GR5526 GPU开发者指南

版本： **1.0**

发布日期： **2023-01-10**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田保税区腾飞工业大厦B座12-13层

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档介绍GR5526系列芯片中GPU（Graphics Processing Unit）模块的基础信息，结合相关示例工程演示如何在GR5526系列芯片上应用GPU模块实现图像处理，旨在帮助开发者了解并应用GR5526 GPU模块。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 开发爱好者
- 文档工程师

版本说明

本文档为第1次发布，对应的产品系列为GR5526。

修订记录

版本	日期	修订内容
1.0	2023-01-10	首次发布

目录

前言 I

1 简介 1

1.1 功能特点 1
1.2 GPU/MCU绘图能力比较 1
1.3 GPU渲染数据流 2
1.4 GPU驱动架构 3
1.4.1 GPU驱动库 3
1.4.2 目录结构 4

2 基本概念 6

2.1 计算机图形学 6
2.2 几何原语 7
2.3 GPU图形管道 7
2.3.1 配置寄存器文件 8
2.3.2 命令列表处理单元 8
2.3.3 光栅化 9
2.3.4 纹理贴图单元 9
2.3.5 片段处理核心 9
2.3.6 渲染输出单元 9
2.4 颜色格式 10
2.5 帧缓冲区 10
2.5.1 压缩 11
2.6 内存使用 11

3 开发 12

3.1 命令列表 12
3.1.1 环形缓冲区 12
3.1.2 创建 12
3.1.3 绑定 13
3.1.4 解除绑定 13
3.1.5 提交 13
3.1.6 等待 13
3.1.7 销毁 13
3.1.8 示例 13
3.2 纹理 14
3.2.1 纹理槽 14
3.2.2 纹理绑定 15
3.2.3 示例 15
3.3 剪切区和脏区 17

3.3.1 剪切区.....17
3.3.2 脏区.....17
3.4 混合.....18
3.4.1 预定义的混合模式.....18
3.4.2 自定义模式.....19
3.4.3 附加操作.....21
3.5 基础绘图.....22
3.6 字体.....24
3.6.1 字距调整.....25
3.7 GPU驱动函数.....26
3.7.1 hal_gfx_blender.h.....26
3.7.1.1 宏定义.....26
3.7.1.2 hal_gfx_blending_mode.....27
3.7.1.3 hal_gfx_set_blend.....28
3.7.1.4 hal_gfx_set_blend_fill.....28
3.7.1.5 hal_gfx_set_blend_fill_compose.....28
3.7.1.6 hal_gfx_set_blend_blit.....28
3.7.1.7 hal_gfx_set_blend_blit_compose.....29
3.7.1.8 hal_gfx_set_const_color.....29
3.7.1.9 hal_gfx_set_src_color_key.....29
3.7.1.10 hal_gfx_set_dst_color_key.....29
3.7.1.11 hal_gfx_debug_overdraws.....30
3.7.2 hal_gfx_cmdlist.h.....30
3.7.2.1 hal_gfx_cl_create_prealloc.....30
3.7.2.2 hal_gfx_cl_create_sized.....30
3.7.2.3 hal_gfx_cl_create.....30
3.7.2.4 hal_gfx_cl_destroy.....31
3.7.2.5 hal_gfx_cl_rewind.....31
3.7.2.6 hal_gfx_cl_bind.....31
3.7.2.7 hal_gfx_cl_bind_circular.....31
3.7.2.8 hal_gfx_cl_unbind.....32
3.7.2.9 hal_gfx_cl_get_bound.....32
3.7.2.10 hal_gfx_cl_submit_no_irq.....32
3.7.2.11 hal_gfx_cl_submit.....32
3.7.2.12 hal_gfx_cl_wait.....33
3.7.2.13 hal_gfx_cl_add_cmd.....33
3.7.2.14 hal_gfx_cl_add_multiple_cmds.....33
3.7.2.15 hal_gfx_cl_get_space.....33
3.7.2.16 hal_gfx_cl_branch.....34
3.7.2.17 hal_gfx_cl_jump.....34
3.7.2.18 hal_gfx_cl_return.....34
3.7.2.19 hal_gfx_cl_almost_full.....34
3.7.2.20 hal_gfx_cl_enough_space.....35

3.7.3 hal_gfx_easing.h....35
3.7.3.1 hal_gfx_ez_linear....35
3.7.3.2 hal_gfx_ez_quad_in....35
3.7.3.3 hal_gfx_ez_quad_out....35
3.7.3.4 hal_gfx_ez_quad_in_out....36
3.7.3.5 hal_gfx_ez_cub_in....36
3.7.3.6 hal_gfx_ez_cub_out....36
3.7.3.7 hal_gfx_ez_cub_in_out....36
3.7.3.8 hal_gfx_ez_quar_in....37
3.7.3.9 hal_gfx_ez_quar_out....37
3.7.3.10 hal_gfx_ez_quar_in_out....37
3.7.3.11 hal_gfx_ez_quin_in....37
3.7.3.12 hal_gfx_ez_quin_out....38
3.7.3.13 hal_gfx_ez_quin_in_out....38
3.7.3.14 hal_gfx_ez_sin_in....38
3.7.3.15 hal_gfx_ez_sin_out....38
3.7.3.16 hal_gfx_ez_sin_in_out....39
3.7.3.17 hal_gfx_ez_circ_in....39
3.7.3.18 hal_gfx_ez_circ_out....39
3.7.3.19 hal_gfx_ez_circ_in_out....39
3.7.3.20 hal_gfx_ez_exp_in....40
3.7.3.21 hal_gfx_ez_exp_out....40
3.7.3.22 hal_gfx_ez_exp_in_out....40
3.7.3.23 hal_gfx_ez_elast_in....40
3.7.3.24 hal_gfx_ez_elast_out....41
3.7.3.25 hal_gfx_ez_elast_in_out....41
3.7.3.26 hal_gfx_ez_back_in....41
3.7.3.27 hal_gfx_ez_back_out....41
3.7.3.28 hal_gfx_ez_back_in_out....42
3.7.3.29 hal_gfx_ez_bounce_out....42
3.7.3.30 hal_gfx_ez_bounce_in....42
3.7.3.31 hal_gfx_ez_bounce_in_out....42
3.7.3.32 hal_gfx_ez....43
3.7.4 hal_gfx_error.h....43
3.7.4.1 宏定义....43
3.7.4.2 hal_gfx_get_error....43
3.7.5 hal_gfx_font.h....44
3.7.5.1 宏定义....44
3.7.5.2 hal_gfx_bind_font....44
3.7.5.3 hal_gfx_string_get_bbox....44
3.7.5.4 hal_gfx_print....45
3.7.5.5 hal_gfx_print_to_position....45
3.7.6 hal_gfx_graphics.h....46
3.7.6.1 宏定义....46

3.7.6.2 枚举类型.....48
3.7.6.3 hal_gfx_checkGPUPresence.....48
3.7.6.4 hal_gfx_bind_tex.....48
3.7.6.5 hal_gfx_set_tex_color.....49
3.7.6.6 hal_gfx_set_const_reg.....49
3.7.6.7 hal_gfx_set_clip.....49
3.7.6.8 hal_gfx_enable_gradient.....49
3.7.6.9 hal_gfx_enable_depth.....50
3.7.6.10 hal_gfx_enable_aa.....50
3.7.6.11 hal_gfx_get_dirty_region.....50
3.7.6.12 hal_gfx_clear_dirty_region.....51
3.7.6.13 hal_gfx_tri_cull.....51
3.7.6.14 hal_gfx_format_size.....51
3.7.6.15 hal_gfx_stride_size.....51
3.7.6.16 hal_gfx_texture_size.....52
3.7.6.17 hal_gfx_rgba.....52
3.7.6.18 hal_gfx_premultiply_rgba.....52
3.7.6.19 hal_gfx_init.....53
3.7.6.20 hal_gfx_bind_src_tex.....53
3.7.6.21 hal_gfx_bind_src2_tex.....53
3.7.6.22 hal_gfx_bind_dst_tex.....54
3.7.6.23 hal_gfx_bind_depth_buffer.....54
3.7.6.24 hal_gfx_clear.....54
3.7.6.25 hal_gfx_clear_depth.....55
3.7.6.26 hal_gfx_draw_line.....55
3.7.6.27 hal_gfx_draw_line_aa.....55
3.7.6.28 hal_gfx_draw_circle.....56
3.7.6.29 hal_gfx_draw_circle_aa.....56
3.7.6.30 hal_gfx_draw_rounded_rect.....56
3.7.6.31 hal_gfx_draw_rect.....57
3.7.6.32 hal_gfx_fill_circle.....57
3.7.6.33 hal_gfx_fill_circle_aa.....57
3.7.6.34 hal_gfx_fill_triangle.....57
3.7.6.35 hal_gfx_fill_rounded_rect.....58
3.7.6.36 hal_gfx_fill_rect.....58
3.7.6.37 hal_gfx_fill_quad.....59
3.7.6.38 hal_gfx_fill_rect_f.....59
3.7.6.39 hal_gfx_fill_quad_f.....59
3.7.6.40 hal_gfx_fill_triangle_f.....60
3.7.6.41 hal_gfx_blit.....60
3.7.6.42 hal_gfx_blit_rounded.....60
3.7.6.43 hal_gfx_blit_rect.....61
3.7.6.44 hal_gfx_blit_subrect.....61
3.7.6.45 hal_gfx_blit_rect_fit.....61

3.7.6.46 hal_gfx_blit_subrect_fit....62
3.7.6.47 hal_gfx_blit_rotate_pivot....62
3.7.6.48 hal_gfx_blit_rotate_pivot_scale....62
3.7.6.49 hal_gfx_blit_rotate....63
3.7.6.50 hal_gfx_blit_rotate_partial....63
3.7.6.51 hal_gfx_blit_tri_fit....63
3.7.6.52 hal_gfx_blit_tri_uv....64
3.7.6.53 hal_gfx_blit_quad_fit....65
3.7.6.54 hal_gfx_blit_subrect_quad_fit....65
3.7.7 hal_gfx_hal.h....66
3.7.7.1 hal_gfx_sys_init....66
3.7.7.2 hal_gfx_wait_irq....66
3.7.7.3 hal_gfx_wait_irq_cl....66
3.7.7.4 hal_gfx_wait_irq_brk....66
3.7.7.5 hal_gfx_reg_read....67
3.7.7.6 hal_gfx_reg_write....67
3.7.7.7 hal_gfx_buffer_create....67
3.7.7.8 hal_gfx_buffer_create_pool....67
3.7.7.9 hal_gfx_buffer_map....68
3.7.7.10 hal_gfx_buffer_unmap....68
3.7.7.11 hal_gfx_buffer_destroy....68
3.7.7.12 hal_gfx_buffer_phys....68
3.7.7.13 hal_gfx_buffer_flush....69
3.7.7.14 hal_gfx_host_malloc....69
3.7.7.15 hal_gfx_host_free....69
3.7.7.16 hal_gfx_rb_init....69
3.7.7.17 hal_gfx_mutex_lock....70
3.7.7.18 hal_gfx_mutex_unlock....70
3.7.8 hal_gfx_interpolators.h....70
3.7.8.1 hal_gfx_interpolate_rect_colors....70
3.7.8.2 hal_gfx_interpolate_tri_colors....71
3.7.8.3 hal_gfx_interpolate_tri_depth....71
3.7.8.4 hal_gfx_interpolate_tx_ty....71
3.7.9 hal_gfx_math.h....72
3.7.9.1 宏定义....72
3.7.9.2 hal_gfx_sin....73
3.7.9.3 hal_gfx_cos....73
3.7.9.4 hal_gfx_tan....73
3.7.9.5 hal_gfx_pow....74
3.7.9.6 hal_gfx_sqrt....74
3.7.9.7 hal_gfx_atan....74
3.7.9.8 hal_gfx_f2fx....74
3.7.10 hal_gfx_transitions.h....75
3.7.10.1 枚举类型....75

3.7.10.2 hal_gfx_transition....75
3.7.10.3 hal_gfx_transition_linear_hor....76
3.7.10.4 hal_gfx_transition_linear_ver....76
3.7.10.5 hal_gfx_transition_cube_hor....76
3.7.10.6 hal_gfx_transition_cube_ver....77
3.7.10.7 hal_gfx_transition_innercube_hor....77
3.7.10.8 hal_gfx_transition_innercube_ver....77
3.7.10.9 hal_gfx_transition_stack_hor....78
3.7.10.10 hal_gfx_transition_stack_ver....78
3.7.10.11 hal_gfx_transition_fade....78
3.7.10.12 hal_gfx_transition_fade_zoom....79
3.7.11 hal_gfx_matrix4x4.h....79
3.7.11.1 hal_gfx_mat4x4_load_identity....79
3.7.11.2 hal_gfx_mat4x4_mul....79
3.7.11.3 hal_gfx_mat4x4_mul_vec....80
3.7.11.4 hal_gfx_mat4x4_translate....80
3.7.11.5 hal_gfx_mat4x4_scale....80
3.7.11.6 hal_gfx_mat4x4_rotate_X....81
3.7.11.7 hal_gfx_mat4x4_rotate_Y....81
3.7.11.8 hal_gfx_mat4x4_rotate_Z....81
3.7.11.9 hal_gfx_mat4x4_load_perspective....81
3.7.11.10 hal_gfx_mat4x4_load_ortho....82
3.7.11.11 hal_gfx_mat4x4_load_ortho_2d....82
3.7.11.12 hal_gfx_mat4x4_obj_to_win_coords....83
4 教程....84
4.1 GPU应用典型代码结构....84
4.2 控制宏参数....85
4.2.1 graphics_defs.h....85
4.2.2 graphics_sys_defs.h....85
4.3 示例工程graphics_animation_effects....86
4.3.1 工程目录....86
4.3.2 应用代码介绍....86
4.3.2.1 外设初始化与缓冲区配置....87
4.3.2.2 渲染任务与屏幕刷新....89
5 常见问题....92
5.1 GPU初始化失败或无法访问GPU寄存器....92
5.2 使用GPU处理渲染任务，屏幕无显示或显示不正确....92
5.3 使用GPU处理纹理图像，屏幕图像失真或FPS低....93
5.4 如何支持GPU应用实现低功耗管理....93
6 术语和缩略语....95

1 简介

图形处理器（Graphics Processing Unit，GPU），又称显示核心、视觉处理器，是一种专用于图像和图形相关运算工作的微处理器。GR5526配备的GPU模块，以极小的硅片、功耗预算，为用户界面带来高质量的图形效果，适用于具有低成本和超低功耗要求的物联网平台、可穿戴和嵌入式设备。开发人员能够以更低的成本，为功率、内存区域受限的物联网设备，创建具有超长电池寿命且引人注目的图形用户界面（Graphical User Interface，GUI）和软件应用程序。

1.1 功能特点

- 硬件组件：具有可编程着色器引擎、基于DMA可减少CPU开销的命令列表、原始光栅化器、纹理映射单元、混合单元
- 绘制基元：像素/线条图、填充矩形、三角形（Gouraud Shaded）、四边形
- 颜色格式：32位RGBA8888/BGRA8888/ABGR8888、24位RGB、16位RGBA5551/RGB565、8位A8/L8/RGB332、TSC™等
- 压缩方案：TSC™4（每像素4位）、TSC™6 / TSC™6a（每像素6位，具备/不局别Alpha通道）
- 图像变换：纹理映射、点采样、双线性过滤、位块传输、任意角度旋转、镜像、拉伸（独立于x和y轴）、源和/或目标颜色键控、即时格式转换、2.5D透视投影
- 文本渲染支持：位图抗锯齿A1/A2/A4/A8、字体紧缩、Unicode（UTF8）
- 混合支持：完全可编程的Alpha混合模式（源和目标）、源/目标颜色键控
- 抗锯齿：8 x MSAA（Multi-Sampling Anti-Aliasing，多重采样抗锯齿）、每条边的四边形抗锯齿、每条边的三角形抗锯齿、抗锯齿粗线、抗锯齿圆

1.2 GPU/MCU绘图能力比较

GPU与MCU绘图能力对比结果如下表所示：

表 1-1 GPU/MCU绘图能力对比

特点	能力	GPU	MCU/DMA
基础绘制	像素/线条图	快速	较慢
	填充矩形	快速	较慢
	三角形/四边形	快速	较慢
图像变换	纹理映射	优良	无
	Alpha混合/位块传输	优良	困难/慢
	镜像	优良	无
	拉伸	优良	无
	点采样	优良	无
	双线性过滤	优良	无

特点	能力	GPU	MCU/DMA
	任意角度旋转	优良	无
增强效果	颜色格式	多种	少
	压缩/解压缩	支持	无
	抗锯齿	支持	无
	2.5D透视投影	支持	无

1.3 GPU渲染数据流

GPU渲染数据流如下图所示：

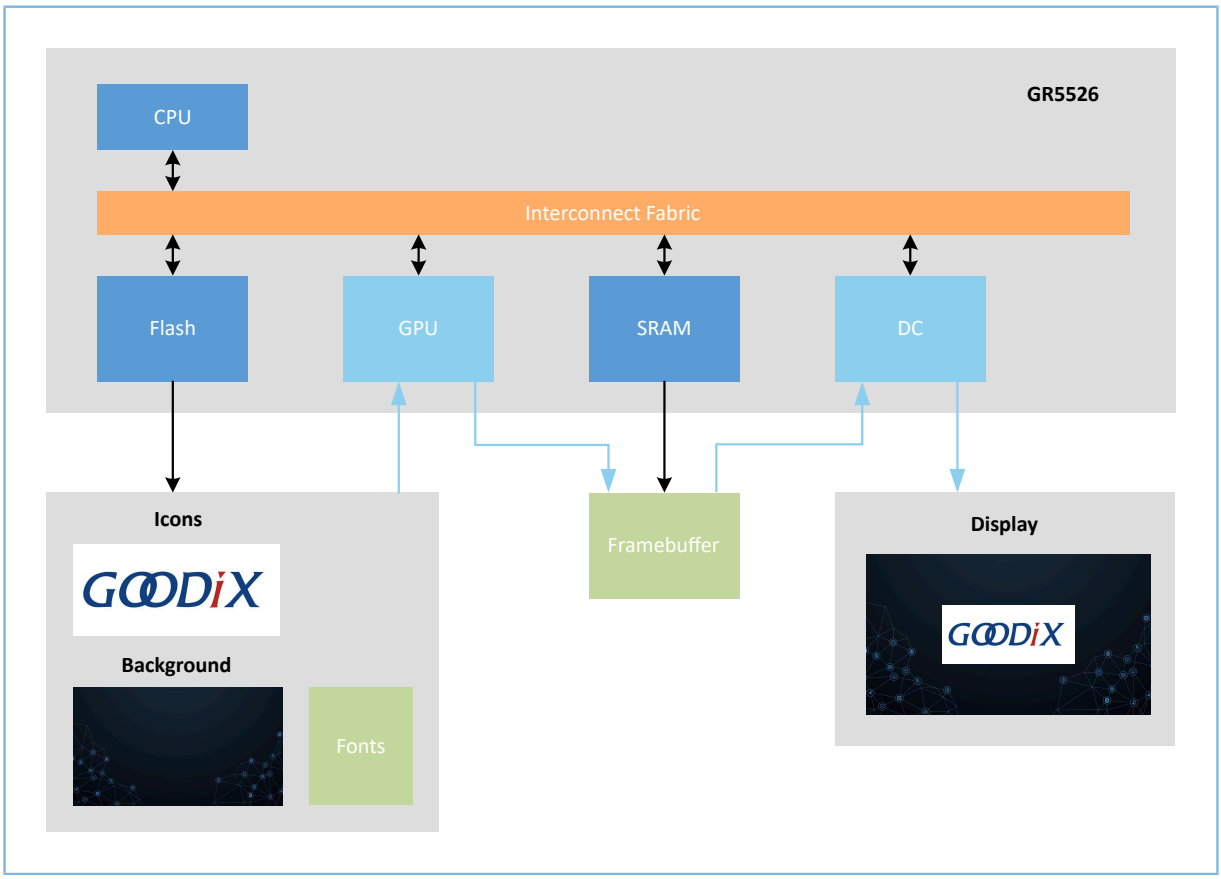


图 1-1 GPU渲染数据流

图片（纹理）素材通常存放于Flash中，运行过程中也可以将图片（纹理）素材拷贝至RAM（SRAM或者PSRAM）区域从而提高传输效率。在GPU渲染过程中，会根据图片（纹理）素材地址加载对应的图片（纹理）素材。Framebuffer通常位于RAM（SRAM或者PSRAM）区域。GPU会将最终渲染结果输出至Framebuffer中，随后可通过显示控制器（Display Controller，DC）或其他传输接口将渲染结果输出至屏幕上。

