Vivante Programming: VGLite® Vector Graphics API

Note: This document is released as part of PSOC™-E84 Early Access Pack (EAP).

About this document

Scope and purpose

This document describes various graphics APIs available for user to make use of GPU in PSOC™ Edge Devices.

Intended audience

This document is intended for anyone who wants to offload the graphics operations to GPU in PSOC™ Edge Devices.

Table of contents

Table of contents

Table of contents

1 Introduction

Vivante’s platform independent VGLite Graphics API (Application Programming Interface) is designed to support 2D vector and 2D raster-based operations for rendering interactive user interface that may include menus, fonts, curves, and images. Its goal is to provide the maximum 2D vector/raster rendering performance, while keeping the memory footprint to the minimum. The Vivante VGLite Graphics API allows users to implement customized applications for mobile or IoT devices that include Vivante Vector Graphics IP.

1.1 Vivante VGLite graphics API

VGLite APIs for GCNanoUltraV is used to control the vector graphics hardware units present in PSOC Edge MCU, which provide accelerated vector and raster operations.

VGLite APIs for GCNanoUltraV is developed for use with GCNanoUltraV hardware. VGLite API driver V4 can generate the most CPU efficient, customized driver build for a specific 2.5D GPU release based on the hardware feature set. VGLite API supported features include Porter-Duff Blending, Gradient Controls, Fast Clear, Arbitrary Rotations, Path Filling rules, Path painting, and Pattern Path Filling.

GCNanoUltraV have limted feature set that are required to pass Khronos OpenVG CTS.

By default, VGLite API driver V4 supports one implicit global application context in single thread. VGLite V4 driver does not support multi-threaded application which is not suitable for embedded IoT devices.

1.2 API function group

VGLite Graphics API has been designed to have independent function groups. It is permissible for user to use only one of the function groups in the VGLite application.

- **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

1.3 API files

For customers with access to source:

VGLite Graphics API functions are defined in the header file *vg_lite.h*.

All VGLite application used enumerations and data types are also defined in *vg_lite.h*.

1.4 Hardware versions

The VGLite APIs for GCNanoUltraV is compatible with a range of Vivante Vector Graphics IPs including: GCNanoLiteV, GCNanoUltraV, GCNanoV.

Note that a specific hardware version has customized feature set which may limit hardware support for some VGLite API options. The VGLite application can use vg_lite_query_feature API to query specific VGLite feature availability.

Users can also check the *vg_lite_options.h* file which includes CHIPID, REVISION, CID to identify specific HW releases, and gcFEATURE_VG_* macros to define the feature set for the HW release.

The gcFEATURE_VG_* macro values (except a few SW features) should NOT be changed. Otherwise, the VGLite driver will not function correctly on the specific HW release. Users can change the "SW Features" macro values to disable some software features, unnecessary error checks, or enable VGLite API trace for debug purposes.

2 Common parameters and error values

2.1 Common parameters

VGLite Graphics API uses a naming convention scheme wherein definitions are preceded by "vg_lite".

Below is the list of the most proprietary defined types and structures in the drivers. Not all may be used in the API functions.

Table 1 Common parameter types

Name	Typedef	Value
vg_lite_bool_t	int	A signed 32-bit integer 0: FALSE; 1: TRUE.
vg_lite_int8_t	char	A signed 8-bit integer
vg_lite_uint8_t	unsigned char	An unsigned 8-bit integer
vg_lite_int16_t	short	A signed 16-bit integer
vg_lite_uint16_t	unsigned short	An unsigned 16-bit integer
vg_lite_int32_t	int	A signed 32-bit integer
vg_lite_uint32_t	unsigned int	An unsigned 32-bit integer
vg_lite_uint64_t	unsigned long long	An unsigned 64-bit integer
vg_lite_float_t	float	A 32-bit single precision floating point number
vg_lite_double_t	double	A 64-bit double precision floating point number
vg_lite_char_t	char	A signed 8-bit integer
vg_lite_string	char*	A pointer to a character string
vg_lite_pointer	void*	A generic address pointer (void *). On 32-bit OS, it is a 32-bit address pointer. On 64-bit OS, it is a 64-bit address pointer.
vg_lite_void	void	The void type
vg_lite_color_t	vg_lite_uint32_t	A 32-bit color value The color value specifies the color used in various functions. The color is formed using 8-bit RGBA channels. The red channel is in the lower 8-bit of the color value, followed by the green and blue channels. The alpha channel is in the upper 8-bit of the color value.

	31:24	23:16	15:8	7:0
vg_lite_color_t	A	B	G	R

		For L8 target formats, the RGB color is converted to L8 by using the default ITU-R BT.709 conversion rules.
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

Name	Typedef	Value
VG_LITE_FP32	enum vg_lite_format_t	A 32-bit floating point coordinate

2.2 Enumeration used for error reporting

2.2.1 vg_lite_error_t enumeration

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

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

vg_lite_error_t string values	Description
VG_LITE_GENERIC_IO	Cannot communicate with the kernel driver
VG_LITE_INVALID_ARGUMENT	Invalid argument specified
VG_LITE_MULTI_THREAD_FAIL	Multi-thread/tasks fail *(available from June 2020)*
VG_LITE_NO_CONTEXT	No context specified
VG_LITE_NOT_SUPPORT	Function call not supported. Hardware support not available.
VG_LITE_OUT_OF_MEMORY	Out of memory (driver heap)
VG_LITE_OUT_OF_RESOURCES	Out of resources (OS heap)
VG_LITE_SUCCESS	Successful with no errors
VG_LITE_TIMEOUT	Timeout
VG_LITE_ALREADY_EXISTS	Object already exists *(available from August 2021)*
VG_LITE_NOT_ALIGNED	Data alignment error *(available from August 2021)*

3 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.

3.1 Enumerations for product and feature queries

3.1.1 vg_lite_feature_t enumeration

The following feature values may be queried for availability in compatible hardware. *(expanded March 2023 to support additional hardware for driver V4)*

Used in information functions: vg_lite_query_feature.

vg_lite_feature_t string values	**Description**
gcFEATURE_BIT_VG_16PIXELS_ALIGN	Require 16 pixels aligned for input pixel buffer
gcFEATURE_BIT_VG_24BIT	RGB888 or RGBA5658 formats support
gcFEATURE_BIT_VG_24BIT_PLANAR	24-bit planar formats support
gcFEATURE_BIT_VG_AYUV_INPUT	AYUV input format support
gcFEATURE_BIT_VG_BORDER_CULLING	Border culling support
gcFEATURE_BIT_VG_COLOR_KEY	Color key support.
gcFEATURE_BIT_VG_COLOR_TRANSFORMATION	Color transform support.
gcFEATURE_BIT_VG_DEC_COMPRESS	DEC compression format output support
gcFEATURE_BIT_VG_DITHER	Dither support
gcFEATURE_BIT_VG_DOUBLE_IMAGE	Support two image source inputs
gcFEATURE_BIT_VG_FLEXA	FLEXA interface support
gcFEATURE_BIT_VG_GAMMA	Gamma support
gcFEATURE_BIT_VG_GAUSSIAN_BLUR	Gaussian blur sampling support
gcFEATURE_BIT_VG_GLOBAL_ALPHA	Global alpha support
gcFEATURE_BIT_VG_HW_PREMULTIPLY	HW supports alpha premultiply for image
gcFEATURE_BIT_VG_IM_DEC_INPUT	DEC compressed format input support
gcFEATURE_BIT_VG_IM_FASTCLEAR	Fast Clear support
gcFEATURE_BIT_VG_IM_INDEX_FORMAT	Index format support for image
gcFEATURE_BIT_VG_IM_INPUT	Blit and draw API support
gcFEATURE_BIT_VG_IM_REPEAT_REFLECT	Image repeat reflect mode support
gcFEATURE_BIT_VG_INDEX_ENDIAN	Index format endian support
gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT	Support for extended linear gradient capabilities
gcFEATURE_BIT_VG_LVGL_SUPPORT	LVGL blend mode support
gcFEATURE_BIT_VG_MASK	Mask support
gcFEATURE_BIT_VG_MIRROR	Mirror support
gcFEATURE_BIT_VG_NEW_BLEND_MODE	New blend mode DARKEN/LIGHTEN support
gcFEATURE_BIT_VG_NEW_IMAGE_INDEX	New CLUT image index support
gcFEATURE_BIT_VG_PARALLEL_PATHS	New parallel path HW support

vg_lite_feature_t string values	Description
gcFEATURE_BIT_VG_PE_CLEAR	Pixel engine clear support
gcFEATURE_BIT_VG_PIXEL_MATRIX	Pixel matrix support
gcFEATURE_BIT_VG_QUALITY_8X	8x anti-aliasing path support
gcFEATURE_BIT_VG_RADIAL_GRADIENT	Radial gradient support
gcFEATURE_BIT_VG_RECTANGLE_TILED_OUT	Rectangle tiled output support
gcFEATURE_BIT_VG_RGBA2_FORMAT	RGBA2222 format support
gcFEATURE_BIT_VG_RGBA8_ETC2_EAC	ETC2/EAC compressed image format support
gcFEATURE_BIT_VG_SCISSOR	Scissor support
gcFEATURE_BIT_VG_SRC_PREMULTIPLIED	Source image alpha premultiplied
gcFEATURE_BIT_VG_STENCIL	Stencil image mode support
gcFEATURE_BIT_VG_STRIPE_MODE	Stripe mode support
gcFEATURE_BIT_VG_TESSELLATION_TILED_OUT	Tessellation tiled output support
gcFEATURE_BIT_VG_USE_DST	Read destination pixel support
gcFEATURE_BIT_VG_YUV_INPUT	YUV input format support
gcFEATURE_BIT_VG_YUV_OUTPUT	YUV format output support
gcFEATURE_BIT_VG_YUV_TILED_INPUT	YUV tiled input format support
gcFEATURE_BIT_VG_YUY2_INPUT	YUY2 input format support

3.2 Structures for product and feature queries

3.2.1 vg_lite_info_t structure

This structure is used to query VGLite driver information.

Used in function: vg_lite_get_info_t

vg_lite_info_t members	Type	Description
api_version	vg_lite_uint32_t	VGLite API version
header_version	vg_lite_uint32_t	VGLite header version
release_version	vg_lite_uint32_t	VGLite driver release version
reserved	vg_lite_uint32_t	Reserved for future use

3.3 Functions for product and feature queries

3.3.1 vg_lite_get_product_info

Description

This function is used to identify the VGLite compatible product.

Syntax

```
vg_lite_uint32_t vg_lite_get_product_info (
    vg_lite_char                *name
    vg_lite_uint32_t            *chip_id
    vg_lite_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.

3.3.2 vg_lite_get_info

Description

This function is used to query the VGLite driver information.

Syntax

```
vg_lite_error_t 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.

3.3.3 vg_lite_get_register

Description

This function can be used to read a Vivante Vector Graphics register value given the AHB Byte address of a register. Refer to Vivante Vector Graphics Accessible Register specification documents compatible with your IP for register descriptions. The value range of AHB/APB accessible addresses for VGLite cores is usually 0x0 to 0x1FF and 0xA00 to 0xA7F.

Syntax

```
vg_lite_error_t vg_lite_get_register (
    vg_lite_uint32_t            address
    vg_lite_uint32_t            *result

);
```

Parameters

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

3.3.4 vg_lite_query_feature

Description

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

Syntax

```
vg_lite_uint32_t vg_lite_query_feature (
    vg_lite_feature_t          feature

);
```

Parameters

feature	Feature being queried, as detailed in enum vg_lite_feature_t.

Returns

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

3.3.5 vg_lite_get_mem_size

Description

This function queries whether or not there is any remaining allocated contiguous video memory *(available from June 2020)*.

Syntax

```
vg_lite_error_t vg_lite_get_mem_size(
    vg_lite_uint32_t            *size

);
```

Parameters

size	Pointer to the remaining allocated contiguous video memory.

Returns

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

4 API control

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

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

4.1 Context initialization and control functions

4.1.1 vg_lite_init

Description

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

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

Syntax

```
vg_lite_error_t vg_lite_init (
    vg_lite_int32_t              tess_width,
    vg_lite_int32_t              tess_height

);
```

Parameters

tess_width	Width of tessellation window. The maximum width must not exceed the render buffer width. If it is less than or equal to '0', no tessellation buffer is created and only blit APIs can be used afterwards.
tess_height	Height of tessellation window. The maximum height must not exceed the render buffer width. If it is less than or equal to '0', no tessellation buffer is created and blit APIs can be used afterwards.

Returns

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

4.1.2 vg_lite_init_mem

Description

This function initializes the hardware memory by setting up the base address for GPU registers and base address for the contiguous memory regions. It also configures the heap size for the memory region.

Syntax

```
void vg_lite_init_mem(
     vg_module_parameters_t     *param
 );
```

Parameters

*param	Pointer to a structure containing memory configuration parameters, including base addresses and heap size.

4.1.3 vg_lite_close

Description

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

Syntax

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

Returns

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

4.1.4 vg_lite_flush

Description

This function explicitly submits the command buffer to the GPU without waiting for it to complete. *(From Dec 2019, return type is vg_lite_error_t, previously was void.)*

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.

4.1.5 vg_lite_finish

Description

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

Syntax

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

Returns

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

4.1.6 vg_lite_frame_delimiter

Description

This function sets a flag for GPU to signal the completion of the current frame. Be default, the `vg_lite_finish` function is called within this API. The enum VG_LITE_FRAME_END_FLAG is the only value that can be set by flag parameter.

Syntax

```
vg_lite_error_t vg_lite_frame_delimiter (
    vg_lite_frame_flag_t flag
);
```

Returns

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

4.1.7 vg_lite_set_command_buffer_size

Description

This function is optional. If used, call it before `vg_lite_init` if you want to change the command buffer size. *(available from March 2020)*

This function is useful for devices where memory is limited and less than the default. The VGLite Command buffer is set to 32K 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 (
    vg_lite_uint32_t            size
);
```

Parameters

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

Returns

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

4.1.8 vg_lite_set_command_buffer

Description

This function sets a user-defined external memory buffer (physical, 64-byte aligned) as the VGLite command buffer. By default, the VGLite driver allocates a static command buffer internally. Thus, it is not necessary for an application to allocate and set the command buffer. This function is only used for devices where an application needs to allocate the command buffer dynamically. *(from December 2021)*

Syntax

```
vg_lite_error_t vg_lite_set_command_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size

);
```

Parameters

physical	The physical address of a memory buffer. The address must be 64-byte aligned.
size	The size of memory buffer. The size must be 128-byte aligned.

Returns

Returns VG_LITE_SUCCESS if the command buffer set is successful. See vg_lite_error_t enum for other return codes.

4.1.9 vg_lite_set_tess_buffer

Description

This function specifies a memory buffer from an application as the VGLite driver's tessellation buffer. By default, the VGLite driver allocates a static tessellation buffer internally. Thus, it is not necessary for an application to allocate and set the tessellation buffer. This function is only used for devices where the application needs to allocate the tessellation buffer dynamically. *(from December 2021)*

Syntax

```
vg_lite_error_t vg_lite_set_tess_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size

);
```

Parameters

physical	The physical address of a tessellation buffer. The address must be 64-byte aligned.
size	The size of tessellation buffer.

	tessellation buffer size = target buffer's height * 128B.

Returns

Returns VG_LITE_SUCCESS if the tessellation buffer set is successful. See vg_lite_error_t enum for other return codes.

4.1.10 vg_lite_set_memory_pool

Description

This function sets the specific memory pool from which certain type of buffers, VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER, should be allocated. By default, all types of buffers are allocated from VG_LITE_MEMORY_POOL_1. This API must be called before `vg_lite_init()` for setting the VG_LITE_COMMAND_BUFFER or VG_LITE_TESSELLATION_BUFFER memory pools. This API can be called anytime for VG_LITE_RENDER_BUFFER to affect the following `vg_lite_allocate()` calls. *(from December 2023)*

Syntax

```
vg_lite_error_t vg_lite_set_memory_pool (
    vg_lite_buffer_type_t           type,
    vg_lite_memory_pool_t           pool

