
AN APPLIED MATERIALS COMPANY

# NEMA®| GFX Extensions

# Vector Graphics

Version v1.1.5

Part Number: D-NVG

November 10, 2023

## Disclaimer

This document is written in good faith with the intent to assist the readers in the use of the product. Circuit diagrams and other information relating to Think Silicon S.A products are included as a means of illustrating typical applications. Although the information has been checked and is believed to be accurate, no responsibility is assumed for inaccuracies. Information contained in this document is subject to continuous improvements and developments.

Think Silicon S.A products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of Think Silicon S.A. will be fully at the risk of the customer.

Think Silicon S.A. disclaims and excludes any and all warranties, including without limitation any and all implied warranties of merchantability, fitness for a particular purpose, title, and infringement and the like, and any and all warranties arising from any course or dealing or usage of trade.

This document may not be copied, reproduced, or transmitted to others in any manner. Nor may any use of information in this document be made, except for the specific purposes for which it is transmitted to the recipient, without the prior written consent of Think Silicon S.A. This specification is subject to change at anytime without notice.

Think Silicon S.A. is not responsible for any errors contained herein. In no event shall Think Silicon S.A. be liable for any direct, indirect, incidental, special, punitive, or consequential damages; or for loss of data, profits, savings or revenues of any kind; regardless of the form of action, whether based on contract; tort; negligence of Think Silicon S.A or others; strict liability; breach of warranty; or otherwise; whether or not any remedy of buyers is held to have failed of its essential purpose, and whether or not Think Silicon S.A. has been advised of the possibility of such damages.



Questions or comments may be directed to:
Think Silicon S.A
Suite B8
Patras Science Park
Rion Achaias 26504, Greece
web: http://www.think-silicon.com
email:info@think-silicon.com
Tel: +30 2610 911543
Fax: +30 2610 911544

# Contents

# 1 Overview

NEMA®| GFX Vector Graphics Extensions (VG Extensions) is a set of software extensions to accelerate vector graphics applications on NEMA®| pico. It offers a minimal yet powerful set of API calls that work with NEMA®| GFX as depicted in Figure 1. By taking advantage of NEMA®| GFX, the VG extensions are able to perform the necessary vector graphics operations such as geometry tessellation, bezier curve rasterization, stencil and masking and many more, providing minimum memory footprint along with true hardware acceleration, as NEMA®| pico is utilized for rendering the desired context.

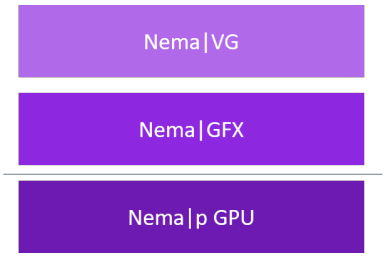


**Figure 1: NEMA®| GFX Vector Graphics Extensions**

The following sections contain:

- fundamental concepts on vector graphics and how these concepts are embodied in NEMA®| GFX Vector Graphics Extensions
- a simple example that can be considered as the "Hello World" of NEMA®| GFX Vector Graphics Extensions
- some utilities that can simplify VG applications significantly

A detailed API documentation of NEMA®| GFX Vector Graphics Extensions is included in the last section of this document.

# 2 Basic Concepts

This section contains fundamental concepts on vector graphics. These concepts are in the core of NEMA®| GFX Vector Graphics Extensions and their basic understanding will help to better utilize NEMA®| GFX Vector Graphics Extensions.

## 2.1 Vector vs Raster Graphics

Raster graphics elements are described by rectangular grids of pixels that have fixed resolution (width and height). In contrast to this concept, vector graphics elements consist of sets of points that are connected by lines and curves described by mathematical formulas. The connected points form a `path`. The main advantage of vector over raster graphics is in scaling up or down, where there is no aliasing (Figure 2).

**Figure 2: Zoom-in, vector vs raster graphics**

Furthermore, a raster element requires the storage of all its pixels, occupying a memory area with size: width x height x pixel_size. The higher the resolution of an image the more memory it requires in order to be stored. On the other hand, storing a vector element in memory requires always the same amount of memory regardless the resolution. Since a vector element is described by points and mathematical equations, storing it can frequently be more memory efficient than storing individual pixels, even for relatively small resolutions. However, this approach has the cost of computing the points based on their connecting curves for the rendering scene. This requires processing power as the necessary computations need to be performed either on the CPU or on a hardware accelerator.

## 2.2 Path

Vectors consist of one or more points (vertices). When the points of a vector element are connected, they form a `path`. In NEMA®| GFX Vector Graphics Extensions, the path

needs always to be a closed geometry. The simplest way to connect two points is by connecting them using a straight line. NEMA®| GFX Vector Graphics Extensions support path segments that are described by:

- A straight line
- A closed polygon (series of straight lines connected in closed geometry)
- An open polyline (series of straight lines connected in open geometry)
- A quadratic Bezier curve (one control point)
- A smooth quadratic Bezier curve (one control point)
- A cubic Bezier curve (two control points)
- A smooth cubic Bezier curve (two control points)
- An elliptical arc (three control points)

Bezier curves are parametric curves defined by the end points and the control points. The quadratic and cubic Bezier curves that are supported by NEMA®| GFX Vector Graphics Extensions are depicted in Figure 3 and Figure 4 respectively.

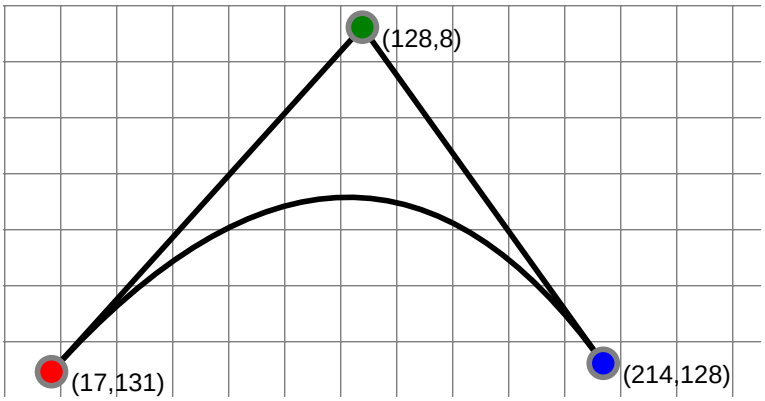


**Figure 3: Quadratic Bezier curve (two end points and one control point)**

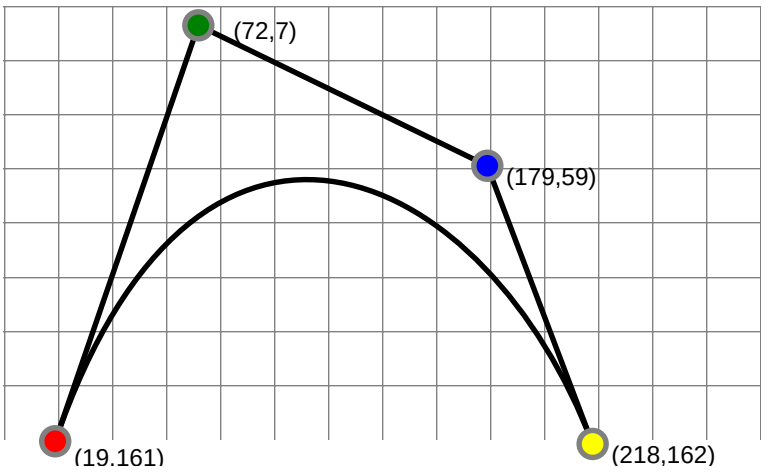


**Figure 4: Cubic Bezier curve (two end points and two control points)**

With Bezier curves, you can specify the end and control points. The curve that connects the two end points is derived based on them.

Smooth control points for cubic and quadratic Bezier are calculated with the following formulas:

1. When a Bezier segment follows a previous Bezier segment, the following equation is used:

```
x1 = 2 * ox - px
y1 = 2 * oy - py
```

   where `x1, y1` is the new control point, `ox, oy` is the previous data point and `px, py` is the previous control point.
2. In any other case:

```
x1 = ox
y1 = oy
```

In NEMA®| GFX VG, the shape that a path defines is stored inside a vertex buffer which is created implicitly using the function:

```
void nema_vg_path_set_shape(nema_vg_path_t* path, int seg_size, uint8_t* seg, int da►
ta_size, nema_vg_float_t* data);
```

Elliptical arc segments join a pair of points with a section of an ellipse with given horizontal and vertical axes and a rotation angle (in degrees). Given these control points, there are four possible arcs distinguished by their direction around the ellipse (clockwise or counter-clockwise) and whether they take the smaller or larger path around the ellipse. Figure 5 shows the two possible ellipses.

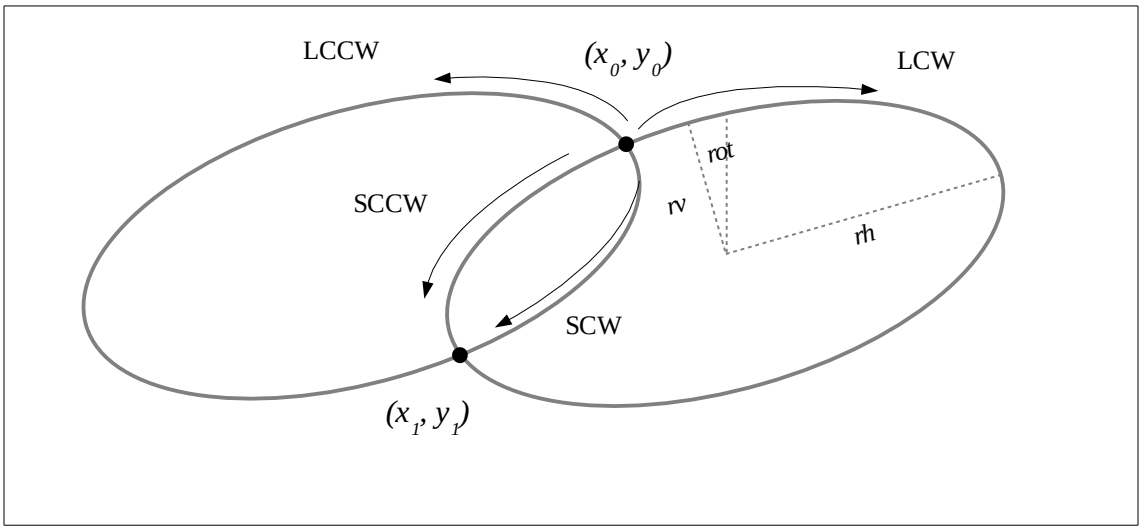


**Figure 5: Elliptical Arcs***

**Note:** All the above segments support also relative coordinates.

`seg_size` segments with respective `data` (eg. segment points) are added to `path`. NEMA®| GFX VG currently supports the following path segments:

- NEMA_VG_PRIM_MOVE (start segment)
- NEMA_VG_PRIM_CLOSE (end segment)
- NEMA_VG_PRIM_LINE (straight line)
- NEMA_VG_PRIM_HLINE (horizontal line)
- NEMA_VG_PRIM_VLINE (vertical line)
- NEMA_VG_PRIM_POLYGON (closed shape polygon)
- NEMA_VG_PRIM_POLYGON_REL (relative closed shape polygon)
- NEMA_VG_PRIM_POLYLINE (open shape polyline)
- NEMA_VG_PRIM_POLYLINE_REL (relative open shape polyline)
- NEMA_VG_PRIM_BEZIER_QUAD (quadratic Bezier curve)
- NEMA_VG_PRIM_BEZIER_CUBIC (cubic Bezier curve)
- NEMA_VG_PRIM_BEZIER_SQUAD (smooth quadratic Bezier curve)
- NEMA_VG_PRIM_BEZIER_SCUBIC (smooth cubic Bezier curve)
- NEMA_VG_PRIM_MOVE_REL (relative start segment)
- NEMA_VG_PRIM_LINE_REL (relative straight line)
- NEMA_VG_PRIM_HLINE_REL (relative horizontal line)
- NEMA_VG_PRIM_VLINE_REL (relative vertical line)
- NEMA_VG_PRIM_BEZIER_QUAD_REL (relative quadratic Bezier curve)
- NEMA_VG_PRIM_BEZIER_CUBIC_REL (relative cubic Bezier curve)

* OpenVG Specification version 1.1 document

- NEMA_VG_PRIM_BEZIER_SQUAD_REL (relative smooth quadratic Bezier curve)
- NEMA_VG_PRIM_BEZIER_SCUBIC_REL (relative smooth cubic Bezier curve)
- NEMA_VG_PRIM_SCCWARC (small counter-clockwise arc)
- NEMA_VG_PRIM_SCWARC (small clockwise arc)
- NEMA_VG_PRIM_LCCWARC (large counter-clockwise arc)
- NEMA_VG_PRIM_LCWARC (large clockwise arc)
- NEMA_VG_PRIM_SCCWARC_REL (relative small counter-clockwise arc)
- NEMA_VG_PRIM_SCWARC_REL (relative small clockwise arc)
- NEMA_VG_PRIM_LCCWARC_REL (relative large counter-clockwise arc)
- NEMA_VG_PRIM_LCWARC_REL (relative large clockwise arc)

The first segment of a path should be `NEMA_VG_PRIM_MOVE` which indicates the start of a path. If the first segment is not a `NEMA_VG_PRIM_MOVE` segment, the path will begin from point (0, 0). Similarly, the last segment should be `NEMA_VG_PRIM_CLOSE`. If the last segment is not a `NEMA_VG_PRIM_CLOSE` segment, the path will close implicitly using a straight line to the start point. Relative path segments imply that the data of the current segment (control and end point vertices), are relative to the previous path.