1.4 GPU驱动架构

1.4.1 GPU驱动库

GPU驱动库（GFX库）提供软件抽象层，用于轻松高效地组织和使用绘图命令。GFX库占用空间小、高效且没有任何外部依赖性。通过GFX库使用结构复杂的GPU，开发者能够以最小的CPU/MCU开销和功耗实现出色的图像处理性能。

GFX库包含一组更高级别的调用，为应用程序形成一个完整独立的图形API。此API能够执行从简单的线条、三角形和四边形到更复杂的绘制操作，如块状和透视纹理映射。

GFX库建立在模块化架构之上。开发者可直接使用硬件通信、同步和基本原语绘图的GFX架构最底层（GFX HAL）。轻量的硬件抽象层允许底层硬件的快速集成。GFX库既可以作为独立的绘图API使用，也可以作为第三方GUI框架的绘制层用于绘制加速。

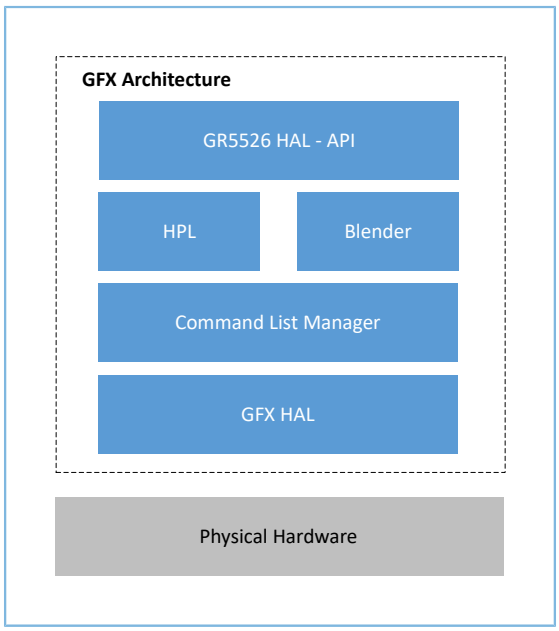


图 1-2 GFX库架构

- GFX库硬件抽象层（GFX HAL）：包括一些用于与硬件进行基本接口的API接口，例如寄存器访问、中断处理等。
- 命令列表管理器（Command List Manager）：为创建、组织和发布命令列表提供相应的API。
- 硬件编程层（HPL）：一组辅助函数，用于组装对GPU进行编程的命令。这些命令将被写入GPU的配置寄存器文件（对GPU的子模块进行编程）。
- 混合器（Blender）模块：对可编程处理核心进行编程。为核心创建二进制可执行文件（对应GPU支持的各种混合模式）。
- GR5526 SDK硬件抽象层（GR5526 HAL-API）：提供API用于绘制几何图元（线、三角形、四边形等）、位块传输图像、渲染文本、变换几何对象、执行透视纹理映射等。

1.4.2 目录结构

GPU驱动目录结构如下表所示：

表 1-2 GPU驱动目录结构

内容	路径	文件
APP驱动	drivers\src	app_graphics_dc.c
		app_graphics_gpu.c
		app_graphics_ospi.c
		app_graphics_qspi.c
	drivers\inc	app_graphics_dc.h
		app_graphics_gpu.h
		app_graphics_ospi.h
		app_graphics_qspi.h
	components\libraries	app_graphics_mem
Graphics组件	components\graphics\gfx	common
		configure
		hal_gdc
		hal_gfx
		include
		porting
	platform\soc\linker\keil	graphics_sdk.lib
	platform\soc\linker\gcc	libgraphics_sdk.a
刷屏驱动参考	components\drivers_ext\graphics_dc	graphics_dc_lcd_drv.h
		graphics_dc_rm69330_qspi_drv.c
		graphics_dc_rm69330_spi_drv.c
		graphics_dc_st7789v_dspi_drv.c
	components\drivers_ext\qspi_device	qspi_flash.c
		qspi_flash.h
		qspi_nand_flash.c
		qspi_nand_flash.h
		qspi_psram.c
		qspi_psram.h
		qspi_screen.c
		qspi_screen.h
示例工程	projects\peripheral\graphics	graphics_animation_effects
		graphics_benchmarks
		graphics_dc

内容	路径	文件
		graphics_fonts
		graphics_rotate_box
		graphics_tsc6a
		graphics_watch

2 基本概念

本章介绍计算机图形学、几何原语及GPU图形管道等相关概念，帮助开发者更好地理解和使用GPU模块。

2.1 计算机图形学

计算机图形学（Computer Graphics）是通过显示器及其交互设备进行视觉交流的科学，是一个物理、数学、人类感知、人机交互和工程融合的交叉学科领域，在编程的帮助下创建人工图像。它涉及大量数据的计算、创建和操作，并基于一组明确定义的原则。计算机图形学重点概念介绍如下。

- 像素（Pixel）：数字成像中的像素是全点可寻址显示设备中最小的可寻址元素。像素通常被认为是数字图像中最小的单个分量，通常用作测量单位。
- 矢量（Vector）：图形是一种使用多边形的技术，平面图形由有限的直线段链封闭一个循环，来表示图像。矢量图形具有固有的放大能力，仅取决于渲染设备的能力。
- 光栅/栅格（Raster）：即图像（或位图图像），用于表示实际图像内容的矩阵数据结构。光栅图形受分辨率限制，无法在没有明显质量损失的情况下放大。
- 光栅化/栅格化（Rasterization）：将矢量图形格式的图像转换为由像素组成的光栅图像，以便在视频显示器上输出或以位图格式存储的过程。
- 纹理（Texture）：物体表面的数字化表达方式。除了颜色和透明度等二维特性外，纹理还包含反射性等三维特性。定义良好的纹理对于逼真的三维图像表示非常重要。
- 纹理映射（Texture Mapping）：在任何二维或三维对象周围包裹预定义纹理的过程。通过这个过程，数字图像和对象可获得高水平的细节。
- 纹理元（Texel）：纹理空间的基本单位。纹理由纹素数组表示，就像图片由像素数组表示一样。
- 顶点（Vertex）：一种数据结构，通过将对象的角正确定义为二维或三维空间中点的位置来描述对象的位置。
- 几何原语（Primitives）：在计算机图形学中是系统可以处理的最简单的几何对象。常见的二维基元集包括线、点、三角形和多边形，而所有其他几何元素都是从这些基元构建的。在三维中，正确定位的三角形或多边形可以用作基元来模拟更复杂的形式。
- 混合（Blending）：将两个或多个图像按像素和权重组合以创建新图片的过程。
- 片段（Fragment）：生成单像素图元所必需的数据。该数据可能包括光栅位置、颜色或纹理坐标。
- 插值（Interpolation）：在计算机图形学中，是在两个已知参考点之间生成中间值以给出连续性和平滑过渡的外观的过程。计算机图形和动画中使用了几种不同的插值技术，例如线性、双线性、样条和多项式插值。
- 图形管道（Graphics Pipeline）：一个抽象序列，结合了通用光栅化实现的基本操作。

2.2 几何原语

在计算机图形学中，几何原语是系统可以绘制的基本几何形状。在GPU架构中，这些原语由光栅化模块生成。光栅化器生成包含在图元中的片段，并将它们提供给可编程核心进行处理。GPU可以绘制以下几何图元：

- 点（Point）
- 线（Line）
- 填充三角形（Filled Triangle）
- 填充矩形（Filled Rectangle）
- 填充四边形（Filled Quadrilateral）

上述基元均可由可编程核心处理以执行简单的操作（例如用恒定的颜色或渐变填充、位块传输等）或更高级的操作（例如模糊、边缘检测等）。GPU功能可以通过软件进行扩展，以绘制：

- 三角形（Triangle）
- 矩形（Rectangle）
- 多边形（Polygon）
- 填充多边形（Filled Polygon）
- 三角形扇（Triangle Fan）
- 三角形带（Triangle Strip）
- 圆（Circle）
- 填充圆（Filled Circle）
- 弧线（Arc）
- 圆角矩形（Rounded Rectangle）

2.3 GPU图形管道

GR5526 GPU专为超紧凑硅区域的图形效率设计。GPU的定点数据路径和指令集架构（Instruction Set Architecture，ISA）专为GUI加速和小型显示应用量身定制，以最节能的方式结合了对多线程和低级矢量处理的硬件级支持。GPU通过32位AHB总线连接到主机SoC处理器。GPU的内部架构及其图形管线的主要组件如图2-1所示。

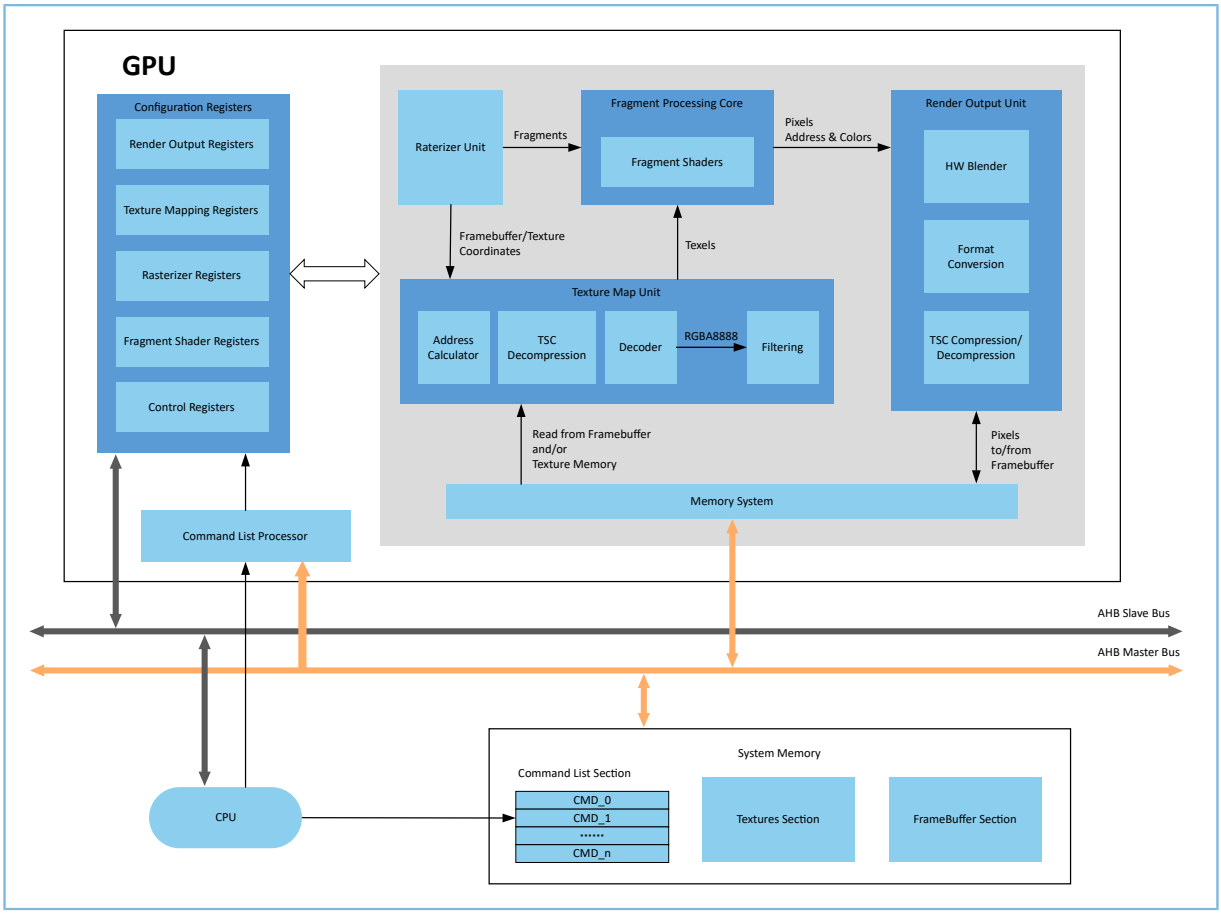


图 2-1 GPU内部架构及其图形管线

2.3.1 配置寄存器文件

GPU通过一组寄存器进行编程。该寄存器集合称为配置寄存器文件（Configuration Register File，CRF），GPU的每个子模块都通过CRF的子集进行编程。CRF可以被内存映射到CPU地址空间，从而使CRF可以直接被CPU访问。直接编写CRF会消耗大量CPU资源并且CPU任务执行与GPU耦合度高，因此可以通过命令列表处理器（Command List Processor，CLP）间接进行读写。在实际使用过程中，可通过CLP对CRF发起控制访问。

2.3.2 命令列表处理单元

为了解耦CPU和GPU的执行并实现更好的性能和更低的功耗，GPU集成了先进的命令列表处理器，能够从主存储器读取整个命令列表并将它们中继到配置寄存器文件。

CPU在将命令列表（Command Lists）提交给命令列表处理器执行之前对其进行预组装，而单个命令列表可以多次提交。这种方法可减少CPU为重复性任务重新计算绘图操作，从而更有效地利用CPU资源。

通过命令列表处理器将命令写入配置寄存器的步骤如下：

1. CPU通过GPU驱动库组装一个命令列表。
2. CPU提交命令列表执行，命令列表处理器被通知一个待处理的命令列表。
3. 命令列表处理器从系统内存中读取命令列表。
4. 命令列表处理器将命令列表中的命令中继到配置寄存器文件。

2.3.3 光栅化

光栅化单元（Rasterizer Unit）读取几何图元顶点的坐标，并为图形管线的其余部分提供几何体中包含的片段。

片段包含有关单个像素的信息，此信息包括光栅位置（坐标）、纹理坐标和插值颜色Alpha值。

2.3.4 纹理贴图单元

纹理贴图单元（Texture Map Unit）产生发送到片段处理核心（Fragment Processing Core）的纹理元。纹理贴图单元执行一些内部处理并输出相应的纹理元。生成纹理元素需要一系列操作，例如：

1. 包装（钳位、镜像、重复等）
2. 从内存中读取相应的颜色值
3. 将颜色值转换为RGBA8888格式
4. 纹理过滤操作
5. 如果使用纹理压缩技术，则会执行动态解压缩

生成后的纹理元包含纹理的基础属性：基地址、尺寸大小、颜色格式以及对应的坐标等。

2.3.5 片段处理核心

片段处理核心是GPU架构的主要处理单元。它是一个64位处理器，对来自光栅化单元的片段和来自纹理映射单元的纹理元执行计算，并计算片段的最终颜色。

片段处理核心可通过称为片段着色器（Fragment Shaders）的二进制可执行文件进行编程。

2.3.6 渲染输出单元

渲染输出单元（Render Output Unit）是图形管道的最后一个阶段。片段处理核心向渲染输出单元提供像素的坐标和颜色值。在将颜色值写入内存之前，颜色会转换为帧缓冲区的格式。如果使用纹理压缩技术，则在读取帧缓冲区时执行解压缩，并在写入帧缓冲区时执行压缩。

如果启用了混合器，则渲染输出单元从片段处理核心（源）读取像素，从帧缓冲区（Framebuffer）（目标）读取像素以执行混合。混合需要在源（前景）和目标（背景）颜色片段之间进行一系列计算，生成最终颜色，并将其写回内存。

2.4 颜色格式

GPU本身支持大量纹理格式，因此能够通过执行动态颜色转换/解压缩来实现快速读取和写入操作。原生格式（从完整的32位RGBA扩展至1位黑白颜色），可选压缩每像素4位有损格式，均可用作源纹理或目标纹理。表 2-1 中列出了GPU支持的格式，供快速参考。

表 2-1 GPU支持的颜色格式

颜色格式	描述
RGBX8888	无透明度的32位色
RGBA8888	有透明度的32位色
XRGB8888	无透明度的32位色
ARGB8888	有透明度的32位色
BGRA8888	有透明度的32位色
BGRX8888	有透明度的32位色
RGBA5650	无透明度的16位色
RGBA5551	1位透明度及16位色
RGBA4444	有透明度的16位色
RGBA3320	无透明度的8位色
L8	8位灰度（亮度）色
A8	8位半透明色
L2	2位灰度（亮度）色
L4	4位灰度（亮度）色
TSC™4	4位TSC压缩
Z24_8	32位（24+4）深度和模板
Z16	16位色深

2.5 帧缓冲区

帧缓冲区是一个内存缓冲区，包含代表完整视频帧中所有像素的数据。在GR5526上，帧缓冲区通常被放置于RAM，若帧缓冲区过大或存在不止一个帧缓冲区，可以将帧缓冲区放置于PSRAM。由于PSRAM的读写访问速率比RAM慢，所以一定程度上会影响图像帧率。

帧缓冲区具有关联的颜色格式，适用于该缓冲区中的所有条目，每一个条目被称之为像素。帧缓冲区的每一存储单元对应屏幕上的一个像素，整个帧缓冲区对应一帧图像。

帧缓冲区具有关联的宽度和高度，通常视为内存的二维部分，可通过X、Y坐标进行索引。

2.5.1 压缩

GPU支持可压缩的颜色格式，帧缓冲区压缩在屏幕块（4x4像素块）中运行，并根据配置实现TSC™4、TSC™6和TSC™6a有损、固定比率压缩。

- TSC™4：6:1压缩（4 bpp）
- TSC™6：4:1压缩（6 bpp）
- TSC™6a：具有Alpha通道的4:1压缩（6 bpp）

压缩是在运行时使用最少的硬件执行的。像素数据以压缩形式存储在帧缓冲区中，并在DC（Display Controller）模块中解压缩。TSC™4压缩方案输出4x4像素块，每块占64位，TSC™6压缩方案输出4x4像素块，每块占96位。

