);
```

Parameters:

*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the clear function.
*rectangle	Pointer to a vg_lite_rectangle_t structure that specifies the area to be filled. If the rectangle is NULL, the entire target buffer will be filled with the specified color.
color	Clear color, as specified in the vg_lite_color_t enum which is the color value to use for filling the buffer. If the buffer is in L8 format, the RGBA color will be converted into a luminance value.

8.4 Premultiply and Scissor Functions

This section provides an overview of the premultiply and scissor functions.

8.4.1 vg_lite_enable_premultiply

Description:

This function will enable premultiply and return a status error code.

Syntax:

```
vg_lite_error_t vg_lite_enable_premultiply (
void
);
```

8.4.2 vg_lite_disable_premultiply

Description:

This function will enable premultiply and return a status error code.

Syntax:

```
vg_lite_error_t vg_lite_disable_premultiply (
void
);
```

8.4.3 vg_lite_enable_scissor

Description:

This function enables scissor operations for a render targets boundary.

Syntax:

```
vg_lite_error_t vg_lite_enable_scissor (
void
);
```

8.4.4 vg_lite_disable_scissor

Description:

This function disables scissor operations for a render targets boundary.

Syntax:

```
vg_lite_error_t vg_lite_disable_scissor (
void
);
```

8.4.5 vg_lite_set_scissor

Description:

This function is used to set a scissor into a render target so that the region outside the scissor boundary is not drawn.

Syntax:

```
vg_lite_error_t vg_lite_set_scissor (
int32_t x,
int32_t y,
int32_t width,
int32_t height
);
```

Parameters:

x y	X and Y specify the boundary origin for the scissor.
width height	Width and height of the scissor

Chapter 9
Vector Path Control

This section provides overview of the vector path enumerations, structures, functions, and opcodes for plotting paths.

9.1 Vector Path Enumerations

This section provides an overview of vector path enumerations.

9.1.1 vg_lite_format_t Enumeration

Values for vg_lite_format_t are defined in the table Common Parameters Types. LINK to Common Parameters table.

9.1.2 vg_lite_quality_t Enumeration

Specifies the level of hardware assisted anti-aliasing.

Used in structure: vg_lite_path_t.

Used in function: vg_lite_init_path.

Table 28. vg_lite_quality_t Enumeration

vg_lite_quality_t String Values	Description
VG_LITE_HIGH	High quality: 16x coverage sample anti-aliasing
VG_LITE_MEDIUM	Medium quality: 4x coverage sample anti-aliasing
VG_LITE_LOW	Low quality: no anti-aliasing

9.2 Vector Path Structures

This section provides an overview of vector path structures.

9.2.1 vg_lite_hw_memory Structure

This structure simply records the memory allocation info by kernel.

Used in structure: vg_lite_path_t.

Table 29. vg_lite_hw_memory structure

vg_lite_hw_memory_t Members	Type	Description
handle	void *	GPU memory object handle
memory	void *	Logical memory address
address	uint32_t	GPU memory address
bytes	uint32_t	Size of memory
property	uint32_t	Bit 0 is used for path upload:

Table continues on the next page...

Table 29. vg_lite_hw_memory structure (continued)

vg_lite_hw_memory_t Members	Type	Description
		0: Disable path data uploading (always embedded into command buffer). 1: Enable auto path data uploading.

9.2.2 vg_lite_path_t Structure

This structure describes VGLite path data.

Path data is composed of op codes and coordinates. The format for op codes is always VG_LITE_S8. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.

Used in init functions: vg_lite_init_path, vg_lite_upload_path, vg_lite_clear_path, vg_lite_path_append.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

Table 30. vg_lite_path_t Structure

vg_lite_path_t Members	Type	Description
bounding_box[4]	vg_lite_float_t	bounding box for path [0] left [1] top [2] right [3] bottom
quality	vg_lite_quality_t	enum for quality hint for the path, anti-aliasing level
format	vg_lite_format_t	enum for coordinate format
uploaded	vg_lite_hw_memory_t	struct with path data that has been uploaded into GPU addressable memory
path_length	int32_t	number of bytes in the path
path	void *	pointer to path data
path_changed	int32_t	0: not changed; 1: changed.

The coordinate may have the formats listed in the following table.

Table 31. Coordinate format

If vg_lite_format_t	Path data alignment in array should be:
VG_LITE_S8	8 bit
VG_LITE_S16	2 bytes
VG_LITE_S32	4 bytes

Special Notes for Path Objects:

- Endianness has no impact, as it is aligned against the boundaries.
- Multiple contiguous op codes should be packed by the size of the specified data format. E.g., by 2 bytes for VG_LITE_S16 or by 4 bytes for VG_LITE_S32.

For example, since opcodes are 8-bits (1 byte), for 16-bit (2 byte) or 32-bit (4 byte) data types:

…

<opcode1_that_needs_data>

<align_to_data_size>

<data_for_opcode1>

<opcode2_that_doesnt_need_data>

<opcode3_that_needs_data>

<align_to_data_size>

<data_for_opcode3>

…

- Path data in the array should always be 1-, 2, or 4-byte aligned, depending on the format:

For example, for 32-bit (4 byte) data types:

…

<opcode1_that_needs_data>

<pad to 4 bytes>

<4 byte data_for_opcode1>

<opcode2_that_doesnt_need_data>

<opcode3_that_needs_data>

<pad to 4 bytes>

<4 byte data_for_opcode3>

…

9.3 Vector Path Functions

When using a small tessellation window and depending on a path's size, a path might be uploaded to the hardware multiple times because the hardware scanline convert path with the provided tessellation window size, so VGLite path rendering performance might go down. So it is better to set the tessellation buffer size to the most common path size, for example if you only render 24-pt fonts, you can set the tessellation buffer to be 24x24.

All the RGBA color formats available in the vg_lite_buffer_format_t are supported as the destination buffer for the draw function.

9.3.1 vg_lite_path_calc_length

Description:

This function calculates the path command buffer length (in bytes).

The application is responsible for allocating a buffer according to the buffer length calculated with this function. Then the buffer is used by the path as a command buffer. The VGLite driver does not allocate the path command buffer.

Syntax:

