

# IMXRTVGLITEAPIRM

i.MX RT VGLite API Reference Manual

Rev. 1.1 — 22 September 2022

Reference manual

## Document information

Information	Content
Keywords	i.MX RT, Vivante VGLite graphics API
Abstract	The Vivante VGLite graphics API is a platform-independent API from VeriSilicon Microelectronics for rendering an interactive graphic user interface that may include menus, fonts, curves, and images. The API supports 2D vector-based and 2D raster-based operations. It is aimed to provide maximum 2D vector/raster rendering performance with minimum memory footprint. The VGLite API can be used as the interface for the 2D GPU driver in NXP i.MX RT platforms.



## 1 Introduction

---

The Vivante VGLite graphics API is a platform-independent API from VeriSilicon Microelectronics for rendering an interactive graphic user interface that may include menus, fonts, curves, and images. The API supports 2D vector-based and 2D raster-based operations. It is aimed to provide maximum 2D vector/raster rendering performance with minimum memory footprint. The VGLite API can be used as the interface for the 2D GPU driver in NXP i.MX RT platforms.

The following i.MX RT devices support the VGLite API:

- i.MX RT500
- i.MX RT1160
- i.MX RT1170

This document contains copyright material disclosed with permission from VeriSilicon Microelectronics.

## 2 Vivante VGLite Graphics API

---

The Vivante VGLite graphics API is used to control 2D GPU hardware in i.MX RT platforms. It provides accelerated vector and raster operations. The API supports the following major features:

- Porter-Duff blending
- Gradient controls
- Fast clear
- Arbitrary rotations
- Path filling rules
- Path painting
- Pattern path filling

### 2.1 API partitions

The Vivante VGLite graphics API is designed to allow 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`.

## 3 Common parameters and error values

---

This chapter provides an overview of the common parameter types and the enumeration used for error reporting.

### 3.1 Common parameter types

The Vivante VGLite graphics API uses a naming convention scheme wherein definitions are preceded by 'vg\_lite'.

The VGLite API currently uses the data types and structures listed in the table below.

Table 1. Common parameter types

Name	Typedef	Value
int32_t	int	A signed 32-bit integer
uint32_t	unsigned int	An 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 bits of the color value. It is followed by the green and blue channels. The alpha channel is in the upper 8 bits 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 Enumerations for error reporting

This section describes enumerations used for error reporting.

#### 3.2.1 vg\_lite\_error\_t enumeration

Most functions in the API return an error status via the `vg_lite_error_t` enumeration. The table below lists possible error values. The error codes are used in many functions, including initialization, flush, blit, draw, gradient, and pattern functions.

If an API function completes successfully with no errors, the returned status is `VG_LITE_SUCCESS`.

Table 2. `vg_lite_error_t` enumeration

Value	Description
<code>VG_LITE_GENERIC_IO</code>	Cannot communicate with the kernel driver
<code>VG_LITE_INVALID_ARGUMENT</code>	An invalid argument was specified
<code>VG_LITE_MULTI_THREAD_FAIL</code>	Multi-thread/tasks fail
<code>VG_LITE_NO_CONTEXT</code>	No context specified
<code>VG_LITE_NOT_SUPPORT</code>	Function call not supported
<code>VG_LITE_OUT_OF_MEMORY</code>	Out of memory (driver heap)
<code>VG_LITE_OUT_OF_RESOURCES</code>	Out of resources (OS heap)
<code>VG_LITE_SUCCESS</code>	Successful with no errors
<code>VG_LITE_TIMEOUT</code>	Timeout

Table 2. `vg_lite_error_t` enumeration...continued

Value	Description
<code>VG_LITE_ALREADY_EXISTS</code>	Object exists
<code>VG_LITE_NOT_ALIGNED</code>	Data alignment error

## 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 describes enumerations used 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

Value	Description
<code>gcFEATURE_BIT_VG_IM_INDEX_FORMAT</code>	Index format support
<code>gcFEATURE_BIT_VG_PE_PREMULTIPLY</code>	Premultiply alpha support for image
<code>gcFEATURE_BIT_VG_RADIAL_GRADIENT</code>	Radial gradient support
<code>gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT</code>	Support for extended linear color gradient capabilities
<code>gcFEATURE_BIT_VG_BORDER_CULLING</code>	Border culling support
<code>gcFEATURE_BIT_VG_COLOR_KEY</code>	Color keying support
<code>gcFEATURE_BIT_VG_DITHER</code>	GPU dithering support
<code>gcFEATURE_BIT_VG_RGBA2_FORMAT</code>	RGBA2222 format support
<code>gcFEATURE_BIT_VG_QUALITY_8X</code>	Vector path 8x anti-aliasing support ( <code>VG_LITE_UPPER</code> )

### 4.2 Structures for product and feature queries

This section describes structures used 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`.

Table 4. `vg_lite_info_t` structure

<code>vg_lite_info_t</code> member	Type	Description
<code>api_version</code>	<code>uint32_t</code>	VGLite API version
<code>header_version</code>	<code>uint32_t</code>	VGLite header version
<code>release_version</code>	<code>uint32_t</code>	VGLite driver release version

Table 4. `vg_lite_info_t` structure...continued

<code>vg_lite_info_t</code> member	Type	Description
reserved	uint32_t	Reserved for future use

### 4.3 Functions for product and feature queries

This section describes functions used for product and feature queries.

#### 4.3.1 `vg_lite_get_product_info` function

**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` function

**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` function

**Description:**

This function can be used to read a GPU AHB register value given the AHB byte address of a register. Refer to the appropriate Vivante GPU 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 to read
result	Value of the register, returned by the function

**Returns:**

VG\_LITE\_SUCCESS. The behavior is undefined if a register is outside the range of VGLite core accessible addresses.

**4.3.4 vg\_lite\_query\_feature function**

**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 to be queried, as detailed in enum <a href="#">vg_lite_feature_t</a>
---------	--

**Returns:**

The feature is either not supported (0) or supported (1).

**4.3.5 vg\_lite\_mem\_avail function**

**Description:**

This function queries the amount of available contiguous video memory.

**Syntax:**

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

**Parameters:**

size	Pointer to the variable where the function should return the amount of remaining contiguous video memory
------	--

**Returns:**

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

## 5 API control

Before calling any VGLite API function, each application task/thread must initialize its VGLite context by calling the `vg_lite_init()` function. This function fills a features table, resets the fast-clear buffer, resets the compositing target buffer, and allocates task-specific 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 supports one context per thread to issue commands to GPU hardware. Multiple contexts can be used simultaneously by different threads/tasks because each thread/task can initialize its own context using the `vg_lite_init()` API.

### 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` function

**Description:**

This function is optional. If used, call it after `vg_lite_init()` if you want to change the GPU command buffer size for the current context.

This function is useful for devices where memory is limited and is less than the default size. The VGLite command buffer size is set to 64 KB 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 to set for the current context. Default is 64 KB.
------	---

#### 5.1.2 `vg_lite_init` function

**Description:**

This function initializes the memory and data structures for VGLite draw/blit functions. It allocates memory for the command buffer and a tessellation buffer of the specified size. The tessellation buffer width and 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 and vector path rendering is disabled (that is, only image blitting is available in the current context).

If the current context is the first context to access the hardware, then the hardware is turned on and initialized. Multiple parallel contexts are not supported for the same task/thread; therefore, in the same thread, `vg_lite_init()` cannot be called multiple times without calling `vg_lite_close()` first.

**Syntax:**

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

**Parameters:**

tessellation_width	The width of tessellation window. The value should be a multiple of 16; minimum width is 16 pixels, maximum cannot be greater than the 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. The 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 function**

**Description:**

The `vg_lite_close()` function deallocates all the resources and frees up the entire memory that was initialized earlier by the `vg_lite_init()` function. If current context is the only active context, then the `vg_lite_close()` function also turns OFF the hardware automatically.

**Syntax:**

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

**5.1.4 vg\_lite\_finish function**

**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 function**

**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.

## 6 Pixel buffers

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

### 6.1 Pixel buffer alignment

To work correctly, VGLite hardware requires the *data address* and *stride* of a pixel buffer to be aligned according to its pixel format. This requirement applies to all image formats. The byte alignment requirement for a pixel depends on the specific pixel format. For more details, see [Table 7](#).

The pixel buffer start address alignment requirement also varies depending on whether the buffer layout format is tiled or linear ([vg\\_lite\\_buffer\\_layout\\_t](#) enum):

- If the buffer layout is tiled (4x4 tiled), then the start address and stride must be 64 bytes aligned
- If the buffer layout is linear, then the start address and stride must be aligned according to the format of the pixel buffer, as described in [Table 7](#)

### 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 is extended to 32 bits as follows:

- If color format is same for the source and destination formats but they differ in the number of bits per color channel, then the source channel is multiplied by  $(2^d - 1)/(2^s - 1)$  and is 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 for a buffer. It applies to both image and render target.

**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 5. `vg_lite_buffer_format_t` enumeration

Value	Description	Supported as source	Supported as destination	Alignment (bytes)										
VG_LITE_ABGR8888	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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>ABGR8888</b></td> <td>R</td> <td>G</td> <td>B</td> <td>A</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>ABGR8888</b>	R	G	B	A	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>ABGR8888</b>	R	G	B	A										
VG_LITE_ARGB8888	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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>ARGB8888</b></td> <td>B</td> <td>G</td> <td>R</td> <td>A</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>ARGB8888</b>	B	G	R	A	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>ARGB8888</b>	B	G	R	A										
VG_LITE_BGRA8888	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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>BGRA8888</b></td> <td>A</td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>BGRA8888</b>	A	R	G	B	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>BGRA8888</b>	A	R	G	B										
VG_LITE_RGBA8888	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 border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>RGBA8888</b></td> <td>A</td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>RGBA8888</b>	A	B	G	R	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>RGBA8888</b>	A	B	G	R										
VG_LITE_BGRX8888	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. <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>BGRX8888</b></td> <td>X</td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>BGRX8888</b>	X	R	G	B	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>BGRX8888</b>	X	R	G	B										

Table 5. `vg_lite_buffer_format_t` enumeration...continued

Value	Description	Supported as source	Supported as destination	Alignment (bytes)										
<code>VG_LITE_RGBX8888</code>	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 border="1" data-bbox="379 573 994 667"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>RGBX8888</b></td> <td>X</td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>RGBX8888</b>	X	B	G	R	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>RGBX8888</b>	X	B	G	R										
<code>VG_LITE_XBGR8888</code>	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 border="1" data-bbox="379 819 994 913"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>XBGR8888</b></td> <td>R</td> <td>G</td> <td>B</td> <td>X</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>XBGR8888</b>	R	G	B	X	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>XBGR8888</b>	R	G	B	X										
<code>VG_LITE_XRGB8888</code>	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 border="1" data-bbox="379 1066 994 1160"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>XRGB8888</b></td> <td>B</td> <td>G</td> <td>R</td> <td>X</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>XRGB8888</b>	B	G	R	X	Yes	Yes	64
	31:24	23:16	15:8	7:0										
<b>XRGB8888</b>	B	G	R	X										
<code>VG_LITE_ABGR1555</code>	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 border="1" data-bbox="379 1339 994 1433"> <tr> <td></td> <td>15:11</td> <td>10:6</td> <td>5:1</td> <td>0:0</td> </tr> <tr> <td><b>ABGR5551</b></td> <td>R</td> <td>G</td> <td>B</td> <td>A</td> </tr> </table>		15:11	10:6	5:1	0:0	<b>ABGR5551</b>	R	G	B	A	Yes	Yes	32
	15:11	10:6	5:1	0:0										
<b>ABGR5551</b>	R	G	B	A										
<code>VG_LITE_ARGB1555</code>	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 border="1" data-bbox="379 1612 994 1706"> <tr> <td></td> <td>15:11</td> <td>10:6</td> <td>5:1</td> <td>0:0</td> </tr> <tr> <td><b>ARGB5551</b></td> <td>B</td> <td>G</td> <td>R</td> <td>A</td> </tr> </table>		15:11	10:6	5:1	0:0	<b>ARGB5551</b>	B	G	R	A	Yes	Yes	32
	15:11	10:6	5:1	0:0										
<b>ARGB5551</b>	B	G	R	A										
<code>VG_LITE_BGRA5551</code>	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 border="1" data-bbox="379 1886 994 1980"> <tr> <td></td> <td>15:15</td> <td>14:10</td> <td>9:5</td> <td>4:0</td> </tr> <tr> <td><b>BGRA5551</b></td> <td>A</td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		15:15	14:10	9:5	4:0	<b>BGRA5551</b>	A	R	G	B	Yes	Yes	32
	15:15	14:10	9:5	4:0										
<b>BGRA5551</b>	A	R	G	B										

Table 5. `vg_lite_buffer_format_t` enumeration...continued

Value	Description	Supported as source	Supported as destination	Alignment (bytes)										
<code>VG_LITE_RGBA5551</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:15</td> <td>14:10</td> <td>9:5</td> <td>4:0</td> </tr> <tr> <td><b>RGBA5551</b></td> <td>A</td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		15:15	14:10	9:5	4:0	<b>RGBA5551</b>	A	B	G	R	Yes	Yes	32
	15:15	14:10	9:5	4:0										
<b>RGBA5551</b>	A	B	G	R										
<code>VG_LITE_BGR565</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:11</td> <td>10:5</td> <td>4:0</td> </tr> <tr> <td><b>BGR565</b></td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		15:11	10:5	4:0	<b>BGR565</b>	R	G	B	Yes	Yes	32		
	15:11	10:5	4:0											
<b>BGR565</b>	R	G	B											
<code>VG_LITE_RGB565</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:11</td> <td>10:5</td> <td>4:0</td> </tr> <tr> <td><b>RGB565</b></td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		15:11	10:5	4:0	<b>RGB565</b>	B	G	R	Yes	Yes	32		
	15:11	10:5	4:0											
<b>RGB565</b>	B	G	R											
<code>VG_LITE_ABGR4444</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:12</td> <td>11:8</td> <td>7:4</td> <td>3:0</td> </tr> <tr> <td><b>ABGR4444</b></td> <td>R</td> <td>G</td> <td>B</td> <td>A</td> </tr> </table>		15:12	11:8	7:4	3:0	<b>ABGR4444</b>	R	G	B	A	Yes	Yes	32
	15:12	11:8	7:4	3:0										
<b>ABGR4444</b>	R	G	B	A										
<code>VG_LITE_ARGB4444</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:12</td> <td>11:8</td> <td>7:4</td> <td>3:0</td> </tr> <tr> <td><b>ARGB4444</b></td> <td>B</td> <td>G</td> <td>R</td> <td>A</td> </tr> </table>		15:12	11:8	7:4	3:0	<b>ARGB4444</b>	B	G	R	A	Yes	Yes	32
	15:12	11:8	7:4	3:0										
<b>ARGB4444</b>	B	G	R	A										
<code>VG_LITE_BGRA4444</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:12</td> <td>11:8</td> <td>7:4</td> <td>3:0</td> </tr> <tr> <td><b>BGRA4444</b></td> <td>A</td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		15:12	11:8	7:4	3:0	<b>BGRA4444</b>	A	R	G	B	Yes	Yes	32
	15:12	11:8	7:4	3:0										
<b>BGRA4444</b>	A	R	G	B										