2.6 内存使用

内存单元是GPU内部各模块及各类Master沟通的桥梁。典型的GPU应用程序存在三类内存占用：

- 命令列表缓冲区（Command List Buffer）
- 纹理图像缓存（Texture Cache）
- 帧缓冲区（Framebuffer）

GR5526提供三种类型的内存，但最终内存使用取决于应用程序所使用的硬件。

表 2-2 GPU不同内存的使用方法

类型	使用方法
内置RAM	通常用作命令列表缓冲区、帧缓冲区，也可以用作纹理图像的缓存
PSRAM	通常用作纹理图像的缓存，也可用作命令列表缓冲区、帧缓冲区
内置Flash	通常存放图像、纹理数据和字体等素材

3 开发

本章介绍如何使用GPU进行应用程序的开发，包含GPU的基本功能、基本命令和驱动函数的介绍。

3.1 命令列表

命令列表（Command List，CL）是GPU模块最重要的功能之一，所有GPU接口及功能均可通过CL驱动。使用CL可促进GPU和CPU的解耦，而其固有的可重用性极大地减少CPU的计算工作量。CL的使用使整体架构能够绘制复杂的场景，同时将CPU工作量保持在最低限度。

CL的设计原则允许开发人员扩展其应用程序的功能，同时优化其功能。例如，一个CL能够跳转到另一个CL，从而形成一个无缝互连的命令链。此外，一个CL能够分支到另一个CL，一旦分支执行结束，在分支点之后恢复切换前CL的功能。

3.1.1 环形缓冲区

环形缓冲区（Ring Buffer）是一种用于表示一个固定尺寸、头尾相连的缓冲区的数据结构，如图 3-1所示。

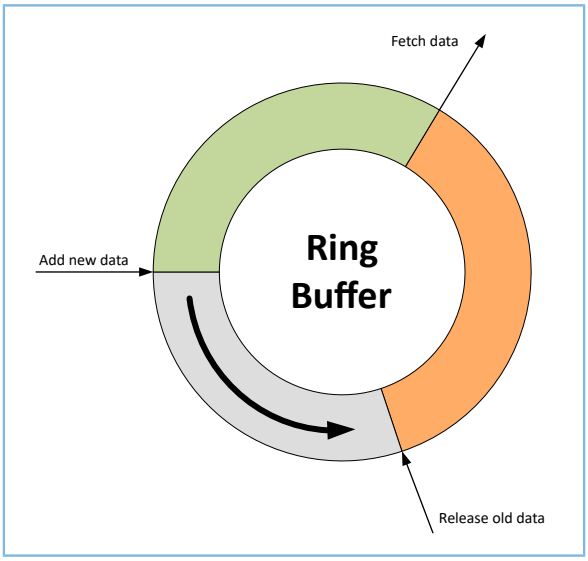


图 3-1 环形缓冲区（Ring Buffer）

开发者可通过设置*graphics_sys_defs.h*中的宏参数，控制Ring Buffer的大小。

```
#define HAL_GFX_RING_BUFFER_SIZE                1024u
```

3.1.2 创建

启动简单编码示例的最直接命令是下面列出的“创建”命令。CL的数据结构是一个基于RAM分配的环形缓冲区，可以通过下列函数基于Ring Buffer创建一个CL列表。“创建”命令可分配和初始化一个新的CL供后续使用。

```
hal_gfx_cmdlist_t hal_gfx_cl_create(void);
```

说明:

- 如果写入CL的操作缓存过多，没有提交执行，存在耗尽Ring Buffer的风险。
- 若绘制命令所需CL空间较大，但创建的CL空间不足且不可拓展，则GPU不会执行该绘制命令的渲染任务。

3.1.3 绑定

绑定命令是将引用的CL设置为当前列表，每个后续的绘图调用都将逐渐合并到当前命令列表中。任何情况下，当有绑定的CL时，才能调用所有的绘图操作。

```
void hal_gfx_cl_bind(hal_gfx_cmdlist_t *cl);
```

3.1.4 解除绑定

解除绑定命令即解绑当前绑定的命令列表。调用解除绑定函数后，需重新绑定相应的命令列表，才能对该命令列表添加相应的绘图操作。

```
void hal_gfx_cl_unbind(void);
```

3.1.5 提交

提交命令是提交引用的命令列表以执行。当提交执行一个CL时，在完成执行前不应改变它，若写入这种中途被修改的CL会导致未定义的行为。

```
void hal_gfx_cl_submit(hal_gfx_cmdlist_t *cl);
```

3.1.6 等待

等待命令即等待CL执行完成。

```
int hal_gfx_cl_wait(hal_gfx_cmdlist_t *cl);
```

裸机环境下，通常是使用一个volatile的标志判断GPU是否完成已提交的CL的执行情况。此处理方式并没有完全发挥出GPU优势。因为在GPU执行CL的渲染任务期间，CPU实际是空闲的。

在RTOS环境下，开发者可利用信号量与任务调度的方式，在等待GPU中断的时间里，让CPU执行其他任务。这样处理可释放出大量CPU资源，从而发挥出GPU处理图像的优势。

3.1.7 销毁

当指定的命令列表执行完成后，可选择销毁该命令列表，释放已申请的内存。命令列表一旦被销毁后，则不能再被使用。如果要创建新的渲染任务，需重新创建命令列表。

```
void hal_gfx_cl_destroy(hal_gfx_cmdlist_t *cl);
```

3.1.8 示例

典型的绘图程序如下：

```
hal_gfx_cmdlist_t cl  = hal_gfx_cl_create();  //Create CL
```

```
hal_gfx_cl_bind(&cl);                  //Bind CL

/* Drawing Operations */               //Draw scene

hal_gfx_cl_unbind();                   //Unbind CL (optionally)
hal_gfx_cl_submit(&cl);                //Submit CL for execution
hal_gfx_cl_wait(&cl);                    //Wait CL
hal_gfx_cl_destroy(&cl);                  //Destroy CL
```

3.2 纹理

纹理（Textures）是应用于表面以改变其颜色、光泽或外观的任何其他部分的图案或图像。每个绘图操作都应该对给定的目标纹理产生影响。纹理所在的内存空间需要能被GPU直接寻址。

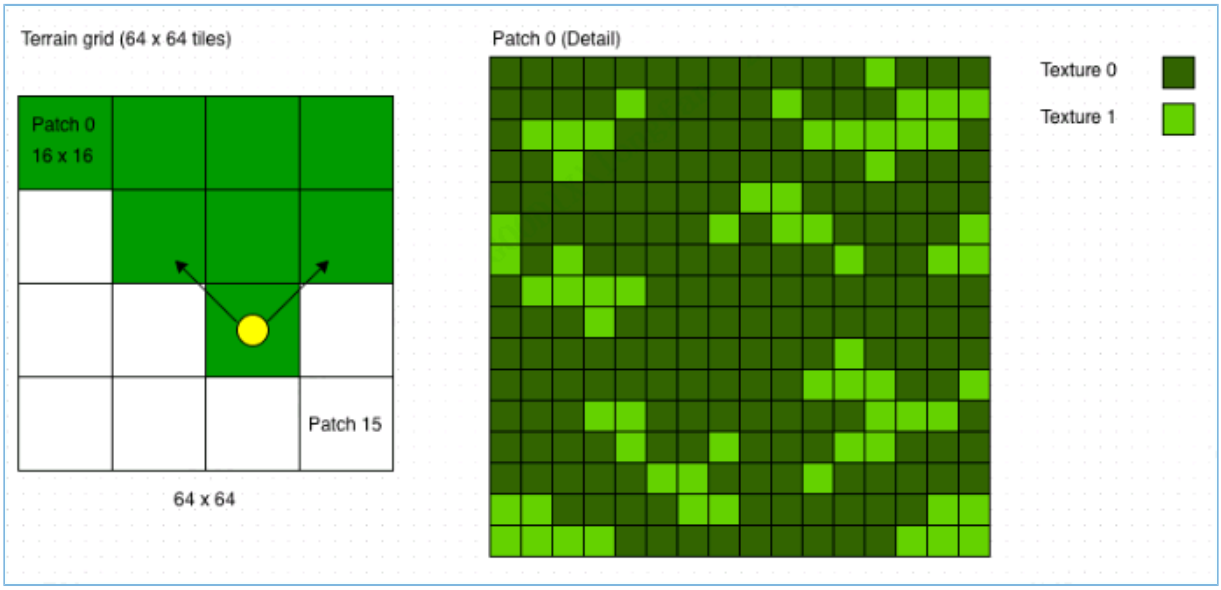


图 3-2 纹理映射与叠加

3.2.1 纹理槽

GPU最多可以支持4个纹理槽，允许同时绑定4个纹理，可分别用于目标、前景、背景等纹理图案叠加/混合。这意味着硬件允许单个着色器读取和/或写4个不同的纹理。在提交着色器执行之前，必须绑定这些纹理，GPU将使用预组装的着色器来执行混合操作。这些着色器建立在表 3-1 的约定关系之上。

表 3-1 纹理槽与着色器的关系

纹理槽	纹理使用
HAL_GFX_TEX0	目的/背景纹理
HAL_GFX_TEX1	前景纹理
HAL_GFX_TEX2	背景纹理
HAL_GFX_TEX3	深度缓冲区

3.2.2 纹理绑定

绑定一个纹理作为目标纹理。将纹理的属性（地址、宽度、高度、格式和步幅）写入绑定的CL内，每个后续的绘制操作都会对这个目标纹理产生影响。下列函数展示将纹理绑定至HAL_GFX_TEX0插槽。

```
void hal_gfx_bind_dst_tex(uintptr_t baseaddr_phys,
                          uint32_t width, uint32_t height,
                          hal_gfx_tex_format_t format, int32_t stride);
```

最常见的图形操作包括某种图像位块传输（复制），如绘制背景图像、GUI图标及字体渲染。以下命令绑定用作前景的纹理：

```
void hal_gfx_bind_src_tex(uintptr_t baseaddr_phys,
                          uint32_t width, uint32_t height,
                          hal_gfx_tex_format_t format, int32_t stride,
                          hal_gfx_tex_mode_t mode);
```

上述函数将纹理绑定至HAL_GFX_TEX1插槽。此函数调用与hal_gfx_bind_dst_tex具有相似功能。它有一个额外的参数hal_gfx_tex_mode_t mode，决定如何读取纹理（点/双线性采样、环绕模式等）。

```
void hal_gfx_bind_src2_tex(uintptr_t baseaddr_phys,
                           uint32_t width, uint32_t height,
                           hal_gfx_tex_format_t format, int32_t stride,
                           hal_gfx_tex_mode_t mode);
```

上述函数将背景纹理绑定至HAL_GFX_TEX2插槽。当要使用的混合模式在混合操作中不使用目标纹理作为背景纹理时，需要调用此函数。

```
void hal_gfx_bind_depth_buffer(uintptr_t baseaddr_phys,
                               uint32_t width, uint32_t height);
```

此函数将深度缓冲区绑定至HAL_GFX_TEX3插槽。

3.2.3 示例

为进一步阐明纹理的使用方法，假设以下示例：需要绘制图 3-3所示的场景，由一个背景图像和两个图标组成。构建该场景需要图 3-4中所示的3个源纹理和一个帧缓冲区（目标纹理）。

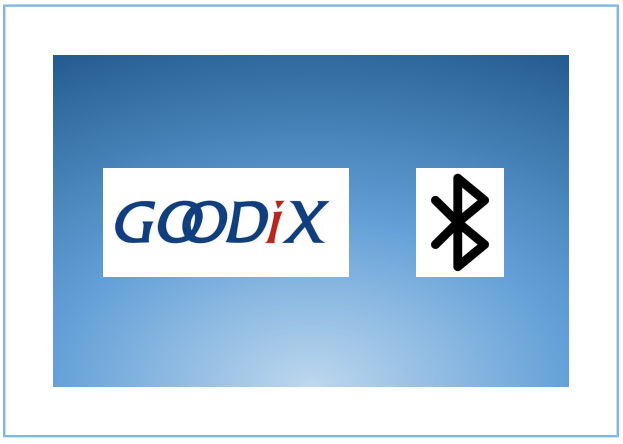


图 3-3 带有两个图标的渲染场景

图 3-4 场景纹理

- 方案一：场景将分3次绘制，首先绘制背景，然后绘制Icon_0，最后绘制Icon_1

1. 绘制背景

   (1) 将Framebuffer绑定至HAL_GFX_TEX0插槽

   (2) 将背景纹理绑定至HAL_GFX_TEX1插槽

   (3) 设置对应的混合模式（HAL_GFX_BL_SRC）

   (4) 位块传输HAL_GFX_TEX1插槽的数据至HAL_GFX_TEX0插槽

2. 绘制Icon_0

   (1) 将Framebuffer绑定至HAL_GFX_TEX0插槽

   (2) 将Icon_0绑定至HAL_GFX_TEX1插槽

   (3) 设置相应的混合模式（例如HAL_GFX_BL_SRC_OVER）

   (4) 位块传输HAL_GFX_TEX1插槽的数据至HAL_GFX_TEX0插槽

3. 绘制Icon_1

   (1) 将Framebuffer绑定至HAL_GFX_TEX0插槽

   (2) 将Icon_1绑定至HAL_GFX_TEX1插槽

   (3) 设置相应的混合模式（例如HAL_GFX_BL_SRC_OVER）

   (4) 位块传输HAL_GFX_TEX1插槽的数据至HAL_GFX_TEX0插槽

如果为上述操作构建一个命令列表，只需要在绘制开始时绑定一次帧缓冲区。此序列将涉及3次位块传输操作。下述方案二则为更高效的方法。

- 方案二：场景分2次绘制，首先在背景之上绘制Icon_0，然后绘制Icon_1

1. 在背景上绘制Icon_0

   (1) 将Framebuffer绑定至HAL_GFX_TEX0插槽

   (2) 将Icon_0绑定至HAL_GFX_TEX1插槽

   (3) 将背景纹理绑定至HAL_GFX_TEX2插槽

   (4) 设置相应的混合模式（例如HAL_GFX_BL_SRC_OVER）

   (5) 将HAL_GFX_TEX2插槽作为后景位块传输HAL_GFX_TEX1插槽的数据至HAL_GFX_TEX0插槽

2. 绘制Icon_1

   (1) 将Framebuffer绑定至HAL_GFX_TEX0插槽

   (2) 将Icon_1绑定至HAL_GFX_TEX1插槽

   (3) 设置相应的混合模式（例如HAL_GFX_BL_SRC_OVER）

   (4) 位块传输HAL_GFX_TEX1插槽的数据至HAL_GFX_TEX0插槽

3.3 剪切区和脏区

3.3.1 剪切区

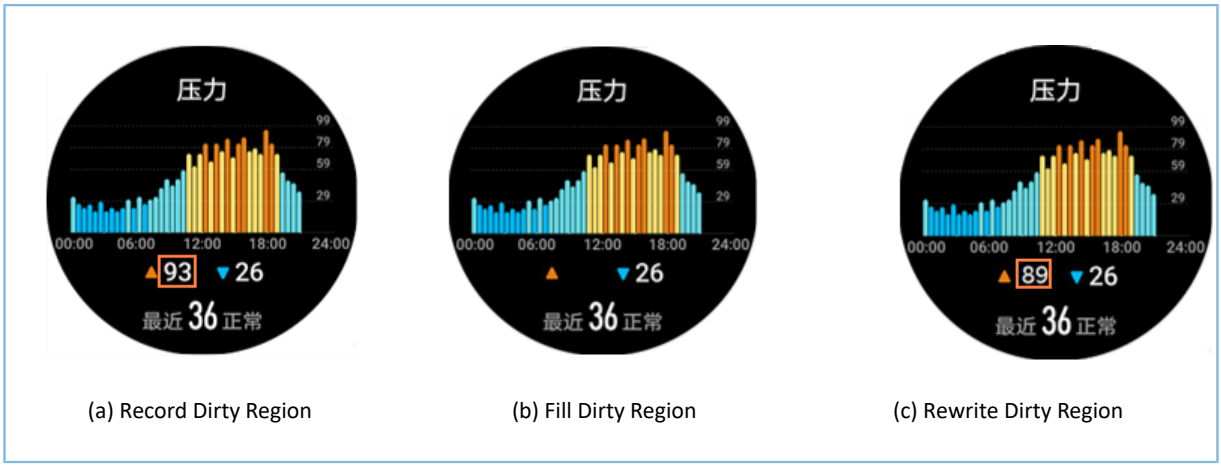
在绘制场景时，通常需要能够定义一个允许GPU绘制的矩形区域，即剪切区。如果图形局部落在Framebuffer或预定的矩形区域之外，这部分是不需要进行计算和绘制的，从而确保正确性、更好的性能和降低功耗。设置剪切区的函数如下：

```
void hal_gfx_set_clip(int32_t x, int32_t y, uint32_t w, uint32_t h);
```

该函数定义了一个裁剪矩形，其左上顶点坐标为(x, y)，尺寸为w * h。

3.3.2 脏区

在绘制场景时，若已在背景图片上对某个局部区域进行图片或文字等内容叠加，但后续需将此图片/文字替换为别的图片或文字，此情况下可使用脏区。



(a) Record Dirty Region (b) Fill Dirty Region (c) Rewrite Dirty Region

图 3-5 脏区的使用

通过使用脏区，可记录上次操作对Framebuffer污染的区域，然后使用背景图对应的区域回写脏区，再使用新的图形/文字填充脏区部分。此方式可有效减少不必要的渲染，从而提升帧率。脏区相关函数如下：

```
void hal_gfx_get_dirty_region(int *minx, int *miny,
                              int *maxx, int *maxy);
void hal_gfx_clear_dirty_region(void);
```

3.4 混合

混合（Blending）是前景图和背景图按照不同的叠加模式形成新图像的过程。GPU模块的混合有两种常用操作：位块传输（Blit）和填充（Fill）。

- 位块传输（Blit）：将一张Texture图绘制到另一个Texture图层
- 填充（Fill）：将色彩填充到另一个Texture图层

在构建图形界面时，开发人员必须定义在画布上绘制像素的结果。由于画布已经包含先前绘制的场景，因此必须以一致的方式来确定源或前景色（将要绘制的颜色）如何与已绘制的目标色或背景色混合。源像素可以是完全不透明的，因此将被绘制在目标像素上，也可以是半透明的，结果将是源颜色和目标颜色的混合。

通过下列函数可以对GPU的混合模式进行设定：

```
void hal_gfx_set_blend_fill(uint32_t blending_mode);
void hal_gfx_set_blend_fill_compose(uint32_t blending_mode);
void hal_gfx_set_blend_blit(uint32_t blending_mode);
void hal_gfx_set_blend_blit_compose(uint32_t blending_mode);
```

3.4.1 预定义的混合模式

混合需要在源（前景）和目标（背景）颜色片段之间进行一系列计算以生成最终颜色，并将其写入内存。预定义的混合模式是一组常用模式，每种模式分别表示颜色（RGB）通道和Alpha通道的源颜色和目标颜色之间的不同计算。表 3-2 显示可用预定义混合模式的完整列表以及产生最终片段颜色的相应计算公式。图 3-6显示每种预定义混合模式的结果。