```
int32_t vg_lite_path_calc_length (
uint8_t *cmd,
uint32_t count,
```

```
vg_lite_format_t format
);
```

Parameters:

*cmd	Pointer to the opcode array to use to construct the path.
count	The opcode count.
format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.

9.3.2 vg_lite_path_append

Description:

This function assembles the command buffer for the path. The command buffer is allocated by the application and assigned to the path. This function makes the final GPU command buffer for the path based on the input opcodes (cmd) and coordinates (data). Note that the application is responsible to allocate a buffer large enough for the path.

Syntax:

```
int32_t vg_lite_path_append (
vg_lite_path_t *path
uint8_t *cmd,
void *data,
uint32_t seg_count
);
```

Parameters:

*path	Pointer to the path definition.
*cmd	Pointer to the opcode array to use to construct the path.
*data	Pointer to the coordinate data array to use to construct the path.
seg_count	The opcode count.

9.3.3 vg_lite_init_path

Description:

This function initializes a path definition with specified values.

Syntax:

```
vg_lite_error_t vg_lite_init_path (
vg_lite_path_t *path,
vg_lite_format_t data_format,
vg_lite_quality_t quality,
uint32_t path_length,
void *path_data,
vg_lite_float_t min_x,
vg_lite_float_t min_y,
vg_lite_float_t max_x,
```

```
vg_lite_float_t max_y
);
```

Parameters:

*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
data_format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
path_length	The length of the path data (in bytes).
*path_data	Pointer to path data.
min_x min_y max_x max_y	Minimum and maximum x and y values specifying the bounding box of the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

9.3.4 vg_lite_upload_path

Description:

This function is used to upload a path to GPU memory.

In normal cases, the VGLite driver will copy any path data into a command buffer structure during runtime. This does take some time if there are many paths to be rendered. Also, in an embedded system the path data won't change - so it makes sense to upload the path data into GPU memory in such a form that the GPU can directly access it.This function will signal the driver to allocate a buffer that will contain the path data and the required command buffer header and footer data for the GPU to access the data directly.

Syntax:

```
vg_lite_error_t vg_lite_upload_path (
vg_lite_path_t        *path
);
```

Parameters:

*path	Pointer to a vg_lite_path_t structure that contains the path to be uploaded.

Returns:

VG_LITE_OUT_OF_MEMORY if not enough GPU memory is available for buffer allocation.

9.3.5 vg_lite_clear_path

Description:

This function will clear and reset path member values. If the path has been uploaded, it frees the GPU memory allocated when uploading the path.

Syntax:

```
vg)lite_error_t vg_lite_clear_path (
vg_lite_path_t *path
);
```

Parameters:

*path	Pointer to the path definition to be cleared.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

9.4 Vector Path Opcodes for Plotting Paths

The following opcodes are path drawing commands available for vector path data.

A Path operation is submitted to the GPU as [Opcode | Coordinates]. The Operation code is stored as a VG_LITE_S8 while the Coordinates are specified via vg_lite_format_t.

Table 32. . Vector Path Data Opcodes

Opcode	Arguments	Description
0x00	None	END. Finish tessellation. Close any open path.
0x02	(x,y)	MOVE. Move to the given vertex. Close any open path. $$start_x = x$$ $$start_y = y$$
0x03	(Δx,Δy)	MOVE_REL. Move to the given relative point. Close any open path. $$start_x = start_x + \Delta x$$ $$start_y = start_y + \Delta y$$
0x04	(x,y)	LINE. Draw a line to the given point. $$Line(start_x, start_y, x, y)$$ $$start_x = x$$ $$start_y = y$$
0x05	(Δx,Δy)	LINE_REL. Draw a line to the given relative point. $$x = start_x + \Delta x$$

Table continues on the next page...

Table 32. . Vector Path Data Opcodes (continued)

Opcode	Arguments	Description
		$$y = start_y + \Delta y$$ $$Line(start_x, start_y, x, y)$$ $$start_x = x$$ $$start_y = y$$
0x06	(cx,cy) (x,y)	QUAD. Draw a quadratic curve to the given end point using the specified control point. $$Quad(start_x, start_y, cx, cy, x, y)$$ $$start_x = x$$ $$start_y = y$$
0x07	(Δcx,Δcy) (Δx,Δy)	QUAD_REL. Draw a quadratic curve to the given relative end point using the specified relative control point. $$cx = start_x + \Delta cx$$ $$cy = start_y + \Delta cy$$ $$x = start_x + \Delta x$$ $$y = start_y + \Delta y$$ $$Quad(start_x, start_y, cx, cy, x, y)$$ $$start_x = x$$ $$start_y = y$$
0x08	(cx-$_1$,cy$_1$) (cx$_2$,cy$_2$) (x,y)	CUBIC. Draw a cubic curve to the given end point using the specified control points. $$Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$$ $$start_x = x$$ $$start_y = y$$
0x09	(Δcx-$_1$,Δcy$_1$) (Δcx$_2$,Δcy$_2$) (Δx,Δy)	CUBIC_REL. Draw a cubic curve to the given relative end point using the specified relative control points. $$cx_1 = start_x + \Delta cx_1$$ $$cy_1 = start_y + \Delta cy_1$$ $$cx_2 = start_x + \Delta cx_2$$ $$cy_2 = start_y + \Delta cy_2$$

Table continues on the next page...

Table 32. . Vector Path Data Opcodes (continued)

Opcode	Arguments	Description
		$x = start_x + \Delta x$ $y = start_y + \Delta y$ $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$

Chapter 10
Vector Based Draw Operations

This part of the API performs the hardware accelerated draw operations.

10.1 Draw and Gradient Enumerations

This section provides an overview of draw and gradient enumerations.

10.1.1 vg_lite_blend_t Enumeration

This enumeration is detailed under the Blit section. LINK to vg_lite_blend_t enumeration.

10.1.2 vg_lite_color_t Parameter

The common parameter vg_lite_color_t is described in Section 1.4 Common Parameter Types.

LINK to vg_lite_color_t color parameter description.

10.1.3 vg_lite_fill_t Enumeration

This enumeration is used to specify the fill rule to use. For drawing any path, the hardware supports both non-zero and odd-even fill rules.

To determine whether any point is contained inside an object, imagine drawing a line from that point out to infinity in any direction such that the line does not cross any vertex of the path. For each edge that is crossed by the line, add 1 to the counter if the edge is crossed from left to right, as seen by an observer walking across the line towards infinity, and subtract 1 if the edge crossed from right to left. In this way, each region of the plane will receive an integer value.

The non-zero fill rule says that a point is inside the shape if the resulting sum is not equal to zero. The even/odd rule says that a point is inside the shape if the resulting sum is odd, regardless of sign.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

Table 33. vg_lite_fill_t enumeration

vg_lite_fill_t String Values	Description
VG_LITE_FILL_NON_ZERO	Non-zero fill rule. A pixel is drawn if it crosses at least one path pixel.
VG_LITE_FILL_EVEN_ODD	Even-odd fill rule. A pixel is drawn if it crosses an odd number of path pixels.

10.1.4 vg_lite_filter_t Enumeration

Defined under Blit. LINK to vg_lite_filter_t enumeration.

10.1.5 vg_lite_pattern_mode_t Enumeration

Defines how the region outside the image pattern is filled for the path.

Used in function: vg_lite_draw_gradient, vg_lite_draw_pattern.

Table 34. vg_lite_pattern_mode_t enumeration

vg_lite_pattern_mode_t String Values	Description
VG_LITE_PATTERN_COLOR	Fill the outside of the pattern by color
VG_LITE_PATTERN_PAD	The color of the pattern border is expanded to fill the region outside the pattern.

10.1.6 vg_lite_radial_gradient_spreadmode_t Enumeration

Defines the radial gradient padding mode.

Used in structure: vg_lite_radial_gradient_t.

Table 35. vg_lite_radial_gradient_spreadmode_t enumeration

vg_lite_radial_gradient_spreadmode_t String Values	Description
VG_LITE_RADIAL_GRADIENT_SPREAD_FILL = 0	Coordinates outside the gradient area filled with black color.
VG_LITE_RADIAL_GRADIENT_SPREAD_PAD	The area is filled with the closest stop color.
VG_LITE_RADIAL_GRADIENT_SPREAD_REPEAT	The gradient is repeated outside the gradient area.
VG_LITE_RADIAL_GRADIENT_SPREAD_REFLECT	The gradient is reflected outside the gradient area.

10.2 Draw and Gradient Structures

This section provides an overview of the draw and gradient structures.

10.2.1 vg_lite_buffer_t Structure

Defined under Pixel Buffer Structures. LINK to vg_lite_buffer_t structure.

10.2.2 vg_lite_color_ramp_t Structure

This structure defines the stops for the radial gradient. The five parameters provide the offset and color for the stop. Each stop is defined by a set of floating point values which specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].

The define for the max number of radial gradient stops is #define MAX_COLOR_RAMP_STOPS 256.

Used in radial gradient structure: vg_lite_radial_gradient_t.

Table 36. vg_lite_color_ramp_t Structure

vg_lite_color_ramp_t Members	Type	Description
stop	vg_lite_float_t	Offset value for the color stop
red	vg_lite_float_t	Red color channel value for the color stop
green	vg_lite_float_t	Green color channel value for the color stop
blue	vg_lite_float_t	Blue color channel value for the color stop

Table continues on the next page...

Table 36. vg_lite_color_ramp_t Structure (continued)

vg_lite_color_ramp_t Members	Type	Description
alpha	vg_lite_float_t	Alpha color channel value for the color stop

10.2.3 vg_lite_linear_gradient_t Structure

This structure defines the organization of a linear gradient in VGLite data. The linear gradient is applied to filling a path. It will generate a 256x1 image according the settings.

Used in init and draw functions: vg_lite_init_grad, vg_lite_set_grad, vg_lite_update_grad, vg_lite_get_grad_matrix, vg_lite_clear_grad, vg_lite_draw_gradient.

Table 37. vg_lite_linear_gradient_t structure

vg_lite_linear_gradient_t Constants	Type	Description
VLC_MAX_GRAD	int32_t	Constant. Maximum number of gradient colors = 16.
VLC_GRADBUFFER_WIDTH	int32_t	Constant. Width of the internal color ramp = 256.

Table 38. vg_lite_linear_gradient_t structure members

vg_lite_linear_gradient_t Members	Type	Description
colors[VLC_MAX_GRAD]	uint32_t	Color array for the gradient
count	uint32_t	Number of colors
stops[VLC_MAX_GRAD]	uint32_t	Number of color stops, from 0 to 255
matrix	vg_lite_matrix_t	Struct for the matrix to be used to transform the gradient color ramp
image	vg_lite_buffer_t	Image object struct to represent the color ramp

10.2.4 vg_lite_matrix_t Structure

Defined under Matrix Structures. LINK to vg_lite_matrix_t structure.

10.2.5 vg_lite_path_t Structure

Defined under Vector Path Structures. LINK to vg_lite_path_t structure.

10.2.6 vg_lite_radial_gradient_parameter_t Structure

This structure defines the gradient radius and the X and Y coordinates for the center and focal points of the gradient.

Used in radial gradient structure: vg_lite_radial_gradient_t.

Table 39. vg_lite_radial_gradient_parameter_t structure

vg_lite_radial_gradient_parameter_t Members	Type	Description
cx	vg_lite_float_t	X coordinate of the gradient's center point
cy	vg_lite_float_t	Y coordinate of the gradient's center point
fx	vg_lite_float_t	X coordinate of the gradient's focal point
fy	vg_lite_float_t	Y coordinate of the gradient's focal point
r	vg_lite_float_t	Radius of the gradient

10.2.7 vg_lite_radial_gradient_t Structure

This structure defines the application of the radial gradient to fill a path. (from November 2020

Used in radial gradient functions: vg_lite_draw_gradient, vg_lite_set_rad_grad, vg_lite_update_rad_grad, vg_lite_get_rad_grad, vg_lite_clear_rad_grad

Table 40. vg_lite_radial_gradient_t structure

vg_lite_radial_gradient_t Members	Type	Description
count	uint32_t	Count of colors, up to 256
matrix	vg_lite_matrix_t	Structure which specifies the transform matrix for the gradient
image	vg_lite_buffer_t	Structure which specifies the image for rendering as a gradient pattern
radialGradient	vg_lite_radial_gradient_parameter_t	Structure which specifies the location of the gradient's center point, focal point and radius
vgColorRampLength	uint32_t	Color ramp parameters for gradient paints provided to the driver
vgColorRamp[MAX_COLOR_RAMP_STOPS]	vg_lite_color_ramp_t	Structure which specifies the color ramp.
intColorRampLength	uint32_t	Converted internal color ramp
intColorRamp[MAX_COLOR_RAMP_STOPS+2]	vg_lite_color_ramp_t	Structure which specifies the Internal color ramp.
colorRampPremultiplied	uint32_t	If this value is set to 1, the color value of vgColorRamp will be multiplied by the alpha value of vgColorRamp.
SpreadMode	vg_lite_radial_gradient_spreadmode_t	Enum which specifies the tiling mode that is applied to the pixels out of the image after transformation.

10.3 Draw Functions

This section provides an overview of the draw functions.

10.3.1 vg_lite_draw

Description:

Performs a hardware accelerated 2D vector draw operation.

The size of the tessellation buffer can be specified, and that size will be aligned to the minimum required alignment of the hardware by the kernel. If you make the tessellation buffer smaller, less memory will be allocated, but a path might be sent down to the hardware multiple times because the hardware will walk the target with the provided tessellation window size, so performance might be lower. It is good practice to set the tessellation buffer size to the most common path size. For example, if all you do is render up to 24-pt fonts, you can set the tessellation buffer to be 24x24.

NOTE

- All the color formats available in the vg_lite_buffer_format_t enum are supported as the destination buffer for the draw function.
- Strokes are not supported by the hardware. They need to be converted to paths before being used in the draw API.

Syntax:

```
vg_lite_error_t vg_lite_draw (
    vg_lite_buffer_t      *target,
    vg_lite_path_t        *path,
    vg_lite_fill_t        fill_rule,
    vg_lite_matrix_t      *matrix,
    vg_lite_blend_t       blend,
    vg_lite_color_t       color
);
```

Parameters:

*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the draw function.
*path	Pointer to the vg_lite_path_tstructure containing path data which describes the path to draw. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*matrix	Pointer to a vg_lite_matrix_tstructure that defines the **affine** transformation matrix of the path. If matrix is NULL, an identity matrix is assumed. **Note:** non-affine transformation is not supported for vg_lite_draw, so a perspective transformation matrix has no effect on path.
blend	Select one of the hardware supported blend modes in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
color	The color applied to each pixel drawn by the path.

10.3.2 vg_lite_draw_gradient

Description:

This function is used to fill a path with a linear_gradient according to specified fill rules. The specified path will be transformed according to the selected matrix and filled with the gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_gradient (
    vg_lite_buffer_t               *target,
    vg_lite_path_t                 *path,
    vg_lite_fill_t                 fill_rule,
    vg_lite_matrix_t               *matrix,
    vg_lite_linear_gradient_t      *grad,
    vg_lite_blend_t                blend
);
```

Parameters:

*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw for the linear gradient. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed which is usually a bad idea since the path can be anything.
*grad	Pointer to the vg_lite_linear_gradient_t structure which contains the values to be used to fill the path.
blend	Specified the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).

