

APPLICATION NOTE

Nema GFX API Debugging

A-SOCAPG-ANGA03EN 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

Revision	Date	Description
1.0	December 2025	Initial Release.

Reference Documents

Document ID	Description
A-SOCAPG-UMGA01EN	Nema DC API Library User's Manual
A-SOCAPG-UMGA02EN	Nema GFX API Library User's Manual
A-SOCAPG-UMGA03EN	Nema GFX Extension Vector Graphics User's Manual
A-SOCAPG-ANGA01EN	Nema DC MiP Panels Configuration Application Note
A-SOCAPG-SGGA01EN	Nema Pixpresso Starting Guide
A-SOCAPG-SGGA02EN	Nema GFX Ext TSVG Supported Elements List 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

1. Preface	5
1.1 About this Manual	5
1.2 Target Audience	5
1.3 Related Documents	5
2. Generic Workflow	6
2.1 Nothing is Drawn	7
2.2 Frame is Different than Expected	8
2.3 Memory Leaks	9
2.4 Performance Issues	9
3. Breakpoints	11
4. GPU Status	13

SECTION

1

Preface

1.1 About this Manual

This manual provides the necessary guidelines in order to perform debugging on applications that make use of Nema GFX.

1.2 Target Audience

The current document is intended to be used by engineers and developers who develop graphics applications using Nema GFX and therefore they are familiar with Nema GFX up to a certain degree.

1.3 Related Documents

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

- Nema GFX User Manual
- Nema Pico Graphics Processing Unit Software User Manual
- Nema Pico Graphics Processing Unit Hardware User Manual

SECTION

2

Generic Workflow

Potential bugs that can be found in Nema GFX based applications can be divided into the following categories:

1. Nothing is drawn (the display is not updated)
2. Frame is different than expected (the drawn frame is not the expected one)
3. Memory leaks (allocated memory that never gets deallocated)
4. Performance issues (the performance is lower than expected)

Resolving each case can be performed by applying a set of actions that attempt to identify and resolve the target malfunction. These actions can be classified as:

- **[M]:** Mandatory
- **[A]:** Advisory
- **[N]:** Note

It must be noted that it is impossible to resolve the bug without performing the actions marked as Mandatory. Advisory actions refer to actions that have high potential of resolving the bug, provided that the mandatory actions have been performed. Finally, actions marked as Note, are generic advices that apply to each category. Although they do not comprise solid solutions, it is highly recommended that the users take them into consideration in order to minimize the chances of inserting bugs within their applications.

When creating a Nema GFX based application, it needs to be taken into account that the workflow follows a specific order. For instance, memory allocation precedes memory deallocation. Having this in mind, the following paragraphs present the actions that need to be applied, in the right order (from the beginning till the termination of the application).

2.1 Nothing is Drawn

1. **[N]** A DC typically serves as a client and the GPU as a server. The GPU draws the FB and the DC reads it and provides the appropriate signaling to drive a display.
2. **[M]** `nema_init()` is the first Nema GFX function that is called.
3. **[M]** `nema_init()` should return with no errors (return 0)
4. **[M]** A FB should be allocated to an appropriate memory region. FB should be able to be written by the GPU and read by the DC.
5. **[N]** `nema_create_buffer()` can be used to allocate a FB.
6. **[N]** Multiple FBs (double/triple buffering) can be used to avoid visual artifacts, in which case synchronisation between DC and GPU must be taken care of.
7. **[N]** All NemaGFX functions that define the rendering pipeline (e.g., anything relative to setting blending mode, textures, fill/blit operations) operate on the bound CL. Nothing is drawn unless the CL is submitted for execution (`nema_cl_submit()`).
8. **[M]** A CL must be created by calling `nema_cl_create()`.
9. **[M]** All GPU context should be defined by the current CL, e.g., a CL must be agnostic of the current GPU state.
10. **[N]** A valid CL should contain at least the following: Clipping Rectangle, destination TEX (FB), blending mode, source TEX (in case of blitting), drawing operation.
11. **[A]** After the `nema_init()`, a CL should be bound at all times (`nema_cl_bind()`).
12. **[N]** Calling `nema_cl_bind()` implicitly unbinds the previously bound CL.
13. **[A]** Rewind CL before using it (`nema_cl_rewind()`).
14. **[M]** A destination surface must be bound before defining rendering commands (`nema_bind_dst()`). In most cases, this destination surface should correspond to an allocated FB.
15. **[M]** Clipping rectangle must be set within the boundaries of the FB (`nema_set_clip(x,y,w,h)`), and must fulfill the following conditions:
 - a. $x \geq 0$ and $y \geq 0$
 - b. $x+w \leq \text{fb.RESX}$ and $y+h \leq \text{fb.RESY}$
16. **[M]** A blending mode must be set before doing any drawing operations (blit or fill) (`nema_set_blend_blit()/nema_set_blend_fill()`).
17. **[M]** When blitting, a Source TEX containing an allocated texture must be bound (`nema_bind_src()`) before calling the corresponding rendering commands.
18. **[N]** PRIMs are drawn using the respective function calls (e.g. `nema_blit()`, `nema_fill_rect()`, `nema_fill_tri()` etc.).