表 3-2 预定义的混合模式

预定义混合模式	RGB	Alpha	说明
HAL_GFX_BL_SIMPLE	$S_c * S_a + D_c * (1 - S_a)$	$S_a * S_a + D_a * (1 - S_a)$	• S_c：Source Color • S_a：Source Alpha • D_c：Destination Color • D_a：Destination Alpha • C_c：Constant Color • C_a：Constant Alpha
HAL_GFX_BL_CLEAR	0	0	
HAL_GFX_BL_SRC	S_c	S_a	
HAL_GFX_BL_SRC_OVER	$S_c + D_c * (1 - S_a)$	$S_a + D_a * (1 - S_a)$	
HAL_GFX_BL_DST_OVER	$S_c * (1 - D_a) + D_c$	$S_a * (1 - D_a) + D_a$	
HAL_GFX_BL_SRC_IN	$S_c * D_a$	$S_a * D_a$	
HAL_GFX_BL_DST_IN	$D_c * S_a$	$D_a * S_a$	
HAL_GFX_BL_SRC_OUT	$S_c * (1 - D_a)$	$S_a * (1 - D_a)$	
HAL_GFX_BL_DST_OUT	$D_c * (1 - S_a)$	$D_a * (1 - S_a)$	
HAL_GFX_BL_SRC_ATOP	$S_c * D_a + D_c * (1 - S_a)$	$S_a * D_a + D_a * (1 - S_a)$	
HAL_GFX_BL_DST_ATOP	$S_c * (1 - D_a) + D_c * S_a$	$S_a * (1 - D_a) + D_a * S_a$	
HAL_GFX_BL_ADD	$S_c + D_c$	$S_a + D_a$	
HAL_GFX_BL_XOR	$S_c * (1 - D_a) + D_c * (1 - S_a)$	$S_a * (1 - D_a) + D_a * (1 - S_a)$	

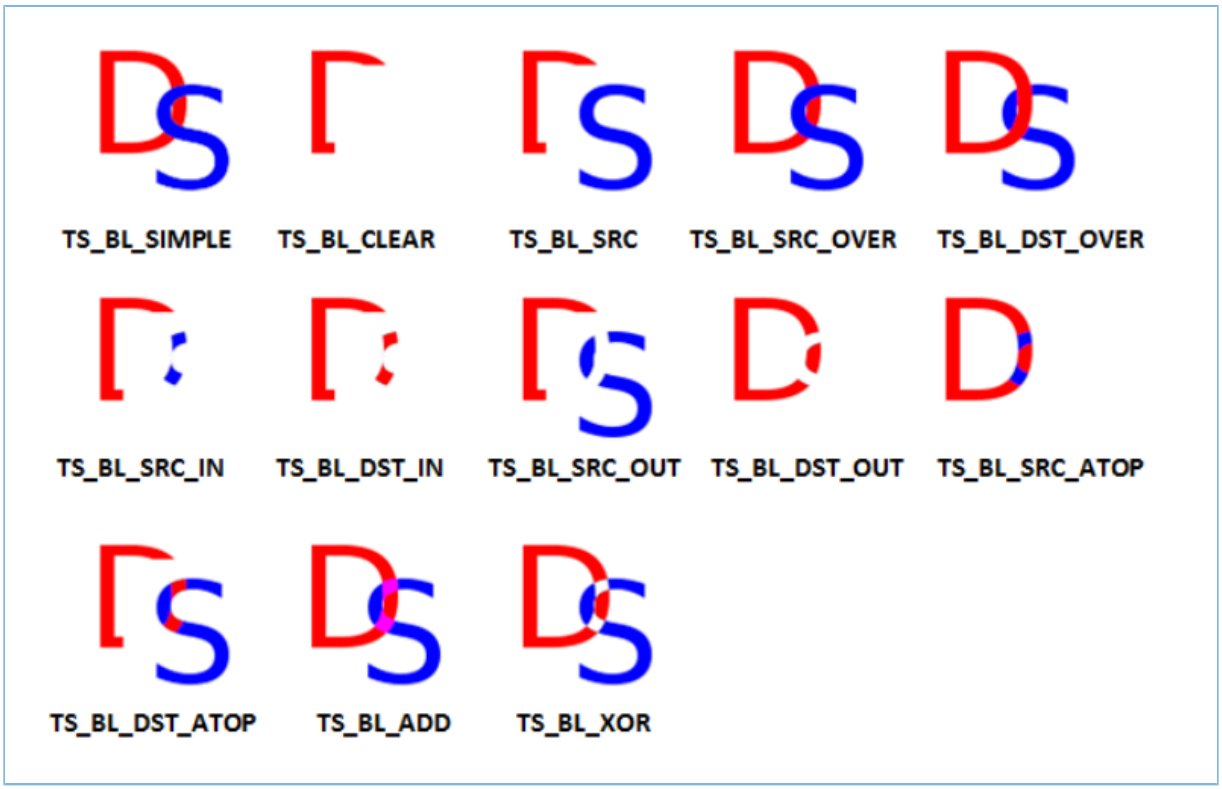


图 3-6 预定义的混合模式

3.4.2 自定义模式

开发人员可以通过以下函数使用源（前景）和目标（背景）颜色的不同因素来创建自定义混合模式。“blops”参数指3.4.3 附加操作中介绍的附加操作。

```
uint32_t hal_gfx_blending_mode(uint32_t src_bf,
                               uint32_t dst_bf,
                               uint32_t blops);
```

使用公式可以计算得出最终颜色的结果。表 3-3 中列出可用的混合因子。图 3-7显示可用的自定义混合模式。

$F_c = S_c * S_f + D_c * D_f$

$F_a = S_a * S_f + D_a * D_f$

说明:

- F_c：Final Color
- F_a：Final Alpha
- S_f：Source Blend Factor (multiplier)
- D_f：Destination Blend Factor (multiplier)

表 3-3 混合因子

混合因子	公式值
HAL_GFX_BF_ZERO	0
HAL_GFX_BF_ONE	1
HAL_GFX_BF_SRCCOLOR	S_C
HAL_GFX_BF_INVSRCCOLOR	$(1 - S_c)$
HAL_GFX_BF_SRCALPHA	S_a
HAL_GFX_BF_INVSRCALPHA	$(1 - S_a)$
HAL_GFX_BF_DESTALPHA	D_a
HAL_GFX_BF_INVDESTALPHA	$(1 - D_a)$
HAL_GFX_BF_DESTCOLOR	D_c
HAL_GFX_BF_INVDESTCOLOR	$(1 - D_c)$
HAL_GFX_BF_CONSTCOLOR	C_c
HAL_GFX_BF_CONSTALPHA	C_a

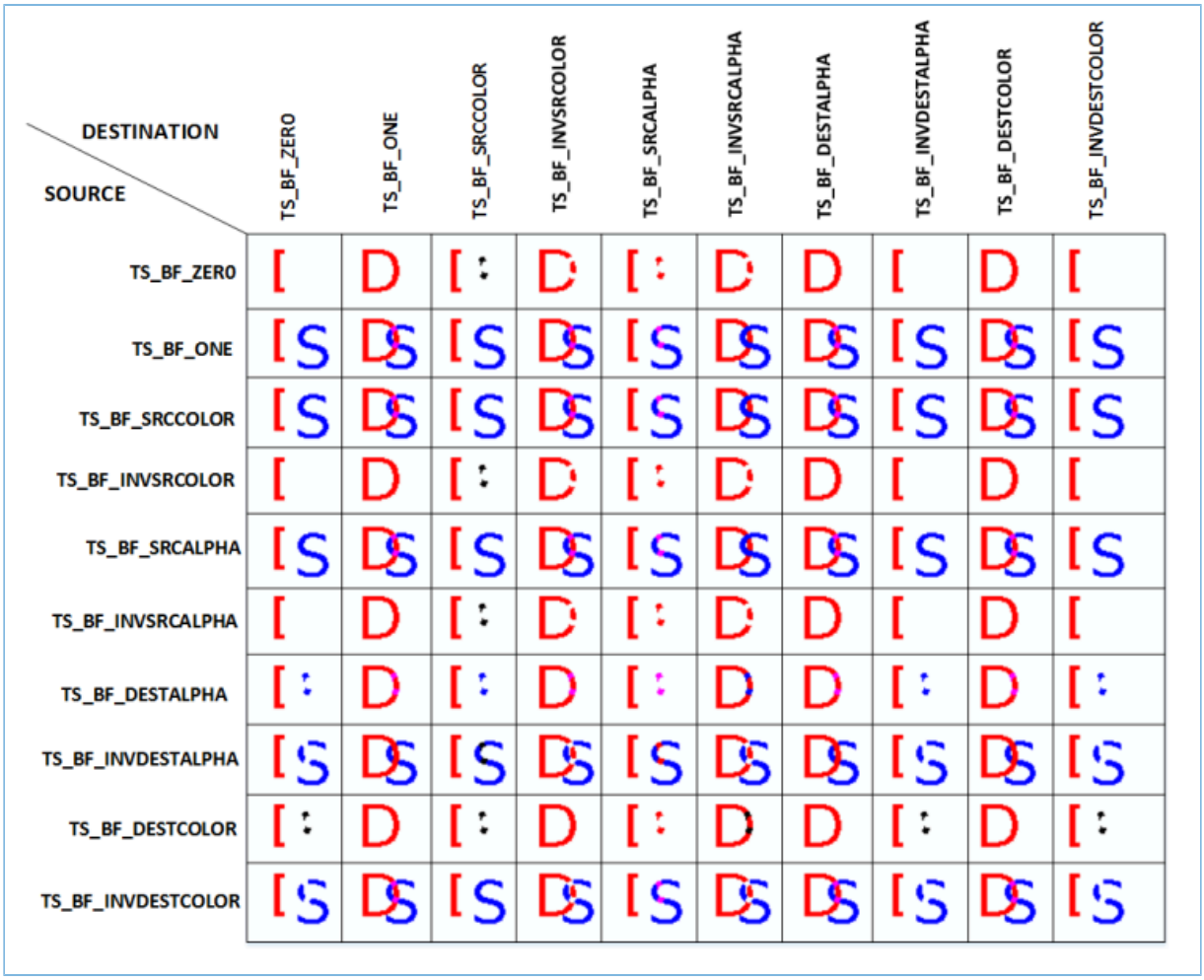


图 3-7 自定义的混合模式

3.4.3 附加操作

GPU运行附加的混合操作，可与混合模式一起应用。表 3-4 中列出其他支持的操作。整个过程的示例如图 3-8和图 3-9所示。

表 3-4 附加操作参数

操作参数	描述
HAL_GFX_BLOP_MODULATE_A	在混合之前将源Alpha通道与Ca常数相乘 通过调用hal_gfx_set_const_color定义Ca
HAL_GFX_BLOP_FORCE_A	在混合之前用Ca替换源Alpha通道 覆盖HAL_GFX_BLOP_MODULATE_A选项 通过调用hal_gfx_set_const_color定义Ca
HAL_GFX_BLOP_MODULATE_RGB	在混合之前将源颜色通道（RGB）与Cc相乘 Cc通过调用hal_gfx_set_const_color定义
HAL_GFX_BLOP_SRC_CKEY	当源颜色与源颜色键匹配时忽略片段，该键通过调 用hal_gfx_set_src_color_key定义

操作参数	描述
HAL_GFX_BLOP_DST_CKEY	当目标颜色与目标颜色键匹配时忽略片段，通过调用hal_gfx_set_dst_color_key定义

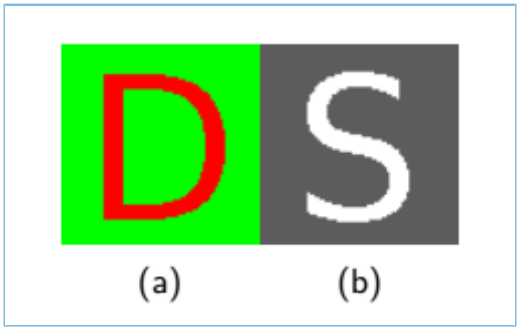


图 3-8 源纹理

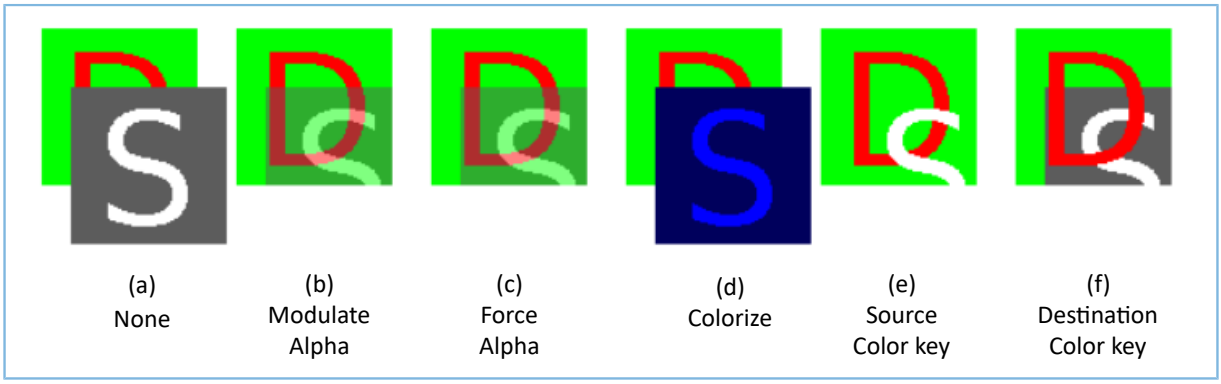


图 3-9 附加操作示例

3.5 基础绘图

GPU的驱动库中有丰富的函数，可用于几何图元的绘制。CL包含对图像进行位块传输处理或者用颜色填充几何图元所需要的所有信息。通过组合各种基础命令，可绘制出各式各样的图像。

假设需要绘制一个背景为320x240的图像，从屏幕坐标(0, 0)（画布的左上角）开始，然后绘制一个从点(20, 30)开始的红色矩形，尺寸为100x200。一个典型的绘图命令如下：

```
hal_gfx_cmdlist_t cl  = hal_gfx_cl_create(); //Create CL
hal_gfx_cl_bind(&cl);          //Bind CL

/* Bind Destination Texture */
hal_gfx_bind_dst_tex(DST_IMAGE,        //Destination address
                     320, 240,   //width, height
                     HAL_GFX_RGBA8888,  //image format
                     320*4);     //stride in bytes

/* Bind Foreground Texture */
hal_gfx_bind_src_tex(SRC_IMAGE,        //Source address
                     320, 240,   //width, height
                     HAL_GFX_RGBA8888,  //image format
                     320*4,      //stride in bytes
```

```
                    HAL_GFX_FILTER_PS);       //Do point sampling

//Define a 320x240 Clipping Rectangle
hal_gfx_set_clip(0, 0, 320, 240);

//Program the Core to draw the source texture without blending it with the
destination texture
hal_gfx_set_blend_blit(HAL_GFX_BL_SRC);

//Blit the bound source texture to destination texture
hal_gfx_blit(0, 0);

//Program the Core to fill the Geometric Primitive without blending it with the
destination texture
hal_gfx_set_blend_fill(HAL_GFX_BL_SRC);

//Fill a rectangular area with red color
hal_gfx_fill_rect(20, 30, 100, 200, RED);

hal_gfx_cl_unbind();       //Unbind CL (optionally)
hal_gfx_cl_submit(&cl);    //Submit CL for execution
hal_gfx_cl_wait(&cl);      //Wait CL
hal_gfx_cl_destroy(&cl);         //Destroy CL
```

上述命令过程会产生下图中显示的输出。

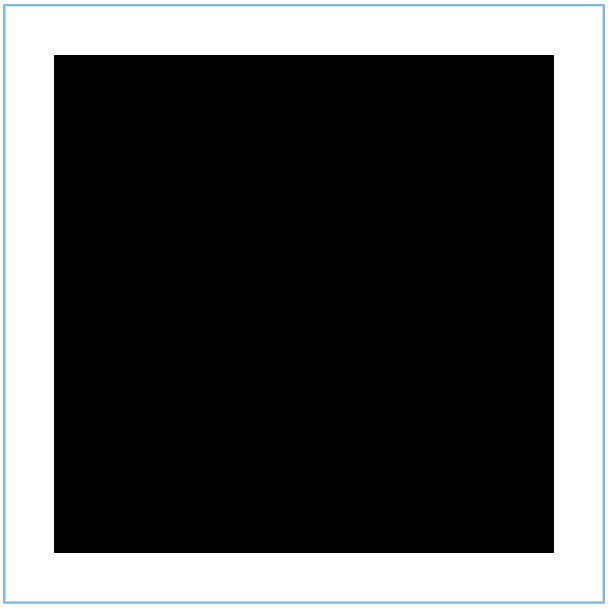


图 3-10 原始的空帧缓冲区

图 3-11 背景

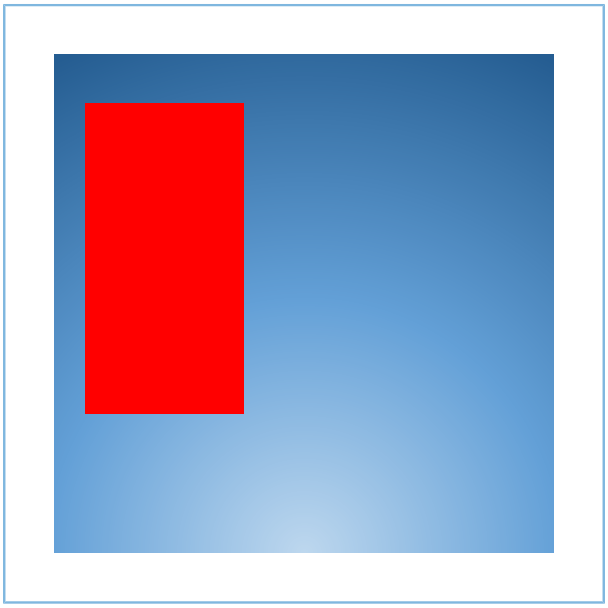


图 3-12 最终输出

3.6 字体

在屏幕上绘制文本是图形用户界面的重要元素。绘制字符串，需要选取字体、待绘制的文本以及实现文本显示的相关属性。

GPU支持的字体来源于TrueType（TTF）文件类型，其中包含描述为矢量曲线的字体的可缩放表示。可缩放字体通过光栅化转换为特定大小和格式的光栅字体（位图字体）。光栅字体在屏幕上绘制为一系列图像，每个字母使用正确的字母宽度依次绘制。

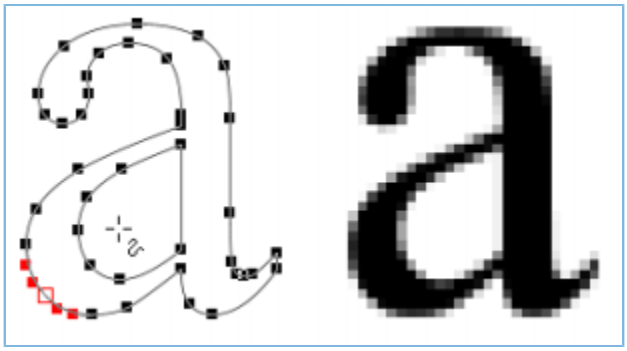


图 3-13 矢量和位图字体

字体文件支持UTF-8字符，每像素占1、2、4或8位。字体位图包含在.c和.bin文件中。可以通过定义HAL_GFX_FONT_LOAD_FROM_BIN从.c文件中排除位图。此情况下，必须使用.bin才能访问字体位图。否则可以忽略.bin文件，通过.c文件访问字体位图。编译项目时，将.c文件添加到已编译的源代码中，并在需要的地方包含.h文件。

