OpenVG 1.1 API Quick Reference Card - Page 1

OpenVG® is an API for hardware-accelerated two-dimensional vector and raster graphics. It provides a device-independent and vendor-neutral interface for sophisticated 2D graphical applications, while allowing device manufacturers to provide hardware acceleration where appropriate.

- **[n.n.n]** refers to sections and tables in the OpenVG 1.1 API specification available at www.khronos.org/openvg/
- Default values are shown in blue.

Errors [4.1]

Error codes and their numerical values are defined by the VGErrorCode enumeration and can be obtained with the function: VGErrorCode **vgGetError**(void). The possible values are as follows:

VG_NO_ERROR	0	VG_UNSUPPORTED_IMAGE_FORMAT_ERROR	0x1004
VG_BAD_HANDLE_ERROR	0x1000	VG_UNSUPPORTED_PATH_FORMAT_ERROR	0x1005
VG_ILLEGAL_ARGUMENT_ERROR	0x1001	VG_IMAGE_IN_USE_ERROR	0x1006
VG_OUT_OF_MEMORY_ERROR	0x1002	VG_NO_CONTEXT_ERROR	0x1007
VG_PATH_CAPABILITY_ERROR	0x1003		

Data Types & Number Representations

Primitive Data Types [3.2]

openvg.h	khronos_type.h	range
VGbyte	khronos_int8_t	[-128, 127]
VGubyte	khronos_uint8_t	[0, 255]
VGshort	khronos_int16_t	[-32768, 32767]
VGushort	khronos_uint16_t	[0, 65535]
VGint	khronos_int32_t	$[-2^{31}, (2^{31}-1)]$
VGuint	khronos_uint32_t	$[0, (2^{32}-1)]$
VGbitfield	khronos_uint32_t	$[0, (2^{32}-1)]$
VGboolean	khronos_int32_t	[0, 1]
VGfloat	khronos_float_t	IEEE 754 Standard

Number Representations [3.3]

VG_MAXSHORT	largest positive value of VGshort, smallest negative value is (–VG_MAXSHORT – 1)
VG_MAXINT	largest positive value of VGint, smallest negative value is (–VG_MAXINT – 1)
VG_MAX_FLOAT	largest floating-point number

Handle-based Data Types [3.6]

typedef VGuint VGHandle;

VGFont	reference to font data
VGImage	reference to image data
VGMaskLayer	reference to mask data
VGPaint	reference to a paint specification
VGPath	reference to path data

Colors [3.4]

Colors in OpenVG other than those stored in image pixels are represented as non-premultiplied sRGBA color values. Image pixel color and alpha values lie in the range [0,1] unless otherwise noted.

Color Space Definitions

The linear lRGB color space is defined in terms of the standard CIE XYZ color space, following ITU Rec. 709 using a D65 white point:

$$R = 3.240479\,X - 1.537150\,Y - 0.498535\,Z$$
$$G = -0.969256\,X + 1.875992\,Y + 0.041556\,Z$$
$$B = 0.055648\,X - 0.204043\,Y + 1.057311\,Z$$

Color Space Conversions

In the following table, the source format is in the left column, and the destination format is in the top row. The numbers indicate the numbered equations **(n)** from this section that are to be applied, in left-to-right order:

Src/Dst	lRGB	sRGB	lL	sL
lRGB	—	1	3	3, 5
sRGB	2	—	2, 3	2, 3, 5
lL	4	4, 1	—	5
sL	7, 2	7	6	—

The sRGB color space defines values R'_{sRGB}, G'_{sRGB}, B'_{sRGB} in terms of the linear lRGB primaries.

Convert from lRGB to sRGB (gamma mapping) (1)	Convert from sRGB to lRGB (inverse gamma mapping) (2)
$R'_{sRGB} = \gamma(R)$	$R = \gamma^{-1}(R'_{sRGB})$
$G'_{sRGB} = \gamma(G)$	$G = \gamma^{-1}(G'_{sRGB})$
$B'_{sRGB} = \gamma(B)$	$B = \gamma^{-1}(B'_{sRGB})$

The linear grayscale (luminance) color space (which we denote as lL) is related to the linear lRGB color space by the equations:

$$L = 0.2126\,R + 0.7152\,G + 0.0722\,B \quad (3)$$
$$R = G = B = L \quad (4)$$

The perceptually-uniform grayscale color space (which we denote as sL) is related to the linear grayscale (luminance) color space by the gamma mapping:

$$L' = \gamma(L) \quad (5)$$
$$L = \gamma^{-1}(L') \quad (6)$$

Conversion from perceptually-uniform grayscale to sRGB is performed by replication:

$$R' = G' = B' = L' \quad (7)$$

Object Parameter Set/Get API [5.3]

void **vgSetParameterf**(VGHandle *obj*, VGint *paramType*, VGfloat *val*)

void **vgSetParameteri**(VGHandle *obj*, VGint *paramType*, VGfloat *val*)

void **vgSetParameterfv**(VGHandle *obj*, VGint *paramType*, VGint *cnt*, const VGfloat * *val*)

void **vgSetParameteriv**(VGHandle *obj*, VGint *paramType*, VGint *cnt*, const VGint * *val*)

VGfloat **vgGetParameterf**(VGHandle *obj*, VGint *paramType*)

VGint **vgGetParameteri**(VGHandle *obj*, VGint *paramType*)

VGint **vgGetParameterVectorSize**(VGHandle *obj*, VGint *paramType*)

void **vgGetParameterfv**(VGHandle *obj*, VGint *paramType*, VGint *cnt*, VGfloat * *val*)

void **vgGetParameteriv**(VGHandle *obj*, VGint *type*, VGint *cnt*, VGint * *val*)

Drawing Context [4]