Table 5. `vg_lite_buffer_format_t` enumeration...continued

Value	Description	Supported as source	Supported as destination	Alignment (bytes)										
<code>VG_LITE_RGBA4444</code>	<p>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.</p> <table border="1"> <tr> <td></td> <td>15:12</td> <td>11:8</td> <td>7:4</td> <td>3:0</td> </tr> <tr> <td><b>RGBA4444</b></td> <td>A</td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		15:12	11:8	7:4	3:0	<b>RGBA4444</b>	A	B	G	R	Yes	Yes	32
	15:12	11:8	7:4	3:0										
<b>RGBA4444</b>	A	B	G	R										
<code>VG_LITE_ABGR2222</code>	<p>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.</p> <p><b>Note:</b> Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a>.</p> <table border="1"> <tr> <td></td> <td>7:6</td> <td>5:4</td> <td>3:2</td> <td>1:0</td> </tr> <tr> <td><b>ABGR2222</b></td> <td>R</td> <td>G</td> <td>B</td> <td>A</td> </tr> </table>		7:6	5:4	3:2	1:0	<b>ABGR2222</b>	R	G	B	A	Yes	Yes	16
	7:6	5:4	3:2	1:0										
<b>ABGR2222</b>	R	G	B	A										
<code>VG_LITE_ARGB2222</code>	<p>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.</p> <p><b>Note:</b> Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a>.</p> <table border="1"> <tr> <td></td> <td>7:6</td> <td>5:4</td> <td>3:2</td> <td>1:0</td> </tr> <tr> <td><b>ARGB2222</b></td> <td>B</td> <td>G</td> <td>R</td> <td>A</td> </tr> </table>		7:6	5:4	3:2	1:0	<b>ARGB2222</b>	B	G	R	A	Yes	Yes	16
	7:6	5:4	3:2	1:0										
<b>ARGB2222</b>	B	G	R	A										
<code>VG_LITE_BGRA2222</code>	<p>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.</p> <p><b>Note:</b> Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a>.</p> <table border="1"> <tr> <td></td> <td>7:6</td> <td>5:4</td> <td>3:2</td> <td>1:0</td> </tr> <tr> <td><b>BGRA2222</b></td> <td>A</td> <td>R</td> <td>G</td> <td>B</td> </tr> </table>		7:6	5:4	3:2	1:0	<b>BGRA2222</b>	A	R	G	B	Yes	Yes	16
	7:6	5:4	3:2	1:0										
<b>BGRA2222</b>	A	R	G	B										
<code>VG_LITE_RGBA2222</code>	<p>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.</p> <p><b>Note:</b> Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a>.</p> <table border="1"> <tr> <td></td> <td>7:6</td> <td>5:4</td> <td>3:2</td> <td>1:0</td> </tr> <tr> <td><b>RGBA2222</b></td> <td>A</td> <td>B</td> <td>G</td> <td>R</td> </tr> </table>		7:6	5:4	3:2	1:0	<b>RGBA2222</b>	A	B	G	R	Yes	Yes	16
	7:6	5:4	3:2	1:0										
<b>RGBA2222</b>	A	B	G	R										
<code>VG_LITE_L8</code>	8-bit luminance value. There is no alpha value.	Yes	Yes	16										

Table 5. `vg_lite_buffer_format_t` enumeration...continued

Value	Description	Supported as source	Supported as destination	Alignment (bytes)										
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). <table border="1" style="margin-left: 20px;"> <tr> <td></td> <td>31:24</td> <td>23:16</td> <td>15:8</td> <td>7:0</td> </tr> <tr> <td><b>YUYV</b></td> <td>V0</td> <td>Y1</td> <td>U0</td> <td>Y0</td> </tr> </table>		31:24	23:16	15:8	7:0	<b>YUYV</b>	V0	Y1	U0	Y0	Yes	No	32
	31:24	23:16	15:8	7:0										
<b>YUYV</b>	V0	Y1	U0	Y0										
VG_LITE_A4	4-bit alpha format. There are no RGB values. <table border="1" style="margin-left: 20px;"> <tr> <td></td> <td>3:0</td> <td></td> </tr> <tr> <td><b>A4</b></td> <td>A</td> <td></td> </tr> </table>		3:0		<b>A4</b>	A		Yes	No	8				
	3:0													
<b>A4</b>	A													
VG_LITE_A8	8-bit alpha format. There are no RGB values. <table border="1" style="margin-left: 20px;"> <tr> <td></td> <td>7:0</td> </tr> <tr> <td><b>A8</b></td> <td>A</td> </tr> </table>		7:0	<b>A8</b>	A	Yes	Yes	16						
	7:0													
<b>A8</b>	A													

Table 6. Formats using color lookup tables

Hardware-dependent formats for <code>vg_lite_buffer_format_t</code>	Description	Supported as source	Supported as destination	Alignment (bytes)
VG_LITE_INDEX_1	1-bit index format. Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a> .	Yes	No	8
VG_LITE_INDEX_2	2-bit index format. Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a> .	Yes	No	8
VG_LITE_INDEX_4	4-bit index format. Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a> .	Yes	No	8
VG_LITE_INDEX_8	8-bit index format. Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see <a href="#">Table 82</a> .	Yes	No	16

6.4.1.1 Alignment notes

**Source image alignment requirement**

The byte alignment requirement for a pixel depends on the specific pixel format. Both *buffer address* and *buffer stride* must be aligned.

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	8	Yes	
VG_LITE_INDEX2	2	8	Yes	
VG_LITE_INDEX4	4	8	Yes	
VG_LITE_INDEX8	8	16	Yes	
VG_LITE_A4	4	8	Yes	
VG_LITE_A8	8	16	Yes	Yes
VG_LITE_L8	8	16	Yes	Yes
VG_LITE_ARGB2222 group	8	16	Yes	Yes
VG_LITE_RGB565 group	16	32	Yes	Yes
VG_LITE_ARGB1555 group	16	32	Yes	Yes
VG_LITE_ARGB4444 group	16	32	Yes	Yes
VG_LITE_YUY2/UYYVY	16	32	Yes	
VG_LITE_ARGB8888/XRGB8888 group	32	64	Yes	Yes

**Destination alignment requirement:**

- For pixel engine (PE) destination, the alignment should be 64 bytes for all tiled (4x4) buffer layouts. The pixel engine has no additional alignment requirement for linear buffer layouts.
- The alignment requirements of backend modules, such as display controller (DC), may limit the destination alignment.

**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

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

Value	Description
VG_LITE_LINEAR	Linear (scanline) layout

Table 9. `vg_lite_buffer_layout_t` enumeration...continued

Value	Description
VG_LITE_TILED	Data is organized in 4x4 pixel tiles. <b>Note:</b> For this layout, the buffer start address and stride must be 64 bytes 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

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. <b>Note:</b> This mode is only valid when image mode ( <code>vg_lite_buffer_image_mode_t</code> ) is either <code>VG_LITE_NORMAL_IMAGE_MODE</code> or <code>VG_LITE_MULTIPLY_IMAGE_MODE</code> .

#### 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

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

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

<code>vg_lite_buffer_t</code> member	Type	Description
<code>width</code>	<code>int32_t</code>	Width of buffer in pixels
<code>height</code>	<code>int32_t</code>	Height of buffer in pixels
<code>stride</code>	<code>int32_t</code>	Stride in bytes
<code>tiled</code>	<a href="#">vg_lite_buffer_layout_t</a>	Linear or tiled format for buffer enum
<code>format</code>	<a href="#">vg_lite_buffer_format_t</a>	Color format enum
<code>handle</code>	<code>void *</code>	Memory handle
<code>memory</code>	<code>void *</code>	Pointer to the start address of the memory
<code>address</code>	<code>uint32_t</code>	GPU address
<code>yuv</code>	<a href="#">vg_lite_yuvinfo_t</a>	YUV format info struct
<code>image_mode</code>	<a href="#">vg_lite_buffer_image_mode_t</a>	Blit image mode enum
<code>transparency_mode</code>	<a href="#">vg_lite_buffer_transparency_mode_t</a>	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

<code>vg_lite_yuvinfo_t</code> member	Type	Description
<code>swizzle</code>	<a href="#">vg_lite_swizzle_t</a>	UV swizzle enum
<code>yuv2rgb</code>	<a href="#">vg_lite_yuv2rgb_t</a>	YUV conversion standard enum
<code>uv_planar</code>	<code>uint32_t</code>	UV (U) planar address for GPU, generated by driver
<code>v_planar</code>	<code>uint32_t</code>	V planar address for GPU, generated by driver
<code>alpha_planar</code>	<code>uint32_t</code>	Alpha planar address for GPU, generated by driver
<code>uv_stride</code>	<code>uint32_t</code>	UV (U) stride in bytes
<code>v_stride</code>	<code>uint32_t</code>	V planar stride in bytes
<code>alpha_stride</code>	<code>uint32_t</code>	Alpha stride in bytes
<code>uv_height</code>	<code>uint32_t</code>	UV (U) height in pixels
<code>v_height</code>	<code>uint32_t</code>	V stride in bytes
<code>uv_memory</code>	<code>void *</code>	Logical pointer to the UV (U) planar memory
<code>v_memory</code>	<code>void *</code>	Logical pointer to the V planar memory

Table 14. `vg_lite_yuvinfo_t` structure...continued

<code>vg_lite_yuvinfo_t</code> member	Type	Description
<code>uv_handle</code>	<code>void *</code>	Memory handle of the UV (U) planar, generated by driver
<code>v_handle</code>	<code>void *</code>	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` function

**Description:**

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

To allow the hardware to access some memory, such as a source image or target buffer, you must first allocate the memory. The supplied `vg_lite_buffer_t` structure must be initialized with the size (width and height) and format of the requested buffer. If the stride is set to zero, then this function fills it in. The only input parameter to this function is the pointer to the buffer structure. If the structure has all the information needed, then appropriate memory is allocated for the buffer.

This function calls the VGLite kernel to allocate the memory. The kernel fills in the memory handle, logical address, and hardware addresses in the `vg_lite_buffer_t` structure.

**Alignment note:**

Vivante GPUs have an alignment requirement of 64 bytes. However, to meet the alignment requirements of Vivante display controller, the VGLite driver sets the render target buffer alignment to 128 bytes. For source image buffer alignment requirement, see the alignment notes available in [Table 7](#).

**Syntax:**

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

**Parameters:**

<code>buffer</code>	Pointer to the buffer that holds the size and format of the buffer being allocated. Either the memory or address field must 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 contiguous buffer was 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` function

**Description:**

This function is used to deallocate the buffer that was previously allocated. It frees 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 calling the <code>vg_lite_allocate()</code> function.
--------	---

**6.6.3 vg\_lite\_buffer\_upload function**

**Description:**

The function uploads the pixel data to a GPU memory buffer object. The format of the data (pixel) to be uploaded must match the format defined for 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:** Only `data[0]` and `stride[0]` arguments are used as planar YUV formats are not supported.

**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 calling the <code>vg_lite_allocate()</code> function
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 function**

**Description:**

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

To use a frame buffer directly as a target buffer:

- Wrap a `vg_lite_buffer_t` structure around the buffer
- 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 calling the <code>vg_lite_allocate()</code> function
--------	--

**6.6.5 `vg_lite_unmap` function**

**Description:**

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

**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 calling the <code>vg_lite_map()</code> function
--------	---

**6.6.6 `vg_lite_set_CLUT` function**

**Description:**

This function sets a context state for indexed color images. After the context is set (not NULL), the color for an indexed image to be rendered is obtained from the color lookup table (CLUT) according to the pixel indexes of the image.

**Note:** *Not all VGLite-compatible i.MX RT platforms support this feature.*

**Syntax:**

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

**Parameters:**

count	Number of colors in the color lookup table: <ul style="list-style-type: none"> <li>• For INDEX_1, up to 2 colors in the table</li> <li>• For INDEX_2, up to 4 colors in the table</li> <li>• For INDEX_4, up to 16 colors in the table</li> <li>• For INDEX_8, up to 256 colors in the table</li> </ul>
colors	This pointer is directly programmed to the command buffer. It only takes effect after the command buffer is submitted. The color is in ARGB format with A located in the high bits. <p><b>Note:</b> <i>The VGLite driver does not validate the CLUT data.</i></p>

**Returns:**

VG\_LITE\_SUCCESS as no checking is done.

### 6.6.7 vg\_lite\_set\_dither

Description:

This function toggles GPU dithering on or off. Dithering is disabled by default. When dithering is enabled, the driver configures the default dither table on the GPU.

**Note:** Not all VGLite-compatible i.MX RT platforms support this feature. For more details, see [Table 41](#).

Syntax:

```
vg_lite_error_t vg_lite_set_dither (
    int enable
);
```

Parameters:

Table 15. Parameters:

Parameter	Description
enable	Zero turns off the dithering function (default). One (1) turns on the dithering function.

Returns:

- VG\_LITE\_SUCCESS if dithering was successfully enabled and the default dither table was successfully configured.
- VG\_LITE\_NOT\_SUPPORT if the platform does not support GPU dithering.
- VG\_LITE\_NO\_CONTEXT if no drawing context is available.
- VG\_LITE\_INVALID\_ARGUMENT if the command buffer offset is valid.
- VG\_LITE\_OUT\_OF\_RESOURCES if the command buffer size is too small to support dithering.

## 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 graphic transformation matrix control structures.

### 7.2.1 vg\_lite\_matrix\_t structure

This structure defines a 3x3 floating point 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 16. `vg_lite_matrix_t` structure

vg_lite_matrix_t member	Type	Description
<code>m[3][3]</code>	<code>vg_lite_float_t</code>	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 function

**Description:**

This function resets a `vg_lite_matrix_t` structure to the identity matrix.

**Syntax:**

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

**Parameters:**

<code>matrix</code>	Pointer to the <a href="#">vg_lite_matrix_t</a> structure that has to be set to the identity matrix
---------------------	---

### 7.3.2 vg\_lite\_rotate function

**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:**

<code>degrees</code>	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.
<code>matrix</code>	Pointer to the <a href="#">vg_lite_matrix_t</a> structure that has to be rotated

### 7.3.3 vg\_lite\_scale function

**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 factor
scale_y	Vertical scale factor
matrix	Pointer to the <a href="#">vg_lite_matrix_t</a> structure that has to be scaled

### 7.3.4 vg\_lite\_translate function

**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 coordinate to translate to
y	Y coordinate to translate to
matrix	Pointer to the <a href="#">vg_lite_matrix_t</a> structure to be translated

## 8 Blits for compositing and blending

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

Compositing rules describe how two images are combined to form a resulting image. Blending rules describe how the colors of the overlapping areas are combined. VGLite supports two blending operations and a subset of the Porter-Duff operations [PD84]. For platforms that do not support alpha premultiplication, the Porter-Duff operators assume that the pixels have the alpha associated (premultiplied). It means, pixels are premultiplied prior to the blending operation.