Each of the segments mention earlier is related to its data (eg. coordinate points (x, y) ). The following table summarizes the data count that each segment needs to have in order to be valid.

| **Segment** | **Data count** |
|---|---|
| NEMA_VG_PRIM_CLOSE | 0 (no data) |
| NEMA_VG_PRIM_MOVE | 2 (x,y start point) |
| NEMA_VG_PRIM_LINE | 2 (x,y end point) |
| NEMA_VG_PRIM_HLINE | 1 (x end point) |
| NEMA_VG_PRIM_VLINE | 1 (y end point) |
| NEMA_VG_PRIM_BEZIER_QUAD | 4 (x,y end point - x,y control point) |
| NEMA_VG_PRIM_BEZIER_CUBIC | 6 (x,y end point - x,y control point 0 - x,y control point 1) |
| NEMA_VG_PRIM_BEZIER_SQUAD | 2 (x,y end point) |
| NEMA_VG_PRIM_BEZIER_SCUBIC | 4 (x,y end point - x,y control point 0) |
| NEMA_VG_PRIM_ARC | 5 (rx - ry - rotation - x,y end point) |
| NEMA_VG_PRIM_POLYGON | variable (a number denoting the polygon points, followed by the x,y coordinates of the points) |

| | |
|---|---|
| NEMA_VG_PRIM_POLYLINE | variable (a number denoting the polyline points, followed by the x,y coordinates of the points) |

**Note:** When defining polygon or polyline segments with the `nema_vg_path_set_shape()` function, the provided data consist of the coordinate points of the segments (array of floats).

A path (consecutive segments that form a closed curve), can then be set to a `path` object using the function:

```
void nema_vg_path_set_shape(nema_vg_path_t* path, int seg_size, uint8_t* seg, int da►
ta_size, nema_vg_float_t* data);
```

Furthermore, a path can be transformed using a transformation matrix. Affine transformations (3x3 matrices) are currently supported. The following function is used to set the transformation matrix of a path (arguments are omitted):

```
void nema_vg_path_set_matrix();
```

### 2.2.1 Predefined Shapes

It is possible with the use of primitive path drawing functions to draw any shape at all, however complicated. In addition, NEMA®| GFX provides functions for drawing some common predefined shapes. These shape drawing functions are optimized for speed.

The supported predefined shapes include rectangles, ellipses, circles and rings. The following are the functions used for creating these shapes:

```
uint32_t nema_vg_draw_rect(float x, float y, float width, float height,
                           nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)
uint32_t nema_vg_draw_rounded_rect(float x, float y, float width, float height,
                           float rx, float ry,
                           nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)
uint32_t nema_vg_draw_ellipse(float cx, float cy, float rx, float ry,
                           nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)
uint32_t nema_vg_draw_circle(float cx, float cy, float r,
                           nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)
uint32_t nema_vg_draw_line(float x1, float y1, float x2, float y2,
                           nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)
uint32_t nema_vg_draw_ring(float cx, float cy, float ring_radius,
                         float angle_start, float angle_end, NEMA_VG_PAINT_HANDLE paint)
```

Figure 6 contains shapes drawn with these functions. Beginning from top to bottom, there are examples with solid fill, gradient fill, conical fill, radial gradient fill, and texture fill.

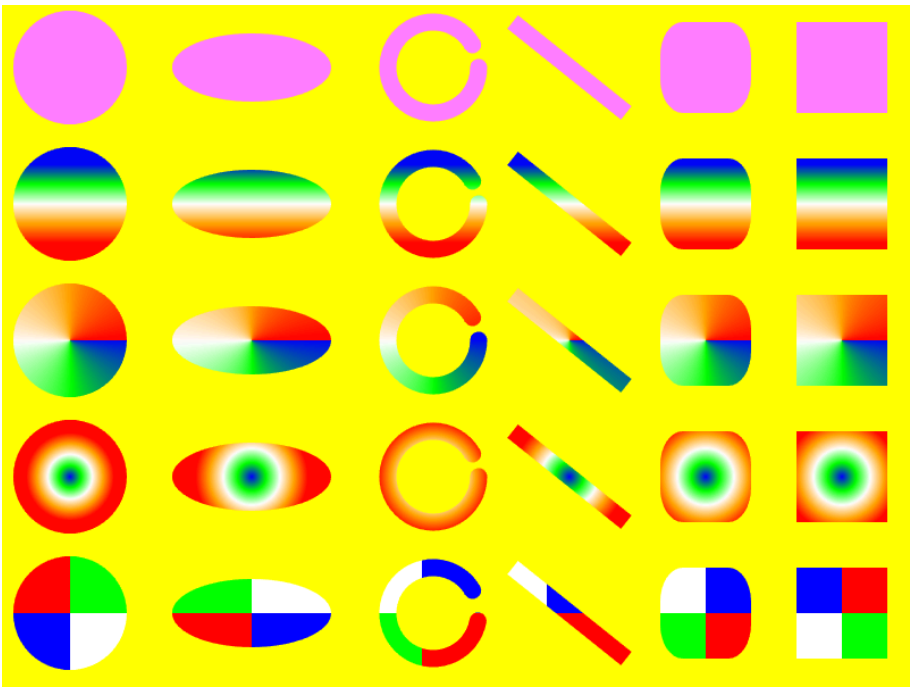

**Figure 6: Examples of drawn predefined shapes**

### 2.2.2 Fonts

NEMA®| GFX VG supports text rendering using vector fonts. A vector font asset can be created using the off-line tool `nema_vg_font_convert`. This tool takes as input a True-Type (TTF) file and converts it to structs that can be used in NEMA®| GFX VG. By converting a TTF file using `nema_vg_font_convert` tool, two files will be created; a header file (.h) which contains the declaration of the font struct, while the source file (.c) contains the definition of the font structs and its sub-components. The source file contains information regarding each character that is included in the font. Characters in vector fonts are described by closed paths, therefore the NEMA®| GFX VG font structs keep the path information (segments and data) of all the included characters.

Prior to drawing text, NEMA®| GFX VG library needs to have a valid bound font. A font can be bound using the following function:

```
void nema_vg_bind_font(nema_vg_font_t *font);
```

The desired font size can be set afterwards using:

```
void nema_vg_set_font_size(float size);
   void nema_vg_bind_font(nema_vg_font_t *font);
```

After the font is bound in the library and its size has been set, text can be rendered using `nema_vg_print()` function:

```
void nema_vg_print(NEMA_VG_PAINT_HANDLE paint, const char *str, float x, float y,
 float w, float h, uint32_t align, nema_matrix3x3_t m);
```

The above function takes as arguments the paint object (more details in Paint) that contains information about how the text should be rendered (eg. fill with a single color, fill with gradient, outlines etc.), the text to be rendered, the position and maximum allowed width and height, alignment flag and a transformation matrix. For more information please refer to the documentation of `nema_vg_font.h` file as described in Vector Graphics API. Figure 7 depicts an example of text drawing using a vector font. The same font is used for drawing all three texts, however different font size and transformation matrix is set to each one.

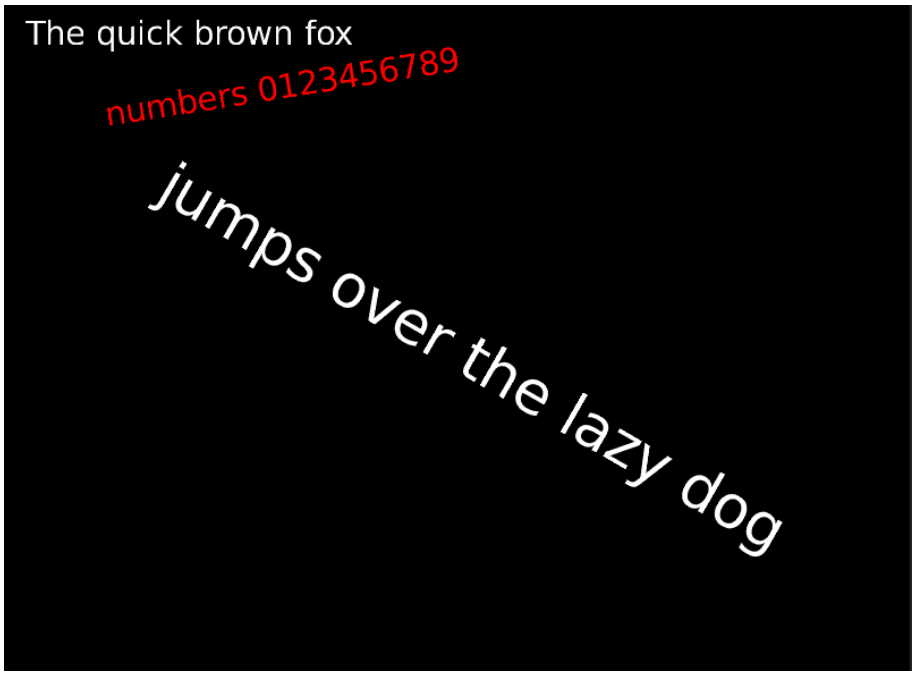


**Figure 7: Text drawing using vector font**

In Figure 7, text is rendered as vector paths filled with a singe color. Nevertheless, depending on the setup of the paint object, text can be rendered in many ways as depicted in Figure 8. In this figure, text is rendered using linear gradient on filled paths as well as on stroked paths (letter outlines). Such rendering, requires that the gradient and/or the stroked width respectively, have been setup by the user prior to executing `nema_vg_print` function.


**Figure 8: Text drawing using vector font**

### 2.2.3 Large Coordinates

As mentioned earlier, the data of a *path* contain coordinate points. In some configurations, due to hardware limitations, the coordinates of the geometry (*path*) need to be inside a certain range. Function `nema_vg_get_coord_limits()` returns the minimum and maximum values of the available range. Paths with coordinates outside of the given range can be handled in the following two ways. The first one is to discard them and do not draw them at all (default option). In this case, calling `nema_vg_draw_path()` function will return the respective error code (*NEMA_VG_ERR_COORDS_OUT_OF_RANGE*). The second option, is to clip these paths in order to fit within the available range and then draw the clipped paths. Clipping such paths, requires to create new paths with new segments and vertices and this needs to allocate memory in order to store them. Therefore, the user needs to allocate memory buffers (one for the segments and one for the data). Once these buffers are created, they need to be bound to NEMA®| GFX VG library. This can be done using `nema_vg_bind_clip_coords_buf()`, so that the library can use them internally. Besides, allocating the buffers needed for storing the clipped path, the option for handling large coordinates must be set as well (using `nema_vg_handle_large_coords()` function). The following code snippet depicts a scenario where a path with large coordinated is clipped:

```
    //Allocate memory using nema_host_malloc()
    uint8_t *segs = (uint8_t *)nema_host_malloc(2048); //2KB for the segments
    nema_vg_float_t *data = (nema_vg_float_t *)nema_host_malloc(8024); //8KB for the
 data

    uint32_t err = nema_vg_bind_clip_coords_buf(segs, segs_size_bytes, data, da►
ta_size_bytes);

    nema_vg_handle_large_coords(1U, 1U); // enable large coordinates handling and in►
ternal memory allocation

    (void) nema_vg_draw_path(path, paint);

    nema_vg_unbind_clip_coords_buf();

    nema_host_free((void*)segs);
    nema_host_free((void*)data);
```

Handling large coordinates is controlled using `nema_vg_handle_large_coords()` function. Its first argument controls if the large coordinates should be handled. In case this argument is '0', large coordinates will not be handled and respective paths will be discarded and not drawn at all. If this is set to be '1', NEMA®| GFX VG will attempt to make use of the bound *clipped coordinates buffers* (allocated by the user). If the clipped path (segments and data) fits inside these buffers, they will be used for storing the clipped path. If however the clipped path does not fit inside these buffers, then the second argument (*allow_internal_alloc*) is used to control the rendering flow. In this case, if this argument is '0' a path with large coordinates will be discarded. Otherwise, NEMA®| GFX VG library will allocate memory internally in order to store the clipped path, draw it and then free the the allocated memory.

**Note:** The previous snippet uses `nema_host_malloc()` for memory allocation. It is up to the application level (user) to decide how memory allocations (and deallocations) should be performed (eg. using stack allocated arrays or other memory allocators). The rest of of the drawing functions (`nema_vg_print()`, `nema_vg_draw_tsvg()` and predefined shapes) behave similarly.