绘制文字前，需将一个字体的数据结构与下列函数绑定。

```
void hal_gfx_bind_font(hal_gfx_font_t *font);
```

以下函数可绘制一些文本：

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

上述函数的参数定义如下：

- const char *str：需要绘制的文本
- int x, int y：文本应绘制到的(x, y)屏幕坐标
- int w, int h：文本区域的宽度和高度
- uint32_t fg_col：文本的颜色
- uint32_t align：文本对齐

  以下参数可设置对齐方式：

  - 水平对齐：HAL_GFX_ALIGNX_LEFT、HAL_GFX_ALIGNX_RIGHT、HAL_GFX_ALIGNX_CENTER、HAL_GFX_ALIGNX_JUSTIFY
  - 垂直对齐：HAL_GFX_ALIGNY_TOP、HAL_GFX_ALIGNY_BOTTOM、HAL_GFX_ALIGNY_CENTER、HAL_GFX_ALIGNY_JUSTIFY
  - 换行：HAL_GFX_TEXT_WRAP

3.6.1 字距调整

在字体渲染中，GPU支持字距调整（Kerning）。字距调整是调整比例字体中字符间距的过程，如下图所示。

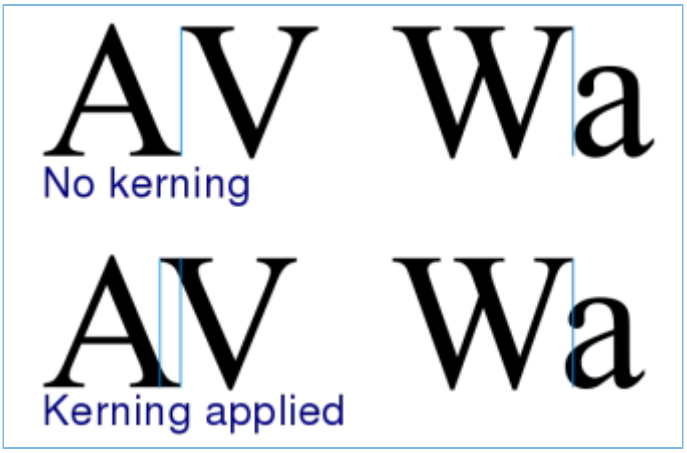


图 3-14 字距调整文本渲染

将TTF转换为位图字体时，间距偏移从TTF文件中包含的紧排和GPOS（Glyph Positioning）表中检索。如果这两个表中都存在字距调整对，则将使用GPOS中包含的偏移量。

在提取字距对后，调整后的字体将存储在生成的.c文件中。因此，字距调整会增加生成字体所需的内存大小。

3.7 GPU驱动函数

3.7.1 hal_gfx_blender.h

3.7.1.1 宏定义

表 3-5 hal_gfx_blender.h宏定义

宏名称	描述
HAL_GFX_BF_ZERO	0
HAL_GFX_BF_ONE	1
HAL_GFX_BF_SRCCOLOR	Sc
HAL_GFX_BF_INVSRCCOLOR	(1-Sc)
HAL_GFX_BF_SRCALPHA	Sa
HAL_GFX_BF_INVSRCALPHA	(1-Sa)
HAL_GFX_BF_DESTALPHA	Da
HAL_GFX_BF_INVDESTALPHA	(1-Da)
HAL_GFX_BF_DESTCOLOR	Dc
HAL_GFX_BF_INVDESTCOLOR	(1-Dc)
HAL_GFX_BF_CONSTCOLOR	Cc
HAL_GFX_BF_CONSTALPHA	Ca
HAL_GFX_BL_SIMPLE	Sa * Sa + Da * (1 - Sa)
HAL_GFX_BL_CLEAR	0
HAL_GFX_BL_SRC	Sa

宏名称	描述
HAL_GFX_BL_SRC_OVER	Sa + Da * (1 - Sa)
HAL_GFX_BL_DST_OVER	Sa * (1 - Da) + Da
HAL_GFX_BL_SRC_IN	Sa * Da
HAL_GFX_BL_DST_IN	Da * Sa
HAL_GFX_BL_SRC_OUT	Sa * (1 - Da)
HAL_GFX_BL_DST_OUT	Da * (1 - Sa)
HAL_GFX_BL_SRC_ATOP	Sa * Da + Da * (1 - Sa)
HAL_GFX_BL_DST_ATOP	Sa * (1 - Da) + Da * Sa
HAL_GFX_BL_ADD	Sa + Da
HAL_GFX_BL_XOR	Sa * (1 - Da) + Da * (1 - Sa)
HAL_GFX_BLOP_NONE	无额外的混合操作
HAL_GFX_BLOP_STENCIL_TXTY	使用TEX3作为掩码
HAL_GFX_BLOP_STENCIL_XY	使用TEX3作为掩码
HAL_GFX_BLOP_NO_USE_ROPBL	不使用Rop混合
HAL_GFX_BLOP_DST_CKEY_NEG	应用反向目标颜色键控（仅在dst颜色与颜色键不匹配时绘制）
HAL_GFX_BLOP_SRC_PREMULT	使用源Alpha预乘源颜色（不能与HAL_GFX_BLOP_MODULATE_RGB一起使用）
HAL_GFX_BLOP_MODULATE_A	通过恒定Alpha值调制
HAL_GFX_BLOP_FORCE_A	强制Alpha值为常数
HAL_GFX_BLOP_MODULATE_RGB	通过恒定颜色（RGB）值调制
HAL_GFX_BLOP_SRC_CKEY	应用源颜色键控（仅在src颜色与颜色键不匹配时绘制）
HAL_GFX_BLOP_DST_CKEY	应用目标颜色键控（仅在dst颜色与颜色键匹配时绘制）
HAL_GFX_BLOP_MASK	功能掩码

3.7.1.2 hal_gfx_blending_mode

表 3-6 hal_gfx_blending_mode接口

函数原型	static inline uint32_t hal_gfx_blending_mode(uint32_t src_bf, uint32_t dst_bf, uint32_t blops)
功能说明	生成最终混合模式（自定义）
输入参数	• uint32_t src_bf：源混合因子（可选择HAL_GFX_BF相关的宏定义） • uint32_t dst_bf：目标混合因子（可选择HAL_GFX_BF相关的宏定义） • uint32_t blops：额外的混合操作（可选择HAL_GFX_BLOP相关的宏定义）
返回值	最终混合模式
备注	

3.7.1.3 hal_gfx_set_blend

表 3-7 hal_gfx_set_blend接口

函数原型	void hal_gfx_set_blend(uint32_t blending_mode, hal_gfx_tex_t dst_tex, hal_gfx_tex_t fg_tex, hal_gfx_tex_t bg_tex)
功能说明	设置混合模式（自定义）
输入参数	• uint32_t blending_mode：混合模式（可选择HAL_GFX_BL和HAL_GFX_BLOP相关的宏定义） • hal_gfx_tex_t dst_tex：目标纹理 • hal_gfx_tex_t fg_tex：前景纹理 • hal_gfx_tex_t bg_tex：后景纹理
返回值	无
备注	位块传输模式仅支持前景纹理槽混合和前景与背景纹理槽混合

3.7.1.4 hal_gfx_set_blend_fill

表 3-8 hal_gfx_set_blend_fill接口

函数原型	static inline void hal_gfx_set_blend_fill(uint32_t blending_mode)
功能说明	设置填充的混合模式
输入参数	uint32_t blending_mode：混合模式（可选择HAL_GFX_BL和HAL_GFX_BLOP相关的宏定义）
返回值	无
备注	

3.7.1.5 hal_gfx_set_blend_fill_compose

表 3-9 hal_gfx_set_blend_fill_compose接口

函数原型	static inline void hal_gfx_set_blend_fill_compose(uint32_t blending_mode)
功能说明	设置填充的混合模式（带后景纹理）
输入参数	uint32_t blending_mode：混合模式（可选择HAL_GFX_BL和HAL_GFX_BLOP相关的宏定义）
返回值	无
备注	

3.7.1.6 hal_gfx_set_blend_blit

表 3-10 hal_gfx_set_blend_blit接口

函数原型	static inline void hal_gfx_set_blend_blit(uint32_t blending_mode)
功能说明	设置位块传输的混合模式
输入参数	uint32_t blending_mode：混合模式（可选择HAL_GFX_BL和HAL_GFX_BLOP相关的宏定义）
返回值	无
备注	

3.7.1.7 hal_gfx_set_blend_blit_compose

表 3-11 hal_gfx_set_blend_blit_compose接口

函数原型	static inline void hal_gfx_set_blend_blit_compose(uint32_t blending_mode)
功能说明	设置位块传输的混合模式（带后景纹理）
输入参数	uint32_t blending_mode：混合模式（可选择HAL_GFX_BL和HAL_GFX_BLOP相关的宏定义）
返回值	无
备注	

3.7.1.8 hal_gfx_set_const_color

表 3-12 hal_gfx_set_const_color接口

函数原型	void hal_gfx_set_const_color(uint32_t rgba)
功能说明	设置Cc和Ca值（额外的混合操作使用）
输入参数	uint32_t rgba：RGBA颜色（格式：R[0,7] G[8,15] B[16,23] A[24,31]）
返回值	无
备注	

3.7.1.9 hal_gfx_set_src_color_key

表 3-13 hal_gfx_set_src_color_key接口

函数原型	void hal_gfx_set_src_color_key(uint32_t rgba)
功能说明	设置源色键值（额外的混合操作使用）
输入参数	uint32_t rgba：RGBA颜色键（格式：R[0,7] G[8,15] B[16,23] A[24,31]）
返回值	无
备注	

3.7.1.10 hal_gfx_set_dst_color_key

表 3-14 hal_gfx_set_dst_color_key接口

函数原型	void hal_gfx_set_dst_color_key(uint32_t rgba)
功能说明	设置目标色键值（额外的混合操作使用）
输入参数	uint32_t rgba：RGBA颜色键（格式：R[0,7] G[8,15] B[16,23] A[24,31]）
返回值	无
备注	

3.7.1.11 hal_gfx_debug_overdraws

表 3-15 hal_gfx_debug_overdraws接口

函数原型	void hal_gfx_debug_overdraws(uint32_t enable)
功能说明	启用/禁用过度绘制调试，禁用渐变和纹理
输入参数	uint32_t enable：非零启用
返回值	无
备注	强制混合模式为HAL_GFX_BL_ADD

3.7.2 hal_gfx_cmdlist.h

3.7.2.1 hal_gfx_cl_create_prealloc

表 3-16 hal_gfx_cl_create_prealloc接口

函数原型	hal_gfx_cmdlist_t hal_gfx_cl_create_prealloc(hal_gfx_buffer_t *bo)
功能说明	在预先分配的空间中创建一个新的命令列表
输入参数	hal_gfx_buffer_t *bo：GPU memory指针
返回值	新命令列表的实例
备注	

3.7.2.2 hal_gfx_cl_create_sized

表 3-17 hal_gfx_cl_create_sized接口

函数原型	hal_gfx_cmdlist_t hal_gfx_cl_create_sized(int size_bytes)
功能说明	创建一个新的不可扩展的特定大小的命令列表
输入参数	int size_bytes：命令列表的大小（以字节为单位）
返回值	新命令列表的实例
备注	

3.7.2.3 hal_gfx_cl_create

表 3-18 hal_gfx_cl_create接口

函数原型	hal_gfx_cmdlist_t hal_gfx_cl_create(void)
功能说明	创建一个新的可扩展命令列表
输入参数	无
返回值	新命令列表的实例
备注	

3.7.2.4 hal_gfx_cl_destroy

表 3-19 hal_gfx_cl_destroy接口

函数原型	void hal_gfx_cl_destroy(hal_gfx_cmdlist_t *cl)
功能说明	销毁/释放命令列表
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.5 hal_gfx_cl_rewind

表 3-20 hal_gfx_cl_rewind接口

函数原型	void hal_gfx_cl_rewind(hal_gfx_cmdlist_t *cl)
功能说明	将要写入的下一条命令的位置复位到开头（不清除列表的内容）
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.6 hal_gfx_cl_bind

表 3-21 hal_gfx_cl_bind接口

函数原型	void hal_gfx_cl_bind(hal_gfx_cmdlist_t *cl)
功能说明	绑定命令列表
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.7 hal_gfx_cl_bind_circular

表 3-22 hal_gfx_cl_bind_circular接口

函数原型	void hal_gfx_cl_bind_circular(hal_gfx_cmdlist_t *cl)
功能说明	绑定循环的命令列表
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.8 hal_gfx_cl_unbind

表 3-23 hal_gfx_cl_unbind接口

函数原型	void hal_gfx_cl_unbind(void)
功能说明	解除当前绑定的命令列表
输入参数	无
返回值	无
备注	

3.7.2.9 hal_gfx_cl_get_bound

表 3-24 hal_gfx_cl_get_bound接口

函数原型	hal_gfx_cmdlist_t *hal_gfx_cl_get_bound(void)
功能说明	获取绑定的命令列表
输入参数	无
返回值	指向命令列表的指针
备注	

3.7.2.10 hal_gfx_cl_submit_no_irq

表 3-25 hal_gfx_cl_submit_no_irq接口

函数原型	void hal_gfx_cl_submit_no_irq(hal_gfx_cmdlist_t *cl)
功能说明	将命令推送到命令列表（不触发中断）
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.11 hal_gfx_cl_submit

表 3-26 hal_gfx_cl_submit接口

函数原型	void hal_gfx_cl_submit(hal_gfx_cmdlist_t *cl)
功能说明	将命令列表排入环形缓冲区以执行
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.12 hal_gfx_cl_wait

表 3-27 hal_gfx_cl_wait接口

函数原型	int hal_gfx_cl_wait(hal_gfx_cmdlist_t *cl)
功能说明	等待命令列表完成
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	非零出错
备注	

3.7.2.13 hal_gfx_cl_add_cmd

表 3-28 hal_gfx_cl_add_cmd接口

函数原型	void hal_gfx_cl_add_cmd(uint32_t reg, uint32_t data)
功能说明	将命令添加到绑定的命令列表
输入参数	uint32_t reg：要写入的硬件寄存器 uint32_t data：要写入的数据
返回值	无
备注	

3.7.2.14 hal_gfx_cl_add_multiple_cmds

表 3-29 hal_gfx_cl_add_multiple_cmds接口

函数原型	int hal_gfx_cl_add_multiple_cmds(int cmd_no, uint32_t *cmd)
功能说明	将多个命令添加到绑定的命令列表
输入参数	• int cmd_no：要添加的命令数 • uint32_t *cmd：指向要添加的命令的指针
返回值	非零出错
备注	

3.7.2.15 hal_gfx_cl_get_space

表 3-30 hal_gfx_cl_get_space接口

函数原型	uint32_t * hal_gfx_cl_get_space(int cmd_no)
功能说明	从命令列表请求可用空间
输入参数	int cmd_no：要写入的命令数
返回值	指向添加命令的指针
备注	

3.7.2.16 hal_gfx_cl_branch

表 3-31 hal_gfx_cl_branch接口

函数原型	void hal_gfx_cl_branch(hal_gfx_cmdlist_t *cl)
功能说明	从绑定的命令列表分支到不同的命令列表（隐含返回）
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.17 hal_gfx_cl_jump

表 3-32 hal_gfx_cl_jump接口

函数原型	void hal_gfx_cl_jump(hal_gfx_cmdlist_t *cl)
功能说明	从绑定的命令列表跳转到不同的命令列表（没有隐含的返回）
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	无
备注	

3.7.2.18 hal_gfx_cl_return

表 3-33 hal_gfx_cl_return接口

函数原型	void hal_gfx_cl_return(void)
功能说明	将显式返回命令添加到绑定的命令列表
输入参数	无
返回值	无
备注	

3.7.2.19 hal_gfx_cl_almost_full

表 3-34 hal_gfx_cl_almost_full接口

函数原型	int hal_gfx_cl_almost_full(hal_gfx_cmdlist_t *cl)
功能说明	查询命令列表是否几乎已满
输入参数	hal_gfx_cmdlist_t *cl：指向命令列表的指针
返回值	非零表示几乎已满
备注	

3.7.2.20 hal_gfx_cl_enough_space

表 3-35 hal_gfx_cl_enough_space接口

函数原型	int hal_gfx_cl_enough_space(int cmd_no)
功能说明	检查是否有足够的空间或扩展所需的命令
输入参数	int cmd_no：要添加的命令数
返回值	非零表示空间不足
备注	

3.7.3 hal_gfx_easing.h

3.7.3.1 hal_gfx_ez_linear

表 3-36 hal_gfx_ez_linear接口

函数原型	float hal_gfx_ez_linear(float p)
功能说明	y = x，函数计算，输入参数x，返回值y
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.2 hal_gfx_ez_quad_in

表 3-37 hal_gfx_ez_quad_in接口

函数原型	float hal_gfx_ez_quad_in(float p)
功能说明	y = x^2
输入参数	float p：输入值，通常在 [0, 1]范围内
返回值	缓动值
备注	

3.7.3.3 hal_gfx_ez_quad_out

表 3-38 hal_gfx_ez_quad_out接口

函数原型	float hal_gfx_ez_quad_out(float p)
功能说明	y = -x^2 + 2x
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.4 hal_gfx_ez_quad_in_out

表 3-39 hal_gfx_ez_quad_in_out接口

函数原型	float hal_gfx_ez_quad_in_out(float p)
功能说明	y = (1/2)((2x)^2) ; [0, 0.5) y = -(1/2)((2x-1)*(2x-3) - 1) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.5 hal_gfx_ez_cub_in

表 3-40 hal_gfx_ez_cub_in接口

函数原型	float hal_gfx_ez_cub_in(float p)
功能说明	y = x^3
输入参数	float p：输入值，通常在 [0, 1]范围内
返回值	缓动值
备注	

3.7.3.6 hal_gfx_ez_cub_out

表 3-41 hal_gfx_ez_cub_out接口

函数原型	float hal_gfx_ez_cub_out(float p)
功能说明	y = (x - 1)^3 + 1
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.7 hal_gfx_ez_cub_in_out

表 3-42 hal_gfx_ez_cub_in_out接口

函数原型	float hal_gfx_ez_cub_in_out(float p)
功能说明	y = (1/2)((2x)^3) ; [0, 0.5) y = (1/2)((2x-2)^3 + 2) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.8 hal_gfx_ez_quar_in

表 3-43 hal_gfx_ez_quar_in接口

函数原型	float hal_gfx_ez_quar_in(float p)
功能说明	y = x^4
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.9 hal_gfx_ez_quar_out

表 3-44 hal_gfx_ez_quar_out接口

函数原型	float hal_gfx_ez_quar_out(float p)
功能说明	y = 1 - (x - 1)^4
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.10 hal_gfx_ez_quar_in_out

表 3-45 hal_gfx_ez_quar_in_out接口

函数原型	float hal_gfx_ez_quar_in_out(float p)
功能说明	y = (1/2)((2x)^4) ; [0, 0.5) y = -(1/2)((2x-2)^4 - 2) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.11 hal_gfx_ez_quin_in

表 3-46 hal_gfx_ez_quin_in接口

函数原型	float hal_gfx_ez_quin_in(float p)
功能说明	y = x^5
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.12 hal_gfx_ez_quin_out

