USER'S MANUAL

Nema GFX API Library

Reference Manual

A-SOCAPG-UMGA02EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Rev	Date	Description
1.0	October 2025	Initial Release

Reference Documents

Document ID	Description
A-SOCAPG-UMGA01EN	Nema DC API Library User's Manual
A-SOCAPG-UMGA03EN	Nema GFX Extensions Vector Graphics User's Manual
A-SOCAPG-ANGA01EN	Nema DC MiP Panels Configuration Application Note
A-SOCAPG-ANGA02EN	Nema GFX Extensions TSVG Supported Elements List Application Note
A-SOCAPG-SGGA01EN	Nema Pix-presso Starting Guide
A-SOCAPG-ANGA03EN	Nema GFX API Debugging Application Note
A-SOCAPG-UMGA04EN	Nema GUI-builder User's Manual
A-SOCAPG-UMGA05EN	Nema GFX Benchmark Suite User's Manual
A-SOCAPG-UMGA06EN	Nema Pico Graphics Processing Unit User's Manual
A-SOCAPG-UMGA07EN	Nema Pico Platform Drivers User's Manual

Table of Contents

List of Tables

List of Figures

SECTION

1

Preface

1.1 About this Manual

This manual presents the basic structural elements of the Nema GFX Library and demonstrates most of its features and capabilities. It also provides some examples of proper library usage along with general information on its overall operation scheme.

1.2 Audience

This manual is intended for engineers and developers that wish to learn some of the principles of computer graphics and start implementing drawing commands in an organized and efficient manner towards creating a Graphics User Interface. Although the document assumes some familiarity in the topics of software development and computer graphics, the authors provided certain references to relevant external sources when considered necessary.

1.3 Related Documents

The following documents are considered relevant for using the full spectrum of features of the Nema GFX Library:

- Nema Pico User Manual

SECTION

2

Introduction to Graphics

Computer graphics is the science of communicating visually via a display and its interaction devices. It is a cross-disciplinary field in which physics, mathematics, human perception, human-computer interaction and engineering blend, towards creating artificial images with the help of programming. It heavily involves computations, creation and manipulation of data and is based on a set of well-defined principles. There are several structural elements that computer graphics are built upon, the most significant of which can be found in the following list.

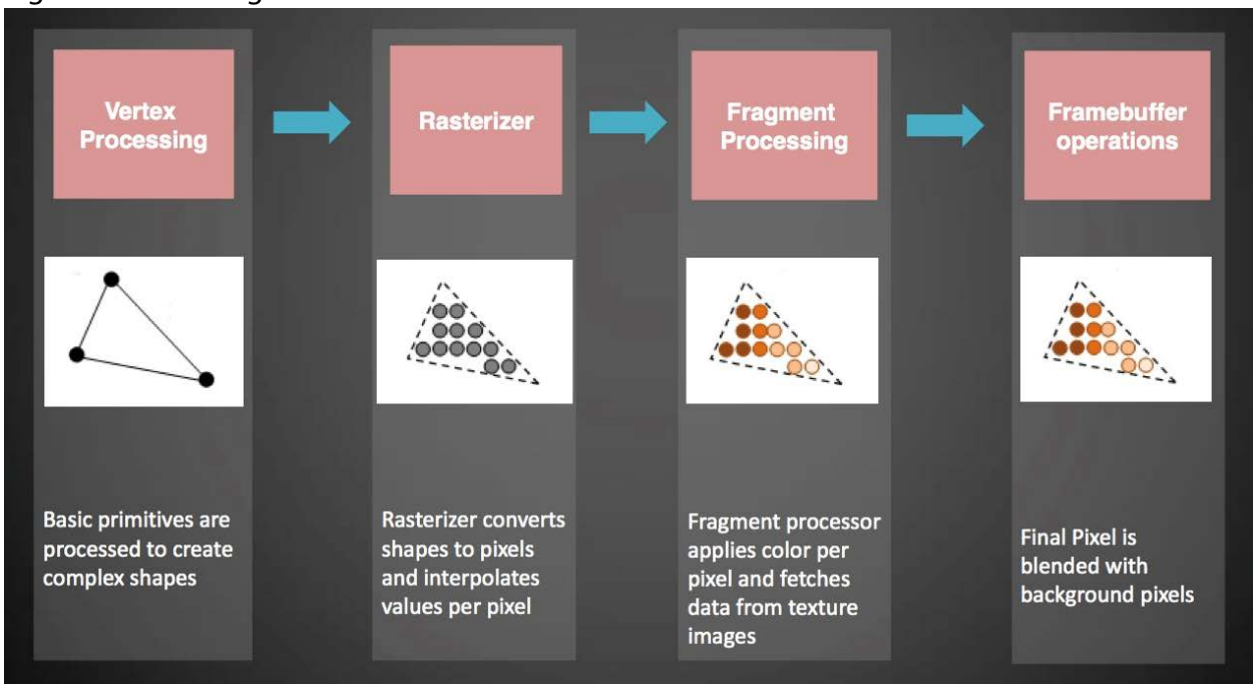
Table 2-1: Graphics Terms and Definitions

Terms	Definition
Pixel	**Pixel** in digital imaging is the smallest addressable element in an all points addressable display device. A pixel is generally considered as the smallest single component of a digital image and is often used as a measurement unit.
Raster	**Raster** image (or bitmapped image) is a matrix data structure representing the actual image content. Raster graphics are resolution-bound therefore unable to scale up without apparent loss of quality.
Vector	**Vector** graphics is a technique of using polygons, plane figures bound by a finite chain of straight-line segments closing a loop, to represent images. Vector graphics have inherent scale up abilities, only depending on the rendering device capability.
Rasterization	**Rasterization** is the process of converting an image described in a vector graphics format to a raster image consists of pixels for output on a video display or for storage in a bitmap format.
Texture	**Texture** is the digital representation of an object's surface. In addition to two-dimensional qualities such as color and transparency, a texture also incorporates three dimensional ones such as reflectiveness. Well-defined textures are very important for realistic three-dimensional image representation.

Table 2-1: Graphics Terms and Definitions *(Continued)*

Terms	Definition
Texture mapping	**Texture mapping** is the process of wrapping a pre-defined texture around any two or three dimensional object. Through this process, digital images and objects obtain a high level of detail.
Texel	**Texel** is the fundamental unit of texture space. Textures are represented by arrays of texels in the same way that pictures are represented by arrays of pixels.
Vertex	**Vertex** is a data structure that describes the location of an object by properly define its corners as positions of points in two or three-dimensional space.
Primitives	**Primitives** in computer graphics are the simplest geometric objects a system can handle. Common sets of two-dimensional primitives include lines, points, triangles and polygons while all other geometric elements are built up from these primitives. In three-dimensions, properly positioned triangles or polygons can be used as primitives to model more complex forms.
Blending	**Blending** is the process in which two or more images are combined per-pixel and weights to create new pictures.
Fragment	**Fragment** is the data necessary to generate a single-pixel primitive. This data is possible to include raster position, color or texture coordinates.
Interpolation	**Interpolation** in computer graphics is the process of generating intermediate values between two known reference points to give the appearance of continuity and smooth transition. Several distinct interpolation techniques are used in both computer graphics and animation, such as linear, bilinear, spline and polynomial interpolation.
Graphics Pipeline	**Graphics Pipeline** is an abstract sequence that incorporates the basic operations of generic rasterizer implementations, in particular: ▪ (Vertex) Per-vertex transformation to screen space ▪ (Rasterize) Per-triangle iteration over pixels with perspective-correct interpolation ▪ (Pixel) Per-pixel shading ▪ (Output Merge) Merging the output of shading with the current color and depth buffers As stated in the term "pipeline" is used due to the sequential steps that are used for the actual transformation from mathematical model to pixels; the results of the one stage are pushed on to the next stage so that the first stage can begin processing the next element immediately.

Figure 2-1: Rending Flow



SECTION

3

Nema GFX Architecture

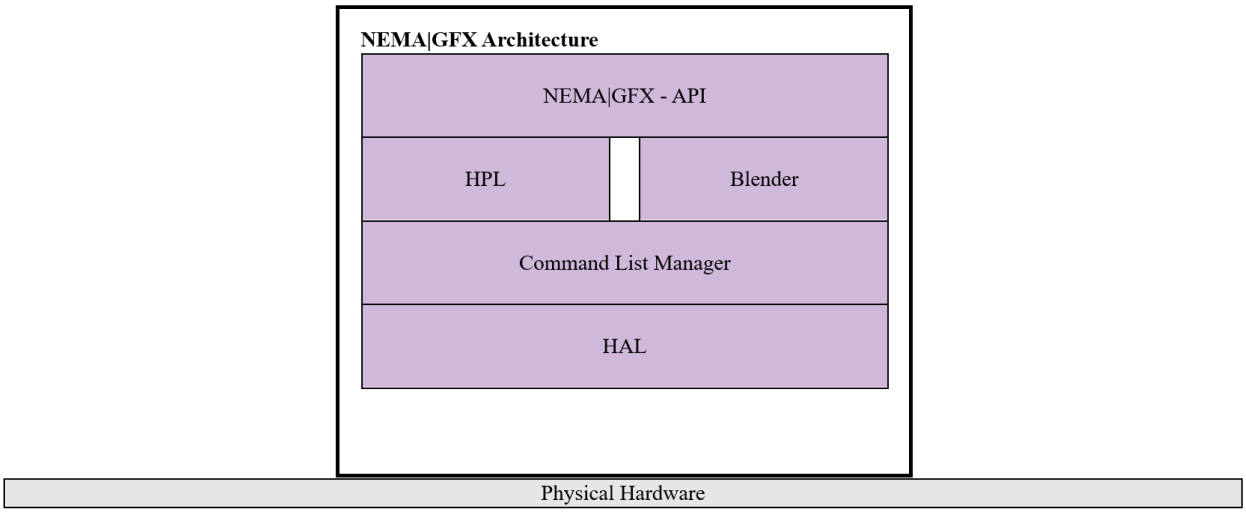
Nema GFX Library is a low level library that interfaces directly with the GPU and provides a software abstraction layer to organize and employ drawing commands with ease and efficiency. The target of Nema GFX is be used as a back-end to existing APIs (such as OpenGL® or any proprietary one) but also to expose higher level drawing functions, so as to be used as a standalone Graphics API. Its small footprint, efficient design and lack of any external dependencies, makes it ideal for use in embedded applications. Its small footprint, efficient design and lack of any external dependencies allow great performance with minimum CPU/MCU usage and power consumption, making it ideal for use in embedded applications.

Nema GFX includes a set of higher level calls, forming a complete standalone Graphics API for applications in systems where no other APIs are needed. This API is able to carry out draw operations from as simple as lines, triangles and quadrilaterals to more complex ones like blitting and perspective correct texture mapping.

Nema GFX is built on a modular architecture. An implementor may use only the lower layers of the architecture that provides communication to the GPU, synchronization and basic primitives drawing. The very thin Hardware Abstraction Layer allows for fast integration to the underlying hardware. The upper low level drawing API acts as a back-end interface for accelerating any higher third party Graphics API.

Nema GFX is built on a modular architecture. These modules are generally stacked one over another, forming a layered scheme. This gives the implementor the freedom to tailor the software stack according to ones needs.

Figure 3-1: Nema GFX Architecture



The lowest layer is a thin **Hardware Abstraction Layer (HAL)**. It includes some hooks for basic interfacing with the hardware such as register accessing, interrupt handling etc.

The layer above is the **Command List Manager**. It provides the appropriate API for creating, organizing and issuing Command Lists. This topic is discussed in detail in *Section 4 Quick Start Guide on page 15*.

Above the Command List Manager lies the **Hardware Programming Layer (HPL)**. This is a set of helper functions that assemble commands for programming the GPU. These commands actually write to the GPU's Configuration Register File, which is used to program the submodules of the GPU.

Alongside the HPL resides the Blender module. This module programs GPU's Programmable Processing Core. It creates binary executables for the Core. These executables correspond to the various blending modes that are supported by the Nema GFX Library.

On top of the Nema GFX stack lies the Nema GFX Graphics API. This API offers function calls to draw geometry primitives (lines, triangles, quadrilaterals etc.), blit images, render text, transform geometry objects, perform perspective correct texture mapping and more. When using Nema GFX as a back-end for a third party Graphics API, much of the Nema GFX Graphics API may be disabled.

SECTION

4

Quick Start Guide

The Quick Start Guide for the Nema GFX Library provides simple and comprehensive guidelines on how to use the Library for developing purposes. The information included in this Section can be found elsewhere in this document, but span across several paragraphs, which makes it daunting to start with a simple code example using the Nema GFX Library in a timely manner.

4.1 Command Lists

A Command List (CL) is one of the most important features of the Nema GFX library. CL usage decouples GPU and CPU, and through its re-usability decreases the computational effort of the CPU. This approach allows drawing of complicated scenes while keeping the CPU workload to the very minimum.

The design principles of CLs allow developers to extend the features of their application while optimizing its functionality at the same time. A CL is capable of jumping to another CL, forming a chain of interconnected commands. In addition, a CL is able to branch to another CL and once the branch execution is concluded, resume its functionality after the branching point.

The Nema GFX Library helps developers to easily take advantage of all these features through certain basic function calls that trigger the whole spectrum of CL capabilities. A short presentation of the most fundamental subset of them is listed in the following sections.

4.1.1 Create

There are three types of Command Lists that can be created:

- created in preallocated space
- non expandable of specific size
- expandable

The most straightforward function for initiating a simple coding example is the **Create** function which is listed below.

```
nema_cmdlist_t nema_cl_create(void)
```

This fundamental function allocates and initializes a new expandable Command List for later use.

For the creation of a Command List in preallocated space or of specific size, the following functions are available.

```
nema_cmdlist_t nema_cl_create_prealloc(nema_buffer_t *bo)
nema_cmdlist_t nema_cl_create_sized(int size_bytes)
```

4.1.2 Bind

Binding a Command List sets it as active. Each subsequent drawing call is incrementally added in the active Command List. At any time, all drawing operations should be called when there is a bound Command List.

There are three functions for binding a Command List.

```
nema_cl_bind(nema_cmdlist_t *cl)
nema_cl_bind_circular(name_cmdlist_t *cl)
nema_cl_bind_sectored_circular(nema_cmdlist_t *cl, int sectors)
```

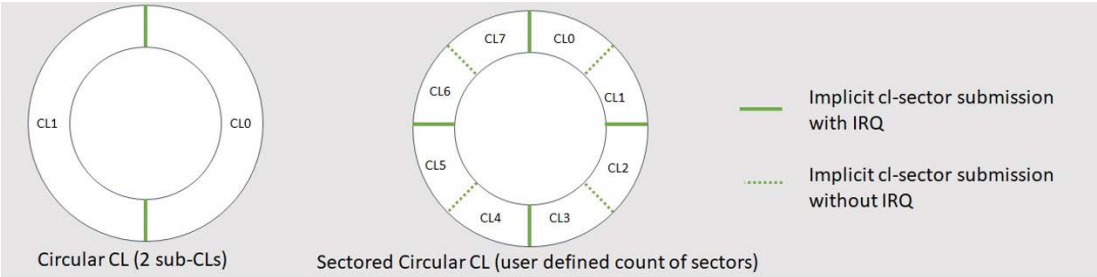
Command Lists that are bound as circular never get full. A check is executed with every new command added. If the limit is reached, the Command List gets implicitly submitted and the new command is added. Circular bound Command Lists cannot be reused, they can however show performance benefits particularly in cases with too many commands (eg. vector drawings with a lot of vector paths). Instead of having the CPU to fill in all the drawing commands and then initiate the GPU to start rendering, the GPU can start processing a first batch of the commands and at the same time the CPU can fill in the rest of them in the bound Command List.

Sectored circular command lists are derived from circular command lists and offer enhanced functionality. They divide a command list into multiple sectors, enabling more fine-grained CPU-GPU overlapping. Whenever the CPU fills a sector, the command list is implicitly submitted, allowing the CPU to continue filling the next sector without waiting for the GPU to complete processing the just submitted sector. This process acts as a synchronization point between the CPU and GPU when all

sectors are full. Sectored circular cl requires the sectors to be a minimum of 512 bytes and the number of sectors needs to have a count of at least 2, otherwise it will be automatically converted to 2. Lastly, it is important to note that like circular command lists, sectored circular command lists cannot be reused.

IMPORTANT: Binding circular command lists with **nema_cl_bind_circular(nema_cmdlist_t *cl)** creates a sectored circular command list with two sectors. Subsequently, there will no longer be a distinction between sectored circular with two sectors and circular command lists in the following pages.

Figure 4-1: Circular CL and Sectored Circular CL with 8 Sectors



4.1.3 Unbind

```
nema_cl_unbind(void)
```

Unbind the currently bound Command List.

4.1.4 Submit

```
nema_cl_submit(nema_cmdlist_t *cl)
```

Submit the referred Command List for execution. If this CL is currently the one that is bound, this call unbinds it. When a CL is submitted for execution, it should never be altered until it finishes execution. Writing in such a CL results in undefined behavior.

A typical routine for drawing would be the following:

```
nema_cmdlist_t cl = nema_cl_create();  // Create a new CL
nema_cl_bind(&cl);                     // Bind it
/* Drawing Operations */              // Draw scene
   nema_cl_unbind();                   // Unbind CL (optionally)
nema_cl_submit(&cl);                   // Submit CL for execution
```

If the current command list is bound as circular (**nema_cl_bind_circular()**), no other command list must be submitted for execution.

> **ATTENTION:** In order to avoid calling **nema_cl_wait()** (explained in *Section 4.1.5 Wait on page 18*), function **nema_cl_submit_no_irq(nema_cmdlist_t *cl)** can be used instead of **nema_cl_submit(nema_cmdlist_t *cl)**. This in not applicable to circular command lists (circular command lists must be submitted using **nema_cl_submit() function)**.

4.1.5 Wait

A command list that has been previously submitted, takes some time to be executed in the GPU. To ensure that the execution has finished, use the **nema_cl_wait()** function:

```
nema_cl_wait(nema_cmdlist_t *cl)
```

Taking this into consideration, the previously presented snippet is now:

```
   nema_cmdlist_t cl = nema_cl_create();  // Create a new CL
nema_cl_bind(&cl);                       // Bind it
    /* Drawing Operations */             // Draw scene
   nema_cl_unbind();                 // Unbind CL (optionally)
nema_cl_submit(&cl);                 // Submit CL for execution
    nema_cl_wait(&cl);               // Wait for the CL to finish
                                     // its execution
```

At this point it must be mentioned that for any command list that has been submitted for execution using the **nema_cl_submit()** it is necessary to call **nema_cl_wait()** at a later point. For command lists that have been submitted using **nema_cl_submit_no_irq()** (not applicable for circular command lists), one should never use **nema_cl_wait()**.

4.2 Binding Textures

Every drawing operation should have an effect on a given destination texture. The texture must reside in some memory space which is visible to the GPU.

```
void nema_bind_dst_tex(uint32_t baseaddr_phys,
uint32_t width, uint32_t height,
nema_tex_format_t format, int32_t stride)
```

The above function binds a texture to serve as destination. The texture's attributes (GPU address, width, height, format and stride) are written inside the bound CL. Each subsequent drawing operation will have an effect on this destination texture.

Most common graphics operations include some kind of image blitting (copying), like drawing a background image, GUI icons or even font rendering. The following function binds a texture to be used as foreground:

```
void nema_bind_src_tex(uint32_t baseaddr_phys,
uint32_t width, uint32_t height,
                       nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode);
```

This function call has a very similar functionality to the **NemaGFX_bind_dst_tex**. It has one extra argument, **NEMA_tex_mode_t** mode, that determines how to read a texture (point/bilinear sampling, wrapping mode etc).

The above example can now be extended as follows:

```
nema_cmdlist_t cl = nema_cl_create();    // Create a new CL
nema_cl_bind(&cl);                       // Bind it
                                  // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,      // Destination address
                  320, 240,       // width, height
                  NEMA_RGBA8888,  // Image format (32bit, rgba)
                  320*4);         // Stride in bytes (width*4 bytes
per pixel)
                                   // Bind Foreground Texture:
nema_bind_src_tex(SRC_IMAGE,       // Source address
320, 240,                          // width, height
               NEMA_RGBA8888,      // Image format (32bit, rgba)
               320*4,     // Stride in bytes (width*4 bytes per
                          // pixel)
                NEMA_FILTER_PS);   // Do point sampling (default
                                   // option)
/* Drawing Operations */           // Draw scene
nema_cl_unbind();                  // Unbind CL (optionally)
nema_cl_submit(&cl);               // Submit CL for execution
```

4.3 Clipping

When drawing a scene, it is often necessary to be able to define a rectangular area that the GPU is allowed to draw. This way, if some parts of a primitive (e.g. a triangle) falls outside the clipping area, that part is not going to be drawn at all, assuring correctness, better performance and improved power efficiency. The Clipping Rectangle can be defined as follows:

```
void nema_set_clip(int32_t x, int32_t y, int32_t w, int32_t h)
```

This function defines a Clipping Rectangle whose upper left vertex coordinates are **(x, y)** and its dimensions are **w*h**.

The default Clipping Rectangle usually is the entire canvas. In the above examples, we used textures with dimensions of 320x240. So, adding Clipping would result the following:

```
nema_cmdlist_t cl = nema_cl_create();  // Create a new CL
nema_cl_bind(&cl);                     // Bind it
                                       // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,           // Destination address
                  320, 240,            // width, height
                  NEMA_RGBA8888,       // Image format (32bit, rgba)
                  320*4);              // Stride in bytes (width*4
                                       // bytes per pixel)
                                       // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,          // Source address
                  320, 240,            // width, height
                  NEMA_RGBA8888,    // Image format (32bit, rgba)
                  320*4,            // Stride in bytes (width*4
                                    // bytes per pixel)
                  NEMA_FILTER_PS);  // Do point sampling (default
                                    // option)
nema_set_clip(0, 0, 320, 240);       // Define a 320x240 Clipping
                                     // Rectangle
/* Drawing Operations */         // Draw scene nema_cl_unbind();
                                 // Unbind CL (optionally)
nema_cl_submit(&cl);             // Submit CL for execution
```

4.4 Blending - Programming the Core

When building a graphical interface, the developer has to define what would be the result of drawing a pixel on the canvas. Since the canvas already contains the previous drawn scene, there must be a consistent way to determine how the source or foreground color (the one that is going to be drawn) will blend with the destination or background color that is already drawn. The source pixel can be fully opaque, thus will be drawn over the destination one, or it can be translucent and the result would be a blend of both the source and destination colors.

For example, blitting a background image of a GUI would require the Source Texture to cover entirely whatever is already drawn on the canvas. Afterwards, blitting an icon would require the background to be partially visible on the translucent areas of the icon. In order to make this possible, Nema GFX incorporates a powerful set of predefined blending modes that allow the developer to build functional and eye catching applications:

```
void nema_set_blend_fill(nema_blend_mode_t blending_mode) void
nema_set_blend_blit(nema_blend_mode_t blending_mode)
```

These two functions refer to blending when filling a primitive (e.g. triangle) with a color or when blitting a texture respectively.