State Element	Description
Drawing Surface	Surface for drawing
Matrix Mode	Trans. to be manipulated
Path user-to-surface Transformation	Affine trans. for filled and stroked geometry
Image user-to-surface Transform	Affine or projective trans. for images
Paint-to-user Transformations	Affine transformations for paint applied to geometry
Glyph user-to-surface Transformation	Affine transformation for glyphs
Glyph origin	(X,Y) origin of glyph
Fill Rule	Rule for filling paths
Quality Settings	Image and rendering quality, pixel layout
Color Transformation	Color Transformation
Blend Mode	Pixel blend function
Image Mode	Image/paint combination
Scissoring	Enable/disable scissoring
Stroke	Stroke parameters
Pixel & Screen layout	Pixel layout information
Tile fill color	Color for FILL tiling mode
Clear color	Color for fast clear
Filter Parameters	Image filtering parameters
Paint	Paint definitions
Mask	Coverage mask and enable/disable
Error	Oldest unreported error

EGL Functions [4.2]

Usable EGLConfigs have EGL_OPENVG_BIT set in EGL_RENDERABLE_TYPE attribute. The EGL_ALPHA_MASK_SIZE attribute contains the bit depth of the mask. *attrib_list* is an array with pairs of *param_name* and value, terminating with EGL_NONE.
See EGL Attribute List

EGLBoolean **eglBindAPI**(EGLenum *api*)
api: use EGL_OPENVG_API. to bind OpenVG

EGLContext **eglCreateContext**(
EGLDisplay *dpy*, EGLConfig *config*,
EGLContext *share_context*,
const EGLint * *attrib_list*)

EGLSurface **eglCreateWindowSurface**(
EGLDisplay *dpy*, EGLConfig *config*,
NativeWindowType *win*,
const EGLint * *attrib_list*)

EGLSurface
eglCreatePbufferFromClientBuffer(
EGLDisplay *dpy*, EGLenum *buftype*,
EGLClientBuffer *buffer*, EGLConfig *config*,
const EGLint * *attrib_list*)
Pbuffer (off-screen buffer) allow rendering into a VGImage.

EGLBoolean **eglMakeCurrent**(
EGLDisplay *dpy*, EGLSurface *draw*,
EGLSurface *read*, EGLContext *context*)
Causes the given context to become current on the running thread.

EGLContext **eglGetCurrentContext**()

EGLBoolean **eglDestroyContext**(
EGLDisplay *dpy*,EGLContext *context*)

EGLBoolean **eglSwapBuffers**(
EGLDisplay *dpy*, EGLSurface *surface*)

EGL Attribute List

param_name
EGL_BUFFER_SIZE
EGL_ALPHA_SIZE
EGL_BLUE_SIZE
EGL_GREEN_SIZE
EGL_RED_SIZE
EGL_DEPTH_SIZE
EGL_STENCIL_SIZE
EGL_CONFIG_CAVEAT
EGL_CONFIG_ID
EGL_LEVEL
EGL_MAX_PBUFFER_HEIGHT
EGL_MAX_PBUFFER_PIXELS
EGL_MAX_PBUFFER_WIDTH
EGL_NATIVE_RENDERABLE
EGL_NATIVE_VISUAL_ID
EGL_NATIVE_VISUAL_TYPE
EGL_PRESERVED_RESOURCES
EGL_SAMPLES
EGL_SAMPLE_BUFFER
EGL_SURFACE_TYPE
EGL_TRANSPARENT_TYPE
EGL_TRANSPARENT_BLUE_VALUE
EGL_TRANSPARENT_GREEN_VALUE
EGL_TRANSPARENT_RED_VALUE
EGL_NONE
EGL_BIND_TO_TEXTURE_RGB
EGL_BIND_TO_TEXTURE_RGBA
EGL_MIN_SWAP_INTERVAL

param_name
EGL_MAX_SWAP_INTERVAL
EGL_LUMINANCE_SIZE
EGL_ALPHA_MASK_SIZE
EGL_COLOR_BUFFER_TYPE
EGL_RENDERABLE_TYPE
EGL_MATCH_NATIVE_PIXMAP
EGL_CONFORMANT
EGL_CONFORMANT_KHR
EGL_SLOW_CONFIG
EGL_NON_CONFORMANT_CONFIG
EGL_TRANSPARENT_RGB
EGL_RGB_BUFFER
EGL_LUMINANCE_BUFFER
EGL_NO_TEXTURE
EGL_TEXTURE_RGB
EGL_TEXTURE_RGBA
EGL_TEXTURE_2D
EGL_PBUFFER_BIT
EGL_PIXMAP_BIT
EGL_WINDOW_BIT
EGL_VG_COLORSPACE_LINEAR_BIT
EGL_VG_ALPHA_FORMAT_PRE_BIT
EGL_OPENGL_ES_BIT
EGL_OPENVG_BIT

Forcing Drawing to Complete API [4.3]

void **vgFlush**(void)
Complete requests in finite time.

void **vgFinish**(void)
Complete requests.

OpenVG 1.1 API Quick Reference Card - Page 2

Context Parameters

Context Parameter Set/Get API [5.2]

void **vgSetf**(VGParamType *paramType*, VGfloat *val*)

void **vgSeti**(VGParamType *paramType*, VGint *val*)

void **vgSetfv**(VGParamType *paramType*, VGint *cnt*, const VGfloat * *val*)

void **vgSetiv**(VGParamType *paramType*, VGint *cnt*, const VGint * *val*)

VGfloat **vgGetf**(VGParamType *paramType*)

VGint **vgGeti**(VGParamType *paramType*)

VGint **vgGetVectorSize**(VGParamType *paramType*)

void **vgGetfv**(VGParamType *paramType*, VGint *cnt*, VGfloat * *val*)

void **vgGetiv**(VGParamType *paramType*, VGint *cnt*, VGint * *val*)

Context Parameters [5.2.1]

The possible values of *paramType* from enumeration VGParamType are shown below, with the legal values for *val*. The type of *val* is shown in parentheses. Default value shown in blue.