19. **[M]** When using triangle culling, call **nema_tri_cull()** right before calling the rendering operation and call **nema_tri_cull(NEMA_CULL_NONE)** immediately afterwards. A misplaced **nema_tri_cull()** call can lead to some geometry PRIMs not to be drawn or to be drawn partially.
20. **[N]** Multiple PRIMs can be drawn within the same CL.
21. **[M]** For any drawing operation to take place, the assembled CL must be submitted for execution (**nema_cl_submit()**).
22. **[M]** Before deleting or reusing a CL, the GPU must have finished its execution (**nema_cl_wait()**).
23. **[N]** Once the CL is executed, it can be resubmitted (**nema_cl_submit()**), resetted and reused (**nema_cl_rewind()**), or destroyed (**nema_cl_destroyed()**).

2.2 Frame is Different than Expected

1. **[M]** FB parameters must be set properly (address, width, height, stride, color format).
2. **[N]** The correct FB color format should be used which should match the DC format.
3. **[M]** Clipping rectangle must be set within the boundaries of the FB (**nema_set_clip(x,y,w,h)**), and must fulfill the following conditions:
 - a. $x \geq 0$ and $y \geq 0$
 - b. $x+w \leq \text{fb.RESX}$ and $y+h \leq \text{fb.RESY}$
4. **[M]** The proper blending mode should be set depending on the desired rendering operation (fill or blit).
5. **[M]** In case of blitting, the source TEX containing the texture should be bound before rendering (**nema_bind_src()**).
6. **[M]** Source TEX parameters should be set correctly (address, width, height, stride, color format, wrapping mode).
7. **[M]** The appropriate rendering function should be used for the desired drawing operation (e.g., **nema_blit()**, **nema_fill_rect()**, **nema_blit_rect_fit()**).
8. **[M]** The CL must have finished its execution (**nema_cl_wait()**) before reusing or destroying it .
9. **[M]** TSc TEX and FB formats should be used only when available on the hardware implementation (GPU).
10. **[M]** Not all color formats are valid for FB. See Nema Pico Software Manual for more details.
11. **[M]** When using depth, **nema_bind_depth_buffer()** and **nema_set_depth()** must be called before calling **nema_enable_depth()**.

12. **[A]** When using depth, call `nema_enable_depth(1)` right before calling the rendering operation and call `nema_enable_depth(0)` immediately afterwards. A misplaced `nema_enable_depth()` call can lead to various artifacts.
13. **[M]** When using gradient, `nema_set_gradient()` should be called before calling `nema_enable_gradient()`.
14. **[A]** When using gradient, call `nema_enable_gradient(1)` right before calling the rendering operation and call `nema_enable_gradient(0)` immediately afterwards. A misplaced `nema_enable_gradient()` call can lead to various artifacts.
15. **[M]** When using triangle culling, call `nema_tri_cull()` right before calling the rendering operation and call `nema_tri_cull(NEMA_CULL_NONE)` immediately afterwards. A misplaced `nema_tri_cull()` call can lead to some geometry PRIMs not to be drawn or to be drawn partially.
16. **[M]** Culling must be used only when rendering triangle PRIMs. Having culling enabled when rendering other PRIMs (rectangles, quadrilaterals) results to undefined behavior.

2.3 Memory Leaks

1. **[A]** Delete any allocated CL if not needed anymore. (`nema_cl_destroy()`)
2. **[A]** Delete any allocated BO if are not needed anymore (`nema_buffer_destroy()`)
3. **[A]** Avoid generic (not useful to the GPU) memory allocations within the graphics memory. Deallocate memory accordingly.

2.4 Performance Issues

1. **[A]** Identify the bottleneck (CPU, GPU, memory bandwidth etc) and optimize accordingly.
2. **[N]** A GPU is orders of magnitude faster than a CPU when rendering graphics. Use hardware graphics acceleration when possible to achieve a better performance.
3. **[A]** Offload the CPU by reusing assembled CL when possible. The same CL can be submitted multiple times for execution.
4. **[A]** Multiple CLs can be submitted at a time. In this case, just wait for the execution of the last submitted one.
5. **[A]** Use a pair of CLs for pipelining, prepare a new CL while the submitted one is being executed by the GPU.
6. **[A]** Draw only needed geometry to minimize the number of PRIMs.

7. **[A]** Adjust Clip Rectangle to the area to be drawn.
8. **[A]** Use triangle culling when drawing 3D geometries to minimize the number of PRIMs to draw.
9. **[A]** Use dirty regions to partially update the frame content - use clipping accordingly.
10. **[A]** Minimize overdraws. Overdraws areas can be inspected for debugging using the `nema_debug_overdraws()` call. This function disables gradients and textures and forces blending modes to `NEMA_BL_ADD`. By doing this, the user can inspect whether different regions are being drawn one of top of the other, detecting possible overdrawn regions that could have an impact on the performance.
11. **[N]** Excessive use of perspective transformations (`nema_matrix3x3`, `nema_matrix4x4`) can increase the CPU load significantly.
12. **[A]** Using appropriate graphics assets (for example, by minimizing texture resolution and selecting a suitable format) can reduce memory usage and bandwidth.
13. **[A]** In GPU-bound applications, prefer point-sampling instead of bilinear texture filtering, when the rendering quality is acceptable.