The previous example, after setting the correct blending mode for blitting a background texture, would evolve to the following:

```
nema_cmdlist_t cl = nema_cl_create();    // Create a new CL
nema_cl_bind(&cl);                       // Bind it
```

```
                                        // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,            // Destination address
                  320, 240,             // width, height
                  NEMA_RGBA8888,        // Image format (32bit, rgba)
                  320*4);               // Stride in bytes (width*4
                                        // bytes per pixel)
                                        // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,           // Source address
                  320, 240,             // width, height
                  NEMA_RGBA8888,        // Image format (32bit, rgba)
                  320*4,                // Stride in bytes (width*4
                                        // bytes per pixel)
                  NEMA_FILTER_PS);     // Do point sampling (default
                                        // option)
nema_set_clip(0, 0, 320, 240);          // Define a 320x240 Clipping
                                        // Rectangle
nema_set_blend_blit(NEMA_BL_SRC);       // Program the Core to draw
                                        // the source color
                                        // without blending it with
                                        // the destination
                                        // texture
/* Drawing Operations */                // Draw scene
nema_cl_unbind();               // Unbind CL (optionally)
nema_cl_submit(&cl);            // Submit CL for execution
```

4.5 Drawing

Finally, after setting up the above, the CL contains all the information needed to blit an image or fill a Geometric Primitive with color. Nema GFX Library has a rich set of functions to do that. For the example above, let's assume that we need to draw a background 320x240 image, starting at screen coordinate (0, 0) (the upper left corner of the canvas), and then draw a red rectangle that starts at point (20, 30) with dimensions 100x200:

```
nema_cmdlist_t cl = nema_cl_create();  // Create a new CL
nema_cl_bind(&cl);                     // Bind it
                                       // Bind Destination Texture:
nema_bind_dst_tex(DST_IMAGE,           // Destination address
                  320, 240,            // width, height
                  NEMA_RGBA8888,       // Image format (32bit, rgba)
                  320*4);              // Stride in bytes (width*4
                                       // bytes per pixel)
                                       // Bind Foreground Texture:
nema_bind_src_tex (SRC_IMAGE,          // Source address
                  320, 240,            // width, height
                  NEMA_RGBA8888,       // Image format (32bit, rgba)
                  320*4,               // Stride in bytes (width*4
                                       // bytes per pixel)
                 NEMA_FILTER_PS);      // Do point sampling (default
option) nema_set_clip(0, 0, 320, 240);    // Define a 320x240
                                          // Clipping Rectangle
```

```
nema_set_blend_blit(NEMA_BL_SRC);   // Program the Core to draw
                                    // the source texture without
                                    // blending it with the
                                    // destination texture
nema_blit(0, 0);                    // Blit the bound Source Texture
                                    // to Destination Texture
nema_set_blend_fill(NEMA_BL_SRC);   // Program the Core to fill the
                                    // Geometric Primitive without
                                    // blending it with the
                                    // destination texture

nema_fill_rect(20, 30, 100, 200, RED); // Fill a rectangular area
                                       // with red color
nema_cl_unbind();               // Unbind CL (optionally)
nema_cl_submit(&cl);            // Submit CL for execution
```

The overall process described in the previous paragraphs, produces the output presented in the following figures.

Figure 4-2: The original empty framebuffer

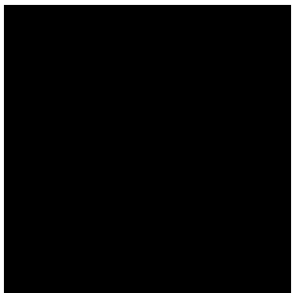
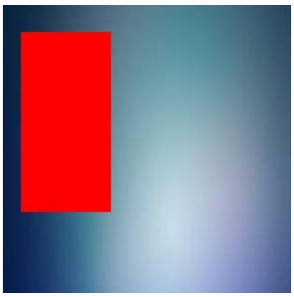


Figure 4-3: Background

Figure 4-4: Final output of the drawing process



SECTION

5

Color Modes and Binding Textures

5.1 Color Modes

Nema GPUs natively support a large set of texture formats. They perform fast read and write operations by executing on the fly color conversion/decompression. Native formats expand from full 32-bit RGBA to 1-bit black and white colors, including an optional proprietary compressed 4-bit-per-pixel lossy format. All formats can be used as source or destination textures. The list of all supported formats is presented in Table 5-1.

Table 5-1: Supported Formats

Color Mode	Description
RGBX8888	32-bit color with no transparency
RGBA8888	32-bit color with transparency
XRGB8888	32-bit color with no transparency
ARGB8888	32-bit color with transparency
RGBA5650	16-bit color with no transparency
RGBA5551	16-bit color with 1-bit transparency
RGBA4444	16-bit color with transparency
A8	8-bit translucent color
L8	8-bit gray scale (luminance) color
A1	1-bit color (black or white) (source only)
L1	1-bit color (luminance) (source only)
UYVY	UYVY 32-bit color (source only)
ABGR8888	32-bit color with transparency

Table 5-1: Supported Formats *(Continued)*

Color Mode	Description
XBGR8888	32-bit color with no transparency
BGRA8888	32-bit color with transparency
BGRX8888	32-bit color with no transparency
TSC4	16-pixel/64-bit proprietary compressed
BGR565	16-bit color with no transparency
TSC6	16-pixel/96-bit proprietary compressed
TSC6A	16-pixel/96-bit proprietary compressed with transparency
A1LE	1-bit color Little-Endian (source only)
A2LE	2-bit translucent color Little-Endian (source only)
A4LE	4-bit translucent color Little-Endian (source only)
L1LE	1-bit color (luminance) Little-Endian (source only)
L2LE	2-bit grayscale color (luminance) Little-Endian (source only)
L4LE	4-bit grayscale color (luminance) Little-Endian (source only)
A2	2-bit translucent color (source only)
A4	4-bit translucent color (source only)
L2	2-bit grayscale (luminance) color (source only)
L4	4-bit grayscale (luminance) color (source only)
RGBA3320	8-bit color with no transparency
BGR24	24-bit color with no transparency
RGB24	24-bit color with no transparency
RGBA2222	8-bit color with transparency (Available if Hardware enabled)
ABGR2222	8-bit color with transparency (Available if Hardware enabled)
BGRA2222	8-bit color with transparency (Available if Hardware enabled)
ARGB2222	8-bit color with transparency (Available if Hardware enabled)
AL88	16-bit color (Available if Hardware enabled)
AL44	8-bit color (Available if Hardware enabled)
ARGB1555	16-bit color with transparency (Available if Hardware enabled)
ARGB4444	16-bit color with transparency (Available if Hardware enabled)
BGRA5551	16-bit color with transparency (Available if Hardware enabled)
ABGR1555	16-bit color with transparency (Available if Hardware enabled)
BGRA4444	16-bit color with transparency (Available if Hardware enabled)

Table 5-1: Supported Formats *(Continued)*

Color Mode	Description
ABGR4444	16-bit color with transparency (Available if Hardware enabled)
TSC12	4-pixel/48-bit proprietary compressed (Available if Hardware enabled)
TSC12A	4-pixel/48-bit proprietary compressed with transparency (Available if Hardware enabled)
TSC6AP	16-pixel/96-bit proprietary compressed with transparency with better quality (Available if Hardware enabled)

5.2 Binding Textures

The Color Modes discussed in *Section 5.1 Color Modes on page 23* can be used for both source and destination textures. Nema Pico incorporates 4 texture slots allowing 4 textures to be bound simultaneously. This means that the hardware allows a single Shader to read from and/or write to 4 different textures. These textures have to be bound before the Shader is submitted for execution. Nema GFX uses pre-assembled Shaders to perform blending operations. These Shaders are built upon the conventions of Table 5-2.

Table 5-2: Shader Conventions

Texture Slot	Texture Usage
`NEMA_TEX0`	Destination/Background Texture
`NEMA_TEX1`	Foreground Texture
`NEMA_TEX2`	Background Texture
`NEMA_TEX3`	Depth Buffer

For further clarifying the aforementioned conventions, let's assume the following example: We need to draw the scene shown in Figure 5-1, consisted of a background image and two icons. The scene requires the 3 source textures shown in Figure 5-2 on page 26, and a Framebuffer (e.g., the destination texture).

Figure 5-1: Rendered scene with two icons

Figure 5-2: Scene Textures

The scene will be drawn in 3 passes: first draw the background, then draw **Icon_0** and last draw **Icon_1**.

1. First Pass: Draw Background
    a. Bind the Framebuffer to **NEMA_TEX0** slot.
    b. Bind the Background texture to **NEMA_TEX1** slot..
    c. Set corresponding blending mode (**NEMA_BL_SRC**).
    d. Blit **NEMA_TEX1** slot to **NEMA_TEX0** slot.
2. Second Pass: Draw **Icon_0**
    a. Bind the Framebuffer to **NEMA_TEX0** slot.
    b. Bind Icon0 to **NEMA_TEX1** slot.
    c. Set corresponding blending mode (e.g., **NEMA_BL_SRC_OVER**).
    d. Blit **NEMA_TEX1** slot to **NEMA_TEX0** slot.
3. Third Pass: Draw **Icon_1**
    a. Bind the Framebuffer to **NEMA_TEX0** slot.
    b. Bind Icon1 to **NEMA_TEX1** slot.
    c. Set corresponding blending mode (e.g., **NEMA_BL_SRC_OVER**).
    d. Blit **NEMA_TEX1** to **NEMA_TEX0** slot.

If we build a single Command List for the above operations, we need to bind the Framebuffer only once, in the beginning of the draw process. This sequence will result in 3 blitting operations. However, there is a more efficient approach, described in Scenario 2.

Scenario 2

The scene will be drawn in 2 passes: first draw **Icon_0** on top of the background and then draw **Icon_1**.

1. First Pass: Draw **Icon_0** on top of the Background
   a. Bind the Framebuffer to **NEMA_TEX0** slot
   b. Bind Icon0 to **NEMA_TEX1** slot
   c. Bind the Background texture to **NEMA_TEX2** slot
   d. Set corresponding blending mode (e.g., **NEMA_BL_SRC_OVER**)
   e. Blit **NEMA_TEX1** on top of **NEMA_TEX2** slot to **NEMA_TEX0** slot
2. Second Pass: Draw **Icon_1**
   a. Bind the Framebuffer to **NEMA_TEX0** slot
   b. Bind Icon1 to **NEMA_TEX1** slot
   c. Set corresponding blending mode (e.g., **NEMA_BL_SRC_OVER**)
   d. Blit **NEMA_TEX1** slot to **NEMA_TEX0** slot

5.2.1 Texture Binding Functions

Use this function to bind the destination texture. It implies binding to **NEMA_TEX0** slot:

```
void nema_bind_dst_tex(uint32_t baseaddr_phys,
uint32_t width,                          uint32_t
height,                         nema_tex_format_t
format,                         int32_t stride)
```

Use the following function to bind the foreground (source) texture. This is needed only for Blit operations. Fill operations don't have a source texture. This function implies binding to **NEMA_TEX1** slot:

```
void nema_bind_src_tex(uint32_t baseaddr_phys,
uint32_t width,                         uint32_t
height,                        nema_tex_format_t
format,                        int32_t stride)
```

The following function binds a background texture to **NEMA_TEX2** slot. This is needed when the Blending Mode to be used does not use the destination texture (**NEMA_TEX0**) as background texture at the blending operation.

```
void nema_bind_src2_tex(uint32_t baseaddr_phys,
                        uint32_t width,
uint32_t height,
nema_tex_format_t format,
                       int32_t stride,
nema_tex_mode_t mode)
```

5.2.2 Look Up Table (LUT) based Textures

Nema GFX provides the option to use LUTs (look up tables) in texture blitting.

To translate an image to a LUT, a color palette table and an indices table are needed. The supported palettes are 2,4,16,256 colors. Color palette format can be any supported by Nema GFX color format. The indices table format is L* and the size depends on the color palette table format.

For example, when you use a 4 color palette, the index needs 2 bits for representation. So the index table needs to be either of L2 or L2 little endian. So, the supported indices tables formats are:

- NEMA_L1,NEMA_L1LE
- NEMA_L2,NEMA_L2LE
- NEMA_L4,NEMA_L4LE
- NEMA_L8

Nema Pix-presso has the functionality to transform a texture to palette and indices tables (target format LUT-2, LUT-4, LUT-16, LUT-256). For further information, refer to the Nema Pix-presso manual. Nema Pix-presso produces the palette table to **.rgba** format and the indices table to **.gray** format.

The following API call is used to bind the color palette and the indices table: **nema_bind_lut_tex()**. In Table 5-3 the relation between color palette table and indices table is mentioned.

In the following table, the provided above example is analyzed. The texture size is 308x313.

Table 5-3: Texture Analysis

Color Palette	Bits to Represent	Index Type	Index Array Size
256	8	L8	308*313=96,404 bytes
16	4	L4	(308/2)*313=48,202 bytes
4	2	L2	(308/4)*313=24,101 bytes
2	1	L1	(308/8)*313=12,207 bytes

ATTENTION: Note that since 308 is not a multiple of 8, a padding is added in every line. 308/8=38.5, so the actual size is 39*313. So when the width is not a multiple of the requested color palette, padding is required.

5.3 Masking

Textures can be used in order to apply masking on other textures. In this case, a mask texture is combined with a source texture so that the rendering output is derived from the intersection of the mask and the source image. The texture used for the mask must be in alpha-only format (A1, A2, A4 or A8) while the source texture can be in any format. In addition, the mask must be bound to the TEX3 slot as shown in the following snippet (A8 format is assumed for the mask):

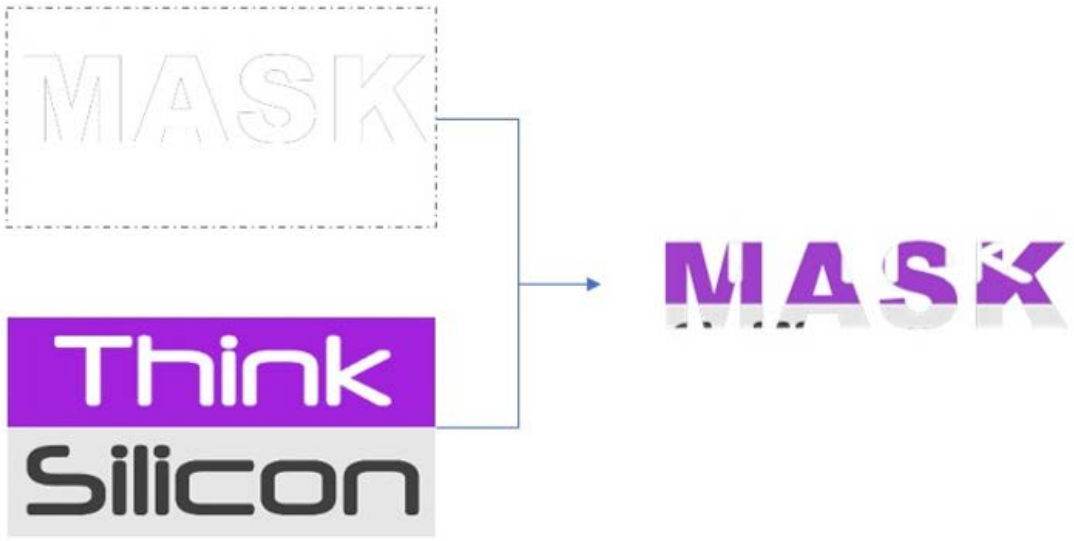
```
void nema_bind_tex( NEMA_TEX3, mask_baseaddr_phys, mask_width,
mask_height, NEMA_A8,  mask_stride, 0);
```

It must be noted that the resolution of the mask must be the same as the resolution of the source texture. The respective blending operator (**NEMA_BLOP_STENCIL_TXTY**) needs also to be used in conjuction with the desired blending mode in order to apply the mask on the source texture, for example by using:

```
nema_set_blend_blit(NEMA_BL_SIMPLE | NEMA_BLOP_STENCIL_TXTY);
```

The following picture, illustrates the concept of masking. On the left hand side there is the mask and the source texture. As mentioned earlier these must have the same resolution. On the right hand side, there is the visual result of applying the mask on the source texture. A relevant example can be found in **NemaGFX_SDK/ examples/NemaGFX/ blit_mask** directory.

Figure 5-3: Mask, source texture and the result of applying the mask on the source



SECTION

6

Geometry Primitives

In computer graphics, Geometry Primitives are the basic geometric shapes that a system can draw. These primitives are generated by the Rasterizer module (internal component of the GPU). The Rasterizer generates the fragments contained inside the Primitive and feeds them to the Programmable Core for processing. The following Geometry Primitives are supported:

- Points
- Lines
- Filled Triangles
- Filled Rectangles
- Filled Quadrilaterals

All the aforementioned Primitives can be processed by the Programmable Core to do simple operations (e.g filling with a constant color or gradient, blitting etc) or more advanced ones (e.g. blurring, edge detection etc). The Geometry Primitives mentioned earlier, enable the support of more complex shapes. Such shapes are:

- Triangles
- Rectangles
- Polygons
- Filled Polygons
- Triangle Fans
- Triangle Strips
- Circles
- Filled Circles
- Arcs
- Rounded Rectangles

SECTION

7

Blending

Alpha blending is a basic process in computer graphics. It refers to a convex combination of two colors, a translucent source (foreground) and a destination (background) one, allowing transparency effects. The basic blending algorithms described by Porter and Duff[1], define a set of mathematical operations for the Color channels (RGB) and the Alpha (transparency) channel of a fragment. Blending process is essential for rendering fonts and/or creating GUIs. In Nema graphics pipeline, blending is carried-out in the Graphics Core.

7.1 Blending in the Graphics Core

Blending operations are calculations between colors inside the GPU's Graphics core. Nema GFX library provides a lightweight and user-friendly interface that employs pre-assembled commands to create a powerful set of blending algorithms.

The Graphics Core can be programmed through the following functions:

```
void nema_set_blend_fill(nema_blend_mode_t blending_mode)
void nema_set_blend_blit(nema_blend_mode_t blending_mode)
```

These functions are defined in NemaGFX_blender.h file. They should be used on fill and blit operations respectively. The blending_mode argument is possible to be a predefined blending mode or a more refined User Defined Mode.

[1] Thomas Porter and Tom Duff, Compositing Digital Images, Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '84, ACM, New York 1984.

7.2 Notations and Conventions

Blending requires a series of calculations between the source (foreground) and destination (background) color fragments for producing the final color, which will be written in memory. The Color and Alpha channels are noted as follows:

- **Sc** -Source Color
- **Sa** - Source Alpha
- **Sf** - Source Blend Factor (multiplier)
- **Dc** - Destination Color
- **Da** - Destination Alpha
- **Df** - Destination Blend Factor (multiplier)
- **Fc** - Final Color
- **Fa** - Final Alpha
- **Cc** - Constant Color
- **Ca** - Constant Alpha

The Color and Alpha values range from 0 to 1, therefore each calculation result is also clamped to the same range. For consistency reasons Color and Alpha calculations are always described separately, as in some cases these calculations are not identical. When a constant color is used (noted as Cc and Ca), it can be set using the following function:

```
void nema_set_const_color(uint32_t rgba)
```

7.3 Predefined Blending Modes

Predefined Blending Modes is a set of commonly used modes, each implying different calculations between the source and destination colors for Color (RGB) channel and Alpha channel respectively. Table 7-1 presents the complete list of the available Predefined Blending Modes along with the corresponding calculations that produce the final fragment color. Figure 7-1 on page 33 shows the result for each Predefined Blending Mode.

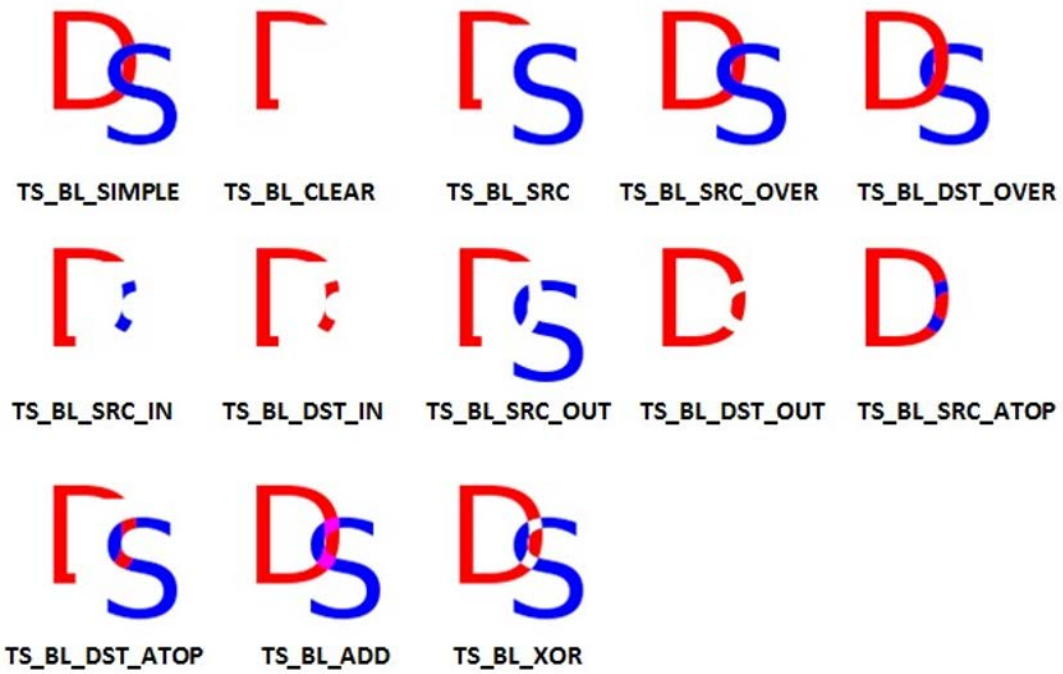
Table 7-1: Predefined Blending Modes

Predefined Blending Modes	RGB	ALPHA
NEMA_BL_SIMPLE	Sc*Sa + Dc*(1-Sa)	Sa*Sa + Da*(1-Sa)
NEMA_BL_CLEAR	0	0
NEMA_BL_SRC	Sc	Sa
NEMA_BL_SRC_OVER	Sc + Dc*(1-Sa)	Sa + Da*(1-Sa)

Table 7-1: Predefined Blending Modes *(Continued)*

Predefined Blending Modes	RGB	ALPHA
NEMA_BL_DST_OVER	Sc*(1-Da) + Dc	Sa*(1-Da) + Da
NEMA_BL_SRC_IN	Sc*Da	Sa*Da
NEMA_BL_DST_IN	Dc*Sa	Da*Sa
NEMA_BL_SRC_OUT	Sc*(1-Da)	Sa*(1-Da)
NEMA_BL_DST_OUT	Dc*(1-Sa)	Da*(1-Sa)
NEMA_BL_SRC_ATOP	Sc*Da + Dc*(1-Sa)	Sa*Da + Da*(1-Sa)
NEMA_BL_DST_ATOP	Sc*(1-Da) + Dc*Sa	Sa*(1-Da) + Da*Sa
NEMA_BL_ADD	Sc + Dc	Sa + Da
NEMA_BL_XOR	Sc*(1-Da) + Dc*(1-Sa)	Sa*(1-Da) + Da*(1-Sa)

Figure 7-1: Predefined Blending Modes



For instance, drawing a translucent red rectangle would require the following calls:

```
nema_set_blend_fill(NEMA_BL_SIMPLE); nema_fill_rect(52, 52, 14,
4, nema_rgba(0xff, 0, 0, 0x80));
```

TIP: When in doubt, usually the NEMA_BL_SIMPLE blending mode is the safest choice.

The overall process starts with an empty Framebuffer, shown in Figure 7-2 on page 34. Next, the process continues by blitting the textures shown in Figure 7-3 on page 35.