VG_MATRIX_MODE (VGMatrixMode)
VG_MATRIX_PATH_USER_TO_SURFACE
VG_MATRIX_IMAGE_USER_TO_SURFACE
VG_MATRIX_FILL_PAINT_TO_USER
VG_MATRIX_STROKE_PAINT_TO_USER
VG_MATRIX_GLYPH_USER_TO_SURFACE

VG_FILL_RULE (VGFillRule)
VG_EVEN_ODD VG_NON_ZERO

VG_IMAGE_QUALITY (VGImageQuality)
VG_IMAGE_QUALITY_NONANTIALIASED
VG_IMAGE_QUALITY_FASTER
VG_IMAGE_QUALITY_BETTER

VG_RENDERING_QUALITY (VGRenderingQuality)
VG_RENDERING_QUALITY_NONANTIALIASED
VG_RENDERING_QUALITY_FASTER
VG_RENDERING_QUALITY_BETTER

VG_BLEND_MODE (VGBlendMode)
VG_BLEND_SRC
VG_BLEND_SRC_OVER
VG_BLEND_DST_OVER
VG_BLEND_SRC_IN
VG_BLEND_DST_IN
VG_BLEND_MULTIPLY
VG_BLEND_SCREEN
VG_BLEND_DARKEN
VG_BLEND_LIGHTEN
VG_BLEND_ADDITIVE

VG_IMAGE_MODE (VGImageMode)
VG_DRAW_IMAGE_NORMAL
VG_DRAW_IMAGE_MULTIPLY
VG_DRAW_IMAGE_STENCIL

VG_SCISSOR_RECTS (VGint *)
{ } (array of length 0)
{sx1,sy1,w1,h1,...}

VG_COLOR_TRANSFORM (VGboolean)
VG_TRUE VG_FALSE

VG_COLOR_TRANSFORM_VALUES (VGfloat[8])
{ 1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f }
{Rf, Gf, Bf, Af, Rb, Gb, Bb, Ab}

VG_STROKE_LINE_WIDTH (VGfloat)
1.0f

VG_STROKE_CAP_STYLE (VGCapStyle)
VG_CAP_BUTT
VG_CAP_ROUND
VG_CAP_SQUARE

VG_STROKE_JOIN_STYLE (VGJoinStyle)
VG_JOIN_MITER
VG_JOIN_ROUND
VG_JOIN_BEVEL

VG_STROKE_MITER_LIMIT (VGfloat)
4.0f

VG_STROKE_DASH_PATTERN (VGfloat *)
{ } (array of length 0) (disabled)
{on1, off1, on2, off2,....}

VG_STROKE_DASH_PHASE (VGfloat)
0.0f

VG_STROKE_DASH_PHASE_RESET (VGboolean)
VG_FALSE VG_TRUE

VG_TILE_FILL_COLOR (VGfloat[4])
{0.0f, 0.0f, 0.0f, 0.0f}
{red,green,blue,alpha}

VG_CLEAR_COLOR (VGfloat[4])
{0.0f, 0.0f, 0.0f, 0.0f}
{red,green,blue,alpha}

VG_GLYPH_ORIGIN (VGfloat[2])
{0.0f, 0.0f} {ox,oy}

VG_MASKING (VGboolean)
VG_TRUE VG_FALSE(disabled)

VG_SCISSORING (VGboolean)
VG_TRUE VG_FALSE(disabled)

VG_SCREEN_LAYOUT (VGPixelLayout)
VG_PIXEL_LAYOUT (VGPixelLayout)
VG_PIXEL_LAYOUT_UNKNOWN*
VG_PIXEL_LAYOUT_RGB_VERTICAL
VG_PIXEL_LAYOUT_BGR_VERTICAL
VG_PIXEL_LAYOUT_RGB_HORIZONTAL
VG_PIXEL_LAYOUT_BGR_HORIZONTAL

* This is the default for VG_PIXEL_LAYOUT only. The default for VG_SCREEN_LAYOUT is the layout of the drawing surface.

VG_FILTER_FORMAT_LINEAR (VGboolean)
VG_TRUE VG_FALSE (disabled)

VG_FILTER_FORMAT_PREMULTIPLIED (VGboolean)
VG_TRUE VG_FALSE(disabled)

VG_FILTER_CHANNEL_MASK (VGbitfield)
{VG_RED | VG_GREEN | VG_BLUE | VG_ALPHA}

Read-Only Context Parameters

VG_MAX_SCISSOR_RECTS (VGint)
VG_MAX_DASH_COUNT (VGint)
VG_MAX_KERNEL_SIZE (VGint)
VG_MAX_SEPARABLE_KERNEL_SIZE (VGint)
VG_MAX_GAUSSIAN_STD_DEVIATION (VGfloat)
VG_MAX_COLOR_RAMP_STOPS (VGint)
VG_MAX_IMAGE_WIDTH (VGint)
VG_MAX_IMAGE_HEIGHT (VGint)
VG_MAX_IMAGE_PIXELS (VGint)
VG_MAX_IMAGE_BYTES (VGint)
VG_MAX_FLOAT (VGfloat)

Matrix Transformation [6.6]

Select Matrix Mode

paramType values for the **vgSet***() and **vgGet***() functions.

VG_MATRIX_PATH_USER_TO_SURFACE	Affine
VG_MATRIX_IMAGE_USER_TO_SURFACE	Perspective
VG_MATRIX_FILL_PAINT_TO_USER	Affine
VG_MATRIX_STROKE_PAINT_TO_USER	Affine
VG_MATRIX_GLYPH_USER_TO_SURFACE	Affine

Example:

```
vgSeti(VG_MATRIX_MODE,
 VG_MATRIX_PATH_USER_TO_
 SURFACE);
```

Matrix Manipulation Functions

Matrix m =
{ sx, shy, w0, shx, sy, w1, tx, ty, w2 }
In affine transform
w0 = w1 = 0.0, w2 = 1.0

void **vgLoadIdentity**(void)

void **vgLoadMatrix**(const VGfloat * *m*)

void **vgMultMatrix**(const VGfloat * *m*)

void **vgGetMatrix**(VGfloat * *m*)

void **vgTranslate**(VGfloat *tx*, VGfloat *ty*)

void **vgScale**(VGfloat *sx*, VGfloat *sy*)

void **vgShear**(VGfloat *shx*, VGfloat *shy*)

void **vgRotate**(VGfloat *angle*)

Path

Segment Commands [8.5.2]

Reference points are defined as: (*sx, sy*): beginning of the current subpath; (*ox, oy*): last point of the previous segment; (*px, py*): last internal control point of the previous segment if the segment was a (regular or smooth) quadratic or cubic Bézier, or else the last point of the previous segment.