10.3.3 vg_lite_draw_radial_gradient

Description:

This function is used to fill a path with a radial gradient according to specified fill rules. The specified path will be transformed according to the selected matrix and filled with the gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_radial_gradient (
    vg_lite_buffer_t               *target,
    vg_lite_path_t                 *path,
    vg_lite_fill_t                 fill_rule,
    vg_lite_matrix_t               *path_matrix,
    vg_lite_radial_gradient_t      *grad,
    vg_lite_color_t                paint_color,
    vg_lite_blend_t                blend,
    vg_lite_filter_t               filter
);
```

Parameters:

*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw for the linear gradient. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path_matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed which is usually a bad idea since the path can be anything.
*grad	Pointer to the vg_lite_radial_gradient_t structure which contains the values to be used to fill the path.
paint_color	Specifies the paint color enum vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL, which set by function vg_lite_set_rad_grad. When pixels are out of the image after transformation, this paint_color is applied to them. See also enum vg_lite_radial_gradient_spreadmode_t.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

10.3.4 vg_lite_draw_pattern

Description:

This function fills a path with an image pattern. The path will be transformed according to the specified matrix and filled with the transformed image pattern.

Syntax:

```
vg_lite_error_t vg_lite_draw_pattern (
vg_lite_buffer_t         *target,
vg_lite_path_t           *path,
vg_lite_fill_t           fill_rule,
vg_lite_matrix_t         *matrix0,
vg_lite_buffer_t         *source,
vg_lite_matrix_t         *matrix1,
vg_lite_blend_t          blend,
vg_lite_pattern_mode_t   pattern_mode,
vg_lite_color_t          pattern_color,
vg_lite_filter_t         filter
);
```

Parameters:

*target	Pointer to the vg_lite_buffer_t structure that defines the path to draw.
*path	Pointer to the vg_lite_path_tstructure containing path data which describes the path to draw. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.

Table continues on the next page...

Table continued from the previous page...

fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*matrix0	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path.If matrix is NULL, an identity matrix is assumed, which is usually a bad idea since the path can be anything.
*source	Pointer to the vg_lite_buffer_tstructure that describes the source of the image pattern.
*matrix1	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target.If matrix is NULL, an identity matrix is assumed, meaning the source will be copied directly onto the target at 0,0 location.
blend	Specifies one of the hardware supported blend modes to be applied to each drawn pixel in the image. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
pattern_mode	Specifies the vg_lite_pattern_mode_t value which defines how the region outside the image pattern is to be filled.
pattern_color	Specifies a 32bpp ARGB color to be applied to the fill outside the image pattern area when the pattern_mode value is VG_LITE_PATTERN_COLOR.
filter	Specifies the filter type. All formats available in the vg_lite_filter_tenum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

10.4 Linear Gradient Initialization and Control Functions

This part of the API performs linear gradient operations.

A color gradient (color progression, color ramp) is a smooth transition between a set of colors (color stops) that is done along a line (linear, or axial color gradient) or radially, along concentric circles (radial color gradient). The color transition is done by linear interpolation between two consecutive color stops.

10.4.1 vg_lite_init_grad

Description:

This function initializes the internal buffer for the linear gradient object with default settings for rendering.

Syntax:

```
vg_lite_error_t vg_lite_init_grad (
vg_lite_linear_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_linear_gradient_t structure which defines the gradient to be initialized. Default values are used.

10.4.2 vg_lite_set_grad

Description:

This function is used to set values for the members of the vg_lite_linear_gradient_t structure.

NOTE

The vg_lite_set_grad API adopts the following rules to set the default gradient colors if the input parameters are incomplete or invalid.

- If no valid stops have been specified (e.g., due to an empty input array, out-of-range, or out-of-order stops), a stop at 0 with (R, G, B, α) color (0.0, 0.0, 0.0, 1.0) (opaque black) and a stop at 1 with color (1.0, 1.0, 1.0, 1.0) (opaque white) are implicitly defined.
- If at least one valid stop has been specified, but none has been defined with an offset of 0, an implicit stop is added with an offset of 0 and the same color as the first user-defined stop.
- If at least one valid stop has been specified, but none has been defined with an offset of 1, an implicit stop is added with an offset of 1 and the same color as the last user-defined stop.

Syntax:

```
vg_lite_error_t vg_lite_set_grad (
      vg_lite_linear_gradient_t *grad,
      uint32_t count,
      uint32_t *colors,
      uint32_t *stops
      );
```

Parameters:

*grad	Pointer to the vg_lite_linear_gradient_t structure to be set.
count	This is the count of the colors in the linear gradient. The maximum color stop count is defined by VLC_MAX_GRAD which is 16.
*colors	Specifies the color array for the gradient stops. The color is in ARGB8888 format with alpha in the upper byte.
*stops	Pointer to the gradient stop offset.