Figure 7-2: Original Framebuffer before Blending

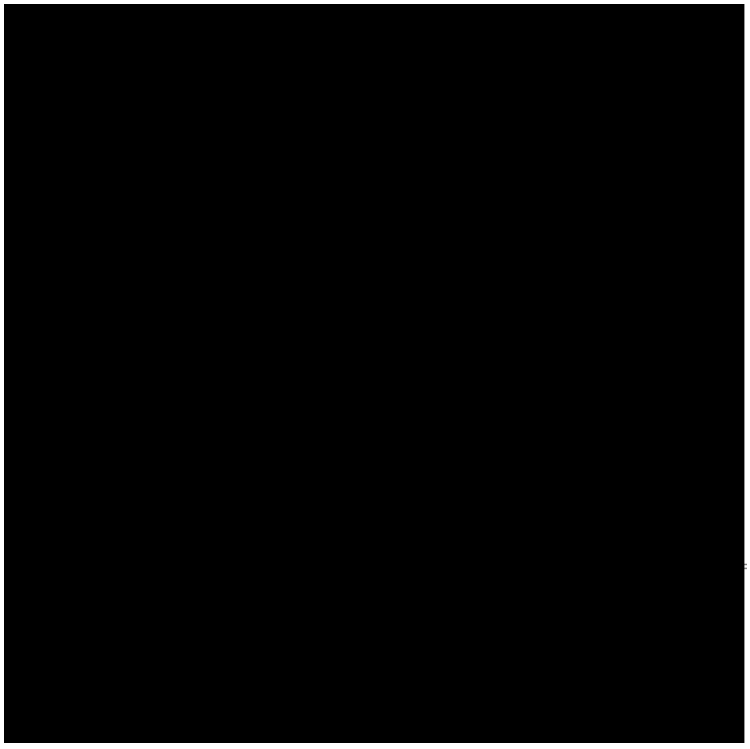


Figure 7-3: Scene Textures

7.4 User Defined Modes

Developers are able to create custom blending modes by using different factors for the source (foreground) and destination (background) color, through the following function:

```
nema_blend_mode_tnema_blending_mode(nema_blend_factor_tsrc,
nema_blend_factor_t dst,
nema_blend_op_t ops)
```

The ops argument of the above function refers to additional operations, presented in *Section 7.5 Additional Operations on page 37*, therefore for the time being should be set to zero (0). The calculations result to the final color, using the following equations:

```
Fc = Sc * Sf + Dc * Df
Fa = Sa * Sf + Da * Df
```

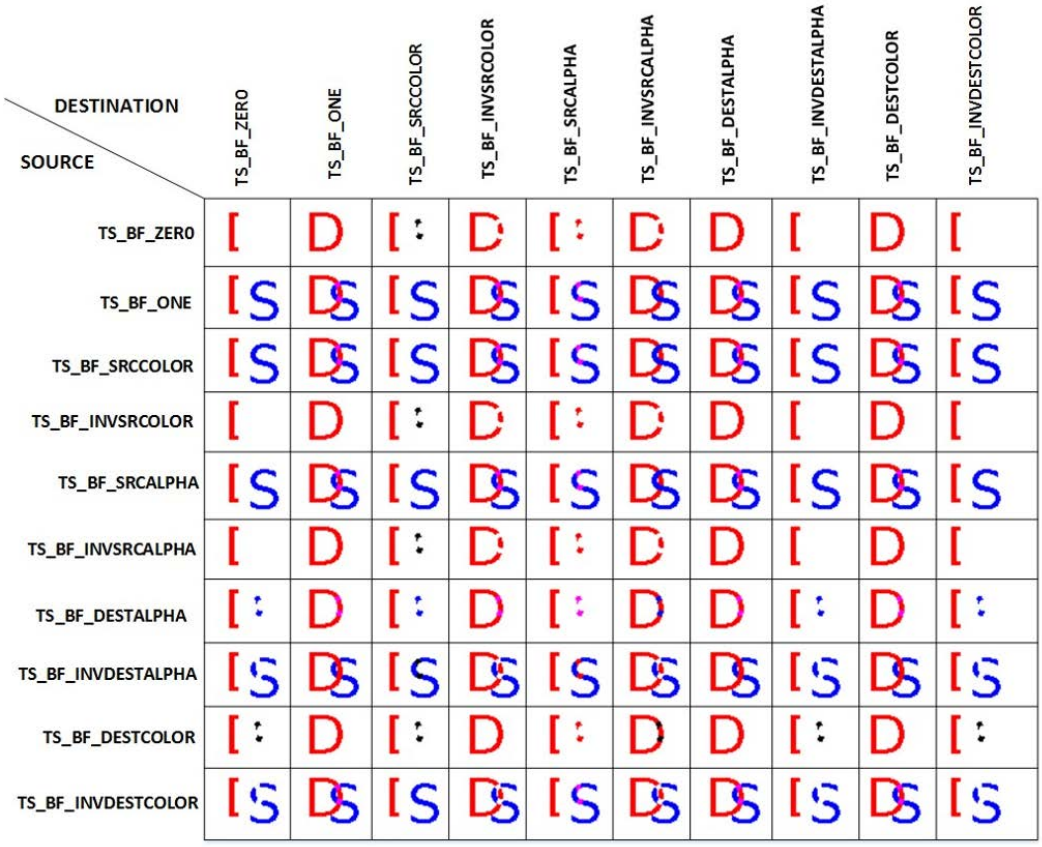
The available Blend Factors are listed in Table 7-2. Figure 7-4 on page 36 shows the available custom blending modes. As a result, the previous example is possible to be rewritten as:

```
nema_set_blend_fill(nema_blending_mode(NEMA_BF_SRCALPHA,
NEMA_BF_INVSRCALPHA, 0)); nema_fill_rect(52, 52, 14, 4,
nema_rgba(0xff, 0, 0, 0x80));
```

Table 7-2: Blend Factors

Blend Factors (Sf or Df)	Mode
NEMA_BF_ZERO	0
NEMA_BF_ONE	1
NEMA_BF_SRCCOLOR	Sc
NEMA_BF_INVSRCCOLOR	(1-Sc)
NEMA_BF_SRCALPHA	Sa
NEMA_BF_SRC_INVSRCALPHA	(1-Sa)
NEMA_BF_DESTALPHA	Da
NEMA_BF_INVDESTALPHA	(1-Da)
NEMA_BF_DESTCOLOR	Dc
NEMA_BF_INVDESTCOLOR	(1-Dc)
NEMA_BF_CONSTCOLOR	Cc
NEMA_BF_CONSTALPHA	Ca

Figure 7-4: User-Defined Blending Modes



7.5 Additional Operations

NEMA GFX Library allows the following operations, which can be applied together with the previously mentioned blending modes through the aforementioned function:

```
nema_blend_mode_tnema_blending_mode(nema_blend_factor_tsrc,
nema_blend_factor_t dst,
nema_blend_op_t ops)
```

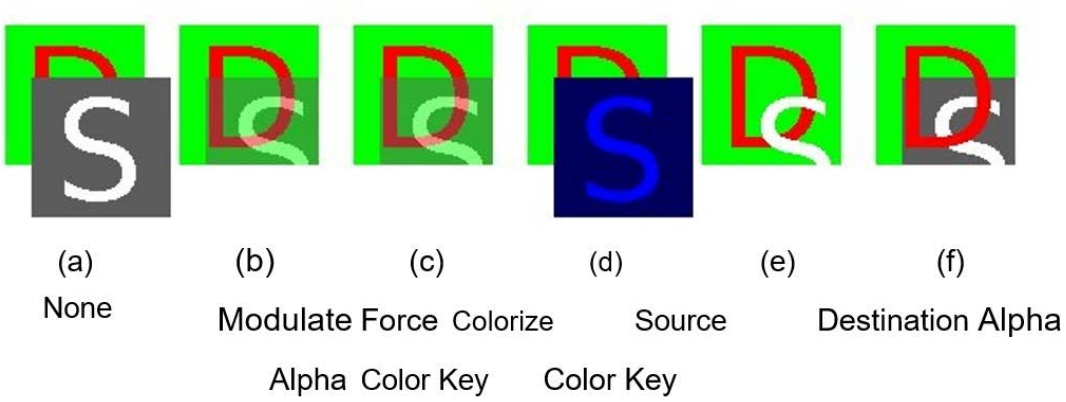
The additional supported operations are listed in Table 7-3 on page 37. An example of the overall process can be found in Figure 7-5 and Figure 7-6 on page 38.

Table 7-3: Ops Arguments

Ops Arguments	Description
SRC_MODULATE_A	Multiply source alpha channel with Ca constant before blending. Ca is defined by calling NemaGFX_set_const_color().
SRC_FORCE_A	Replace source alpha channel with Ca before blending. Overrides SRC_MODULATE_A option. Ca is defined by calling NemaGFX_set_const_color().
SRC_COLORIZE	Multiply source color channels (RGB) with Cc before blending. Cc is defined by calling NemaGFX_set_const_color().
SRC_COLORKEY	Ignore fragment when source color matches the source color key, which is defined by calling NemaGFX_set_src_color_key().
DST_COLORKEY	Ignore fragment when destination color matches the destination color key, which is defined by calling NemaGFX_set_dst_color_key().

Figure 7-5: Source Textures

Figure 7-6: Additional Operations Example



SECTION

8

Interpolators

Nema GFX's interpolators can be used to interpolate the colors of a triangle's vertices and produce a three color gradient effect.

The following API call activates the GPU interpolators:

```
name_enable_gradient(1);
```

The following API calls program the GPU interpolators to produce gradient color.

```
nema_interpolate_rect_colors()
nema_interpolate_tri_colors()
```

For example, if you want to produce a triangle with the following gradient:

- Vertex 1: blue
- Vertex 2: red
- Vertex 3: green you have to do the following:

```
color_var_t col_red = {0xff, 0x00, 0x00, 0xff};
color_var_t col_blue = {0x00, 0xff, 0x00, 0xff};
color_var_t col_green = {0x00, 0x00, 0xff, 0xff};
int width = 50;
int height = 50;
int x = 0;
int y = 0;
nema_enable_gradient(1);

nema_interpolate_tri_colors(x, y, x + width, y, x + width, y + height,
&col_red, &col_blue, &col_green);

nema_raster_triangle_f(x, y, x + width, y, x + width, y + height)
//nema_fill_triangle(x, y, x + width, y, x + width, y + height, 0);
// will also work it will ignore the black color when
nema_enable_gradient is enabled
```

Figure 8-1: Triangle with Color Interpolation

ATTENTION: In NEMA GPUs, the order of the vertices doesn't matter, as long as the upper left coordinate of the triangle is always a part of the nema_interpolate_* function. Otherwise, the outcome will be inconsistent.

There is a known hardware limitation regarding clipping gradients. The upper left vertex of the triangle that will be rendered must be inside the clipping rectangle.

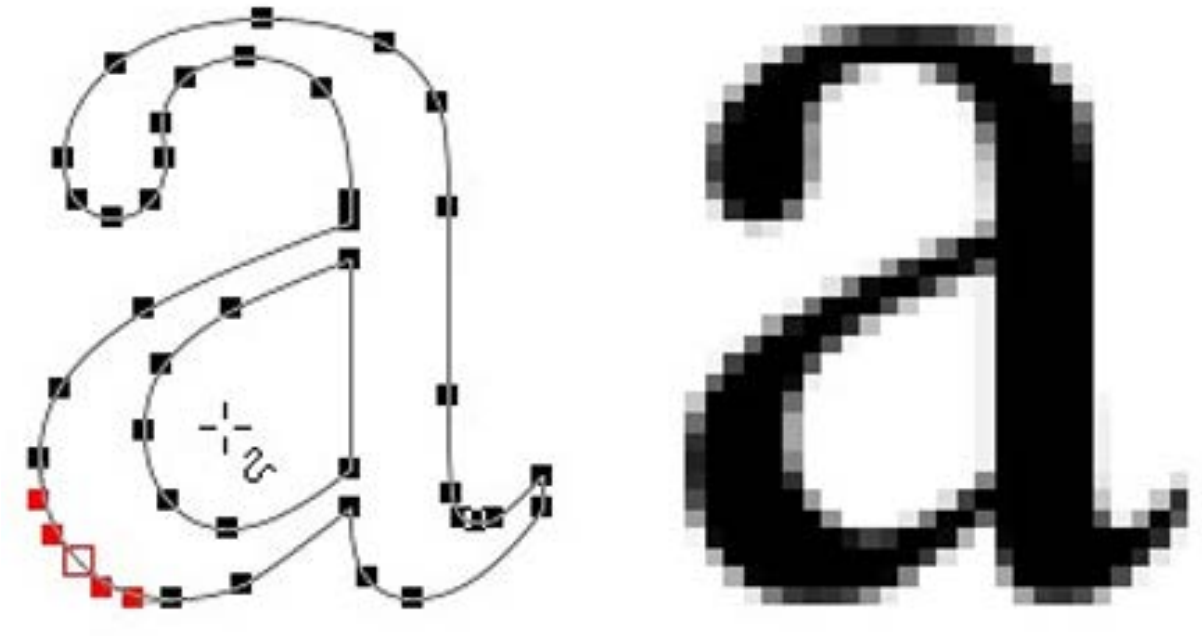
SECTION

9

Fonts

Drawing text on the screen is an important element of any Graphical User Interface. To draw a string, you will need a Typeface, the text to be drawn and some attributes on how the text is to be displayed. Typefaces are sourced in TrueType (TTF) file type, which contains scalable representations of typefaces described as vector curves. Scalable fonts are converted to raster fonts (bitmap fonts) by rasterization to a particular size and format. Raster fonts are drawn on the screen as a series of images with each letter drawn after the other using the correct letter width. To facilitate this process, Nema GFX Library handles text display and alignment using special functions.

Figure 9-1: Vector and Bitmap Fonts



The first step is to convert a TrueType font to Bitmap font. This is done off-line with the provided **nema_font_convert** tool. The following information is printed when running the tool with no arguments:

```
NAME
       nema_font_convert - convert TTF fonts to .bin and .h, compatible with
NEMA|gfx  graphics API
SYNOPSIS         nema_font_convert [OPTION]...
[FILE]...

DESCRIPTION
        Convert TTF fonts to .bin, .c and .h, compatible with NEMA|gfx
graphics API
        -s, --size
font size
        -b, --bpp
                set bits per pixel, e.g.: -b 8
        -1, --bpp1
                set bits per pixel to 1
        -2, --bpp2
                set bits per pixel to 2
        -4, --bpp4
                set bits per pixel to 4
        -8, --bpp8
                set bits per pixel to 8
        -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
```

The resulted font files support UTF-8 characters, 1, 2, 4 or 8 bits per pixel. The conversion generates a .c, a .h and a .bin file. After a conversion is performed successfully, the memory requirements for the converted font are reported. The total memory needed for the converted font, consists of the graphics memory (memory needed for the font bitmaps) along with the host memory (font related parameters needed by the CPU). The font bitmaps are included both inside the .c and the .bin files. The bitmaps can be excluded from the .c file by defining **NEMA_FONT_LOAD_FROM_BIN** (#define **NEMA_FONT_LOAD_FROM_BIN**). In this case, the .bin has to be used in order to access the font bitmaps. Otherwise, the .bin file can be ignored and the font bitmaps can be accessed by the .c file. The latter scenario can be particularly useful for platforms without a file system, as they are not able to have the .bin file stored. When compiling the project, add .c file to the compiled sources and include the .h file where needed.

Before drawing text, we need to bind the data structure of a typeface with the following function:

```
void nema_bind_font (font_t *fontdata);
```

In order to draw some text, use **nema_print()**:

```
void nema_print(const char *str, int x, int y,
int w, int h, uint32_t fg_col, uint32_t
 align);
```

The arguments of the above function refer to:

- The text to be drawn
- The (x, y) screen coordinates that the text should be drawn to
- The width and height of the text area
- The color of the text
- Text alignment

The following parameters may be passed as the last argument (align) to define vertical and horizontal alignment:

Horizontal alignment:

9 Fonts

- NEMA_ALIGNX_LEFT
- NEMA_ALIGNX_RIGHT
- NEMA_ALIGNX_CENTER

Vertical alignment:

- NEMA_ALIGNY_TOP
- NEMA_ALIGNY_BOTTOM
- NEMA_ALIGNY_CENTER
- NEMA_ALIGNY_JUSTIFY

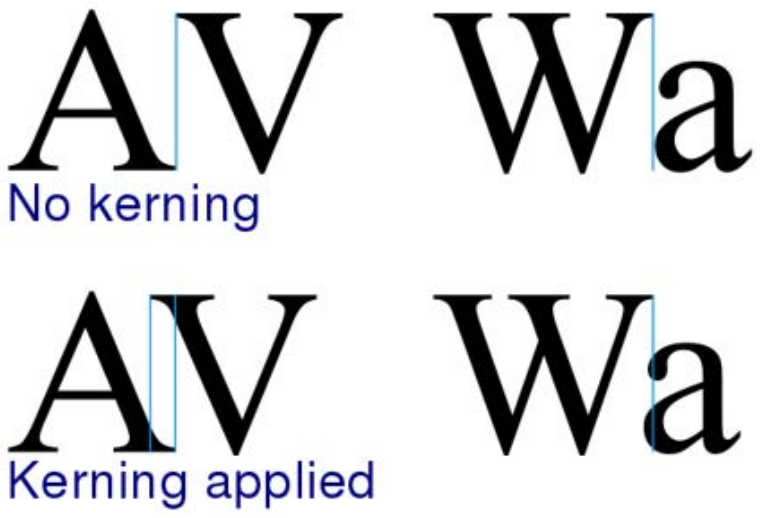
Wrapping:

- NEMA_TEXT_WRAP

9.1 Kerning

In font rendering, NEMA GFX supports kerning. Kerning is the process of adjusting the spacing between characters in a proportional font[1], as depicted in the following figure.

Figure 9-2: Text rendering with and without kerning



Kerning is used for adjusting the horizontal spacing in character pairs. When converting a TTF to a Bitmap font, using the nema font convert tool, such pairs are automatically extracted (using the -k or --kerning command line arguments). The spacing offsets are retrieved from the kern and GPOS tables contained in the TTF file. In case whereas a kerning pair exists in both of these tables, the offset contained in the GPOS will be used. After the kerning pairs are extracted, they are stored in the generated .c file. As a consequence, kerning will increase the memory size needed by the generated font.

[1] Kerning. (2020). Retrieved September 11, 2020, from https://en.wikipedia.org/wiki/Kerning.

SECTION

10

Nema GFX Platform Porting

Library has been designed to be easily portable in a variety of different platforms. This includes systems with or without an operating system. In order to port NEMA GFX successfully, one must take into account the target platform and adapt the HAL (Hardware Abstraction Layer) accordingly.

The HAL is a thin layer of the library that:

- Communicates directly with the system hardware and the GPU drivers (when drivers are available)
- Performs the communication between the host and the GPU (access to the GPU registers)
- Handles interrupts
- Implements the memory management scheme (memory allocation, deallocation, mapping and unmapping)

10.1 Platform Specific HAL

Each target platform has:

- A **nema_sys_defs.h** header file located in NemaGFX_SDK/platforms/PLATFORM/common/ folder
- A separate **nema_hal.c** file, located in NemaGFX_SDK/platforms/PLATFORM/NemaGFX folder

In order to port NEMA GFX to a new platform, it is advised that the corresponding source files of an already ported platform are used as templates. Especially for the case of Bare metal platforms, there is a template platform called **baremetal_generic** within the respective directory.

10.2 nema_sys_defs.h

nema_sys_defs.h contains all the global definitions and inclusions that are platform specific. For example, Nema GFX uses a set of integer types defined inside the C standard **stdint.h** header. If the platform's compiler supports the **stdint.h**, then it should be included in the **nema_sys_defs.h** header file. If the compiler does not support the stdint.h, then the following types should be defined:

- int8_t, uint8_t,
- int16_t, uint16_t,
- int32_t, uint32_t,
- int64_t, uint64_t

If Nema GFX is compiled for a platform that runs multiple processes and/or multiple threads, the following definitions should be added respectively:

```
#define NEMA_MULTI_PROCESS
#define NEMA_MULTI_THREAD
```

In addition, the following parameters should be defined inside **nema_sys_defs.h** if they are not defined somewhere else (e.g., the kernel driver takes care off them, or they are defined in other platform specific files):

- **NEMA_BASEADDR**, the GPU registers memory base address
- **VMEM_BASEADDR**, the graphics (video) memory base address
- **VMEM_SIZE**, the size of the graphics memory For instance the following defines:

  #define NEMA_BASEADDR 0x43d00000U
  #define VMEM_BASEADDR 0x30000000U
  #define VMEM_SIZE (15*1024*1024)

mean that the base address of the GPU registers is address 0x43d00000, the base address of the graphics memory is address 0x30000000 (both addresses refer to physical memory space), and the graphics memory size is 15 MB.

10.3 nema_hal.c

nema_hal.c contains all the platform specific functions that implement hardware register read/write operations, interrupt handling, memory management and mutex support. It acts as a thin layer which incorporates all the platform dependent portions of a Nema GFX implementation.

10.3.1 System Initialization

The **nema_init()** function initializes the Nema GFX and calls the **nema_sys_init()** function which is responsible for the system initialization. The system initialization includes:

- GPU register memory mapping
- Graphics Memory mapping
- Mutex initialization
- Ring Buffer allocation and initialization

10.3.2 Register Read/Write

The host CPU writes to and reads from the GPU's configuration registers. The functions **nema_reg_read()** and **nema_reg_write()** are used for the communication of the CPU with the GPU.

When the target platform does not need memory virtualization (e.g., BareMetal systems), the access to the GPU's registers is straightforward by using the register's physical memory address. The only prerequisite in this case, is the appropriate memory mapping of the GPU registers to the system's main memory.

The following examples illustrate the register read and write operations for BareMetal systems.

```
uint32_t  nema_reg_read(uint32_t reg) {
    uint32_t *ptr = (uint32_t *)(nema_regs + reg);
    return *ptr; }
void nema_reg_write(uint32_t reg, uint32_t value) {
    uint32_t *ptr = (uint32_t *)(nema_registers_base_addr + reg);
    *ptr = value;
}
```

On platforms that support virtual memory (e.g., Linux, Android), the GPU registers' physical addresses must be mapped to the virtual memory addresses before an access is attempted. This can be performed in three ways.

The first one is to utilize the GPU driver (if applicable), and perform the memory mapping using the mmap system call.

```
nema_regs_base_virt = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE,
MAP_SHARED, nema_fd, 0);
```

The second way in Linux systems, is to use the /dev/mem instead of the uNema driver as shown in the following example.

```
 // Map and initialize Graphics Memory
    nema_devmem_mmap(&gmem_base_virt, gmem_base_phys, gmem_size,
"VideoMemory");
    // Memory map nema registers
```

```
    nema_devmem_mmap((void **)&nema_regs, nema_regs_base_phys,
0x1000, "NemaRegis▶
ters");
```

The third way in Linux systems, is to perform the read/write operations to the registers via respective IOCTL calls to the uNema driver.

```
uint32_t  nema_reg_read(uint32_t reg) {     unema_ioctl_read-
write_t ioctl_rw;
    ioctl_rw.reg = reg;
    ioctl(nema_fd, UNEMA_IOCTL_REG_READ, &ioctl_rw);
    return ioctl_rw.value;
}
void nema_reg_write(uint32_t reg, uint32_t value) {
    unema_ioctl_readwrite_t ioctl_rw;
    ioctl_rw.reg   = reg;
ioctl_rw.value = value;
    ioctl(nema_fd, UNEMA_IOCTL_REG_WRITE, &ioctl_rw);
}
```

10.3.3 Interrupt Handling

The interrupt handler is executed when an interrupt is triggered by the GPU. Its purpose is to awaken suspended processes and clear the interrupt.

When the GPU kernel driver is available (Linux and Android systems), the interrupt handler is defined in the uNema driver. When the GPU kernel driver is not available (BareMetal and RTOS based systems), the interrupt handler has to be defined in the nema_hal.c file.

A typical interrupt handler for BareMetal systems is the following:

```
void irq_handler (void)
{
      //Clear the interrupt    nema_reg_write(NEMA_INTERRUPT, 0);
}
```

The HAL should implement the **nema_wait_irq** function for interrupt handling. Its purpose is to suspend the process (put to sleep) if the GPU is idle or until the GPU signals an interrupt. Its implementation is platform (CPU) dependent. If the kernel driver is available, then IOCTL calls are used, otherwise it is manually defined according to the CPU platform.

```
void nema_wait_irq(void)
{
    ioctl(nema_fd, UNEMA_IOCTL_WAIT_IDLE);
}
```

Typically, the GPU will raise an interrupt when it has finished executing a Command List. The ID of the last Command List that has been executed can be read from the Configuration Register NEMA_CLID. The function **nema_wait_irq_cl(int**

cl_id) should wait until the content of the **NEMA_CLID** register is greater or equal than the **cl_id** argument.

```
void nema_wait_irq_cl(int cl_id)
{
    while ( nema_reg_read(NEMA_CLID) < cl_id) {
        nema_wait_irq();
    }
}
```

10.3.4 Memory Management

At this stage, the memory management scheme should be implemented. Memory can be considered to consist of two parts: host memory (memory available only to the CPU) and graphics memory (memory available both to the GPU and the CPU).

Host memory can be allocated by using systems' default malloc method:

```
//System malloc
void  nema_host_free     (void *ptr ) {
    free(ptr);
}
void *nema_host_malloc   (unsigned size) {
    return malloc(size);
}
```

If such a method is not available, the graphics memory allocator can be used:

```
//Think Silicon's graphics memory allocator
void  nema_host_free     (void *ptr ) {
    tsi_free(ptr);
}
void *nema_host_malloc   (unsigned size) {
    return tsi_malloc(size);
}
```

Graphics memory is the portion of system memory that the GPU is allowed to have access. In this case, it is necessary that the graphics memory occupies a contiguous physical memory space. On systems that do not support virtual memory (e.g., BareMetal systems), graphics memory can be allocated in the same way as host memory (using malloc, tsi_malloc or other custom memory allocator).

Nema GFX includes the following API calls for memory allocation, deallocation and mapping:

- **nema_buffer_create()** - Allocate memory
- **nema_buffer_create_pool()** - Allocate memory from specific memory pool
- **nema_buffer_map()** - Map allocated memory space for CPU access
- **nema_buffer_unmap()** - Unmap previously mapped memory space
- **nema_buffer_destroy()** - Deallocate memory space

A reference example that illustrates this memory allocation scheme can be found in: Software/NemaGFX_SDK/platforms/lattice_mico32_no_OS/NemaGFX/nema_hal.c.

In this file, functions **nema_buffer_create** and **nema_buffer_destroy** adopt this specific scheme.

When virtual memory is used (e.g., Linux or Android systems), graphics memory should be allocated by the system's contiguous memory allocator (e.g., ION for Android, CMA for Linux).