**Note:** Ensure 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 a foreground 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 foreground 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 17. 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 18. `vg_lite_blend_t` enumeration

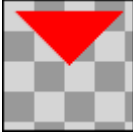
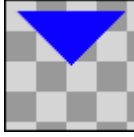


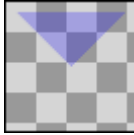

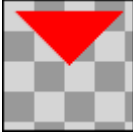
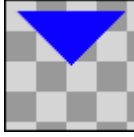


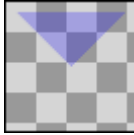

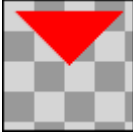
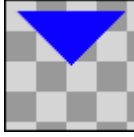


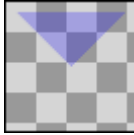


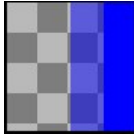
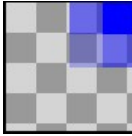
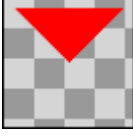
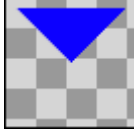
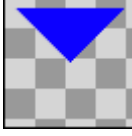
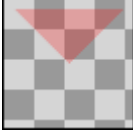
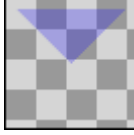


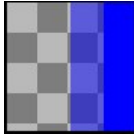
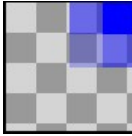
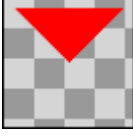
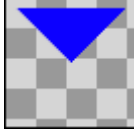
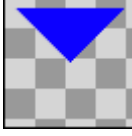
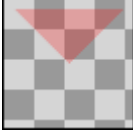
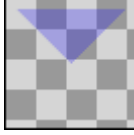


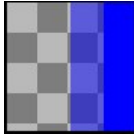
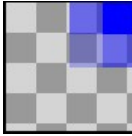
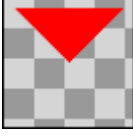
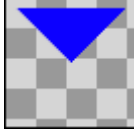
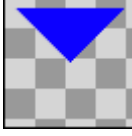
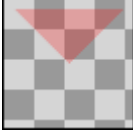
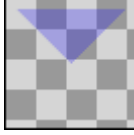

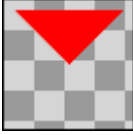
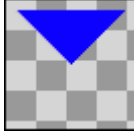
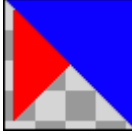

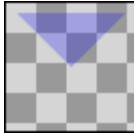

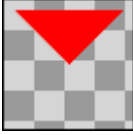
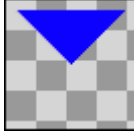
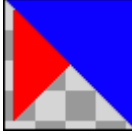

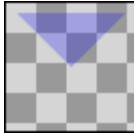

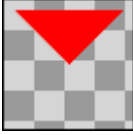
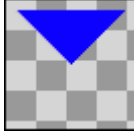
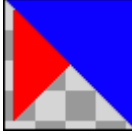

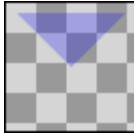

Value	Description																
<code>VG_LITE_BLEND_ADDITIVE</code>	<p><b>Table 19. Porter-Duff compositing mode: plus</b></p> <table border="1"> <thead> <tr> <th>S + D</th> <th></th> <th></th> <th>= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>Plus</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table>	S + D			= Result		Plus				50 %						
S + D			= Result														
	Plus																
	50 %																
<code>VG_LITE_BLEND_DST_IN</code>	<p><b>Table 20. Porter-Duff compositing mode: dst-in</b></p> <table border="1"> <thead> <tr> <th>Sa * D</th> <th></th> <th></th> <th>= Result</th> </tr> </thead> <tbody> <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> </tbody> </table>	Sa * D			= Result		DstIn				DstIn				50 %		
Sa * D			= Result														
	DstIn																
	DstIn																
	50 %																
<code>VG_LITE_BLEND_DST_OVER</code>	<p><b>Table 21. Porter-Duff compositing mode: dst-over</b></p> <table border="1"> <thead> <tr> <th>(1 - Da) * S + D</th> <th></th> <th></th> <th>= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>DstOver</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table>	(1 - Da) * S + D			= Result		DstOver				50 %						
(1 - Da) * S + D			= Result														
	DstOver																
	50 %																

Table 18. `vg_lite_blend_t` enumeration...continued





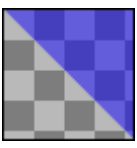





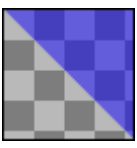





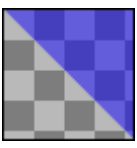


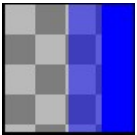
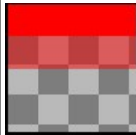

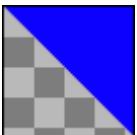
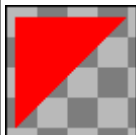
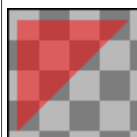
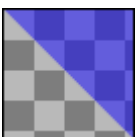
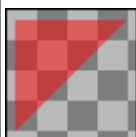

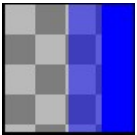
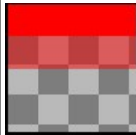

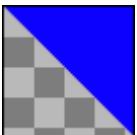
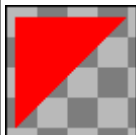
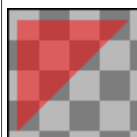
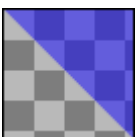
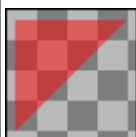

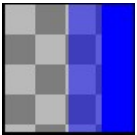
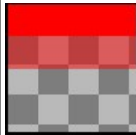

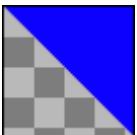
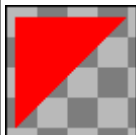
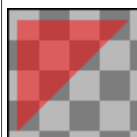
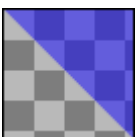
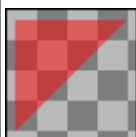
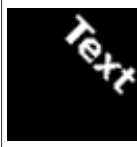



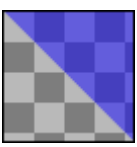

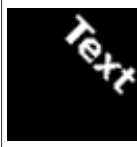



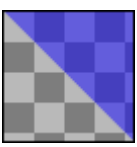

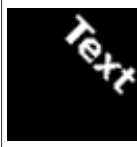



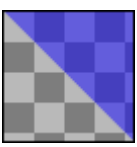

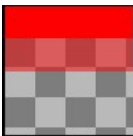
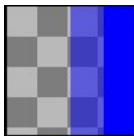
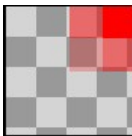
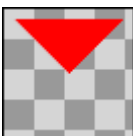


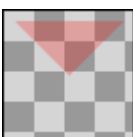


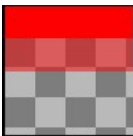
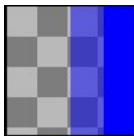
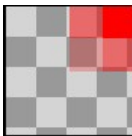
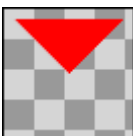


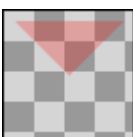


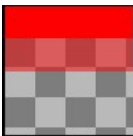
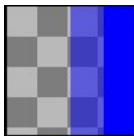
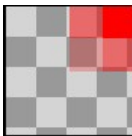
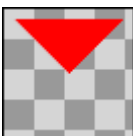


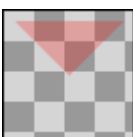




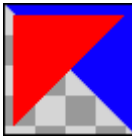
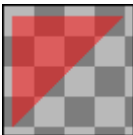
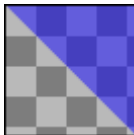
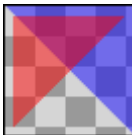


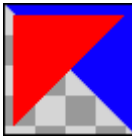
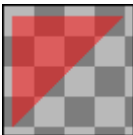
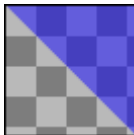
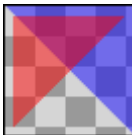


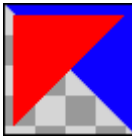
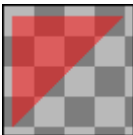
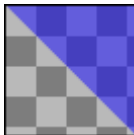
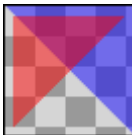
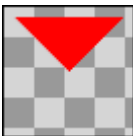


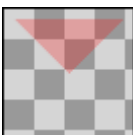
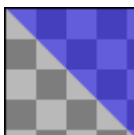
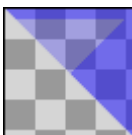
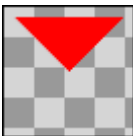


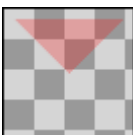
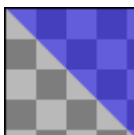
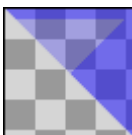
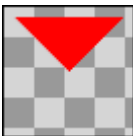


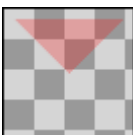
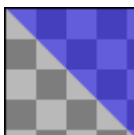
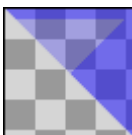
Value	Description																
<code>VG_LITE_BLEND_MULTIPLY</code>	<p><b>Table 22. Blending mode: mathematical multiply</b></p> <table border="1"> <thead> <tr> <th colspan="3"><math>S * (1 - D_a) + D * (1 - S_a) + S * D</math></th> <th>= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>Multiply</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table> <p>See <a href="https://www.w3.org/TR/compositing-1/#blendingmultiply">https://www.w3.org/TR/compositing-1/#blendingmultiply</a>) make white transparent for diagrams/text.</p>	$S * (1 - D_a) + D * (1 - S_a) + S * D$			= Result		Multiply				50 %						
$S * (1 - D_a) + D * (1 - S_a) + S * D$			= Result														
	Multiply																
	50 %																
<code>VG_LITE_BLEND_NONE</code>	<p><b>Table 23. Porter-Duff compositing mode: src</b></p> <table border="1"> <thead> <tr> <th>S</th> <th></th> <th></th> <th>= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>Src</td> <td></td> <td></td> </tr> <tr> <td></td> <td>Src</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table>	S			= Result		Src				Src				50 %		
S			= Result														
	Src																
	Src																
	50 %																
<code>VG_LITE_BLEND_SCREEN</code>	<p><b>Table 24. Blending Mode: mathematical screen</b></p> <table border="1"> <thead> <tr> <th colspan="3"><math>S + D - S * D</math></th> <th>= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>Screen</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table> <p>See <a href="https://www.w3.org/TR/compositing-1/#blendingscreen">https://www.w3.org/TR/compositing-1/#blendingscreen</a>) make black transparent for diagrams/text.</p>	$S + D - S * D$			= Result		Screen				50 %						
$S + D - S * D$			= Result														
	Screen																
	50 %																

Table 18. `vg_lite_blend_t` enumeration...continued

Value	Description																
VG_LITE_BLEND_SRC_IN	<p><b>Table 25. Porter-Duff compositing mode: src-in, also known as clipping</b></p> <table border="1"> <thead> <tr> <th colspan="2"><math>D \alpha * S</math></th> <th colspan="2">= Result</th> </tr> </thead> <tbody> <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> </tbody> </table>	$D \alpha * S$		= Result			SrcIn								50 %		
$D \alpha * S$		= Result															
	SrcIn																
																	
	50 %																
VG_LITE_BLEND_SRC_OVER	<p><b>Table 26. Porter-Duff compositing mode: src-over</b></p> <table border="1"> <thead> <tr> <th colspan="2"><math>S + (1 - S \alpha) * D</math></th> <th colspan="2">= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>SrcOver</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table>	$S + (1 - S \alpha) * D$		= Result			SrcOver				50 %						
$S + (1 - S \alpha) * D$		= Result															
	SrcOver																
	50 %																
VG_LITE_BLEND_SUBTRACT	<p><b>Table 27. Porter-Duff compositing mode: dst-out</b></p> <table border="1"> <thead> <tr> <th colspan="2"><math>D * (1 - S \alpha)</math></th> <th colspan="2">= Result</th> </tr> </thead> <tbody> <tr> <td></td> <td>DestOut</td> <td></td> <td></td> </tr> <tr> <td></td> <td>50 %</td> <td></td> <td></td> </tr> </tbody> </table>	$D * (1 - S \alpha)$		= Result			DestOut				50 %						
$D * (1 - S \alpha)$		= Result															
	DestOut																
	50 %																

### 8.1.2 `vg_lite_color_t` parameter

The common parameter `vg_lite_color_t` is described in [Table 1](#).

### 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 28. `vg_lite_filter_t` enumeration

Value	Description
<code>VG_LITE_FILTER_POINT</code>	Fetch only the nearest image pixel
<code>VG_LITE_FILTER_LINEAR</code>	Use linear interpolation along horizontal line
<code>VG_LITE_FILTER_BI_LINEAR</code>	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 the "Pixel buffer structures" section (see [Section 6.5.1](#)).

### 8.2.2 `vg_lite_color_key_t` structure

A "color key" contains R,G,B channels which are noted as `high_rgb` and `low_rgb` respectively.

When the `enable` attribute is `true`, the specified color key is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are in range [`low_rgb`, `high_rgb`]. The color keying should be disabled by calling the `vg_lite_set_color_key` API again when no longer required.

Used in structure: `vg_lite_color_key4_t`

Table 29. `vg_lite_color_key_t` structure

<code>vg_lite_color_key_t</code> members	Type	Description
<code>enable</code>	<code>uint8_t</code>	When set (true), this color key is enabled
<code>low_r</code>	<code>uint8_t</code>	The R channel of <code>low_rgb</code>
<code>low_g</code>	<code>uint8_t</code>	The G channel of <code>low_rgb</code>
<code>low_b</code>	<code>uint8_t</code>	The B channel of <code>low_rgb</code>
<code>alpha</code>	<code>uint8_t</code>	The alpha value to replace the destination pixel alpha channel value with
<code>high_r</code>	<code>uint8_t</code>	The R channel of <code>high_rgb</code>
<code>high_g</code>	<code>uint8_t</code>	The G channel of <code>high_rgb</code>

Table 29. `vg_lite_color_key_t` structure ...continued

<code>vg_lite_color_key_t</code> members	Type	Description
<code>high_b</code>	<code>uint8_t</code>	The B channel of <code>high_rgb</code>

### 8.2.3 `vg_lite_color_key4_t` structure

An array of 4 color keying parameters. The priority order is `color_key_0 > color_key_1 > color_key_2 > color_key_3`.

Used in function: `vg_lite_set_color_key`

Table 30. `vg_lite_color_key4_t` structure

<code>vg_lite_color_key4_t</code> members	Type	Description
<code>color_key_0</code>	<code>vg_lite_color_key_t</code>	Parameters for color key #0
<code>color_key_1</code>	<code>vg_lite_color_key_t</code>	Parameters for color key #1
<code>color_key_2</code>	<code>vg_lite_color_key_t</code>	Parameters for color key #2
<code>color_key_3</code>	<code>vg_lite_color_key_t</code>	Parameters for color key #3

### 8.2.4 `vg_lite_matrix_t` structure

Defined under the "Matrix control structures" section (see [Section 7.2.1](#)).