);
```

Parameters

type	The buffer type (VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER) to be allocated from memory pool.
pool	The memory pool (VG_LITE_MEMORY_POOL_1, VG_LITE_MEMORY_POOl_2) from which the buffer type should be allocated.

Returns

Returns VG_LITE_SUCCESS if the tessellation buffer set is successful. See vg_lite_error_t enum for other return codes.

5 Pixel buffers

5.1 Pixel buffer alignment

The VGLite hardware requires the pixel buffer start address and stride to be properly byte aligned to work correctly. The start address and stride alignment requirement for a pixel buffer depends on the specific pixel format, and gcFEATURE_VG_16PIXELS_ALIGNED value (0/1) in vg_lite_options.h file.

Refer to the Alignment notes Table 2: Image Source Start Address and Stride Alignment Summary later in this document.

5.2 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.

5.3 Internal representation

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

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

Where,

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

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

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

5.4 Pixel buffer enumerations

5.4.1 vg_lite_buffer_format_t enumeration

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

Application can use vg_lite_query_feature API to determine support for some hardware dependent formats. For example, related vg_lite_feature_t enum values include gcFEATURE_BIT_VG_RGBA2_FORMAT and gcFEATURE_BIT_VG_IM_INDEX_FORMAT.

Note: *See Alignment notes Table 2 following the value descriptions for alignment requirements summary for the image formats. (alignment columns refined March and Sept 2023)*

Used in structure: vg_lite_buffer_t.

See also vg_lite_blit, vg_lite_clear, vg_lite_draw.

OpenVG VGImageFormat note: The bits for each color channel are stored within a machine word from MSB to LSB in the order indicated by the pixel format name. This is the opposite of the original VG_LITE_* formats which are ordered from LSB to MSB. Formats with the same organization are listed in the same row as their VG_Lite counterparts.

Original VGLite API image format note: The bits for each color channel are stored within a machine word from LSB to MSB in the order indicated by the pixel format name. This is the opposite of the OPENVG VG_* formats which are ordered from MSB to LSB.

The following codes, as also used in OpenVG 1.1 Specification Table 11, are used for format description:

- A: Alpha channel
- B: Blue color channel
- G: Green color channel
- R: Red color channel
- X: Uninterpreted padding byte or bit
- L: Grayscale
- BW: 1-bit black and white
- L: Linear color space
- s: Non-linear (sRGB) color space
- PRE: Alpha values are premultiplied

vg_lite_buffer_format_t string value	Description					Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_LITE_ABGR8888 VG_sRGBA_8888 VG_sRGBA_8888_PRE VG_lRGBA_8888 VG_lRGBA_8888_PRE	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					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	ABGR8888	R	G	B	A			
VG_LITE_ARGB8888 VG_sBGRA_8888 VG_sBGRA_8888_PRE VG_lBGRA_8888 VG_lBGRA_8888_PRE	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	ARGB8888	B	G	R	A			
VG_LITE_BGRA8888 VG_sARGB_8888 VG_sARGB_8888_PRE VG_lARGB_8888 VG_lARGB_8888_PRE	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	BGRA888	A	R	G	B			
VG_LITE_RGBA8888 VG_sABGR_8888 VG_sABGR_8888_PRE VG_lABGR_8888	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			

vg_lite_buffer_format_t string value	Description					Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_lABGR_8888_PRE	RGBA8888	A	B	G	R			
VG_LITE_BGRX8888 VG_sXRGB_8888 VG_lXRGB_8888	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					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	BGRX8888	X	R	G	B			
VG_LITE_RGBX8888 VG_sXBGR_8888 VG_lXBGR_8888	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	RGBX8888	X	B	G	R			
VG_LITE_XBGR8888 RGBX VG_sRGBX_8888 VG_lRGBX_8888	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	XBGR888	R	G	B	X			
VG_LITE_XRGB8888 VG_sBGRX_8888 VG_lBGRX_8888	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.					Yes	Yes	Start 4B / Stride 64B
		31:24	23:16	15:8	7:0			
	XRGB8888	B	G	R	X			
VG_LITE_ABGR1555 VG_sRGBA_5551	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.					Yes	Yes	Start 4B / Stride 32B
		15:11	10:6	5:1	0:0			
	ABGR1555	R	G	B	A			
VG_LITE_ARGB1555 VG_sBGRA_5551	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.					Yes	Yes	Start 4B / Stride 32B
		15:11	10:6	5:1	0:0			
	ARGB1555	B	G	R	A			
VG_LITE_BGRA5551 VG_sARGB_1555	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.					Yes	Yes	Start 4B / Stride 32B
		15:15	14:0	9:5	4:0			
	BGRA5551	A	R	G	B			
VG_LITE_RGBA5551 VG_sABGR_1555	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.					Yes	Yes	Start 4B / Stride 32B
		15:15	14:0	9:5	4:0			
	RGBA5551	A	B	G	R			

vg_lite_buffer_format_t string value	Description				Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_LITE_BGR565 VG_sRGB_565	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.				Yes	Yes	Start 4B / Stride 32B
		15:11	10:5	4:0			
	BGR565	R	G	B			
VG_LITE_RGB565 VG_sBGR_565	16-bit RGB format with 5 or 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.				Yes	Yes	Start 4B / Stride 32B
		15:11	10:5	4:0			
	RGB565	B	G	R			
VG_LITE_ABGR4444 VG_sRGBA_4444	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				Yes	Yes	Start 4B / Stride 32B
		15:12	11:8	7:4	3:0		
	ABGR4444	R	G	B	A		
VG_LITE_ARGB4444 VG_sBGRA_4444	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.				Yes	Yes	Start 4B / Stride 32B
		15:12	11:8	7:4	3:0		
	ARGB4444	B	G	R	A		
VG_LITE_BGRA4444 VG_sARGB_4444	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.				Yes	Yes	Start 4B / Stride 32B
		15:12	11:8	7:4	3:0		
	BGRA4444	A	R	G	B		
VG_LITE_RGBA4444 VG_sABGR_4444	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.				Yes	Yes	Start 4B / Stride 32B
		15:12	11:8	7:4	3:0		
	RGBA4444	A	B	G	R		
VG_LITE_YUY2 VG_LITE_YUYV	Packed YUV format, 32-bit for 2 pixels. Y0 is in bits 7:0 and V0 is in bits 31:23. (available for Source IMAGE only)				Yes	No	Start 4B / Stride 32B
		31:24	23:16	15:8	7:0		
	YUV2	V0	Y1	U0	Y0		
		V2	Y3	U2	Y2		
VG_LITE_A4 VG_A_4	4-bit alpha format. There are no RGB values.				Yes	No	Start 4B / Stride 8B
		3:0					
	A4	A					
VG_LITE_A8 VG_A_8	8-bit alpha format. There are no RGB values.				Yes	Yes	Start 4B / Stride 16B
		7:0					
	A8	A					
VG_LITE_L8 VG_sL_8	8-bit grayscale format. There are no RGB values.				Yes	Yes	Start 4B / Stride 16B
		7:0					

vg_lite_buffer_format_t string value	Description		Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_lL_8	**L8**	A			

Hardware dependent formats for vg_lite_buffer_format_t	Description					Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_LITE_ABGR2222	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.					Yes	Yes	Start 4B / Stride 16B
		7:6	5:4	3:2	1:0			
	ABGR2222	R	G	B	A			
VG_LITE_ARGB2222	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.					Yes	Yes	Start 4B / Stride 16B
		7:6	5:4	3:2	1:0			
	ARGB2222	B	G	R	A			
VG_LITE_BGRA2222	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.					Yes	Yes	Start 4B / Stride 16B
		7:6	5:4	3:2	1:0			
	BGRA2222	A	R	G	B			
VG_LITE_RGBA2222	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					Yes	Yes	Start 4B / Stride 16B
		7:6	5:4	3:2	1:0			
	RGBA2222	A	B	G	R			
VG_LITE_INDEX_1	1-bit index format.					Yes	No	8B
VG_LITE_INDEX_2	2-bit index format.					Yes	No	both 8B
VG_LITE_INDEX_4	4-bit index format.					Yes	No	both 8B
VG_LITE_INDEX_8	8-bit index format.					Yes	No	both 16B
VG_LITE_NV12_TILED	Supertiled (8x8 pixels), planar YUV format, 96-bit for 4 pixels. Y plane is 32 bits for 4 pixels and is organized in 64 pixel super tiles (8x8 Y); UV plane is 64 bits for 4 pixels. Pixels are organized in super tiles (4x4 UV pairs). (available for Source IMAGE only on supporting hardware)					Yes	No	Y: both 16 Bytes UV: both 8 Bytes
		31:24	23:16	15:8	7:0			
	Y buffer	Y3	Y2	Y1	Y0			
	Y buffer	…						
		31:24	23:16	15:8	7:0			
	UV buffer	V1	U1	V0	U0			
		V3	U3	V2	U2			
VG_LITE_ANV12_TILED	Pixel organization as NV12_TILED but with an Alpha Buffer, also supertiled. (available for Source IMAGE only on supporting hardware)					Yes	No	A, Y: both 16 Bytes

Pixel buffers

Hardware dependent formats for vg_lite_buffer_format_t	Description					Supported as source	Supported as dest	Start address / stride alignment: Bytes
		31:24	23:16	15:8	7:0			UV: both 8 Bytes
	Alpha buffer	A3	A2	A1	A0			
		31:24	23:16	15:8	7:0			
	Y buffer	Y3	Y2	Y1	Y0			
	Y buffer							
		31:24	23:16	15:8	7:0			
	UV buffer	V1	U1	V0	U0			
		V3	U3	V2	U2			
VG_LITE_AYUY2_TILED	Supertiled (8x8) and packed YUV format with separate tiled Alpha Buffer. Y0 is in bits 7:0 and V is in bits 31:23. (available for Source IMAGE only on supporting hardware)					Yes	No	Both 32B
		31:24	23:16	15:8	7:0			
	Alpha buffer	A3	A2	A1	A0			
		31:24	23:16	15:8	7:0			
	YUY2 buffer	V0	Y1	U0	Y0			
		V2	Y3	U2	Y2			
		V4	Y5	U4	Y4			
VG_LITE_RGB888	24-bit RGB format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16.					Yes	Yes	Start 4B / Stride 32B
			23:16	15:8	7:0			
	RGB888		B	G	R			
VG_LITE_BGR888	24-bit RGB format with 8 bits per color channel. Blue is in bits 7:0, green in bits 15:8, red in bits 23:16.					Yes	Yes	Start 4B / Stride 32B
			23:16	15:8	7:0			
	BGR888		R	G	B			
VG_LITE_ARGB8565	24-bit RGBA format with 4 and 5 bits per color channel. Alpha channel is in bit 7:0, red in bits 12:8, green in bits 18:13 and blue in bits 23:19.					Yes	Yes	Start 4B / Stride 32B
		23:19	18:13	12:8	7:0			
	ARGB8565	B	G	R	A			
VG_LITE_BGRA5658	24-bit RGBA format with 4 and 5 bits per color channel. Blue is in bits 4:0, green in bits 10:5, red in bits 15:11,alpha channel is in bit 23:16.					Yes	Yes	Start 4B / Stride 32B
		23:16	15:11	10:5	4:0			
	BGRA5658	A	R	G	B			
VG_LITE_ABGR8565	24-bit RGBA format with 4 and 5 bits per color channel. Alpha channel is in bit 7:0, blue in bits 12:8, green in bits 18:13 and red in bits 23:19.					Yes	Yes	Start 4B / Stride 32B
		23:19	18:13	12:8	7:0			
	ABGR8565	R	G	B	A			

Hardware dependent formats for vg_lite_buffer_format_t	Description	Supported as source	Supported as dest	Start address / stride alignment: Bytes
VG_LITE_RGBA5658	24-bit RGBA format with 4 and 5 bits per color channel. Red is in bits 4:0, green in bits 10:5, blue in bits 15:11 and the alpha channel is in bits 23:16	Yes	Yes	Start 4B / Stride 32B

	23:16	15:11	10:5	4:0
RGBA5658	A	B	G	R

5.4.2 Image buffer alignment requirement

The image (or source) buffer alignment requirement depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defined in the *vg_lite_options.h* file.

Table 2 Image buffer alignment summary

Image format	Bits per pixel	Source tile mode	Start address alignment requirement in Bytes	Stride alignment requirement in Bytes	Buffer height alignment requirement	Supported for destination
VG_LITE_INDEX1	1	linear	8B	2B	1	
		tile	8B	1B	4	
VG_LITE_INDEX2	2	linear	8B	4B	1	
		tile	8B	1B	4	
VG_LITE_INDEX4	4	linear	8B	8B	1	
		tile	8B	2B	4	
VG_LITE_INDEX8	8	linear	16B	16B	1	
		tile	16B	4B	4	
VG_LITE_A4	4	linear	8B	8B	1	
		tile	8B	2B	4	
VG_LITE_A8	8	linear	16B	16B	1	Yes
		tile	16B	4B	4	Yes
VG_LITE_L8	8	linear	16B	16B	1	Yes
		tile	16B	4B	4	Yes
VG_LITE_ARGB2222	8	linear	16B	16B	1	Yes
		tile	16B	4B	4	Yes
VG_LITE_RGB565	16	linear	32B	32B	1	Yes
		tile	32B	8B	4	Yes
VG_LITE_ARGB1555	16	linear	32B	32B	1	Yes
		tile	32B	8B	4	Yes
VG_LITE_ARGB4444	16	linear	32B	32B	1	Yes
		tile	32B	8B	4	Yes
VG_LITE_ARGB8888	32	linear	64B	64B	1	Yes
		tile	64B	16B	4	Yes
VG_LITE_XRGB8888	32	linear	64B	64B	1	Yes

Image format	Bits per pixel	Source tile mode	Start address alignment requirement in Bytes	Stride alignment requirement in Bytes	Buffer height alignment requirement	Supported for destination
		tile	64B	16B	4	Yes
VG_LITE_ARGB8565	24	linear	64B	48B*	1	Yes
		tile	64B	12B*	4	Yes
VG_LITE_RGB888	24	linear	64B	48B*	1	Yes
		tile	64B	12B*	4	
VG_LITE_YUV2_UYVY	16	linear	32B	32B	1	
		tile	32B	8B	4	
VG_LITE_NV12	12	linear	Y: 32B UV: 32B	Y: 32B UV: 32B	1	
VG_LITE_YV12	12	linear	Y: 32B U: 16B V: 16B	Y: 32B U: 16B V: 16B	1	
VG_LITE_NV16	16	linear	Y: 32B UV: 32B	Y: 32B UV: 32B	1	
VG_LITE_YV16	16	linear	Y: 32B U: 16B V: 16B	Y: 32B U: 16B V: 16B	1	
VG_LITE_YV24	24	linear	Y: 32B U: 32B V: 32B	Y: 32B U: 32B V: 32B	1	
VG_LITE_ETC2	8	tile	16B	4B	4	

Note:

1. *The values in Table 2 reflect the alignment requirements of the data in memory. The stride of ARGB8888/ARGB8565 is seen as 4 bytes per pixel when configuring the hardware.*
2. *In tile mode, the stride refers to the byte size of a single row of pixels in the buffer, not four rows.*
3. *When the DECNano function is enabled for the buffer, the total buffer size must meet specific alignment requirements: 64 bytes times the compression rate for ARGB8888 or XRGB8888 formats, and 48 bytes times the compression rate for RGB888 format.*

Additional alignment requirement

1. The buffer starting address must be 16 pixel-byte-size aligned. That is, an 8-bit-per-pixel format buffer must be 16 bytes aligned; a 16-bit-per-pixel format buffer must be 32 bytes aligned; and 24 and 32-bit-per pixel format buffer must be 64 bytes aligned.
2. For linear mode buffer, the buffer stride must be 16 pixel-byte-size aligned.
3. For tile mode buffer, the buffer width and height must be 4 pixel aligned so buffer width and height end at a tile boundary.

5.4.3 Destination buffer alignment requirement

The requirement of destination (or render target) buffer alignment depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defines in the vg_lite_options.h file.

Table 3 Destination buffer alignment summary

Target format	Bits per pixel	Target tile mode	VG tile mode	Start address alignment requirement in Bytes	Stride alignment requirement in Bytes	Buffer height alignment requirement
VG_LITE_A8	8	linear	linear	4B	1B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_L8	8	linear	linear	4B	1B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_ARGB2222	8	linear	linear	4B	1B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_RGB565	16	linear	linear	4B	2B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_ARGB1555	16	linear	linear	4B	2B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_ARGB4444	16	linear	linear	4B	2B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_ARGB8888	32	linear	linear	4B	4B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_XRGB8888	32	linear	linear	4B	4B	1
			tile	64B	64B	4
		tile	linear	64B	64B	4
			tile	64B	16B	4
VG_LITE_ARGB8565	24	linear	linear	64B	3B*	1
			tile	64B	48B*	4

Target format	Bits per pixel	Target tile mode	VG tile mode	Start address alignment requirement in Bytes	Stride alignment requirement in Bytes	Buffer height alignment requirement
		tile	linear	64B	48B*	4
			tile	64B	12B*	4
VG_LITE_RGB888	24	linear	linear	64B	3B*	1
			tile	64B	48B*	4
		tile	linear	64B	48B*	4
			tile	64B	12B*	4

Note:

1. *The values in Table 3 reflect the alignment requirements of pixel data in memory. The stride of ARGB8888/ARGB8565 is seen as 4 bytes per pixel when configuring the hardware.*
2. *In tile mode, the buffer stride refers to the byte size of a row of pixels, not four rows of pixels.*
3. *For PE clear functions, the clear size must align to 48 bytes for RGB888 or ARGB8565 format.*
4. *For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for RGB888 and align to 64 bytes for ARGB8888 or XRGB8888.*
5. *If DECNano function is enabled for the buffer, the target buffer start address needs to align to 64 Bytes.*
6. *If DECNano function is enabled for the buffer, the total buffer size must meet specific alignment requirements: 64 bytes times the compression rate for ARGB8888 or XRGB8888 formats, and 48 bytes times the compression rate for RGB888 format.*

Additional alignment requirement

1. For linear buffer, the starting address must be at least 4-byte aligned. The buffer stride must be at least one pixel size aligned.
2. The buffer starting address must be 64-byte aligned for 24 bit-per-pixel format, or tile mode, or DECNano enabled.
3. The buffer height must be 4-pixel aligned for tile mode buffer.
4. For tile mode buffer, the buffer stride must be 16-byte aligned for non-24bit-per-pixel formats. Therefore, 8-bits-per-pixel format buffer width must be 16 pixel aligned; 16 bits-per-pixel format buffer width must be 8 pixel aligned; and 32 bit-per-pixel format buffer width must be 4 pixel aligned.
5. For tile mode buffer, the buffer stride must be 12 byte aligned for 24 bits-per-pixel formats, i.e., the buffer width must be 4-pixel aligned.
6. For PE clear function, the clear size must align to 48 bytes for 24-bits-per-pixel formats.
7. For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for 24 bits-per-pixel formats and align to 64 Bytes for 32 bits-per-pixel formats.
8. If source buffer tile mode is different from destination buffer tile mode, the starting address must be 64 byte aligned, buffer stride must be 64 Byte aligned for non-24 bits-per-pixel formats, and buffer stride must be 48 byte aligned for 24 bits-per-pixel formats.
9. The VGLite hardware requires the raster image width to be a multiple of 16 pixels for linear gradient and radial gradient operations. This requirement applies to all image formats. Therefore, the user needs to pad an arbitrary image width to a multiple of 16 pixels for VGLite linear gradient and radial gradient APIs

5.4.4 vg_lite_buffer_layout_t enumeration

Specifies the buffer data layout in memory.

Used in structure: vg_lite_buffer.

vg_lite_buffer_layout_t string value	Description
VG_LITE_LINEAR	Linear (scanline) layout.
VG_LITE_TILED	Data is organized in 4x4 pixel tiles. Note: for this layout, the buffer start address and stride need to be 64 byte aligned.

5.4.5 vg_lite_compress_mode_t enumeration

Specifies the DECNano comprssion mode. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_compress_mode_t string value	Description
VG_LITE_DEC_DISABLE	Disable compression.
VG_LITE_DEC_NON_SAMPLE	Compression ratio is 1.6 for ARGB8888, 2.0 for XRGB8888
VG_LITE_DEC_HSAMPLE	Compression ratio is 2.0 for ARGB8888, 2.6 for XRGB8888
VG_LITE_DEC_HV_SAMPLE	Compression ratio is 2.6 for ARGB8888, 4.0 for XRGB8888

5.4.6 vg_lite_gamma_conversion_t enumeration

Specifies the gamma conversion mode. *(from Sept 2022)*

Used in function: vg_lite_set_gamma.

vg_lite_gamma_conversion_t string value	Description
VG_LITE_GAMMA_NO_CONVERSION	Leave color as is.
VG_LITE_GAMMA_LINEAR	Convert from sRGB to linear.
VG_LITE_GAMMA_NON_LINEAR	Convert from linear to sRGB space.

5.4.7 vg_lite_index_endian_t enumeration

Specifies the endian order parsing mode for index formats. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_index_endian_t string value	Description
VG_LITE_INDEX_ENDIAN_LITTLE_ENDIAN	Parse the index pixel from low to high, when using index1, the parsing order is bit0~bit7. when using index2, the parsing order is bit0:1,bit2:3,bit4:5.bit6:7. when using index4, the parsing order is bit0:3,bit4:7.

VG_LITE_INDEX_ENDIAN_BIG_ENDIAN	Parse the index pixel from low to high, when using index1, the parsing order is bit7~bit0. when using index2, the parsing order is bit7:6,bit5:4,bit3:2.bit1:0. when using index4, the parsing order is bit4:7,bit0:3.

5.4.8 vg_lite_image_mode_t enumeration

Specifies how an image is rendered onto a buffer. *(prior to Sept 2022 name was vg_lite_buffer_image_mode_t)*

Used in structure: vg_lite_buffer_t.

vg_lite_image_mode_t string value	Description
VG_LITE_ZERO	
VG_LITE_NORMAL_IMAGE_MODE	Image drawn with blending mode
VG_LITE_MULTIPLY_IMAGE_MODE	Image is multiplied with paint color
VG_LITE_STENCIL_MODE	
VG_LITE_NONE_IMAGE_MODE	Image input is ignored
VG_LITE_RECOLOR_MODE	

5.4.9 vg_lite_map_flag_t enumeration

Specifies whether mapping is for user memory or the DMA buffer. *(from March 2023)*

Used in function: vg_lite_map.

vg_lite_map_flag_t string value	Description
VG_LITE_MAP_USER_MEMORY	Mapping is for user memory.
VG_LITE_MAP_DMABUF	Mapping is for the DMA buffer.

5.4.10 vg_lite_paint_type_t enumeration

Specifies paint type. *(from May 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_paint_type_t string value	Description
VG_LITE_PAINT_ZERO	
VG_LITE_PAINT_COLOR	Color
VG_LITE_PAINT_LINEAR_GRADIENT	Linear Gradient
VG_LITE_PAINT_RADIAL_GRADIENT	Radial Gradient
VG_LITE_PAINT_PATTERN	Pattern

5.4.11 vg_lite_transparency_t enumeration

Specifies the transparency mode for a buffer. *(prior to Sept 2022 name was vg_lite_buffer_transparency_mode_t)*

Used in structure: vg_lite_buffer.

vg_lite_transparency_t string value	Description
VG_LITE_IMAGE_OPAQUE	Opaque image: all image pixels are copied to the VG PE for rasterization
VG_LITE_IMAGE_TRANSPARENT	Transparent image: only the non-transparent image pixels are copied to the VG PE. Note: this mode is only valid when IMAGE_MODE (vg_lite_image_mode_t) is either VG_LITE_NORMAL_IMAGE_MODE or VG_LITE_MULTIPLY_IMAGE_MODE.

5.4.12 vg_lite_swizzle_t enumeration

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

Used in structure: vg_lite_yuvinfo.

vg_lite_swizzle_t string value	Description
VG_LITE_SWIZZLE_UV	U in lower bits, V in upper bits
VG_LITE_SWIZZLE_VU	V in lower bits, U in upper bits

5.4.13 vg_lite_yuv2rgb_t enumeration

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

Used in structure: vg_lite_yuvinfo.

vg_lite_yuv2rgb_t string value	Description
VG_LITE_YUV601	YUV Converting with ITC.BT-601 standard
VG_LITE_YUV709	YUV Converting with ITC.BT-709 standard

5.5 Pixel buffer structures

5.5.1 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_upload_buffer, vg_lite_map, vg_lite_unmap.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect, vg_lite_clear, vg_lite_create_masklayer, vg_lite_fill_masklayer, vg_lite_blend_masklayer, vg_lite_set_masklayer, vg_lite_render_masklayer, vg_lite_destroy_masklayer,

Used in draw functions: vg_lite_draw, vg_lite_draw_pattern, vg_lite_draw_grad, vg_lite_draw_radial_grad.

vg_lite_buffer_t members	**Type**	**Description**
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
tiled	vg_lite_buffer_layout_t	Linear or tiled format for buffer enum
format	vg_lite_buffer_format_t	color format enum
handle	vg_lite_pointer	memory handle
memory	vg_lite_pointer	pointer to the start address of the memory
address	vg_lite_uint32_t	GPU address
yuv	vg_lite_yuvinfo_t	YUV format info struct
image_mode	vg_lite_image_mode_t	Blit image mode enum
transparency_mode	vg_lite_transparency_t	Image transparency mode enum
fc_buffer[3]	vg_lite_fc_buffer_t	Three (3) fast clear buffers, reserved YUV format *(from March 2023)*
compress_mode	vg_lite_compress_mode	Compression mode *(from March 2023)*
index_endian	vg_lite_index_endian_t	Big/Little Endian setting for index formats *(from March 2023)*
paintType	vg_lite_paint_type_t	Paint type enum *(from May 2023)*
fc_enable	vg_lite_int8_t	Enable Image fast clear *(moved from Aug 2023)*
scissor_layer	vg_lite_int8_t	Get paintcolor from different paint type *(from Aug 2023)*
premulitplied	vg_lite_int8_t	The RGB pixel values are alpha-premultipled *(from Aug 2023)*

5.5.2 vg_lite_fc_buffer_t structure

This structure defines the organization of a fast clear buffer. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_fc_buffer_t members	Type	Description
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
handle	vg_lite_pointer	memory handle as allocated by the VGLite kernel
memory	vg_lite_pointer	logical pointer to the start address of the memory for the CPU
address	vg_lite_uint32_t	address to the buffer's memory for the GPU hardware
color	vg_lite_uint32_t	The fast clear color value

5.5.3 vg_lite_yuvinfo_t structure

This structure defines the organization of VGLite YUV data.

Used in structure: vg_lite_buffer_t.

vg_lite_yuvinfo_t members	Type	Description
swizzle	vg_lite_swizzle_t	UV swizzle enum
yuv2rgb	vg_lite_yuv2rgb_t	YUV conversion standard enum
uv_planar	vg_lite_uint32_t	UV (U) planar address for GPU, generated by driver
v_planar	vg_lite_uint32_t	V planar address for GPU, generated by driver
alpha_planar	vg_lite_uint32_t	Alpha planar address for GPU, generated by driver
uv_stride	vg_lite_uint32_t	UV (U) stride in bytes
v_stride	vg_lite_uint32_t	V planar stride in bytes
alpha_stride	vg_lite_uint32_t	Alpha stride in bytes
uv_height	vg_lite_uint32_t	UV (U) height in pixels
v_height	vg_lite_uint32_t	V stride in bytes
uv_memory	vg_lite_pointer	Logical pointer to the UV (U) planar memory
v_memory	vg_lite_pointer	Logical pointer to the V planar memory
uv_handle	vg_lite_pointer	Memory handle of the UV (U) planar, generated by driver
v_handle	vg_lite_pointer	Memory handle of the V planar, generated by driver

5.6 Pixel buffer functions

5.6.1 vg_lite_allocate

Description

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

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

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

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

Syntax

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

Parameters

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

Returns

- VG_LITE_SUCCESS if the allocating contiguous buffer is allocated successfully.
- VG_LITE_OUT_OF_RESOURCES if there is insufficient memory in the host OS heap for the buffer
- VG_LITE_OUT_OF_MEMORY if allocation of a contiguous buffer failed.

5.6.2 vg_lite_free

Description

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

Syntax

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

Parameters

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

Returns

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

5.6.3 vg_lite_upload_buffer

Description

The function uploads the pixel data to a GPU memory buffer object. Note that the format of the data (pixel) to be uploaded must be the same as described in the buffer object. The input data memory buffer should contain enough data to be uploaded to the GPU buffer pointed by the input parameter "buffer". *(replaces deprecated vg_lite_buffer_upload from Sept 2022)*

Note: *Vivante Vector Graphics IP only uses data[0] and stride[0] as it does not support planar YUV formats.*

Syntax