Linux based system: The uNema kernel driver pre-allocates a contiguous physical memory space using Linux CMA. When Nema GFX is initialized, this memory space is mapped to the process' virtual space. Graphics memory is then managed by Think Silicon's custom memory allocator (tsi_malloc/tsi_free). The code that implements this specific memory management scheme can be found in: Software/NemaGFX_SDK/platforms/zc70x_linux_ioctl/NemaGFX/nema_hal.c

Android based system: When NEMA GFX is initialized, the ION kernel module is opened. On each subsequent graphics memory allocation or deallocation (nema_buffer_create or nema_buffer_destroy), the corresponding IOCTL to the ION module is called. This implementation can be found in: Software/NemaGFX_SDK/platforms/NemaGFX/zc70x_android

Nema GFX supports multiple memory pools. This can be useful for systems with nonuniform memory hierarchy that have different characteristics (e.g. latency, throughput, etc). If such a feature is not needed, **nema_buffer_create_pool()** can be set to redirect to **nema_buffer_create()**:

```
nema_buffer_t nema_buffer_create_pool(int pool, unsigned size) {
#ifdef NEMA_MULTI_MEM_POOLS //defined in nema_sys_defs.h
nema_mutex_lock(MUTEX_MALLOC);
    nema_buffer_t bo;
    bo.base_virt = tsi_malloc_pool(pool, size);
    bo.base_phys = (uint32_t) tsi_virt2phys(bo.base_virt);
    bo.size      = size;
bo.fd        = 0;
    nema_mutex_unlock(MUTEX_MALLOC);
    return bo;
#else
    return nema_buffer_create(size);
#endif
}
```

10.3.5 Support for Multi-Process Multi-Threaded systems

Nema GFX is designed to support a wide variety of systems, from BareMetal to Linux platforms. These systems might also support multiple processes and/or multiple threads within a process. Nema GFX is a graphics API that can manage resource sharing among multiple processes/threads if needed, using mutices and

thread local storage (TLS). To support multiple processes, only mutices are necessary. For multiple threads, both mutices and TSL are needed.

The most obvious shared resource is the GPU itself. Multiple processes/threads send work to the GPU using Command Lists (CL). When a CL is executed by the GPU, it is guaranteed that it will not be interrupted by another CL. Each CL also needs to set the entire state of the GPU and not rely on previous CLs. So the only thing that needs to be taken care of, when multiple processes/threads are running, is the submission of a CL for execution. In this case, a simple mutex is used by the library.

The same applies for memory management. When a buffer is created or destroyed, a mutex ensures that no allocation or deallocation is performed by two processes or threads concurrently.

During system initialization, **nema_init()** should call **nema_sys_init()** only once for each process. New threads within the process must not call **nema_init()** again. The **nema_sys_init()** should not reinitialize the memory allocator nor the Ring Buffer, unless no other running process performed the aforementioned initializations.

For multi-threaded environments, **TLS_VAR** should be defined inside **nema_sys_defs.h**. Nema GFX uses **TLS_VAR** as a prefix to declare thread local variables. For example, when using GCC the following definition should be added:

```
#ifdef NEMA_MULTI_THREAD
#define TLS_VAR __thread
#else
#define TLS_VAR
#endif
```

SECTION

11

Nema GFX Library Functions

This section provides an overview of the implemented functions of the Nema GFX Library.

11.1 Files

Here is a list of all files with brief descriptions:

11.1.1 nema_blender.h

```
#include "nema_sys_defs.h"
#include "nema_graphics.h"
```

Macros

```
#define NEMA_BF_ZERO (0x0U)
#define NEMA_BF_ONE (0x1U)
#define NEMA_BF_SRCCOLOR (0x2U)
#define NEMA_BF_INVSRCCOLOR (0x3U)
#define NEMA_BF_SRCALPHA (0x4U)
#define NEMA_BF_INVSRCALPHA (0x5U)
#define NEMA_BF_DESTALPHA (0x6U)
#define NEMA_BF_INVDESTALPHA (0x7U)
#define NEMA_BF_DESTCOLOR (0x8U)
#define NEMA_BF_INVDESTCOLOR (0x9U)
#define NEMA_BF_CONSTCOLOR (0xaU)
#define NEMA_BF_CONSTALPHA (0xbU)
#define NEMA_BL_SIMPLE ( (uint32_t)NEMA_BF_SRCALPHA |
((uint32_t)NEMA_BF_INVSRCALPHA <<8) )
```

```
#define NEMA_BL_CLEAR ( (uint32_t)NEMA_BF_ZERO /*|
((uint32_t)NEMA_BF_ZERO <<8)*/)

#define NEMA_BL_SRC ( (uint32_t)NEMA_BF_ONE /*|
((uint32_t)NEMA_BF_ZERO <<8)*/)

#define NEMA_BL_SRC_OVER ( (uint32_t)NEMA_BF_ONE |
((uint32_t)NEMA_BF_INVSRCALPHA <<8) )

#define NEMA_BL_DST_OVER ( (uint32_t)NEMA_BF_INVDESTALPHA |
((uint32_t)NEMA_BF_ONE <<8) )

#define NEMA_BL_SRC_IN ( (uint32_t)NEMA_BF_DESTALPHA /*|
((uint32_t)NEMA_BF_ZERO <<8)*/)

#define NEMA_BL_DST_IN (/*(uint32_t)NEMA_BF_ZERO |*/
((uint32_t)NEMA_BF_SRCALPHA <<8) )

#define NEMA_BL_SRC_OUT ( (uint32_t)NEMA_BF_INVDESTALPHA/*|
((uint32_t)NEMA_BF_ZERO <<8)*/ )

#define NEMA_BL_DST_OUT (/*(uint32_t)NEMA_BF_ZERO |*/
((uint32_t)NEMA_BF_INVSRCALPHA <<8) )

#define NEMA_BL_SRC_ATOP ( (uint32_t)NEMA_BF_DESTALPHA |
((uint32_t)NEMA_BF_INVSRCALPHA <<8) )

#define NEMA_BL_DST_ATOP ( (uint32_t)NEMA_BF_INVDESTALPHA |
((uint32_t)NEMA_BF_SRCALPHA <<8) )

#define NEMA_BL_ADD ( (uint32_t)NEMA_BF_ONE |
((uint32_t)NEMA_BF_ONE <<8) )

#define NEMA_BL_XOR ( (uint32_t)NEMA_BF_INVDESTALPHA |
((uint32_t)NEMA_BF_INVSRCALPHA <<8) )

#define NEMA_BLOP_NONE (0U)

#define NEMA_BLOP_RECOLOR (0x00100000U)

#define NEMA_BLOP_LUT (0x00200000U)

#define NEMA_BLOP_STENCIL_XY (0x00400000U)

#define NEMA_BLOP_STENCIL_TXTY (0x00800000U)

#define NEMA_BLOP_NO_USE_ROPBL (0x01000000U)

#define NEMA_BLOP_DST_CKEY_NEG (0x02000000U)

#define NEMA_BLOP_SRC_PREMULT (0x04000000U)

#define NEMA_BLOP_MODULATE_A (0x08000000U)

#define NEMA_BLOP_FORCE_A (0x10000000U)

#define NEMA_BLOP_MODULATE_RGB (0x20000000U)

#define NEMA_BLOP_SRC_CKEY (0x40000000U)

#define NEMA_BLOP_DST_CKEY (0x80000000U)

#define NEMA_BLOP_MASK (0xfff00000U)
```

11.1.1.1 Functions

`static uint32_t nema_blending_mode(uint32_t src_bf, uint32_t dst_bf, uint32_t blops)`

Return blending mode given source and destination blending factors and additional blending operations.

`void nema_set_blend(uint32_t blending_mode, nema_tex_t dst_tex, nema_tex_t fg_tex, nema_tex_t bg_tex)`

Set blending mode. static void

`nema_set_blend_fill(uint32_t blending_mode)`

Set blending mode for filling.

`static void nema_set_blend_fill_compose(uint32_t blending_mode)`

Set blending mode for filling with composing.

`static void nema_set_blend_blit(uint32_t blending_mode)`

Set blending mode for blitting.

`static void nema_set_blend_blit_compose(uint32_t blending_mode)`

Set blending mode for blitting with composing.

`void nema_set_const_color(uint32_t rgba)`

Set constant color.

`void nema_set_recolor_color(uint32_t rgba)`

Set recolor color. Overrides constant color.

`void nema_set_src_color_key(uint32_t rgba)`

Set source color key.

`void nema_set_dst_color_key(uint32_t rgba)`

Set destination color key.

`void nema_debug_overdraws(uint32_t enable)`

Enable/disable ovedraw debugging. Disables gradient and texture, forces blending mode to **NEMA_BL_ADD**.

11.1.1.2 Detailed Description

Macro Definition Documentation

`#define NEMA_BF_CONSTALPHA`

Ca

`#define NEMA_BF_CONSTCOLOR`

Cc

`#define NEMA_BF_DESTALPHA`

Da

`#define NEMA_BF_DESTCOLOR`

Dc

`#define NEMA_BF_INVDESTALPHA`

(1-Da)

`#define NEMA_BF_INVDESTCOLOR`

(1-Dc)

`#define NEMA_BF_INVSRCALPHA`

(1-Sa)

`#define NEMA_BF_INVSRCCOLOR`

(1-Sc)

`#define NEMA_BF_ONE`

1

`#define NEMA_BF_SRCALPHA`

Sa

`#define NEMA_BF_SRCCOLOR`

Sc

`#define NEMA_BF_ZERO`

0

`#define NEMA_BLOP_DST_CKEY`

Apply Destination Color Keying - draw only when dst color matches colorkey

`#define NEMA_BLOP_DST_CKEY_NEG`

Apply Inverse Destination Color Keying - draw only when dst color doesn't match colorkey

#define NEMA_BLOP_FORCE_A

Force Constant Alpha value

#define NEMA_BLOP_LUT

src_tex as index, src2_tex as palette

#define NEMA_BLOP_MODULATE_A

Modulate by Constant Alpha value

#define NEMA_BLOP_MODULATE_RGB

Modulate by Constant Color (RGB) values

#define NEMA_BLOP_NONE

No extra blending operation

#define NEMA_BLOP_NO_USE_ROPBL

Don't use Rop Blender even if present

#define NEMA_BLOP_RECOLOR

Cconst*Aconst + Csrc*(1-Aconst). Overrides MODULATE_RGB. On NemaP GPU, recolor is available only when HW Rop Blender is enabled

#define NEMA_BLOP_SRC_CKEY

Apply Source Color Keying - draw only when src color doesn't match colorkey

#define NEMA_BLOP_SRC_PREMULT

Premultiply Source Color with Source Alpha (cannot be used with **NEMA_BLOP_MODULATE_RGB**)

#define NEMA_BLOP_STENCIL_TXTY

Use TEX3 as mask/stencil

#define NEMA_BLOP_STENCIL_XY

Use TEX3 as mask/stencil

#define NEMA_BL_ADD

Sa + Da

#define NEMA_BL_CLEAR

0

#define NEMA_BL_DST_ATOP

Sa * (1 - Da) + Da * Sa

#define NEMA_BL_DST_IN

Da * Sa

#define NEMA_BL_DST_OUT

Da * (1 - Sa)

#define NEMA_BL_DST_OVER

Sa * (1 - Da) + Da

#define NEMA_BL_SIMPLE

Sa * Sa + Da * (1 - Sa)

#define NEMA_BL_SRC

Sa

#define NEMA_BL_SRC_ATOP

Sa * Da + Da * (1 - Sa)

#define NEMA_BL_SRC_IN

Sa * Da

#define NEMA_BL_SRC_OUT

Sa * (1 - Da)

#define NEMA_BL_SRC_OVER

Sa + Da * (1 - Sa)

#define NEMA_BL_XOR

Sa * (1 - Da) + Da * (1 - Sa)

11.1.1.3 Function Documentation

uint32_t nema_blending_mode (uint32_t src_bf, uint32_t dst_bf, uint32_t blops)

Return blending mode given source and destination blending factors and additional blending operations.

Parameters	Description
src	Source Blending Factor
dst	Destination Blending Factor
ops	Additional Blending Operations

Return

Final Blending Mode

`void nema_debug_overdraws ( uint32_t enable )`

Enable/disable ovedraw debugging. Disables gradient and texture, forces blending mode to **NEMA_BL_ADD**.

Parameters	Description
enable	Enables overdraw debugging if non-zero

`void nema_set_blend ( uint32_t blending_mode, nema_tex_t dst_tex, nema_tex_t fg_tex, nema_tex_t bg_tex )`

Set blending mode.

Parameters	Description
blending_mode	Blending mode to be set
dst_tex	Destination Texture
fg_tex	Foreground (source) Texture
bg_tex	Background (source2) Texture

`void nema_set_blend_blit ( uint32_t blending_mode )`

Set blending mode for blitting.

Parameters	Description
blending_mode	Blending mode to be set

`void nema_set_blend_blit_compose ( uint32_t blending_mode )`

Set blending mode for blitting with composing.

Parameters	Description
blending_mode	Blending mode to be set

`void nema_set_blend_fill ( uint32_t blending_mode )`

Set blending mode for filling.

Parameters	Description
blending_mode	Blending mode to be set

```
void nema_set_blend_fill_compose (  uint32_t blending_mode )
```

Set blending mode for filling with composing.

Parameters	Description
blending_mode	Blending mode to be set

```
void nema_set_const_color (  uint32_t rgba )
```

Set constant color.

Parameters	Description
rgba	RGBA color

Also see **nema_rgba()**.

```
void nema_set_dst_color_key (  uint32_t rgba )
```

Set destination color key.

Parameters	Description
rgba	RGBA color key

Also see **nema_rgba()**.

```
void nema_set_recolor_color (  uint32_t rgba )
```

Set recolor color. Overrides constant color.

Parameters	Description
rgba	RGBA color

Also see **nema_rgba(), nema_set_const_color()**.

```
void nema_set_src_color_key (  uint32_t rgba )
```

Set source color key.

Parameters	Description
rgba	RGBA color key

Also see **nema_rgba()**.

11.1.2 nema_cmdlist.h

```
#include "nema_sys_defs.h"
#include "nema_hal.h"
```

Data Structures

```
struct
nema_cmdlist_t
```

More...

Macros

```
#define CL_NOP 0x010000U
#define CL_PUSH 0x020000U
#define CL_RETURN 0x040000U
#define CL_ABORT 0x080000U
#define CL_BATCH_SHIFT 12
#define CL_BATCH_LOOP 0x8000
#define SUBMISSION_ID_MASK 0xffffff
#define CL_ALIGNMENT_MASK (0x00000007U)
```

11.1.2.1 Functions

```
nema_cmdlist_t nema_cl_create_prealloc(nema_buffer_t *bo)
```

Create a new Command List into a preallocated space.

```
nema_cmdlist_t nema_cl_create_sized(int size_bytes)
```

Create a new, non expandable Command List of specific size.

```
nema_cmdlist_t nema_cl_create(void)
```

Create a new expandable Command List.

```
void nema_cl_destroy(nema_cmdlist_t *cl)
```

Destroy/Free a Command List.

```
void nema_cl_rewind(nema_cmdlist_t *cl)
```

Reset position of next command to be written to the beginning. Does not clear the List's contents.

```
void nema_cl_bind(nema_cmdlist_t *cl)
```

Define in which Command List each subsequent commands are going to be inserted.

```
void nema_cl_bind_circular(nema_cmdlist_t *cl)
```

Define in which Command List each subsequent commands are going to be inserted. Bind this command list as Circular. It never gets full, it never expands, it may get implicitly submitted, it cannot be reused. No other CL should be submitted while a circular CL is bound.

```
void nema_cl_bind_sectored_circular(nema_cmdlist_t *cl, int
sectors)
```

Define in which Command List each subsequent commands are going to be inserted. Bind this command list as Circular which consists of multiple sectors. Minimum number of sectors is 2. Input sector number less than 2 automatically defaults to 2 sectors. Each sector is a sub-CL and must be at least 512 bytes in size. The CL never gets full, it never expands, it may get implicitly submitted, it cannot be reused. No other CL should be submitted while a sectored circular CL is bound.

```
void nema_cl_unbind(void)
```

Unbind current bound Command List, if any.

```
nema_cmdlist_t* nema_cl_get_bound(void)
```

Get bound Command List.

```
void nema_cl_submit(nema_cmdlist_t *cl)
```

Enqueue Command List to the Ring Buffer for execution.

```
int nema_cl_wait(nema_cmdlist_t *cl)
```

Wait for Command List to finish.

```
void nema_cl_add_cmd(uint32_t reg, uint32_t data)
```

Add a command to the bound Command List.

```
int nema_cl_add_multiple_cmds(int cmd_no, uint32_t *cmd)
```

Add multiple commands to the bound Command List.

```
void nema_cl_branch(nema_cmdlist_t *cl)
```

Branch from the bound Command List to a different one. Return is implied.

```
void nema_cl_jump(nema_cmdlist_t *cl)
```

Jump from the bound Command List to a different one. No return is implied.

`void nema_cl_return(void)`

Add an explicit return command to the bound Command List.

`int nema_cl_almost_full(nema_cmdlist_t *cl)`

Returns positive number if the Command List is almost full, otherwise returns 0.

`int nema_cl_enough_space(int cmd_no)`

Check if there is enough space or expansion can be performed for required commands.

11.1.2.2 Detailed Description

Macro Definition Documentation

11.1.2.3 Function Documentation

`void nema_cl_add_cmd ( uint32_t reg, uint32_t data )`

Add a command to the bound Command List.

Parameters	Description
reg	Hardware register to be written
data	Data to be written

`int nema_cl_add_multiple_cmds ( int cmd_no, uint32_t * cmd )`

Add multiple commands to the bound Command List.

Parameters	Description
cmd_no	Numbers of commands to add
cmd	Pointer to the commands to be added

Return

0 if no error has occurred.

`int nema_cl_almost_full ( nema_cmdlist_t * cl )`

Returns positive number if the Command List is almost full, otherwise returns 0.

Parameters	Description
cl	Pointer to the Command List

```
void nema_cl_bind (  nema_cmdlist_t * cl )
```

Define in which Command List each subsequent commands are going to be inserted.

Parameters	Description
cl	Pointer to the Command List

```
void nema_cl_bind_circular (  nema_cmdlist_t * cl )
```

Define in which Command List each subsequent commands are going to be inserted. Bind this command list as Circular. It never gets full, it never expands, it may get implicitly submitted, it cannot be reused. No other CL should be submitted while a circular CL is bound.

Parameters	Description
cl	Pointer to the Command List

```
void nema_cl_bind_sectored_circular (  nema_cmdlist_t * cl,  int
sectors )
```

Define in which Command List each subsequent commands are going to be inserted. Bind this command list as Circular which consists of multiple sectors. Minimum number of sectors is 2. Input sector number less than 2 automatically defaults to 2 sectors. Each sector is a sub-CL and must be at least 512 bytes in size. The CL never gets full, it never expands, it may get implicitly submitted, it cannot be reused. No other CL should be submitted while a sectored circular CL is bound.

Parameters	Description
cl	Pointer to the Command List
sectors	The number of sectors that the Command List is consisted of

```
void nema_cl_branch (  nema_cmdlist_t * cl )
```

Branch from the bound Command List to a different one. Return is implied.

Parameters	Description
cl	Pointer to the Command List to branch to

```
nema_cmdlist_t nema_cl_create (  void )
```

Create a new expandable Command List.

Return

The instance of the new Command List

```
nema_cmdlist_t nema_cl_create_prealloc (  nema_buffer_t * bo )
```

Create a new Command List into a preallocated space.

Parameters	Description
addr_virt	Command List's address (preallocated)
size_bytes	Command List's size in bytes

Return

The instance of the new Command List

```
nema_cmdlist_t nema_cl_create_sized (  int size_bytes )
```

Create a new, non expandable Command List of specific size.

Parameters	Description
size_bytes	Command List's size in bytes

Return

The instance of the new Command List

```
void nema_cl_destroy (  nema_cmdlist_t * cl )
```

Destroy/Free a Command List.

Parameters	Description
cl	Pointer to the Command List

```
int nema_cl_enough_space (  int cmd_no )
```

Check if there is enough space or expansion can be performed for required commands.

Parameters	Description
cmd_no	Numbers of commands to be checked if they fit zero is commands fit or expansion xan be performed else return negative.

```
nema_cmdlist_t * nema_cl_get_bound (  void )
```

Get bound Command List.

Return

Pointer to the bound Command List

```
uint32_t * nema_cl_get_space (  int cmd_no ) private
void nema_cl_jump (  nema_cmdlist_t * cl )
```

Jump from the bound Command List to a different one. No return is implied.

Parameters	Description
cl	Pointer to the Command List to jump to

```
void nema_cl_rewind (  nema_cmdlist_t * cl )
```

Reset position of next command to be written to the beginning. Does not clear the List's contents.

Parameters	Description
cl	Pointer to the Command List

```
void nema_cl_submit (  nema_cmdlist_t * cl )
```

Enqueue Command List to the Ring Buffer for execution.

Parameters	Description
cl	Pointer to the Command List

```
int nema_cl_wait (  nema_cmdlist_t * cl )
```

Wait for Command List to finish.

Parameters	Description
cl	Pointer to the Command List

Return

0 if no error has occurred.

11.1.3 nema_easing.h

11.1.3.1 Functions

`float nema_ez_linear(float p)`

Linear easing, no acceleration.

`float nema_ez_quad_in(float p)`

Quadratic easing in, accelerate from zero.

`float nema_ez_quad_out(float p)`

Quadratic easing out, decelerate to zero velocity.

`float nema_ez_quad_in_out(float p)`

Quadratic easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_cub_in(float p)`

Cubic easing in, accelerate from zero.

`float nema_ez_cub_out(float p)`

Cubic easing out, decelerate to zero velocity.

`float nema_ez_cub_in_out(float p)`

Cubic easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_quar_in(float p)`

Quartic easing in, accelerate from zero.

`float nema_ez_quar_out(float p)`

Quartic easing out, decelerate to zero velocity.

`float nema_ez_quar_in_out(float p)`

Quartic easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_quin_in(float p)`

Quintic easing in, accelerate from zero.

`float nema_ez_quin_out(float p)`

Quintic easing out, decelerate to zero velocity.

`float nema_ez_quin_in_out(float p)`

Quintic easing in and out, accelerate to halfway, then decelerate.

Sinusoidal easing in, accelerate from zero.

`float nema_ez_sin_out(float p)`

Sinusoidal easing out, decelerate to zero velocity.

`float nema_ez_sin_in_out(float p)`

Sinusoidal easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_circ_in(float p)`

Circular easing in, accelerate from zero.

`float nema_ez_circ_out(float p)`

Circular easing out, decelerate to zero velocity.

`float nema_ez_circ_in_out(float p)`

Circular easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_exp_in(float p)`

Exponential easing in, accelerate from zero.

`float nema_ez_exp_out(float p)`

Exponential easing out, decelerate to zero velocity.

`float nema_ez_exp_in_out(float p)`

Exponential easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_elast_in(float p)`

Elastic easing in, accelerate from zero.

`float nema_ez_elast_out(float p)`

Elastic easing out, decelerate to zero velocity.

`float nema_ez_elast_in_out(float p)`

Elastic easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_back_in(float p)`

Overshooting easing in, accelerate from zero.

`float nema_ez_back_out(float p)`

Overshooting easing out, decelerate to zero velocity.

`float nema_ez_back_in_out(float p)`

Overshooting easing in and out, accelerate to halfway, then decelerate.

`float nema_ez_bounce_out(float p)`

Bouncing easing in, accelerate from zero.

`float nema_ez_bounce_in(float p)`

Bouncing easing out, decelerate to zero velocity.

`float nema_ez_bounce_in_out(float p)`

Bouncing easing in and out, accelerate to halfway, then decelerate.

`float nema_ez(float A, float B, float steps, float cur_step, float(*ez_func)(float p))`

Convenience function to perform easing between two values given number of steps, current step and easing function.

Detailed Description

11.1.3.2 Function Documentation

`float nema_ez ( float A, float B, float steps, float cur_step, float(*)(float p) ez_func )`

Convenience function to perform easing between two values given number of steps, current step and easing function.