表 3-47 hal_gfx_ez_quin_out接口

函数原型	float hal_gfx_ez_quin_out(float p)
功能说明	y = (x - 1)^5 + 1
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.13 hal_gfx_ez_quin_in_out

表 3-48 hal_gfx_ez_quin_in_out接口

函数原型	float hal_gfx_ez_quin_in_out(float p)
功能说明	y = (1/2)((2x)^5) ; [0, 0.5) y = (1/2)((2x-2)^5 + 2) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.14 hal_gfx_ez_sin_in

表 3-49 hal_gfx_ez_sin_in接口

函数原型	float hal_gfx_ez_sin_in(float p)
功能说明	模拟正弦波的四分之一周期
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.15 hal_gfx_ez_sin_out

表 3-50 hal_gfx_ez_sin_out接口

函数原型	float hal_gfx_ez_sin_out(float p)
功能说明	模拟正弦波的四分之一周期（不同相位）
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.16 hal_gfx_ez_sin_in_out

表 3-51 hal_gfx_ez_sin_in_out接口

函数原型	float hal_gfx_ez_sin_in_out(float p)
功能说明	模拟半正弦波
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.17 hal_gfx_ez_circ_in

表 3-52 hal_gfx_ez_circ_in接口

函数原型	float hal_gfx_ez_circ_in(float p)
功能说明	模拟单位圆移位IV象限
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.18 hal_gfx_ez_circ_out

表 3-53 hal_gfx_ez_circ_out接口

函数原型	float hal_gfx_ez_circ_out(float p)
功能说明	模拟单位圆移位II象限
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.19 hal_gfx_ez_circ_in_out

表 3-54 hal_gfx_ez_circ_in_out接口

函数原型	float hal_gfx_ez_circ_in_out(float p)
功能说明	y = (1/2)(1 - sqrt(1 - 4x^2)) ; [0, 0.5) y = (1/2)(sqrt(-(2x - 3)*(2x - 1)) + 1) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.20 hal_gfx_ez_exp_in

表 3-55 hal_gfx_ez_exp_in接口

函数原型	float hal_gfx_ez_exp_in(float p)
功能说明	y = 2^(10(x - 1))
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.21 hal_gfx_ez_exp_out

表 3-56 hal_gfx_ez_exp_out接口

函数原型	float hal_gfx_ez_exp_out(float p)
功能说明	y = -2^(-10x) + 1
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.22 hal_gfx_ez_exp_in_out

表 3-57 hal_gfx_ez_exp_in_out接口

函数原型	float hal_gfx_ez_exp_in_out(float p)
功能说明	y = (1/2)2^(10(2x - 1)) ; [0,0.5) y = -(1/2)*2^(-10(2x - 1))) + 1 ; [0.5,1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.23 hal_gfx_ez_elast_in

表 3-58 hal_gfx_ez_elast_in接口

函数原型	float hal_gfx_ez_elast_in(float p)
功能说明	y = sin(13pi/2*x)*pow(2, 10 * (x - 1))
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.24 hal_gfx_ez_elast_out

表 3-59 hal_gfx_ez_elast_out接口

函数原型	float hal_gfx_ez_elast_out(float p)
功能说明	y = sin(-13pi/2*(x + 1))*pow(2, -10x) + 1
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.25 hal_gfx_ez_elast_in_out

表 3-60 hal_gfx_ez_elast_in_out接口

函数原型	float hal_gfx_ez_elast_in_out(float p)
功能说明	y = (1/2)*sin(13pi/2*(2*x))*pow(2, 10 * ((2*x) - 1)) ; [0,0.5) y = (1/2)*(sin(-13pi/2*((2x-1)+1))*pow(2,-10(2*x-1)) + 2) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.26 hal_gfx_ez_back_in

表 3-61 hal_gfx_ez_back_in接口

函数原型	float hal_gfx_ez_back_in(float p)
功能说明	y = x^3-x*sin(x*pi)
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.27 hal_gfx_ez_back_out

表 3-62 hal_gfx_ez_back_out接口

函数原型	float hal_gfx_ez_back_out(float p)
功能说明	y = 1-((1-x)^3-(1-x)*sin((1-x)*pi))
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.28 hal_gfx_ez_back_in_out

表 3-63 hal_gfx_ez_back_in_out接口

函数原型	float hal_gfx_ez_back_in_out(float p)
功能说明	y = (1/2)*((2x)^3-(2x)*sin(2*x*pi)) ; [0, 0.5) y = (1/2)*(1-((1-x)^3-(1-x)*sin((1-x)*pi))+1) ; [0.5, 1]
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.29 hal_gfx_ez_bounce_out

表 3-64 hal_gfx_ez_bounce_out接口

函数原型	float hal_gfx_ez_bounce_out(float p)
功能说明	Bounce函数，实现Bounce缓动效果
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.30 hal_gfx_ez_bounce_in

表 3-65 hal_gfx_ez_bounce_in接口

函数原型	float hal_gfx_ez_bounce_in(float p)
功能说明	Bounce函数，实现Bounce缓动效果
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.31 hal_gfx_ez_bounce_in_out

表 3-66 hal_gfx_ez_bounce_in_out接口

函数原型	float hal_gfx_ez_bounce_in_out(float p)
功能说明	Bounce
输入参数	float p：输入值，通常在[0, 1]范围内
返回值	缓动值
备注	

3.7.3.32 hal_gfx_ez

表 3-67 hal_gfx_ez接口

函数原型	float hal_gfx_ez(float A, float B, float steps, float cur_step, float (*ez_func)(float p))
功能说明	自定义执行缓动
输入参数	• float A：[0, 1]范围内的初始值 • float B：[0, 1]范围内的最终值 • float steps：总步数 • float cur_step：当前步数 • float (*ez_func)(float p)：指向所需缓动函数的指针
返回值	缓动值
备注	

3.7.4 hal_gfx_error.h

3.7.4.1 宏定义

表 3-68 hal_gfx_error.h宏定义

宏名称	描述
HAL_GFX_ERR_NO_ERROR	没有发生错误
HAL_GFX_ERR_SYS_INIT_FAILURE	系统初始化失败
HAL_GFX_ERR_GPU_ABSENT	GPU不存在
HAL_GFX_ERR_RB_INIT_FAILURE	环形缓冲区初始化失败
HAL_GFX_ERR_NON_EXPANDABLE_CL_FULL	不可扩展的命令列表已满
HAL_GFX_ERR_CL_EXPANSION	命令列表扩展错误
HAL_GFX_ERR_OUT_OF_GFX_MEMORY	显存已满
HAL_GFX_ERR_OUT_OF_HOST_MEMORY	主机内存已满
HAL_GFX_ERR_NO_BOUND_CL	没有绑定的命令列表
HAL_GFX_ERR_NO_BOUND_FONT	没有绑定字体
HAL_GFX_ERR_GFX_MEMORY_INIT	显存初始化失败
HAL_GFX_ERR_DRIVER_FAILURE	GPU内核驱动程序故障
HAL_GFX_ERR_MUTEX_INIT	互斥初始化失败
HAL_GFX_ERR_INVALID_BO	提供的缓冲区无效
HAL_GFX_ERR_INVALID_CL	提供的命令列表无效

3.7.4.2 hal_gfx_get_error

表 3-69 hal_gfx_ez_linear接口

函数原型	uint32_t hal_gfx_get_error(void)

功能说明	获取错误码
输入参数	无
返回值	错误码（可选择HAL_GFX_ERR相关的宏定义）
备注	

3.7.5 hal_gfx_font.h

3.7.5.1 宏定义

表 3-70 hal_gfx_font.h宏定义

宏名称	描述
HAL_GFX_ALIGNX_LEFT	向左水平对齐
HAL_GFX_ALIGNX_RIGHT	水平向右对齐
HAL_GFX_ALIGNX_CENTER	水平居中对齐
HAL_GFX_ALIGNX_JUSTIFY	水平对齐
HAL_GFX_ALIGNX_MASK	水平对齐掩码
HAL_GFX_ALIGNY_TOP	垂直对齐到顶部
HAL_GFX_ALIGNY_BOTTOM	垂直对齐到底部
HAL_GFX_ALIGNY_CENTER	垂直居中对齐
HAL_GFX_ALIGNY_JUSTIFY	垂直对齐
HAL_GFX_ALIGNY_MASK	垂直对齐掩码
HAL_GFX_TEXT_WRAP	使用文字环绕

3.7.5.2 hal_gfx_bind_font

表 3-71 hal_gfx_bind_font接口

函数原型	void hal_gfx_bind_font(hal_gfx_font_t *font)
功能说明	绑定字体以在hal_gfx_print()调用中使用
输入参数	hal_gfx_font_t *font：指向字体的指针
返回值	无
备注	

3.7.5.3 hal_gfx_string_get_bbox

表 3-72 hal_gfx_string_get_bbox接口

函数原型	int hal_gfx_string_get_bbox(const char *str, int *w, int *h, int max_w, uint32_t wrap)
功能说明	获取字符串边界框的宽度和高度
输入参数	• const char *str：指向字符串的指针

	• int *w：指向应写入宽度的变量的指针 • int *h：指向应写入高度的变量的指针 • int max_w：最大允许宽度 • uint32_t wrap：环绕模式（可选择HAL_GFX_TEXT_WRAP相关的宏定义）
返回值	回车数
备注	

3.7.5.4 hal_gfx_print

表 3-73 hal_gfx_print接口

函数原型	void hal_gfx_print(const char *str, int x, int y, int w, int h, uint32_t fg_col, uint32_t align)
功能说明	打印预格式化的文本
输入参数	• const char *str：指向字符串的指针 • int x：文本区域左上角的X坐标 • int y：文本区域左上角的Y坐标 • int w：文本区域的宽度 • int h：文本区域的高度 • uint32_t fg_col：文本的颜色 • uint32_t align：对齐和环绕模式（可选择HAL_GFX_ALIGNX和HAL_GFX_ALIGNY和HAL_GFX_TEXT_WRAP相关的宏定义）
返回值	无
备注	

3.7.5.5 hal_gfx_print_to_position

表 3-74 hal_gfx_print_to_position接口

函数原型	void hal_gfx_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)
功能说明	打印预格式化的文本（指定位置）
输入参数	• const char *str：指向字符串的指针 • int *pos_x：要绘制的下一个字符的X位置 • int *pos_y：要绘制的下一个字符的Y位置 • int x：文本区域左上角的X坐标 • int y：文本区域左上角的Y坐标 • int w：文本区域的宽度 • int h：文本区域的高度 • uint32_t fg_col：文本的颜色 • uint32_t align：对齐和环绕模式（可选择HAL_GFX_ALIGNX、HAL_GFX_ALIGNY和HAL_GFX_TEXT_WRAP相关的宏定义）
返回值	无
备注	

3.7.6 hal_gfx_graphics.h

3.7.6.1 宏定义

表 3-75 hal_gfx_graphics.h宏定义

宏名称	描述
HAL_GFX_RGBX8888	RGBX8888
HAL_GFX_RGBA8888	RGBA8888
HAL_GFX_XRGB8888	XRGB8888
HAL_GFX_ARGB8888	ARGB8888
HAL_GFX_RGB565	RGBA5650
HAL_GFX_RGBA5650	RGBA5650
HAL_GFX_RGBA5551	RGBA5551
HAL_GFX_RGBA4444	RGBA4444
HAL_GFX_RGBA0800	RGBA0800
HAL_GFX_A8	RGBA0008
HAL_GFX_RGBA0008	RGBA0008
HAL_GFX_L8	L8
HAL_GFX_RGBA3320	RGBA3320（仅输入）
HAL_GFX_RGB332	RGBA3320（仅输入）
HAL_GFX_BW1	BW1（仅输入）
HAL_GFX_A1	A1（仅输入）
HAL_GFX_L1	L1（仅输入）
HAL_GFX_UYVY	UYVY
HAL_GFX_ABGR8888	ABGR8888
HAL_GFX_BGRA8888	BGRA
HAL_GFX_BGRX8888	BGRX
HAL_GFX_TSC4	TSC4
HAL_GFX_TSC6	TSC6
HAL_GFX_TSC6A	TSC6A
HAL_GFX_RV	RV
HAL_GFX_GU	GU
HAL_GFX_BY	BY
HAL_GFX_YUV	YUV
HAL_GFX_Z24_8	Z24_8
HAL_GFX_Z16	Z16
HAL_GFX_UV	UV

宏名称	描述
HAL_GFX_A1LE	A1LE
HAL_GFX_A2LE	A2LE
HAL_GFX_A4LE	A4LE
HAL_GFX_L1LE	L1LE
HAL_GFX_L2LE	L2LE
HAL_GFX_L4LE	L4LE
HAL_GFX_A2	A2
HAL_GFX_A4	A4
HAL_GFX_L2	L2
HAL_GFX_L4	L4
HAL_GFX_BGR24	BGR24
HAL_GFX_RGB24	RGB24
HAL_GFX_RV10	RV-10bit
HAL_GFX_GU10	GU-10bit
HAL_GFX_BY10	BY-10bit
HAL_GFX_DITHER	GPU抖动显示
HAL_GFX_FORMAT_MASK	格式掩码
HAL_GFX_FILTER_PS	点采样
HAL_GFX_FILTER_BL	双线性滤波采样
HAL_GFX_TEX_CLAMP	钳住寻址模式
HAL_GFX_TEX_REPEAT	重复寻址模式
HAL_GFX_TEX_BORDER	边界寻址模式
HAL_GFX_TEX_MIRROR	镜像寻址模式
HAL_GFX_TEX_MORTON_ORDER	Morton排布
HAL_GFX_TEX_RANGE_0_1	插值坐标范围：0 ～ 1
HAL_GFX_TEX_LEFT_HANDED	(0,0) 是左下角
HAL_GFX_ROT_000_CCW	无旋转
HAL_GFX_ROT_090_CCW	逆时针旋转90度
HAL_GFX_ROT_180_CCW	逆时针旋转180度
HAL_GFX_ROT_270_CCW	逆时针旋转270度
HAL_GFX_ROT_000_CW	无旋转
HAL_GFX_ROT_270_CW	顺时针旋转270度
HAL_GFX_ROT_180_CW	顺时针旋转180度
HAL_GFX_ROT_090_CW	顺时针旋转90度
HAL_GFX_MIR_VERT	垂直镜像

宏名称	描述
HAL_GFX_MIR_HOR	水平镜像

3.7.6.2 枚举类型

表 3-76 hal_gfx_graphics.h枚举类型

枚举类型	成员	描述
hal_gfx_tex_t	HAL_GFX_NOTEX	无纹理
	HAL_GFX_TEX0	纹理槽0
	HAL_GFX_TEX1	纹理槽1
	HAL_GFX_TEX2	纹理槽2
	HAL_GFX_TEX3	纹理槽3
hal_gfx_tri_cull_t	HAL_GFX_CULL_NONE	禁用三角形/四边形剔除
	HAL_GFX_CULL_CW	顺时针剔除三角形/四边形
	HAL_GFX_CULL_CCW	逆时针剔除三角形/四边形
	HAL_GFX_CULL_ALL	剔除所有

3.7.6.3 hal_gfx_checkGPUPresence

表 3-77 hal_gfx_checkGPUPresence接口

函数原型	int hal_gfx_checkGPUPresence(void)
功能说明	检查是否存在已知的GPU
输入参数	无
返回值	如果不存在则返回-1
备注	

3.7.6.4 hal_gfx_bind_tex

表 3-78 hal_gfx_bind_tex接口

函数原型	void hal_gfx_bind_tex(hal_gfx_tex_t texid, uintptr_t addr_gpu, uint32_t width, uint32_t height, hal_gfx_tex_format_t format, int32_t stride, hal_gfx_tex_mode_t wrap_mode)
功能说明	绑定纹理单元（自定义）
输入参数	• hal_gfx_tex_t texid：要编写的纹理单元 • uintptr_t addr_gpu：纹理地址 • uint32_t width：纹理宽度 • uint32_t height：纹理高度 • hal_gfx_tex_format_t format：纹理颜色格式 • int32_t stride：纹理步长（<0待计算）

	• hal_gfx_tex_mode_t wrap_mode：要使用的环绕/重复模式（可选择HAL_GFX_FILTER和HAL_GFX_TEX相关的宏定义）
返回值	无
备注	

3.7.6.5 hal_gfx_set_tex_color

表 3-79 hal_gfx_set_tex_color接口

函数原型	void hal_gfx_set_tex_color(uint32_t color)
功能说明	设置纹理映射默认颜色
输入参数	uint32_t color：32位RGBA格式的默认颜色
返回值	无
备注	

3.7.6.6 hal_gfx_set_const_reg

表 3-80 hal_gfx_set_const_reg接口

函数原型	void hal_gfx_set_const_reg(int reg, uint32_t value)
功能说明	将值写入GPU的常量寄存器
输入参数	• int reg：要写入的常量寄存器 • uint32_t value：要写入的值
返回值	无
备注	

3.7.6.7 hal_gfx_set_clip

表 3-81 hal_gfx_set_clip接口

函数原型	void hal_gfx_set_clip(int32_t x, int32_t y, uint32_t w, uint32_t h)
功能说明	设置绘图区域的剪切矩形
输入参数	• int32_t x：剪辑窗口左上角x坐标 • int32_t y：剪辑窗口左上角y坐标 • uint32_t w：剪辑窗口宽度 • uint32_t h：剪辑窗口高度
返回值	无
备注	

3.7.6.8 hal_gfx_enable_gradient

表 3-82 hal_gfx_enable_gradient接口

函数原型	void hal_gfx_enable_gradient(int enable)

功能说明	启用颜色渐变
输入参数	int enable：非零启用
返回值	无
备注	

3.7.6.9 hal_gfx_enable_depth

表 3-83 hal_gfx_enable_depth接口

函数原型	void hal_gfx_enable_depth(int enable)
功能说明	启用深度
输入参数	int enable：非零启用
返回值	无
备注	

3.7.6.10 hal_gfx_enable_aa

表 3-84 hal_gfx_enable_aa接口

函数原型	uint32_t hal_gfx_enable_aa(uint8_t e0, uint8_t e1, uint8_t e2, uint8_t e3)
功能说明	每个边缘启用多重采样抗锯齿MSAA（Multi-Sampling Anti-Aliasing）
输入参数	• uint8_t e0：为边0（顶点0-1）启用MSAA • uint8_t e1：为边1（顶点1-2）启用MSAA • uint8_t e2：为边2（顶点2-3）启用MSAA • uint8_t e3：为边3（顶点3-0）启用MSAA
返回值	先前的AA标志（可以忽略）
备注	

3.7.6.11 hal_gfx_get_dirty_region

表 3-85 hal_gfx_get_dirty_region接口

函数原型	void hal_gfx_get_dirty_region(int *minx, int *miny, int *maxx, int *maxy)
功能说明	返回自上次调用以来已修改的所有像素的边界矩形
输入参数	• int *minx：脏区左上角的x坐标 • int *miny：脏区左上角的y坐标 • int *maxx：脏区右下角的x坐标 • int *maxy：脏区右下角的y坐标
返回值	无
备注	

3.7.6.12 hal_gfx_clear_dirty_region

表 3-86 hal_gfx_clear_dirty_region接口

函数原型	void hal_gfx_clear_dirty_region(void)
功能说明	清除脏区信息，通过绑定的命令列表运行
输入参数	无
返回值	无
备注	