```
vg_lite_error_t vg_lite_upload_buffer (
    vg_lite_buffer_t           *buffer,
    vg_lite_uint8_t            *data[3],
    vg_lite_uint32_t           stride[3]
);
```

Parameters

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

Returns

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

5.6.4 vg_lite_map

Description

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

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

Syntax

```
vg_lite_error_t vg_lite_map (
    vg_lite_buffer_t           *buffer,
    vg_lite_map_flag_t         flag,
    int32_t                    fd
);
```

Parameters

*buffer	Pointer to a buffer structure that was filled in by vg_lite_allocate.
flag	Enum vg_lite_map_flag_t value which specifies whether mapping is for user memory or DMA buffer. *(from March 2023)*
fd	File descriptor for dma_buf if flag is VG_LITE_MAP_DMABUF. Otherwise this parameter is ignored. *(from March 2023)*

Returns

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

5.6.5 vg_lite_unmap

Description

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

Syntax

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

Parameters

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

Returns

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

5.6.6 vg_lite_flush_mapped_buffer

Description

This function flushes the CPU cache for the mapped buffer to make sure the buffer contents are written to GPU memory.

Syntax

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

Parameters

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

Returns

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

5.6.7 vg_lite_set_CLUT

Description

This function sets the Color Lookup Table (CLUT) in context state for index color image. Once the CLUT is set (Not NULL), the image pixel color for index format image rendering is obtained from the Color Lookup Table (CLUT) according to the pixel's color index value.

Note: Available only for IP with Indexed color support.

Syntax

```
      vg_lite_error_t vg_lite_set_CLUT (
          vg_lite_uint32_t               count,
          vg_lite_uint32_t               *colors
      );
```

Parameters

count	This is the count of the colors in the color look up table. For INDEX_1, there can be up to 2 colors in the table; For INDEX_2, there can be up to 4 colors in the table; For INDEX_4, there can be up to 16 colors in the table; For INDEX_8, there can be up to 256 colors in the table.
*colors	The Color Lookup Table (CLUT) pointed by "colors" will be stored in the context and programmed to the command buffer when needed. The CLUT will not take effect until the command buffer is submitted to HW. The color is in ARGB format with A located in the upper bits. *Note: The VGLite driver does not validate the CLUT contents from application.*

Returns:

VG_LITE_SUCCESS since no checking is done.

5.6.8 vg_lite_enable_dither

Description

This function is used to enable the dither function. Dither is turned off by default.

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_DITHER) to determine HW support for dither.

Syntax

```
vg_lite_error_t vg_lite_enable_dither (
);
```

Parameters

None

Returns

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

5.6.9 vg_lite_disable_dither

Description:

This function is used to disable the dither function. Dither is turned off by default.

Syntax

```
vg_lite_error_t vg_lite_disable_dither (
);
```

Parameters:

None

Returns

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

5.6.10 vg_lite_set_gamma

Description

This function sets a gamma value.

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_GAMMA) to determine HW support for gamma.

Syntax

```
vg_lite_error_t vg_lite_set_gamma (
    vg_lite_gamma_conversion_t    gamma_value
);
```

Parameters

gamma_value	Sets a gamma value. See enum vg_lite_gamma_conversion_t.

Returns

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

6 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.*

6.1 Matrix control float parameter type

Name	Typedef	Value
vg_lite_float_t	float	A single precision floating point number
vg_lite_pixel_matrix_t[20]	vg_lite_float_t	Pixel transform matrix m[20] which transforms each pixel as follows:
		`|a'|    |m0  m1  m2  m3  m4 |  |a|`
		`|r'|    |m5  m6  m7  m8  m9 |  |r|`
		`|g'| =  |m10 m11 m12 m13 m14|.|g|`
		`|b'|    |m15 m16 m17 m18 m19|  |b|`
		`|1 |    |0   0   0   0   1  |  |1|`

6.2 Matrix control structures

6.2.1 vg_lite_matrix_t structure

This structure defines a 3x3 float matrix.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in function: vg_lite_render_masklayer.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern, vg_lite_identity, vg_lite_scale, vg_lite_translate.

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

6.2.2 vg_lite_pixel_channel_enable_t structure

This structure provides enable disable flags for hardware pixel channels A,R,G,B.

Used in function: vg_lite_set_pixel_matrix_t.

vg_lite_pixel_channel_enable_t members	Type	Description
enable_a	vg_lite_uint8_t	Enable A channel
enable_b	vg_lite_uint8_t	Enable B channel
enable_g	vg_lite_uint8_t	Enable G channel
enable_r	vg_lite_uint8_t	Enable R channel

6.3 Matrix control functions

6.3.1 vg_lite_identity

Description

This function loads an identity matrix into a matrix variable.

Syntax

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

Parameters

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

Returns

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

6.3.2 vg_lite_set_pixel_matrix

Description

This function sets up a pixel transform matrix m[20] which transforms each pixel as follows:

```
|a'|    |m0  m1  m2  m3  m4 |  |a|
|r'|    |m5  m6  m7  m8  m9 |  |r|
|g'| =  |m10 m11 m12 m13 m14|.|g|
|b'|    |m15 m16 m17 m18 m19|  |b|
|1 |    |0   0   0   0   1  |  |1|
```

The pixel transform for the A, R, G, B channels can be enabled/disabled individually with the channel parameter.

Applications can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_PIXEL_MATRIX) to determine HW support for gaussian blur.

Syntax

```
      vg_lite_error_t vg_lite_set_pixel_matrix (
          vg_lite_pixel_matrix_t           matrix,
          vg_lite_pixel_channel_enable_t   *channel
      );
```

Parameters

matrix	Specifies the vg_lite_pixel_matrix_t pixel transform matrix that will be loaded
*channel	Pointer to the vg_lite_pixel_channel_enable_t structure used to enable/disable individual channels.

Returns

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

6.3.3 vg_lite_rotate

Description

This function rotates a matrix a specified number of degrees.

Syntax

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

Parameters

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

Returns

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

6.3.4 vg_lite_scale

Description

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

Syntax

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

Parameters

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

Returns

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

6.3.5 vg_lite_translate

Description

This function translates a matrix to a new location.

Syntax

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

Parameters

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

Returns

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

7 Blits for compositing and blending

This part of the API performs hardware accelerated blit operations.

Compositing rules describe how two areas are combined to form a single area. Blending rules describes how combining the colors of the overlapping areas are combined. VGLite supports two blending operations and a subset of the Porter-Duff operations [PD84]. The Porter-Duff operators assume the pixels have the alpha associated (pre-multiplied), i.e., pixels are premultiplied prior to the blending operation. Note that GC555, GC355 and some GCNanoUltraV hardware supports alpha premultiply for RGB image, but GCNanoLiteV does not.

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

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

7.1 BLIT enumerations

7.1.1 vg_lite_blend_t enumeration

This enumeration defines the blending modes supported by some VGLite API functions. S and D represent source and destination non-pre-multiplied RGB color channels. Sa and Da represent the source and destination alpha channels. SP and DP represent source and destination alpha-pre-multiplied RGB color channels (SP = S*Sa, DP = D*Da).

Note: VG_LITE_BLEND_*_LVGL modes are supported on all VG cores. On VG cores that do not support gcFEATURE_BIT_VG_LVGL_SUPPORT, the LVGL blend modes are supported by a combination of software and hardware operations. OPENVG_BLEND_* modes can only be supported on GC355 and GC555 cores.

Used in blit functions: vg_lite_blit, vg_lite_blit2, vg_lite_blit_rect. Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_blend_t string values	**Description**
VG_LITE_BLEND_NONE	**S, no bleeding** Non-premultiplied
VG_LITE_BLEND_SRC_OVER	**S + D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_DST_OVER	**S * (1 – Da) + D** Non-premultiplied
VG_LITE_BLEND_SRC_IN	**S * Da** Non-premultiplied
VG_LITE_BLEND_DST_IN	**D * Sa** Non-premultiplied
VG_LITE_BLEND_MULTIPLY	**S * (1 - Da) + D * (1 - Sa) + S * D** Non-premultiplied
VG_LITE_BLEND_SCREEN	**S + D - S * D**

vg_lite_blend_t string values	Description
	Non-premultiplied
VG_LITE_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_ADDITIVE	**S + D** Non-premultiplied
VG_LITE_BLEND_SUBTRACT	**D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_NORMAL_LVGL	**S * Sa + (1 - Sa) * D.** Non-premultiplied *(from March 2023)*
VG_LITE_BLEND_ADDITIVE_LVGL	**(S + D) * Sa + D * (1 - Sa) .** Non-premultiplied *(from March 2023)*
VG_LITE_BLEND_SUBTRACT_LVGL	**(S - D) * Sa + D * (1 - Sa) .** Non-premultiplied *(from March 2023)*
VG_LITE_BLEND_MULTIPLY_LVGL	**(S * D) * Sa + D * (1 - Sa).** Non-premultiplied *(from March 2023)*
OpenVG porter-duff string values	*(from Aug 2023)*
OPENVG_BLEND_NONE	**SP, no blending** Premultiplied
OPENVG_BLEND_SRC_OVER	**(SP + DP * (1 - Sa)) / (Sa + Da * (1 - Sa))** Premultiplied
OPENVG_BLEND_DST_OVER	**(SP * (1 - Da) + DP) / (Sa * (1 - Da) + Da)** Premultiplied
OPENVG_BLEND_SRC_IN	**(SP * Da) / (Sa * Da)** Premultiplied
OPENVG_BLEND_DST_IN	**(DP * Sa) / (Sa * Da)** Premultiplied
OPENVG_BLEND_MULTIPLY	**(SP*DP + SP*(1 - Da) + DP*(1 - Sa)) / (Sa + Da*(1 - Sa)**) Premultiplied
OPENVG_BLEND_SCREEN	**(SP + DP - (SP*DP)) / (Sa + Da*(1 - Sa))** Premultiplied
OPENVG_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_ADDITIVE	**(SP + DP) / (Sa + Da)** Premultiplied

7.1.2 vg_lite_color_t parameter

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

7.1.3 vg_lite_color_transform_t structure

Specifies the pixel color_transform values for scale and bias.

Used in functions: vg_lite_set_color_transform.

vg_lite_color_transform_t members	Type	Description
a_scale	vg_lite_float_t	Scale value for alpha.
a_bias	vg_lite_float_t	Bias value for alpha.
r_scale	vg_lite_float_t	Scale value for red.
r_bias	vg_lite_float_t	Bias value for red.
g_scale	vg_lite_float_t	Scale value for green.
g_bias	vg_lite_float_t	Bias value for green.
b_scale	vg_lite_float_t	Scale value for blue.
b_bias	vg_lite_float_t	Bias value for blue.

7.1.4 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_grad, vg_lite_draw_pattern.

vg_lite_filter_t string values	Description
VG_LITE_FILTER_POINT	Fetch only the nearest image pixel.
VG_LITE_FILTER_LINEAR	Use linear interpolation along horizontal line.
VG_LITE_FILTER_BI_LINEAR	Use a 2x2 box around the image pixel and perform an interpolation.
VG_LITE_FILTER_GAUSSIAN	Perform 3x3 gaussian blur with the convolution for image pixel. *(from March 2023)*

7.1.5 vg_lite_global_alpha_t enumeration

Specifies the global alpha mode in VGLite blit APIs.

Used in blit function: vg_lite_dest_global_alpha.

vg_lite_global_alpha_t string values	Description
VG_LITE_NORMAL	= 0: Use original src/dst alpha value.
VG_LITE_GLOBAL	Use global src/dst alpha value to replace original src/dst alpha value.
VG_LITE_SCALED	Multiply global src/dst alpha value and original src/dst alpha value.

7.1.6 vg_lite_mask_operation_t enumeration

Specifies the mask operation mode in VGLite blit APIs.

Used in functions: vg_lite_blend_masklayer, vg_lite_render_masklayer.

vg_lite_mask_operation_t string values	Description
VG_LITE_CLEAR_MASK	This operation sets all mask values in the region of interest to 0, ignoring the new mask layer.
VG_LITE_FILL_MASK	This operation sets all mask values in the region of interest to 1, ignoring the new mask layer.
VG_LITE_SET_MASK	This operation copies values in the region of interest from the new mask layer, overwriting the previous mask values.
VG_LITE_UNION_MASK	This operation replaces the previous mask in the region of interest by its union with the new mask layer. The resulting values are always greater than or equal to their previous value.
VG_LITE_INTERSECT_MASK	This operation replaces the previous mask in the region of interest by its intersection with the new mask layer. The resulting mask values are always less than or equal to their previous value.
VG_LITE_SUBTRACT_MASK	This operation subtracts the new mask from the previous mask and replaces the previous mask in the region of interest by the resulting mask. The resulting values are always less than or equal to their previous value.

7.1.7 vg_lite_orientation_t enumeration

Specifies the mirror orientation in VGLite blit APIs.

Used in functions: vg_lite_set_mirror.

vg_lite_orientation_t string values	Description
VG_LITE_ORIENTATION_TOP_BOTTOM	Target output orientation is from top to bottom (default).
VG_LITE_ORIENTATION_BOTTOM_TOP	Target output orientation is from bottom to top.

7.1.8 vg_lite_param_type_t enumeration

Specifies the parameter type in VGLite blit APIs.

Used in functions: vg_lite_get_parameter.

vg_lite_param_type_t string values	Description
VG_LITE_GPU_IDLE_STATE	Count must be 1 for GPU idle state TRUE or FALSE
VG_LITE_SCISSOR_RECT	Count must be 4n for x, y, right, bottom

7.2 BLIT structures

7.2.1 vg_lite_buffer_t structure

Defined in the vg_lite_buffer_t structure.

7.2.2 vg_lite_color_key_t structure

A "color key" have two sections, where each section contains R,G,B channels which are noted as high_rgb and low_rgb respectively. *(from April 2022)*

When the enable value is true, the color key specified 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]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Used in structure: vg_lite_color_key4_t.

vg_lite_color_key_t members	Type	Description
enable	vg_lite_uint8_t	When set (true), this color key is enabled.
low_r	vg_lite_uint8_t	The R channel of low_rgb.
low_g	vg_lite_uint8_t	The G channel of low_rgb.
low_b	vg_lite_uint8_t	The B channel of low_rgb.
alpha	vg_lite_uint8_t	The alpha channel to replace the destination pixel alpha channel.
high_r	vg_lite_uint8_t	The R channel of high_rgb.
high_g	vg_lite_uint8_t	The G channel of high_rgb.
high_b	vg_lite_uint8_t	The B channel of high_rgb.

7.2.3 vg_lite_color_key4_t structure

The priority order is color_key_0 > color_key_1 > color_key_2 > color_key_3. *(from April 2022)*

Used in blit function: vg_lite_set_color_key

vg_lite_color_key4_t members	Type	Description
color_key_0		high_rgb_0, low_rgb_0, alpha_0, enable_0
color_key_1		high_rgb_1, low_rgb_1, alpha_1, enable_1
color_key_2		high_rgb_2, low_rgb_2, alpha_2, enable_2
color_key_3		high_rgb_3, low_rgb_3, alpha_3, enable_3

7.2.4 vg_lite_matrix_t structure

Defined in the vg_lite_matrix_t structure.

7.2.5 vg_lite_path_t structure

Definedin the vg_lite_path_t structure.

7.2.6 vg_lite_rectangle_t structure

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

Used in blit function: vg_lite_clear.

vg_lite_rectangle_t members	Type	Description
x	vg_lite_int32_t	X Origin of rectangle, left coordinate in pixels
y	vg_lite_int32_t	Y Origin of rectangle, top coordinate in pixels
width	vg_lite_int32_t	X Width of rectangle in pixels
height	vg_lite_int32_t	Y Height of rectangle in pixels

7.2.7 vg_lite_point_t structure

This structure defines a 2D Point. *(from March 2021)*

Used in structure: vg_lite_point4_t.

vg_lite_point_t members	Type	Description
x	vg_lite_int32_t	X value of coordinate
y	vg_lite_int32_t	Y value of coordinate

7.2.8 vg_lite_point4_t structure

This structure defines four 2D points that form a polygon. The points are defined by structure vg_lite_point_t. *(from March 2021)*

vg_lite_point4_t members	Type	Description
vg_lite_point[4]	vg_lite_int32_t each	a set of four points

7.2.9 vg_lite_float_point_t structure

This structure defines a 2D float point. *(from March 2024)*

Used in structure: vg_lite_float_point4_t.

vg_lite_float_point_t members	Type	Description
X	vg_lite_float_t	X value of coordinate
Y	vg_lite_float_t	Y value of coordinate

7.2.10 vg_lite_float_point4_t structure

This structure defines four 2D float points that form a polygon. The points are defined by structure vg_lite_float_point_t. *(from March 2024)*

Used in blit function: vg_lite_get_transform_matrix.

vg_lite_float_point4_t members	Type	Description
vg_lite_float_point[4]	vg_lite_float_t each	a set of four points

7.3 BLIT functions

7.3.1 vg_lite_blit

Description

This is the blit function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth especially with a rotation matrix. VGLite path rendering can be used to achieve high quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

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

Syntax

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

Parameters

*target	Points to the vg_lite_buffer_t structure which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*source	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of source pixels into the target. If matrix is NULL, an identity matrix is assumed, meaning the source will be copied directly on the target at (0,0) location.
blend	Specifies one of the enum vg_lite_blend_t values for hardware supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the “matrix” parameter is specified with rotation or perspective, and the “blend" parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver will overwrite the application’s setting for the BLIT operation as follows:

	• If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, the transparency mode will always be set to TRANSPARENT. • If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mix color gets multiplied with each source pixel before blending happens. If you don't need a mix color, set the color parameter to 0. *Note:* *This parameter has no effect if the source vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.*
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns

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

7.3.2 vg_lite_blit2

Description

This is the blit function for use with two sources. The blit2 operation is performed using two source buffers and one destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Source0 and Source1 are first blended according to the blend mode with a specific transformation matrix for each image. Source1 is used as the source while Source0 is used as dest and is directly output to the render target buffer.

The specified matrices can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth especially with a rotation matrix. VGLite path rendering can be used to achieve high quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_DOUBLE_IMAGE) to determine HW support for double image.

Note:

- *The vg_lite_blit function can be used for color conversion for Source0 or Source1 before merging sources with vg_lite_blit2.*

Syntax

```
      vg_lite_error_t vg_lite_blit2 (
          vg_lite_buffer_t              *target,
          vg_lite_buffer_t              *source0,
          vg_lite_buffer_t              *source1,
          vg_lite_matrix_t              *matrix0,
          vg_lite_matrix_t              *matrix1,
          vg_lite_blend_t               blend,
          vg_lite_filter_t              filter
     );
```

Parameters

*target	Points to the vg_lite_buffer_t structure which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*source0, *source1	Points to the vg_lite_buffer_t structure for the source0 and source1 buffers. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit functions.
*matrix0, *matrix1	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix0 for the source0 pixels and matrix1 for the source1 pixels. If matrix0 and matrix1 are both NULL, the identity matrix is assumed, meaning the blending result of Source0 and Source1 is copied directly on the target at location(0,0).
blend	Specifies one of the enum vg_lite_blend_t values for hardware supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the “matrix” parameter is specified with rotation or perspective, and the “blend" parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver will overwrite the application’s setting for the BLIT operation as follows: If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, the transparency mode will always be set to TRANSPARENT. If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns

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

7.3.3 vg_lite_blit_rect

Description

This is the blit rectangle function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth especially with a rotation matrix. VGLite path rendering can be used to achieve high quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

- *The blit_rect function can be used with or without the blend function (vg_lite_blend_t).*
- *The blit_rect function can be used with or without specifying any color value (vg_lite_color_t) .*
- *The blit_rect function can be used for color conversion with an identity matrix and appropriate formats specified for the source and destination buffers. In this case do not specify blend mode and color value.*
- *The vg_lite_blit_rect rectangle start origin point is always (0,0) for hardware versions prior to GCNanoLiteV 1311p which do not support a non-zero rectangle origin.*

Syntax

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

Parameters

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

Returns

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

7.3.4 vg_lite_copy_image

Description

This API copied a pixel rectangle with dimension (width, height) from source buffer to destination buffer. The source image pixel (sx + i, sy + j) is copied to the destination image pixel (dx + i, dy + j), for 0 ≤ i < width and 0 ≤ j < height. Pixels whose source or destination lie outside of the bounds of the respective image are ignored. Pixel format conversion is applied as needed.

No pre-multiply, transformation, blending, filtering operations are applied to the pixel copy.

Syntax