Parameters	Description
A	Initial value within range [0, 1]
B	Finale value within range [0, 1]
steps	Total number of steps
cur_step	Current Step
ez_func	pointer to the desired easing function

Return

Eased value

`float nema_ez_back_in ( float p )`

Overshooting easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

`float nema_ez_back_in_out ( float p )`

Overshooting easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_back_out (  float p )
```

Overshooting easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_bounce_in (  float p )
```

Bouncing easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_bounce_in_out (  float p )
```

Bouncing easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_bounce_out (  float p ) Bouncing easing in, accelerate
from zero.
```

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_circ_in (  float p )
```

Circular easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_circ_in_out (  float p )
```

Circular easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_circ_out (  float p )
```

Circular easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_cub_in (  float p )
```

Cubic easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_cub_in_out (  float p )
```

Cubic easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_cub_out (  float p )
```

Cubic easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_elast_in (  float p )
```

Elastic easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_elast_in_out (  float p )
```

Elastic easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_elast_out (  float p )
```

Elastic easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_exp_in (  float p )
```

Exponential easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_exp_in_out (  float p )
```

Exponential easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_exp_out (  float p )
```

Exponential easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_linear (  float p )
```

Linear easing, no acceleration.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quad_in (  float p )
```

Quadratic easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quad_in_out (  float p )
```

Quadratic easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quad_out (  float p )
```

Quadratic easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quar_in (  float p )
```

Quartic easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quar_in_out (  float p )
```

Quartic easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quar_out (  float p )
```

Quartic easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_quin_in (  float p )
```

Quintic easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

`float nema_ez_quin_in_out ( float p )`

Quintic easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

`float nema_ez_quin_out ( float p )`

Quintic easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

`float nema_ez_sin_in ( float p )`

Sinusoidal easing in, accelerate from zero.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

`float nema_ez_sin_in_out ( float p )`

Sinusoidal easing in and out, accelerate to halfway, then decelerate.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

```
float nema_ez_sin_out (  float p )
```

Sinusoidal easing out, decelerate to zero velocity.

Parameters	Description
p	Input value, typically within the [0, 1] range

Return

Eased value

11.1.4 nema_error.h

```
#include "nema_sys_defs.h"
```

Macros

```
#define NEMA_ERR_NO_ERROR (0x00000000U)
#define NEMA_ERR_SYS_INIT_FAILURE (0x00000001U)
#define NEMA_ERR_GPU_ABSENT (0x00000002U)
#define NEMA_ERR_RB_INIT_FAILURE (0x00000004U)
#define NEMA_ERR_NON_EXPANDABLE_CL_FULL (0x00000008U)
#define NEMA_ERR_CL_EXPANSION (0x00000010U)
#define NEMA_ERR_OUT_OF_GFX_MEMORY (0x00000020U)
#define NEMA_ERR_OUT_OF_HOST_MEMORY (0x00000040U)
#define NEMA_ERR_NO_BOUND_CL (0x00000080U)
#define NEMA_ERR_NO_BOUND_FONT (0x00000100U)
#define NEMA_ERR_GFX_MEMORY_INIT (0x00000200U)
#define NEMA_ERR_DRIVER_FAILURE (0x00000400U)
#define NEMA_ERR_MUTEX_INIT (0x00000800U)
#define NEMA_ERR_INVALID_BO (0x00001000U)
#define NEMA_ERR_INVALID_CL (0x00002000U)
#define NEMA_ERR_INVALID_CL_ALIGMENT (0x00004000U)
```

```
#define NEMA_ERR_NO_INIT (0x00008000U)
#define NEMA_ERR_INVALID_SECTORED_CL_SIZE (0x00010000U)
```

11.1.4.1 Functions

```
uint32_t nema_get_error(void)
```

Return

Error Id.

11.1.4.2 Detailed Description

Macro Definition Documentation

```
#define NEMA_ERR_CL_EXPANSION
```

Command list expansion error

```
#define NEMA_ERR_DRIVER_FAILURE
```

Nema GPU Kernel Driver failure

```
#define NEMA_ERR_GFX_MEMORY_INIT
```

Graphics memory initialization failure

```
#define NEMA_ERR_GPU_ABSENT
```

Nema GPU is absent

```
#define NEMA_ERR_INVALID_BO
```

Invalid buffer provided

```
#define NEMA_ERR_INVALID_CL
```

Invalid CL provided

```
#define NEMA_ERR_INVALID_CL_ALIGMENT
```

Invalid CL buffer alignment

```
#define NEMA_ERR_INVALID_SECTORED_CL_SIZE
```

Invalid sectored CL size, each sector should be at least 512 bytes

```
#define NEMA_ERR_MUTEX_INIT
```

Mutex initialization failure

```
#define NEMA_ERR_NON_EXPANDABLE_CL_FULL
```

Non expandable command list is full

```
#define NEMA_ERR_NO_BOUND_CL
```

There is no bound command list

#define NEMA_ERR_NO_BOUND_FONT

There is no bound font

#define NEMA_ERR_NO_ERROR

No error has occurred

#define NEMA_ERR_NO_INIT

GFX uninitialised

#define NEMA_ERR_OUT_OF_GFX_MEMORY

Graphics memory is full

#define NEMA_ERR_OUT_OF_HOST_MEMORY

Host memory is full

#define NEMA_ERR_RB_INIT_FAILURE

Ring buffer initialization failure

#define NEMA_ERR_SYS_INIT_FAILURE

System initialization failure

11.1.4.3 Function Documentation

uint32_t nema_get_error (void) Return Error Id.

Return

0 if no error exists

11.1.5 nema_font.h

#include "nema_hal.h"

Data Structures

struct nema_kern_pair_t

More...

struct nema_glyph_t

More...

struct nema_glyph_indexed_t

More...

struct nema_font_range_t

More...

```
struct nema_font_t
```

More...

Macros

```
#define NEMA_ALIGNX_LEFT (0x00U)
#define NEMA_ALIGNX_RIGHT (0x01U)
#define NEMA_ALIGNX_CENTER (0x02U)
#define NEMA_ALIGNX_JUSTIFY (0x03U)
#define NEMA_ALIGNX_MASK (0x03U)
#define NEMA_ALIGNY_TOP (0x00U)
#define NEMA_ALIGNY_BOTTOM (0x04U)
#define NEMA_ALIGNY_CENTER (0x08U)
#define NEMA_ALIGNY_JUSTIFY (0x0cU)
#define NEMA_ALIGNY_MASK (0x0cU)
#define NEMA_TEXT_WRAP (0x10U)
```

11.1.5.1 Functions

```
void nema_bind_font(nema_font_t *font)
```

Bind the font to use in future nema_print() calls.

```
int nema_string_get_bbox(const char *str, int *w, int *h, int
max_w, uint32_t wrap)
```

Get the bounding box's width and height of a string.

```
void nema_print(const char *str, int x, int y, int w, int h,
uint32_t fg_col, uint32_t align)
```

Print pre-formatted text.

```
void nema_print_to_position(const char *str, int *pos_x, int
*pos_y, int x, int y, int w, int h, uint32_t fg_col, uint32_t
align)
```

Print pre-formatted text.

```
void nema_print_indexed(const int *ids, int id_count, int x, int
y, uint32_t fg_col)
```

Print text (not formated) with indexed glyphs. Text is printed in a single line, from left to right.

```
void nema_print_char_indexed(const int id, int x, int y, uint32_t
fg_col)
```

Print a single character with indexed glyph.

```
void nema_string_indexed_get_bbox(const int *ids, int id_count,
int *w, int *h, int max_w)
```

Returns the bounding box's width and height of a string with indexed glyphs.

```
int nema_font_get_x_advance(void)
```

Returns the horizontal advance (in pixels) of the bound font.

11.1.5.2 Detailed Description

Macro Definition Documentation

```
#define NEMA_ALIGNX_CENTER
```

Align horizontally centered

```
#define NEMA_ALIGNX_JUSTIFY
```

Justify horizontally

```
#define NEMA_ALIGNX_LEFT
```

Align horizontally to the left

```
#define NEMA_ALIGNX_MASK
```

Horizontal alignment mask

```
#define NEMA_ALIGNX_RIGHT
```

Align horizontally to the right

```
#define NEMA_ALIGNY_BOTTOM
```

Align vertically to the bottom

```
#define NEMA_ALIGNY_CENTER
```

Align vertically centered

```
#define NEMA_ALIGNY_JUSTIFY
```

Justify vertically

```
#define NEMA_ALIGNY_MASK
```

Vertical alignment mask

```
#define NEMA_ALIGNY_TOP
```

Align vertically to the top

```
#define NEMA_TEXT_WRAP
```

Use text wrapping

11.1.5.3 Function Documentation

void nema_bind_font (nema_font_t * font)

Bind the font to use in future nema_print() calls.

Parameters	Description
font	Pointer to font

```
void nema_print ( const char * str, int x, int y, int w, int h,
uint32_t fg_col, uint32_t align )
```

Print pre-formatted text.

Parameters	Description
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	Width of the text area
h	Height of the text area
fg_col	Foreground color of text
align	Alignment and wrapping mode

```
void nema_print_char_indexed ( const int id, int x, int y,
uint32_t fg_col )
```

Print a single character with indexed glyph.

Parameters	Description
id	Array with the glyphs indices
x	X coordinate of the character's top-left corner
y	Y coordinate of the character's top-left corner
fg_col	Character's color

```
void nema_print_indexed ( const int * ids, int id_count, int x,
int y, uint32_t fg_col )
```

Print text (not formated) with indexed glyphs. Text is printed in a single line, from left to right.

Parameters	Description
ids	Array with the glyphs indices
id_count	Count of the characters to be drawn
x	X coordinate of the text-area's top-left corner
y	Y coordinate of the text-area's top-left corner
fg_col	Foreground color of text

```
void nema_print_to_position ( const char * str, int * pos_x, int * pos_y, int x, int y, int w, int h, uint32_t fg_col, uint32_t align )
```

Print pre-formatted text.

Parameters	Description
*str	Pointer to string
*cursor_x	X position of next character to be drawn. Usually initialized to 0 by the user and then updated internally by the library
*cursor_y	Y position of next character to be drawn. Usually initialized to 0 by the user and then updated internally by the library
x	X coordinate of text-area's top-left corner
y	Y coordinate of text-area's top-left corner
w	Width of the text area
h	Height of the text area
fg_col	Foreground color of text
align	Alignment and wrapping mode

```
int nema_string_get_bbox ( const char * str, int * w, int * h, int max_w, uint32_t wrap )
```

Get the bounding box's width and height of a string.

Parameters	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

Return

Number of carriage returns

```
void nema_string_indexed_get_bbox (  const int * ids,  int id_count,
int * w, int * h,  int max_w )
```

Returns the bounding box's width and height of a string with indexed glyphs.

The string must be specified as a single line text, due to the restriction that the characters are described by respective glyph indices. The height of the bounding box will be equal to the height of the bound font.

Parameters	Description
ids	Array with the glyphs indices
id_count	Count of the characters contained in the array with the glyphs indices
w	Pointer to variable where width should be written
h	Pointer to variable where height should be written
max_w	Maximum allowed width (if w is greater than this value, it will saturate to this)

11.1.6 nema_graphics.h

```
#include "nema_sys_defs.h"
#include "nema_hal.h" #include "nema_matrix3x3.h"
```

Data Structures

```
struct
img_obj_t
More...
```

Macros

```
#define NEMA_RGBX8888 0x00U
#define NEMA_RGBA8888 0x01U
#define NEMA_XRGB8888 0x02U
  NEMA_ARGB8888 0x03U
  NEMA_RGB565 0x04U NEMA_RGBA5650 0x04U
  NEMA_RGBA5551 0x05U
  NEMA_RGBA4444 0x06U
#define NEMA_RGBA0800 0x07U
#define NEMA_A8 0x08U
```

```
#define NEMA_RGBA0008 0x08U
#define NEMA_L8 0x09U
#define NEMA_L1 0x0BU
#define NEMA_BW1 0x0CU
#define NEMA_A1 0x0CU
#define NEMA_UYVY 0x0DU
#define NEMA_ABGR8888 0x0EU
#define NEMA_XBGR8888 0x0FU
#define NEMA_BGRA8888 0x10U
#define NEMA_BGRX8888 0x11U
#define NEMA_TSC4 0x12U
#define NEMA_BGRA5650 0x13U
#define NEMA_BGR565 0x13U
#define NEMA_TSC6 0x16U
#define NEMA_TSC6A 0x17U
#define NEMA_RV 0x18U
  NEMA_GU 0x19U
  NEMA_BY 0x1AU
  NEMA_YUV 0x1BU
  NEMA_Z24_8 0x1cU
  NEMA_Z16 0x1dU
#define NEMA_UV 0x1eU
#define NEMA_A1LE 0x27U
#define NEMA_A2LE 0x28U
#define NEMA_A4LE 0x29U
#define NEMA_L1LE 0x2AU
#define NEMA_L2LE 0x2BU
#define NEMA_L4LE 0x2CU
#define NEMA_A2 0x30U
#define NEMA_L2 0x31U
#define NEMA_A4 0x34U
#define NEMA_L4 0x35U
#define NEMA_RGBA3320 0x38U
#define NEMA_RGB332 0x38U
#define NEMA_BGR24 0x39U
#define NEMA_RGB24 0x3CU
```

```
#define NEMA_RV10 0x3DU
#define NEMA_GU10 0x3EU
#define NEMA_BY10 0x3FU
  NEMA_RGBA2222 0x40U
  NEMA_ABGR2222 0x41U
  NEMA_BGRA2222 0x42U
  NEMA_ARGB2222 0x43U
  NEMA_AL88 0x44U
#define NEMA_AL44 0x45U
#define NEMA_ARGB1555 0x46U
#define NEMA_ARGB4444 0x47U
#define NEMA_BGRA5551 0x48U
#define NEMA_ABGR1555 0x49U
#define NEMA_BGRA4444 0x4aU
#define NEMA_ABGR4444 0x4bU
#define NEMA_TSC12 0x4cU
#define NEMA_TSC12A 0x4dU
#define NEMA_TSC6AP 0x4eU
#define NEMA_DITHER 0x80U
#define NEMA_FORMAT_MASK 0x7FU
#define NEMA_FILTER_PS 0x00U
#define NEMA_FILTER_BL 0x01U
#define NEMA_TEX_CLAMP (0x00U)
#define NEMA_TEX_REPEAT (0x01U<<2)
#define NEMA_TEX_BORDER (0x02U<<2)
#define NEMA_TEX_MIRROR (0x03U<<2)
#define NEMA_TEX_MORTON_ORDER (0x10U)
  NEMA_TEX_RANGE_0_1 (0x1U<<5)
  NEMA_TEX_LEFT_HANDED (0x1U<<6)
  NEMA_ROT_000_CCW (0x0U)
  NEMA_ROT_090_CCW (0x1U)
#define NEMA_ROT_180_CCW (0x2U)
#define NEMA_ROT_270_CCW (0x3U)
#define NEMA_ROT_000_CW (0x0U)
#define NEMA_ROT_270_CW (0x1U)
#define NEMA_ROT_180_CW (0x2U)
```

```
#define NEMA_ROT_090_CW (0x3U)

#define NEMA_MIR_VERT (0x4U)

#define NEMA_MIR_HOR (0x8U)
```

Typedefs

```
typedef img_obj_t  nema_img_obj_t typedef img_obj_t
nema_img_obj_t typedef uint32_t  nema_tex_format_t typedef
uint32_t nema_tex_format_t typedef uint8_t  nema_tex_mode_t
typedef uint8_t nema_tex_mode_t
```

Enumerations enum

```
nema_tex_t

NEMA_NOTEX= -1, NEMA_TEX0= 0, NEMA_TEX1= 1, NEMA_TEX2= 2,
NEMA_TEX3= 3,

NEMA_TEXMAX= 4

enum nema_tri_cull_t

NEMA_CULL_NONE= 0, NEMA_CULL_CW= (1U<<28), NEMA_CULL_CCW=
(1U<<29), NEMA_CULL_ALL= NEMA_CULL_CW | NEMA_CULL_CCW
```

11.1.6.1 Functions

```
int nema_checkGPUPresence(void)
```

Check if a known GPU is present.

```
void nema_bind_tex(nema_tex_t texid, uintptr_t addr_gpu, uint32_t
width, uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t wrap_mode)
```

Program a Texture Unit.

```
void nema_set_tex_color(uint32_t color)
```

Set Texture Mapping default color.

```
void nema_set_const_reg(int reg, uint32_t value)
```

Write a value to a Constant Register of the GPU.

```
void nema_set_clip(int32_t x, int32_t y, uint32_t w, uint32_t h)
```

Sets the drawing area's Clipping Rectangle.

```
uint32_t nema_enable_aa(uint8_t e0, uint8_t e1, uint8_t e2,
uint8_t e3)
```

Enables MSAA per edge.

```
uint32_t nema_enable_aa_flags(uint32_t aa)
```

Enables MSAA per edge.

void nema_get_dirty_region(int *minx, int *miny, int *maxx, int *maxy)

Returns the bounding rectangle of all the pixels that have been modified since its previous call. Available only on Nema|P and Nema|PVG GPUs.

void nema_clear_dirty_region(void)

Clear dirty region information - runs via the bound command-list Available only on Nema|P and Nema|PVG GPUs.

void nema_clear_dirty_region_imm(void)

Clear dirty region information immediately, no command-list involved Available only on Nema|P and Nema|PVG GPUs.

void nema_tri_cull(nema_tri_cull_t cull)

Set triangle/quadrilateral culling mode.

int nema_format_size(nema_tex_format_t format)

Return pixel size in bytes.

int nema_stride_size(nema_tex_format_t format, nema_tex_mode_t wrap_mode, int width)

Return stride in bytes.

int nema_texture_size(nema_tex_format_t format, nema_tex_mode_t wrap_mode, int width, int height)

Return texture size in bytes.

uint32_t nema_rgba(unsigned char R, unsigned char G, unsigned char B, unsigned char A)

Return Nema internal RGBA color.

uint32_t nema_premultiply_rgba(uint32_t rgba)

Premultiply RGB channels with Alpha channel.

int nema_init(void)

Initialize NemaGFX library.

int nema_reinit(void)

Reinitialize NemaGFX library.

void nema_bind_src_tex(uintptr_t baseaddr_phys, uint32_t width, uint32_t height, nema_tex_format_t format, int32_t stride, nema_tex_mode_t mode)

Program Texture Unit with a foreground (source) texture (NEMA_TEX1)

```
void nema_bind_src2_tex(uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode)
```

Program Texture Unit with a background texture ((NEMA_TEX2)

```
void nema_bind_dst_tex(uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride)
```

Program Texture Unit with a destination texture (NEMA_TEX0)

```
void nema_bind_lut_tex(uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode, uintptr_t palette_baseaddr_phys,
nema_tex_format_t palette_format)
```

Program Texture Unit with a lut/palette texture (NEMA_TEX2) and index texture (NEMA_TEX1_)

```
void nema_bind_depth_buffer(uintptr_t baseaddr_phys, uint32_t
width, uint32_t height)
```

Bind Depth Buffer.

```
void nema_clear(uint32_t rgba8888)
```

Clear destination texture with color.

```
void nema_clear_depth(uint32_t val)
```

Clear depth buffer with specified value.

```
void nema_draw_line(int x0, int y0, int x1, int y1, uint32_t
rgba8888)
```

Draw a colored line.

```
void nema_draw_line_aa(float x0, float y0, float x1, float y1,
float w, uint32_t rgba8888)
```

Draw a line with width. Apply AA if available.

```
void nema_draw_circle(int x, int y, int r, uint32_t rgba8888)
```

Draw a colored circle with 1 pixel width.

```
void nema_draw_circle_aa(float x, float y, float r, float w,
uint32_t rgba8888)
```

Draw a colored circle with Anti-Aliasing (if available) and specified width.

```
void nema_draw_rounded_rect(int x0, int y0, int w, int h, int r,
uint32_t rgba8888)
```

Draw a colored rectangle with rounded edges.

```
void nema_draw_rect(int x, int y, int w, int h, uint32_t rgba8888)
```

Draw a colored rectangle.

void nema_fill_circle(int x, int y, int r, uint32_t rgba8888)

Fill a circle with color.

void nema_fill_circle_aa(float x, float y, float r, uint32_t rgba8888)

Fill a circle with color, use Anti-Aliasing if available.

void nema_fill_triangle(int x0, int y0, int x1, int y1, int x2, int y2, uint32_t rgba8888)

Fill a triangle with color.

void nema_fill_rounded_rect(int x0, int y0, int w, int h, int r, uint32_t rgba8888)

Fill a rectangle with rounded edges with color.

void nema_fill_rect(int x, int y, int w, int h, uint32_t rgba8888)

Fill a rectangle with color.

void nema_fill_quad(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3, uint32_t rgba8888)

Fill a quadrilateral with color.

void nema_fill_rect_f(float x, float y, float w, float h, uint32_t rgba8888)

Fill a rectangle with color (float coordinates)

void nema_fill_quad_f(float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, uint32_t rgba8888)

Fill a quadrilateral with color (float coordinates)

void nema_fill_triangle_f(float x0, float y0, float x1, float y1, float x2, float y2, uint32_t rgba8888)

Fill a triangle with color (float coordinates)

void nema_blit(int x, int y)

Blit source texture to destination texture.

void nema_blit_rounded(int x, int y, int r)

Blit source texture to destination texture with rounded corners.

void nema_blit_rect(int x, int y, int w, int h)

Blit source texture to destination's specified rectangle (crop or wrap when needed)

void nema_blit_subrect(int dst_x, int dst_y, int w, int h, int src_x, int src_y)

Blit part of a source texture to destination's specified rectangle (crop or wrap when needed)

```
void nema_blit_rect_fit(int x, int y, int w, int h)
```

Blit source texture to destination. Fit (scale) texture to specified rectangle.

```
void nema_blit_subrect_fit(int dst_x, int dst_y, int dst_w, int
dst_h, int src_x, int src_y, int src_w, int src_h)
```

Blit part of source texture to destination. Fit (scale) texture to specified rectangle.

```
void nema_blit_rotate_pivot(float cx, float cy, float px, float
py, float degrees_cw)
```

Rotate around pivot point and Blit source texture.

```
void nema_blit_rotate(int x, int y, uint32_t rotation)
```

Rotate and Blit source texture to destination.

```
void nema_blit_rotate_partial(int sx, int sy, int sw, int sh, int
x, int y, uint32_t rotation)
```

Rotate and Blit partial source texture to destination.

```
void nema_blit_tri_fit(float dx0, float dy0, int v0, float dx1,
float dy1, int v1, float dx2, float dy2, int v2)
```

Blit source texture to destination. Fit texture to specified triangle.

```
void nema_blit_tri_uv(float dx0, float dy0, float dw0, float dx1,
float dy1, float dw1, float dx2, float dy2, float dw2, float sx0,
float sy0, float sx1, float sy1, float sx2, float sy2)
```

Blit a triangular part of the source tecture to a triangular destination area.

```
void nema_blit_quad_fit(float dx0, float dy0, float dx1, float
dy1, float dx2, float dy2, float dx3, float dy3)
```

Blit source texture to destination. Fit texture to specified quadrilateral.

```
void nema_blit_subrect_quad_fit(float dx0, float dy0, float dx1,
float dy1, float dx2, float dy2, float dx3, float dy3, int sx, int
sy, int sw, int sh)
```

Blit source texture to destination. Fit rectangulare area of texture to specified quadrilateral.

```
void nema_blit_quad_m(float dx0, float dy0, float dx1, float dy1,
float dx2, float dy2, float dx3, float dy3, nema_matrix3x3_t m)
```

Blit source texture to destination. Use the matrix provided by the user.

```
void nema_brk_enable(void)
```

Enable breakpoints.

void nema_brk_disable(void)

Disable breakpoints.

int nema_brk_add(void)

Add a breakpoint to the current Command List.

int nema_brk_wait(int brk_id)

Add a breakpoint to the current Command List.

void nema_brk_continue(void)

Instruct the GPU to resume execution.

void nema_ext_hold_enable(uint32_t hold_id)

Enable external hold signals.

void nema_ext_hold_disable(uint32_t hold_id)

Disable external hold signals.

void nema_ext_hold_irq_enable(uint32_t hold_id)

Enable Interrupt Request when GPU reaches hold point.

void nema_ext_hold_irq_disable(uint32_t hold_id)

Disable external hold signals.

void nema_ext_hold_assert(uint32_t hold_id, int stop)

Assert hold signals internally via a Command List.

void nema_ext_hold_deassert(uint32_t hold_id)

Dessert hold signals internally via a Command List.

void nema_ext_hold_assert_imm(uint32_t hold_id)

Assert hold signals from the CPU (no Command List)

void nema_ext_hold_deassert_imm(uint32_t hold_id)

Dessert hold signals from the CPU (no Command List)

const char* nema_get_sw_device_name(void)

Check for which architecture is the library compiled.