### 8.2.5 `vg_lite_path_t` structure

Defined under the "Vector path structures" section (see [Section 9.2.2](#)).

### 8.2.6 `vg_lite_rectangle_t` structure

This structure defines a rectangle by using coordinates.

Used in blit function: `vg_lite_clear`.

Table 31. `vg_lite_rectangle_t` structure

<code>vg_lite_rectangle_t</code> member	Type	Description
<code>x</code>	<code>int32_t</code>	X origin of rectangle, left coordinate in pixels
<code>y</code>	<code>int32_t</code>	Y origin of rectangle, top coordinate in pixels
<code>width</code>	<code>int32_t</code>	Width of rectangle in pixels
<code>height</code>	<code>int32_t</code>	Height of rectangle in pixels

### 8.2.7 `vg_lite_point_t` structure

This structure defines a 2D point.

Used in structure: `vg_lite_point4_t`.

Table 32. `vg_lite_point_t` structure

<code>vg_lite_point_t</code> member	Type	Description
<code>X</code>	<code>int32_t</code>	X value of coordinate
<code>Y</code>	<code>int32_t</code>	Y value of coordinate

### 8.2.8 vg\_lite\_point4\_t structure

This structure defines four 2D points that form a quadrilateral. The points are defined using the `vg_lite_point_t` data structure.

Used in blit function: `vg_lite_get_transform_matrix`.

Table 33. `vg_lite_point4_t` structure

vg_lite_point4_t member	Type	Description
<code>vg_lite_point_t[4]</code>	<code>int32_t</code>	A set of four points designating the vertices of the quadrilateral

## 8.3 Blit functions

This section provides an overview on blit functions.

### 8.3.1 vg\_lite\_blit function

**Description:**

This function performs the blit operation using a source buffer and 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. The `vg_lite_blit()` function does not support coverage sample anti-aliasing; therefore, the destination buffer edge may not be smooth especially with a rotation matrix. VGLite vector 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 a foreground 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.
- The blit function has a hardware limitation on the i.MX RT500 platform. Because of the limited capabilities of the math unit in the GPU, the output image quality may be degraded when blitting images larger than 256x256 pixels. When required to blit larger images on this platform, it is recommended to split the images in multiple tiles which are less than the mentioned threshold size and to blit them individually, reassembling the original image on the target buffer.

**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 <a href="#">vg_lite_buffer_t</a> structure which defines the destination buffer. See <a href="#">Image Source Alignment</a> Requirement for valid destination color formats for the blit functions.
source	Points to the <a href="#">vg_lite_buffer_t</a> structure for the source buffer. All color formats available in the <a href="#">vg_lite_buffer_format_t</a> enum are valid source formats for the blit function.
matrix	Points to a <a href="#">vg_lite_matrix_t</a> structure that defines the transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
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). <b>Note:</b> If the <i>matrix</i> parameter is specified with rotation or perspective, and the <i>blend</i> parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN; then, the VGLite driver overwrites the application setting for the blit operation as follows: <ul style="list-style-type: none"> <li>• If gcFEATURE_BIT_VG_BORDER_CULLING (<a href="#">vg_lite_feature_t</a>) is supported, then Transparency mode is always set to TRANSPARENT</li> <li>• If gcFEATURE_BIT_VG_BORDER_CULLING (<a href="#">vg_lite_feature_t</a>) is not supported, then Blend mode is always set to VG_LITE_BLEND_SRC_OVER</li> </ul> It happens 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 do not need a mix color, then set the color parameter to 0.
filter	Specifies the filter type. All formats available in the <a href="#">vg_lite_filter_t</a> enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

**8.3.2 vg\_lite\_blit\_rect function**

**Description:**

This function performs a blit rectangle operation using a source buffer and 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. The `vg_lite_blit_rect()` function does not support coverage sample anti-aliasing; therefore, the destination buffer edge may not be smooth especially with a rotation matrix. VGLite vector 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 a foreground 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.
- The blit function has a hardware limitation on the i.MX RT500 platform. Because of the limited capabilities of the math unit in the GPU, the output image quality may be

degraded when blitting images larger than 256x256 pixels. When required to blit larger images on this platform it is recommended to split the images in multiple tiles which are less than the mentioned threshold size and to blit them individually, reassembling the original image on the target buffer.

**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 <a href="#">vg_lite_buffer_t</a> structure which defines the destination buffer. See <a href="#">Source Image Alignment</a> Requirement for valid destination color formats for the <a href="#">blit_rect</a> functions.
source	Points to the <a href="#">vg_lite_buffer_t</a> structure for the source buffer. All color formats available in the <a href="#">vg_lite_buffer_format_t</a> enum are valid source formats for the <a href="#">blit_rect</a> function.
rect	Specifies the rectangle area of the source image to blit. <code>rect[0]/[1]/[2]/[3]</code> are x, y, width, and height of the source rectangle respectively.
matrix	Points to a <a href="#">vg_lite_matrix_t</a> structure that defines the 3x3 transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
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 <code>VG_LITE_BLEND_NONE</code> (0). <b>Note:</b> If the <i>matrix</i> parameter is specified with rotation or perspective, and the <i>blend</i> parameter is specified as <code>VG_LITE_BLEND_NONE</code> , <code>VG_LITE_BLEND_SRC_IN</code> , or <code>VG_LITE_BLEND_DST_IN</code> ; then, the VGLite driver overwrites the application setting for the blit operation as follows: <ul style="list-style-type: none"> <li>• If <code>gcFEATURE_BIT_VG_BORDER_CULLING</code> (<a href="#">vg_lite_feature_t</a>) is supported, then Transparency mode is always set to <code>TRANSPARENT</code></li> <li>• If <code>gcFEATURE_BIT_VG_BORDER_CULLING</code> (<a href="#">vg_lite_feature_t</a>) is not supported, then Blend mode is always set to <code>VG_LITE_BLEND_SRC_OVER</code>.</li> </ul> It happens 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 do not need a mix color, then set the color parameter to 0.
filter	Specifies the filter type. All formats available in the <a href="#">vg_lite_filter_t</a> enum are valid formats for this function. A value of zero (0) indicates <code>VG_LITE_FILTER_POINT</code> .

**8.3.3 vg\_lite\_get\_transform\_matrix function**

**Description:**

This function calculates a 3x3 homogenous transform matrix for `vg_lite_blit` and `vg_lite_blit_rect` based on source polygon coordinates and target polygon



coordinates. It takes a source quadrilateral and destination quadrilateral as inputs and calculates a transformation matrix that can be used to transform the source quadrilateral into the destination quadrilateral. The function is intended to support image perspective transformations.

**Syntax:**

```
vg_lite_error_t vg_lite_get_transform_matrix (
    vg_lite_point4_t src,
    vg_lite_point4_t dst,
    vg_lite_matrix_t *mat
);
```

**Parameters:**

src	Pointer to the four 2D points that form a source polygon
dst	Pointer to the four 2D points that form a destination polygon
mat	Pointer to 3*3 homogenous matrix that transforms source polygon to destination polygon. The matrix can be used as input parameter for <code>vg_lite_blit</code> and <code>vg_lite_blit_rect</code> .

**Returns:**

Returns the status as defined by `vg_lite_error_t`.

**8.3.4 `vg_lite_clear` function**

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

**8.3.5 `vg_lite_set_color_key` function**

**Description:**

This function enables color keying. Color keying can be used for blit or for draw pattern operations.

A “color key” contains R, G, B channels which are noted as `high_rgb` and `low_rgb` respectively. When the `vg_lite_color_key_t` structure `enable` attribute is set to `true`, the color key is effective and the specified alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are within range [`low_rgb`, `high_rgb`]. The color keying should be disabled when no longer needed.

**Note:** *Not all VGLite-compatible i.MX RT platforms support color keying. For more details, see [Table 41](#).*

**Syntax:**

```
vg_lite_error_t vg_lite_set_color_key (
    vg_lite_color_key4_t colorkey,
);
```

**Parameters:**

Table 34. `vg_lite_set_color_key` function

Parameter	Description
<code>colorkey</code>	Color keying parameters as defined by <code>vg_lite_color_key4_t</code> . here are 4 groups of color key states: <ul style="list-style-type: none"> <li>• <code>color_key_0</code>, <code>high_rgb_0</code>, <code>low_rgb_0</code>, <code>alpha_0</code>, <code>enable_0</code></li> <li>• <code>color_key_1</code>, <code>high_rgb_1</code>, <code>low_rgb_1</code>, <code>alpha_1</code>, <code>enable_1</code></li> <li>• <code>color_key_2</code>, <code>high_rgb_2</code>, <code>low_rgb_2</code>, <code>alpha_2</code>, <code>enable_2</code></li> <li>• <code>color_key_3</code>, <code>high_rgb_3</code>, <code>low_rgb_3</code>, <code>alpha_3</code>, <code>enable_3</code></li> </ul> The priority order of these states is: <code>color_key_0 &gt; color_key_1 &gt; color_key_2 &gt; color_key_3</code> .

**Returns:**

`VG_LITE_SUCCESS` if successful. Otherwise `VG_LITE_NOT_SUPPORT` if color keying is not supported by hardware.

## 8.4 Premultiply and scissor functions

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

### 8.4.1 `vg_lite_enable_premultiply` function

**Description:**

This function enables alpha premultiplication in hardware and returns a status error code. Not all VGLite-compatible i.MX RT platforms support alpha premultiplication.

**Note:** *Not all VGLite-compatible i.MX RT platforms support color premultiplication. For more details, see [Table 82](#).*

**Syntax:**

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

#### 8.4.2 vg\_lite\_disable\_premultiply function

**Description:**

This function disables alpha premultiplication and returns a status error code. Not all VGLite-compatible i.MX RT platforms support alpha premultiplication.

**Note:** *Not all VGLite-compatible i.MX RT platforms support color premultiplication. For more details, see [Table 82](#).*

**Syntax:**

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

#### 8.4.3 vg\_lite\_enable\_scissor function

**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 function

**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 function

**Description:**

This function is used to configure a rectangular scissoring area 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:**

Table 35. vg\_lite\_set\_scissor function

Parameter	Description
x	X coordinate of the scissoring window origin
Y	Y coordinate of the scissoring window origin

Table 35. `vg_lite_set_scissor` function...continued

Parameter	Description
<code>width</code>	Width of the scissoring window in pixels
<code>height</code>	Height of the scissoring window in pixels

## 9 Vector path control

This chapter 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` enum are defined in [Table 1](#).

#### 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`, `vg_lite_init_arc_path`.

Table 36. `vg_lite_quality_t` enumeration

Value	Description
<code>VG_LITE_HIGH</code>	High quality: 16x coverage sample anti-aliasing
<code>VG_LITE_UPPER</code>	Upper quality: 8x coverage sample anti-aliasing. Not all VGLite-compatible i.MX RT platforms support this setting. For more details, see <a href="#">Table 82</a> .
<code>VG_LITE_MEDIUM</code>	Medium quality: 4x coverage sample anti-aliasing
<code>VG_LITE_LOW</code>	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 gets the memory allocation information recorded by kernel.

Used in structure: `vg_lite_path_t`.

Table 37. `vg_lite_hw_memory` structure

<code>vg_lite_hw_memory_t</code> member	Type	Description
<code>handle</code>	<code>void *</code>	GPU memory object handle
<code>memory</code>	<code>void *</code>	Logical memory address
<code>address</code>	<code>uint32_t</code>	GPU memory address

Table 37. `vg_lite_hw_memory` structure...continued

vg_lite_hw_memory_t member	Type	Description
bytes	uint32_t	Size of memory
property	uint32_t	Bit 0 is used for path upload: <ul style="list-style-type: none"> <li>• 0: Disable path data uploading (always embedded into command buffer)</li> <li>• 1: Enable auto path data uploading</li> </ul>

### 9.2.2 `vg_lite_path_t` structure

This structure describes VGLite path data.

Path data is made of op codes and coordinates. The format for opcodes is always `VG_LITE_S8`. For more details on opcodes, see [Section 9.4](#).

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 38. `vg_lite_path_t` structure

vg_lite_path_t member	Type	Description
bounding_box[4]	vg_lite_float_t	bounding box for path [0] left [1] top [2] right [3] bottom
quality	<a href="#">vg_lite_quality_t</a>	enum for quality hint for the path, anti-aliasing level
format	<a href="#">vg_lite_format_t</a>	enum for coordinate format
uploaded	<a href="#">vg_lite_hw_memory_t</a>	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 39. Coordinate format

If <code>vg_lite_format_t</code> is:	Path data alignment in array should be:
<code>VG_LITE_S8</code>	8 bits
<code>VG_LITE_S16</code>	2 bytes
<code>VG_LITE_S32</code>	4 bytes

#### Special notes for path objects:

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

For example, because opcodes are 8-bit (1-byte), 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>
<align_to_data_size>
<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>
<pad to 4 bytes>
<opcode3_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode3>
...
```

### 9.3 Vector path functions

If a small tessellation window is used, then depending on the size of the path, a path might be uploaded to the hardware multiple times because the hardware scanline converts the path according to the specified tessellation window size. This may result in reduced VGLite path-rendering performance. Therefore, the tessellation buffer size must be set to the most common path size. For example, if you only render 24-point fonts, then you can set the tessellation buffer to 24x24.

All the 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 function

**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:**

Table 40. `vg_lite_path_calc_length` function

Parameter	Description
<code>cmd</code>	Pointer to the opcode array to use to construct the path
<code>count</code>	The opcode count
<code>format</code>	The coordinate data format. All formats available in the <code>vg_lite_format_t</code> enum are valid formats for this function.

### 9.3.2 `vg_lite_path_append` function

**Description:**

This function assembles the command buffer for the path. It prepares the final GPU command buffer for the path based on the input opcodes (`cmd`) and coordinates (`data`). The application allocates the command buffer and assigns it to the path. 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:**

Table 41. `vg_lite_path_append` function

Parameter	Description
<code>path</code>	Pointer to the path definition
<code>cmd</code>	Pointer to the opcode array to use to construct the path
<code>data</code>	Pointer to the coordinate data array to use to construct the path
<code>seg_count</code>	The opcode count

### 9.3.3 `vg_lite_init_path` function

**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:**

**Table 42. vg\_lite\_init\_path function**

Parameter	Description
path	Pointer to the <a href="#">vg_lite_path_t</a> structure for the path object to be initialized with the member values specified.
data_format	The coordinate data format. All formats available in the <a href="#">vg_lite_format_t</a> enum are valid formats for this function.
quality	The quality for the path object. All formats available in the <a href="#">vg_lite_quality_t</a> 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\_init\_arc\_path function**

**Description:**

This function initializes an arc path definition with specified vectors.

**Syntax:**

```

vg_lite_error_t vg_lite_init_arc_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:**

**Table 43. vg\_lite\_init\_arc\_path function**

Parameter	Function
path	Pointer to the <a href="#">vg_lite_path_t</a> structure for the path object to be initialized with the member values specified.