Returns:

Always returns VG_LITE_SUCCESS.

10.4.3 vg_lite_update_grad

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_linear_gradient_t object has an image buffer which is used to render the gradient pattern. The image buffer will be created or updated with the corresponding grad parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_grad (
vg_lite_linear_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_linear_gradient_t structure which contains the update values to be used for the object to be rendered.

10.4.4 vg_lite_get_grad_matrix

Description:

This function is used to get a pointer to the gradient object's transformation matrix. This allows an application to manipulate the matrix to facilitate correct rendering of the gradient path.

Syntax:

```
vg_lite_error_t vg_lite_get_grad_matrix (
vg_lite_linear_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_linear_gradient_t structure which contains the matrix to be retrieved.

10.4.5 vg_lite_clear_grad

Description:

This function is used to clear the values of a linear gradient object and free the image buffer's memory.

Syntax:

```
vg_lite_error_t vg_lite_clear_grad (
vg_lite_linear_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_linear_gradient_t structure which is to be cleared.

10.5 Radial Gradient Functions

NOTE

There is no init function required for radial gradients. Buffer initialization is done through the vg_lite_update_rad_grad function.

10.5.1 vg_lite_set_rad_grad

Description:

This function is used to set the values for the radial linear gradient definition

Syntax:

```
vg_lite_error_t vg_lite_set_rad_grad (
vg_lite_radial_gradient_t *grad,
uint32_t count,
vg_lite_color_ramp_t *vgColorRamp,
```

```
vg_lite_radial_gradient_parameter_t radialGradient,
vg_lite_radial_gradient_spreadmode_t SpreadMode,
uint8_t colorRampPremultiplied
);
```

Parameters:

*grad	Pointer to the vg_lite_radial_gradient_t structure for the radial gradient which will be set
count	This is the count of the color stops in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is currently 256.
*vgColorRamp	Pointer to the vg_lite_color_ramp_t structure which defines the stops for the radial gradient. The five parameters provide the offset and color for the stop. Each stop is defined by a set of floating point values which specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].
radialGradient	The radial gradient parameters are supplied as a vector of 5 floats in the order {cx,cy,fx,fy,r}. Parameters(cx,cy) specify the center point, (fx,fy) the focal point and r the radius. See structure vg_lite_radial_gradient_parameter_t.
SpreadMode	The tiling mode that is applied to pixels out of the paint after transformation. See enum vg_lite_radial_gradient_spreadmode_t.
colorRampPremultiplied	Controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form. If this value is set to 1, the color value of vgColorRamp will be multipled by the alpha value of vgColorRamp.

Returns:

Returns VG_LITE_INVALID_ARGUMENTS to indicate the parameters are wrong.

10.5.2 vg_lite_update_rad_grad

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_radial_gradient_t object has an image buffer which is used to render the gradient pattern. The image buffer will be created or updated with the corresponding gradient parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_rad_grad (
vg_lite_radial_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_radial_gradient_t structure which contains the update values to be used for the object to be rendered.

10.5.3 vg_lite_get_rad_grad_matrix

Description:

This function is used to get a pointer to the radial gradient object's transformation matrix. This allows an application to manipulate the matrix to facilitate correct rendering of the gradient path.

Syntax:

```
vg_lite_error_t vg_lite_get_rad_grad_matrix (
vg_lite_radial_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_radial_gradient_t structure which contains the matrix to be retrieved.

10.5.4 vg_lite_clear_rad_grad

Description:

This function is used to clear the values of a radial gradient object and free the image buffer's memory.

Syntax:

```
vg_lite_error_t vg_lite_clear_rad_grad (
vg_lite_radial_gradient_t *grad,
);
```

Parameters:

*grad	Pointer to the vg_lite_radial_gradient_t structure which is to be cleared.

Chapter 11
VGLite API Programming Examples

This section provides an overview of VGLite API programming examples.

11.1 vg_lite_clear Example

The following code snippet demonstrates the basic flow of a VGLite application program and the usage of the `vg_lite_clear API`. First, the program initializes the VGLite API with:

```
error = vg_lite_init(0, 0);
```

NOTE

The tessellation buffer width and height are defined as (0, 0) in this `vg_lite_init` API call. This program cannot use the path rendering `vg_lite_draw` APIs. Only clear and blit APIs can be used in this program.

After initialization, the program allocates a 256x256 render buffer with a format of VG_LITE_RGB565.

```
buffer.width = 256;

buffer.height = 256;

buffer.format = VG_LITE_RGB565;

error = vg_lite_allocate(&buffer);

fb = &buffer;
```

It clears the entire render buffer with blue color first with the vg_lite_clear API.

```
error = vg_lite_clear(fb, NULL, 0xFFFF0000);
```

Then it clears a 64x64 square at the position (64, 64) relative to the top-left origin of the render buffer.

```
vg_lite_rectangle_t rect = { 64, 64, 64, 64 };

error = vg_lite_clear(fb, &rect, 0xFF0000FF);
```

After that, it calls vg_lite_finish to flush the commands to GCNanoLiteV hardware and then frees up the allocated render buffer. Finally it calls vg_lite_close to destroy the VGLite context which is initialized by `vg_lite_init`.