11.1.6.2 Detailed Description

Macro Definition Documentation

#define NEMA_A1

A1 (source only)

#define NEMA_A1LE

A1LE (source only)

#define NEMA_A2

A2 (source only)

#define NEMA_A2LE

A2LE (source only)

#define NEMA_A4

A4 (source only)

#define NEMA_A4LE

A4LE (source only)

#define NEMA_A8

RGBA0008

#define NEMA_ABGR1555

ABGR1555 (Available if HW enabled - check HW manual)

#define NEMA_ABGR2222

ABGR2222 (Available if HW enabled - check HW manual)

#define NEMA_ABGR4444

ABGR4444 (Available if HW enabled - check HW manual)

#define NEMA_ABGR8888

ABGR8888

#define NEMA_AL44

AL44 (Available if HW enabled - check HW manual)

#define NEMA_AL88

AL88 (Available if HW enabled - check HW manual)

#define NEMA_ARGB1555

ARGB1555 (Available if HW enabled - check HW manual)

#define NEMA_ARGB2222

ARGB2222 (Available if HW enabled - check HW manual)

#define NEMA_ARGB4444

ARGB4444 (Available if HW enabled - check HW manual)

#define NEMA_ARGB8888

ARGB8888

#define NEMA_BGR24

BGR24

#define NEMA_BGR565

BGRA5650 (Available if HW enabled - check HW manual)

#define NEMA_BGRA2222

BGRA2222 (Available if HW enabled - check HW manual)

#define NEMA_BGRA4444

BGRA4444 (Available if HW enabled - check HW manual)

#define NEMA_BGRA5551

BGRA5551 (Available if HW enabled - check HW manual)

#define NEMA_BGRA5650

BGRA5650 (Available if HW enabled - check HW manual)

#define NEMA_BGRA8888

BGRA

#define NEMA_BGRX8888

BGRX

#define NEMA_BW1

A1 (source only)

#define NEMA_BY

BY

#define NEMA_BY10

BY-10bit (Available if HW enabled - check HW manual

#define NEMA_DITHER

Nema Dithering

#define NEMA_FILTER_BL

Bilinear filtering.

#define NEMA_FILTER_PS

Point Sampling.

#define NEMA_FORMAT_MASK

Format Mask

#define NEMA_GU

GU

#define NEMA_GU10

GU-10bit (Available if HW enabled - check HW manual

#define NEMA_L1

L1 (source only)

#define NEMA_L1LE

L1LE (source only)

#define NEMA_L2

L2 (source only)

#define NEMA_L2LE

L2LE (source only)

#define NEMA_L4

L4 (source only)

#define NEMA_L4LE

L4LE (source only)

#define NEMA_L8

L8

#define NEMA_MIR_HOR

Mirror Horizontally

#define NEMA_MIR_VERT

Mirror Vertically

#define NEMA_RGB24

RGB24

#define NEMA_RGB332

RGBA3320

#define NEMA_RGB565

RGBA5650

#define NEMA_RGBA0008

RGBA0008

#define NEMA_RGBA0800

RGBA0800 (Available if HW enabled - check HW manual)

#define NEMA_RGBA2222

RGBA2222 (Available if HW enabled - check HW manual)

#define NEMA_RGBA3320

RGBA3320

#define NEMA_RGBA4444

RGBA4444

#define NEMA_RGBA5551

RGBA5551

#define NEMA_RGBA5650

RGBA5650

#define NEMA_RGBA8888

RGBA8888

#define NEMA_RGBX8888

RGBX8888

#define NEMA_ROT_000_CCW

No rotation

#define NEMA_ROT_000_CW

No rotation

#define NEMA_ROT_090_CCW

Rotate 90 degrees counter-clockwise

#define NEMA_ROT_090_CW

Rotate 90 degrees clockwise

#define NEMA_ROT_180_CCW

Rotate 180 degrees counter-clockwise

#define NEMA_ROT_180_CW

Rotate 180 degrees clockwise

#define NEMA_ROT_270_CCW

Rotate 270 degrees counter-clockwise

#define NEMA_ROT_270_CW

Rotate 270 degrees clockwise

#define NEMA_RV

RV

#define NEMA_RV10

RV-10bit (Available if HW enabled - check HW manual #define NEMA_TEX_BORDER Border

#define NEMA_TEX_CLAMP

Clamp

#define NEMA_TEX_LEFT_HANDED

(0,0) is bottom left corner

#define NEMA_TEX_MIRROR

Mirror

#define NEMA_TEX_RANGE_0_1

Interpolated Coordinates range: 0-1

#define NEMA_TEX_REPEAT

Repeat

#define NEMA_TSC12

TSC12 (Available if HW enabled - check HW manual)

#define NEMA_TSC12A

TSC12A (Available if HW enabled - check HW manual)

#define NEMA_TSC4

TSC4

#define NEMA_TSC6

TSC6

#define NEMA_TSC6A

TSC6A

#define NEMA_TSC6AP

TSC6AP (Available if HW enabled - check HW manual)

#define NEMA_UV

UV, 32-bit format that consists of two 16-bit values in 12.4 fixed point

#define NEMA_UYVY

UYVY

#define NEMA_XBGR8888

XBGR8888

#define NEMA_XRGB8888

XRGB8888

#define NEMA_YUV

YUV

#define NEMA_Z16

Z16

#define NEMA_Z24_8

Z24_8

Typed of Documentation

Enumeration Type Documentation

11.1.6.3 Function Documentation

void nema_bind_depth_buffer (uintptr_t baseaddr_phys, uint32_t width, uint32_t height)

Bind Depth Buffer.

Parameters	Description
baseaddr_phys	Address of the depth buffer, as seen by the GPU
width	Buffer width
height	Buffer hight

void nema_bind_dst_tex (uintptr_t baseaddr_phys, uint32_t width, uint32_t height, nema_tex_format_t format, int32_t stride)

Program Texture Unit with a destination texture (NEMA_TEX0).

Parameters	Description
baseaddr_phys	Address of the destination texture, as seen by the GPU
width	Texture width
height	Texture hight
format	Texture format
stride	Texture stride. If negative, it's calculated internally.

```
void nema_bind_lut_tex ( uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode, uintptr_t palette_baseaddr_phys,
nema_tex_format_t palette_format )
```

Program Texture Unit with a lut/palette texture (NEMA_TEX2) and index texture (NEMA_TEX1_)

Parameters	Description
baseaddr_phys	Address of the index texture
width	Index texture width
height	Index texture hight
format	Index texture format
stride	Index texture stride. If negative, it's calculated internally.
mode	Index texture sampling mode. When using 'NEMA_TEX_REPEAT' or 'NEMA_TEX_MIRROR' wrapping mode, texture dimensions must be a power of two, otherwise the behavior is undefined. NEMA_FILTER_BL is not supported, texture filtering is always performed using point sampling.
palette_baseaddr_phys	Address of the lut/palette texture
palette_format	lut/palette texture format

```
void nema_bind_src2_tex ( uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode )
```

Program Texture Unit with a background texture ((NEMA_TEX2)

Parameters	Description
baseaddr_phys	Address of the source2 texture, as seen by the GPU
width	Texture width
height	Texture hight
format	Texture format
stride	Texture stride. If negative, it's calculated internally.
wrap_mode	Wrap/Repeat mode to be used. When using 'repeat' or 'mirror', texture dimensions must be a power of two. Otherwise the behavior is undefined.

```
void nema_bind_src_tex ( uintptr_t baseaddr_phys, uint32_t width,
uint32_t height, nema_tex_format_t format, int32_t stride,
nema_tex_mode_t mode )
```

Program Texture Unit with a foreground (source) texture (NEMA_TEX1)

Parameters	Description
baseaddr_phys	Address of the source texture, as seen by the GPU
width	Texture width
height	Texture hight
format	Texture format
stride	Texture stride. If negative, it's calculated internally.
wrap_mode	Wrap/Repeat mode to be used. When using 'repeat' or 'mirror', texture dimensions must be a power of two. Otherwise the behavior is undefined.

```
void nema_bind_tex ( nema_tex_t texid, uintptr_t addr_gpu,
uint32_t width, uint32_t height, nema_tex_format_t format, int32_t
stride, nema_tex_mode_t wrap_mode )
```

Program a Texture Unit.

Parameters	Description
texid	Texture unit to be programmed
addr_gpu	Texture's address as seen by the GPU
width	Texture's width
height	Texture's height
format	Texture's format
stride	Texture's stride. If stride < 0, it's left to be calculated
wrap_mode	Wrap/Repeat mode to be used. When using 'repeat' or 'mirror', texture dimensions must be a power of two. Otherwise the behavior is undefined

```
void nema_blit ( int x, int y )
```

Blit source texture to destination texture.

Parameters	Description
x	destination x coordinate
y	destination y coordinate

See also **nema_set_blend_fill()**.

```
void nema_blit_quad_fit ( float dx0, float dy0, float dx1, float
dy1, float dx2, float dy2, float dx3, float dy3 )
```

Blit source texture to destination. Fit texture to specified quadrilateral.

Parameters	Description
dx0	x coordinate at the first vertex of the quadrilateral
dy0	y coordinate at the first vertex of the quadrilateral
dx1	x coordinate at the second vertex of the quadrilateral
dy1	y coordinate at the second vertex of the quadrilateral
dx2	x coordinate at the third vertex of the quadrilateral
dy2	y coordinate at the third vertex of the quadrilateral
dx3	x coordinate at the fourth vertex of the quadrilateral
dy3	y coordinate at the fourth vertex of the quadrilateral

See also **nema_set_blend_blit()**.

See also **nema_blit_subrect_quad_fit()**.

```
void nema_blit_quad_m ( float dx0, float dy0, float dx1, float
dy1, float dx2, float dy2, float dx3, float dy3, nema_matrix3x3_t
m )
```

Blit source texture to destination. Use the matrix provided by the user.

Parameters	Description
dx0	x coordinate at the first vertex of the quadrilateral
dy0	y coordinate at the first vertex of the quadrilateral
dx1	x coordinate at the second vertex of the quadrilateral
dy1	y coordinate at the second vertex of the quadrilateral
dx2	x coordinate at the third vertex of the quadrilateral
dy2	y coordinate at the third vertex of the quadrilateral
dx3	x coordinate at the fourth vertex of the quadrilateral
dy3	y coordinate at the fourth vertex of the quadrilateral
m	3x3 matrix (screen coordinates to texture coordinates)

See also **nema_set_blend_blit()**.

```
void nema_blit_rect (  int x,  int y,  int w,  int h )
```

Blit source texture to destination's specified rectangle (crop or wrap when needed)

Parameters	Description
x	destination x coordinate
y	destination y coordinate
w	destination width
h	destination height

See also **nema_set_blend_blit()**.

```
void nema_blit_rect_fit (  int x,  int y,  int w,  int h )
```

Blit source texture to destination. Fit (scale) texture to specified rectangle.

Parameters	Description
x	destination x coordinate
y	destination y coordinate
w	destination width
h	destination height

See also nema_set_blend_blit().

```
void nema_blit_rotate (  int x,  int y,  uint32_t rotation )
```

Rotate and Blit source texture to destination.

Parameters	Description
x	destination x coordinate
y	destination y coordinate
rotation	Rotation to be done

See also **nema_set_blend_blit()**.

```
void nema_blit_rotate_partial (  int sx,  int sy,  int sw,  int sh,
int x,  int y, uint32_t rotation )
```

Rotate and Blit partial source texture to destination.

Parameters	Description
sx	source upper left x coordinate
sy	source upper left y coordinate
sw	source width of partial region
sh	source height of partial region
x	destination x coordinate
y	destination y coordinate
rotation	Rotation to be done

See also **nema_set_blend_blit()**.

```
void nema_blit_rotate_pivot ( float cx, float cy, float px, float
py, float degrees_cw )
```

Rotate around pivot point and Blit source texture.

Parameters	Description
cx	destination rotation center x coordinate
cy	destination rotation center y coordinate
px	source pivot point x coordinate
py	source pivot point y coordinate
degrees_cw	degrees of clockwise rotation in range [0, 360]

See also **nema_set_blend_blit()**.

```
void nema_blit_rounded ( int x, int y, int r )
```

Blit source texture to destination texture with rounded corners.

Parameters	Description
x	destination x coordinate
y	destination y coordinate
r	destination corner radius

See also **nema_set_blend_fill()**.

```
void nema_blit_subrect ( int dst_x, int dst_y, int w, int h, int
src_x, int src_y )
```

Blit part of a source texture to destination's specified rectangle (crop or wrap when needed)

Parameters	Description
x	destination x coordinate
y	destination y coordinate
w	destination width
h	destination height
x	source x coordinate
y	source y coordinate

See also **nema_blit_subrect()**.

See also **nema_set_blend_blit()**.

```
void nema_blit_subrect_fit ( int dst_x, int dst_y, int dst_w, int
dst_h, int src_x, int src_y, int src_w, int src_h )
```

Blit part of source texture to destination. Fit (scale) texture to specified rectangle.

Parameters	Description
x	destination x coordinate
y	destination y coordinate
w	destination width
h	destination height
x	source x coordinate
y	source y coordinate
w	source width
h	source height

See also **nema_blit_rect_fit()**.

See also **nema_set_blend_blit()**.

```
void nema_blit_subrect_quad_fit ( float dx0, float dy0, float dx1,
float dy1, float dx2, float dy2, float dx3, float dy3, int sx,
int sy, int sw, int sh )
```

Blit source texture to destination. Fit rectangulare area of texture to specified quadrilateral.

Parameters	Description
dx0	x coordinate at the first vertex of the quadrilateral
dy0	y coordinate at the first vertex of the quadrilateral
dx1	x coordinate at the second vertex of the quadrilateral
dy1	y coordinate at the second vertex of the quadrilateral
dx2	x coordinate at the third vertex of the quadrilateral
dy2	y coordinate at the third vertex of the quadrilateral
dx3	x coordinate at the fourth vertex of the quadrilateral
dy3	y coordinate at the fourth vertex of the quadrilateral
sx	x coordinate of the top left corner of the texture's rectangular area to be blitted
sy	y coordinate of the top left corner of the texture's rectangular area to be blitted
sw	width of the texture's rectangular area to be blitted
sh	height of the texture's rectangular area to be blitted

See also **nema_set_blend_blit()**.

See also **nema_blit_quad_fit()**.

```
void nema_blit_tri_fit (  float dx0,  float dy0,  int v0,  float dx1,
float dy1, int v1,  float dx2,  float dy2,  int v2 )
```

Blit source texture to destination. Fit texture to specified triangle.

Parameters	Description
dx0	x coordinate at the first vertex of the triangle
dy0	y coordinate at the first vertex of the triangle
v0	in [0, 3] indicates the corner of the texture that fits to the first vertex of the triangle 0 _ _ 1 |_ _| 3 2
dx1	x coordinate at the second vertex of the triangle
dy1	y coordinate at the second vertex of the triangle
v1	in [0, 3] indicates the corner of the texture that fits to the second vertex of the triangle
dx2	x coordinate at the third vertex of the triangle
dy2	y coordinate at the third vertex of the triangle
v2	in [0, 3] indicates the corner of the texture that fits to the third vertex of the triangle

See also **nema_set_blend_blit()**.

```
void nema_blit_tri_uv ( float dx0, float dy0, float dw0, float
dx1, float dy1, float dw1, float dx2, float dy2, float dw2,
float sx0, float sy0, float sx1, float sy1, float sx2, float sy2
)
```

Blit a triangular part of the source tecture to a triangular destination area.

Parameters	Description
dx0	x coordinate at the first vertex of the destination triangle
dy0	y coordinate at the first vertex of the destination triangle
dw0	w coordinate at the first vertex of the destination triangle
dx1	x coordinate at the second vertex of the destination triangle
dy1	y coordinate at the second vertex of the destination triangle
dw1	w coordinate at the second vertex of the destination triangle
dx2	x coordinate at the third vertex of the destination triangle
dy2	y coordinate at the third vertex of the destination triangle
dw2	w coordinate at the third vertex of the destination triangle
sx0	x coordinate at the first vertex of the source triangle
sy0	y coordinate at the first vertex of the source triangle
sx1	x coordinate at the second vertex of the source triangle
sy1	y coordinate at the second vertex of the source triangle
sx2	x coordinate at the third vertex of the source triangle
sy2	y coordinate at the third vertex of the source triangle

See also **nema_set_blend_blit()**.

```
int nema_brk_add ( void )
```

Add a breakpoint to the current Command List.

Return

Breakpoint ID

```
void nema_brk_disable ( void )
```

Disable breakpoints.

See also **nema_brk_enable()**.

```
void nema_brk_enable (  void )
```

Enable breakpoints.

See also **nema_brk_disable()**.

```
int nema_brk_wait (  int brk_id )
```

Add a breakpoint to the current Command List.

Parameters	Description
brk_id	Breakpoint ID to wait for. If zero (0), wait until next Breakpoint

Return

ID of reached Breakpoint

```
int nema_checkGPUPresence (  void )
```

Check if a known GPU is present.

Return

-1 if no known GPU is present

```
void nema_clear (  uint32_t rgba8888 )
```

Clear destination texture with color.

Parameters	Description
rgba8888	32-bit RGBA color

See also **nema_rgba()**.

```
void nema_clear_depth (  uint32_t val )
```

Clear depth buffer with specified value.

Parameters	Description
val	Clear value

```
void nema_clear_dirty_region (  void )
```

Clear dirty region information - runs via the bound command-list. Available only on Nema P and Nema PVG GPUs.

See also **nema_get_dirty_region()**.

See also **nema_clear_dirty_region_imm()**.

`void nema_clear_dirty_region_imm ( void )`

Clear dirty region information immediately, no command-list involved. Available only on Nema|P and Nema|PVG GPUs.

See also **nema_get_dirty_region()**.

See also **nema_clear_dirty_region()**.

`void nema_draw_circle ( int x, int y, int r, uint32_t rgba8888 )`

Draw a colored circle with 1 pixel width.

Parameters	Description
x	x coordinate of the circle's center
y	y coordinate of the circle's center
r	circle's radius
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

`void nema_draw_circle_aa ( float x, float y, float r, float w, uint32_t rgba8888 )`

Draw a colored circle with Anti-Aliasing (if available) and specified width.

Parameters	Description
x	x coordinate of the circle's center
y	y coordinate of the circle's center
r	circle's radius
w	pencil width
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

`void nema_draw_line ( int x0, int y0, int x1, int y1, uint32_t rgba8888 )`

Draw a colored line.

Parameters	Description
x0	x coordinate at the beginning of the line
y0	y coordinate at the beginning of the line
x1	x coordinate at the end of the line
y1	y coordinate at the end of the line
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_draw_line_aa ( float x0, float y0, float x1, float y1,
float w, uint32_t rgba8888 )
```

Draw a line with width. Apply AA if available.

Parameters	Description
x0	x coordinate at the beginning of the line
y0	y coordinate at the beginning of the line
x1	x coordinate at the end of the line
y1	y coordinate at the end of the line
w	line width
rgba8888	Color to be used

See also **nema_draw_line()**.

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_draw_rect ( int x, int y, int w, int h, uint32_t
rgba8888 )
```

Draw a colored rectangle.

Parameters	Description
x	x coordinate of the upper left vertex of the rectangle
y	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle

Parameters	Description
h	height of the rectangle
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_draw_rounded_rect ( int x0, int y0, int w, int h, int
r, uint32_t rgba8888 )
```

Draw a colored rectangle with rounded edges.

Parameters	Description
x0	x coordinate of the upper left vertex of the rectangle
y0	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
r	corner radius
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
uint32_t nema_enable_aa ( uint8_t e0, uint8_t e1, uint8_t e2,
uint8_t e3 )
```

Enables MSAA per edge.

Parameters	Description
e0	Enable MSAA for edge 0 (vertices 0-1)
e1	Enable MSAA for edge 1 (vertices 1-2)
e2	Enable MSAA for edge 2 (vertices 2-3)
e3	Enable MSAA for edge 3 (vertices 3-0)

Return

previous AA flags (may be ignored)

```
uint32_t nema_enable_aa_flags ( uint32_t aa )
```

Enables MSAA per edge.

Parameters	Description
aa	A combination of the flags RAST_AA_E0, RAST_AA_E1, RAST_AA_E2, RAST_AA_E3

Return

Previous AA flags (may be ignored)

```
void nema_ext_hold_assert (  uint32_t hold_id,  int stop )
```

Assert hold signals internally via a Command List.

Parameters	Description
hold_id	Hold signal to be asserted
stop	If not zero, force Command List Processor to wait for FLAG to be deasserted

See also **nema_ext_hold_deassert()**.

```
void nema_ext_hold_assert_imm (  uint32_t hold_id )
```

Assert hold signals from the CPU (no Command List)

Parameters	Description
hold_id	Hold signal to be asserted

See also **nema_ext_hold_deassert()**.

```
void nema_ext_hold_deassert (  uint32_t hold_id )
```

Dessert hold signals internally via a Command List.

Parameters	Description
hold_id	Hold signal to be deasserted

See also **nema_ext_hold_assert()**.

```
void nema_ext_hold_deassert_imm (  uint32_t hold_id )
```

Dessert hold signals from the CPU (no Command List)

Parameters	Description
hold_id	Hold signal to be deasserted

See also **nema_ext_hold_assert()**.

void nema_ext_hold_disable (uint32_t hold_id)

Disable external hold signals.

Parameters	Description
hold_id	Hold signals to be disabled [0-3]

See also **nema_ext_hold_enable()**.

void nema_ext_hold_enable (uint32_t hold_id)

Enable external hold signals.

Parameters	Description
hold_id	Hold signals to be enabled [0-3]

See also **nema_ext_hold_disable()**.

void nema_ext_hold_irq_disable (uint32_t hold_id)

Disable external hold signals.

Parameters	Description
hold_id	Hold signals' IRQ to be disabled [0-3]

See also **nema_ext_hold_enable()**.

void nema_ext_hold_irq_enable (uint32_t hold_id)

Enable Interrupt Request when GPU reaches hold point.

Parameters	Description
hold_id	Hold signals' IRQ to be enabled [0-3]

See also **nema_ext_hold_disable()**.

void nema_fill_circle (int x, int y, int r, uint32_t rgba8888)

Fill a circle with color.

Parameters	Description
x	x coordinate of the circle's center
y	y coordinate of the circle's center
r	circle's radius
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_circle_aa ( float x, float y, float r, uint32_t
rgba8888 )
```

Fill a circle with color, use Anti-Aliasing if available.

Parameters	Description
x	x coordinate of the circle's center
y	y coordinate of the circle's center
r	circle's radius
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_quad ( int x0, int y0, int x1, int y1, int x2,
int y2, int x3, int y3, uint32_t rgba8888 )
```

Fill a quadrilateral with color.

Parameters	Description
x0	x coordinate at the first vertex of the quadrilateral
y0	y coordinate at the first vertex of the quadrilateral
x1	x coordinate at the second vertex of the quadrilateral
y1	y coordinate at the second vertex of the quadrilateral
x2	x coordinate at the third vertex of the quadrilateral
y2	y coordinate at the third vertex of the quadrilateral
x3	x coordinate at the fourth vertex of the quadrilateral

Parameters	Description
y3	y coordinate at the fourth vertex of the quadrilateral
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_quad_f ( float x0, float y0, float x1, float y1,
float x2, float y2, float x3, float y3, uint32_t rgba8888 )
```

Fill a quadrilateral with color (float coordinates)

Parameters	Description
x0	x coordinate at the first vertex of the quadrilateral
y0	y coordinate at the first vertex of the quadrilateral
x1	x coordinate at the second vertex of the quadrilateral
y1	y coordinate at the second vertex of the quadrilateral
x2	x coordinate at the third vertex of the quadrilateral
y2	y coordinate at the third vertex of the quadrilateral
x3	x coordinate at the fourth vertex of the quadrilateral
y3	y coordinate at the fourth vertex of the quadrilateral
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_rect ( int x, int y, int w, int h, uint32_t
rgba8888 )
```

Fill a rectangle with color.

Parameters	Description
x	x coordinate of the upper left vertex of the rectangle
y	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_rect_f ( float x, float y, float w, float h,
uint32_t rgba8888 )
```

Fill a rectangle with color (float coordinates)

Parameters	Description
x	x coordinate of the upper left vertex of the rectangle
y	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_rounded_rect ( int x0, int y0, int w, int h, int
r, uint32_t rgba8888 )
```

Fill a rectangle with rounded edges with color.

Parameters	Description
x0	x coordinate of the upper left vertex of the rectangle
y0	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
r	corner radius
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

See also **nema_rgba()**.

```
void nema_fill_triangle ( int x0, int y0, int x1, int y1, int x2,
int y2, uint32_t rgba8888 )
```

Fill a triangle with color.