## 2.3 Paint

Another important concept of vector graphics is the `paint` object. This is responsible for drawing a `path`. On each path, the paint object can be applied with certain fill rules in order to determine how the path outline and interior areas will be handled. A paint object in NEMA®| GFX VG can be created using the following function:

```
NEMA_VG_PAINT_HANDLE nema_vg_paint_create();
```

Once a paint object is created, its type needs to be set. The types that are currently supported by NEMA®| GFX VG are:

**NEMA_VG_PAINT_COLOR**
: fills the path's interior with a specific color

**NEMA_VG_PAINT_GRAD_LINEAR**
: given two points p0, p1 it fills the interior path from p0 to p1 with linear gradient form p0's color to p1's color

**NEMA_VG_PAINT_TEXTURE**
: fills the path's interior with a texture or a LUT texture. A texture is set with `nema_vg_paint_set_tex()`, and a LUT texture is set with `nema_vg_pain_set_lut_tex()`. The texture orientation can be handled by providing a transformation matrix

**NEMA_VG_PAINT_GRAD_RADIAL**
: fills the path's interior with radial gradient, based on values of center point, radius, stops, and stops color values

**NEMA_VG_PAINT_GRAD_CONICAL**
fills the path's interior with conical gradient, based on values of center point, stops, and stops color values

**Note:** The radial gradient feature is available only if the GPU (NEMA®| pico ) has also this feature enabled.

The type of the paint object can be set using the following function (the arguments are omitted):

```
void nema_vg_paint_set_type();
```

Depending on the paint type set, you may need to set additional parameters. For instance, when the paint object's type is set as `NEMA_VG_PAINT_COLOR`, the desired color for the paint object is also required. You can set this by calling `nema_vg_paint_set_paint_color()`. If this function is not called, the default color (black) will be used.

When using gradient, first you need to create the gradient buffer, by calling `nema_vg_grad_create()`. Then you need to set the gradient buffer to the desired gradient by calling `nema_vg_grad_set()`. The values `stops color` and `stops` denote the colors to be used in the gradient and where they stop respectively. `stops` can take float values in the range [0, 1]. By adjusting their value, you can manipulate how the colors are displayed within the gradient. The value 0 is the start (p0 point for linear gradient) or the center (radial or conical gradient) and value 1 is the end (p1 point for linear gradient),the radius point (radial gradient) or 360 degree angle (conical gradient). In case stops values are out of range or in incoherent order, they are omitted and a gradient starting from 0 with black color and ending at 1 with white color will be set. In case stops values exceed `NEMA_VG_PAINT_MAX_GRAD_STOPS`, the exceeding values will be ignored.

You also need to set the sampling mode, as it defines how the paint object should behave when the path is outside of the end (linear gradient), radius point (radial) or angle(conical gradient). The above parameters can be set by using the following functions (arguments are omitted):

```
void nema_vg_paint_set_grad_linear();
void nema_vg_paint_set_grad_radial();
void nema_vg_paint_set_grad_conical();
void nema_vg_paint_set_paint_color();
void nema_vg_paint_set_stroke_width();
void nema_vg_paint_set_tex_matrix();
void nema_vg_paint_set_tex();
void nema_vg_paint_set_lut_tex();
```

## 2.4 Context

Performing a drawing operation requires various information to be available at drawing time. Such pieces of information are the drawing surface, the stencil buffer, the mask operation (if enabled), the blending mode, a look-up table, the fill rule and the quality level. These are grouped in the `context` object which is an opaque object to the user. Although that the user can not access this object directly, it is affected by using specific functions described in this section.

**Note:** Before attempting to perform a draw operation, you need to initialize the NEMA®| GFX VG library first. Initialization takes place using one of the functions `void nema_vg_init(int width, int height)`, `void nema_vg_init_stencil_pool(int width, int height, int pool)` or `void nema_vg_init_stencil_prealloc(int width, int height, nema_buffer_t stencil_bo)`. When using `nema_vg_init_stencil_prealloc`, the stencil buffer must have a size RESX*RESY, where RESX and RESY are the framebuffer dimensions.

You can also provide a global matrix to apply a transformation to the entire geometry. This is done by calling `nema_vg_set_global_matrix()` with the desired transformation matrix. The feature is deactivated by calling `nema_vg_reset_global_matrix()`.

### 2.4.1 Error Handling

When an error occurs at runtime, it is recorded in the `context` object. All possible error codes can be found in Vector Graphics API. The error code can be retrieved using the following function:

```
uint32_t nema_vg_get_error(void);
```

### 2.4.2 Fill Rules

Path drawing requires the use of a fill rule. The fill rule determines how *paint* is applied to the interior areas of the *path*. NEMA®| GFX VG supports the following fill rules:

- `NEMA_VG_STROKE`
- `NEMA_VG_FILL_EVEN_ODD`
- `NEMA_VG_FILL_NON_ZERO` (default)

The fill rule can be set be running the following function:

```
void nema_vg_set_fill_rule(uint8_t fill_rule);
```

The *stroke* fill rule specifies that only the outline of the path will be drawn. Figure 9 illustrates the difference between the even-odd and the non-zero fill rules:

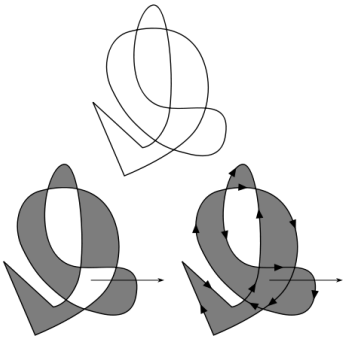

**Figure 9: Upper part: path, lower part: even-odd (left), non-zero (right)** †

Outlines also support variable width. Joins are implemented as bevel, and this fill rule is supported only for solid color fill. In case of stroking, the stroke width is assigned to the `paint` object.

**Figure 10: Stroking with width greater than 1**

### 2.4.3 Rendering Quality and Blending

You can control the rendering quality with the quality modes. These quality modes favor performance over quality or vice versa. The quality modes that are currently supported in NEMA®| GFX VG are the following:

**NEMA_VG_QUALITY_BETTER**
: Better rendering quality (default option, balances rendering quality and performance)

**NEMA_VG_QUALITY_FASTER**
: Faster rendering quality (favors performance over rendering quality)

**NEMA_VG_QUALITY_MAXIMUM**
: Maximum rendering quality (favors rendering quality over performance)

**NEMA_VG_QUALITY_NON_AA**
: Rendering quality without anti-aliasing

† Source Wikipedia

The quality level is controlled by running this function before starting a rendering operation (draw a path):

```
void nema_vg_set_quality(uint8_t quality);
```

Another parameter that affects a rendering operation is the blending mode, which configures the way rendering is done. For example, a path can be blended in the framebuffer or drawn over it. The default blending mode is: NEMA_BL_SRC_OVER. For more information regarding the blending modes, please refer to *NEMA®| GFX API User Manual*. The blending mode can be set using the following function:

```
void nema_vg_set_blend(uint32_t blend);
```

### 2.4.4 Masking

Masking is an important feature supported in NEMA®| GFX VG. A `mask` object contains the alpha channel (opacity) that will be used for blending a vector object (for example, a `path`). This is depicted in Figure 11 where the mask object (upper right picture) is applied to the paths (upper left picture) that need to be drawn. The result can be viewed in the bottom part of the figure.

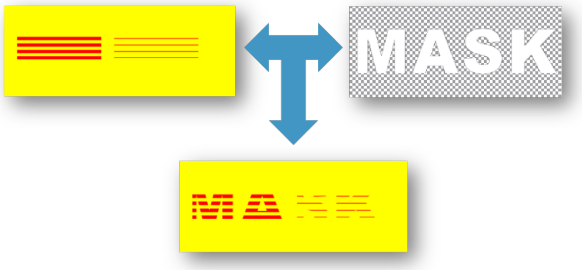


**Figure 11: Example of applying a mask object**

The mask object is an image object with a color format A1, A2, A4 or A8. When setting a mask object of a different color format, an error code is returned by the `nema_vg_set_mask` function. This function is used to set the mask object.

```
uint32_t nema_vg_set_mask(nema_img_obj_t *mask_obj);
```

Prior to setting the mask object, the `masking` parameter of the `context` needs to be enabled as well. This is performed using the following function:

```
void nema_vg_masking(uint8_t masking);
```

Running this function using 1U value in the `masking` argument, will enable masking, while using 0U value will dissable it.

Moreover, it is also possible to move the mask object to the area of interest with the following function:

```
void nema_vg_set_mask_translation(float x, float y);
```

### 2.4.5 Global Transformation

You can provide a global matrix to apply an affine transformation to the entire geometry. This is done by calling: `nema_vg_set_global_matrix()` with the desired transformation matrix.

By doing so, all subsequent drawing operations will use this matrix. The feature is deactivated by calling `nema_vg_reset_global_matrix()`.

## 2.5 Memory Allocations

Apart from the functionality of each component (path, paint and object) found in NEMA®| GFX VG that is presented in previous chapters, it is necessary to clarify any implicit memory allocation, so you can easily estimate the necessary memory for a NEMA®| GFX VG application and perform any optimizations, if needed.

The NEMA®| GFX VG library is initialized by calling one of the following functions:

```
void nema_vg_init(int width, int height);

void nema_vg_init_stencil_pool(int width, int height, int pool)
```

Calling these functions implies some memory allocations. The first one is for the `stencil buffer`. This buffer is used internally by the library in order to render the specified `path`. Provided that the biggest possible path that may be encountered in an NEMA®| GFX VG application could occupy the full resolution of the framebuffer, the `stencil buffer` occupies `width` x `height` bytes, where `width`, `height` is the resolution of the framebuffer.

The second one is for a look-up table. This look-up table is part of the `context` and is necessary internally, when drawing anti-aliased paths using the `non-zero` fill rule; it occupies 256 bytes.

NEMA®| GFX VG contains a special module used for rendering TSVG formated files (for more information regarding TSVG format, please check *Pixpresso Starting Guide* and *NemaVG TSVG Elements*). Calling the initialization functions, the TSVG renderer gets initialized as well. Its initialization implies three memory allocations; for the paint, path and gradient objects that are used internally.

It must be noted that the memory allocations mentioned above, take place at library initialization time and all the occupied memory gets deallocated by deinitializing the library ( calling *nema_vg_deinit()* function).

In addition, large coordinate handling (see Large Coordinates for more information) may allocate memory internally for handling paths with large coordinates, depending on the user preference. This is controlled via `nema_vg_handle_large_coords()` function. In this case, memory will be deallocated also internally.

Any object created explicitly by the user, using the `create` functions, (such as `nema_vg_path_create`, `nema_vg_paint_create`, or `nema_vg_grad_create`) need also to be explicitly destroyed using the respective `destroy` function (for instance `nema_vg_path_destroy`, `nema_vg_paint_destroy`, or `nema_vg_grad_destroy`).

# 3 Hello NEMA®| GFX VG

This section provides a simple application that demonstrates how NEMA®| GFX VG Extensions is used in practice. A star-path is created and painted using various configurations (fill rules and paint type) as seen in Figure 12.

The full code for this example can be found under `NemaGFX_SDK/examples/NemaVG/paint_example` folder. The following chapters provide snippets of this code with relative explanations.