Table 43. `vg_lite_init_arc_path` function ...continued

Parameter	Function
<code>data_format</code>	The coordinate data format. All formats available in the <a href="#">vg_lite_format_t</a> enum are valid formats for this function.
<code>quality</code>	The quality for the path object. All formats available in the <a href="#">vg_lite_quality_t</a> enum are valid formats for this function.
<code>path_length</code>	The length of the path data (in bytes)
<code>path_data</code>	Pointer to path data
<code>min_x</code> <code>min_y</code> <code>max_x</code> <code>max_y</code>	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.

**9.3.5 `vg_lite_upload_path` function**

**Description:**

This function is used to upload a path to GPU memory. Usually, the VGLite driver copies any path data into a command buffer structure during runtime. This process takes more time if several paths are to be rendered. In an embedded system, the path data does not change; therefore, the path data must be uploaded into GPU memory in such a form that the GPU can access it directly. This function signals the driver to allocate a buffer containing the path data and the required command buffer header and footer data for the GPU to access the data directly. Call the `vg_lite_clear_path` function to free this buffer after the path is used.

**Syntax:**

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

**Parameters:**

Table 44. `vg_lite_upload_path` function Description:

Parameter	Description
<code>path</code>	Pointer to a <a href="#">vg_lite_path_t</a> structure that contains the path to be uploaded

**Returns:**

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

**9.3.6 `vg_lite_clear_path` function**

**Description:**

This function clears and resets 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:**

**Table 45. vg\_lite\_clear\_path function**

Parameter	Description
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 46. 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. <i>start<sub>x</sub> = x</i> <i>start<sub>y</sub> = y</i>
0x03	(Δx, Δy)	MOVE_REL. Move to the given relative point. Close any open path. <i>start<sub>x</sub> = start<sub>x</sub> + Δx</i> <i>start<sub>y</sub> = start<sub>y</sub> + Δy</i>
0x04	(x, y)	LINE. Draw a line to the given point: <i>Line(start<sub>x</sub>, start<sub>y</sub>, x, y)</i> <i>start<sub>x</sub> = x</i> <i>start<sub>y</sub> = y</i>
0x05	(Δx, Δy)	LINE_REL. Draw a line to the given relative point: <i>x = start<sub>x</sub> + Δx</i> <i>y = start<sub>y</sub> + Δy</i> <i>Line(start<sub>x</sub>, start<sub>y</sub>, x, y)</i> <i>start<sub>x</sub> = x</i> <i>start<sub>y</sub> = y</i>

Table 46. Vector path data opcodes...continued

Opcode	Arguments	Description
0x06	(cx, cy) (x, y)	<p>QUAD. Draw a quadratic curve to the given endpoint using the specified control point:</p> $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x07	(Δcx, Δcy) (Δx, Δy)	<p>QUAD_REL. Draw a quadratic curve to the given relative endpoint using the specified relative control point:</p> $cx = start_x + \Delta cx$ $cy = start_y + \Delta cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x08	(cx <sub>-1</sub> , cy <sub>1</sub> ) (cx <sub>2</sub> , cy <sub>2</sub> ) (x, y)	<p>CUBIC. Draw a cubic curve to the given endpoint using the specified control points:</p> $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$
0x09	(Δcx <sub>-1</sub> , Δcy <sub>1</sub> ) (Δcx <sub>2</sub> , Δcy <sub>2</sub> ) (Δx, Δy)	<p>CUBIC_REL. Draw a cubic curve to the given relative endpoint using the specified relative control points:</p> $cx_1 = start_x + \Delta cx_1$ $cy_1 = start_y + \Delta cy_1$ $cx_2 = start_x + \Delta cx_2$ $cy_2 = start_y + \Delta cy_2$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$

The table below shows the opcodes available for arc paths. Here, CW and CCW stand for "clockwise" and "counter-clockwise", respectively.

Table 47. Vector path data opcodes for arc paths

Opcode for arc paths	Arguments	Description
0x0A	(rh, rv, rot, x, y)	SCCWARC. Draw a small CCW arc to the given endpoint using the specified radius and rotation angle: $SCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x0B	(rh, rv, rot, x, y)	SCCWARC_REL. Draw a small CCW arc to the given relative endpoint using the specified radius and rotation angle: $x = start_x + \Delta x$ $y = start_y + \Delta y$ $SCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x0C	(rh, rv, rot, x, y)	SCWARC. Draw a small CW arc to the given endpoint using the specified radius and rotation angle: $SCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x0D	(rh, rv, rot, x, y)	SCWARC_REL. Draw a small CW arc to the given relative endpoint using the specified radius and rotation angle: $x = start_x + \Delta x$ $y = start_y + \Delta y$ $SCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x0E	(rh, rv, rot, x, y)	LCCWARC. Draw a large CCW arc to the given endpoint using the specified radius and rotation angle: $LCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x0F	(rh, rv, rot, x, y)	LCCWARC_REL. Draw a large CCW arc to the given relative endpoint using the specified radius and rotation angle: $x = start_x + \Delta x$ $y = start_y + \Delta y$ $LCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x10	(rh, rv, rot, x, y)	LCWARC. Draw a large CW arc to the given endpoint using the specified radius and rotation angle: $LCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$

Table 47. Vector path data opcodes for arc paths...continued

Opcode for arc paths	Arguments	Description
0x11	(rh, rv, rot, x, y)	<p>LCWARC_REL. Draw a large CW arc to the given relative endpoint using the specified radius and rotation angle:</p> $x = start_x + \Delta x$ $y = start_y + \Delta y$ <p><i>LCWARC(rh, rv, rot, x, y)</i></p> $start_x = x$ $start_y = y$

## 10 Vector-dased 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 defined under the "Blit enumerations" section (see [Section 8.1.1](#)).

#### 10.1.2 vg\_lite\_color\_t parameter

The common parameter `vg_lite_color_t` is described in [Section 3.1](#).

#### 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 toward infinity, and subtract 1 if the edge crossed from right to left. In this way, each region of the plane receives 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.

The fill algorithm is limited to 256 intersection points when `VG_LITE_LOW` or `VG_LITE_MEDIUM` quality is selected for the vector path, 64 crossing points for `VG_LITE_UPPER` and only 3 for `VG_LITE_HIGH`. If the polygon to render has a complex shape (many vertices and/or many self-intersecting edges) it is recommended to use a lower rendering quality (such as `VG_LITE_MEDIUM`) in order not to overflow the crossing points buffer which, in turn, could degrade rendering quality.

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

Table 48. `vg_lite_fill_t` enumeration

Value	Description
<code>VG_LITE_FILL_NON_ZERO</code>	Non-zero fill rule. A pixel is drawn if it crosses at least one path pixel.
<code>VG_LITE_FILL_EVEN_ODD</code>	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**

This enum is defined under the "Blit enumerations" section (see [Section 8.1.3](#)).

**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 49. `vg_lite_pattern_mode_t` enumeration

Value	Description
<code>VG_LITE_PATTERN_COLOR</code>	Fill the area outside the pattern with a specified color
<code>VG_LITE_PATTERN_PAD</code>	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 50. `vg_lite_radial_gradient_spreadmode_t` enumeration

Value	Description
<code>VG_LITE_RADIAL_GRADIENT_SPREAD_FILL = 0</code>	Coordinates outside the gradient area filled with black color
<code>VG_LITE_RADIAL_GRADIENT_SPREAD_PAD</code>	The area is filled with the closest stop color
<code>VG_LITE_RADIAL_GRADIENT_SPREAD_REPEAT</code>	The gradient is repeated outside the gradient area
<code>VG_LITE_RADIAL_GRADIENT_SPREAD_REFLECT</code>	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**

This structure is defined under the "Pixel buffer structures" section (see [Section 6.5.1](#)).

**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 that 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 maximum number of radial gradient stops is:

```
#define MAX_COLOR_RAMP_STOPS 256
```

Used in radial gradient structure: `vg_lite_radial_gradient_t`.

**Table 51. `vg_lite_color_ramp_t` structure**

<code>vg_lite_color_ramp_t</code> member	Type	Description
stop	<code>vg_lite_float_t</code>	Offset value for the color stop
red	<code>vg_lite_float_t</code>	Red color channel value for the color stop
green	<code>vg_lite_float_t</code>	Green color channel value for the color stop
blue	<code>vg_lite_float_t</code>	Blue color channel value for the color stop
alpha	<code>vg_lite_float_t</code>	Alpha color channel value for the color stop

### 10.2.3 `vg_lite_linear_gradient_parameter_t` structure

This structure defines radial direction for a linear gradient.

*Line0* connects point (X0, Y0) to point (X1, Y1) and represents the radial direction of the linear gradient. *Line1* is a line parallel to *line0* which passes through point (X0, Y0). *Line2* is a line parallel to *line0* which passes through point (X1, Y1)

The linear gradient starts from line1 and ends at line 2.

Used in structure: `vg_lite_linear_gradient_ext`

Used in functions: `vg_lite_set_linear_grad`

**Table 52. `vg_lite_linear_gradient_parameter_t` structure**

<code>vg_lite_linear_gradient_parameter_t</code> members	Type	Description
X0	<code>vg_lite_float_t</code>	X origin of linear gradient radial direction
Y0	<code>vg_lite_float_t</code>	Y origin of linear gradient radial direction
X1	<code>vg_lite_float_t</code>	X end point of linear gradient radial direction
Y1	<code>vg_lite_float_t</code>	Y end point of linear gradient radial direction

### 10.2.4 `vg_lite_linear_gradient_t` structure

This structure defines the organization of a linear gradient in VGLite data. The linear gradient may be applied when filling a path. It generates a 256x1 image according to the configuration 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 53. `vg_lite_linear_gradient_t` structure constants

<code>vg_lite_linear_gradient_t</code> constant	Type	Description
<code>VLC_MAX_GRAD</code>	<code>int32_t</code>	Constant. Maximum number of gradient colors = 16.
<code>VLC_GRADBUFFER_WIDTH</code>	<code>int32_t</code>	Constant. Width of the internal color ramp = 256.

Table 54. `vg_lite_linear_gradient_t` structure members

<code>vg_lite_linear_gradient_t</code> member	Type	Description
<code>colors[VLC_MAX_GRAD]</code>	<code>uint32_t</code>	Color array for the gradient
<code>count</code>	<code>uint32_t</code>	Number of colors
<code>stops[VLC_MAX_GRAD]</code>	<code>uint32_t</code>	Color stop offsets, from 0 to 255
<code>matrix</code>	<a href="#">vg_lite_matrix_t</a>	Transformation matrix to be used to transform the gradient color ramp
<code>image</code>	<a href="#">vg_lite_buffer_t</a>	Image object struct to represent the color ramp

### 10.2.5 `vg_lite_linear_gradient_ext_t` structure

This structure defines the organization of the extended parameters possible for a linear gradient.

Used in functions: `vg_lite_draw_linear_gradient`

Table 55. `vg_lite_linear_gradient_ext_t` structure

<code>vg_lite_linear_gradient_ext_t</code> members	Type	Description
<code>count</code>	<code>uint32_t</code>	Count of colors, up to 256
<code>matrix</code>	<code>vg_lite_matrix_t</code>	The matrix to transform the gradient
<code>image</code>	<code>vg_lite_buffer_t</code>	The image for rendering gradient as pattern
<code>linearGradient</code>	<code>vg_lite_linear_gradient_parameter_t</code>	Linear gradient parameters
<code>vgColorRampLength</code>	<code>uint32_t</code>	Color ramp length for gradient paints provided to the driver
<code>vgColorRamp[MAX_COLOR_RAMP_STOPS]</code>	<code>vg_lite_color_ramp_t</code>	Color ramp colors for gradient paints provided to the driver
<code>intColorRampLength</code>	<code>uint32_t</code>	Converted internal color ramp length
<code>intColorRamp[MAX_COLOR_RAMP_STOPS + 2]</code>	<code>vg_lite_color_ramp_t</code>	Converted internal color ramp
<code>colorRampPremultiplied</code>	<code>uint8_t</code>	If this value is set to 1, the color value of <code>vgColorRamp</code> will be multiplied with the alpha value of <code>vgColorRamp</code>
<code>SpreadMode</code>	<code>vg_lite_radial_gradient_spreadmode_t</code>	The gradient spread mode. This is the same spread mode enumeration type like for radial gradients.



**10.2.6 vg\_lite\_matrix\_t structure**

This structure is defined under the "Matrix control structures" section (see [Section 7.2.1](#)).

**10.2.7 vg\_lite\_path\_t structure**

This structure is defined under the "Vector path structures" section (see [Section 9.2.2](#)).

**10.2.8 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 56. vg\_lite\_radial\_gradient\_parameter\_t structure**

vg_lite_radial_gradient_parameter_t member	Type	Description
<code>cx</code>	<code>vg_lite_float_t</code>	X coordinate of the center point of the gradient
<code>cy</code>	<code>vg_lite_float_t</code>	Y coordinate of the center point of the gradient
<code>r</code>	<code>vg_lite_float_t</code>	Radius of the gradient
<code>fx</code>	<code>vg_lite_float_t</code>	X coordinate of the focal point of the gradient
<code>fy</code>	<code>vg_lite_float_t</code>	Y coordinate of the focal point of the gradient

**10.2.9 vg\_lite\_radial\_gradient\_t structure**

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

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 57. vg\_lite\_radial\_gradient\_t structure**

vg_lite_radial_gradient_t member	Type	Description
<code>count</code>	<code>uint32_t</code>	Count of colors, up to 256
<code>matrix</code>	<a href="#">vg_lite_matrix_t</a>	Structure that specifies the transform matrix for the gradient
<code>image</code>	<a href="#">vg_lite_buffer_t</a>	Structure that specifies the image for rendering as a gradient pattern
<code>radialGradient</code>	<a href="#">vg_lite_radial_gradient_parameter_t</a>	Structure that specifies the location of the center point, focal point, and radius of the gradient
<code>vgColorRampLength</code>	<code>uint32_t</code>	Number of colors in color ramp
<code>vgColorRamp[MAX_COLOR_RAMP_STOPS]</code>	<a href="#">vg_lite_color_ramp_t</a>	Structure that specifies the color ramp
<code>intColorRampLength</code>	<code>uint32_t</code>	Converted internal color ramp length
<code>intColorRamp[MAX_COLOR_RAMP_STOPS+2]</code>	<a href="#">vg_lite_color_ramp_t</a>	Structure that specifies the internal color ramp

Table 57. `vg_lite_radial_gradient_t` structure...continued

vg_lite_radial_gradient_t member	Type	Description
<code>colorRampPremultiplied</code>	<code>uint32_t</code>	If this value is set to 1, then the color value of <code>vgColorRamp</code> is multiplied by the alpha value of <code>vgColorRamp</code>
<code>SpreadMode</code>	<a href="#">vg_lite_radial_gradient_spreadmode_t</a>	Enum that specifies the tiling mode, which 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` function

**Description:**

This function performs a hardware accelerated 2D vector draw operation.