The following table describes each segment command type and the side effects of the segment command on the termination of the current subpath.

VGPathSegment	Coordinates	Implicit Points	Description (Side Effects)
VG_CLOSE_PATH	none		(*px,py*)=(*ox,oy*)=(*sx,sy*) End current subpath.
VG_MOVE_TO	x0,y0		(*sx,sy*)=(*px,py*)=(*ox,oy*)=(*x0,y0*) End current subpath.
VG_LINE_TO	x0, y0		(*px,py*)=(*ox,oy*)=(*x0,y0*)
VG_HLINE_TO	x0	*y0=oy*	(*px,py*)=(*x0,oy*), *ox=x0*
VG_VLINE_TO	y0	*x0=ox*	(*px,py*)=(*ox,y0*), *oy=y0*
VG_QUAD_TO	x0,y0,x1,y1		(*px,py*)=(*x0,y0*) (*ox,oy*)=(*x1,y1*)
VG_CUBIC_TO	x0,y0,x1,y1,x2,y2		(*px,py*)=(*x1,y1*) (*ox,oy*)=(*x2,y2*)
VG_SQUAD_TO	x1,y1	(*x0,y0*)= (2**ox-px*,2**oy-py*)	(*px,py*)=(2**ox-px*, 2**oy-py*) (*ox,oy*)=(*x1,y1*)
VG_SCUBIC_TO	x1,y1,x2,y2	(*x0,y0*)= (2**ox-px*,2**oy-py*)	(*px,py*)=(*x1,y1*),(*ox,oy*)=(*x2,y2*)
VG_SCCWARC_TO	rh,rv,rot,x0,y0		(*px,py*)=(*ox,oy*)=(*x0,y0*)
VG_SCWARC_TO	rh,rv,rot,x0,y0		(*px,py*)=(*ox,oy*)=(*x0,y0*)
VG_LCCWARC_TO	rh,rv,rot,x0,y0		(*px,py*)=(*ox,oy*)=(*x0,y0*)
VG_LCWARC_TO	rh,rv,rot,x0,y0		(*px,py*)=(*ox,oy*)=(*x0,y0*)

Scissor, Mask, and Fast Clear

Scissoring [7.1]

paramType values for the **vgSet***() and **vgGet***() functions. Defaults are in blue.

VG_SCISSORING (VGboolean)
VG_TRUE
VG_FALSE (disabled)

VG_SCISSOR_RECTS (VGint *)
{ } (array of length 0)
{sx1,sy1,w1,h1,...}

Example:

```
#define NUM_RECTS 2
VGint coords[4*NUM_RECTS]
    = { 20, 30, 100, 200,
    50, 70, 80, 80 };
// order of x, y, w, h
vgSetiv (VG_SCISSOR_RECTS,
     4*NUM_RECTS, coords);
```

Masking [7.2]

void **vgMask**(VGHandle *mask*, VGMaskOperation *op*, VGint *x*, VGint *y*, VGint *width*, VGint *height*)

void **vgRenderToMask**(VGPath *path*, VGbitfield *paintMode*, VGMaskOperation *op*)

VGMaskLayer **vgCreateMaskLayer**(VGint *width*, VGint *height*)

void **vgDestroyMaskLayer**(VGMaskLayer *masklayer*)

void **vgFillMaskLayer**(VGMaskLayer *masklayer*, VGint *x*, VGint *y*, VGint *width*, VGint *height*, VGfloat *val*)

void **vgCopyMask**(VGMaskLayer *masklayer*, VGint *x*, VGint *y*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*, VGfloat *val*)

VGMaskOperation

Mr=resulting mask, Mn=input mask, Mp=previous mask

VG_CLEAR_MASK . . . . . .Mr = 0
VG_FILL_MASK . . . . . .Mr = 1
VG_SET_MASK . . . . . .Mr = Mn
VG_UNION_MASK. . . . . .Mr = 1 - (1-Mn)*(1-Mp)
VG_INTERSECT_MASK . .Mr = Mn * Mp
VG_SUBTRACT_MASK. . .Mr = Mp * (1-Mn)

Fast Clear [7.3]

void **vgClear**(VGint *x*, VGint *y*, VGint *width*, VGint *height*)

Path Operations [8.6]

Path capabilities are specified as bits in a VGbitfield, with the following values defined in the VGPathCapabilities enumeration:

VG_PATH_CAPABILITY_APPEND_FROM
VG_PATH_CAPABILITY_APPEND_TO
VG_PATH_CAPABILITY_MODIFY
VG_PATH_CAPABILITY_TRANSFORM_FROM
VG_PATH_CAPABILITY_TRANSFORM_TO
VG_PATH_CAPABILITY_INTERPOLATE_FROM
VG_PATH_CAPABILITY_INTERPOLATE_TO
VG_PATH_CAPABILITY_PATH_LENGTH
VG_PATH_CAPABILITY_POINT_ALONG_PATH
VG_PATH_CAPABILITY_TANGENT_ALONG_PATH
VG_PATH_CAPABILITY_PATH_BOUNDS
VG_PATH_CAPABILITY_PATH_TRANSFORMED_BOUNDS
VG_PATH_CAPABILITY_ALL

Path Object Parameter [8.6.3]

paramType values for the **vgSetParameter**() and **vgGetParameter**() functions. Default value shown in blue.