```
vg_lite_error_t vg_lite_copy_image(
    vg_lite_buffer_t             *target,
    vg_lite_buffer_t             *source,
    vg_lite_int32_t              sx,
    vg_lite_int32_t              sy,
    vg_lite_int32_t              dx,
    vg_lite_int32_t              dy,
    vg_lite_int32_t              width,
    vg_lite_int32_t              height,
);
```

Parameters

*target	Points to the vg_lite_buffer_t structure which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*source	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
sx,sy	Pixel coordinates of the lower-left corner of pixel rectangle within the source buffer.
dx,dy	Pixel coordinates of the lower-left corner of pixel rectangle within the target buffer.
width	Width of the copied pixel rectangle.
height	Height of the copied pixel rectangle.

Returns

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

7.3.5 vg_lite_get_transform_matrix

Description

This function generates a 3x3 homogenous transform matrix from 4 source coordinates and 4 target coordinates. *(from March 2021)*

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 a set of four 2D points that form a source polygon.
dst	Pointer to a set of four 2D points that form a destination polygon.
mat	Output parameter, pointer to a 3x3 homogenous matrix that transforms the source polygon to a destination polygon.

7.3.6 vg_lite_clear

Description

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

Syntax

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

Parameters

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

7.3.7 vg_lite_set_color_key

Description

This function sets a color key. Color key can be used for blit or for draw pattern operations. *(from April 2022)*

A “color key” has two sections, where each section contains R,G,and B channels which are noted as high_rgb and low_rgb respectively.

When the vg_lite_color_key_t structure value enable is true, the color key specified is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are within range [low_rgb, high_rgb]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Hardware support for color key is not available for GCNanoLiteV. Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_KEY) to determine HW support for color key.

Syntax

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

Parameters

colorkey	A color key as defined by vg_lite_color_key4_t. There are 4 groups of color key states: - color_key_0: high_rgb_0, low_rgb_0, alpha_0, enable_0. - color_key_1: high_rgb_1, low_rgb_1, alpha_1, enable_1. - color_key_2: high_rgb_2, low_rgb_2, alpha_2, enable_2. - color_key_3: high_rgb_3, low_rgb_3, alpha_3, enable_3. The priority order of these states is: color_key_0 > color_key_1 > color_key_2 > color_key_3.

Return

VG_LITE_SUCCESS if successful. Otherwise, VG_LITE_NOT_SUPPORT if color key is not supported in hardware.

7.3.8 vg_lite_gaussian_filter

Description

This function sets 3x3 gaussian blur weighted values to filter an image pixel. *(from March 2023)*

The parameters w0, w1, w2 define a 3x3 gaussian blur weight matrix as:

```
|  w2   w1   w2 |
|  w1   w0   w1 |
|  w2   w1   w2 |
```

The sum of the 9 kernel weights must be 1.0 to avoid convolution overflow (w0 + 4*w1 + 4*w2 = 1.0).

The 3x3 weight matrix applies to a 3x3 pixel block:

```
| pixel[i-1][j-1]     pixel[i][j-1]       pixel[i+1][j-1]|
| pixel[i-1][j]       pixel[i][j]         pixel[i+1][j]  |
| pixel[i-1][j+1]     pixel[i][j+1]       pixel[i+1][j+1]|
```

With the following dot product equation:

```
color[i][j] = w2*pixel[i-1][j-1] + w1*pixel[i][j-1] + w2*pixel[i+1][j-1]
          + w1*pixel[i-1][j]   + w0*pixel[i][j]   + w1*pixel[i+1][j]
             + w2*pixel[i-1][j+1] + w1*pixel[i][j+1] + w2*pixel[i+1][j+1];
```

Applications can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_GAUSSIAN_BLUR) to determine HW support for gaussian blur.

Syntax

```
vg_lite_error_t vg_lite_gaussian_filter (
    vg_lite_float_t               w0
    vg_lite_float_t               w1
    vg_lite_float_t               w2
);
```

Parameters

w0, w1, w2	w0, w1, w2 define a 3x3 gaussian blur weighted matrix as:
	`|  w2   w1   w2 |`
	`|  w1   w0   w1 |`

	`| w2 w1 w2 |`

Return

VG_LITE_SUCCESS if successful. Otherwise, VG_LITE_NOT_SUPPORT if gaussian blur is not supported in hardware.

7.4 Blit/Draw extended functions

The following BLIT or DRAW related functions typically require GC355 or GC555 hardware and are not available for all Vivante Vector Graphics hardware configurations.

Applications can use VGLite API vg_lite_query_feature to determine HW support for the related functionality.

7.4.1 vg_lite_get_parameter

Description

This function returns the selected VGLite / GPU states to application.
(from Aug 2023)

Syntax

```
vg_lite_error_t vg_lite_get_parameter (
    vg_lite_param_type_t          type,
    vg_lite_int32_t               count,
    vg_lite_pointer               params

);
```

Parameters

Type	The parameter type to be queried. (VG_LITE_GPU_IDLE_STATE, VG_SCISSOR_RECT)
count	The number of returned parameters.
*params	The pointer to the array of returned parameters.

7.4.2 vg_lite_set_scissor

Description

This is a legacy scissor API function that can be used to set a single scissor rectangle for the render target. This scissor API is supported by a different hardware mechanism other than the mask layer, and it has better performance than the mask layer scissor function.

This API is not enabled or disabled by the vg_lite_enable_scissor and vg_lite_disable_scissor APIs. The **vg_lite_set_scissor** API calls with a valid scissor rectangle input (x, y, right, bottom), which enables the scissor function by default. The **vg_lite_set_scissor** API call with input parameter (-1, -1, -1, -1) disables the scissor function.

Syntax

```
vg_lite_error_t vg_lite_set_scissor (
    vg_lite_int32_t               x,
    vg_lite_int32_t               y,
    vg_lite_int32_t               right,
    vg_lite_int32_t               bottom
);
```

Parameters

x	X Origin of rectangle, left coordinate in pixels
y	Y Origin of rectangle, top coordinate in pixels
right	X rightmost pixel of rectangle
bottom	Y bottom pixel of rectangle

Returns

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

7.4.3 vg_lite_disable_color_transform

Description

This function is used to disable color transformation. By default, color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_TRANSFORMATION) to determine HW support for color transformation. Support is available with GC355 and GC555.

Syntax

```
      vg_lite_error_t vg_lite_disable_color_transform (
      );
```

Parameters

None

Returns

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

7.4.4 vg_lite_enable_color_transform

Description

This function is used to enable color transformation. By default, color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Syntax

```
      vg_lite_error_t vg_lite_enable_color_transform (
      );
```

Parameters

None

Returns

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

7.4.5 vg_lite_set_color_transform

Description

This function is used to set pixel scale and bias values for color transformation for each pixel channel. *(from August 2022, only for GC355 and GC555 hardware)*

Syntax

```
vg_lite_error_t vg_lite_set_color_transform (
    vg_lite_color_transform_t     *values
);
```

Parameters

*values	Pointer to the color transformation values to set. See enum vg_lite_color_transform_t.

Returns

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

7.4.6 vg_lite_enable_masklayer

Description

This function controls the availability of mask functionality. Mask is turned off by default. *(from August -Sept 2022, requires GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_MASK) to determine HW support for mask. The blit and draw mask functions below require GC555 hardware support. These functions were introduced in August 2022 and syntax or name further refined in September 2022.

Syntax

```
vg_lite_error_t vg_lite_enable_masklayer (
    void
  );
```

Returns

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

7.4.7 vg_lite_disable_masklayer

Description

This function controls the availability of mask functionality. Mask is turned off by default. *(from August -Sept 2022, requires GC555 hardware, prior to Sept 2022 name was vg_lite_disable_mask_layer)*

Syntax

```
vg_lite_error_t vg_lite_disable_masklayer (
    void
```

```
        );
```

Returns

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

7.4.8 vg_lite_create_masklayer

Description

This function creates a mask layer with the specified width and height. The mask format defaults to A8 and the default mask value is 255. *(from August 2022-Sept, requires GC555 hardware)*

Syntax

```
        vg_lite_error_t vg_lite_create_masklayer (
            vg_lite_buffer_t              *masklayer,
            vg_lite_uint32_t              width,
            vg_lite_uint32_t              height
        );
```

Parameters

*masklayer	Points to the address of the buffer of the mask layer to be created.
width	Mask layer width (in pixels).
height	Mask layer height (in pixels).

Returns

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

7.4.9 vg_lite_fill_masklayer

Description

This function sets the values of a given mask layer within a given rectangular region to a given value. The value must be between 0 and 255. *(from August-Sept 2022, requires GC555 hardware)*

Syntax

```
        vg_lite_error_t vg_lite_fill_masklayer (
            vg_lite_buffer_t             *masklayer,
            vg_lite_rectangle_t          *rect,
            vg_lite_uint8_t              value
        );
```

Parameters

*masklayer	Points to the address of the buffer of the mask layer to be filled.
*rect	The rectangle area (x, y, width, height) to be filled with value.
value	The value of the fill area. The value must be between 0 and 255.

Returns

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

7.4.10 vg_lite_blend_masklayer

Description

This function blends the specified area of the source mask layer with the destination mask layer according to an vg_lite_mask_operation_t enumeration value, to create a blended destination mask layer. *(from August-Sept 2022, requires GC555 hardware)*

Syntax

```
vg_lite_error_t vg_lite_blend_masklayer (
    vg_lite_buffer_t              *dst_masklayer,
    vg_lite_buffer_t              *src_masklayer,
    vg_lite_mask_operation        operation,
    vg_lite_rectangle_t           *rect,
);
```

Parameters

*dst_masklayer	Points to the address of the buffer of the destination mask layer.
*src_masklayer	Points to the address of the buffer of the source mask layer.
operation	Blending mode to be applied to each image pixel, as defined by enum vg_lite_mask_operation_t.
*rect	The rectangle area (x, y, width, height) of the blend operation.

Returns

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

7.4.11 vg_lite_set_masklayer

Description:

This function sets the given mask layer to the hardware. *(from August-Sept 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_masklayer (
    vg_lite_buffer_t             *masklayer
);
```

Parameters

*masklayer	Points to the address of the buffer of the mask layer to be set.

Returns

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

7.4.12 vg_lite_render_masklayer

Description

This function draws the mask layer according to the specified path, color and matrix information. *(from August-Sept 2022, requires GC555 hardware)*

Syntax

```
vg_lite_error_t vg_lite_render_masklayer (
    vg_lite_buffer_t                *masklayer,
    vg_lite_mask_operation          operation,
    vg_lite_path_t                  *path,
    vg_lite_fill_t                  fill_rule,
    vg_lite_color_t                 color,
    vg_lite_matrix_t                *matrix
);
```

Parameters

*masklayer	Points to the address of the buffer of the destination mask layer.
operation	Blending mode to be applied to each image pixel, as defined by enum vg_lite_mask_operation_t.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
color	Specifies the color vg_lite_color_t RGBA value to be applied to each pixel drawn by the path.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed, meaning the source will be copied directly on the target at 0,0 location. which is usually a bad idea since the path can be anything.

Returns

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

7.4.13 vg_lite_destroy_masklayer

Description

This function is used to free a mask layer. *(from August-Sept 2022, requires GC555 hardware)*

Syntax

```
vg_lite_error_t vg_lite_destroy_masklayer (
    vg_lite_buffer_t                masklayer
);
```

Parameters

*masklayer	Points to the address of the buffer of the mask layer to be destroyed.

Returns

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

7.4.14 vg_lite_set_mirror

Description

This function is used to control mirror functionality. By default, mirror is turned off and the default output orientation is from top to bottom. *(from August 2022, only for GC555 hardware)*

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_MIRROR) to determine HW support for mirror. Mirror functions require GC555 hardware.

Syntax

```
vg_lite_error_t vg_lite_set_mirror (
    vg_lite_orientation_t         orientation
);
```

Parameters

orientation	The orientation mode as defined by enum vg_lite_orientation_t.

Returns

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if not supported.

7.4.15 vg_lite_source_global_alpha

Description

This function will set image/source global alpha and return a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_GLOBAL_ALPHA) to determine HW support for global alpha. The global alpha BLIT related functions require GCNanoUltraV or GC555 hardware.

Syntax

```
vg_lite_error_t vg_lite_source_global_alpha (
    vg_lite_global_alpha_t        alpha_mode,
    vg_lite_uint8_t               alpha_value
);
```

Parameters

alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The image/source global alpha value to set.

Returns

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

7.4.16 vg_lite_dest_global_alpha

Description

This function will set destination global alpha and return a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Syntax

```
vg_lite_error_t vg_lite_dest_global_alpha (
    vg_lite_global_alpha_t          alpha_mode,
    vg_lite_uint8_t                 alpha_value
);
```

Parameters

alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The destination global alpha value to set.

Returns

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

8 Vector path control

8.1 Vector path enumerations

8.1.1 vg_lite_format_t enumeration

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

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

8.1.2 vg_lite_quality_t enumeration

Specifies the level of hardware assisted anti-aliasing.

Used in structure: vg_lite_path_t.

Used in functions: vg_lite_init_path, vg_lite_init_arc_path.

vg_lite_quality_t string values	Description
VG_LITE_HIGH	High quality: 16x coverage sample anti-aliasing
VG_LITE_UPPER	Upper quality: 8x coverage sample anti-aliasing. Use vg_lite_query_feature to determine availability of 8x CSAA (feature enum value gcFEATURE_BIT_VG_QUALITY_8X. (deprecated from June 2020, available with supported hardware from August 2022)
VG_LITE_MEDIUM	Medium quality: 4x coverage sample anti-aliasing
VG_LITE_LOW	Low quality: no anti-aliasing

8.2 Vector path structures

8.2.1 vg_lite_hw_memory structure

This structure simply records the memory allocation info by kernel.

Used in structure: vg_lite_path_t.

vg_lite_hw_memory_t members	Type	Description
handle	vg_lite_pointer	GPU memory object handle
memory	vg_lite_pointer	Logical memory address
address	vg_lite_uint32_t	GPU memory address
bytes	vg_lite_uint32_t	Size of memory
property	vg_lite_uint32_t	Bit 0 is used for path upload: 0: Disable path data uploading (always embedded into command buffer). 1: Enable auto path data uploading.

8.2.2 vg_lite_path_t structure

This structure describes VGLite path data.

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

- Used in init functions: vg_lite_init_path, vg_lite_init_arc_path, vg_lite_upload_path, vg_lite_clear_path, vg_lite_append_path.
- Used in function: vg_lite_render_masklayer.
- Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_path_t members	Type	Description	
bounding_box[4]	vg_lite_float_t	bounding box for path [0] left [1] top [2] right [3] bottom	
quality	vg_lite_quality_t	enum for quality hint for the path, anti-aliasing level	
format	vg_lite_format_t	enum for coordinate format. The coordinates may have these formats:	
		If vg_lite_format_t	**Path data alignment in array should be**
		VG_LITE_S8	8 bit
		VG_LITE_S16	2 bytes
		VG_LITE_S32	4 bytes
uploaded	vg_lite_hw_memory_t	struct with path data that has been uploaded into GPU addressable memory	
path_length	vg_lite_uint32_t	number of bytes in the path data	
path	vg_lite_pointer	pointer to the physical description of the path	
path_changed	vg_lite_int8_t	0: not changed; 1: changed.	
pdata_internal	vg_lite_int8_t	0: path data memory is allocated by the application; 1: path data memory is allocated by the driver.	
path_type	vg_lite_path_type_t	The draw path type as specified in enum vg_lite_path_type_t. *(added for stroke control, from March 2022)*	
*stroke	vg_lite_stroke_t	As defined by structure vg_lite_stroke_t *(added for stroke control, from March 2022)*	
stroke_path	vg_lite_pointer	Pointer to the physical description of the stroke path. *(added for stroke control, from March 2022)*	
stroke_size	vg_lite_uint32_t	Number of bytes in the stroke path data. *(added for stroke control, from March 2022)*	
stroke_color	vg_lite_color_t	The stroke path fill color. *(from Sept 2022)*	
add_end	vg_lite_int8_t	Flag that add end_path in driver *(from March 2023)*	

Special notes for path objects:

- Endianness has no impact, as it is aligned against the boundaries.
- Multiple contiguous op codes should be packed by the size of the specified data format. E.g., by 2 bytes for VG_LITE_S16 or by 4 bytes for VG_LITE_S32.
  - For example, since opcodes are 8-bits (1 byte), for 16-bit (2 byte) or 32-bit (4 byte) data types:

```
…
<opcode1_that_needs_data>
<align_to_data_size>
<data_for_opcode1>
<opcode2_that_doesnt_need_data>
<opcode3_that_needs_data>
<align_to_data_size>
<data_for_opcode3>
…
```

- Path data in the array should always be 1-, 2, or 4-byte aligned, depending on the format:
  - For example, for 32-bit (4 byte) data types:

```
…
<opcode1_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode1>
<opcode2_that_doesnt_need_data>
<opcode3_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode3>
…
```

8.3 Vector path functions

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

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

8.3.1 vg_lite_get_path_length

Description

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

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

Syntax

```
vg_lite_uint32_t vg_lite_get_path_length (
    vg_lite_uint8_t              *opcode,
    vg_lite_uint32_t             count,
    vg_lite_format_t             format
);
```

Parameters

*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*
count	The opcode count.
format	The coordinate data format. All formats available for vg_lite_format_t are valid formats for this function.

Returns

Returns the command buffer length in bytes.