The size of the tessellation buffer can be specified at initialization and it is aligned with the minimum hardware alignment requirements of the kernel. Specifying a smaller size for tessellation buffer allocates less memory but reduces performance. Because the hardware walks the target with the provided tessellation window size, a path may be sent to the hardware multiple times. It is a 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-point fonts, you can set the tessellation buffer to 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
- The hardware does not support strokes; they must be converted to paths before you use them 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:**

Table 58. `vg_lite_draw` function

Parameter	Description
<code>target</code>	Pointer to the <code>vg_lite_buffer_t</code> structure for the destination buffer. All color formats available in the <code>vg_lite_buffer_format_t</code> enum are valid destination formats for the draw function.

Table 58. `vg_lite_draw` function...continued

Parameter	Description
<code>path</code>	Pointer to the <code>vg_lite_path_t</code> structure containing path data that describes the path to draw. See opcode details in <a href="#">fontxml-text-placeholder text="Type the link text"</a> .
<code>fill_rule</code>	Specifies the <code>vg_lite_fill_t</code> enum value for the fill rule for the path
<code>matrix</code>	Pointer to a <code>vg_lite_matrix_t</code> structure that defines the <i>affine</i> transformation matrix of the path. If <code>matrix</code> is NULL, an identity matrix is assumed. <b>Note:</b> <i>Non-affine transformations are not supported by <code>vg_lite_draw</code>; therefore, a perspective transformation matrix might have unexpected effects on path rendering.</i>
<code>blend</code>	Select one of the hardware-supported blend modes in the <code>vg_lite_blend_t</code> enum to be applied to each drawn pixel. If no blending is required, set this value to <code>VG_LITE_BLEND_NONE</code> (0).
<code>color</code>	The color applied to each pixel drawn by the path

### 10.3.2 `vg_lite_draw_gradient` function

**Description:**

This function is used to fill a path with a linear gradient according to the specified fill rules. The specified path is transformed according to the selected matrix and is filled with the specified color 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:**

Table 59. `vg_lite_draw_gradient` function

Parameter	Description
<code>target</code>	Pointer to the <code>vg_lite_buffer_t</code> structure for the destination buffer. All color formats available in the <code>vg_lite_buffer_format_t</code> enum are valid destination formats for this draw function.
<code>path</code>	Pointer to the <code>vg_lite_path_t</code> structure containing path data that describes the path to draw and fill with the linear gradient. See opcode details in <a href="#">Section 9.4</a> .
<code>fill_rule</code>	Specifies the <code>vg_lite_fill_t</code> enum value for the fill rule for the path
<code>matrix</code>	Pointer to a <code>vg_lite_matrix_t</code> structure that defines the 3x3 transformation matrix of the path. If <code>matrix</code> is NULL, an identity matrix is assumed.
<code>grad</code>	Pointer to the <code>vg_lite_linear_gradient_t</code> structure that contains the description of the color gradient to be used to fill the path
<code>blend</code>	Specifies the blend mode in the <code>vg_lite_blend_t</code> enum to be applied to each drawn pixel. If no blending is required, set this value to <code>VG_LITE_BLEND_NONE</code> (0).

10.3.3 `vg_lite_draw_linear_gradient` function

**Description:**

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

**Note:** *Not all VGLite-compatible i.MX RT platforms support the linear gradients extensions. For more details, see [Table 41](#).*

**Syntax:**

```

vg_lite_error_t vg_lite_draw_linear_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_linear_gradient_t *grad,
    vg_lite_color_t        paint_color,
    vg_lite_blend_t        blend,
    vg_lite_filter_t      filter
);
    
```

**Parameters:**

Table 60. `vg_lite_draw_linear_gradient` function

Parameter	Description
target	Pointer to the <a href="#">vg_lite_buffer_t</a> structure for the destination buffer.
path	Pointer to the <a href="#">vg_lite_path_t</a> structure containing path data which designates the path to draw for the linear gradient.
fill_rule	Specifies the <a href="#">vg_lite_fill_t</a> enum value for the fill rule for the path.
path_matrix	Pointer to a <a href="#">vg_lite_matrix_t</a> structure that defines the 3x3 transformation matrix of the vector path. If this matrix is NULL, an identity matrix is assumed.
grad	Pointer to the <a href="#">vg_lite_linear_gradient_ext_t</a> structure which contains the values to be used to fill the path.
paint_color	Specifies the paint color <a href="#">vg_lite_color_t</a> value to be used when <code>VG_LITE_RADIAL_GRADIENT_SPREAD_FILL</code> spread mode is selected using the <a href="#">vg_lite_set_linear_grad</a> API. This paint color is applied to all pixels outside the gradient after transformation. See also enum <a href="#">vg_lite_radial_gradient_spreadmode_t</a> .
blend	Specifies the blend mode in the <a href="#">vg_lite_blend_t</a> enum to be applied to each drawn pixel. If no blending is required, set this argument to <code>VG_LITE_BLEND_NONE</code> (0).

Table 60. `vg_lite_draw_linear_gradient` function...continued

Parameter	Description
<code>filter</code>	Specifies the filter mode <code>vg_lite_filter_t</code> enum value to be applied to each drawn pixel. If no filtering is required, set this argument to <code>VG_LITE_BLEND_POINT (0)</code> .

### 10.3.4 `vg_lite_draw_radial_gradient` function

**Description:**

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

**Note:** Not all VGLite-compatible i.MX RT platforms support the radial gradients feature. For more details, see [Table 82](#)

**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:**

Table 61. `vg_lite_draw_radial_gradient` function

Parameter	Description
<code>target</code>	Pointer to the <code>vg_lite_buffer_t</code> structure for the destination buffer. All color formats available in the <code>vg_lite_buffer_format_t</code> enum are valid destination formats for this draw function.
<code>path</code>	Pointer to the <code>vg_lite_path_t</code> structure containing path data which describes the path to draw for and fill with the radial gradient. See opcode details in <a href="#">Section 9.4</a> .
<code>fill_rule</code>	Specifies the <code>vg_lite_fill_t</code> enum value for the fill rule for the path
<code>path_matrix</code>	Pointer to a <code>vg_lite_matrix_t</code> structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed.
<code>grad</code>	Pointer to the <code>vg_lite_radial_gradient_t</code> structure, which contains the values to be used to fill the path
<code>paint_color</code>	Specifies the paint color enum <code>vg_lite_color_t</code> RGBA value to be applied by <code>VG_LITE_RADIAL_GRADIENT_SPREAD_FILL</code> , which is set by function <code>vg_lite_set_rad_grad</code> . When pixels are out of the image after transformation, this <code>paint_color</code> is applied to them. See also <a href="#">Section 10.1.6</a> .
<code>blend</code>	Specifies the blend mode in the <code>vg_lite_blend_t</code> enum to be applied to each drawn pixel. If no blending is required, set this value to <code>VG_LITE_BLEND_NONE (0)</code> .
<code>filter</code>	Specifies the filter mode <code>vg_lite_filter_t</code> enum value to be applied to each drawn pixel. If no filtering is required, set this value to <code>VG_LITE_BLEND_POINT (0)</code> .

### 10.3.5 vg\_lite\_draw\_pattern function

**Description:**

This function fills a path with an image pattern. The path is transformed according to the specified matrix and is 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:**

Table 62. vg\_lite\_draw\_pattern function

Parameter	Description
target	Pointer to the <code>vg_lite_buffer_t</code> structure for the destination buffer. All color formats available in the <code>vg_lite_buffer_format_t</code> enum are valid destination formats for this draw function.
path	Pointer to the <code>vg_lite_path_t</code> structure containing path data which describes the path to draw. See opcode details in <a href="#">Section 9.4</a>
fill_rule	Specifies the <code>vg_lite_fill_t</code> enum value for the fill rule for the path
matrix0	Pointer to a <code>vg_lite_matrix_t</code> structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed.
source	Pointer to the <code>vg_lite_buffer_t</code> structure that describes the source of the image pattern
matrix1	Pointer to a <code>vg_lite_matrix_t</code> structure that defines the 3x3 transformation matrix of the source pixels into the target. If matrix is NULL, an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
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 <code>VG_LITE_BLEND_NONE (0)</code> .
pattern_mode	Specifies the <code>vg_lite_pattern_mode_t</code> value which defines how the region outside the image pattern is to be filled
pattern_color	Specifies a 32 bpp ARGB color to be applied to the fill outside the image pattern area when the <code>pattern_mode</code> value is <code>VG_LITE_PATTERN_COLOR</code>
filter	Specifies the filter type. All formats available in the <code>vg_lite_filter_t</code> enum are valid formats for this function. A value of zero (0) indicates <code>VG_LITE_FILTER_POINT</code> .

## 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` function

**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:**

Table 63. `vg_lite_init_grad` function

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_linear_gradient_t</a> structure, which defines the gradient to be initialized. Default values are used.

### 10.4.2 `vg_lite_set_grad` function

**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 (for example, due to an empty input array, out-of-range or out-of-order stops), a stop at 0 with (R, G, B, A) 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, then 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, then 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:**

Table 64. `vg_lite_set_grad` function

Parameter	Description
<code>grad</code>	Pointer to the <a href="#">vg_lite_linear_gradient_t</a> structure to be set
<code>count</code>	The number of colors in the linear gradient. The maximum color stop count is defined by <code>VLC_MAX_GRAD</code> which is 16.
<code>colors</code>	Specifies the color array for the gradient stops. The color is in ARGB8888 format with alpha in the upper byte.
<code>stops</code>	Pointer to the gradient stop offset

**Returns:**

Always returns `VG_LITE_SUCCESS`.

**10.4.3 `vg_lite_update_grad` function**

**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 is created or updated with the corresponding gradient parameters.

**Syntax:**

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

**Parameters:**

Table 65. `vg_lite_update_grad` function

Parameter	Description
<code>grad</code>	Pointer to the <a href="#">vg_lite_linear_gradient_t</a> structure, which contains the update values to be used for the object to be rendered

**10.4.4 `vg_lite_get_grad_matrix` function**

**Description:**

This function is used to get a pointer to the transformation matrix of the gradient object. It 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:**

Table 66. `vg_lite_get_grad_matrix` function

Parameter	Description
<code>grad</code>	Pointer to the <a href="#">vg_lite_linear_gradient_t</a> structure, which contains the matrix to be retrieved



10.4.5 `vg_lite_clear_grad` function

**Description:**

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

**Syntax:**

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

**Parameters:**

Table 67. `vg_lite_clear_grad` function

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_linear_gradient_t</a> structure which is to be cleared

10.5 Extended linear gradient initialization and control functions

The following functions are available only on i.MX RT platforms including hardware support for extended linear gradient capabilities. For details about your specific platform, refer to [Table 82](#).

10.5.1 `vg_lite_set_linear_gradient` function

**Description:**

This function is used to set the values that define the linear gradient.

**Note:** *Not all VGLite-compatible i.MX RT platforms support the linear gradients' extensions. For more details, see [Table 82](#).*

**Syntax:**

```
vg_lite_error_t vg_lite_set_linear_grad (
    vg_lite_linear_gradient_ext_t *grad,
    uint32_t count,
    vg_lite_color_ramp_t *vgColorRamp,
    vg_lite_linear_gradient_parameter_t linearGradient,
    vg_lite_radial_gradient_spreadmode_t SpreadMode,
    uint8_t colorRampPremultiplied
);
```

**Parameters:**

Table 68. `vg_lite_set_linear_gradient` function

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_linear_gradient_ext_t</a> structure to configure.
count	Count of the colors in the gradient. The maximum color stop count is defined by <code>MAX_COLOR_RAMP_STOPS</code> , which is set to 256.

Table 68. `vg_lite_set_linear_gradient` function...continued

Parameter	Description
<code>vgColorRamp</code>	It is the array of stops for the linear gradient. The number of parameters for each stop is 5, and gives the offset and color of the stop. Each stop is defined by a floating-point <i>offset</i> value and four floating-point values containing the sRGBA color and alpha value associated with each stop, in the form of a non-premultiplied (R, G, B, alpha) quad. The range of all parameters is [0,1].
<code>linearGradient</code>	Gradient parameters as specified in structure <code>vg_lite_linear_gradient_parameter_t</code> .
<code>SpreadMode</code>	The fill mode applied to the pixels out of the paint after transformation. Uses the same spread mode enumeration types as radial gradient. See <code>vg_lite_radial_gradient_spreadmode_t</code> enum.
<code>colorRampPremultiplied</code>	This parameter controls whether color and alpha values are interpolated in pre-multiplied or non-pre-multiplied form.

### 10.5.2 `vg_lite_get_linear_grad_matrix` function

**Description:**

This function gets the pointer to the linear gradient object's matrix. It allows applications to manipulate the matrix to correctly position the color gradient over the vector polygon.

**Note:** *Not all VGLite-compatible i.MX RT platforms support the linear gradients' extensions. For more details, see [Table 82](#).*

**Syntax:**

```
vg_lite_matrix_t * vg_lite_get_linear_grad_matrix (
    vg_lite_linear_gradient_ext_t *grad
);
```

**Parameters:**

Table 69. `vg_lite_get_linear_grad_matrix` function

Parameter	Description
<code>grad</code>	Pointer to the <code>vg_lite_linear_gradient_ext_t</code> object whose matrix to retrieve.

### 10.5.3 `vg_lite_update_linear_grad` function

**Description:**

This function is used to update or generate the corresponding image object to render.

The `vg_lite_linear_gradient_ext_t` object has an image buffer which is used to render the linear gradient paint. The image buffer is created/updated according to the `grad` parameters specified via the previous call to `vg_lite_set_linear_grad`.

**Note:** *Not all VGLite-compatible i.MX RT platforms support the linear gradients' extensions. For more details, see [Table 82](#).*

**Syntax:**

```
vg_lite_error_t vg_lite_update_linear_gradient (
    vg_lite_linear_gradient_ext_t *grad
);
```

**Parameters:**

Table 70. `vg_lite_update_linear_grad` function

Parameter	Description
grad	Pointer to the <code>vg_lite_linear_gradient_ext_t</code> structure which is to be updated or created.

**10.5.4 `vg_lite_clear_linear_grad` function**

**Description:**

This function is used to clear the linear gradient object. It resets the grad members and free the image buffer's memory.

**Note:** *Not all VGLite-compatible i.MX RT platforms support the linear gradients extensions. For more details, see [Table 82](#).*

**Syntax:**

```
vg_lite_error_t vg_lite_clear_linear_grad (
    vg_lite_linear_gradient_ext_t *grad
);
```

**Parameters:**

Table 71. `vg_lite_clear_linear_grad` function

Parameter	Description
grad	Pointer to the <code>vg_lite_linear_gradient_ext_t</code> structure which is to be cleared.