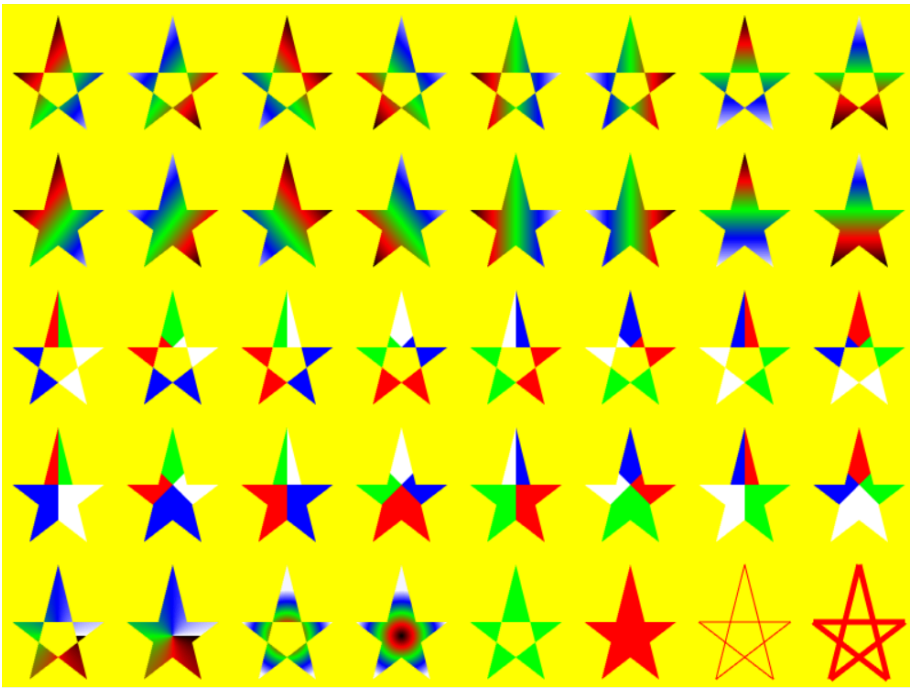

**Figure 12: Various paint configurations applied to a star path. Fist row: even-odd fill rule with linear gradient paint, Second row: non-zero fill rule, with linear gradient paint, Third row: even-odd fill rule with a texture paint and different orientation matrices, Fourth row: non-zero fill rule with a texture paint and different orientation matrices, Fifth row: even-odd and non-zero fill rules with conical and radial gradient paint, fill paint and stroke.**

## 3.1 Library Initialization

The program starts by initializing the graphics system (NEMA®| GFX library) and display controller. After the NEMA®| GFX library is initialized, the memory needed for the framebuffer is allocated and initialized (function `load_objects()`). In this example, the framebuffer is the variable `fb`. Then a command list is created and bound as the current

command list. The framebuffer is cleared with yellow color, and the clipping rectangle is set. Finally, the VG library is initialized by calling `nema_vg_init()`.

```
//Initialize NemaGFX
if ( nema_init() != 0 ) {
    return -1;
}

//Initialize NemaDC
if ( nemadc_init() != 0 ) {
    return -2;
}

load_objects();

//Format        | Pixclock | RESX | FP | SYNC | BP | RESY | FP | SYNC | BP
//800x600, 60Hz | 40.000   | 800  | 40 | 128  | 88 | 600  | 1  | 4    | 23
nemadc_timing(800, 40, 128, 88, 600, 1, 4, 23);
nemadc_set_layer(0, &dc_layer);

nema_cmdlist_t cl  = nema_cl_create();
nema_cl_bind(&cl);

//clear fb
nema_bind_dst_tex(fb.bo.base_phys, fb.w, fb.h, fb.format, fb.stride);
nema_set_clip(0, 0, RESX, RESY);
nema_clear(nema_rgba(0xff,0xff,0x00,0xff)); //yellow

nema_vg_init(RESX, RESY);
```

## 3.2 Application Initialization

After the library initialization, the next step is to create the NEMA®| GFX VG objects and configure them properly. The first object is the `paint` object. After `paint` is created, its type is set along with the `context's` fill rule and quality parameters. Then a path object is created with the function `nema_vg_path_create()`. Path shape is set by `nema_vg_path_set_shape()`. The matrices that will be used afterwards for transforming the path are defined here as well. Also the gradient buffer is created by calling `nema_vg_grad_create()`.

```
NEMA_VG_PAINT_HANDLE paint =  nema_vg_paint_create();
nema_vg_paint_set_type(paint, NEMA_VG_PAINT_GRAD_LINEAR);

nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);

NEMA_VG_PATH_HANDLE path = nema_vg_path_create();
NEMA_VG_GRAD_HANDLE gradient = nema_vg_grad_create();
nema_matrix3x3_t m_path;
nema_matrix3x3_t m_paint;

nema_vg_path_set_shape(path, 6, cmds_star, 12, data_star_small);
```

### 3.3 Draw Operation

Once the NEMA®| GFX VG objects are ready, they can be used to perform draw operations. The first step is to draw the triangles as seen in the first row (even-odd fill rule) of Figure 12. The position of each triangle needs to be calculated which means that the path will be translated to the calculated position using a matrix-based transformation. Furthermore, the linear gradient is set. The fill rule needs to be set once, thus it is set outside of the loop. Inside the loop, the position of each triangle is calculated (x_pos and y_pos) and respective matrix is applied to the path. The following snippet summarizes these operations:

```
  nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);
  for ( int i = 0; i < 8 ; i++) {
    x_pos= star_dist_x*i;
    y_pos = 0;

    nema_mat3x3_load_identity(m_path);
    nema_mat3x3_translate(m_path, x_pos, y_pos);
    nema_vg_path_set_matrix(path, m_path);
    nema_vg_paint_set_grad_linear(paint, gradient, grad[i][0] + x_pos, grad[i][1] +
 y_pos, grad[i][2] + x_pos, grad[i][3] + y_pos);
    nema_vg_draw_path(path, paint);
}
```

Similarly, the drawing operations for the stars of the second raw (non-zero fill rule) would be as in the next snippet. The same settings, except the vertical position (y_pos) and fill rule (non_zero in this case).

```
  nema_vg_set_fill_rule(NEMA_VG_FILL_NON_ZERO);
  for ( int i = 0; i < 8 ; i++) {
    x_pos= star_dist_x*i;
    y_pos = star_dist_y;
    nema_mat3x3_load_identity(m_path);
    nema_mat3x3_translate(m_path, x_pos, y_pos);
    nema_vg_path_set_matrix(path, m_path);
    nema_vg_paint_set_grad_linear(paint, gradient, grad[i][0] + x_pos, grad[i][1] +
 y_pos, grad[i][2] + x_pos, grad[i][3] + y_pos);
    nema_vg_draw_path(path, paint);
}
```

Drawing a texture in a path, requires that the paint type as well as the texture will be set accordingly. The next snippet contains the code for drawing the first triangle of the third row. This is a textured path, with even-odd fill rule. The rest of the textured triangles (third and fourth rows) are also drawn in a similar manner.

```
  int star_x_off = 10, star_y_off = 50;
  nema_vg_paint_set_type(paint, NEMA_VG_PAINT_TEXTURE);
```

```
  nema_vg_paint_set_tex(paint, &ref_img);

  x_pos = 0;
  y_pos = star_dist_y*2;
  nema_mat3x3_load_identity(m_path);
  nema_mat3x3_translate(m_path, x_pos, y_pos);
  nema_vg_path_set_matrix(path, m_path);
  nema_mat3x3_load_identity(m_paint);
  nema_mat3x3_translate(m_paint, x_pos + star_x_off, y_pos + star_y_off);
  nema_vg_paint_set_tex_matrix(paint, m_paint);
  nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);
  nema_vg_draw_path(path, paint);
```

Drawing the same path using conical gradient requires that the gradient's parameters are configured accordingly, as depicted in the following snippet:

```
  float x_star_center;
  float y_star_center;
  x_pos = 0.f;
  x_star_center = x_pos + star_length/2 + star_x_off;
  y_star_center = y_pos + star_height/2 + star_y_off + 12.f;
  nema_mat3x3_load_identity(m_path);
  nema_mat3x3_translate(m_path, x_pos, y_pos);
  nema_vg_path_set_matrix(path, m_path);
  nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);
  nema_vg_paint_set_type(paint, NEMA_VG_PAINT_GRAD_CONICAL);
  nema_vg_paint_set_grad_conical(paint, gradient, x_star_center, y_star_center, NE►
MA_TEX_CLAMP  | NEMA_FILTER_BL);
  nema_vg_draw_path(path, paint);
```

The drawing operation for the radial gradient is also performed in a similar manner:

```
x_pos = STAR_DIST_X*2;
x_star_center = x_pos + star_length/2 + star_x_off;
y_star_center = y_pos + star_height/2 + star_y_off + 12.f;
nema_mat3x3_load_identity(m_path);
nema_mat3x3_translate(m_path, x_pos, y_pos);
nema_vg_path_set_matrix(path, m_path);
nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);
nema_vg_paint_set_type(paint, NEMA_VG_PAINT_GRAD_RADIAL);
nema_vg_paint_set_grad_radial(paint, gradient, x_star_center, y_star_center,
 star_length/2, NEMA_TEX_CLAMP | NEMA_FILTER_BL);
nema_vg_draw_path(path, paint);
```

The stops and colors used in the previous cases with gradient (linear, conical and radial) can be configured in the gradient buffer by calling `nema_vg_grad_set()`, as in the following snippet:

```
#define COLOR_STOPS       5

float stops[COLOR_STOPS] = {0.0f, 0.25f, 0.50f, 0.75f, 1.0f};
color_var_t colors[COLOR_STOPS] = {{0  , 0  , 0  , 255}, //black
                                   {255, 0  , 0  , 255}, //red
                                   {0  , 255, 0  , 255}, //green
                                   {0  , 0  , 255, 255}, //blue
                                   {255, 255, 255, 255}}; //white
nema_vg_grad_set(gradient, COLOR_STOPS, stops, colors);
```

A colorized path as it can be seen in the last row of Figure 12, can be drawn similarly, by setting the `paint color` instead of the texture.

```
    int star_x_off = 10, star_y_off = 50;
    nema_vg_paint_set_type(paint, NEMA_VG_PAINT_TEXTURE);
    nema_vg_paint_set_tex(paint, &ref_img);

    x_pos = 0;
    y_pos = star_dist_y*2;
    nema_mat3x3_load_identity(m_path);
    nema_mat3x3_translate(m_path, x_pos, y_pos);
    nema_vg_path_set_matrix(path, m_path);
    nema_mat3x3_load_identity(m_paint);
    nema_mat3x3_translate(m_paint, x_pos + star_x_off, y_pos + star_y_off);
    nema_vg_paint_set_tex_matrix(paint, m_paint);
    nema_vg_set_fill_rule(NEMA_VG_FILL_EVEN_ODD);
    nema_vg_draw_path(path, paint);
```

Finally, the outline of a shape can be drawn using two different ways; the first one would be to simply draw the outline using 1 pixel width for the border:

```
  x_pos = STAR_DIST_X*6;
  nema_mat3x3_load_identity(m_path);
  nema_mat3x3_translate(m_path, x_pos, y_pos);
  nema_vg_path_set_matrix(path, m_path);
  nema_vg_set_fill_rule(NEMA_VG_STROKE);
  nema_vg_paint_set_paint_color(paint, nema_rgba(0xff, 0x00, 0x000, 0xff)); // red
  nema_vg_draw_path(path, paint);
```

The second one would be to draw the outline using stroke:

```
x_pos = STAR_DIST_X*7;
nema_mat3x3_load_identity(m_path);
nema_mat3x3_translate(m_path, x_pos, y_pos);
nema_vg_path_set_matrix(path, m_path);
nema_vg_paint_set_stroke_width(paint, 5.f);
nema_vg_draw_path(path, paint);
```

The drawing operations that have been presented in this section are added to the bound command list (for more information about the command lists, please refer to NEMA®| GFX Manual). Therefore you must submit the bound command list to the GPU for execution. Otherwise, the framebuffer will not be updated.

## 3.4 Memory Deallocation

The last part of a NEMA®| GFX VG application is to free the occupied memory. The overall flow of a NEMA®| GFX VG application requires the creation of the necessary objects (path and paint) and their configuration so that they can perform drawing operations. Subsequently, these objects must be destroyed after they are not necessary anymore for the needs of the application. The following function calls destroy these objects.

```
nema_vg_paint_destroy(paint);
nema_vg_path_destroy(path);
nema_vg_grad_destroy(gradient);
nema_vg_deinit();
```

These will free the memory occupied by the `paint` object, the `path` object, the `gradient` object, and the stencil buffer and look up table (function `nema_vg_deinit()`) that were created (see Memory Allocations).

# 4 Vector Graphics API

## 4.1 Files

Here is a list of all files with brief descriptions:

### 4.1.1 nema_vg.h File

Core NemaVG API drawing and initialization functions.

```
#include "nema_core.h"
#include "nema_sys_defs.h"
#include "nema_vg_path.h"
#include "nema_vg_paint.h"
#include "nema_vg_context.h"
```

#### Functions

void nema_vg_init(int width, int height)

*Initializes NemaVG library and allocates the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) Call either this or nema_vg_init_stencil_pool to allocate the stencil buffer to a different memory pool or nema_vg_init_stencil_prealloc to provide the stencil buffer.*

void nema_vg_init_stencil_pool(int width, int height, int pool)

*Initializes NemaVG library and allocate the stencil buffer in a specific memory pool. Call either this or nema_vg_init to allocate the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) or nema_vg_init_stencil_prealloc to provide the stencil buffer.*

void nema_vg_init_stencil_prealloc(int width, int height, nema_buffer_t stencil_bo)

*Initializes NemaVG library without allocating the stencil buffer which is provided by the user. Call either this or nema_vg_init to allocate the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) or nema_vg_init_stencil_pool to allocate the stencil buffer to a different memory pool.*

void nema_vg_reinit()

*Reinitialize NemaVG library after a gpu powerofff.*

void nema_vg_deinit()

*Deinitialize NemaVG library. Free memory from implicitly allocated objects (stencil buffer if created inside the library, lut buffer and tsvgs' path, paint and gradient buffers)*

uint32_t nema_vg_draw_path(NEMA_VG_PATH_HANDLE path, NEMA_VG_PAINT_HANDLE paint)

*Draw a path using a specified paint object.*

uint32_t nema_vg_draw_line(float x1, float y1, float x2, float y2, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)

*Draw a line shape.*

uint32_t nema_vg_draw_rect(float x, float y, float width, float height, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)

*Draw a rectangle shape.*

uint32_t nema_vg_draw_rounded_rect(float x, float y, float width, float height, float rx, float ry, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)

*Draw a rounded rectangle shape.*

uint32_t nema_vg_draw_ellipse(float cx, float cy, float rx, float ry, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)

*Draw a ellipse shape.*

uint32_t nema_vg_draw_circle(float cx, float cy, float r, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint)

*Draw a circle shape.*

uint32_t nema_vg_draw_ring(float cx, float cy, float ring_radius, float angle_start, float angle_end, NEMA_VG_PAINT_HANDLE paint)

*Draw a filled ring with rounded caps shape. In case of a conical gradient paint type, the conical gradient center should be at the center of the ring(cx, cy). In other case, where the two centers do not match, the ring should be drawn with NEMA_VG_QUALITY_MAXIMUM. The ring width can be set with the paint's stroke_width.*

void nema_vg_get_coord_limits(float *min_coord, float *max_coord)

*Returns the minimum and maximum values for the coordinates that can be handled by the underlying hardware.*

### Detailed Description

Core NemaVG API drawing and initialization functions.