VG_PATH_FORMAT (VGint)
VG_PATH_FORMAT_STANDARD 0

VG_PATH_DATATYPE (VGPathDatatype)
VG_PATH_DATATYPE_S_{8, 16, 32}
VG_PATH_DATATYPE_F

VG_PATH_BIAS (VGfloat)
0.0f

VG_PATH_NUM_SEGMENTS (VGint)
0

VG_PATH_NUM_COORDS (VGint)
0

VG_PATH_SCALE (VGfloat)
1.0f

(Continued >)

OpenVG 1.1 API Quick Reference Card - Page 3

Path (continued)

Create and Destroy Path [8.6.2]

VGPath **vgCreatePath**(VGint *pathFormat*, VGPathDatatype *datatype*, VGfloat *scale*, VGfloat *bias*, VGint *segCapacityHint*, VGint *coordCapacityHint*, VGbitfield *capabilities*)

void **vgClearPath**(VGPath *path*, VGbitfield *capabilities*)

void **vgDestroyPath**(VGPath *path*)

Query & Modify Path Capabilities [8.6.4]

VGbitfield **vgGetPathCapabilities**(VGPath *path*)

void **vgRemovePathCapabilities**(VGPath *path*, VGbitfield *capabilities*)

Copy Data Between Paths [8.6.5-6]

void **vgAppendPath**(VGPath *dstPath*, VGPath *srcPath*)

void **vgAppendPathData**(VGPath *dstPath*, VGint *numSeg*, const VGubyte * *pathSeg*, const void * *pathData*)

Modify Path Data [8.6.7]

void **vgModifyPathCoords**(VGPath *dstPath*, VGint *startIdx*, VGint *sumSeg*, const void * *pathData*)

Transform Path [8.6.8]

(VG_MATRIX_PATH_USER_TO_SURFACE is applied)

void **vgTransformPath**(VGPath *dstPath*, VGPath *srcPath*)

Interpolate Between Paths [8.6.9]

VGboolean **vgInterpolatePath**(VGPath *dstPath*, VGPath *startPath*, VGPath *endPath*, VGfloat *amount*)

Length of Path [8.6.10]

VGfloat **vgPathLength**(VGPath *path*, VGint *startSeg*, VGint *numSeg*)

Position & Tangent Along Path [8.6.11]

void **vgPointAlongPath**(VGPath *path*, VGint *startSeg*, VGint *numSeg*, VGfloat *distance*, VGfloat * *x*, VGfloat * *y*, VGfloat * *tanX*, VGfloat **tanY*)

Query Bounding Box [8.6.12]

void **vgPathBounds**(VGPath *path*, VGfloat **minx*, VGfloat * *miny*, VGfloat * *width*, VGfloat * *height*)

void **vgPathTransformedBounds**(VGPath *path*, VGfloat * *minx*, VGfloat * *miny*, VGfloat * *width*, VGfloat * *height*)

Draw Path [8.8]

VGfloat **vgPathLength**(VGPath *path*, VGint *startSeg*, VGint *numSeg*)

VGfloat **vgDrawPath**(VGPath *path*, VGbitfield *paintModes*)

paintModes: bitwise OR of {VG_FILL_PATH | VG_STROKE_PATH}

Paint

Paint Definition [9.1]

typedef **VGHandle** VGPaint;

Create & Destroy Paint [9.1.1]

VGPaint **vgCreatePaint**(void)

void **vgDestroyPaint**(VGPaint *paint*)

Set the Current Paint [9.1.2]

void **vgSetPaint**(VGPaint *paint*, VGbitfield *paintMode*)

VGPaint **vgGetPaint**(VGPaintMode *paintModes*)

paintModes: bitwise OR of {VG_FILL_PATH | VG_STROKE_PATH}

Paint Object Parameter (VGPaintParamType) [9.1.3]

paramType values for the **vgSetParameter**() and **vgGetParameter**() functions. Default value in blue.

VG_PAINT_TYPE (VGPaintType) VG_PAINT_TYPE_COLOR VG_PAINT_TYPE_{LINEAR, RADIAL}_GRADIENT VG_PAINT_TYPE_PATTERN
VG_PAINT_COLOR (VGfloat[4]) {0.0f, 0.0f, 0.0f, 1.0f} {red,green,blue,alpha}
VG_PAINT_COLOR_RAMP_SPREAD_MODE (VGColorRampSpreadMode) VG_COLOR_RAMP_SPREAD_PAD VG_COLOR_RAMP_SPREAD_{REPEAT, REFLECT}
VG_PAINT_COLOR_RAMP_PREMULTIPLIED (VGboolean) VG_TRUE VG_FALSE (disabled)
VG_PAINT_COLOR_RAMP_STOPS (VGfloat *) NULL {stop0, red0, green0, blue0, alpha0,...}
VG_PAINT_LINEAR_GRADIENT (VGfloat[4]) {0.0f, 0.0f, 1.0f, 0.0f} {startx, starty, endx, endy}
VG_PAINT_RADIAL_GRADIENT (VGfloat[5]) {0.0f, 0.0f, 0.0f, 0.0f, 1.0f} {centerx, centery, focusx, focusy, radius}
VG_PAINT_PATTERN_TILING_MODE (VGTilingMode) VG_TILE_FILL VG_TILE_{PAD, REPEAT, REFLECT}

Color Paint [9.2]

Color paint uses a fixed color and alpha for all pixels. Colors are specified in non-premultiplied sRGBA format.

void **vgSetParameterfv**(VGPaint *paint*, VG_PAINT_COLOR, 4, VGfloat *col*[4])

void **vgSetColor**(VGPaint *paint*, VGuint *rgba*)

VGuint **vgGetColor**(VGPaint *paint*)

Gradient Paint [9.3]

Linear Gradients

Enable using **vgSetParameteri** to set the paint type to VG_PAINT_TYPE_LINEAR_GRADIENT. Set parameters using **vgSetParameterfv** with a *paramType* argument of VG_PAINT_LINEAR_GRADIENT.

Radial Gradients

Enable using **vgSetParameteri** to set the paint type to VG_PAINT_TYPE_RADIAL_GRADIENT. Set parameters using **vgSetParameterfv** with a *paramType* argument of VG_PAINT_RADIAL_GRADIENT.