SECTION

3

Breakpoints

In order to further debug applications, NEMA| GFX implements support for breakpoints. Breakpoints can be inserted at different positions within a CL in order to pause its execution. When the GPU reaches a breakpoint, it signals the CPU about it and pauses its execution. The GPU will continue its operation when the CPU instructs it so. While the GPU is paused, the user can have an overview of the state of the GPU (by calling respective functions) and the rendering process. In order to use the breakpoints, NEMA| GFX offers the following functions:

- **void nema_brk_enable(void):** enables the breakpoints.
- **void nema_brk_disable(void):** disables the breakpoints.
- **int nema_brk_add(void):** adds a new breakpoint to the CL and returns its id (positive value).
- **int nema_brk_wait(int brk_id):** the CPU waits for the GPU to reach the breakpoint with id equal to **brk_id**. Setting the **brk_id = 0** will make the CPU wait until the GPU encounters the next breakpoint, regardless of its id. The function returns the id of the encountered breakpoint.
- **void nema_brk_continue(void):** instruct the GPU to resume execution.

The following example illustrates the usage of breakpoints. The GPU is programmed to draw three rectangles filled with different colors (red, green and blue). A breakpoint is placed after each respective function call. The assembled CL is submitted to the GPU. The CPU will wait for the GPU to reach the the first breakpoint (brk1). While the GPU is paused, the id of the breakpoint along with the current color value are printed. Afterwards the CPU instructs the GPU to continue its operation. It then waits for the GPU to reach the third breakpoint (brk3). The second breakpoint is intentionally ignored. When brk3 is encountered, its id and the drawing color will be printed. The CPU instructs again the GPU to continue and the program finishes its execution.

```
void fill_rectangles(void) {  
    nema_cmdlist_t cl = nema_cl_create();  
    nema_cl_bind(&cl);
```

```
nema_cl_branch(&context_cl);
    //Enable breakpoints
nema_brk_enable();
    nema_set_blend_fill( NEMA_BL_SRC );
    nema_fill_rect(10, 10, 20, 20, nema_rgba(255, 0, 0, 255));
    //Add first breakpoint
int brk1 = nema_brk_add();
    nema_fill_rect(40, 10, 20, 20, nema_rgba(0, 255, 0, 255));
    //Add second breakpoint
int brk2 = nema_brk_add();
    nema_fill_rect(70, 10, 20, 20, nema_rgba(0, 0, 255, 255));
    //Add third breakpoint

    int brk3 = nema_brk_add();
    nema_cl_unbind();
    //Submit Command List for execution
nema_cl_submit(&cl);    int brk;
    //Wait for the GPU to reach any breakpoint
    brk = nema_brk_wait(0);
    printf("Breakpoint %d: Texture color register value: %x\n", brk,
nema_reg_read(NEMA_DRAW_COLOR));
    //Let the GPU resume execution
nema_brk_continue();
    //Wait for the GPU to reach the third breakpoint (ignore any other
breakpoints)
    brk = nema_brk_wait(brk3);
    printf("Breakpoint %d: Texture color register value: %x\n", brk,
nema_reg_read(NEMA_DRAW_COLOR));
    //Let the GPU resume execution
nema_brk_continue();
    //Wait for the Command List to finish execution
    nema_cl_wait(&cl);
nema_cl_destroy(&cl);
    //Disable breakpoints
nema_brk_disable();
}
```

SECTION

4

GPU Status

Nema GFX is able to communicate with Nema GPUs by writing or reading the GPU registers as well as by creating command lists and submitting them for execution to Nema GPU. Writing invalid values in such structs that are read by the GPU (registers and command lists) could potentially make the GPU stall. For instance, when the framebuffer address has not been set at all, the GPU will attempt to write non accessible memory, and will stall. In this case (GPU stall), the necessary information about which subcomponent of the GPU encountered a malfunction can be found in the GPU status register (0x0fc). When the GPU is idle, it contains the zero value ('0'), otherwise it contains a value different than zero. This value denotes the GPU subcomponents that are currently busy. It can be decoded as follows:

Bit	Function
31	System busy
30	Memory System Unit busy
29	Command List bus busy
28	Command List Processor busy
27-24	Rasterizer Unit busy
23-13	Reserved
12	Render Output Unit busy
11-9	Reserved
8	Texture Map Unit busy
7-5	Reserved
4	Fragment Processor busy
3-1	Reserved
0	Fragment Processor Scheduler busy



© 2025 Ambiq Micro, Inc. All rights reserved.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730

www.ambiq.com

sales@ambiq.com

+1 512.879.2850

A-SOCAPG-ANGA03EN v1.0

December 2025