**10.6 Radial gradient functions initialization and control functions**

This part of the API performs radial gradient operations.

**Note:** *Not all VGLite-compatible i.MX RT platforms support the radial gradients feature. For more details, see [Table 82](#).*

**Note:** *There is no init function required for radial gradients. Buffer initialization is done through the `vg_lite_update_rad_grad()` function.*

**10.6.1 `vg_lite_set_rad_grad` function**

**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:**

**Table 72. vg\_lite\_set\_rad\_grad function**

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_radial_gradient_t</a> structure for the radial gradient that has to be set
count	The number of color stops in the gradient. The maximum color stop count is defined by <code>MAX_COLOR_RAMP_STOPS</code> , which is currently 256.
vgColorRamp	Pointer to the <a href="#">vg_lite_color_ramp_t</a> structure which defines the stops for the radial gradient. The five parameters provide the offset and color for each stop. Each stop is defined by a set of floating point values that 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. Parameters (cx, cy) specify the center point, parameters (fx, fy) specify the focal point, and r specifies the radius. See structure <a href="#">vg_lite_radial_gradient_parameter_t</a> .
spreadMode	The tiling mode that is applied to pixels out of the paint after transformation. See enum <a href="#">vg_lite_radial_gradient_spreadmode_t</a> .
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 <code>vgColorRamp</code> is multiplied by the alpha value of <code>vgColorRamp</code> .

**Returns:**

Returns `VG_LITE_INVALID_ARGUMENTS` to indicate that the parameters are wrong.

**10.6.2 vg\_lite\_update\_rad\_grad function**

**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 is 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:**

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_radial_gradient_t</a> structure, which contains the updated values to be used for the object to be rendered

10.6.3 `vg_lite_get_rad_grad_matrix` function

**Description:**

This function is used to get a pointer to the transformation matrix of the radial gradient object. It 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:**

Table 73. `vg_lite_get_rad_grad_matrix` function

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_radial_gradient_t</a> structure, which contains the matrix to be retrieved

10.6.4 `vg_lite_clear_rad_grad` function

**Description:**

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

**Syntax:**

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

**Parameters:**

Table 74. `vg_lite_clear_rad_grad` function

Parameter	Description
grad	Pointer to the <a href="#">vg_lite_radial_gradient_t</a> structure which is to be cleared

11 Stroke operations

This part of the API performs line stroking operations.

11.1 Stroke enumerations

11.1.1 `vg_lite_cap_style_t` enumeration

Defines the style of a cap at the end of a stroke.

Used in function: `vg_lite_set_stroke`

Table 75. `vg_lite_cap_style_t` enumeration

<code>vg_lite_cap_style_t</code> values	Description
<code>VG_LITE_CAP_BUTT</code>	The <i>butt</i> end cap style terminates each segment with a line perpendicular to the tangent at each endpoint.
<code>VG_LITE_CAP_ROUND</code>	The <i>round</i> end cap style appends a semicircle with a diameter equal to the line width centered around each endpoint.
<code>VG_LITE_CAP_SQUARE</code>	The <i>square</i> end cap style appends a rectangle with two sides of length equal to the line width perpendicular to the tangent, and two sides of length equal to half the line width parallel to the tangent, at each endpoint.

### 11.1.2 `vg_lite_draw_path_type_t` enumeration

Defines the type of vector path to draw.

Used in structure: `vg_lite_path_t`

Used in function: `vg_lite_set_draw_path_type`

Table 76. `vg_lite_draw_path_type_t` enumeration

<code>vg_lite_draw_path_type_t</code> values	Description
<code>VG_LITE_DRAW_FILL_PATH</code>	Vector path is filled (solid color, color gradient, or pattern filled)
<code>VG_LITE_DRAW_STROKE_PATH</code>	Vector path is stroked
<code>VG_LITE_DRAW_FILL_STROKE_PATH</code>	Vector path is both filled (solid color, color gradient or pattern filled) and stroked

### 11.1.3 `vg_lite_join_style_t` enumeration

Defines the type of styles available for line joins.

Used in function: `vg_lite_set_stroke`

Table 77. `vg_lite_join_style_t` enumeration

<code>vg_lite_join_style_t</code> values	Description
<code>VG_LITE_JOIN_MITER</code>	The <i>miter</i> join style appends a trapezoid with one vertex at the intersection point of the two original lines, two adjacent vertices at the outer endpoints of the two "thickened" lines and a fourth vertex at the extrapolated intersection point of the outer perimeters of the two "thickened" lines.
<code>VG_LITE_JOIN_ROUND</code>	The <i>round</i> join style appends a wedge-shaped portion of a circle, centered at the intersection point of the two original lines, having a radius equal to half the line width.
<code>VG_LITE_JOIN_BEVEL</code>	The <i>bevel</i> type join style appends a triangle with two vertices at the outer endpoints of the two "thickened" lines and a third vertex at the intersection point of the two original lines.

### 11.1.4 `vg_lite_join_style_t` enumeration

Defines the type of styles available for line joins.

Used in function: `vg_lite_set_stroke`

Table 78. `vg_lite_join_style_t` enumeration

<code>vg_lite_join_style_t</code> values	Description
<code>VG_LITE_JOIN_MITER</code>	The <i>miter</i> join style appends a trapezoid with one vertex at the intersection point of the two original lines, two adjacent vertices at the outer endpoints of the two “thickened” lines and a fourth vertex at the extrapolated intersection point of the outer perimeters of the two “thickened” lines.
<code>VG_LITE_JOIN_ROUND</code>	The <i>round</i> join style appends a wedge-shaped portion of a circle, centered at the intersection point of the two original lines, having a radius equal to half the line width.
<code>VG_LITE_JOIN_BEVEL</code>	The <i>bevel</i> type join style appends a triangle with two vertices at the outer endpoints of the two “thickened” lines and a third vertex at the intersection point of the two original lines.

## 11.2 Stroke structures

### 11.2.1 `vg_lite_path_t` structure

Defined under Vector Path Structures - [vg\\_lite\\_path\\_t](#) structure.

## 11.3 Stroke functions

### 11.3.1 `vg_lite_set_draw_path_type` function

**Description:**

This function sets the vector path type. By default, a vector path is considered `VG_LITE_DRAW_FILL_PATH`.

**Syntax:**

```
vg_lite_error_t vg_lite_set_draw_path_type (
    vg_lite_path_t *path,
    vg_lite_draw_path_type_t path_type
);
```

**Parameters:**

Table 79. `vg_lite_set_draw_path_type` function

Parameter	Description
<code>path</code>	Pointer to the <a href="#">vg_lite_path_t</a> structure that describes the vector path.
<code>path_type</code>	The type to set for the mentioned vector path.

### 11.3.2 `vg_lite_set_stroke` function

**Description:**

This function sets the attributes of a stroked vector path.

**Syntax:**

```

vg_lite_error_t vg_lite_set_stroke (
    vg_lite_path_t          *path,
    vg_lite_cap_style_t     stroke_cap_style,
    vg_lite_join_style_t    stroke_join_style,
    vg_lite_float_t         stroke_line_width,
    vg_lite_float_t         stroke_miter_limit,
    vg_lite_float_t         *stroke_dash_pattern,
    uint32_t                stroke_dash_pattern_count,
    vg_lite_float_t         stroke_dash_phase
);
    
```

**Parameters:**

**Table 80. vg\_lite\_set\_stroke function**

Parameter	Description
path	Pointer to the <a href="#">vg_lite_path_t</a> structure that describes the vector path.
stroke_cap_style	The end cap style defined by the <a href="#">vg_lite_cap_style_t</a> enum.
stroke_join_style	The line join style defined by the <a href="#">vg_lite_join_style_t</a> enum.
stroke_line_width	The line width of the stroked path. A line width less than or equal to 0 prevents stroking from taking place.
stroke_miter_limit	When stroking using the <i>miter</i> line join style, the miter length (that is, the length between the intersection of the inner and outer perimeters of the two “thickened” lines) is compared to the product of the user-set miter limit and the line width. If the miter length exceeds this product, the <i>miter</i> join is not drawn and a <i>bevel</i> join is substituted. Note: All miter limit values less than 1 are silently clamped to 1.
stroke_dash_pattern	Pointer to a dash pattern which consists of a sequence of lengths of alternating “on” and “off” dash segments. The first value of the dash array defines the length, in user coordinates, of the first “on” dash segment. The second value defines the length of the following “off” segment. Each subsequent pair of values defines one “on” and one “off” segment. Note: If the dash pattern has an odd number of elements, the final element is ignored.
stroke_dash_pattern_count	The count of dash on/off segments.
stroke_dash_phase	Defines the starting point in the dash pattern that is associated with the start of the first segment of the path. For example, if the dash pattern is [10 20 30 40] and the dash phase is 35, the path will be stroked with an “on” segment of length 25 (skipping the first “on” segment of length 10, the following “off” segment of length 20, and the first 5 units of the next “on” segment), followed by an “off” segment of length 40. The pattern will then repeat from the beginning, with an “on” segment of length 10, an “off” segment of length 20, an “on” segment of length 30.



### 11.3.3 vg\_lite\_set\_update\_stroke function

**Description:**

This function uses the path and stroke attributes as specified by function `vg_lite_set_stroke` to update the stroked path's parameters and generate the stroked path data.

**Syntax:**

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

**Parameters:**

Table 81. `vg_lite_set_update_stroke` function

Parameter	Description
<code>path</code>	Pointer to the <a href="#">vg_lite_path_t</a> structure that describes the vector path.

## 12 Platform-specific features

The table below describes VGLite features that are supported by some but not all NXP VGLite-compatible i.MX RT platforms. The features that are not mentioned here are supported by all NXP VGLite-compatible i.MX RT platforms.

Table 82. Platform-specific VGLite features

VGLite feature	Supported? (Yes/No)		
	i.MX RT500	i.MX RT1160	i.MX RT1170
2 bits per channel image formats (ARGB2222, BGRA2222, ABGR2222, RGBA2222)	Yes	No	No
Indexed image formats (1, 2, 4, and 8 bits per pixel)	Yes	No	No
8x coverage sample anti-aliasing for vector paths ( <code>VG_LITE_UPPER</code> )	Yes	No	No
Border culling	Yes	No	No
Alpha channel premultiplication during <code>vg_lite_blit</code>	No	Yes	Yes
Dithering	No	Yes	Yes
Color Keying	No	Yes	Yes
Radial gradients	No	Yes	Yes
Linear gradients extensions	No	Yes	Yes

## 13 VGLite API programming examples

This chapter provides a set of VGLite API programming examples.

### 13.1 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:** Because the tessellation buffer width and height are defined as (0, 0) in the call to `vg_lite_init`. This application cannot use the path rendering `vg_lite_draw` APIs. Only clear and blit APIs can be used in this context.

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 using the `vg_lite_clear` API:

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

Then, it paints red 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 Vivante GPU 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();
```

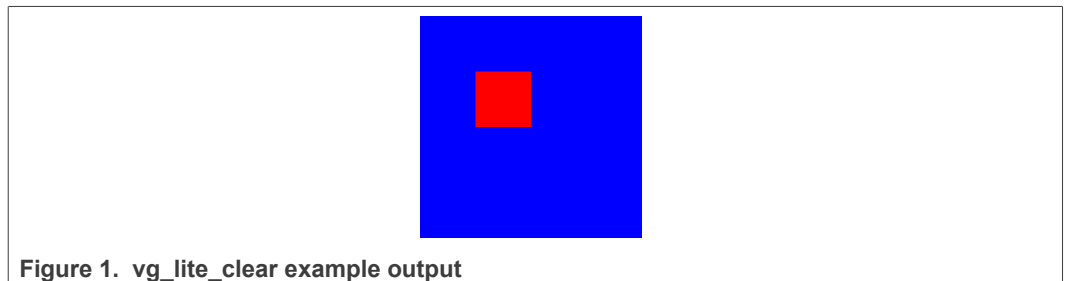


Figure 1. `vg_lite_clear` example output

### 13.2 vg\_lite\_blit example

This section describes an example program demonstrating the usage of the `vg_lite_blit` API. It first clears a 320x480 render buffer with blue background color, and then blits six 256x256 icon images to six different positions on the render buffer for each icon, using a blit matrix. 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 sets the blend mode to `VG_LITE_BLEND_SRC_OVER` so the icon image pixels with an alpha value `0xFF` over 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. `vg_lite_blit` example output

### 13.3 `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. It also 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 foreground 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)` should be called to free and reset the path data.

### 13.4 vg\_lite\_draw\_gradient example

This section demonstrates the usage of the `vg_lite_draw_gradient` API. It defines five colors (black, red, green, blue, white) in the `ramps[]` array and five stops in `stops[]` array which are used for gradient color transition. The application uses the following sequence of calls to set up 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 highest byte.

The application configures 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 gradient draw 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);
```

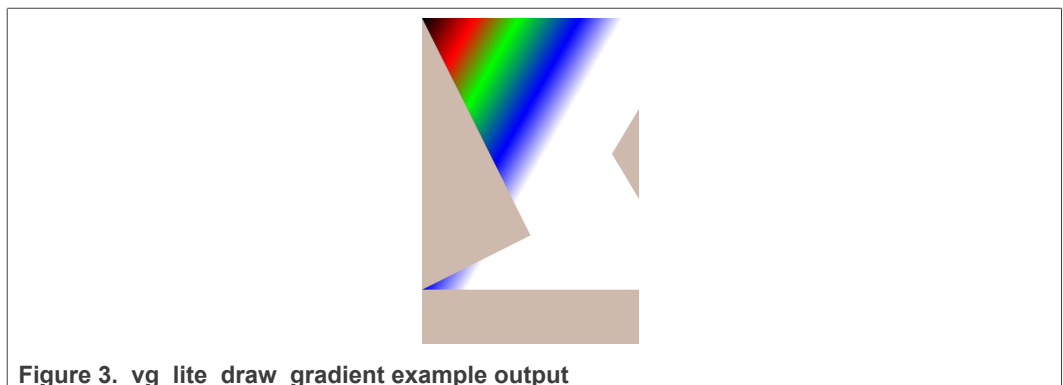


Figure 3. `vg_lite_draw_gradient` example output

### 13.5 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 (shown in [Figure 4](#)) and loads an image file `landscape.raw` to be used to fill the interior area of the polygon.