Pattern Paint [9.4]

void **vgPaintPattern**(VGPaint *paint*, VGImage *pattern*)

Images

Image Definition [10.3]

typedef **VGHandle** VGImage;

paramType values for the **vgSet***() and **vgGet***() functions.

VG_IMAGE_QUALITY (VGImageQuality) VG_IMAGE_QUALITY_NONANTIALIASED VG_IMAGE_QUALITY_FASTER VG_IMAGE_QUALITY_BETTER
VG_MAX_IMAGE_WIDTH
VG_MAX_IMAGE_HEIGHT
VG_MAX_IMAGE_PIXELS
VG_MAX_IMAGE_BYTES

Create & Destroy Image [10.3]

VGImage **vgCreateImage**(VGImageFormat *fmt*, VGint *width*, VGint *height*, VGbitfield *quality*)

void **vgDestroyImage**(VGImage *image*)

Image Object Parameter (VGImageParamType) [10.4]

paramType values for the vgSetParameter() and vgGetParameter() functions.

VG_IMAGE_FORMAT (VGImageFormat)	
// RGB{A,X} channel ordering: VG_sRGB_565 VG_{s,l}RGBX_8888 VG_{s,l}RGBA_PRE VG_sRGBA_{5551,4444} VG_{sL,lL,A}_8 VG_{BW,A}_1	// {A,X}RGB channel ordering: VG_{s,l}{XRGB,ARGB}_8888 VG_{s,l}ARGB_8888_PRE VG_{sARGB}_{1555,4444}
// {BGR{A,X} channel ordering: VG_{s,l}{BGRX,BGRA}_8888 VG_{s,l}BGRA_8888_PRE VG_{sBGRA}_{1555,4444}	// {A,X}BGR channel ordering: VG_{s,l}{XBGR,ABGR}_8888 VG_{s,l}ABGR_8888_PRE VG_{sABGR}_{1555,4444}
VG_IMAGE_WIDTH (VGint)	// default value = 0
VG_IMAGE_HEIGHT (VGint)	// default value = 0

Read and Write Image Pixels [10.5]

void **vgClearImage**(VGImage *image*, VGint *x*, VGint *y*, VGint *width*, VGint *height*)

void **vgImageSubData**(VGImage *image*, const void * *data*, VGint *dataStride*, VGImageFormat *fmt*, VGint *x*, VGint *y*, VGint *width*, VGint *height*)

void **vgGetImageSubData**(VGImage *image*, void * *data*, VGint *dataStride*, VGImageFormat *fmt*, VGint *x*, VGint *y*, VGint *width*, VGint *height*)

Child Images [10.6]

VGImage **vgChildImage**(VGImage *parent*, VGint *x*, VGint *y*, VGint *width*, VGint *height*)

VGImage **vgGetParent**(VGImage *image*)

Copy Between Images [10.7]

void **vgCopyImage**(VGImage *dst*, VGint *dx*, VGint *dy*, VGImage *src*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*, VGboolean *dither*)

Draw Image [10.8]

void **vgDrawImage**(VGImage *image*)

Read and Write Drawing Surface Pixels [10.9]

void **vgSetPixels**(VGint *dx*, VGint *dy*, VGImage *src*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*)

void **vgWritePixels**(const void * data, VGint dataStride, VGImageFormat *fmt*, VGint *dx*, VGint *dy*, VGint *width*, VGint *height*)

void **vgGetPixels**(VGImage *dst*, VGint *dx*, VGint *dy*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*)

void **vgReadPixels**(void * *data*, VGint *dataStride*, VGImageFormat *fmt*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*)

void **vgCopyPixel**(VGint *dx*, VGint *dy*, VGint *sx*, VGint *sy*, VGint *width*, VGint *height*)

Pixel Copy Functions [10.9]

Src/Dst	Memory	VGImage	Surface
Memory	—	vgImageSubData	vgWritePixels
VGImage	vgGetImageSubData	vgCopyImage	vgSetPixels
Surface	vgReadPixels	vgGetPixels	vgCopyPixels

Text and Font Operations

OpenVG provides a fast, low-level hardware-accelerated API that is capable of supporting both hinted and unhinted vector glyph outlines, as well as glyphs represented as bitmaps.

Font Definition [11.4]

typedef VGHandle VGFont;

Manage VGFont Object [11.4.2]

VGFont **vgCreateFont**(VGint *glyphCapacityHint*)

void **vgDestroyFont**(VGFont *font*)

Font Object Parameter (VGFontParamType) [11.4.3]

paramType value for the vgGetParameter() function.

VG_FONT_NUM_GLYPHS (VGint) // default value = 0

Add/Modify Glyphs in Fonts [11.4.4]

Applications are responsible for destroying path or image objects they have assigned as font glyphs. It is recommended that applications destroy the path or image using vgDestroyPath or vgDestroyImage immediately after setting the object as a glyph.

void **vgSetGlyphToPath**(VGFont *font*, VGuint *glyphIndex*, VGPath *path*, VGboolean *inHinted*, const VGfloat *origin*[2], const VGfloat *escape*[2])

void **vgSetGlyphToImage**(VGFont *font*, VGuint *glyphIndex*, VGImage *image*, const VGfloat *origin*[2], const VGfloat *escape*[2])

void **vgClearGlyph**(VGFont *font*, VGuint *glyphIndex*)

Font Sharing [11.4.5]

In order for VGFont objects to be shared, the VGFont (and underlying VGPath and VGImage objects) must be bound to a shared context.

Draw Text [11.5]

void **vgDrawGlyph**(VGFont *font*, VGuint *glyphIndex*, VGbitfield *paintModes*, VGboolean *allowAutoHinting*)

void **vgDrawGlyphs**(VGFont *font*, VGint *glyphCount*, const VGuint * *glyphIndices*, const VGfloat * *adjustments_x*, const VGfloat * *adjustments_y*, VGbitfield *paintModes*, VGboolean *allowAutoHinting*)

OpenVG 1.1 API Quick Reference Card - Page 4

Image Filter

Image filters allow images to be modified or combined using a variety of imaging operations.