8.3.2 vg_lite_append_path

Description:

This function assembles the command buffer for the path. The command buffer is allocated by the application and assigned to the path. This function makes the final GPU command buffer for the path based on the input opcodes (cmd) and coordinates (data). Note that the application is responsible for allocating a buffer large enough for the path. *(from Jan 2022, returns a vg_lite_error_t status code)*

Syntax

```
vg_lite_error_t vg_lite_append_path (
    vg_lite_path_t               *path
    vg_lite_uint8_t              *opcode,
    vg_lite_pointer              data,
    vg_lite_uint32_t             seg_count
);
```

Parameters

*path	Pointer to the vg_lite_path_t structure with the path definition.
*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*

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

Returns

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

8.3.3 vg_lite_init_path

Description

This function initializes a path definition with specified values. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

Syntax

```
vg_lite_error_t vg_lite_init_path (
    vg_lite_path_t              *path,
    vg_lite_format_t            format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

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

Returns

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

8.3.4 vg_lite_init_arc_path

Description:

This function initializes an arc path definition with specified values. *(from February 2021)*

Syntax:

```
vg_lite_error_t vg_lite_init_arc_path (
    vg_lite_path_t              *path,
    vg_lite_format_t            format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
format	The coordinate data format. The vg_lite_format_t enum value should be FP32.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
length	The length of the path data (in bytes).
*data	Pointer to path data.
min_x min_y max_x max_y	Minimum and maximum x and y values specify the bounding box of the path.

Returns:

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

8.3.5 vg_lite_upload_path

Description

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

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

Syntax

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

Parameters

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

Returns

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

8.3.6 vg_lite_clear_path

Description:

This function will clear and reset path member values. If the path has been uploaded, it frees the GPU memory allocated when uploading the path. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

Syntax:

```
Vg_lite_error_t vg_lite_clear_path (
    vg_lite_path_t                 *path
);
```

Parameters:

***path**	Pointer to the vg_lite_path_t path definition to be cleared.

Returns:

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

8.4 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 4 Vector path data opcodes

Opcode	Arguments	Description
0x00	None	**VLC_OP_END**. Finish tessellation. Close any open path.
0x01	None	**VLC_OP_CLOSE**. For VGLite driver, internal use only. Application should not use this OP directly.
0x02	(x, y)	**VLC_OP_MOVE**. Move to the given vertex. Close any open path. $start_x = x$ $start_y = y$
0x03	(Δx, Δy)	**VLC_OP_MOVE_REL**. Move to the given relative point. Close any open path. $start_x = start_x + \Delta x$ $start_y = start_y + \Delta y$
0x04	(x, y)	**VLC_OP_LINE**. Draw a line to the given point. $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x05	(Δx, Δy)	**VLC_OP_LINE_REL**. Draw a line to the given relative point. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x06	(cx, cy) (x, y)	**VLC_OP_QUAD**. Draw a quadratic Bezier curve to the given end point using the specified control point. $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x07	(Δcx, Δcy) (Δx, Δy)	**VLC_OP_QUAD_REL**. Draw a quadratic Bezier curve to the given relative end point using the specified relative control point. $cx = start_x + \Delta cx$ $cy = start_y + \Delta cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$

Opcode	Arguments	Description
0x08	(cx1, cy1) (cx2, cy2) (x, y)	**VLC_OP_CUBIC**. Draw a cubic Bezier curve to the given end point using the specified control points. $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$
0x09	(Δcx1, Δcy1) (Δcx2, Δcy2) (Δx, Δy)	**VLC_OP_CUBIC_REL**. Draw a cubic Bezier curve to the given relative end point using the specified relative control points. $cx_1 = start_x + \Delta cx_1$ $cy_1 = start_y + \Delta cy_1$ $cx_2 = start_x + \Delta cx_2$ $cy_2 = start_y + \Delta cy_2$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$
0x0A	None	**VLC_OP_BREAK.** Indicates 64-bit path data (including the opcode) is a no-op.
0x0B	(x)	**VLC_OP_HLINE.** Draw a horizontal line to the given point. $Line(start_x, start_y, x, start_y)$ $start_x = x$
0x0C	(Δx)	**VLC_OP_HLINE_REL**. Draw a horizontal line to the given relative point. $x = start_x + \Delta x$ $Line(start_x, start_y, x, start_y)$ $start_x = x$
0x0D	(y)	**VLC_OP_VLINE.** Draw a vertical line to the given point. $Line(start_x, start_y, start_x, y)$ $start_y = y$
0x0E	(Δy)	**VLC_OP_VLINE_REL.** Draw a vertical line to the given relative point. $y = start_y + \Delta y$ $Line(start_x, start_y, start_x, y)$ $start_y = y$

Opcode	Arguments	Description
0x0F	(x,y)	**VLC_OP_SQUAD.** Draw a smooth quadratic Bezier curve to the given end point. The curve starts at the current vertex and the control point is computed as twice the current vertex minus the current control point. $cx = 2 * start_x - cx$ $cy = 2 * start_y - cy$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x10	(Δx,Δy)	**VLC_OP_SQUAD_REL.** Draw a smooth quadratic Bezier curve to the given relative end point. The curve starts at the current vertex and the control point is computed as twice the current vertex minus the current control point. $cx = 2 * start_x - cx$ $cy = 2 * start_y - cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x11	(cx_2,cy_2) (x,y)	**VLC_OP_SCUBIC.** Draw a smooth cubic Bezier curve to the given end point. The curve starts at the current vertex and the first control point (cx_1, cy_1) is computed as twice the current vertex minus the current control point. The second control point (cx_2, cy_2) is specified in arguments. $cx = 2 * start_x - cx$ $cy = 2 * start_y - cy$ $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$
0x12	(Δcx_2, Δcy_2) (Δx,Δy)	**VLC_OP_SCUBIC_REL.** Draw a smooth cubic Bezier curve to the given relative end point. The curve starts at the current vertex and the first control point (cx_1, cy_1) is computed as twice the current vertex minus the current control point. The second control point (cx_2, cy_2) is specified in arguments. $cx_2 = start_x + \Delta cx_2$ $cy_2 = start_y + \Delta cy_2$ $cx = 2 * start_x - cx$ $cy = 2 * start_y - cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$

Opcode	Arguments	Description
		$Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$ $start_y = y$
0x13	(rh, rv, rot, x, y)	**VLC_OP_SCCWARC**. Draw a small CCW Arc to the given end point using the specified radius and rotation angle. $SCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x14	(rh,rv,rot,x,y)	**VLC_OP_SCCWARC_REL**. Draw a small CCW Arc to the given relative end point using the specified radius and rotation angle. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $SCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x15	(rh,rv,rot,x,y)	**VLC_OP_SCWARC**. Draw a small CW Arc to the given end point using the specified radius and rotation angle. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $SCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x16	(rh,rv,rot,x,y)	**VLC_OP_SCWARC_REL**. Draw a small CW Arc to the given relative end point using the specified radius and rotation angle. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $SCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x17	(rh,rv,rot,x,y)	**VLC_OP_LCCWARC**. Draw a large CCW Arc to the given end point using the specified radius and rotation angle. $LCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x18	(rh,rv,rot,x,y)	**VLC_OP_LCCWARC_REL**. Draw a large CCW Arc to the given relative end point using the specified radius and rotation angle. $x = start_x + \Delta x$

Opcode	Arguments	Description
		$y = start_y + \Delta y$ $LCCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x19	(rh,rv,rot,x,y)	**VLC_OP_LCWARC**. Draw a large CW Arc to the given end point using the specified radius and rotation angle. $LCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$
0x1A	(rh,rv,rot,x,y)	**VLC_OP_LCWARC_REL**. Draw a large CW Arc to the given relative end point using the specified radius and rotation angle. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $LCWARC(rh, rv, rot, x, y)$ $start_x = x$ $start_y = y$

9 Vector-based draw operations

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

9.1 Draw and gradient enumerations

9.1.1 vg_lite_blend_t enumeration

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

9.1.2 vg_lite_color_t parameter

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

LINK to vg_lite_color_t color parameter description.

9.1.3 vg_lite_fill_t enumeration

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

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

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

Used in function: vg_lite_render_masklayer.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

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

9.1.4 vg_lite_filter_t enumeration

Defined in the vg_lite_filter_t enumeration.

9.1.5 vg_lite_gradient_spreadmode_t enumeration

vg_lite_gradient_spreadmode_t enum is defined to match OpenVG enum VGColorRampSpreadMode *(from March 2023, replaces vg_lite_radial_gradient_spreadmode, requires GC355/GC555 hardware)*

The application may only define stops with offsets between 0 and 1. Spread modes define how the given set of stops are repeated or extended in order to define interpolated color values for arbitrary input values outside the [0,1] range.

Used in structure: vg_lite_radial_gradient_t.

vg_lite_gradient_spreadmode_t string values	Description
VG_LITE_GRADIENT_SPREAD_FILL	The current fill color is used for all stop values less than 0 or greater than 1, respectively.
VG_LITE_GRADIENT_SPREAD_PAD	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1, respectively.
VG_LITE_GRADIENT_SPREAD_REPEAT	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_GRADIENT_SPREAD_REFLECT	Color values defined between 0 and 1 are repeated indefinitely in both directions, but with alternate copies of the range reversed.

9.1.6 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_grad, vg_lite_draw_pattern.

vg_lite_pattern_mode_t string values	Description
VG_LITE_PATTERN_COLOR	Pixels outside the bounds of the source image should be taken as the color
VG_LITE_PATTERN_PAD	Pixels outside the bounds of the source image should be taken as having the same color as the closest edge pixel. The color of the pattern border is expanded to fill the region outside the pattern.
VG_LITE_PATTERN_REPEAT	Pixels outside the bounds of the source image should be repeated indefinitely in all directions. *(from March 2023)*
VG_LITE_PATTERN_REFLECT	Pixels outside the bounds of the source image should be reflected indefinitely in all directions. *(from March 2023)*

9.1.7 vg_lite_radial_gradient_spreadmode_t enumeration

(Deprecated March 2023) use vg_lite_gradient_spreadmode_t. Defines the radial gradient padding mode. *(from Nov 2020, requires GC355 hardware)*

Used in structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_spreadmode_t string values	Description
VG_LITE_RADIAL_GRADIENT_SPREAD_FILL = 0	The current fill color is used for all stop values less than 0 or greater than 1, respectively.
VG_LITE_RADIAL_GRADIENT_SPREAD_PAD	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1, respectively.
VG_LITE_RADIAL_GRADIENT_SPREAD_REPEAT	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_RADIAL_GRADIENT_SPREAD_REFLECT	Color values defined between 0 and 1 are repeated indefinitely in both directions, but with alternate copies of the range reversed.

9.2 Draw and gradient structures

9.2.1 vg_lite_buffer_t structure

Defined in the vg_lite_buffer_t structure section.

9.2.2 vg_lite_color_ramp_t Structure

This structure defines the stops for the radial gradient. The five parameters provide the offset and color for the stop. Each stop is defined by a set of floating point values which specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255]. *(from November 2020, requires GC355 hardware)*

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

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_color_ramp_t members	**Type**	**Description**
stop	vg_lite_float_t	Offset value for the color stop
red	vg_lite_float_t	Red color channel value for the color stop
green	vg_lite_float_t	Green color channel value for the color stop
blue	vg_lite_float_t	Blue color channel value for the color stop
alpha	vg_lite_float_t	Alpha color channel value for the color stop

9.2.3 vg_lite_linear_gradient_t structure

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

vg_lite_linear_gradient_t constants	**Type**	**Description**
VLC_MAX_GRADIENT_STOPS	vg_lite_int32_t	Constant. Maximum number of gradient colors = 16.
vg_lite_linear_gradient_t Members	Type	Description
colors[VLC_MAX_GRADIENT_STOPS]	vg_lite_uint32_t	Color array for the gradient
count	vg_lite_uint32_t	Number of colors
stops[VLC_MAX_GRADIENT_STOPS]	vg_lite_uint32_t	Number of color stops, from 0 to 255
matrix	vg_lite_matrix_t	Struct for the matrix to transform the gradient color ramp
image	vg_lite_buffer_t	Image object struct to represent the color ramp

9.2.4 vg_lite_ext_linear_gradient structure

This structure defines the organization of the extended parameters possible for a linear gradient. *(from April 2022)*

Used in functions: vg_lite_draw_linear_grad.

vg_lite_ext_linear_gradient_t members	Type	Description
count	vg_lite_uint32_t	Count of colors, up to 256.
matrix	vg_lite_matrix_t	The matrix to transform the gradient.
image	vg_lite_buffer_t	The image for rendering as gradient pattern.
linear_grad	vg_lite_linear_gradient_parameter_t	Linear gradient parameters. Includes center point, focal point and radius.
ramp_length	vg_lite_uint32_t	Color ramp length for gradient paints provided to the driver
color_ramp[VLC_MAX_COLOR_RAMP_STOPS]	vg_lite_color_ramp_t	Color ramp parameter for gradient paints provided to the driver
converted_length	vg_lite_uint32_t	Converted internal color ramp length.
converted_ramp[VLC_MAX_COLOR_RAMP_STOPS+2]	vg_lite_color_ramp_t	Converted internal color ramp.
pre-multiplied	vg_lite_uint8_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_gradient_spreadmode_t	The spread mode that is applied to the pixels out of the image after transformed.

9.2.5 vg_lite_linear_gradient_parameter structure

This structure defines radial direction for a linear gradient. *(from April 2022)*

Line0 connects point (X0, Y0) to point (X1, Y1) and represents the radial direction of the linear gradient.

Line1 is a line perpendicular to line0 which passes through point (X0, Y0).

Line2 is a line perpendicular to line0 which passes through point (X1, Y1)

The linear gradient paint is applied at the intersection of the path fill area and the plane starting from line 1 and ending at line 2.

Used in structure: vg_lite_ext_linear_gradient.

Used in functions: vg_lite_set_linear_grad.

vg_lite_linear_gradient_parameter_t members	Type	Description
X0	vg_lite_float_t	X origin of linear gradient radial direction.
Y0	vg_lite_float_t	Y origin of linear gradient radial direction.
X1	vg_lite_float_t	X end point of linear gradient radial direction.
Y1	vg_lite_float_t	Y end point of linear gradient radial direction.

9.2.6 vg_lite_matrix_t structure

Defined in the vg_lite_matrix_t structure section.

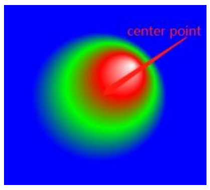
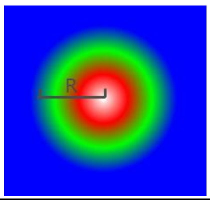
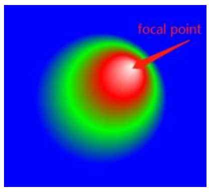
9.2.7 vg_lite_path_t structure

Defined in the vg_lite_path_t structure section.

9.2.8 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. (from November 2020, requires GC355 or GC555 hardware)

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_parameter_t Members	**Type**	**Description** *(member order updated from August 2021)*
cx	vg_lite_float_t	Coordinates x and y of the gradient color center point.  Center point refers to the center of the gradient color.
cy	vg_lite_float_t	
r	vg_lite_float_t	Radius of the gradient. 
fx	vg_lite_float_t	Coordinates x and y of the gradient color focal point  Focal point refers to the center of the gradient color.
fy	vg_lite_float_t	

9.2.9 vg_lite_radial_gradient_t structure

This structure defines the application of the radial gradient to fill a path. *(from November 2020, requires GC355 or GC555 hardware).*

Used in radial gradient functions: vg_lite_draw_grad, vg_lite_set_radial_grad, vg_lite_update_radial_grad, vg_lite_get_radial_grad, vg_lite_clear_radial_grad

vg_lite_radial_gradient_t members	**Type**	**Description**
count	vg_lite_uint32_t	Count of colors, up to 256
matrix	vg_lite_matrix_t	Structure which specifies the transform matrix for the gradient
image	vg_lite_buffer_t	Structure which specifies the image for rendering as a gradient pattern
radial_grad	vg_lite_radial_gradient_parameter_t	Structure which specifies the location of the gradient's center point (cx, cy), focal point(fx, fy) and radius(r)
ramp_length	vg_lite_uint32_t	Color ramp parameters for gradient paints provided to the driver
color_ramp[VLC_MAX_COLOR_RAMP_STOPS]	vg_lite_color_ramp_t	Structure which specifies the color ramp.
converted_length	vg_lite_uint32_t	Converted internal color ramp
converted_ramp[VLC_MAX_COLOR_RAMP_STOPS+2]	vg_lite_color_ramp_t	Structure which specifies the Internal color ramp.
pre_multiplied	vg_lite_uint32_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_gradient_spreadmode_t	Enum which specifies the tiling mode that is applied to the pixels out of the image after transformation.