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

```
void nema_fill_triangle_f ( float x0, float y0, float x1, float
y1, float x2, float y2, uint32_t rgba8888 )
```

Fill a triangle with color (float coordinates)

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
rgba8888	Color to be used

See also **nema_set_blend_fill()**.

```
int nema_format_size ( nema_tex_format_t format )
```

Return pixel size in bytes.

Parameters	Description
format	Color format

Return

Pixel size in bytes

```
void nema_get_dirty_region (  int * minx,  int * miny,  int * maxx,
int * maxy)
```

Returns the bounding rectangle of all the pixels that have been modified since its previous call. Available only on Nema|P and Nema|PVG GPUs.

Parameters	Description
minx	x coordinate of the upper left corner of the dirty region
miny	y coordinate of the upper left corner of the dirty region
maxx	x coordinate of the lower right corner of the dirty region
maxy	y coordinate of the lower right corner of the dirty region

```
const char * nema_get_sw_device_name (  void )
```

Check for which architeture is the library compiled.

Return

Returns string with the architecture name

```
int nema_init (  void )
```

Initialize NemaGFX library.

Return

negative value on error

```
uint32_t nema_premultiply_rgba (  uint32_t rgba )
```

Pre-multiply RGB channels with Alpha channel.

Parameters	Description
rgba	RGBA color

Return

Premultiplied RGBA color

```
int nema_reinit (  void )
```

Reinitialize NemaGFX library.

This function reinitializes the NemaGFX library after a GPU poweroff No memory allocation for ringbuffer etc is performed.

Return

negative value on error

```
uint32_t nema_rgba ( unsigned char R, unsigned char G, unsigned
char B, unsigned char A )
```

Return Nema internal RGBA color.

Parameters	Description
R	Red component
G	Green component
B	Blue component
A	Alpha component

Return

RGBA value

```
void nema_set_clip ( int32_t x, int32_t y, uint32_t w, uint32_t h
)
```

Sets the drawing area's Clipping Rectangle.

Parameters	Description
x	Clip Window top-left x coordinate
y	Clip Window minimum y
w	Clip Window width
h	Clip Window height

```
void nema_set_const_reg ( int reg, uint32_t value )
```

Write a value to a Constant Register of the GPU.

Parameters	Description
reg	Constant Register to be written
value	Value to be written

```
void nema_set_tex_color ( uint32_t color )
```

Set Texture Mapping default color.

Parameters	Description
color	default color in 32-bit RGBA format

See also **nema_rgba()**.

```
int nema_stride_size ( nema_tex_format_t format, nema_tex_mode_t
wrap_mode, int width )
```

Return stride in bytes.

Parameters	Description
format	Color format
wrap_mode	Wrap/Repeat mode to be used. When using 'repeat' or 'mirror', texture dimensions must be a power of two. Otherwise the behavior is undefined.
width	Texture color format

Return

Stride in bytes

```
int nema_texture_size ( nema_tex_format_t format, nema_tex_mode_t
wrap_mode, int width, int height )
```

Return texture size in bytes.

Parameters	Description
format	Texture color format
wrap_mode	Wrap/Repeat mode to be used. When using 'repeat' or 'mirror', texture dimensions must be a power of two. Otherwise the behavior is undefined.
width	Texture width
height	Texture height

Return

Texture size in bytes

```
void nema_tri_cull ( nema_tri_cull_t cull )
```

Set triangle/quadrilateral culling mode.

Parameters	Description
cull	Culling mode

11.1.7 nema_hal.h

```
#include "nema_sys_defs.h"
```

Data Structures

```
struct nema_buffer_t
```

More...

```
struct nema_ringbuffer_t
```

More...

Macros

```
#define MUTEX_RB 0
#define MUTEX_MALLOC 1
#define MUTEX_FLUSH 2
#define MUTEX_MAX 2
```

11.1.7.1 Functions

```
int32_t nema_sys_init(void)
```

Initialize system. Implementor defined. Called in

```
nema_init() int nema_wait_irq(void)
```

Wait for interrupt from the GPU.

```
int nema_wait_irq_cl(int cl_id)
```

Wait for a Command List to finish.

```
int nema_wait_irq_brk(int brk_id)
```

Wait for a Breakpoint.

```
uint32_t nema_reg_read(uint32_t reg)
```

Read Hardware register.

```
void nema_reg_write(uint32_t reg, uint32_t value)
```

Write Hardware Register.

```
nema_buffer_t nema_buffer_create(int size)
```

Create memory buffer.

```
nema_buffer_t nema_buffer_create_pool(int pool, int size)
```

Create memory buffer at a specific pool.

```
void* nema_buffer_map(nema_buffer_t *bo)
```

Maps buffer.

`void nema_buffer_unmap(nema_buffer_t *bo)`

Unmaps buffer.

`void nema_buffer_destroy(nema_buffer_t *bo)`

Destroy/deallocate buffer.

`uintptr_t nema_buffer_phys(nema_buffer_t *bo)`

Get physical (GPU) base address of a given buffer.

`void nema_buffer_flush(nema_buffer_t *bo)`

Write-back buffer from cache to main memory.

`void* nema_host_malloc(size_t size)`

Allocate memory for CPU to use (typically, standard malloc() is called)

`void nema_host_free(void *ptr)`

Free memory previously allocated with

`nema_host_malloc() int nema_rb_init(nema_ringbuffer_t *rb, int reset)`

Initialize Ring Buffer. Should be called from inside **nema_sys_init()**. This is a private function, the user should never call it.

`int nema_mutex_lock(int mutex_id)`

Mutex Lock for multiple processes/threads.

`int nema_mutex_unlock(int mutex_id)`

Mutex Unlock for multiple processes/threads.

11.1.7.2 Detailed Description

Macro Definition Documentation

11.1.7.3 Function Documentation

nema_buffer_t nema_buffer_create (int size)

Create memory buffer.

Parameters	Description
size	Size of buffer in bytes

Return

nema_buffer_t struct

```
nema_buffer_t nema_buffer_create_pool (  int pool,  int size )
```

Create memory buffer at a specific pool.

Parameters	Description
pool	ID of the desired memory pool
size	Size of buffer in bytes

Return

nema_buffer_t struct

```
void nema_buffer_destroy (  nema_buffer_t * bo )
```

Destroy/deallocate buffer.

Parameters	Description
bo	Pointer to buffer struct

Return

void

```
void nema_buffer_flush (  nema_buffer_t * bo )
```

Write-back buffer from cache to main memory.

Parameters	Description
bo	Pointer to buffer struct

Return

void

```
void * nema_buffer_map (  nema_buffer_t * bo )
```

Maps buffer.

Parameters	Description
bo	Pointer to buffer struct

Return

Virtual pointer of the buffer (same as in bo->base_virt)

```
uintptr_t nema_buffer_phys (  nema_buffer_t * bo )
```

Get physical (GPU) base address of a given buffer.

Parameters	Description
bo	Pointer to buffer struct

Return

Physical base address of a given buffer

```
void nema_buffer_unmap (  nema_buffer_t * bo )
```

Unmaps buffer.

Parameters	Description
bo	Pointer to buffer struct

Return

void

```
void nema_host_free (  void * ptr )
```

Free memory previously allocated with **nema_host_malloc()**.

Parameters	Description
ptr	Pointer to allocated memory (virtual)

Return

void

See also **nema_host_malloc()**.

```
void * nema_host_malloc (  size_t size )
```

Allocate memory for CPU to use (typically, standard **malloc()** is called).

Parameters	Description
size	Size in bytes

Return

Pointer to allocated memory (virtual).

See also **nema_host_free()**.

```
int nema_mutex_lock (  int mutex_id )
```

Mutex Lock for multiple processes/threads.

Parameters	Description
MUTEX_RB	or MUTEX_MALLOC

Return

int

```
int nema_mutex_unlock (  int mutex_id )
```

Mutex Unlock for multiple processes/threads.

Parameters	Description
MUTEX_RB	or MUTEX_MALLOC

Return

int

```
int nema_rb_init (  nema_ringbuffer_t * rb,  int reset )
```

Initialize Ring Buffer. Should be called from inside **nema_sys_init()**. This is a private function, the user should never call it.

Parameters	Description
*rb	Pointer to nema_ring_buffer_t struct
reset	Resets the Ring Buffer if non-zero

Return

Negative number on error.

See also **nema_sys_init()**.

```
uint32_t nema_reg_read (  uint32_t reg )
```

Read Hardware register.

Parameters	Description
reg	Register to read

Return

Value read from the register.

See also **nema_reg_write**.

```
void nema_reg_write (  uint32_t reg,  uint32_t value )
```

Write Hardware Register.

Parameters	Description
reg	Register to write
value	Value to be written

Return

void()

See also **nema_reg_read()**.

```
int32_t nema_sys_init (  void )
```

Initialize system. Implementor defined. Called in **nema_init()**.

Parameters	Description
void	

Return

0 if no errors occurred.

See also **nema_init()**.

```
int nema_wait_irq (  void )
```

Wait for interrupt from the GPU.

Parameters	Description
void	

Return

0 on success

```
int nema_wait_irq_brk (  int brk_id )
```

Wait for a Breakpoint.

Parameters	Description
cl_id	Breakpoint ID

Return

0 on success

```
int nema_wait_irq_cl (  int cl_id )
```

Wait for a Command List to finish.

Parameters	Description
cl_id	Command List ID

Return

0 on success

11.1.8 nema_interpolators.h

#include "nema_sys_defs.h"

Data Structures struct color_var_t

More...

11.1.8.1 Functions

```
void nema_interpolate_rect_colors(int x0, int y0, int w, int h,
color_var_t *col0, color_var_t *col1, color_var_t *col2)
```

Interpolate color gradient for rectangle.

```
void nema_interpolate_tri_colors(float x0, float y0, float x1,
float y1, float x2, float y2, color_var_t *col0, color_var_t
*col1, color_var_t *col2)
```

Interpolate color gradient for triangle.

```
void nema_interpolate_tri_depth(float x0, float y0, float z0,
float x1, float y1, float z1, float x2, float y2, float z2)
```

Interpolate depth buffer values for triangle.

```
void nema_interpolate_tx_ty(float x0, float y0, float w0, float
tx0, float ty0, float x1, float y1, float w1, float tx1, float
ty1, float x2, float y2, float w2, float tx2, float ty2, int
tex_width, int tex_height)
```

Interpolate texture values for triangle.

Detailed Description

11.1.8.2 Function Documentation

```
void nema_interpolate_rect_colors ( int x0, int y0, int w, int h,
color_var_t * col0, color_var_t * col1, color_var_t * col2 )
```

Interpolate color gradient for rectangle.

Parameters	Description
x0	x coordinate of the upper left vertex of the rectangle
y0	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
col0	color for the first vertex
col1	color for the second vertex
col1	color for the third vertex

```
void nema_interpolate_tri_colors ( float x0, float y0, float x1,
float y1, float x2, float y2, color_var_t * col0, color_var_t *
col1, color_var_t * col2 )
```

Interpolate color gradient for triangle.

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
col0	color for the first vertex
col1	color for the second vertex
col1	color for the third vertex

```
void nema_interpolate_tri_depth ( float x0, float y0, float z0,
float x1, float y1, float z1, float x2, float y2, float z2 )
```

Interpolate depth buffer values for triangle.

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
z0	z coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
z1	z coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
z2	z coordinate at the third vertex of the triangle

```
void nema_interpolate_tx_ty ( float x0, float y0, float w0, float
tx0, float ty0, float x1, float y1, float w1, float tx1, float
ty1, float x2, float y2, float w2, float tx2, float ty2, int
tex_width, int tex_height )
```

Interpolate texture values for triangle.

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
w0	w coordinate at the first vertex of the triangle
tx0	x texture coordinate at the first vertex of the triangle
ty0	y texture coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
w1	w coordinate at the second vertex of the triangle
tx1	x texture coordinate at the second vertex of the triangle
ty1	y texture coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
w2	w coordinate at the third vertex of the triangle

Parameters	Description
tx2	x texture coordinate at the third vertex of the triangle
ty2	x texture coordinate at the third vertex of the triangle
tex_width	texture width
tex_height	texture height

11.1.9 nema_math.h

Macros

```
#define NEMA_E 2.71828182845904523536f
#define NEMA_LOG2E 1.44269504088896340736f
#define NEMA_LOG10E 0.434294481903251827651f
#define NEMA_LN2 0.693147180559945309417f
#define NEMA_LN10 2.30258509299404568402f
#define NEMA_PI 3.14159265358979323846f
#define NEMA_PI_2 1.57079632679489661923f
#define NEMA_PI_4 0.785398163397448309616f
#define NEMA_1_PI 0.318309886183790671538f
#define NEMA_2_PI 0.636619772367581343076f
#define NEMA_2_SQRTPI 1.12837916709551257390f
#define NEMA_SQRT2 1.41421356237309504880f
#define NEMA_SQRT1_2 0.707106781186547524401f
#define nema_min2 (((a)<(b))?( a):(b))
```

Find the minimum of two values.

```
#define nema_max2 (((a)>(b))?( a):(b))
```

Find the maximum of two values.

```
#define nema_clamp nema_min2((max), nema_max2((min), (val)))
```

Clamp value.

```
#define nema_abs (((a)< 0 )?(-(a)):(a))
```

Calculate the absolute value of int.

```
#define nema_absf (((a)< 0.f )?(-(a)):(a))
```

Calculate the absolute value of float.

```
#define nema_floats_equal (nema_absf((x) - (y)) <= 0.00001f *
nema_min2(nema_absf(x), nema_absf(y)))
```

Compare two floats.

```
#define nema_float_is_zero (nema_absf(x) <= 0.00001f)
```

Checks if value x is zero.

```
#define nema_deg_to_rad (0.0174532925199f * (d))
```

Convert degrees to radians.

```
#define nema_rad_to_deg (57.295779513f * (r))
```

Convert radians to degries.

```
#define nema_i2fx ((a)*0x10000)
```

Convert integer to 16.16 fixed point.

```
#define nema_floor ((int)(f) - ( (int)(f) > (f) ))
```

Floor function.

```
#define nema_ceil ((int)(f) + ( (int)(f) < (f) ))
```

Ceiling function.

```
#define nema_truncf (x < 0.0f ? nema_ceil(x) : nema_floor(x))
```

Truncate function.

```
#define nema_fmod ( (x) - nema_truncf( ( (x) / (y) ) ) * (y) )
```

Float Modulo function.

11.1.9.1 Functions

```
float nema_sin(float angle_degrees)
```

Fast sine approximation of a given angle.

```
float nema_cos(float angle_degrees)
```

Fast cosine approximation of a given angle.

```
float nema_tan(float angle_degrees)
```

Fast tangent approximation of a given angle.

```
float nema_sin_r(float angle_radians)
```

Fast sine approximation of a given angle.

```
float nema_cos_r(float angle_radians)
```

Fast cosine approximation of a given angle.

```
float nema_tan_r(float angle_radians)
```

Fast tangent approximation of a given angle.

float nema_atan2(float y, float x)

Fast arc tangent approximation of a y/x.

float nema_atan2_r(float y, float x)

Fast arc tangent approximation of a y/x.

float nema_pow(float x, float y)

A rough approximation of x raised to the power of y. USE WITH CAUTION!

float nema_sqrt(float x)

A rough approximation of the square root of x. USE WITH CAUTION!

float nema_atan(float x)

A floating-point approximation of the inverse tangent of x.

int nema_f2fx(float f)

Convert float to 16.16 fixed point.

11.1.9.2 Detailed Description

Macro Definition Documentation

#define NEMA_1_PI

1/pi

#define NEMA_2_PI

2/pi

#define NEMA_2_SQRTPI

2/sqrt(pi)

#define NEMA_E e

#define NEMA_LN10

ln(10)

#define NEMA_LN2

ln(2)

#define NEMA_LOG10E

log10(e)

#define NEMA_LOG2E

log2(e)

#define NEMA_PI

pi

#define NEMA_PI_2

pi/2

#define NEMA_PI_4

pi/4

#define NEMA_SQRT1_2

1/sqrt(2)

#define NEMA_SQRT2

sqrt(2)

#define nema_abs

Calculate the absolute value of int.

Parameters	Description
a	Value

Return

The absolute value of a

#define nema_absf

Calculate the absolute value of float.

Parameters	Description
a	Value

Return

The absolute value of a

#define nema_ceil

Ceiling function.

Parameters	Description
a	Value to be ceiled

Return

ceiled value

#define nema_clamp

Clamp value.

Parameters	Description
val	Value to clamp
min	Minimum value
max	Minimum value

Return

Clamped value

#define nema_deg_to_rad

Convert degrees to radians.

Parameters	Description
d	Angle in degrees

Return

Angle in radians

#define nema_float_is_zero

Checks if value x is zero.

Parameters	Description
x	X value

Return

1 if x == 0, 0 if x != 0

#define nema_floats_equal

Compare two floats.

Parameters	Description
x	First float
y	Second float

Return

1 if x == y, 0 if x != y

`#define nema_floor`

Floor function.

Parameters	Description
a	Value to be floored

Return

floored value

`#define nema_fmod`

Float Modulo function.

Parameters	Description
x	Dividend
y	Divisor

Return

Remainder

`#define nema_i2fx`

Convert integer to 16.16 fixed point.

Parameters	Description
a	Value to be converted

Return

16.16 fixed point value

`#define nema_max2`

Find the maximum of two values.

Parameters	Description
a	First value
b	Second value

Return

The maximum of a and b

#define nema_min2

Find the minimum of two values.

Parameters	Description
a	First value
b	Second value

Return

The minimum of a and b

#define nema_rad_to_deg

Convert radians to degrees.

Parameters	Description
r	Angle in radians

Return

Angle in degrees

#define nema_truncf

Truncate function.

Parameters	Description
x	Value to be truncated

Return

truncated value

11.1.9.3 Function Documentation

float nema_atan (float x)

A floating-point approximation of the inverse tangent of x.

Parameters	Description
x	X value

Return

Inverse tangent (angle) of x in degrees

```
float nema_atan2 (  float y,  float x )
```

Fast arc tangent approximation of a y/x.

Parameters	Description
y	value
x	value

Return

Arc tangent of the given y/x in degrees

```
float nema_atan2_r (  float y,  float x )
```

Fast arc tangent approximation of a y/x.

Parameters	Description
y	value
x	value

Return

Arc tangent of the given y/x in radians

```
float nema_cos (  float angle_degrees )
```

Fast cosine approximation of a given angle.

Parameters	Description
angle_degrees	Angle in degrees

Return

Cosine of the given angle

```
float nema_cos_r (  float angle_radians )
```

Fast cosine approximation of a given angle.

Parameters	Description
angle_radians	Angle in radians

Return

Cosine of the given angle

```
int nema_f2fx (  float f )
```

Convert float to 16.16 fixed point.

Parameters	Description
a	Value to be converted

Return

16.16 fixed point value

```
float nema_pow (  float x,  float y )
```

A rough approximation of x raised to the power of y. USE WITH CAUTION!

Parameters	Description
x	base value. Must be non negative.
y	power value

Return

the result of raising x to the power y

float nema_sin (float angle_degrees)

Fast sine approximation of a given angle.

Parameters	Description
angle_degrees	Angle in degrees

Return

Sine of the given angle

```
float nema_sin_r (  float angle_radians )
```

Fast sine approximation of a given angle.

Parameters	Description
angle_radians	Angle in radians

Return

Sine of the given angle.

```
float nema_sqrt (  float x )
```

A rough approximation of the square root of x. USE WITH CAUTION!

Parameters	Description
x	X value. Must be non negative.

```
float nema_tan (  float angle_degrees )
```

Fast tangent approximation of a given angle.

Parameters	Description
angle_degrees	Angle in degrees

Return

Tangent of the given angle

```
float nema_tan_r (  float angle_radians )
```

Fast tangent approximation of a given angle.

Parameters	Description
angle_radians	Angle in radians

Return

Tangent of the given angle

11.1.10 nema_matrix3x3.h

Typedefs

```
typedef float  nema_matrix3x3_t [3][3]typedef float
nema_matrix3x3_t[3][3]
```

11.1.10.1 Functions

`void nema_mat3x3_load_identity(nema_matrix3x3_t m)`

Load Identity Matrix.

`void nema_mat3x3_copy(nema_matrix3x3_t m, nema_matrix3x3_t _m)`

Copy matrix _m to matrix m.

`void nema_mat3x3_translate(nema_matrix3x3_t m, float tx, float ty)`

Apply translate transformation.

`void nema_mat3x3_scale(nema_matrix3x3_t m, float sx, float sy)`

Apply scale transformation.

Apply shear transformation.

`void nema_mat3x3_mirror(nema_matrix3x3_t m, int mx, int my)`

Apply mirror transformation.

`void nema_mat3x3_rotate(nema_matrix3x3_t m, float angle_degrees)`

Apply rotation transformation.

`void nema_mat3x3_rotate2(nema_matrix3x3_t m, float cosa, float sina)`

Apply rotation transformation.

`void nema_mat3x3_mul(nema_matrix3x3_t m, nema_matrix3x3_t _m)`

Multiply two 3x3 matrices (m = m*_m)

`void nema_mat3x3_mul_vec(nema_matrix3x3_t m, float *x, float *y)`

Multiply vector with matrix.

`void nema_mat3x3_mul_vec_affine(nema_matrix3x3_t m, float *x, float *y)`

Multiply vector with affine matrix.

`void nema_mat3x3_adj(nema_matrix3x3_t m)`

Calculate adjoint.

`void nema_mat3x3_div_scalar(nema_matrix3x3_t m, float s)`

Divide matrix with scalar value.

`int nema_mat3x3_invert(nema_matrix3x3_t m)`

Invert matrix.

int nema_mat3x3_quad_to_rect(int width, int height, float sx0, float sy0, float sx1, float sy1, float sx2, float sy2, float sx3, float sy3, nema_matrix3x3_t m)

Map rectangle to quadrilateral.

void nema_mat3x3_rotate_pivot(nema_matrix3x3_t m, float angle_degrees, float x, float y)

Apply rotation around a pivot point.

void nema_mat3x3_scale_rotate_pivot(nema_matrix3x3_t m, float sx, float sy, float angle_degrees, float x, float y)

Apply scale and then rotation around a pivot point.

Detailed Description

Typedef Documentation

11.1.10.2 Function Documentation

void nema_mat3x3_adj (nema_matrix3x3_t m)

Calculate adjoint.

Parameters	Description
m	Matrix

void nema_mat3x3_copy (nema_matrix3x3_t m, nema_matrix3x3_t _m)

Copy matrix _m to matrix m.

Parameters	Description
m	Destination matrix
m	Source matrix

void nema_mat3x3_div_scalar (nema_matrix3x3_t m, float s)

Divide matrix with scalar value.

Parameters	Description
m	Matrix to divide
s	scalar value

int nema_mat3x3_invert (nema_matrix3x3_t m)

Invert matrix.

Parameters	Description
m	Matrix to invert

void nema_mat3x3_load_identity (nema_matrix3x3_t m)

Load Identity Matrix.

Parameters	Description
m	Matrix to be loaded

void nema_mat3x3_mirror (nema_matrix3x3_t m, int mx, int my)

Apply mirror transformation.

Parameters	Description
m	Matrix to apply transformation
mx	if non-zero, mirror horizontally
my	if non-zero, mirror vertically

void nema_mat3x3_mul (nema_matrix3x3_t m, nema_matrix3x3_t _m)

Multiply two 3x3 matrices (m = m*_m)

Parameters	Description
m	left matrix, will be overwritten by the result
_m	right matrix

void nema_mat3x3_mul_vec (nema_matrix3x3_t m, float * x, float * y)

Multiply vector with matrix.

Parameters	Description
m	Matrix to multiply with
x	Vector x coefficient
y	Vector y coefficient

void nema_mat3x3_mul_vec_affine (nema_matrix3x3_t m, float * x, float * y)

Multiply vector with affine matrix.

Parameters	Description
m	Matrix to multiply with
x	Vector x coefficient
y	Vector y coefficient

```
int nema_mat3x3_quad_to_rect ( int width, int height, float sx0,
float sy0, float sx1, float sy1, float sx2, float sy2, float sx3,
float sy3, nema_matrix3x3_t m )
```

Map rectangle to quadrilateral.