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

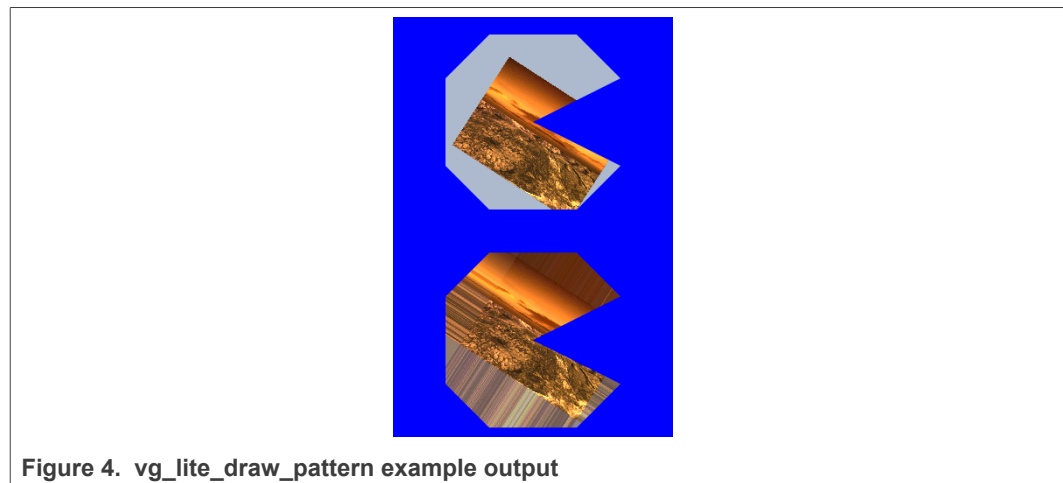


Figure 4. `vg_lite_draw_pattern` example output

### 13.6 Vector-based font-rendering example

This section demonstrates vector-based font rendering with the `vg_lite_draw` API. This API can be used to draw quadratic curves and cubic curves based on end point and control point coordinates in the path data. Font path data can be generated either using

a third-party font engine that can produce VGLite path data directly or using VGLite tools from VeriSilicon that can convert font data of other formats (such as SVG) 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 on each line is the path opcode, followed by the coordinates for each opcode. As listed in [Section 9.4](#):

- 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 the following function to initialize VGLite with a 256x256 path tessellation buffer, and then allocates a 320x320 render buffer with the VG\_LITE\_RGBA8888 format. The size of the tessellation buffer is large enough to cover the font character bounding box.

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

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

```
/* Draw the path using the matrix.*/
error = vg_lite_draw(fb, &path, VG_LITE_FILL_EVEN_ODD, &matrix,
    VG_LITE_BLEND_NONE, 0xFF0000FF);
```

The vector path of the character is rendered without blending (VG\_LITE\_BLEND\_NONE). The path interior is filled with the red color (0xFF0000FF).



*Hello,  
Verisilicon!*

Figure 5. Text rendering example output

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 the 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”.



*H*

Figure 6. Letter rendering example output

## 14 Revision history

The [table](#) below summarizes the revisions to this document.

Table 83. Revision history

Revision	Date	Topic cross-reference	Change description
Rev. 1.1	22 September 2022		<ul style="list-style-type: none"> <li>• Paragraph 4.1.1 Updated <i>Table 3 - vg_lite_feature_t</i> enumeration.</li> <li>• Paragraph 6.6 Added documentation for new API <i>vg_lite_set_dither</i></li> <li>• Paragraph 8.2 Blit structures- Added documentation for new data structure <i>vg_lite_color_key_t</i> -; added documentation for new data structure <i>vg_lite_color_key4_t</i></li> <li>• Paragraph 8.3.1, <i>vg_lite_blit</i> function- added note related to HW limitation on RT500 platform</li> <li>• Paragraph 8.3.2, <i>vg_lite_blit_rect</i> function -added note related to HW limitation on RT500 platforms</li> <li>• Paragraph 8.3.3, <i>vg_lite_get_transform_matrix</i> function- adjusted function description, adjusted function parameters description</li> <li>• Paragraph 8.3, blit functions- added documentation for new API <i>vg_lite_set_color_key</i></li> <li>• Paragraph 8.4.1, <i>vg_lite_enable_premultiply</i> function- added note about limited support on specific platforms</li> <li>• Paragraph 8.4.2, <i>vg_lite_disable_premultiply</i> function- added note about limited support on specific platforms</li> <li>• Paragraph 10.1.3, <i>vg_lite_fill_t</i> enumeration- added note about crossing points buffer limitation</li> <li>• Paragraph 10.2, draw and gradient structures- added documentation for new data structure <i>vg_lite_gradient_parameter_t -done</i>- added documentation for new data structure <i>vg_lite_gradient_ext_t</i></li> <li>• Paragraph 10.3, draw functions- added documentation for new API <i>vg_lite_draw_linear_gradient</i></li> <li>• Paragraph 10, vector-Based Draw Operations - added new paragraph 10.5 <i>Extended linear gradient initialization and control functions</i>; added documentation for new API <i>vg_lite_set_linear_gradient</i> ; added documentation for new API <i>vg_lite_get_linear_grad_matrix</i>; added documentation for new API <i>vg_lite_update_linear_grad</i>; Added documentation for new API <i>vg_lite_clear_linear_grad</i></li> <li>• Paragraph 10.5, Radial gradient functions - adjusted paragraph title</li> <li>• Added new Chapter <i>Stroke Operations</i></li> <li>• Chapter <i>Platform-Specific Features</i> - updated <i>Table 41</i> - Platform-specific VGLite features</li> </ul>



**Table 83. Revision history...continued**

Revision	Date	Topic cross-reference	Change description
Rev. 1	27 January 2022	<a href="#">Section 1</a>	Added i.MX RT1160 to the list of NXP devices that support VGLite graphics API
		<a href="#">Section 3.2.1</a>	Updated <a href="#">Table 2</a>
		<a href="#">Section 4.1.1</a>	Updated <a href="#">Table 3</a>
		<a href="#">Section 5</a>	Updated chapter introductory text
		<a href="#">Section 5.1.2</a>	Updated section
		<a href="#">Section 6.1</a>	Updated section
		<a href="#">Section 6.4.1</a>	Updated section
		<a href="#">Section 6.4.1.1</a>	Updated section
			Removed "vg_lite_perspective function" section
		<a href="#">Section 8.2.7</a>	Added as a new section
		<a href="#">Section 8.2.8</a>	Added as a new section
		<a href="#">Section 8.3.3</a>	Added as a new section
		<a href="#">Section 8.4.5</a>	Updated section
		<a href="#">Section 9.1.2</a>	Updated section
		<a href="#">Section 9.3.4</a>	Added as a new section
<a href="#">Section 9.4</a>	Added <a href="#">Table 47</a>		
<a href="#">Section 12</a>	Added as a new chapter		
Rev. 0	22 February 2021		Initial release

## 15 Legal information

### 15.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

### 15.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

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

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <http://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at [PSIRT@nxp.com](mailto:PSIRT@nxp.com)) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

### 15.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>	7.3.1	vg_lite_identity function	22
<b>2</b>	<b>Vivante VGLite Graphics API</b>	<b>2</b>	7.3.2	vg_lite_rotate function	22
2.1	API partitions	2	7.3.3	vg_lite_scale function	23
2.2	API files	2	7.3.4	vg_lite_translate function	23
<b>3</b>	<b>Common parameters and error values</b>	<b>2</b>	<b>8</b>	<b>Blits for compositing and blending</b>	<b>23</b>
3.1	Common parameter types	3	8.1	Blit enumerations	24
3.2	Enumerations for error reporting	3	8.1.1	vg_lite_blend_t enumeration	24
3.2.1	vg_lite_error_t enumeration	3	8.1.2	vg_lite_color_t parameter	28
<b>4</b>	<b>Hardware product and feature information</b>	<b>4</b>	8.1.3	vg_lite_filter_t enumeration	28
4.1	Enumerations for product and feature queries	4	8.2	Blit structures	28
4.1.1	vg_lite_feature_t enumeration	4	8.2.1	vg_lite_buffer_t structure	28
4.2	Structures for product and feature queries	4	8.2.2	vg_lite_color_key_t structure	28
4.2.1	vg_lite_info_t structure	4	8.2.3	vg_lite_color_key4_t structure	29
4.3	Functions for product and feature queries	5	8.2.4	vg_lite_matrix_t structure	29
4.3.1	vg_lite_get_product_info function	5	8.2.5	vg_lite_path_t structure	29
4.3.2	vg_lite_get_info function	5	8.2.6	vg_lite_rectangle_t structure	29
4.3.3	vg_lite_get_register function	5	8.2.7	vg_lite_point_t structure	29
4.3.4	vg_lite_query_feature function	6	8.2.8	vg_lite_point4_t structure	30
4.3.5	vg_lite_mem_avail function	6	8.3	Blit functions	30
<b>5</b>	<b>API control</b>	<b>7</b>	8.3.1	vg_lite_blit function	30
5.1	Context initialization and control functions	7	8.3.2	vg_lite_blit_rect function	31
5.1.1	vg_lite_set_command_buffer_size function	7	8.3.3	vg_lite_get_transform_matrix function	32
5.1.2	vg_lite_init function	7	8.3.4	vg_lite_clear function	33
5.1.3	vg_lite_close function	8	8.3.5	vg_lite_set_color_key function	33
5.1.4	vg_lite_finish function	8	8.4	Premultiply and scissor functions	34
5.1.5	vg_lite_flush function	8	8.4.1	vg_lite_enable_premultiply function	34
<b>6</b>	<b>Pixel buffers</b>	<b>9</b>	8.4.2	vg_lite_disable_premultiply function	35
6.1	Pixel buffer alignment	9	8.4.3	vg_lite_enable_scissor function	35
6.2	Pixel cache	9	8.4.4	vg_lite_disable_scissor function	35
6.3	Internal representation	9	8.4.5	vg_lite_set_scissor function	35
6.4	Pixel buffer enumerations	9	<b>9</b>	<b>Vector path control</b>	<b>36</b>
6.4.1	vg_lite_buffer_format_t enumeration	10	9.1	Vector path enumerations	36
6.4.1.1	Alignment notes	14	9.1.1	vg_lite_format_t enumeration	36
6.4.2	vg_lite_buffer_image_mode_t enumeration	15	9.1.2	vg_lite_quality_t enumeration	36
6.4.3	vg_lite_buffer_layout_t enumeration	15	9.2	Vector path structures	36
6.4.4	vg_lite_buffer_transparency_mode_t enumeration	16	9.2.1	vg_lite_hw_memory structure	36
6.4.5	vg_lite_swizzle_t enumeration	16	9.2.2	vg_lite_path_t structure	37
6.4.6	vg_lite_yuv2rgb_t enumeration	16	9.3	Vector path functions	38
6.5	Pixel buffer structures	16	9.3.1	vg_lite_path_calc_length function	38
6.5.1	vg_lite_buffer_t structure	16	9.3.2	vg_lite_path_append function	39
6.5.2	vg_lite_yuvinfo_t structure	17	9.3.3	vg_lite_init_path function	39
6.6	Pixel buffer functions	18	9.3.4	vg_lite_init_arc_path function	40
6.6.1	vg_lite_allocate function	18	9.3.5	vg_lite_upload_path function	41
6.6.2	vg_lite_free function	18	9.3.6	vg_lite_clear_path function	41
6.6.3	vg_lite_buffer_upload function	19	9.4	Vector path opcodes for plotting paths	42
6.6.4	vg_lite_map function	19	<b>10</b>	<b>Vector-dased draw operations</b>	<b>45</b>
6.6.5	vg_lite_unmap function	20	10.1	Draw and gradient enumerations	45
6.6.6	vg_lite_set_CLUT function	20	10.1.1	vg_lite_blend_t enumeration	45
6.6.7	vg_lite_set_dither	21	10.1.2	vg_lite_color_t parameter	45
<b>7</b>	<b>Matrices</b>	<b>21</b>	10.1.3	vg_lite_fill_t enumeration	45
7.1	Matrix control float parameter type	21	10.1.4	vg_lite_filter_t enumeration	46
7.2	Matrix control structures	21	10.1.5	vg_lite_pattern_mode_t enumeration	46
7.2.1	vg_lite_matrix_t structure	22	10.1.6	vg_lite_radial_gradient_spreadmode_t enumeration	46
7.3	Matrix control functions	22	10.2	Draw and gradient structures	46
			10.2.1	vg_lite_buffer_t structure	46

10.2.2	vg_lite_color_ramp_t structure .....	46	15	Legal information .....	74
10.2.3	vg_lite_linear_gradient_parameter_t structure .....	47			
10.2.4	vg_lite_linear_gradient_t structure .....	47			
10.2.5	vg_lite_linear_gradient_ext_t structure .....	48			
10.2.6	vg_lite_matrix_t structure .....	49			
10.2.7	vg_lite_path_t structure .....	49			
10.2.8	vg_lite_radial_gradient_parameter_t structure .....	49			
10.2.9	vg_lite_radial_gradient_t structure .....	49			
10.3	Draw functions .....	50			
10.3.1	vg_lite_draw function .....	50			
10.3.2	vg_lite_draw_gradient function .....	51			
10.3.3	vg_lite_draw_linear_gradient function .....	52			
10.3.4	vg_lite_draw_radial_gradient function .....	53			
10.3.5	vg_lite_draw_pattern function .....	54			
10.4	Linear gradient initialization and control functions .....	54			
10.4.1	vg_lite_init_grad function .....	55			
10.4.2	vg_lite_set_grad function .....	55			
10.4.3	vg_lite_update_grad function .....	56			
10.4.4	vg_lite_get_grad_matrix function .....	56			
10.4.5	vg_lite_clear_grad function .....	57			
10.5	Extended linear gradient initialization and control functions .....	57			
10.5.1	vg_lite_set_linear_gradient function .....	57			
10.5.2	vg_lite_get_linear_grad_matrix function .....	58			
10.5.3	vg_lite_update_linear_grad function .....	58			
10.5.4	vg_lite_clear_linear_grad function .....	59			
10.6	Radial gradient functions initialization and control functions .....	59			
10.6.1	vg_lite_set_rad_grad function .....	59			
10.6.2	vg_lite_update_rad_grad function .....	60			
10.6.3	vg_lite_get_rad_grad_matrix function .....	61			
10.6.4	vg_lite_clear_rad_grad function .....	61			
<b>11</b>	<b>Stroke operations .....</b>	<b>61</b>			
11.1	Stroke enumerations .....	61			
11.1.1	vg_lite_cap_style_t enumeration .....	61			
11.1.2	vg_lite_draw_path_type_t enumeration .....	62			
11.1.3	vg_lite_join_style_t enumeration .....	62			
11.1.4	vg_lite_join_style_t enumeration .....	62			
11.2	Stroke structures .....	63			
11.2.1	vg_lite_path_t structure .....	63			
11.3	Stroke functions .....	63			
11.3.1	vg_lite_set_draw_path_type function .....	63			
11.3.2	vg_lite_set_stroke function .....	63			
11.3.3	vg_lite_set_update_stroke function .....	65			
<b>12</b>	<b>Platform-specific features .....</b>	<b>65</b>			
<b>13</b>	<b>VGLite API programming examples .....</b>	<b>65</b>			
13.1	vg_lite_clear example .....	66			
13.2	vg_lite_blit example .....	66			
13.3	vg_lite_draw example .....	67			
13.4	vg_lite_draw_gradient example .....	68			
13.5	vg_lite_draw_pattern example .....	68			
13.6	Vector-based font-rendering example .....	69			
<b>14</b>	<b>Revision history .....</b>	<b>71</b>			

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.