```
vg_lite_finish();

vg_lite_free(&buffer);

vg_lite_close();
```

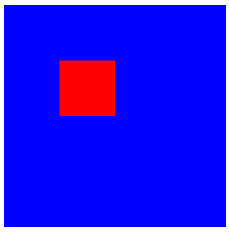


Figure 1. Example using vg_lite_clear

11.2 vg_lite_blit Example

The following example test program demonstrates the usage of the `vg_lite_blit` API. It clears a 320x480 render buffer with blue background color first, then it blits six 256x256 icon images to six different positions in the render buffer with a blit matrix for each

icon. The blit matrix scales the original icon image to a proper size and translates the scaled icon to the right position in the render buffer. The `vg_lite_blit` API call is set as VG_LITE_BLEND_SRC_OVER so the icon image pixels with alpha value 0xFF cover the background blue color.

```
vg_lite_blit(fb, &icons[icon_id], &icon_matrix, VG_LITE_BLEND_SRC_OVER, 0, VG_LITE_FILTER_POINT);
```

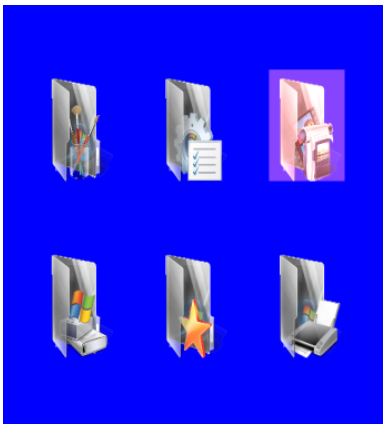


Figure 2. Example using vg_lite_blit

11.3 vg_lite_draw Example

This section demonstrates the usage of the `vg_lite_draw` API with which it draws a highlighted rectangle on the top-right icon in above image. The program defines a path (path_data[]) for a 10x10 square bounding box, and it sets up a proper “highlight_matrix” to translate/scale the 10x10 square to cover the top-right icon. The `vg_lite_draw` API call uses blend parameter `VG_LITE_BLEND_SRC_OVER` and blend color 0x22444488 (alpha value 0x22) to draw a semi-transparent rectangle on the top-right icon.

```
static char path_data[] = {

2, 0, 0, // moveto 0, 0

4, 10, 0, // lineto 10, 0

4, 10, 10, // lineto 10, 10

4, 0, 10, // lineto 0, 10

0, // end

};

static vg_lite_path_t path = {

{-10, -10, 10, 10}, // bounding box left, top, right, bottom

VG_LITE_HIGH, // quality

VG_LITE_S8, // -128 to 127 coordinate range

{0}, // uploaded

sizeof(path_data), // path length

path_data, // path data

1 // path changed

};

error = vg_lite_draw(fb, &path, VG_LITE_FILL_EVEN_ODD, &highlight_matrix,
VG_LITE_BLEND_SRC_OVER, 0x22444488);
```

After the vg_lite_draw call, vg_lite_clear_path(&path) is called to free and reset the path data.

11.4 vg_lite_draw_gradient Example

The following section demonstrates the usage of the vg_lite_draw_gradient API. It defines 5 colors (black, red, green, blue, white) in ramps[] and 5 stops in stops[] which are used for gradient color transition. It calls the following to setup the color gradient image.

```
uint32_t ramps[] = {0xff000000, 0xffff0000, 0xff00ff00, 0xff0000ff, 0xffffffff};

uint32_t stops[] = {0, 66, 122, 200, 255};

vg_lite_set_grad(&grad, 5, ramps, stops);

vg_lite_update_grad(&grad);
```

NOTE

The “colors” parameter (ramps[]) in `vg_lite_set_grad` API must be in ARGB8888 format with alpha at the higher byte. It also sets up the gradient transformation matrix “matGrad” with a proper scale factor and 30 degree rotation.

```
matGrad = vg_lite_get_grad_matrix(&grad);

vg_lite_identity(matGrad);

vg_lite_rotate(30.0f, matGrad);
```

Then it calls:

```
vg_lite_draw_gradient(fb, &path, VG_LITE_FILL_EVEN_ODD, &matPath, &grad, VG_LITE_BLEND_NONE);
```

with a polygon path and color gradient image/matrix so that it generates the rendering effect as illustrated in the image below.

After the draw gradient API, it calls the following to flush the VGLite commands and clean up the gradient image buffer.

```
vg_lite_finish();

vg_lite_clear_grad(&grad);
```

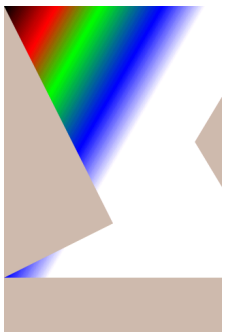


Figure 3. Example using vg_lite_draw_gradient

11.5 vg_lite_draw_pattern Example

This section demonstrates the usage of the vg_lite_draw_pattern API. It defines a vg_lite_path_t path for a convex polygon shape as shown below, and loads an image file "landscape.raw" with which to fill the polygon interior area.

It also defines two matrices, one named “matrix” for the image, another named “matPath” for the “path”. The image matrix rotates the image 33 degrees clockwise based on the image center.