### Function Documentation

**uint32_t nema_vg_draw_circle ( float cx, float cy, float r, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint )**

Draw a circle shape.

**Parameters**

| Parameter | description |
|---|---|
| cx | The x center of the circle |
| cy | The y center of the circle |
| r | Radius of the circle |
| m | 3x3 affine transformation matrix |
| paint | The paint to draw |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**uint32_t nema_vg_draw_ellipse ( float cx, float cy, float rx, float ry, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint )**

Draw a ellipse shape.

**Parameters**

| Parameter | description |
|---|---|
| cx | The x position of the ellipse |
| cy | The y position of the ellipse |
| rx | Radius on the x axis |
| ry | Radius on the y axis |
| m | 3x3 affine transformation matrix |

| Parameter | description |
|---|---|
| paint | The paint to draw |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**uint32_t nema_vg_draw_line ( float x1, float y1, float x2, float y2, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint )**

Draw a line shape.

**Parameters**

| Parameter | description |
|---|---|
| x1 | Upper left x coordinate |
| y1 | Upper left y coordinate |
| x2 | The width |
| y2 | The height |
| m | 3x3 affine transformation matrix |
| paint | The paint to draw |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**uint32_t nema_vg_draw_path ( NEMA_VG_PATH_HANDLE path, NEMA_VG_PAINT_HANDLE paint )**

Draw a path using a specified paint object.

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer (handle) to the path that will be drawn |
| paint | Pointer (handle) to the paint object that wil be used for drawing |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**uint32_t nema_vg_draw_rect ( float x, float y, float width, float height, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint )**

Draw a rectangle shape.

**Parameters**

| Parameter | description |
|---|---|
| x | Upper left x coordinate |
| y | Upper left y coordinate |
| width | The width |
| height | The height |
| m | 3x3 affine transformation matrix |
| paint | The paint to draw |

**Return**

Error code

**uint32_t nema_vg_draw_ring ( float cx, float cy, float ring_radius, float angle_start, float angle_end, NEMA_VG_PAINT_HANDLE paint )**

Draw a filled ring with rounded caps shape. In case of a conical gradient paint type, the conical gradient center should be at the center of the ring(cx, cy). In other case, where the two centers do not match, the ring should be drawn with NEMA_VG_QUALITY_MAXIMUM. The ring width can be set with the paint's stroke_width.

**Parameters**

| Parameter | description |
|---|---|
| cx | The center x coordinate of the ring |
| cy | The center y coordinate of the ring |
| ring_radius | The radius of the ring |
| angle_start | The angle in degrees of the ring |
| angle_end | The angle in degrees that ends this ring |
| paint | The paint to draw |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**uint32_t nema_vg_draw_rounded_rect ( float x, float y, float width, float height, float rx, float ry, nema_matrix3x3_t m, NEMA_VG_PAINT_HANDLE paint )**

Draw a rounded rectangle shape.

**Parameters**

| Parameter | description |
|---|---|
| x | Upper left x coordinate |
| y | Upper left y coordinate |

| Parameter | description |
|---|---|
| width | The width |
| height | The height |
| rx | Horizontal cornel radius |
| ry | Vertical cornel radius |
| m | 3x3 affine transformation matrix |
| paint | The paint to draw |

**Return**

Error code. See NEMA_VG_ERR_* defines in "nema_vg_context.h" header file for the error codes.

**void nema_vg_get_coord_limits ( float * min_coord, float * max_coord )**

Returns the minimum and maximum values for the coordinates that can be handled by the underlying hardware.

**Parameters**

| Parameter | description |
|---|---|
| min_coord | Minimum coordinate (x or y) value (pointer) |
| max_coord | Maximum coordinate (x or y) value (pointer) |

**void nema_vg_init ( int width, int height )**

Initializes NemaVG library and allocates the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) Call either this or nema_vg_init_stencil_pool to allocate the stencil buffer to a different memory pool or nema_vg_init_stencil_prealloc to provide the stencil buffer.

**Parameters**

| Parameter | description |
|---|---|
| width | Framebuffer width |
| height | Framebuffer height |

**void nema_vg_init_stencil_pool ( int width, int height, int pool )**

Initializes NemaVG library and allocate the stencil buffer in a specific memory pool. Call either this or nema_vg_init to allocate the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) or nema_vg_init_stencil_prealloc to provide the stencil buffer.

**Parameters**

| Parameter | description |
|---|---|
| width | Framebuffer width |
| height | Framebuffer height |
| pool | Memory pool for allocating the stencil buffer (memory pools are platform specific and defined in nema_sys_defs.h file) |

**void nema_vg_init_stencil_prealloc ( int width, int height, nema_buffer_t stencil_bo )**

Initializes NemaVG library without allocating the stencil buffer which is provided by the user. Call either this or nema_vg_init to allocate the stencil buffer to the default memory pool (NEMA_MEM_POOL_FB) or nema_vg_init_stencil_pool to allocate the stencil buffer to a different memory pool.

**Parameters**

| Parameter | description |
|---|---|
| width | Framebuffer width |
| height | Framebuffer height |

| Parameter | description |
|---|---|
| stencil_bo | stencil buffer |

### 4.1.2 nema_vg_context.h File

NemaVG Context interface.

```
#include "nema_graphics.h"
#include "nema_matrix3x3.h"
```

**Macros**

#define NEMA_VG_HANDLE void*

#define NEMA_VG_PATH_HANDLE NEMA_VG_HANDLE

#define NEMA_VG_PAINT_HANDLE NEMA_VG_HANDLE

#define NEMA_VG_GRAD_HANDLE NEMA_VG_HANDLE

#define NEMA_VG_ERR_NO_ERROR (0x00000000U)

#define NEMA_VG_ERR_BAD_HANDLE (0x00000001U)

#define NEMA_VG_ERR_BAD_BUFFER (0x00000002U)

#define NEMA_VG_ERR_INVALID_FILL_RULE (0x00000004U)

#define NEMA_VG_ERR_INVALID_PAINT_TYPE (0x00000008U)

#define NEMA_VG_ERR_INVALID_VERTEX_DATA (0x00000010U)

#define NEMA_VG_ERR_NO_RADIAL_ENABLED (0x00000020U)

#define NEMA_VG_ERR_NO_BOUND_CL (0x00000040U)

#define NEMA_VG_ERR_INVALID_ARGUMENTS (0x00000080U)

#define NEMA_VG_ERR_INVALID_ARC_DATA (0x00000100U)

#define NEMA_VG_ERR_CL_FULL (0x00000200U)

#define NEMA_VG_ERR_DRAW_OUT_OF_BOUNDS (0x00000400U)

#define NEMA_VG_ERR_INVALID_MASKING_OBJ (0x00000800U)

#define NEMA_VG_ERR_INVALID_MASKING_FORMAT (0x00001000U)

#define NEMA_VG_ERR_INVALID_LUT_IDX_FORMAT (0x00002000U)

#define NEMA_VG_ERR_COORDS_OUT_OF_RANGE (0x00004000U)

#define NEMA_VG_ERR_EMPTY_TSVG (0x00008000U)

#define NEMA_VG_ERR_NO_BOUND_FONT (0x00010000U)

#define NEMA_VG_ERR_UNSUPPORTED_FONT (0x00020000U)

#define NEMA_VG_ERR_NON_INVERTIBLE_MATRIX (0x00040000U)

#define NEMA_VG_ERR_INVALID_GRAD_STOPS (0x00080000U)

#define NEMA_VG_ERR_NO_INIT (0x00100000U)

#define NEMA_VG_ERR_INVALID_STROKE_WIDTH (0x00200000U)

#define NEMA_VG_ERR_INVALID_OPACITY (0x00400000U)

#define NEMA_VG_FILL_DRAW (0x00U)

#define NEMA_VG_STROKE (0x00U)

#define NEMA_VG_FILL_EVEN_ODD (0x01U)

#define NEMA_VG_FILL_NON_ZERO (0x02U)

#define NEMA_VG_QUALITY_BETTER (0x00U)

#define NEMA_VG_QUALITY_FASTER (0x01U)

#define NEMA_VG_QUALITY_MAXIMUM (0x02U)

#define NEMA_VG_QUALITY_NON_AA (0x10U)

### Typedefs

typedef float  nema_vg_float_t typedef float nema_vg_float_t
More...

**Functions**

uint32_t nema_vg_set_global_matrix(nema_matrix3x3_t m)

*Set the global transformation matrix. Global matrix will be applied in all NemaVG rendering operations that will follow.*

void nema_vg_reset_global_matrix(void)

*Disable the global transformation matrix.*

void nema_vg_set_fill_rule(uint8_t fill_rule)

*Set the fill rule that will be applied when rendering a path.*

void nema_vg_masking(uint8_t masking)

*Enable/Disable Masking.*

uint32_t nema_vg_set_mask(nema_img_obj_t *mask_obj)

*Set the mask object (texture)*

void nema_vg_set_mask_translation(float x, float y)

*Translate the mask object (texture) with respect to origin point (0, 0). Sets the position of the mask object.*

void nema_vg_set_quality(uint8_t quality)

*Set the rendering quality.*

void nema_vg_set_blend(uint32_t blend)

*Set the blending mode for VG operations (see nema_blender.h documentation in NemaGFX API Manual) Additional Blending Operations: only NEMA_BLOP_SRC_PREMULT is supported.*

uint32_t nema_vg_get_error(void)

*Get the current error code. Clears the error afterwards.*

void nema_vg_handle_large_coords(uint8_t enable, uint8_t allow_internal_alloc)

*Enable/disable large coordinates handling when rendering a TSVG, a path or a predefined shape.*

uint32_t nema_vg_bind_clip_coords_buf(void *segs, uint32_t segs_size_bytes, void *data, uint32_t data_size_bytes)

*Bind segment and data buffers to be used for handling large coordinates.*

void nema_vg_unbind_clip_coords_buf(void)

*Unbind segment and data buffers to be used for handling large coordinates.*

### Detailed Description

NemaVG Context interface.

Contains NemaVG error codes, fill rules, rendering quality defines and functions for updating various rendering parameters. The functions defined here can be used to access the context parameters. The Context is an internal (opaque) struct of NemaVG.