3.7.6.13 hal_gfx_tri_cull

表 3-87 hal_gfx_tri_cull接口

函数原型	void hal_gfx_tri_cull(hal_gfx_tri_cull_t cull)
功能说明	设置三角形/四边形剔除模式
输入参数	hal_gfx_tri_cull_t cull：剔除模式
返回值	无
备注	

3.7.6.14 hal_gfx_format_size

表 3-88 hal_gfx_format_size接口

函数原型	int hal_gfx_format_size (hal_gfx_tex_format_t format)
功能说明	以字节为单位返回像素大小
输入参数	hal_gfx_tex_format_t format：颜色格式
返回值	以字节为单位返回像素大小
备注	

3.7.6.15 hal_gfx_stride_size

表 3-89 hal_gfx_stride_size接口

函数原型	int hal_gfx_stride_size(hal_gfx_tex_format_t format, hal_gfx_tex_mode_t wrap_mode, int width)
功能说明	以字节为单位返回步长
输入参数	• hal_gfx_tex_format_t format：颜色格式 • hal_gfx_tex_mode_t wrap_mode：要使用的环绕/重复模式（可选择HAL_GFX_FILTER和HAL_GFX_TEX相关的宏定义） • int width：纹理宽度
返回值	以字节为单位返回步长
备注	

3.7.6.16 hal_gfx_texture_size

表 3-90 hal_gfx_texture_size接口

函数原型	int hal_gfx_texture_size(hal_gfx_tex_format_t format, hal_gfx_tex_mode_t wrap_mode, int width, int height)
功能说明	以字节为单位返回纹理大小
输入参数	• hal_gfx_tex_format_t format：颜色格式 • hal_gfx_tex_mode_t wrap_mode：要使用的环绕/重复模式（可选择HAL_GFX_FILTER和HAL_GFX_TEX相关的宏定义） • int width：纹理宽度 • int height：纹理高度
返回值	以字节为单位返回纹理大小
备注	

3.7.6.17 hal_gfx_rgba

表 3-91 hal_gfx_rgba接口

函数原型	uint32_t hal_gfx_rgba(unsigned char R, unsigned char G, unsigned char B, unsigned char A)
功能说明	返回RGBA值
输入参数	• unsigned char R：红色 • unsigned char G：绿色 • unsigned char B：蓝色 • unsigned char A：透明度
返回值	返回RGBA值
备注	

3.7.6.18 hal_gfx_premultiply_rgba

表 3-92 hal_gfx_premultiply_rgba接口

函数原型	uint32_t hal_gfx_premultiply_rgba(uint32_t rgba)
功能说明	预乘RGB通道和Alpha通道
输入参数	uint32_t rgba：RGBA颜色
返回值	预乘RGBA颜色
备注	

3.7.6.19 hal_gfx_init

表 3-93 hal_gfx_init接口

函数原型	int hal_gfx_init(void)
功能说明	初始化hal_gfx库
输入参数	无
返回值	错误返回负值
备注	

3.7.6.20 hal_gfx_bind_src_tex

表 3-94 hal_gfx_bind_src_tex接口

函数原型	void hal_gfx_bind_src_tex(uintptr_t baseaddr_phys, uint32_t width, uint32_t height, hal_gfx_tex_format_t format, int32_t stride, hal_gfx_tex_mode_t mode)
功能说明	绑定前景（源）纹理单元
输入参数	• uintptr_t baseaddr_phys：纹理地址 • uint32_t width：纹理宽度 • uint32_t height：纹理高度 • hal_gfx_tex_format_t format：纹理颜色格式 • int32_t stride：纹理步长（<0待计算） • hal_gfx_tex_mode_t mode：要使用的环绕/重复模式（可选择HAL_GFX_FILTER和HAL_GFX_TEX相关的宏定义）
返回值	无
备注	HAL_GFX_TEX1

3.7.6.21 hal_gfx_bind_src2_tex

表 3-95 hal_gfx_bind_src2_tex接口

函数原型	void hal_gfx_bind_src2_tex(uintptr_t baseaddr_phys, uint32_t width, uint32_t height, hal_gfx_tex_format_t format, int32_t stride, hal_gfx_tex_mode_t mode)
功能说明	绑定后景（源）纹理单元
输入参数	• uintptr_t baseaddr_phys：纹理地址 • uint32_t width：纹理宽度 • uint32_t height：纹理高度 • hal_gfx_tex_format_t format：纹理颜色格式 • int32_t stride：纹理步长（<0待计算） • hal_gfx_tex_mode_t mode：要使用的环绕/重复模式（可选择HAL_GFX_FILTER和HAL_GFX_TEX相关的宏定义）
返回值	无

备注	HAL_GFX_TEX2

3.7.6.22 hal_gfx_bind_dst_tex

表 3-96 hal_gfx_bind_dst_tex接口

函数原型	void hal_gfx_bind_dst_tex(uintptr_t baseaddr_phys, uint32_t width, uint32_t height, hal_gfx_tex_format_t format, int32_t stride)
功能说明	绑定目标纹理单元
输入参数	• uintptr_t baseaddr_phys：纹理地址 • uint32_t width：纹理宽度 • uint32_t height：纹理高度 • hal_gfx_tex_format_t format：纹理颜色格式 • int32_t stride：纹理步长（<0待计算）
返回值	无
备注	HAL_GFX_TEX0

3.7.6.23 hal_gfx_bind_depth_buffer

表 3-97 hal_gfx_bind_depth_buffer接口

函数原型	void hal_gfx_bind_depth_buffer(uintptr_t baseaddr_phys, uint32_t width, uint32_t height)
功能说明	绑定深度缓冲区
输入参数	• uintptr_t baseaddr_phys：缓冲区地址 • uint32_t width：缓冲区宽度 • uint32_t height：缓冲区高度
返回值	无
备注	

3.7.6.24 hal_gfx_clear

表 3-98 hal_gfx_clear接口

函数原型	void hal_gfx_clear(uint32_t rgba8888)
功能说明	用颜色清除目标纹理
输入参数	uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.25 hal_gfx_clear_depth

表 3-99 hal_gfx_clear_depth接口

函数原型	void hal_gfx_clear_depth(uint32_t val)
功能说明	清除深度缓冲区
输入参数	uint32_t val：指定值
返回值	无
备注	

3.7.6.26 hal_gfx_draw_line

表 3-100 hal_gfx_draw_line接口

函数原型	void hal_gfx_draw_line(int x0, int y0, int x1, int y1, uint32_t rgba8888)
功能说明	绘制直线
输入参数	• int x0：起始坐标x • int y0：起始坐标y • int x1：终点坐标x • int y1：终点坐标y • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.27 hal_gfx_draw_line_aa

表 3-101 hal_gfx_draw_line_aa接口

函数原型	void hal_gfx_draw_line_aa(float x0, float y0, float x1, float y1, float w, uint32_t rgba8888)
功能说明	使用抗锯齿（如果可用）和指定宽度绘制直线
输入参数	• float x0：起始坐标x • float y0：起始坐标y • float x1：终点坐标x • float y1：终点坐标y • float w：线宽 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.28 hal_gfx_draw_circle

表 3-102 hal_gfx_draw_circle接口

函数原型	void hal_gfx_draw_circle(int x, int y, int r, uint32_t rgba8888)
功能说明	绘制圆
输入参数	• int x：圆心坐标x • int y：圆心坐标y • int r：半径 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.29 hal_gfx_draw_circle_aa

表 3-103 hal_gfx_draw_circle_aa接口

函数原型	void hal_gfx_draw_circle_aa(float x, float y, float r, float w, uint32_t rgba8888)
功能说明	使用抗锯齿（如果可用）和指定宽度绘制圆
输入参数	• float x：圆心坐标x • float y：圆心坐标y • float r：半径 • float w：线宽 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.30 hal_gfx_draw_rounded_rect

表 3-104 hal_gfx_draw_rounded_rect接口

函数原型	void hal_gfx_draw_rounded_rect(int x0, int y0, int w, int h, int r, uint32_t rgba8888)
功能说明	绘制带圆角的矩形
输入参数	• int x0：矩形左上顶点的x坐标 • int y0：矩形左上顶点的y坐标 • int w：矩形的宽度 • int h：矩形的高度 • int r：拐角半径 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.31 hal_gfx_draw_rect

表 3-105 hal_gfx_draw_rect接口

函数原型	void hal_gfx_draw_rect(int x, int y, int w, int h, uint32_t rgba8888)
功能说明	绘制矩形
输入参数	• int x：矩形左上顶点的x坐标 • int y：矩形左上顶点的y坐标 • int w：矩形的宽度 • int h：矩形的高度 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.32 hal_gfx_fill_circle

表 3-106 hal_gfx_fill_circle接口

函数原型	void hal_gfx_fill_circle(int x, int y, int r, uint32_t rgba8888)
功能说明	用颜色填充一个圆
输入参数	• int x：圆心坐标x • int y：圆心坐标y • int r：半径 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.33 hal_gfx_fill_circle_aa

表 3-107 hal_gfx_fill_circle_aa接口

函数原型	void hal_gfx_fill_circle_aa(float x, float y, float r, uint32_t rgba8888)
功能说明	使用抗锯齿（如果可用）用颜色填充一个圆
输入参数	• float x：圆心坐标x • float y：圆心坐标y • float r：半径 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.34 hal_gfx_fill_triangle

表 3-108 hal_gfx_fill_triangle接口

函数原型	void hal_gfx_fill_triangle(int x0, int y0, int x1, int y1, int x2, int y2, uint32_t rgba8888)

功能说明	用颜色填充三角形
输入参数	• int x0：三角形第一个顶点的x坐标 • int y0：三角形第一个顶点的y坐标 • int x1：三角形第二个顶点的x坐标 • int y1：三角形第二个顶点的y坐标 • int x2：三角形第三个顶点的x坐标 • int y2：三角形第三个顶点的y坐标 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.35 hal_gfx_fill_rounded_rect

表 3-109 hal_gfx_fill_rounded_rect接口

函数原型	void hal_gfx_fill_rounded_rect(int x0, int y0, int w, int h, int r, uint32_t rgba8888)
功能说明	用颜色填充一个圆角的矩形
输入参数	• int x0：矩形左上顶点的x坐标 • int y0：矩形左上顶点的y坐标 • int w：矩形的宽度 • int h：矩形的高度 • int r：拐角半径 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.36 hal_gfx_fill_rect

表 3-110 hal_gfx_fill_rect接口

函数原型	void hal_gfx_fill_rect(int x, int y, int w, int h, uint32_t rgba8888)
功能说明	用颜色填充矩形
输入参数	• int x：矩形左上顶点的x坐标 • int y：矩形左上顶点的y坐标 • int w：矩形的宽度 • int h：矩形的高度 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.37 hal_gfx_fill_quad

表 3-111 hal_gfx_fill_quad接口

函数原型	void hal_gfx_fill_quad(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3, uint32_t rgba8888)
功能说明	用颜色填充四边形
输入参数	• int x0：四边形第一个顶点的x坐标 • int y0：四边形第一个顶点的y坐标 • int x1：四边形第二个顶点的x坐标 • int y1：四边形第二个顶点的y坐标 • int x2：四边形第三个顶点的x坐标 • int y2：四边形第三个顶点的y坐标 • int x3：四边形第四个顶点的x坐标 • int y3：四边形第四个顶点的y坐标 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.38 hal_gfx_fill_rect_f

表 3-112 hal_gfx_fill_rect_f接口

函数原型	void hal_gfx_fill_rect_f(float x, float y, float w, float h, uint32_t rgba8888)
功能说明	用颜色填充矩形（浮点坐标）
输入参数	• float x：矩形左上顶点的x坐标 • float y：矩形左上顶点的y坐标 • float w：矩形的宽度 • float h：矩形的高度 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.39 hal_gfx_fill_quad_f

表 3-113 hal_gfx_fill_quad_f接口

函数原型	void hal_gfx_fill_quad_f(float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, uint32_t rgba8888)
功能说明	用颜色填充四边形（浮点坐标）
输入参数	• float x0：四边形第一个顶点的x坐标 • float y0：四边形第一个顶点的y坐标 • float x1：四边形第二个顶点的x坐标 • float y1：四边形第二个顶点的y坐标 • float x2：四边形第三个顶点的x坐标 • float y2：四边形第三个顶点的y坐标 • float x3：四边形第四个顶点的x坐标

	• float y3：四边形第四个顶点的y坐标 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.40 hal_gfx_fill_triangle_f

表 3-114 hal_gfx_fill_triangle_f接口

函数原型	void hal_gfx_fill_triangle_f(float x0, float y0, float x1, float y1, float x2, float y2, uint32_t rgba8888)
功能说明	用颜色填充三角形（浮点坐标）
输入参数	• float x0：三角形第一个顶点的x坐标 • float y0：三角形第一个顶点的y坐标 • float x1：三角形第二个顶点的x坐标 • float y1：三角形第二个顶点的y坐标 • float x2：三角形第三个顶点的x坐标 • float y2：三角形第三个顶点的y坐标 • uint32_t rgba8888：RGBA颜色
返回值	无
备注	

3.7.6.41 hal_gfx_blit

表 3-115 hal_gfx_blit接口

函数原型	void hal_gfx_blit (int x, int y)
功能说明	位块传输源纹理到目标纹理
输入参数	• int x：目标纹理的x坐标 • int y：目标纹理的y坐标
返回值	无
备注	快速位块传输函数目标的宽度和高度取决于前景（源）纹理（HAL_GFX_TEX1） 当背景纹理（HAL_GFX_TEX2）生效时，背景纹理的坐标始终在（0,0）处，而不会根据设定的目标纹理的x坐标和y坐标进行偏移

3.7.6.42 hal_gfx_blit_rounded

表 3-116 hal_gfx_blit_rounded接口

函数原型	void hal_gfx_blit_rounded (int x, int y, int r)
功能说明	位块传输源纹理到带有圆角的目标纹理
输入参数	• int x：目标纹理的x坐标 • int y：目标纹理的y坐标 • int r：拐角半径
返回值	无

备注	

3.7.6.43 hal_gfx_blit_rect

表 3-117 hal_gfx_blit_rect接口

函数原型	void hal_gfx_blit_rect (int x, int y, int w, int h)
功能说明	位块传输源纹理到目标的指定矩形
输入参数	• int x：目标纹理的x坐标 • int y：目标纹理的y坐标 • int w：目标纹理的宽度 • int h：目标纹理的高度
返回值	无
备注	

3.7.6.44 hal_gfx_blit_subrect

表 3-118 hal_gfx_blit_subrect接口

函数原型	void hal_gfx_blit_subrect(int dst_x, int dst_y, int w, int h, int src_x, int src_y)
功能说明	将源纹理的一部分位块传输到目标的指定矩形
输入参数	• int dst_x：目标纹理的x坐标 • int dst_y：目标纹理的y坐标 • int w：目标纹理的宽度 • int h：目标纹理的高度 • int src_x：源纹理的x坐标 • int src_y：源纹理的y坐标
返回值	无
备注	

3.7.6.45 hal_gfx_blit_rect_fit

表 3-119 hal_gfx_blit_rect_fit接口

函数原型	void hal_gfx_blit_rect_fit(int x, int y, int w, int h)
功能说明	位块传输源纹理到目标纹理，适合（缩放）纹理到指定的矩形
输入参数	• int x：目标纹理的x坐标 • int y：目标纹理的y坐标 • int w：目标纹理的宽度 • int h：目标纹理的高度
返回值	无
备注	

3.7.6.46 hal_gfx_blit_subrect_fit

表 3-120 hal_gfx_blit_subrect_fit接口

函数原型	void hal_gfx_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)
功能说明	将源纹理的一部分位块传输到目标，适合（缩放）纹理到指定的矩形
输入参数	• int dst_x：目标纹理的x坐标 • int dst_y：目标纹理的y坐标 • int dst_w：目标纹理的宽度 • int dst_h：目标纹理的高度 • int src_x：源纹理的x坐标 • int src_y：源纹理的y坐标 • int src_w：源纹理的宽度 • int src_h：源纹理的高度
返回值	无
备注	

3.7.6.47 hal_gfx_blit_rotate_pivot

表 3-121 hal_gfx_blit_rotate_pivot接口

函数原型	void hal_gfx_blit_rotate_pivot(float cx, float cy, float px, float py, float degrees_cw)
功能说明	围绕枢轴点旋转位块传输源纹理
输入参数	• float cx：目标纹理旋转中心x坐标 • float cy：目标纹理旋转中心y坐标 • float px：源枢轴点x坐标 • float py：源枢轴点y坐标 • float degrees_cw：[0, 360]范围内顺时针旋转的度数
返回值	无
备注	

3.7.6.48 hal_gfx_blit_rotate_pivot_scale

表 3-122 hal_gfx_blit_rotate_pivot_scale接口

函数原型	void hal_gfx_blit_rotate_pivot_scale(float cx, float cy, float px, float py, float degrees_cw, float scale)
功能说明	围绕枢轴点旋转位块传输源纹理（带比例）
输入参数	• float cx：目标纹理旋转中心x坐标 • float cy：目标纹理旋转中心y坐标 • float px：源枢轴点x坐标 • float py：源枢轴点y坐标 • float degrees_cw：[0, 360]范围内顺时针旋转的度数

	• float scale：比例
返回值	无
备注	

3.7.6.49 hal_gfx_blit_rotate

表 3-123 hal_gfx_blit_rotate接口

函数原型	void hal_gfx_blit_rotate(int x, int y, uint32_t rotation)
功能说明	将源纹理旋转并位块传输到目标
输入参数	• int x：目标纹理的x坐标 • int y：目标纹理的y坐标 • uint32_t rotation：旋转模式（可选择HAL_GFX_ROT和HAL_GFX_MIR相关的宏定义）
返回值	无
备注	

3.7.6.50 hal_gfx_blit_rotate_partial

表 3-124 hal_gfx_blit_rotate_partial接口

函数原型	void hal_gfx_blit_rotate_partial(int sx, int sy, int sw, int sh, int x, int y, uint32_t rotation)
功能说明	旋转和位块传输部分源纹理到目标
输入参数	• int sx：源纹理的左上x坐标 • int sy：源纹理的左上y坐标 • int sw：源纹理部分区域的宽度 • int sh：源纹理部分区域的高度 • int x：目标纹理的x坐标 • int y：目标纹理的y坐标 • uint32_t rotation：旋转模式（可选择HAL_GFX_ROT和HAL_GFX_MIR相关的宏定义）
返回值	无
备注	

3.7.6.51 hal_gfx_blit_tri_fit

表 3-125 hal_gfx_blit_tri_fit接口

函数原型	void hal_gfx_blit_tri_fit (float dx0, float dy0, int v0, float dx1, float dy1, int v1, float dx2, float dy2, int v2)
功能说明	位块传输源纹理到目标纹理，使纹理适合指定的三角形

输入参数	• float dx0：三角形第一个顶点的x坐标 • float dy0：三角形第一个顶点的y坐标 • int v0：[0, 3]表示源纹理哪个角至第一顶点 • float dx1：三角形第二个顶点的x坐标 • float dy1：三角形第二个顶点的y坐标 • int v1：[0, 3]表示源纹理哪个角至第二顶点 • float dx2：三角形第三个顶点的x坐标 • float dy2：三角形第三个顶点的y坐标 • int v2：[0, 3]表示源纹理哪个角至第三顶点
返回值	无
备注	