```
vg_lite_identity(&matrix);

vg_lite_translate(fb_width / 2.0f, fb_height / 4.0f, &matrix);

vg_lite_rotate(33.0f, &matrix);
```

```
vg_lite_scale(0.4f, 0.4f, &matrix);

vg_lite_translate(fb_width / -2.0f, fb_height / -4.0f, &matrix);

vg_lite_identity(&matPath);

vg_lite_translate(fb_width / 2.0f, fb_height / 4.0f, &matPath);

vg_lite_scale(10, 10, &matPath);
```

Then it calls vg_lite_draw_pattern API two times with different parameters to draw the polygon twice.

```
error = vg_lite_draw_pattern(fb, &path, VG_LITE_FILL_EVEN_ODD, &matPath, &image, &matrix,
VG_LITE_BLEND_NONE,VG_LITE_PATTERN_COLOR, 0xffaabbcc, VG_LITE_FILTER_POINT);

error = vg_lite_draw_pattern(fb, &path, VG_LITE_FILL_EVEN_ODD, &matPath,&image, &matrix,
VG_LITE_BLEND_NONE,VG_LITE_PATTERN_PAD, 0xffaabbcc, VG_LITE_FILTER_POINT);
```

With the vg_lite_pattern_mode_t setting of VG_LITE_PATTERN_COLOR, the polygon area outside the pattern image of the upper polygon is filled with color 0xffaabbcc. With the vg_lite_pattern_mode_t setting of VG_LITE_PATTERN_PAD, the polygon area outside the pattern image of the lower polygon is filled with the border pixel color of the pattern image.

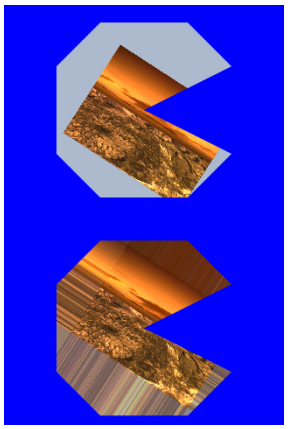


Figure 4. Example using vg_lite_draw_pattern

11.6 Vector-based Font Rendering Example

This section demonstrates vector-based font rendering with the `vg_lite_draw` API, which is capable of drawing quadratic curves and cubic curves based on end point and control point coordinates in the path data. The font path data can be generated by using a third-party font engine that can produce VGLite path data directly, or by using VeriSilicon's VGLite tools to convert other formats of font data, such as SVG, etc., to VGLite path data. Here is an example of path data for the character "~" (ASCII code 126):

```
float ascii_font_126[] =

{

2,15.984375,20.273438,

4,16.296875,20.476563,

6,15.781250,21.351563,14.921875,21.992188,

6,13.953125,22.710938,13.046875,22.710938,

6,12.375000,22.710938,10.898438,22.203125,

6,9.421875,21.695313,8.656250,21.695313,

6,7.937500,21.695313,7.375000,22.117188,

6,7.015625,22.382813,6.421875,23.117188,
```

```
4,6.109375,22.914063,

6,7.593750,20.664063,9.453125,20.664063,

6,10.156250,20.664063,11.492188,21.140625,

6,12.828125,21.617188,13.531250,21.617188,

6,14.921875,21.617188,15.984375,20.273438,

0

};
```

The first integer in each line is the path opcode, followed by the coordinates for each opcode. As listed in Section 8.4, opcode (2, x, y) moves the current position to (x, y); opcode (4, x, y) draws a line from the current position to (x, y); opcode (6, cx, cy, x, y) draws a quadratic curve from the current position to the given end point (x, y) using the specified control point (cx, cy).

The program calls:

```
error = vg_lite_init(256, 256);
```

to initialize VGLite with a 256x256 path tessellation buffer, then allocates a 320x320 render buffer with the format `VG_LITE_RGBA8888.The size of the` tessellation buffer is big enough to cover the font character bounding box.

The program renders the path for each character in the string `"Hello,\nVerisilicon!"`in a loop with calls to:

```
/* Draw the path using the matrix.*/

error = vg_lite_draw(fb, &path, VG_LITE_FILL_EVEN_ODD, &matrix, VG_LITE_BLEND_NONE, 0xFF0000FF);
```

The character's vector path is rendered without blending (VG_LITE_BLEND_NONE). The path interior is filled with the color red (0xFF0000FF).

Figure 5. Example using Vector Based Font Rendering

To demonstrate the smooth curve of vector-based path rendering with any scale factor, the program renders a single character "H" with a scaled size of 8X using following API calls.

```
vg_lite_identity(&matrix);

vg_lite_translate(startX, startY, &matrix);

vg_lite_scale(8.0, 8.0, &matrix);

error = vg_lite_draw(fb, &path, VG_LITE_FILL_EVEN_ODD, &matrix, VG_LITE_BLEND_NONE, 0xFF0000FF);
```

The following image example shows the resulting vector path rendering of character "H".

Figure 6. Example with Vector Based Font Rendering Upscaled 8X.

Chapter 12
Revision history

This table summarizes the revisions of this document.

Table 41. Revision history

Revision number	Date	Substantive changes
0	22 Feb 2021	Initial Draft

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 February 2021

Document identifier: IMXRTVGLITEAPIRM