### Macro Definition Documentation

#### #define NEMA_VG_ERR_BAD_BUFFER

Bad buffer

#### #define NEMA_VG_ERR_BAD_HANDLE

Bad handle

#### #define NEMA_VG_ERR_CL_FULL

reserved

#### #define NEMA_VG_ERR_COORDS_OUT_OF_RANGE

Path coordinates out of supported range

#### #define NEMA_VG_ERR_DRAW_OUT_OF_BOUNDS

Path is out of the drawing area

#### #define NEMA_VG_ERR_EMPTY_TSVG

Tsvg has no geometries

#### #define NEMA_VG_ERR_INVALID_ARC_DATA

reserved

#### #define NEMA_VG_ERR_INVALID_ARGUMENTS

Invalid arguments

**#define NEMA_VG_ERR_INVALID_FILL_RULE**

Invalid fill rule

**#define NEMA_VG_ERR_INVALID_GRAD_STOPS**

Gradient stops exceed maximum available stops

**#define NEMA_VG_ERR_INVALID_LUT_IDX_FORMAT**

Invalid LUT indices object Format

**#define NEMA_VG_ERR_INVALID_MASKING_FORMAT**

Invalid Masking object Format

**#define NEMA_VG_ERR_INVALID_MASKING_OBJ**

Masking object was not set

**#define NEMA_VG_ERR_INVALID_OPACITY**

Invalid opacity

**#define NEMA_VG_ERR_INVALID_PAINT_TYPE**

Invalid paint type

**#define NEMA_VG_ERR_INVALID_STROKE_WIDTH**

Invalid stroke width

**#define NEMA_VG_ERR_INVALID_VERTEX_DATA**

Invalid vertex data

**#define NEMA_VG_ERR_NON_INVERTIBLE_MATRIX**

A matrix that needs to be inverted, is not invertible

**#define NEMA_VG_ERR_NO_BOUND_CL**

No bound CL

**#define NEMA_VG_ERR_NO_BOUND_FONT**

There is no bound font

**#define NEMA_VG_ERR_NO_ERROR**

No Error

**#define NEMA_VG_ERR_NO_INIT**

VG uninitialized

**#define NEMA_VG_ERR_NO_RADIAL_ENABLED**

Radial not present in HW

**#define NEMA_VG_ERR_UNSUPPORTED_FONT**

The font is not supported (eg. older version) by NemaVG API

**#define NEMA_VG_FILL_DRAW**

DEPRECATED Stroke fill rule

**#define NEMA_VG_FILL_EVEN_ODD**

Evenodd fill rule

**#define NEMA_VG_FILL_NON_ZERO**

Non zero fill rule

**#define NEMA_VG_GRAD_HANDLE**

NemaVG gradient handle (pointer to gradient object)

**#define NEMA_VG_HANDLE**

NemaVG handle object (void pointer)

**#define NEMA_VG_PAINT_HANDLE**

NemaVG paint handle (pointer to paint object)

**#define NEMA_VG_PATH_HANDLE**

NemaVG path handle (pointer to path object)

**#define NEMA_VG_QUALITY_BETTER**

Better rendering quality (default option, balances rendering quality and performance)

**#define NEMA_VG_QUALITY_FASTER**

Faster rendering quality (favors performance over rendering quality)

**#define NEMA_VG_QUALITY_MAXIMUM**

Maximum rendering quality (favors rendering quality over performance)

**#define NEMA_VG_QUALITY_NON_AA**

Rendering quality without AA

**#define NEMA_VG_STROKE**

Stroke fill rule

**Typedef Documentation**

**typedef float nema_vg_float_t**

Floating point data type (default is 'float')

**Function Documentation**

**uint32_t nema_vg_bind_clip_coords_buf ( void * segs, uint32_t segs_size_bytes, void * data, uint32_t data_size_bytes )**

Bind segment and data buffers to be used for handling large coordinates.

**Parameters**

| Parameter | description |
|---|---|
| segs | Pointer to segment buffer for large coordinates |

| Parameter | description |
|---|---|
| segs_size_bytes | Segment buffer size in bytes |
| data | Pointer to data buffer for large coordinates |
| data_size_bytes | Data buffer size in bytes |

**uint32_t nema_vg_get_error ( void )**

Get the current error code. Clears the error afterwards.

**Return**

Error code. See NEMA_VG_ERR_* defines for all the possible error codes.

**void nema_vg_handle_large_coords ( uint8_t enable, uint8_t allow_internal_alloc )**

Enable/disable large coordinates handling when rendering a TSVG, a path or a predefined shape.

**Parameters**

| Parameter | description |
|---|---|
| enable | 0 to disable, 1 to enable |
| allow_internal_alloc | 0 to not allow internal allocation, 1 to allow |

**void nema_vg_masking ( uint8_t masking )**

Enable/Disable Masking.

**Parameters**

| Parameter | description |
|---|---|
| masking | 1 to enable, 0 to disable |

**void nema_vg_set_blend ( uint32_t blend )**

Set the blending mode for VG operations (see nema_blender.h documentation in NemaGFX API Manual) Additional Blending Operations: only NEMA_BLOP_SRC_PREMULT is supported.

**Parameters**

| Parameter | description |
|---|---|
| blend | Blending mode |

**See also**

nema_blending_mode()

**void nema_vg_set_fill_rule ( uint8_t fill_rule )**

Set the fill rule that will be applied when rendering a path.

**Parameters**

| Parameter | description |
|---|---|
| fill_rule | fill rule (NEMA_VG_STROKE, NEMA_VG_FILL_EVEN_ODD, NEMA_VG_FILL_NON_ZERO) |

**uint32_t nema_vg_set_global_matrix ( nema_matrix3x3_t m )**

Set the global transformation matrix. Global matrix will be applied in all NemaVG rendering operations that will follow.

**Parameters**

| Parameter | description |
|---|---|
| m | transformation matrix |

**Return**

Error code

**uint32_t nema_vg_set_mask ( nema_img_obj_t * mask_obj )**

Set the mask object (texture)

**Parameters**

| Parameter | description |
|---|---|
| mask_obj | Texture to be used as mask. Its format must be NEMA_A1, NEMA_A2, NEMA_A4 or Nema_A8, otherwise it will return an error. |

**Return**

Error code. If no error occurs, NEMA_VG_ERR_NO_ERROR otherwise NEMA_VG_ERR_INVALID_MASKING_FORMAT.

**void nema_vg_set_mask_translation ( float x, float y )**

Translate the mask object (texture) with respect to origin point (0, 0). Sets the position of the mask object.

**Parameters**

| Parameter | description |
|---|---|
| x | Horizontal position to place the mask object |
| y | Horizontal position to place the mask object |

**void nema_vg_set_quality ( uint8_t quality )**

Set the rendering quality.

**Parameters**

| Parameter | description |
|---|---|
| quality | level (NEMA_VG_QUALITY_BETTER, NEMA_VG_QUALITY_FASTER, NEMA_VG_QUALITY_MAXIMUM, NEMA_VG_QUALITY_NON_AA) |

### 4.1.3 nema_vg_font.h File

Vector font rendering.

```
#include "nema_matrix3x3.h"
#include "nema_vg.h"
#include "nema_vg_context.h"
```

**Data Structures**

struct nema_vg_kern_pair_t

More...

struct nema_vg_glyph_t

More...

struct nema_vg_font_range_t

More...

struct nema_vg_font_t

More...

**Macros**

#define NEMA_VG_ALIGNX_LEFT (0x00U)

#define NEMA_VG_ALIGNX_RIGHT (0x01U)

#define NEMA_VG_ALIGNX_CENTER (0x02U)

#define NEMA_VG_ALIGNX_JUSTIFY (0x03U)

#define NEMA_VG_ALIGNX_MASK (0x03U)

#define NEMA_VG_ALIGNY_TOP (0x00U)

#define NEMA_VG_ALIGNY_BOTTOM (0x04U)

#define NEMA_VG_ALIGNY_CENTER (0x08U)

#define NEMA_VG_ALIGNY_JUSTIFY (0x0cU)

#define NEMA_VG_ALIGNY_MASK (0x0cU)

#define NEMA_VG_TEXT_WRAP (0x10U)

#define NEMA_VG_CHAR_LTR (0x00U)

#define NEMA_VG_CHAR_RTL (0x01U)

#define NEMA_VG_CHAR_TTB (0x00U)

#define NEMA_VG_CHAR_BTT (0x02U)

**Functions**

void nema_vg_bind_font(nema_vg_font_t *font)

*Bind the font to use in future* nema_vg_print() *calls. Sets error code if font is not supported.*

void nema_vg_set_font_size(float size)

*Sets the size of the bound font. Future* nema_vg_print() *and* nema_vg_print_char() *calls will print using the last set size.*

void nema_vg_print(NEMA_VG_PAINT_HANDLE paint, const char *str, float x, float y, float w, float h, uint32_t align, nema_matrix3x3_t m)

*Print pre-formatted text.*

int nema_vg_string_get_bbox(const char *str, float *w, float *h, float max_w, uint32_t wrap)

*Get the bounding box's width and height of a vector string. Prior to calling this function, "nema_vg_set_font_size" must be called first.*

int nema_vg_get_ascender_pt()

*Get the text ascender value in point units. Font size must be set pror to calling this function.*

float nema_vg_print_char(NEMA_VG_PAINT_HANDLE paint, char ch, float x, float y, nema_matrix3x3_t m, uint32_t orientation)

*Print a single character.*

## Detailed Description

Vector font rendering.

This file includes the necessary structs and functions that are used for rendering text (strings and single characters), using vector fonts. The accompanying vector font converter utility, converts truetype fonts (ttf files) to instances of the structs defined here. A use case of this module is included in the respective examples (examples/NemaVG/render_vg_font).

## Macro Definition Documentation

### #define NEMA_VG_ALIGNX_CENTER

Align horizontally centered

### #define NEMA_VG_ALIGNX_JUSTIFY

Justify horizontally

### #define NEMA_VG_ALIGNX_LEFT

Align horizontally to the left

### #define NEMA_VG_ALIGNX_MASK

Horizontal alignment mask

### #define NEMA_VG_ALIGNX_RIGHT

Align horizontally to the right

### #define NEMA_VG_ALIGNY_BOTTOM

Align vertically to the bottom

### #define NEMA_VG_ALIGNY_CENTER

Align vertically centered

### #define NEMA_VG_ALIGNY_JUSTIFY

Justify vertically

### #define NEMA_VG_ALIGNY_MASK

Vertical alignment mask

### #define NEMA_VG_ALIGNY_TOP

Align vertically to the top

### #define NEMA_VG_CHAR_BTT

Character follows bottom to top orientation

### #define NEMA_VG_CHAR_LTR

Character follows left to right orientation

### #define NEMA_VG_CHAR_RTL

Character follows right to left orientation

### #define NEMA_VG_CHAR_TTB

Character follows top to bottom orientation

### #define NEMA_VG_TEXT_WRAP

Use text wrapping

## Function Documentation

**void nema_vg_bind_font ( nema_vg_font_t * font )**

Bind the font to use in future nema_vg_print() calls. Sets error code if font is not supported.

**Parameters**

| Parameter | description |
|---|---|
| font | Pointer to the vector font |

**int nema_vg_get_ascender_pt**

Get the text ascender value in point units. Font size must be set pror to calling this function.

**Return**

Ascender pt

**void nema_vg_print ( NEMA_VG_PAINT_HANDLE paint, const char * str, float x, float y, float w, float h, uint32_t align, nema_matrix3x3_t m )**

Print pre-formatted text.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer to the current paint object (contains the text color) |
| str | Pointer to string |
| x | X coordinate of text-area's top-left corner |
| y | Y coordinate of text-area's top-left corner |
| w | Max allowed width |
| h | Max allowed height |
| align | Alignment and wrapping mode |
| m | Transformation matrix |

**float nema_vg_print_char ( NEMA_VG_PAINT_HANDLE paint, char ch, float x, float y, nema_matrix3x3_t m, uint32_t orientation )**

Print a single character.

The position of the character is determined by the 'orientation' argument. x and y arguments define a point on the baseline. If the orientation is left to right (LTR), the character

will be placed to the right of the (x, y) point. Right to left (RTL) will place the character to the left of the (x, y) point. Top to bottom (TTB) will have the same effect as RTL and bottom to top (BTT) will place the character higher than the (x, y) point by an offset equal to the font height.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer to the current paint object (contains the text color) |
| ch | Character to be printed |
| x | X coordinate of character's top-left or top-right corner (controlled by the 'orientation' parameter) |
| y | Y coordinate of character's top-left or bottom-left corner (controlled by the 'orientation' parameter) |
| m | Transformation matrix |
| orientation | Character orientation (see NEMA_VG_CHAR_* defines) |

**Return**

Character width in pixels

**void nema_vg_set_font_size ( float size )**

Sets the size of the bound font. Future nema_vg_print() and nema_vg_print_char() calls will print using the last set size.

**Parameters**

| Parameter | description |
|---|---|
| font | Pointer to the vector font |

**int nema_vg_string_get_bbox ( const char * str, float * w, float * h, float max_w, uint32_t wrap )**

Get the bounding box's width and height of a vector string. Prior to calling this function, "nema_vg_set_font_size" must be called first.