Format Normalization [12.1]

Source pixels are converted to one of sRGBA, sRGBA_PRE, lRGBA, or lRGBA_PRE formats, as determined by the current values of the VG_FILTER_FORMAT_PREMULTIPLIED and VG_FILTER_FORMAT_LINEAR parameters. Filtered pixels are then converted into the destination format using the normal pixel format conversion rules described in [3.4].

Channel Masks [12.2]

The VG_FILTER_CHANNEL_MASK parameter specifies which destination channels are to be written. The parameter is supplied as a bitwise OR of values from the VGImageChannel enumeration.

```
typedef enum {
    VG_RED    = (1 << 3),
    VG_GREEN  = (1 << 2),
    VG_BLUE   = (1 << 1),
    VG_ALPHA  = (1 << 0)
} VGImageChannel;
```

Color Combination [12.3]

4x4 color multiplication

void **vgColorMatrix**(VGImage *dst*, VGImage *src*, const VGfloat * *matrix*)

Convolution [12.4]

void **vgConvolve**(VGImage *dst*, VGImage *src*, VGint *kernelW*, VGint *KernelH*, VGint *shiftX*, VGint *shiftY*, const VGshort * *kernel*, VGfloat *scale*, VGfloat *bias*, VGTilingMode *tilingMode*)

void **vgSeparableConvolve**(VGImage *dst*, VGImage *src*, VGint *kernelW*, VGint *KernelH*, VGint *shiftX*, VGint *shiftY*, const VGshort * *kernelX*, const VGshort *kernelY*, VGfloat *scale*, VGfloat *bias*, VGTilingMode *tilingMode*)

void **vgGaussianBlur**(VGImage *dst*, VGImage *src*, VGfloat *stdDevX*, VGfloat *stdDevY*, VGTilingMode *tilingMode*)

Convolution Parameters

Read-only *paramType* values for the **vgGetParameter**() function.

VG_MAX_KERNEL_SIZE
Largest legal value of *width* and *height* (vgConvolve)

VG_MAX_SEPARABLE_KERNEL_SIZE
Largest legal value of the *size* parameter (vgSeparableConvolve)

VG_MAX_GAUSSIAN_STD_DEVIATION
Largest legal value of the *stdDeviationX* and *stdDeviationY* parameters (vgGaussianBlur)

Lookup Table [12.5]

void **vgLookup**(VGImage *dst*, VGImage *src*, const VGubyte * *redLUT*, const VGubyte * *greenLUT*, const VGubyte * *blueLUT*, const VGubyte * *alphaLUT*, VGboolean *outputLinear*, VGboolean *outputPremultiplied*)

void **vgLookupSingle**(VGImage *dst*, VGImage *src*, const VGuint * *LUT*, VGImageChannel *sourceChannel*, VGboolean *outputLinear*, VGboolean *outputPremultiplied*)

Querying Hardware Capabilities [14]

vgHardwareQuery

Indicates whether a given setting of a property of a type given by key is generally accelerated in hardware.

VGHardwareQueryResult **vgHardwareQuery**(VGHardwareQueryType *key*, VGint *setting*)

key: VG_IMAGE_FORMAT_QUERY, VG_PATH_DATATYPE_QUERY

setting: One of the constants from the enumerations VGImageFormat [10.2] or VGPathDataType [8.5.3]

Returns VG_HARDWARE_ACCELERATED, VG_HARDWARE_UNACCELERATED

Blending and Stencil Equations

Blending Equations [13.2-5]

Blending modes define alpha and color blending functions. Alpha blending function $\alpha(\alpha_{src}, \alpha_{dst})$; Color blending function $c(c_{src}, c_{dst}, \alpha_{src}, \alpha_{dst})$; Pre-mult alpha form $c'(\alpha_{src} * c_{src}, \alpha_{dst} * c_{dst}, \alpha_{src}, \alpha_{dst}) = c'(c'_{src}, c'_{dst}, \alpha_{src}, \alpha_{dst})$

Blend Mode	Color blending function $c'(c'_{src}, c'_{dst}, \alpha_{src}, \alpha_{dst})$	Apha blending function $\alpha(\alpha_{src}, \alpha_{dst})$
Porter-Duff Blending		
VG_BLEND_SRC	c'_{src}	α_{src}
VG_BLEND_SRC_OVER	$c'_{src} + c'_{dst} * (1-\alpha_{src})$	$\alpha_{src} + \alpha_{dst} * (1-\alpha_{src})$
VG_BLEND_DST_OVER	$c'_{src} * (1-\alpha_{dst}) + c'_{dst}$	$\alpha_{src} * (1-\alpha_{dst}) + \alpha_{dst}$
VG_BLEND_SRC_IN	$c'_{src} * \alpha_{dst}$	$\alpha_{src} * \alpha_{dst}$
VG_BLEND_DST_IN	$c'_{dst} * \alpha_{src}$	$\alpha_{dst} * \alpha_{src}$
Additional Blending		
VG_BLEND_MULTIPLY	$c'_{src} * (1-\alpha_{dst}) + c'_{dst} * (1-\alpha_{src}) + c'_{src} * c'_{dst}$	$\alpha_{src} + \alpha_{dst} * (1-\alpha_{src})$
VG_BLEND_SCREEN	$c'_{src} + c'_{dst} - c'_{src} * c'_{dst}$	$\alpha_{src} + \alpha_{dst} * (1-\alpha_{src})$
VG_BLEND_DARKEN	$\min(c'_{src} + c'_{dst} * (1-\alpha_{src}), c'_{dst} + c'_{src} * (1-\alpha_{dst}))$	$\alpha_{src} + \alpha_{dst} * (1-\alpha_{src})$
VG_BLEND_LIGHTEN	$\max(c'_{src} + c'_{dst} * (1-\alpha_{src}), c'_{dst} + c'_{src} * (1-\alpha_{dst}))$	$\alpha_{src} + \alpha_{dst} * (1-\alpha_{src})$
Additive Blending		
VG_BLEND_ADDITIVE	$\min(c'_{src} + c'_{dst}, 1)$	$\min(\alpha_{src} + \alpha_{dst}, 1)$