9.3 Draw functions

9.3.1 vg_lite_draw

Description

Performs a hardware accelerated 2D vector draw operation.

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

Note:

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

Syntax

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

Parameters

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

Returns:

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

9.3.2 vg_lite_draw_grad

Description

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

Syntax

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

Parameters

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

Returns:

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

9.3.3 vg_lite_draw_radial_grad

Description

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

Application can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_RADIAL_GRADIENT) to determine HW support for radial gradient.

Syntax

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

Parameters

*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw for the linear gradient. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path_matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed which is usually a bad idea since the path can be anything.
*grad	Pointer to the vg_lite_radial_gradient_t structure which contains the values to be used to fill the path. Note: grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE.
paint_color	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL, which set by function vg_lite_set_radial_grad. When pixels are out of the image after transformation, this paint_color is applied to them. See also enum vg_lite_radial_gradient_spreadmode_t.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

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

9.3.4 vg_lite_draw_pattern

Description

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

Syntax

```
    vg_lite_error_t vg_lite_draw_pattern (
        vg_lite_buffer_t            *target,
        vg_lite_path_t              *path,
        vg_lite_fill_t              fill_rule,
        vg_lite_matrix_t            *path_matrix,
        vg_lite_buffer_t            *pattern_image,
        vg_lite_matrix_t            *pattern_matrix,
        vg_lite_blend_t             blend,
        vg_lite_pattern_mode_t      pattern_mode,
        vg_lite_color_t             pattern_color,
        vg_lite_color_t             color,
        vg_lite_filter_t            filter
    );
```

Parameters

*target	Pointer to the vg_lite_buffer_t structure that defines the path to draw.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path_matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed, which is usually a bad idea since the path can be anything.
*pattern_image	Pointer to the vg_lite_buffer_t structure that describes the image pattern. Note: pattern_image->image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE in this API.
*pattern_matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target. If matrix is NULL, an identity matrix is assumed, meaning the source will be copied directly onto the target at 0,0 location.
blend	Specifies one of the vg_lite_blend_t enum values for hardware supported blend modes to be applied to each drawn pixel in the image. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
pattern_mode	Specifies the vg_lite_pattern_mode_t value which defines how the region outside the image pattern is to be filled.
pattern_color	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied to the fill outside the image pattern area when the pattern_mode value is VG_LITE_PATTERN_COLOR. *(from Dec 2019, type now vg_lite_color_t, previously was uint32_t)*

color	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied as a mix color. If non-zero, the mix color value gets multiplied with each source pixel before blending happens. If a mix color is not needed, set the color parameter to 0. *(from May 2023)*
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

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

9.4 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.

Note: *VGLite supports linear color gradients for GCNanoLiteV and GCNanoUltraV. Both linear and radial gradients are supported with GC355 and GC555.*

9.4.1 vg_lite_init_grad

Description

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

Syntax

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

Parameters

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

Returns:

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

9.4.2 vg_lite_clear_grad

Description

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

Syntax

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

Parameters

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

Returns:

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

9.4.3 vg_lite_set_grad

Description

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

Note: `vg_lite_set_grad` *API adopts the following rules to set the default gradient colors if the input parameters are incomplete or invalid.*

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

Syntax

```
        vg_lite_error_t vg_lite_set_grad (
            vg_lite_linear_gradient_t   *grad,
            vg_lite_uint32_t            count,
            vg_lite_uint32_t            *colors,
            vg_lite_uint32_t            *stops
        );
```

Parameters

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

Returns

Always returns VG_LITE_SUCCESS.

9.4.4 vg_lite_get_grad_matrix

Description

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

Syntax

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

Parameters

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

Returns:

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

9.4.5 vg_lite_update_grad

Description

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

Syntax

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

Parameters

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

Returns:

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

9.5 Linear gradient extended functions

The following functions are available only with IP which includes hardware support for extended linear gradient capabilities, such as GC355 and GC555. These functions are not available with GCNanoLiteV, GCNanoUltraV or GCNanoV.

Applications can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT) to determine HW support for linear gradient.

9.5.1 vg_lite_set_linear_grad

Description

This function is used to set the values which define the linear gradient. (from April 2022)

Syntax

```
    vg_lite_error_t vg_lite_set_linear_grad (
        vg_lite_ext_linear_gradient_t         *grad,
        vg_lite_uint32_t                      count,
        vg_lite_color_ramp_t                  *color_ramp,
        vg_lite_linear_gradient_parameter_t   grad_param,
        vg_lite_radial_gradient_spreadmode_t  spread_mode,
        vg_lite_uint8_t                       pre_mult
    );
```

Parameters

*grad	Pointer to the vg_lite_ext_linear_gradient_t structure which is to be set.
count	Count of the colors in the gradient. The maxmum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is currently 256.
*color_ramp	This is the stop for the linear gradient. The number of parameters is 5, and gives the offset and color of the stop. Each stop is defined by a floating-point offset 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. And the range of all parameters in it is [0,1].
grad_param	Gradient parameters as specified in structure vg_lite_linear_gradient_parameter_t.
spread_mode	The tiling mode applied to the pixels out of the paint after transformation. Uses the same spread mode enumeration types as radial gradient. See vg_lite_radial_gradient_spreadmode_t enum.
pre_mult	This parameter controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form.

Returns

Returns VG_LITE_INVALID_ARGUMENTS to indicate the parameters are wrong.

9.5.2 vg_lite_get_linear_grad_matrix

Description

This function returns a pointer to an extended linear gradient object's matrix. *(from March 2023)*

Syntax

```
vg_lite_matrix_t* vg_lite_get_linear_grad_matrix (
    vg_lite_ext_linear_gradient_t   *grad,
);
```

Parameters

*grad	Pointer to the vg_lite_ext_linear_gradient_t structure

Returns

Returns a pointer to a vg_lite_matrix_t for the specified extended linear gradient.

9.5.3 vg_lite_draw_linear_grad

Description

This function is used to fill a path with a linear gradient according to specified fill rules. The specified path will be transformed according to the selected matrix and filled with the transformed linear gradient. (from April 2022)

Syntax

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

Parameters

*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data which describes the path to draw for the linear gradient. Refer to the section on Vector Path Data Opcodes in this document for opcode detail.
fill_rule	Specifies the vg_lite_fill_t enum value for the fill rule for the path.

*path_matrix	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If matrix is NULL, an identity matrix is assumed which is usually a bad idea since the path can be anything.
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure which contains the values to be used to fill the path. Note: grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE.
paint_color	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL, which set by function vg_lite_set_linear_grad. When pixels are out of the image after transformation, this paint_color is applied to them. See also enum vg_lite_radial_gradient_spreadmode_t.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

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

9.5.4 vg_lite_update_linear_grad

Description

This function is used to update or generate the corresponding image object to render. *(from April 2022)*

The vg_lite_ext_linear_gradient_t object has an image buffer which is used to render the linear gradient paint. The image buffer will be created/updated according to the specified grad parameters.

Syntax

```
vg_lite_error_t vg_lite_update_linear_grad (
    vg_lite_ext_linear_gradient_t   *grad,
);
```

Parameters

*grad	Pointer to the vg_lite_ext_linear_gradient_t structure which is to be updated or created.

Returns:

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

9.5.5 vg_lite_clear_linear_grad

Description

This function is used to clear the linear gradient object. This will reset the grad members and free the image buffer's memory. (from April 2022)

Syntax

```
vg_lite_error_t vg_lite_clear_linear_grad (
    vg_lite_ext_linear_gradient_t    *grad,
);
```

Parameters

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

Returns:

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

9.6 Radial gradient functions

The following functions are available only with IP which supports radial gradients, such as GC355 and GC555. These functions are not available with GCNanoLiteV, or GCNanoUltraV or GCNanoV. Note: There is no init function required for radial gradients. Buffer initialization is done through the vg_lite_update_radial_grad function. *(from Nov 2020, requires GC355 or GC555 hardware)*

9.6.1 vg_lite_set_radial_grad

Description

This function is used to set the values for the radial linear gradient definition. (from November 2020, requires GC355 or GC555 hardware.

Syntax

```
vg_lite_error_t vg_lite_set_radial_grad (
    vg_lite_radial_gradient_t             *grad,
    vg_lite_uint32_t                      count,
    vg_lite_color_ramp_t                  *color_ramp,
    vg_lite_radial_gradient_parameter_t   grad_param,
    vg_lite_radial_gradient_spreadmode_t  spread_mode,
    vg_lite_uint8_t                       pre_mult
);
```

Parameters

*grad	Pointer to the vg_lite_radial_gradient_t structure for the radial gradient which will be set
count	This is the count of the color stops in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is currently 256.

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

Returns

Returns VG_LITE_INVALID_ARGUMENTS to indicate the parameters are wrong.

9.6.2 vg_lite_update_radial_grad

Description

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_radial_gradient_t object has an image buffer which is used to render the gradient pattern. The image buffer will be created or updated with the corresponding gradient parameters. *(from November 2020, requires GC355 or GC555 hardware)*

Syntax

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

Parameters

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

Returns:

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

9.6.3 vg_lite_get_radial_grad_matrix

Description

This function is used to get a pointer to the radial gradient object's transformation matrix. This allows an application to manipulate the matrix to facilitate correct rendering of the gradient path. *(from Nov 2020, requires GC355 or GC555 hardware)*

Syntax

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

Parameters

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

Returns:

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

9.6.4 vg_lite_clear_radial_grad

Description

This function is used to clear the values of a radial gradient object and free the image buffer's memory. *(from Nov 2020, requires GC355 or GC555 hardware)*

Syntax

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

Parameters

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

Returns:

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

10 Stroke operations

This part of the API performs stroke operations. *(from March 2022)*

10.1 Stroke enumerations

10.1.1 vg_lite_cap_style_t enumeration

Defines the style of cap at the end of a stroke. *(from March 2022)*

- Used in structure: vg_lite_stroke_t

Used in function: vg_lite_set_stroke

vg_lite_cap_style_t string values	Description
VG_LITE_CAP_BUTT	The Butt end cap style terminates each segment with a line perpendicular to the tangent at each endpoint.
VG_LITE_CAP_ROUND	The Round end cap style appends a semicircle with a diameter equal to the line width centered around each endpoint.
VG_LITE_CAP_SQUARE	The Square 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.

10.1.2 vg_lite_path_type_t enumeration

Defines the type of draw path. *(from March 2022)*

- Used in structure: vg_lite_path_t, vg_lite_stroke_t
- Used in function: vg_lite_set_path_type

vg_lite_path_type_t string values	Description
VG_LITE_DRAW_FILL_PATH	Draw path is fill.
VG_LITE_DRAW_STROKE_PATH	Draw path is stroke.
VG_LITE_DRAW_FILL_STROKE_PATH	Draw path is both fill and stroke.

10.1.3 vg_lite_join_style_t enumeration

Defines the type of styles available for line joints. *(from March 2022)*

- Used in structure: vg_lite_stroke_t.
- Used in function: vg_lite_set_stroke.

vg_lite_join_style_t string values	**Description**
VG_LITE_JOIN_MITER	The Miter 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 “fattened” lines and a fourth vertex at the extrapolated intersection point of the outer perimeters of the two “fattened” lines.
VG_LITE_JOIN_ROUND	The Round 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.
VG_LITE_JOIN_BEVEL	The Bevel join style appends a triangle with two vertices at the outer endpoints of the two "fattened” lines and a third vertex at the intersection point of the two original lines.

10.2 Stroke structures

10.2.1 vg_lite_path_t structure

Defined in the vg_lite_path_t structure. *(additional members added for stroke from March 2022)*

10.2.2 vg_lite_path_list_t structure

The structure vg_lite_path_list_ptr points to a vg_lite_path_list structure which provides divided path data according to MOVE/MOVE_REL. *(from Aug 2023)*

- Used (vg_lite_path_list_ptr) in structures: vg_lite_stroke_t.

vg_lite_path_list_t members	**Type**	**Description**
path_points	vg_lite_path_point_ptr	
path_end	vg_lite_path_point_ptr	
point_count	vg_lite_uint32_t	
next	vg_lite_path_list_ptr	
closed	vg_lite_uint8_t	

10.2.3 vg_lite_path_point_t structure

The structure vg_lite_path_point_ptr points to a vg_lite_path_point structure which provides path detail. *(from March 2022)*

- Used (vg_lite_path_point_ptr) in structures: vg_lite_path_point_t, vg_lite_stroke_conversion. vg_lite_sub_path_t.

vg_lite_path_point_t members	**Type**	**Description**
x	vg_lite_float_t	X coordinate
y	vg_lite_float_t	Y coordinate
flatten_flag	vg_lite_uint8_t	Flatten flag for flattened path
curve_type	vg_lite_uint8_t	Curve type for the stroke path
tangentX	vg_lite_float_t	X tangent (Note: #define centerX tangent)
tangentY	vg_lite_float_t	Y tangent (Note: #define centerX tangent)
length	vg_lite_float_t	Line length
prev	vg_lite_path_point_ptr	Pointer to the previous point node

10.2.4 vg_lite_stroke_t structure

The structure provides stroke parameters and pointers to temp storage for a stroke sub path. Refer to function vg_lite_set_stroke parameter descriptions for additional description for some members. *(from March 2022)*

- Used in structure: vg_lite_path_t.

vg_lite_stroke_t members	**Type**	**Description**
cap_style	vg_lite_cap_style_t	Stroke cap style
join_style	vg_lite_join_style_t	Stroke joint style
line_width	vg_lite_float_t	Stroke line width
miter_limit	vg_lite_float_t	Stroke miter limit
*dash_pattern	vg_lite_float_t	Pointer to stroke dash pattern
pattern_count	vg_lite_uint32_t	Number of dash pattern repetitions
dash_phase	vg_lite_float_t	Stroke dash phrase
dash_length	vg_lite_float_t	Stroke dash initial length
dash_index	vg_lite_uint32_t	Stroke dash initial index
half_width	vg_lite_float_t	Half line width
pattern_length	vg_lite_float_t	Total length of stroke dash patterns.
miter_square	vg_lite_float_t	For fast checking
path_points	vg_lite_path_point_ptr	Temp storage for stroke sub path
path_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
point_count	unint32_t	Temp storage for stroke sub path
left_point	vg_lite_path_point_ptr	Temp storage for stroke sub path
right_pont	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_points	vg_lite_path_point_ptr	Temp storage for stroke sub path

vg_lite_stroke_t members	Type	Description
stroke_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_count	vg_lite_uint32_t	Temp storage for stroke sub path
path_list_divide	vg_lite_path_list_ptr	Divide stroke path according to move or move_rel for avoiding implicit closure. *(from Aug 2023)*
cur_list	vg_lite_path_list_ptr	Pointer to current divided path data. *(from Aug 2023)*
add_end	vg_lite_uint8_t	Flag that adds end_path in driver *(from Aug 2023)*
dash_reset	vg_lite_uint8_t	*(from Aug 2023)*
stroke_paths	vg_lite_sub_path_ptr	
last_stroke	vg_lite_sub_path_ptr	
swing_handling	vg_lite_uint32_t	
swing_deltax	vg_lite_float_t	
swing_deltay	vg_lite_float_t	
swing_start	vg_lite_path_point_ptr	
swing_stroke	vg_lite_path_point_ptr	
swing_length	vg_lite_float_t	
swing_centlen	vg_lite_float_t	
swing_count	vg_lite_uint32_t	
need_swing	vg_lite_uint8_t	
swing_ccw	vg_lite_uint8_t	
stroke_length	vg_lite_float_t	
stroke_size	vg_lite_uint32_t	
fattened	vg_lite_uint8_t	the stroke line is fat line.
closed	vg_lite_uint8_t	

10.2.5 vg_lite_sub_path_t structure

The structure vg_lite_sub_path_ptr points to a vg_lite_sub_path structure which provides sub path detail and a pointer to the next sub path. *(from March 2022)*

- Used in structure: vg_lite_stroke_conversion.

vg_lite_path_point_t members	Type	Description
next	vg_lite_sub_path_ptr	Pointer to the next sub path
point_count	vg_lite_uint32_t	Number of points in the sub path
point_list	vg_lite_path_point_ptr	Pointer to the point list.
end_point	vg_lite_path_point_ptr	Pointer to the last point.
closed	vg_lite_uint8_t	Indicates whether or not the path is closed.
length	vg_lite_float_t	Length of the sub path.

10.3 Stroke functions

All return vg_lite_error_t status.

10.3.1 vg_lite_set_path_type

Description

This function sets the path type. *(from March 2022*

Syntax

```
     vg_lite_error_t vg_lite_set_path_type (
         vg_lite_path_t              *path,
         vg_lite_path_type_t         path_type
     );
```

Parameters:

*path	Pointer to the vg_lite_path_t structure that describes the path.
path_type	Pointer to a vg_lite_path_type_t structure that describes the path type.