**Parameters**

| Parameter | description |
|---|---|
| str | Pointer to string |
| w | Pointer to variable where width should be written |
| h | Pointer to variable where height should be written |
| max_w | Max allowed width |
| size | font size |
| wrap | enable text wraping |

**Return**

Number of carriage returns

### 4.1.4 nema_vg_paint.h File

Paint operation related fuctions. Paint is an internal (opaque) struct of NemaVG. The functions defined here can be used access its parameters.

```
#include "nema_interpolators.h"
#include "nema_matrix3x3.h"
#include "nema_vg_context.h"
#include "nema_graphics.h"
```

**Macros**

#define NEMA_VG_PAINT_COLOR (0x00U)

#define NEMA_VG_PAINT_FILL (0x00U)

#define NEMA_VG_PAINT_GRAD_LINEAR (0x01U)

#define NEMA_VG_PAINT_TEXTURE (0x02U)

#define NEMA_VG_PAINT_GRAD_RADIAL (0x03U)

#define NEMA_VG_PAINT_GRAD_CONICAL (0x04U)

#define NEMA_VG_PAINT_MAX_GRAD_STOPS (32)

**Functions**

NEMA_VG_PAINT_HANDLE nema_vg_paint_create()

*Create a paint object.*

void nema_vg_paint_destroy(NEMA_VG_PAINT_HANDLE paint)

*Destroy a paint object.*

void nema_vg_paint_clear(NEMA_VG_PAINT_HANDLE paint)

*Clear the parameters of a paint object.*

void nema_vg_paint_set_type(NEMA_VG_PAINT_HANDLE paint, uint8_t type)

*Set the paint type.*

void nema_vg_paint_lock_tran_to_path(NEMA_VG_PAINT_HANDLE paint, int locked)

*Lock paint transformation to path. If locked, path and paint transformation will be in sync.*

void nema_vg_paint_set_grad_linear(NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float x1, float y1, nema_tex_mode_t sampling_mode)

*Set linear gradient to a paint object.*

void nema_vg_paint_set_paint_color(NEMA_VG_PAINT_HANDLE paint, uint32_t rgba)

*Set the paint color.*

void nema_vg_paint_set_opacity(NEMA_VG_PAINT_HANDLE paint, float opacity)

*Set the paint opacity.*

void nema_vg_paint_set_stroke_width(NEMA_VG_PAINT_HANDLE paint, float stroke_width)

*Set stroke width.*

void nema_vg_paint_set_tex_matrix(NEMA_VG_PAINT_HANDLE paint, nema_matrix3x3_t m)

*Set transformation matrix for texture.*

void nema_vg_paint_set_tex(NEMA_VG_PAINT_HANDLE paint, nema_img_obj_t *tex)

*Set texture to paint object.*

void nema_vg_paint_set_lut_tex(NEMA_VG_PAINT_HANDLE paint, nema_img_obj_t *lut_palette, nema_img_obj_t *lut_indices)

*Set Lut-based (look-up-table) texture to paint object. See Nema Pixpresso User Manual regarding Lut formats.*

void nema_vg_paint_set_grad_conical(NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float cx, float cy, nema_tex_mode_t sampling_mode)

*Set Conical gradient to paint object.*

void nema_vg_paint_set_grad_radial(NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float r, nema_tex_mode_t sampling_mode)

*Set radial gradient to paint object.*

void nema_vg_paint_set_grad_radial2(NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float rx, float ry, nema_tex_mode_t sampling_mode)

*Set radial gradient to paint object, with different horizontal and vertical radius.*

NEMA_VG_GRAD_HANDLE nema_vg_grad_create(void)

*Create gradient object.*

void nema_vg_grad_destroy(NEMA_VG_GRAD_HANDLE grad)

*Destroy gradient object.*

void nema_vg_grad_set(NEMA_VG_GRAD_HANDLE grad, int stops_count, float *stops, color_var_t *colors)

*Set gradient parameters to a gradient object.*

### Detailed Description

Paint operation related fuctions. Paint is an internal (opaque) struct of NemaVG. The functions defined here can be used access its parameters.

### Macro Definition Documentation

#### #define NEMA_VG_PAINT_COLOR

Fill with color

#### #define NEMA_VG_PAINT_FILL

Deprecated - Fill with color (same as NEMA_VG_PAINT_COLOR)

#### #define NEMA_VG_PAINT_GRAD_CONICAL

Fill with conical gradient

**#define NEMA_VG_PAINT_GRAD_LINEAR**

Fill with linear gradient

**#define NEMA_VG_PAINT_GRAD_RADIAL**

Fill with radial gradient

**#define NEMA_VG_PAINT_MAX_GRAD_STOPS**

Maximum gradient stops

**#define NEMA_VG_PAINT_TEXTURE**

Fill with texture

**Function Documentation**

NEMA_VG_GRAD_HANDLE **nema_vg_grad_create ( void )**

Create gradient object.

**Return**

Handle (pointer) to the created gradient object

**void nema_vg_grad_destroy (** NEMA_VG_GRAD_HANDLE **grad )**

Destroy gradient object.

**Parameters**

| Parameter | description |
|---|---|
| grad | Pointer to the gradient object |

**void nema_vg_grad_set (** NEMA_VG_GRAD_HANDLE **grad, int stops_count, float * stops, color_var_t * colors )**

Set gradient parameters to a gradient object.

**Parameters**

| Parameter | description |
|---|---|
| grad | Pointer (handle) to gradient object |
| stops_count | Number of stop colors |
| stops | Pointer to stop colors coordinates |
| colors | Pointer to stop color values |

**void nema_vg_paint_clear ( NEMA_VG_PAINT_HANDLE paint )**

Clear the parameters of a paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint object |

NEMA_VG_PAINT_HANDLE **nema_vg_paint_create**

Create a paint object.

**Return**

Handle to the created paint object

**void nema_vg_paint_destroy ( NEMA_VG_PAINT_HANDLE paint )**

Destroy a paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Handle to paint object that should be destroyed |

**void nema_vg_paint_lock_tran_to_path ( NEMA_VG_PAINT_HANDLE paint, int locked )**

Lock paint transformation to path. If locked, path and paint transformation will be in sync.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer to paint object |
| locked | 1 if locked (default), 0 if not locked |

**void nema_vg_paint_set_grad_conical ( NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float cx, float cy, nema_tex_mode_t sampling_mode )**

Set Conical gradient to paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint |
| grad | Pointer (handle) to gradient |
| cx | Conical gradient center point x coordinate |
| cy | Conical gradient center point y coordinate |
| sampling_mode | Sampling mode |

**void nema_vg_paint_set_grad_linear ( NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float x1, float y1, nema_tex_mode_t sampling_mode )**

Set linear gradient to a paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer to paint object |
| grad | Pointer to gradient object |
| x0 | Linear gradient start point x coordinate |
| y0 | Linear gradient start point y coordinate |
| x1 | Linear gradient end point x coordinate |
| y1 | Linear gradient end point y coordinate |
| sampling_mode | Sampling mode. NEMA_TEX_BORDER defaults to NEMA_TEX_CLAMP |

**void nema_vg_paint_set_grad_radial ( NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float r, nema_tex_mode_t sampling_mode )**

Set radial gradient to paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint |
| grad | Pointer (handle) to gradient |
| x0 | Radial gradient center point x coordinate |
| y0 | Radial gradient center point y coordinate |
| r | Radial gradient radius |
| sampling_mode | Sampling mode |

**void nema_vg_paint_set_grad_radial2 ( NEMA_VG_PAINT_HANDLE paint, NEMA_VG_GRAD_HANDLE grad, float x0, float y0, float rx, float ry, nema_tex_mode_t sampling_mode )**

Set radial gradient to paint object, with different horizontal and vertical radius.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint |
| grad | Pointer (handle) to gradient |
| x0 | Radial gradient center point x coordinate |
| y0 | Radial gradient center point y coordinate |
| rx | Radial gradient radius on x axis |
| ry | Radial gradient radius on y axis |
| sampling_mode | Sampling mode |

**void nema_vg_paint_set_lut_tex ( NEMA_VG_PAINT_HANDLE paint, nema_img_obj_t * lut_palette, nema_img_obj_t * lut_indices )**

Set Lut-based (look-up-table) texture to paint object. See Nema Pixpresso User Manual regarding Lut formats.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint object |
| lut_palette | Pointer to the Palette of the Lut image object |
| lut_indices | Pointer to the indices of the Lut image object |

**void nema_vg_paint_set_opacity ( NEMA_VG_PAINT_HANDLE paint, float opacity )**

Set the paint opacity.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (pointer) to paint object |
| opacity | Opacity to be set, 1 is fully opaque and 0 is fully transparent |

**void nema_vg_paint_set_paint_color ( NEMA_VG_PAINT_HANDLE paint, uint32_t rgba )**

Set the paint color.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint object |
| rgba | Color to be set, in rgba (hex 0xAABBGGRR) format |

**void nema_vg_paint_set_stroke_width ( NEMA_VG_PAINT_HANDLE paint, float stroke_width )**

Set stroke width.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint object |
| stroke_width | Stroke width to be set |

**void nema_vg_paint_set_tex ( NEMA_VG_PAINT_HANDLE paint, nema_img_obj_t * tex )**

Set texture to paint object.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint |
| text | Pointer to texture image object |

**void nema_vg_paint_set_tex_matrix ( NEMA_VG_PAINT_HANDLE paint, nema_matrix3x3_t m )**

Set transformation matrix for texture.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint object |
| m | 3x3 transformation matrix |

**void nema_vg_paint_set_type ( NEMA_VG_PAINT_HANDLE paint, uint8_t type )**

Set the paint type.

**Parameters**

| Parameter | description |
|---|---|
| paint | Pointer (handle) to paint |

| Parameter | description |
|---|---|
| type | Paint type (NEMA_VG_PAINT_COLOR, NEMA_VG_PAINT_GRAD_LINEAR, NEMA_VG_PAINT_TEXTURE, NEMA_VG_PAINT_GRAD_RADIAL, NEMA_VG_PAINT_GRAD_CONICAL) |

### 4.1.5 nema_vg_path.h File

Path operation related fuctions.

```
#include "nema_interpolators.h"
#include "nema_matrix3x3.h"
#include "nema_sys_defs.h"
#include "nema_vg_context.h"
```

**Macros**

#define NEMA_VG_PRIM_CLOSE (0x00U)

#define NEMA_VG_PRIM_MOVE (0x01U)

#define NEMA_VG_PRIM_LINE (0x02U)

#define NEMA_VG_PRIM_HLINE (0x03U)

#define NEMA_VG_PRIM_VLINE (0x04U)

#define NEMA_VG_PRIM_BEZIER_QUAD (0x05U)

#define NEMA_VG_PRIM_BEZIER_CUBIC (0x06U)

#define NEMA_VG_PRIM_BEZIER_SQUAD (0x07U)

#define NEMA_VG_PRIM_BEZIER_SCUBIC (0x08U)

#define NEMA_VG_PRIM_ARC (0x09U)

#define NEMA_VG_PRIM_POLYGON (0x0AU)

#define NEMA_VG_PRIM_POLYLINE (0x0BU)

#define NEMA_VG_PRIM_MASK (0x0FU)

#define NEMA_VG_REL (0x10U)

#define NEMA_VG_ARC_LARGE (0x20U)

#define NEMA_VG_ARC_CW (0x40U)

#define NEMA_VG_PRIM_SCCWARC (NEMA_VG_PRIM_ARC )

#define NEMA_VG_PRIM_SCWARC (NEMA_VG_PRIM_ARC | NEMA_VG_ARC_CW )

#define NEMA_VG_PRIM_LCCWARC (NEMA_VG_PRIM_ARC | NEMA_VG_ARC_LARGE)

#define NEMA_VG_PRIM_LCWARC (NEMA_VG_PRIM_ARC | NEMA_VG_ARC_CW |NEMA_VG_ARC_LARGE)

#define NEMA_VG_PRIM_MOVE_REL (NEMA_VG_PRIM_MOVE | NEMA_VG_REL)

#define NEMA_VG_PRIM_LINE_REL (NEMA_VG_PRIM_LINE | NEMA_VG_REL)

#define NEMA_VG_PRIM_HLINE_REL (NEMA_VG_PRIM_HLINE | NEMA_VG_REL)

#define NEMA_VG_PRIM_VLINE_REL (NEMA_VG_PRIM_VLINE | NEMA_VG_REL)

#define NEMA_VG_PRIM_BEZIER_QUAD_REL (NEMA_VG_PRIM_BEZIER_QUAD | NEMA_VG_REL)

#define NEMA_VG_PRIM_BEZIER_CUBIC_REL (NEMA_VG_PRIM_BEZIER_CUBIC | NEMA_VG_REL)

#define NEMA_VG_PRIM_BEZIER_SQUAD_REL (NEMA_VG_PRIM_BEZIER_SQUAD | NEMA_VG_REL)

#define NEMA_VG_PRIM_BEZIER_SCUBIC_REL (NEMA_VG_PRIM_BEZIER_SCUBIC | NEMA_VG_REL)

#define NEMA_VG_PRIM_SCCWARC_REL (NEMA_VG_PRIM_SCCWARC | NEMA_VG_REL)

#define NEMA_VG_PRIM_SCWARC_REL (NEMA_VG_PRIM_SCWARC | NEMA_VG_REL)

#define NEMA_VG_PRIM_LCCWARC_REL (NEMA_VG_PRIM_LCCWARC | NEMA_VG_REL)

#define NEMA_VG_PRIM_LCWARC_REL (NEMA_VG_PRIM_LCWARC | NEMA_VG_REL)

#define NEMA_VG_PRIM_POLYGON_REL (NEMA_VG_PRIM_POLYGON | NEMA_VG_REL)

#define NEMA_VG_PRIM_POLYLINE_REL (NEMA_VG_PRIM_POLYLINE | NEMA_VG_REL)

### Functions

NEMA_VG_PATH_HANDLE nema_vg_path_create()

*Create path.*

void nema_vg_path_destroy(NEMA_VG_PATH_HANDLE path)

*Destroy path.*

void nema_vg_path_clear(NEMA_VG_PATH_HANDLE path)

*Clear path.*

void nema_vg_path_set_shape(NEMA_VG_PATH_HANDLE path, const size_t seg_size, const uint8_t *seg, const size_t data_size, const nema_vg_float_t *data)

*Set path shape (vertex buffer)*

void nema_vg_path_set_shape_and_bbox(NEMA_VG_PATH_HANDLE path, const size_t seg_size, const uint8_t *seg, const size_t data_size, const nema_vg_float_t *data, const nema_vg_float_t *bbox)

*Set path shape (vertex buffer) and bounding box. Same functionality as* nema_vg_path_set_shape() *but bbox is given by user (reduces CPU utilization)*

void nema_vg_path_set_matrix(NEMA_VG_PATH_HANDLE path, nema_matrix3x3_t m)

*Set affine transformation matrix.*

### Detailed Description

Path operation related fuctions.