Stencil Equations [10.8]

In stencil mode, equations for blending are changed as follows:

$\alpha_{tmp} = \alpha(\alpha_{image} * \alpha_{paint}, \alpha_{dst})$
$R_{dst} \leftarrow c(R_{paint}, R_{dst}, R_{image} * \alpha_{image} * \alpha_{paint}, \alpha_{dst}) / \alpha_{tmp}$
$G_{dst} \leftarrow c(G_{paint}, G_{dst}, G_{image} * \alpha_{image} * \alpha_{paint}, \alpha_{dst}) / \alpha_{tmp}$
$B_{dst} \leftarrow c(B_{paint}, B_{dst}, B_{image} * \alpha_{image} * \alpha_{paint}, \alpha_{dst}) / \alpha_{tmp}$
$\alpha_{dst} \leftarrow \alpha_{tmp}$

If drawing surface has a luminance-only format:

$\alpha_{tmp} = \alpha(\alpha_{image} * \alpha_{paint}, \alpha_{dst})$
$L_{dst} \leftarrow c(L_{paint}, L_{dst}, L_{image} * \alpha_{image} * \alpha_{paint}, \alpha_{dst}) / \alpha_{tmp}$
$\alpha_{dst} \leftarrow \alpha_{tmp}$

VGU Utility Library [17]

Applications may choose whether to link to VGU at compile time; the library is not guaranteed to be present on the run-time platform. VGU is designed so it may be implemented in a portable manner using only the public functionality provided by the OpenVG library.

VGU Version

For the current version, the constant `VGU_VERSION_1_1` is defined. The older version `VGU_VERSION_1_0` continues to be defined for backwards compatibility.

```
#define VGU_VERSION_1_0     1
#define VGU_VERSION_1_1     2
```

High-Level Geometric Primitives [17.1]

These functions allow applications to specify high-level geometric primitives to be appended to a path. Each primitive is reduced to a series of line segments, Bézier curves, and arcs. Input coordinates are mapped to input values for the **vgAppendPathData** command by subtracting the path's bias and dividing by its scale value. Coordinates may overflow silently if the resulting values fall outside the range defined by the path datatype.

vguErrorCode **vguLine**(VGPath *path*, VGfloat *x0*, VGfloat *y0*, VGfloat *x1*, VGfloat *y1*)
Appends a line segment to a path.

vguErrorCode **vguPolygon**(VGPath *path*, const VGfloat * *points*, VGint *count*, VGboolean *closed*)
Appends a polyline (connected sequence of line segments) or polygon to a path.

vguErrorCode **vguRect**(VGPath *path*, VGfloat *x*, VGfloat *y*, VGfloat *width*, VGfloat *height*)
Appends an axis-aligned rectangle with its lower-left corner at (*x*,*y*) and a given *width* and *height* to a path.

vguErrorCode **vguRoundRect**(VGPath *path*, VGfloat *x*, VGfloat *y*, VGfloat *width*, VGfloat *height*, VGfloat *arcW*, VGfloat *arcH*)
Appends an axis-aligned round-cornered rectangle with the lower-left corner of its rectangular bounding box at (*x*, *y*) and a given *width*, *height*, *arcWidth*, and *arcHeight* to a path.

vguErrorCode **vguEllipse**(VGPath *path*, VGfloat *cx*, VGfloat *cy*, VGfloat *width*, VGfloat *height*)
Appends an axis-aligned ellipse to a path. The center of the ellipse is given by (*cx*, *cy*) and the dimensions of the axis-aligned rectangle enclosing the ellipse are given by *width* and *height*.

vguErrorCode **vguArc**(VGPath *path*, VGfloat *x*, VGfloat *y*, VGfloat *width*, VGfloat *height*, VGfloat *startAngle*, VGfloat *angleExt*, VGUArcType *arcType*)
Appends an elliptical arc to a path, possibly along with one or two line segments, according to the *arcType* parameter. The *startAngle* and *angleExtent* parameters are given in degrees, proceeding counter-clockwise from the positive X axis.

arcType may be one of the constants from the following table:

VGU_ARC_OPEN	arc segment only
VGU_ARC_CHORD	arc, plus line between arc endpoints
VGU_ARC_PIE	arc, plus lines from each endpoint to the ellipse center

Image Warping [17.2]

These functions compute 3x3 projective transform matrices. The first two compute the transformation from an arbitrary quadrilateral onto the unit square, and vice versa. The third computes the transformation from an arbitrary quadrilateral to an arbitrary quadrilateral.

vguErrorCode **vguComputeWarpQuadToSquare**(VGfloat *sx0*, VGfloat *sy0*, VGfloat *sx1*, VGfloat *sy1*, VGfloat *sx2*, VGfloat *sy2*, VGfloat *sx3*, VGfloat *sy3*, VGfloat * *matrix*)

vguErrorCode **vguComputeWarpSquareToQuad**(VGfloat *dx0*, VGfloat *dy0*, VGfloat *dx1*, VGfloat *dy1*, VGfloat *dx2*, VGfloat *dy2*, VGfloat *dx3*, VGfloat *dy3*, VGfloat * *matrix*)

vguErrorCode **vguComputeWarpQuadToQuad**(VGfloat *sx0*, VGfloat *sy0*, VGfloat *sx1*, VGfloat *sy1*, VGfloat *sx2*, VGfloat *sy2*, VGfloat *sx3*, VGfloat *sy3*, VGfloat *dx0*, VGfloat *dy0*, VGfloat *dx1*, VGfloat *dy1*, VGfloat *dx2*, VGfloat *dy2*, VGfloat *dx3*, VGfloat *dy3*, VGfloat* *matrix*)