Returns:

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

10.3.2 vg_lite_set_stroke

Description

This function uses input parameters to set stroke attributes. *(from March 2022)*

Syntax

```
vg_lite_error_t vg_lite_set_stroke (
    vg_lite_path_t              *path,
    vg_lite_cap_style_t         cap_style,
    vg_lite_join_style_t        join_style,
    vg_lite_float_t             line_width,
    vg_lite_float_t             miter_limit,
    vg_lite_float_t             *dash_pattern,
    vg_lite_uint32_t            pattern_count,
vg_lite_float_t             dash_phase,
    vg_lite_color_t             color
);
```

Parameters

*path	Pointer to the vg_lite_path_t structure that describes the path.
cap_style	The end cap style defined by the vg_lite_cap_style_t enum.

join_style	The line join style defined by the vg_lite_join_style_t enum.
line_width	The line width of the stroke path. Note: A line width less than 0.5 prevents stroking from taking place.
miter_limit	When stroking using the Miter stroke vg_lite_join_style_t, the miter length (i.e., the length between the intersection points of the inner and outer perimeters of the two “fattened” lines) is compared to the product of the user-set miter limit and the line width. If the miter length exceeds this product, the Miter join is not drawn and a Bevel join is substituted. Note: Miter limit values less than 1 are silently clamped to 1.
*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.
pattern_count	The count of dash on/off segments.
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.
color	The stroke color.

Returns:

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

10.3.3 vg_lite_update_stroke

Description

This function uses the path and stroke attributes as specified with function vg_lite_set_stroke to update the stroke path's parameters and generate stroke path data. *(from March 2022*

Syntax

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

Parameters

*path	Pointer to the vg_lite_path_t structure that describes the path.

Returns:

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

11 Deprecated and renamed APIs

The following functions are deprecated and are either obsolete or replaced by a more efficient implementation. Their use is discouraged and will produce unpredictable behaviors.

The names of some functions, enums and structures were modified during code refinements in 2022Q3. If the parameters did not change, the deprecated syntax detail is not provided below. Changes to enums and structs are not mentioned here, instead refer to the item itself.

Deprecated or renamed API	Recommended replacement API	Source file	Date deprecated
vg_lite_perspective	n/a	vg_lite.h	August 2022
vg_lite_set_dither	vg_lite_enable_dither vg_lite_disable_dither	vg_lite.h	August 2022
vg_lite_append_path	vg_lite_path_append	vg_lite.h	Sept 2022
vg_lite_path_calc_length	vg_lite_get_path_length	vg_lite.h	Sept 2022
vg_lite_set_image_global_alpha	vg_lite_set_source_global_alpha	vg_lite.h	Sept 2022
vg_lite_dest_global_alpha	vg_lite_set_dest_global_alpha	vg_lite.h	Sept 2022
vg_lite_mem_avail	vg_lite_get_mem_size	vg_lite.h	Sept 2022
vg_lite_enable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_disable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_set_premultiply	n/a	vg_lite.h	Aug 2023
vg_lite_radial_gradient_spreadmode_t enum	vg_lite_gradient_spreadmode_t enum	vg_lite.h	March 2023
API Name Refinement	(no change to parameters)		
vg_lite_buffer_upload	vg_lite_upload_buffer_	vg_lite.h	Sept 2022
vg_lite_*mask*	most vg_lite_*mask_layer	vg_lite.h	Sept 2022
vg_lite_*_grad	vg_lite_*_gradient (parameters unchanged)	vg_lite.h	Sept 2022
vg_lite_*_radial_grad*	vg_lite_*_rad_grad*	vg_lite.h	Sept 2022
vg_lite_buffer_image_mode_t	vg_lite_image_mode_t	vg_lite.h	Sept 2022
vg_lite_transparency_mode_t	vg_lite_ transparency_t	vg_lite.h	Sept 2022
vg_lite_set_update_stroke	vg_lite_update_stroke	vg_lite.h	Sept 2022
vg_lite_set_draw_path_type	vg_lite_set_path_type	vg_lite.h	Sept 2022

11.1 Deprecated vg_lite syntax

Syntax for deprecated functions is provided below for reference. Note: this list does not include items renamed during code refinement of Sept 2022.

11.1.1 vg_lite_perspective *(deprecated)*

Syntax

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

11.1.2 vg_lite_set_dither *(deprecated)*

Syntax

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

11.1.3 vg_lite_enable_premultiply *(deprecated)*

Syntax

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

11.1.4 vg_lite_disable_premultiply *(deprecated)*

Syntax

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

11.1.5 vg_lite_set_premultiply *(deprecated)*

Syntax

```
vg_lite_error_t vg_lite_set_premultiply (
    vg_lite_uint8_t              src_premult,
    vg_lite_uint8_t              dst_premult,
);
```

12 VGLite API programming examples

12.1 vg_lite_clear example

The *Conformance/samples/clear/clear.c* test program 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 that as the tessellation buffer width and height are defined as (0, 0) in this vg_lite_init API call, this program cannot use the path rendering vg_lite_draw APIs. Only clear and blit APIs can be used in this program.

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

```
buffer.width  = 256;
buffer.height = 256;
buffer.format = VG_LITE_RGB565;
error = vg_lite_allocate(&buffer);
fb = &buffer;
```

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

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

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

```
vg_lite_rectangle_t rect = { 64, 64, 64, 64 };
error = vg_lite_clear(fb, &rect, 0xFF0000FF);
```

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

```
vg_lite_finish();
vg_lite_free(&buffer);
vg_lite_close();
```

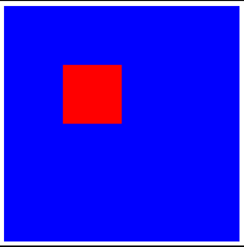


Figure 1 Example using vg_lite_clear

12.2 vg_lite_blit example

The *Conformance/samples/ui/main.c* test program demonstrates the usage of the vg_lite_blit API. It clears a 320x480 render buffer with blue background color first, then it blits six 256x256 icon images to six different positions in the render buffer with a blit matrix for each icon. The blit matrix scales the original icon image to a proper size and translates the scaled icon to the right position in the render buffer. The vg_lite_blit API call is set as VG_LITE_BLEND_SRC_OVER so the icon image pixels with alpha value 0xFF cover the background blue color.

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

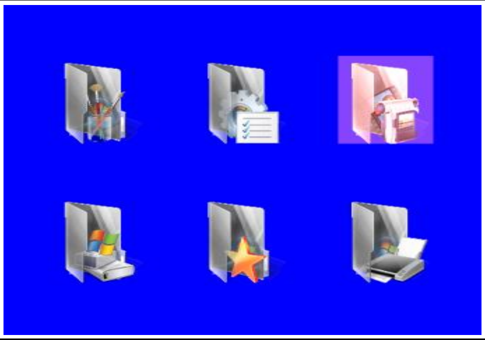


Figure 2 Example using vg_lite_blit

12.3 vg_lite_draw example

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

```
static char path_data[] = {
    2,   0,   0,          // moveto     0,   0
    4,  10,   0,          // lineto    10,   0
    4,  10,  10,          // lineto    10,  10
    4,  0,   10,          // lineto     0,  10
    0,                    // end
};
static vg_lite_path_t path = {
    {-10, -10, 10, 10},   // bounding box left, top, right, bottom
    VG_LITE_HIGH,         // quality
    VG_LITE_S8,           // -128 to 127 coordinate range
    {0},                  // uploaded
    sizeof(path_data),    // path length
    path_data,            // path data
    1                     // path changed
};


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

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

12.4 vg_lite_draw_gradient example

The *Conformance/samples/linearGrad/linearGrad.c* test program demonstrates the usage of the vg_lite_draw_grad API. It defines 5 colors (black, red, green, blue, white) in ramps[] and 5 stops in stops[] which are used for gradient color transition. It calls the following to setup the color gradient image.

```
vg_lite_uint32_t ramps[] = {0xff000000, 0xffff0000, 0xff00ff00,
0xff0000ff, 0xffffffff};
vg_lite_uint32_t stops[] = {0, 66, 122, 200, 255};
vg_lite_set_grad(&grad, 5, ramps, stops);
vg_lite_update_grad(&grad);
```

Note that the “colors” parameter (ramps[]) in vg_lite_set_grad API must be in ARGB8888 format with alpha at the higher byte.

It also sets up the gradient transformation matrix “matGrad” with a proper scale factor and 30 degree rotation.

```
matGrad = vg_lite_get_grad_matrix(&grad);
vg_lite_identity(matGrad);
vg_lite_rotate(30.0f, matGrad);
```

Then it calls:

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

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

After the draw gradient API, it calls:

```
vg_lite_finish();
vg_lite_clear_grad(&grad);
```

to flush the VGLite commands and clean up the gradient image buffer.

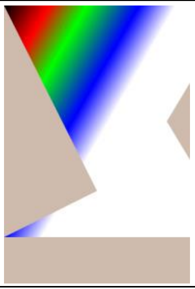


Figure 3 Example using vg_lite_draw_gradient

12.5 vg_lite_draw_pattern example

The *Conformance/samples/patternFill/patternFill.c* test program demonstrates the usage of the vg_lite_draw_pattern API. It defines a vg_lite_path_t path for a convex polygon shape as shown below, and loads an image file "landscape.raw" with which to fill the polygon interior area.

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

```
vg_lite_identity(&matrix);
vg_lite_translate(fb_width / 2.0f, fb_height / 4.0f, &matrix);
vg_lite_rotate(33.0f, &matrix);
vg_lite_scale(0.4f, 0.4f, &matrix);
vg_lite_translate(fb_width / -2.0f, fb_height / -4.0f, &matrix);

vg_lite_identity(&matPath);
vg_lite_translate(fb_width / 2.0f, fb_height / 4.0f, &matPath);
vg_lite_scale(10, 10, &matPath);
```

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

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

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

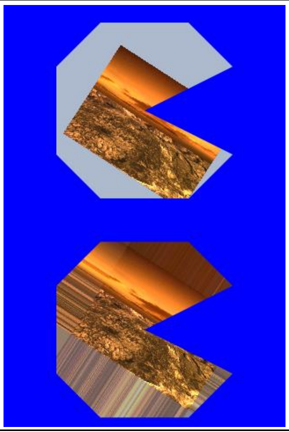


Figure 4 Example using vg_lite_draw_pattern

12.6 Vector-based font rendering example

The *Conformance/samples/glyphs2/glyphs2.c* test program demonstrates vector-based font rendering with the vg_lite_draw API, which is capable of drawing quadratic curves and cubic curves based on end point and control point coordinates in the path data. The font path data can be generated by using a third-party font engine that can produce VGLite path data directly, or by using VeriSilicon's VGLite tools to convert other formats of font data, such as SVG, etc., to VGLite path data. Here is an example of path data for the character "~" (ASCII code 126):

```
float ascii_font_126[] =
{
    2,15.984375,20.273438,
    4,16.296875,20.476563,
    6,15.781250,21.351563,14.921875,21.992188,
    6,13.953125,22.710938,13.046875,22.710938,
    6,12.375000,22.710938,10.898438,22.203125,
    6,9.421875,21.695313,8.656250,21.695313,
    6,7.937500,21.695313,7.375000,22.117188,
    6,7.015625,22.382813,6.421875,23.117188,
    4,6.109375,22.914063,
    6,7.593750,20.664063,9.453125,20.664063,
    6,10.156250,20.664063,11.492188,21.140625,
    6,12.828125,21.617188,13.531250,21.617188,
    6,14.921875,21.617188,15.984375,20.273438,
    0
};
```

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

The program calls:

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

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

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

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

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

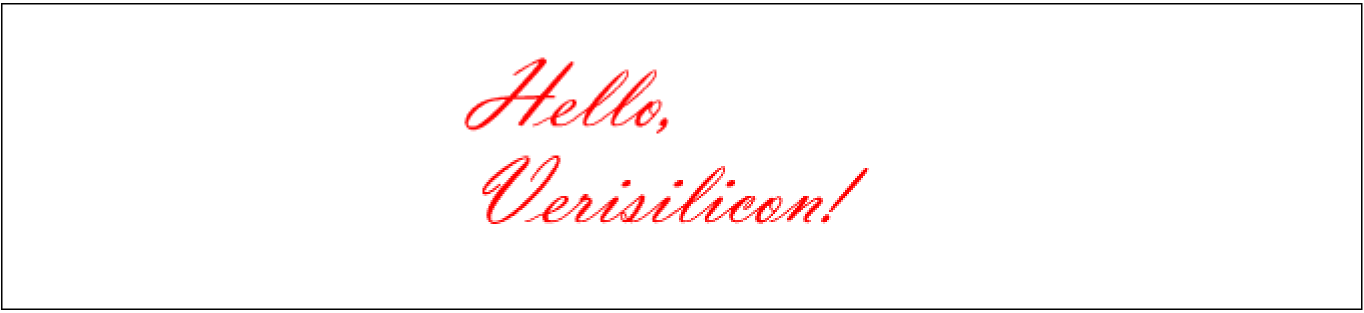


Figure 5 Example using vector-based font rendering

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

```
vg_lite_identity(&matrix);
vg_lite_translate(startX, startY, &matrix);
vg_lite_scale(8.0, 8.0, &matrix);
error = vg_lite_draw(fb, &path, VG_LITE_FILL_EVEN_ODD, &matrix,
VG_LITE_BLEND_NONE, 0xFF0000FF);
```

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

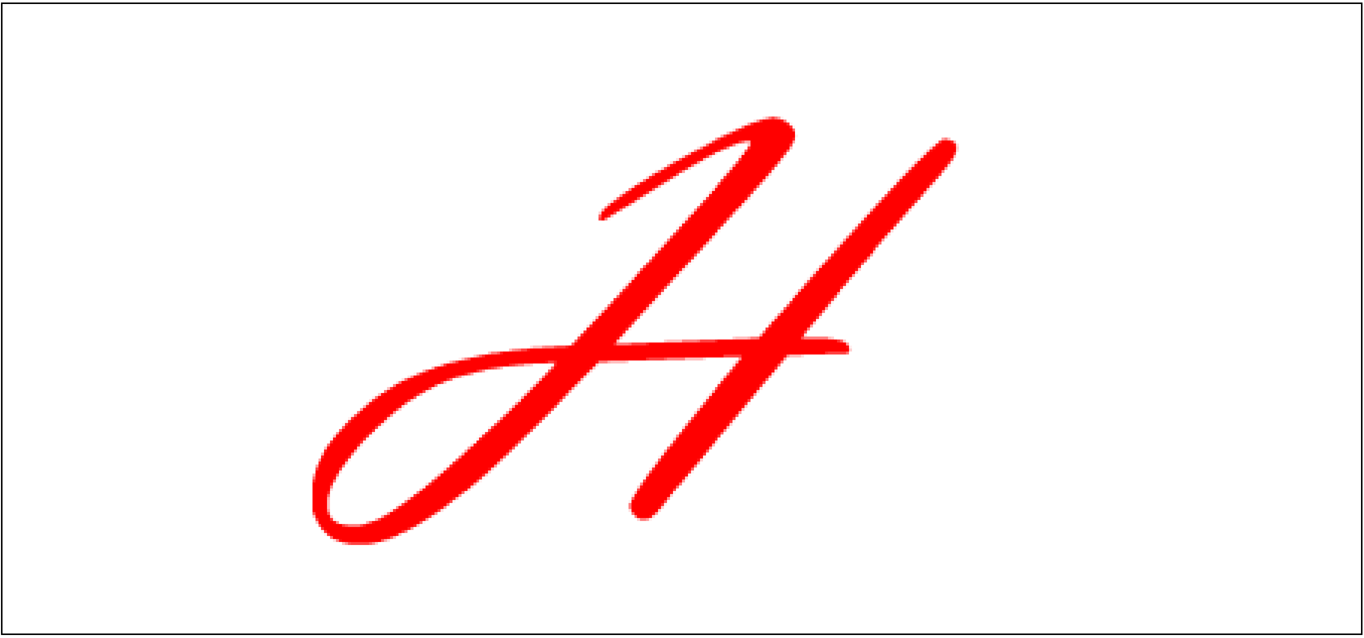


Figure 6 Example with vector-based font rendering upscaled 8X

Revision history

Document revision	Date	Description of changes
**	2024-09-17	Initial release
*A	2025-04-29	Added new APIs and structures, including ‘vg_lite_init_mem()’, to align with updated documentation and support PDL requirements. Removed unsupported scissor-related APIs (enable/disable scissor and scissor rect). Added default return statements that were missing previously and corrected minor typos.

Edition 2025-04-29

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-39840 Rev. *A