### Macro Definition Documentation

#### #define NEMA_VG_ARC_CW

Clockwise arc segment

#### #define NEMA_VG_ARC_LARGE

Large arc segment

#### #define NEMA_VG_PRIM_ARC

Arc segment

**#define NEMA_VG_PRIM_BEZIER_CUBIC**

Cubic bezier segment

**#define NEMA_VG_PRIM_BEZIER_CUBIC_REL**

Relative cubic bezier segment

**#define NEMA_VG_PRIM_BEZIER_QUAD**

Quadratic bezier segment

**#define NEMA_VG_PRIM_BEZIER_QUAD_REL**

Relative quadratic bezier segment

**#define NEMA_VG_PRIM_BEZIER_SCUBIC**

Smooth cubic bezier segment

**#define NEMA_VG_PRIM_BEZIER_SCUBIC_REL**

Relative smooth cubic bezier segment

**#define NEMA_VG_PRIM_BEZIER_SQUAD**

Smooth quadratic bezier segment

**#define NEMA_VG_PRIM_BEZIER_SQUAD_REL**

Relative smooth quadratic bezier segment

**#define NEMA_VG_PRIM_CLOSE**

Close segment

**#define NEMA_VG_PRIM_HLINE**

Horizontal line segment

**#define NEMA_VG_PRIM_HLINE_REL**

Relative horizontal line segment

**#define NEMA_VG_PRIM_LCCWARC**

Large counterclockwise arc segment

**#define NEMA_VG_PRIM_LCCWARC_REL**

Relative lareg counterclockwise arc segment

**#define NEMA_VG_PRIM_LCWARC**

Large clockwise arc segment

**#define NEMA_VG_PRIM_LCWARC_REL**

Relative lareg rclockwise arc segment

**#define NEMA_VG_PRIM_LINE**

Line segment

**#define NEMA_VG_PRIM_LINE_REL**

Relative line segment

**#define NEMA_VG_PRIM_MASK**

Mask for all segments

**#define NEMA_VG_PRIM_MOVE**

Move segment

**#define NEMA_VG_PRIM_MOVE_REL**

Relative move segment

**#define NEMA_VG_PRIM_POLYGON**

Polygon segment

**#define NEMA_VG_PRIM_POLYGON_REL**

Relative polygon segment

**#define NEMA_VG_PRIM_POLYLINE**

Polyline segment

**#define NEMA_VG_PRIM_POLYLINE_REL**

Relative polyline segment

**#define NEMA_VG_PRIM_SCCWARC**

Small counterclockwise arc segment

**#define NEMA_VG_PRIM_SCCWARC_REL**

Relative small counterclockwise arc segment

**#define NEMA_VG_PRIM_SCWARC**

Small clockwise arc segment

**#define NEMA_VG_PRIM_SCWARC_REL**

Relative small clockwise arc segment

**#define NEMA_VG_PRIM_VLINE**

Vertical line segment

**#define NEMA_VG_PRIM_VLINE_REL**

Relative vertical line segment

**#define NEMA_VG_REL**

Rel segment

**Function Documentation**

**void nema_vg_path_clear (** NEMA_VG_PATH_HANDLE **path )**

Clear path.

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer to Path |

**Return**

void

NEMA_VG_PATH_HANDLE **nema_vg_path_create**

Create path.

**Return**

Created path

**void nema_vg_path_destroy (** NEMA_VG_PATH_HANDLE **path )**

Destroy path.

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer to Path |

**Return**

void

**void nema_vg_path_set_matrix (** NEMA_VG_PATH_HANDLE **path, nema_matrix3x3_t m )**

Set affine transformation matrix.

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer to path |

| Parameter | description |
|---|---|
| m | 3x3 affine transformation matrix |

**void nema_vg_path_set_shape ( NEMA_VG_PATH_HANDLE path, const size_t seg_size, const uint8_t * seg, const size_t data_size, const nema_vg_float_t * data )**

Set path shape (vertex buffer)

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer to path |
| seg_size | Number of segments to be added |
| seg | Pointer to segments |
| data_size | Number of data to be added |
| data | Pointer to coordinates |

**void nema_vg_path_set_shape_and_bbox ( NEMA_VG_PATH_HANDLE path, const size_t seg_size, const uint8_t * seg, const size_t data_size, const nema_vg_float_t * data, const nema_vg_float_t * bbox )**

Set path shape (vertex buffer) and bounding box. Same functionality as nema_vg_path_set_shape() but bbox is given by user (reduces CPU utilization)

**Parameters**

| Parameter | description |
|---|---|
| path | Pointer to path |
| seg_size | Number of segments to be added |
| seg | Pointer to segments |

| Parameter | description |
|---|---|
| data_size | Number of data to be added |
| data | Pointer to coordinates |
| bbox | Pointer to shape bound box coordinates {min_x, min_y, max_x, max_y} |

### 4.1.6 nema_vg_tsvg.h File

API for rendering .tsvg images.

```
#include "nema_vg_context.h"
```

**Functions**

void nema_vg_draw_tsvg(const void *buffer)

*Draws a TSVG buffer.*

void nema_vg_get_tsvg_resolution(const void *buffer, uint32_t *width, uint32_t *height)

*Get the width and height of tsvg.*

**Detailed Description**

API for rendering .tsvg images.

**Function Documentation**

**void nema_vg_draw_tsvg ( const void * buffer )**

Draws a TSVG buffer.

**Parameters**

| Parameter | description |
|---|---|
| buffer | Pointer to the TSVG buffer that will be drawn |

**void nema_vg_get_tsvg_resolution ( const void * buffer, uint32_t * width, uint32_t * height )**

Get the width and height of tsvg.

**Parameters**

| Parameter | description |
|---|---|
| buffer | Tsvg buffer |
| width | return Tsvg width |
| height | return Tsvg height |

### 4.1.7 nema_vg_version.h File

Contains version numbers for NemaVG API and the currently supported font version.

**Macros**

#define NEMA_VG_MAJOR_VERSION 0x01U

#define NEMA_VG_MINOR_VERSION 0x01U

#define NEMA_VG_REVISION_VERSION 0x05U

#define NEMA_VG_IMP_VERSION 0x00231000U

#define NEMA_VG_API_VERSION ((NEMA_VG_MAJOR_VERSION << 16) + (NEMA_VG_MINOR_VERSION << 8) + (NEMA_VG_REVISION_VERSION))

#define NEMA_VG_FONT_VERSION 0x01U

**Detailed Description**

Contains version numbers for NemaVG API and the currently supported font version.

**Macro Definition Documentation**

**#define NEMA_VG_API_VERSION**

NemaVG API version in format 0x00MMmmrr (M:major, m:minor, r:revision if any)

**#define NEMA_VG_FONT_VERSION**

Current font version

**#define NEMA_VG_IMP_VERSION**

NemaVG API version, implementation in format 0x00YYMM00 (Y: year, M: month)

**#define NEMA_VG_MAJOR_VERSION**

NemaVG API version, major number

**#define NEMA_VG_MINOR_VERSION**

NemaVG API version, minor number

**#define NEMA_VG_REVISION_VERSION**

NemaVG API version, revision number

## 4.2 Directories

Here is a list of all directories with brief descriptions:

### 4.2.1 File List

**NemaVG**

nema_vg.h

Core NemaVG API drawing and initialization functions.

nema_vg_context.h

NemaVG Context interface.

nema_vg_font.h

Vector font rendering.

nema_vg_paint.h

Paint operation related fuctions. Paint is an internal (opaque) struct of NemaVG. The functions defined here can be used access its parameters.

nema_vg_path.h

Path operation related fuctions.

nema_vg_tsvg.h

API for rendering .tsvg images.

nema_vg_version.h

Contains version numbers for NemaVG API and the currently supported font version.

## 4.3 Data Structures

Here is a list of all data structures with brief descriptions:

### 4.3.1 nema_vg_font_range_t Data Structure

```
#include <nema_vg_font.h>
```

**Data Fields**

const uint32_t first

*Unicode value of the first value of the range*

const uint32_t last

*Unicode value of the last value of the range*

const nema_vg_glyph_t * glyphs

*Pointer to the array of glyphs*

NemaVG vector font range data struct

**Detailed Description**

NemaVG vector font range data struct

### 4.3.2 nema_vg_font_t Data Structure

```
#include <nema_vg_font.h>
```

### Data Fields

const uint32_t version

*Font version*

const nema_vg_font_range_t * ranges

*Pointer to the array of ranges*

const nema_vg_float_t * data

*Pointer to the data of the vector font*

const size_t data_length

*Length of the vector font data*

const uint8_t * segment

*Pointer to the segments of the vector font*

const size_t segment_length

*Length of the vector font segments*

const float size

*Default font size (height)*

const float xAdvance

*Default advance width. If the space character is included in the ranges, then its advance width is set*

const float ascender

*Vertical distance from the baseline to the highest point of the font*

const float descender

*Vertical distance from the baseline to the lowest point of the font*

const nema_vg_kern_pair_t * kern_pairs

*Pointer to the array of the font's kerning pairs*

uint32_t flags

*Bit field, reserved for future use*

NemaVG vector font data struct

### Detailed Description

NemaVG vector font data struct

### 4.3.3 nema_vg_glyph_t Data Structure

```
#include <nema_vg_font.h>
```

**Data Fields**

const uint32_t data_offset

*Offset value for the data of the glyph in the respective data array*

const size_t data_length

*Length of the data in the respective data array*

const uint32_t segment_offset

*Offset value for the segments of the glyph in the respective segment array*

const size_t segment_length

*Length of the segments in the respective segment array*

const float xAdvance

*Advance width*

const uint32_t kern_offset

*Kerning offset of the glyph in the respective kerning array*

const uint8_t kern_length

*Length of the kerning information of the glyph*

const int16_t bbox_xmin

*Minimum x of the glyph's bounding box*

const int16_t bbox_ymin

*Minimum y of the glyph's bounding box*

const int16_t bbox_xmax

*Maximum x of the glyph's bounding box*

const int16_t bbox_ymax

*Maximum y of the glyph's bounding box*

NemaVG data struct of a glyph in vector format

**Detailed Description**

NemaVG data struct of a glyph in vector format

### 4.3.4 nema_vg_kern_pair_t Data Structure

```
#include <nema_vg_font.h>
```

#### Data Fields

const uint32_t left

*Neighbor character to the left of the current one (Unicode value)*

const float x_offset

*Kerning offset value (horizontally)*

NemaVG Kerning pair information data struct

#### Detailed Description

NemaVG Kerning pair information data struct

# 5 Utilities

Besides the core NEMA®| GFX VG API discussed in the previous sections, a set of utilities (under development) for rendering Vector Graphics is also provided. These are an SVG parser and a TTF parser.

## 5.1 NEMA|TSVG Converter

TSVG is a proprietary binary format used in NemaVG, to encode SVG information for faster rendering of SVG images.

SVG images need to be converted to this format, in order to be used. The conversion is performed with the use of NEMA®| pix-presso. For more information, refer to the *NEMA®| pix-presso Starting Guide* document.

**Figure 13: Ghostscript Tiger rendered from a tsvg file**

## 5.2 Vector Font Converter

Nema vector font converter is a utility application, that works in a similar manner as the Nema font converter. More specifically, it converts a TrueType font (.ttf files) to vector font structs compatible with NemaVG API. The generated files (.c and .h files) can then

be used along with other source files in NemaVG applications. When running this tool without any argument, the following information is printed:

```
NAME
        nema_vg_font_convert - convert TTF fonts to .bin, .c and .h vector fonts, com►
patible with NEMA|VG API

SYNOPSIS
        nema_vg_font_convert [OPTION]... [FILE]...

DESCRIPTION
        Convert TTF fonts to .c and .h, compatible with NEMA|gfx graphics API

        -r, --range
                add range of characters (start-end), e.g.: -r 0x20-0x7e, -r 32-127
        -h, --help
                display this help and exit
        -a, --ascii
                add ascii range. Equivalent to -r 0x20-0x7e
        -g, --greek
                add greek range. Equivalent to -r 0x370-0x3ff
        -k, --kerning
                add kerning
```

Using vector fonts is enabled by using the respective NemaVG module (which can be found in nema_vg_font.h header file). This module defines the structs and functions that are used for processing (at application run time) the generated files that are produced by the vector font converter (offline process). The following figure illustrates a frame where japanese characters are drawn in the display using this module.

**Figure 14: Japanese characters drawn using NemaVG API**