Parameters	Description
width	Rectangle width
height	Rectangle height
sx0	x coordinate at the first vertex of the quadrilateral
sy0	y coordinate at the first vertex of the quadrilateral
sx1	x coordinate at the second vertex of the quadrilateral
sy1	y coordinate at the second vertex of the quadrilateral
sx2	x coordinate at the third vertex of the quadrilateral
sy2	y coordinate at the third vertex of the quadrilateral
sx3	x coordinate at the fourth vertex of the quadrilateral
sy3	y coordinate at the fourth vertex of the quadrilateral
m	Mapping matrix

```
void nema_mat3x3_rotate ( nema_matrix3x3_t m, float angle_degrees )
```

Apply rotation transformation.

Parameters	Description
m	Matrix to apply transformation
angle_degrees	Angle to rotate in degrees

```
void nema_mat3x3_rotate2 ( nema_matrix3x3_t m, float cosa, float
sina )
```

Apply rotation transformation.

Parameters	Description
m	Matrix to apply transformation
cosa	Cos of angle to rotate
sina	Sin of angle to rotate

```
void nema_mat3x3_rotate_pivot ( nema_matrix3x3_t m, float
angle_degrees, float x, float y )
```

Apply rotation around a pivot point.

Parameters	Description
m	Matrix to apply transformation
angle_degrees	Angle to rotate in degrees
x	X coordinate of the pivot point
y	Y coordinate of the pivot point

```
void nema_mat3x3_scale ( nema_matrix3x3_t m, float sx, float sy )
```

Apply scale transformation.

Parameters	Description
m	Matrix to apply transformation
sx	X scaling factor
sy	Y scaling factor

```
void nema_mat3x3_scale_rotate_pivot ( nema_matrix3x3_t m, float sx,
float sy, float angle_degrees, float x, float y )
```

Apply scale and then rotation around a pivot point.

Parameters	Description
m	Matrix to apply transformation
sx	X scaling factor
sy	Y scaling factor
angle_degrees	Angle to rotate in degrees
x	X coordinate of the pivot point
y	Y coordinate of the pivot point

void nema_mat3x3_shear (nema_matrix3x3_t m, float shx, float shy)

Apply shear transformation.

Parameters	Description
m	Matrix to apply transformation
shx	X shearing factor
shy	Y shearing factor

void nema_mat3x3_translate (nema_matrix3x3_t m, float tx, float ty)

Apply translate transformation.

Parameters	Description
m	Matrix to apply transformation
tx	X translation factor
ty	Y translation factor

11.1.11 nema_matrix4x4.h

Typedefs

typedef float nema_matrix4x4_t [4][4]typedef float nema_matrix4x4_t[4][4]

11.1.11.1 Functions

void nema_mat4x4_load_identity(nema_matrix4x4_t m)

Load a 4x4 Identity Matrix.

void nema_mat4x4_mul(nema_matrix4x4_t m, nema_matrix4x4_t m_l, nema_matrix4x4_t m_r)

Multiply two 4x4 matrices.

void nema_mat4x4_mul_vec(nema_matrix4x4_t m, float *x, float *y, float *z, float *w)

Multiply a 4x1 vector with a 4x4 matrix.

void nema_mat4x4_translate(nema_matrix4x4_t m, float tx, float ty, float tz)

Apply translate transformation.

void nema_mat4x4_scale(nema_matrix4x4_t m, float sx, float sy, float sz)

Apply scale transformation.

```
void nema_mat4x4_rotate_X(nema_matrix4x4_t m, float
angle_degrees)
```

Apply rotate transformation around X axis.

```
void nema_mat4x4_rotate_Y(nema_matrix4x4_t m, float
angle_degrees)
```

Apply rotate transformation around Y axis.

```
void nema_mat4x4_rotate_Z(nema_matrix4x4_t m, float
angle_degrees)
```

Apply rotate transformation around Z axis.

```
void nema_mat4x4_load_perspective(nema_matrix4x4_t m, float
fovy_degrees, float aspect, float nearVal, float farVal)
```

Set up a perspective projection matrix.

```
void nema_mat4x4_load_perspective_rh(nema_matrix4x4_t m, float
fovy_degrees, float aspect, float nearVal, float farVal)
```

Set up a Right Hand perspective projection matrix.

```
void nema_mat4x4_load_ortho(nema_matrix4x4_t m, float left, float
right, float bottom, float top, float nearVal, float farVal)
```

Set up an orthographic projection matrix.

```
void nema_mat4x4_load_ortho_2d(nema_matrix4x4_t m, float left,
float right, float bottom, float top)
```

Set up a 2D orthographic projection matrix.

```
void nema_mat4x4_look_at_rh(nema_matrix4x4_t m, float eye_x,
float eye_y, float eye_z, float center_x, float center_y, float
center_z, float up_x, float up_y, float up_z)
```

Set up a Right Hand view matrix.

```
int nema_mat4x4_obj_to_win_coords(nema_matrix4x4_t mvp, float
x_orig, float y_orig, int width, int height, float nearVal, float
farVal, float *x, float *y, float *z, float *w)
```

Convenience Function to calculate window coordinates from object coordinates.

Detailed Description

Typedef Documentation

11.1.11.2 Function Documentation

```
void nema_mat4x4_load_identity (  nema_matrix4x4_t m )
```

Load a 4x4 Identity Matrix.

Parameters	Description
m	Matrix to be loaded

```
void nema_mat4x4_load_ortho ( nema_matrix4x4_t m, float left,
float right, float bottom, float top, float nearVal, float farVal )
```

Set up an orthographic projection matrix.

Parameters	Description
m	A 4x4 Matrix
left	Left vertical clipping plane
right	Right vertical clipping plane
bottom	bottom horizontal clipping plane
top	Top horizontal clipping plane
nearVal	Distance from the viewer to the near clipping plane (always positive)
farVal	Distance from the viewer to the far clipping plane (always positive)

```
void nema_mat4x4_load_ortho_2d ( nema_matrix4x4_t m, float left,
float right, float bottom, float top )
```

Set up a 2D orthographic projection matrix.

Parameters	Description
m	A 4x4 Matrix
left	Left vertical clipping plane
right	Right vertical clipping plane
bottom	bottom horizontal clipping plane
top	Top horizontal clipping plane

```
void nema_mat4x4_load_perspective ( nema_matrix4x4_t m, float
fovy_degrees, float aspect, float nearVal, float farVal )
```

Set up a perspective projection matrix.

Parameters	Description
m	A 4x4 Matrix
fovy_degrees	Field of View in degrees
aspect	Aspect ratio that determines the field of view in the x direction.
nearVal	Distance from the viewer to the near clipping plane (always positive)
farVal	Distance from the viewer to the far clipping plane (always positive)

```
void nema_mat4x4_load_perspective_rh ( nema_matrix4x4_t m, float fovy_degrees, float aspect, float nearVal, float farVal )
```

Set up a Right Hand perspective projection matrix.

Parameters	Description
m	A 4x4 Matrix
fovy_degrees	Field of View in degrees
aspect	Aspect ratio that determines the field of view in the x direction.
nearVal	Distance from the viewer to the near clipping plane (always positive)
farVal	Distance from the viewer to the far clipping plane (always positive)

```
void nema_mat4x4_look_at_rh ( nema_matrix4x4_t m, float eye_x, float eye_y, float eye_z, float center_x, float center_y, float center_z, float up_x, float up_y, float up_z )
```

Set up a Right Hand view matrix.

Parameters	Description
m	A 4x4 Matrix
eye_x	Eye position x.
eye_y	Eye position y.
eye_z	Eye position z.
center_x	Center x to look at
center_y	Center y to look at

Parameters	Description
center_z	Center z to look at
up_x	Up vector x. (Usually 0)
up_y	Up vector y. (Usually 1)
up_z	Up vector z. (Usually 0)

```
void nema_mat4x4_mul ( nema_matrix4x4_t m, nema_matrix4x4_t m_l,
nema_matrix4x4_t m_r )
```

Multiply two 4x4 matrices.

Parameters	Description
m	Result Matrix
m_l	Left operand
m_r	Right operand

```
void nema_mat4x4_mul_vec ( nema_matrix4x4_t m, float * x, float *
y, float * z, float * w )
```

Multiply a 4x1 vector with a 4x4 matrix.

Parameters	Description
m	Matrix to be multiplied
x	Vector first element
y	Vector second element
z	Vector third element
w	Vector forth element

```
int nema_mat4x4_obj_to_win_coords ( nema_matrix4x4_t mvp, float
x_orig, float y_orig, int width, int height, float nearVal, float
farVal, float * x, float * y, float * z, float * w )
```

Convenience Function to calculate window coordinates from object coordinates.

Parameters	Description
mvp	Model, View and Projection Matrix
x_orig	Window top left X coordinate
y_orig	Window top left Y coordinate
width	Window width

Parameters	Description
height	Window height
nearVal	Distance from the viewer to the near clipping plane (always positive)
farVal	Distance from the viewer to the far clipping plane (always positive)
x	X object coordinate
y	Y object coordinate
z	Z object coordinate
w	W object coordinate

Return

1 if vertex is outside frustum (should be clipped)

`void nema_mat4x4_rotate_X ( nema_matrix4x4_t m, float angle_degrees )`

Apply rotate transformation around X axis.

Parameters	Description
m	Matrix to apply transformation
angle_degrees	Angle to rotate in degrees

`void nema_mat4x4_rotate_Y ( nema_matrix4x4_t m, float angle_degrees )`

Apply rotate transformation around Y axis.

Parameters	Description
m	Matrix to apply transformation
angle_degrees	Angle to rotate in degrees

`void nema_mat4x4_rotate_Z ( nema_matrix4x4_t m, float angle_degrees )`

Apply rotate transformation around Z axis.

Parameters	Description
m	Matrix to apply transformation
angle_degrees	Angle to rotate in degrees

```
void nema_mat4x4_scale (  nema_matrix4x4_t m,  float sx,  float sy,
float sz )
```

Apply scale transformation.

Parameters	Description
m	Matrix to apply transformation
sx	X scaling factor
sy	Y scaling factor
sz	Z scaling factor

```
void nema_mat4x4_translate ( nema_matrix4x4_t m, float tx, float ty,
float tz )
```

Apply translate transformation.

Parameters	Description
m	Matrix to apply transformation
tx	X translation factor
ty	Y translation factor
tz	Z translation factor

11.1.12 nema_provisional.h

```
#include "nema_sys_defs.h"
```

11.1.12.1 Functions

```
void nema_fill_triangle_strip_f(float *vertices, int
num_vertices, int stride, uint32_t rgba8888)
```

Fill a triangle strip with color (float coordinates)

```
void nema_fill_triangle_fan_f(float *vertices, int num_vertices,
int stride, uint32_t rgba8888)
```

Fill a traingle fan with color (float coordinates)

```
void nema_draw_triangle_aa(float x0, float y0, float x1, float y1,
float x2, float y2, float border_width, uint32_t color)
```

Draws a triangle with specific border width. Apply AA if available. Degenerated triangles have undefined behavior.

```
void nema_draw_rounded_rect_aa(float x, float y, float w, float h,
float r, float border_width, uint32_t rgba8888)
```

Draw a colored rectangle with rounded edges and specific border width. Apply AA if available.

```
void nema_fill_rounded_rect_aa(float x, float y, float w, float h,
float r, uint32_t rgba8888)
```

Draw a filled colored rectangle with rounded edges and specific border width. Apply AA if available.

```
void nema_draw_quad_aa(float x0, float y0, float x1, float y1,
float x2, float y2, float x3, float y3, float border_width,
uint32_t color)
```

Draws a quadrilateral with specific border width. Apply AA if available. Only Convex quadrilaterals are supported.

Detailed Description

11.1.12.2 Function Documentation

```
void nema_draw_quad_aa ( float x0, float y0, float x1, float y1,
float x2, float y2, float x3, float y3, float border_width,
uint32_t color )
```

Draws a quadrilateral with specific border width. Apply AA if available. Only Convex quadrilaterals are supported.

Parameters	Description
x0	x coordinate at the first vertex of the quadrilateral
y0	y coordinate at the first vertex of the quadrilateral
x1	x coordinate at the second vertex of the quadrilateral
y1	y coordinate at the second vertex of the quadrilateral
x2	x coordinate at the third vertex of the quadrilateral
y2	y coordinate at the third vertex of the quadrilateral
x3	x coordinate at the fourth vertex of the quadrilateral
y3	y coordinate at the fourth vertex of the quadrilateral
border_width	trianquadrilateralgle's border width
color	color of the quadrilateral

```
void nema_draw_rounded_rect_aa ( float x, float y, float w, float
h, float r, float border_width, uint32_t rgba8888 )
```

Draw a colored rectangle with rounded edges and specific border width. Apply AA if available.

Parameters	Description
x	x coordinate of the upper left vertex of the rectangle
y	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle
r	corner radius
border_width	border width
rgba8888	rgba color of the rounded rectangle

```
void nema_draw_triangle_aa ( float x0, float y0, float x1, float
y1, float x2, float y2, float border_width, uint32_t color )
```

Draws a triangle with specific border width. Apply AA if available. Degenerated triangles have undefined behavior.

Parameters	Description
x0	x coordinate at the first vertex of the triangle
y0	y coordinate at the first vertex of the triangle
x1	x coordinate at the second vertex of the triangle
y1	y coordinate at the second vertex of the triangle
x2	x coordinate at the third vertex of the triangle
y2	y coordinate at the third vertex of the triangle
border_width	triangle's border width
color	color of the triangle

```
void nema_fill_rounded_rect_aa ( float x, float y, float w, float
h, float r, uint32_t rgba8888 )
```

Draw a filled colored rectangle with rounded edges and specific border width. Apply AA if available.

Parameters	Description
x	x coordinate of the upper left vertex of the rectangle
y	y coordinate at the upper left vertex of the rectangle
w	width of the rectangle
h	height of the rectangle

Parameters	Description
r	corner radius
rgba8888	rgba color of the rounded rectangle

```
void nema_fill_triangle_fan_f ( float * vertices, int num_vertices,
int stride, uint32_t rgba8888 )
```

Fill a traingle fan with color (float coordinates)

Parameters	Description
vertices	pointer to vertices coordinated (first x coordinate of vertex, then y coordinate of vertex)
num_vertices	number of vertices
stride	Distance between two vertices
rgba8888	Color to be used

```
void nema_fill_triangle_strip_f ( float * vertices, int
num_vertices, int stride, uint32_t rgba8888 )
```

Fill a triangle strip with color (float coordinates)

Parameters	Description
vertices	pointer to vertices coordinated (first x coordinate of vertex, then y coordinate of vertex)
num_vertices	number of vertices
stride	Distance between two vertices
rgba8888	Color to be used

11.1.13 nema_transitions.h

```
#include "nema_blender.h"
```

```
Enumerations enum nema_transition_t
```

```
NEMA_TRANS_LINEAR_H, NEMA_TRANS_CUBE_H, NEMA_TRANS_INNERCUBE_H,
NEMA_TRANS_STACK_H, NEMA_TRANS_LINEAR_V, NEMA_TRANS_CUBE_V,
NEMA_TRANS_INNERCUBE_V, NEMA_TRANS_STACK_V, NEMA_TRANS_FADE,
NEMA_TRANS_FADE_ZOOM, NEMA_TRANS_MAX, NEMA_TRANS_NONE
```

11.1.13.1 Functions

`void nema_transition(nema_transition_t effect, nema_tex_t initial, nema_tex_t final, uint32_t blending_mode, float step, int width, int height)`

Transition from 'initial' texture to 'final' texture. The transition is complete when 'step' is 0 or 1.

`void nema_transition_linear_hor(nema_tex_t left, nema_tex_t right, uint32_t blending_mode, float step, int width)`

Linear transition horizontally. When 'step' changes from zero to one, textures move from right to left, otherwise textures move from left to right. The transition is complete when 'step' is 0 or 1.

`void nema_transition_linear_ver(nema_tex_t up, nema_tex_t down, uint32_t blending_mode, float step, int height)`

Linear transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

`void nema_transition_cube_hor(nema_tex_t left, nema_tex_t right, uint32_t blending_mode, float step, int width, int height)`

Cubic (textures are mapped on the external faces of a cube) transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

`void nema_transition_cube_ver(nema_tex_t up, nema_tex_t down, uint32_t blending_mode, float step, int width, int height)`

Cube (textures are mapped on the external faces of a cube) transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

`void nema_transition_innercube_hor(nema_tex_t left, nema_tex_t right, uint32_t blending_mode, float step, int width, int height)`

Inner Cube (textures are mapped on the internal faces of a cube) transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

`void nema_transition_innercube_ver(nema_tex_t up, nema_tex_t down, uint32_t blending_mode, float step, int width, int height)`

Inner Cube (textures are mapped on the internal faces of a cube) transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition The transition is complete when 'step' is 0 or 1.

void nema_transition_stack_hor(nema_tex_t left, nema_tex_t right, float step, int width, int height)

Stack transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

void nema_transition_stack_ver(nema_tex_t up, nema_tex_t down, float step, int width, int height)

Stack transition vertically. When 'step' moves from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

void nema_transition_fade(nema_tex_t initial, nema_tex_t final, uint32_t blending_mode, float step, int width, int height)

Fade transition. Initial texture is being faded out, while final texture is being faded in. The transition is complete when 'step' is 0 or 1.

void nema_transition_fade_zoom(nema_tex_t initial, nema_tex_t final, uint32_t blending_mode, float step, int width, int height)

Fade-zoom transition. Initial texture is being zoomed and faded out, while final texture is being zoomed and faded in. The transition is complete when 'step' is 0 or 1.

11.1.13.2 Detailed Description

Enumeration Type Documentation

11.1.13.3 Function Documentation

void nema_transition (nema_transition_t effect, nema_tex_t initial, nema_tex_t final, uint32_t blending_mode, float step, int width, int height)

Transition from 'initial' texture to 'final' texture. The transition is complete when 'step' is 0 or 1.

Parameters	Description
effect	Transition effect
initial	Initial texture
final	Final texture
blending_mode	Blending mode
step	Transition step within [0.f , 1.f] range

Parameters	Description
width	Texture width
height	Texture height

```
void nema_transition_cube_hor ( nema_tex_t left, nema_tex_t right,
uint32_t blending_mode, float step, int width, int height )
```

Cubic (textures are mapped on the external faces of a cube) transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

Parameters	Description
left	Texture on the left side
right	Texture on the right side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_cube_ver ( nema_tex_t up, nema_tex_t down,
uint32_t blending_mode, float step, int width, int height )
```

Cube (textures are mapped on the external faces of a cube) transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

Parameters	Description
up	Texture on the top side
down	Texture on the bottom side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_fade ( nema_tex_t initial, nema_tex_t final,
uint32_t blending_mode, float step, int width, int height )
```

Fade transition. Initial texture is being faded out, while final texture is being faded in. The transition is complete when 'step' is 0 or 1.

Parameters	Description
left	Texture on the left side
right	Texture on the right side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_fade_zoom ( nema_tex_t initial, nema_tex_t
final, uint32_t blending_mode, float step, int width, int height )
```

Fade-zoom transition. Initial texture is being zoomed and faded out, while final texture is being zoomed and faded in. The transition is complete when 'step' is 0 or 1.

Parameters	Description
initial	Initial texture
final	Final texture
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_innercube_hor ( nema_tex_t left, nema_tex_t
right, uint32_t blending_mode, float step, int width, int height )
```

Inner Cube (textures are mapped on the internal faces of a cube) transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

Parameters	Description
left	Texture on the left side
right	Texture on the right side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range

Parameters	Description
width	Texture width
height	Texture height

```
void nema_transition_innercube_ver ( nema_tex_t up, nema_tex_t
down, uint32_t blending_mode, float step, int width, int height )
```

Inner Cube (textures are mapped on the internal faces of a cube) transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition The transition is complete when 'step' is 0 or 1.

Parameters	Description
up	Texture on the top side
down	Texture on the bottom side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_linear_hor ( nema_tex_t left, nema_tex_t
right, uint32_t blending_mode, float step, int width )
```

Linear transition horizontally. When 'step' changes from zero to one, textures move from right to left, otherwise textures move from left to right. The transition is complete when 'step' is 0 or 1.

Parameters	Description
left	Texture on the left side
right	Texture on the right side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width

```
void nema_transition_linear_ver ( nema_tex_t up, nema_tex_t down,
uint32_t blending_mode, float step, int height )
```

Linear transition vertically. When 'step' changes from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

Parameters	Description
up	Texture on the top side
down	Texture on the bottom side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
height	Texture height

```
void nema_transition_stack_hor ( nema_tex_t left, nema_tex_t right,
float step, int width, int height )
```

Stack transition horizontally. When 'step' changes from zero to one, textures move from left to right, otherwise textures move from right to left. The transition is complete when 'step' is 0 or 1.

Parameters	Description
up	Texture on the top side
down	Texture on the bottom side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

```
void nema_transition_stack_ver ( nema_tex_t up, nema_tex_t down,
float step, int width, int height )
```

Stack transition vertically. When 'step' moves from zero to one, textures move from top to bottom, otherwise textures move from bottom to top. The transition is complete when 'step' is 0 or 1.

Parameters	Description
up	Texture on the top side
down	Texture on the bottom side
blending_mode	Blending mode
step	Current step within [0.f , 1.f] range
width	Texture width
height	Texture height

11.2 Directories

Here is a list of all directories with brief descriptions:

11.2.1 File List

NemaGFX

nema_blender.h
nema_cmdlist.h
nema_easing.h
nema_error.h
nema_font.h
nema_graphics.h
nema_hal.h
nema_interpolators.h
nema_math.h
nema_matrix3x3.h
nema_matrix4x4.h
nema_provisional.h
nema_transitions.h

11.3 Data Structures

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

11.3.1 color_var_t

Data Structure

```
#include <nema_interpolators.h>
```

Data Fields

float r

Red

float g

Green

float b

Blue

float a

Alpha

Detailed Description

11.3.2 img_obj_t

Data Structure

```
#include <nema_graphics.h>
```

Data Fields

```
nema_buffer_t
bo uint16_t w
uint16_t h int
stride
uint32_t color
uint8_t format
uint8_t sampling_mode
```

Detailed Description

11.3.3 nema_buffer_t

Data Structure

```
#include <nema_hal.h>
```

Data Fields

int size

Size of buffer

int fd

File Descriptor of buffer

void * base_virt

Virtual address of buffer

uintptr_t base_phys

Physical address of buffer

Detailed Description

11.3.4 nema_cmdlist_t

Data Structure

```
#include <nema_cmdlist.h>
```

Data Fields

nema_buffer_t bo
int size

Number of entries in the command

list int offset

Points to the next address to write

uint32_t flags

Flags int32_t

submission_id

CL id to wait for struct

nema_cmdlist_t_ * next

Points to next command list

struct nema_cmdlist_t_ * root

Points to the head of the list

int sectors

Number of the sectors that the cl consists of int

sector_size

Size of each sector

uint32_t sector_id

Pointer to the current sector of the **cl int32_t**

internal_submitted_id

Submitted cl id by the CPU **int32_t**

internal_executed_id

Executed cl id by the GPU

Detailed Description

11.3.5 nema_font_range_t

Data Structure

```
#include <nema_font.h>
```

Data Fields

uint32_t first uint32_t last const nema_glyph_t * glyphs

Detailed Description

11.3.6 nema_font_t

Data Structure

```
#include <nema_font.h>
```

Data Fields

```
nema_buffer_t bo const nema_font_range_t * ranges const
int bitmap_size const
uint8_t * bitmap
uint32_t flags uint8_t xAdvance
uint8_t yAdvance uint8_t max_ascender
uint8_t bpp const nema_kern_pair_t * kern_pairs const
nema_glyph_indexed_t * indexed_glyphs
```

Detailed Description

11.3.7 nema_glyph_indexed_t

Data Structure

```
#include <nema_font.h>
```

Data Fields

```
int
bitmapOffset
uint8_t width
uint8_t
xAdvance
int8_t xOffset
int8_t yOffset
int id
```

Detailed Description

11.3.8 nema_glyph_t

Data Structure

```
#include <nema_font.h>
```

Data Fields

```
uint32_t
bitmapOffset
uint8_t width
uint8_t xAdvance
int8_t xOffset
int8_t yOffset
uint32_t
kern_offset
uint8_t
kern_length
```

Detailed Description

11.3.9 nema_kern_pair_t

Data Structure

```
#include <nema_font.h>
```

Data Fields

```
uint32_t left
```

Neighbor character to the left of the current one (Unicode value) **int8_t x_offset**

Kerning offset (horizontally)

11.3.9.1 Detailed Description

11.3.10 nema_ringbuffer_t

Data Structure

Data Fields

```
nema_buffer_t bo
int offset
int last_submission_id
```

Detailed Description

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 512. 879.2850

A-SOCAPG-UMGA02EN v1.0
October 2025