3.7.6.52 hal_gfx_blit_tri_uv

表 3-126 hal_gfx_blit_tri_uv接口

函数原型	void hal_gfx_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)
功能说明	位块传输源纹理的三角形部分到三角形目标区域
输入参数	• float dx0：目标三角形第一个顶点的x坐标 • float dy0：目标三角形第一个顶点的y坐标 • float dw0：目标三角形第一个顶点的w坐标 • float dx1：目标三角形第二个顶点的x坐标 • float dy1：目标三角形第二个顶点的y坐标 • float dw1：目标三角形第二个顶点的w坐标 • float dx2：目标三角形第三个顶点的x坐标 • float dy2：目标三角形第三个顶点的y坐标 • float dw2：目标三角形第三个顶点的w坐标 • float sx0：源三角形第一个顶点的x坐标 • float sy0：源三角形第一个顶点的y坐标 • float sx1：源三角形第二个顶点的x坐标 • float sy1：源三角形第二个顶点的y坐标 • float sx2：源三角形第三个顶点的x坐标 • float sy2：源三角形第三个顶点的y坐标
返回值	无
备注	

3.7.6.53 hal_gfx_blit_quad_fit

表 3-127 hal_gfx_blit_quad_fit接口

函数原型	void hal_gfx_blit_quad_fit (float dx0, float dy0, float dx1, float dy1, float dx2, float dy2, float dx3, float dy3)
功能说明	位块传输源纹理到目标纹理，将纹理拟合到指定的四边形
输入参数	• float dx0：四边形第一个顶点的x坐标 • float dy0：四边形第一个顶点的y坐标 • float dx1：四边形第二个顶点的x坐标 • float dy1：四边形第二个顶点的y坐标 • float dx2：四边形第三个顶点的x坐标 • float dy2：四边形第三个顶点的y坐标 • float dx3：四边形第四个顶点的x坐标 • float dy3：四边形第四个顶点的y坐标
返回值	无
备注	

3.7.6.54 hal_gfx_blit_subrect_quad_fit

表 3-128 hal_gfx_blit_subrect_quad_fit接口

函数原型	void hal_gfx_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)
功能说明	位块传输源纹理到目标纹理，将纹理的矩形区域拟合到指定的四边形
输入参数	• float dx0：四边形第一个顶点的x坐标 • float dy0：四边形第一个顶点的y坐标 • float dx1：四边形第二个顶点的x坐标 • float dy1：四边形第二个顶点的y坐标 • float dx2：四边形第三个顶点的x坐标 • float dy2：四边形第三个顶点的y坐标 • float dx3：四边形第四个顶点的x坐标 • float dy3：四边形第四个顶点的y坐标 • int sx：源纹理矩形区域左上角的x坐标 • int sy：源纹理矩形区域左上角的y坐标 • int sw：源纹理矩形区域的宽度 • int sh：源纹理矩形区域的高度

返回值	无
备注	

3.7.7 hal_gfx_hal.h

3.7.7.1 hal_gfx_sys_init

表 3-129 hal_gfx_sys_init接口

函数原型	int32_t hal_gfx_sys_init(void)
功能说明	初始化系统，在hal_gfx_init()中调用
输入参数	无
返回值	非零出错
备注	

3.7.7.2 hal_gfx_wait_irq

表 3-130 hal_gfx_wait_irq接口

函数原型	int hal_gfx_wait_irq(void)
功能说明	等待来自GPU的中断
输入参数	无
返回值	非零出错
备注	

3.7.7.3 hal_gfx_wait_irq_cl

表 3-131 hal_gfx_wait_irq_cl接口

函数原型	int hal_gfx_wait_irq_cl(int cl_id)
功能说明	等待命令列表完成
输入参数	int cl_id：命令列表ID
返回值	非零出错
备注	

3.7.7.4 hal_gfx_wait_irq_brk

表 3-132 hal_gfx_wait_irq_brk接口

函数原型	int hal_gfx_wait_irq_brk(int brk_id)
功能说明	等待断点
输入参数	int brk_id：断点ID
返回值	非零出错

备注	

3.7.7.5 hal_gfx_reg_read

表 3-133 hal_gfx_reg_read接口

函数原型	uint32_t hal_gfx_reg_read(uint32_t reg)
功能说明	读硬件寄存器
输入参数	uint32_t reg：寄存器
返回值	寄存器值
备注	

3.7.7.6 hal_gfx_reg_write

表 3-134 hal_gfx_reg_write接口

函数原型	void hal_gfx_reg_write(uint32_t reg, uint32_t value)
功能说明	写硬件寄存器
输入参数	• uint32_t reg：寄存器 • uint32_t value：写入的数值
返回值	无
备注	

3.7.7.7 hal_gfx_buffer_create

表 3-135 hal_gfx_buffer_create接口

函数原型	hal_gfx_buffer_t hal_gfx_buffer_create(int size)
功能说明	创建内存缓冲区
输入参数	int size：缓冲区大小（以字节为单位）
返回值	hal_gfx_buffer_t结构体
备注	

3.7.7.8 hal_gfx_buffer_create_pool

表 3-136 hal_gfx_buffer_create_pool接口

函数原型	hal_gfx_buffer_t hal_gfx_buffer_create_pool(int pool, int size)
功能说明	在特定池中创建内存缓冲区
输入参数	• int pool：所需内存池的ID • int size：缓冲区大小（以字节为单位）
返回值	hal_gfx_buffer_t结构体
备注	

3.7.7.9 hal_gfx_buffer_map

表 3-137 hal_gfx_buffer_map接口

函数原型	void *hal_gfx_buffer_map(hal_gfx_buffer_t *bo)
功能说明	映射缓冲区
输入参数	hal_gfx_buffer_t *bo：指向缓冲区结构的指针
返回值	缓冲区的虚拟指针（与bo->base_virt相同）
备注	

3.7.7.10 hal_gfx_buffer_unmap

表 3-138 hal_gfx_buffer_unmap接口

函数原型	void hal_gfx_buffer_unmap(hal_gfx_buffer_t *bo)
功能说明	解除映射缓冲区
输入参数	hal_gfx_buffer_t *bo：指向缓冲区结构的指针
返回值	无
备注	

3.7.7.11 hal_gfx_buffer_destroy

表 3-139 hal_gfx_buffer_destroy接口

函数原型	void hal_gfx_buffer_destroy(hal_gfx_buffer_t *bo)
功能说明	销毁/释放缓冲区
输入参数	hal_gfx_buffer_t *bo：指向缓冲区结构的指针
返回值	无
备注	

3.7.7.12 hal_gfx_buffer_phys

表 3-140 hal_gfx_buffer_phys接口

函数原型	uintptr_t hal_gfx_buffer_phys(hal_gfx_buffer_t *bo)
功能说明	获取给定缓冲区的物理基地址
输入参数	hal_gfx_buffer_t *bo：指向缓冲区结构的指针
返回值	给定缓冲区的物理基地址
备注	

3.7.7.13 hal_gfx_buffer_flush

表 3-141 hal_gfx_buffer_flush接口

函数原型	void hal_gfx_buffer_flush(hal_gfx_buffer_t * bo)
功能说明	将缓冲区从缓存写回到主存
输入参数	hal_gfx_buffer_t * bo：指向缓冲区结构的指针
返回值	无
备注	

3.7.7.14 hal_gfx_host_malloc

表 3-142 hal_gfx_host_malloc接口

函数原型	void *hal_gfx_host_malloc(size_t size)
功能说明	分配内存供CPU使用
输入参数	size_t size：申请的大小（以字节为单位）
返回值	指向已分配内存的指针（虚拟）
备注	

3.7.7.15 hal_gfx_host_free

表 3-143 hal_gfx_host_free接口

函数原型	void hal_gfx_host_free(void *ptr)
功能说明	释放使用hal_gfx_host_malloc()分配的内存
输入参数	void *ptr：指向已分配内存的指针（虚拟）
返回值	无
备注	

3.7.7.16 hal_gfx_rb_init

表 3-144 hal_gfx_rb_init接口

函数原型	int hal_gfx_rb_init(hal_gfx_ringbuffer_t *rb, int reset)
功能说明	初始化环形缓冲区，应该从hal_gfx_sys_init()内部调用
输入参数	• hal_gfx_ringbuffer_t *rb：指向hal_gfx_ring_buffer_t结构的指针 • int reset：如果非零，则重置环形缓冲区
返回值	负数出错
备注	

3.7.7.17 hal_gfx_mutex_lock

表 3-145 hal_gfx_mutex_lock接口

函数原型	int hal_gfx_mutex_lock(int mutex_id)
功能说明	多个进程/线程的互斥锁
输入参数	int mutex_id：MUTEX_RB or MUTEX_MALLOC
返回值	非零出错
备注	

3.7.7.18 hal_gfx_mutex_unlock

表 3-146 hal_gfx_mutex_unlock接口

函数原型	int hal_gfx_mutex_unlock(int mutex_id)
功能说明	解除多个进程/线程的互斥锁
输入参数	int mutex_id：MUTEX_RB or MUTEX_MALLOC
返回值	非零出错
备注	

3.7.8 hal_gfx_interpolators.h

3.7.8.1 hal_gfx_interpolate_rect_colors

表 3-147 hal_gfx_interpolate_rect_colors接口

函数原型	void hal_gfx_interpolate_rect_colors(int x0, int y0, int w, int h, color_var_t* col0, color_var_t* col1, color_var_t* col2)
功能说明	为矩形插入颜色渐变
输入参数	• int x0：矩形左上顶点的x坐标 • int y0：矩形左上顶点的y坐标 • int w：矩形的宽度 • int h：矩形的高度 • color_var_t* col0：第一个顶点的颜色 • color_var_t* col1：第二个顶点的颜色 • color_var_t* col2：第三个顶点的颜色
返回值	无
备注	需使能渐变hal_gfx_enable_gradient()

3.7.8.2 hal_gfx_interpolate_tri_colors

表 3-148 hal_gfx_interpolate_tri_colors接口

函数原型	void hal_gfx_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)
功能说明	为三角形插入颜色渐变
输入参数	• float x0：三角形第一个顶点的x坐标 • float y0：三角形第一个顶点的y坐标 • float x1：三角形第二个顶点的x坐标 • float y1：三角形第二个顶点的y坐标 • float x2：三角形第三个顶点的x坐标 • float y2：三角形第三个顶点的y坐标 • color_var_t* col0：第一个顶点的颜色 • color_var_t* col1：第二个顶点的颜色 • color_var_t* col2：第三个顶点的颜色
返回值	无
备注	需使能渐变hal_gfx_enable_gradient()

3.7.8.3 hal_gfx_interpolate_tri_depth

表 3-149 hal_gfx_interpolate_tri_depth接口

函数原型	void hal_gfx_interpolate_tri_depth(float x0, float y0, float z0, float x1, float y1, float z1, float x2, float y2, float z2)
功能说明	插入三角形的深度缓冲区值
输入参数	• float x0：三角形第一个顶点的x坐标 • float y0：三角形第一个顶点的y坐标 • float z0：三角形第一个顶点的z坐标 • float x1：三角形第二个顶点的x坐标 • float y1：三角形第二个顶点的y坐标 • float z1：三角形第二个顶点的z坐标 • float x2：三角形第三个顶点的x坐标 • float y2：三角形第三个顶点的y坐标 • float z2：三角形第三个顶点的z坐标
返回值	无
备注	需使能深度hal_gfx_enable_depth()

3.7.8.4 hal_gfx_interpolate_tx_ty

表 3-150 hal_gfx_interpolate_tx_ty接口

函数原型	void hal_gfx_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)
功能说明	插入三角形的纹理值
输入参数	• float x0：三角形第一个顶点的x坐标 • float y0：三角形第一个顶点的y坐标 • float w0：三角形第一个顶点的w坐标 • float tx0：三角形第一个顶点的x纹理坐标 • float ty0：三角形第一个顶点的y纹理坐标 • float x1：三角形第二个顶点的x坐标 • float y1：三角形第二个顶点的y坐标 • float w1：三角形第二个顶点的w坐标 • float tx1：三角形第二个顶点的x纹理坐标 • float ty1：三角形第二个顶点的y纹理坐标 • float x2：三角形第三个顶点的x坐标 • float y2：三角形第三个顶点的y坐标 • float w2：三角形第三个顶点的w坐标 • float tx2：三角形第三个顶点的x纹理坐标 • float ty2：三角形第三个顶点的y纹理坐标 • int tex_width：纹理宽度 • int tex_height：纹理高度
返回值	无
备注	

3.7.9 hal_gfx_math.h

3.7.9.1 宏定义

表 3-151 hal_gfx_math.h宏定义

宏名称	描述
HAL_GFX_E	e
HAL_GFX_LOG2E	log2(e)
HAL_GFX_LOG10E	log10(e)
HAL_GFX_LN2	ln(2)
HAL_GFX_LN10	ln(10)
HAL_GFX_PI	pi
HAL_GFX_PI_2	pi/2
HAL_GFX_PI_4	pi/4
HAL_GFX_1_PI	1/pi
HAL_GFX_2_PI	2/pi
HAL_GFX_2_SQRTPI	2/sqrt(pi)
HAL_GFX_SQRT2	sqrt(2)

宏名称	描述
HAL_GFX_SQRT1_2	1/sqrt(2)
hal_gfx_min2(a,b)	求两个值的最小值
hal_gfx_max2(a,b)	求两个值的最大值
hal_gfx_clamp(val, min, max)	钳位值
hal_gfx_abs(a)	计算int类型的绝对值
hal_gfx_absf(a)	计算float类型的绝对值
hal_gfx_floats_equal(x, y)	比较两个浮点数
hal_gfx_float_is_zero(x)	检查值是否为零
hal_gfx_deg_to_rad(d)	将度数转换为弧度
hal_gfx_rad_to_deg(r)	将弧度转换为度数
hal_gfx_i2fx(a)	将整数转换为16.16定点
hal_gfx_floor(f)	向下取整
hal_gfx_ceil(f)	向上取整

3.7.9.2 hal_gfx_sin

表 3-152 hal_gfx_sin接口

函数原型	float hal_gfx_sin(float angle_degrees)
功能说明	计算sin
输入参数	float angle_degrees：以度为单位的角度
返回值	返回给定角度的正弦
备注	

3.7.9.3 hal_gfx_cos

表 3-153 hal_gfx_cos接口

函数原型	float hal_gfx_cos(float angle_degrees)
功能说明	计算cos
输入参数	float angle_degrees：以度为单位的角度
返回值	返回给定角度的余弦
备注	

3.7.9.4 hal_gfx_tan

表 3-154 hal_gfx_tan接口

函数原型	float hal_gfx_tan(float angle_degrees)

功能说明	计算tan
输入参数	float angle_degrees：以度为单位的角度
返回值	返回给定角度的正切
备注	

3.7.9.5 hal_gfx_pow

表 3-155 hal_gfx_pow接口

函数原型	float hal_gfx_pow(float x, float y)
功能说明	计算x^y的粗略近似值
输入参数	• float x：基值，必须是非负数 • float y：幂
返回值	x^y结果
备注	

3.7.9.6 hal_gfx_sqrt

表 3-156 hal_gfx_sqrt接口

函数原型	float hal_gfx_sqrt(float x)
功能说明	计算x的平方根的粗略近似值
输入参数	float x：x值，必须是非负数
返回值	x的平方根
备注	

3.7.9.7 hal_gfx_atan

表 3-157 hal_gfx_atan接口

函数原型	float hal_gfx_atan(float x)
功能说明	计算x的反正切的浮点近似值
输入参数	float x：x值
返回值	x的反正切（角度），以度为单位
备注	

3.7.9.8 hal_gfx_f2fx

表 3-158 hal_gfx_f2fx接口

函数原型	int hal_gfx_f2fx(float f)
功能说明	将浮点数转换为16.16定点数
输入参数	float f：要转换的值

返回值	16.16定点数
备注	

3.7.10 hal_gfx_transitions.h

3.7.10.1 枚举类型

表 3-159 hal_gfx_transitions.h枚举类型

枚举类型	成员	描述
hal_gfx_transition_t	HAL_GFX_TRANS_LINEAR_H	水平线性过渡
	HAL_GFX_TRANS_CUBE_H	水平立方过渡
	HAL_GFX_TRANS_INNERCUBE_H	内部立方体水平过渡
	HAL_GFX_TRANS_STACK_H	水平堆叠过渡
	HAL_GFX_TRANS_LINEAR_V	垂直线性过渡
	HAL_GFX_TRANS_CUBE_V	垂直立方过渡
	HAL_GFX_TRANS_INNERCUBE_V	内部立方体垂直过渡
	HAL_GFX_TRANS_STACK_V	垂直堆叠过渡
	HAL_GFX_TRANS_FADE	淡化过渡
	HAL_GFX_TRANS_FADE_ZOOM	淡化-缩放过渡
	HAL_GFX_TRANS_COVER	覆盖过渡
	HAL_GFX_TRANS_MAX	参数检查
	HAL_GFX_TRANS_NONE	无效果

3.7.10.2 hal_gfx_transition

表 3-160 hal_gfx_transition接口

函数原型	void hal_gfx_transition(hal_gfx_transition_t effect, hal_gfx_tex_t initial, hal_gfx_tex_t final, uint32_t blending_mode, float step, int width, int height)
功能说明	从“初始”纹理过渡到“最终”纹理，当'step'为0或1时转换完成
输入参数	• hal_gfx_transition_t effect：过渡效果 • hal_gfx_tex_t initial：初始纹理 • hal_gfx_tex_t final：最终纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的过渡步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.3 hal_gfx_transition_linear_hor

表 3-161 hal_gfx_transition_linear_hor接口

函数原型	void hal_gfx_transition_linear_hor(hal_gfx_tex_t left, hal_gfx_tex_t right, uint32_t blending_mode, float step, int width)
功能说明	水平线性过渡，0->1从右向左，1->0从左向右
输入参数	• hal_gfx_tex_t left：左侧的纹理 • hal_gfx_tex_t right：右侧的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度
返回值	无
备注	

3.7.10.4 hal_gfx_transition_linear_ver

表 3-162 hal_gfx_transition_linear_ver接口

函数原型	void hal_gfx_transition_linear_ver(hal_gfx_tex_t up, hal_gfx_tex_t down, uint32_t blending_mode, float step, int height)
功能说明	垂直线性过渡，0->1从上向下，1->0从下向上
输入参数	• hal_gfx_tex_t up：顶部的纹理 • hal_gfx_tex_t down：底部的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int height：纹理高度
返回值	无
备注	

3.7.10.5 hal_gfx_transition_cube_hor

表 3-163 hal_gfx_transition_cube_hor接口

函数原型	void hal_gfx_transition_cube_hor(hal_gfx_tex_t left, hal_gfx_tex_t right, uint32_t blending_mode, float step, int width, int height)
功能说明	立方（纹理映射在立方体的外表面上）水平过渡，0->1从左向右，1->0从右向左
输入参数	• hal_gfx_tex_t left：左侧的纹理 • hal_gfx_tex_t right：右侧的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无

备注	

3.7.10.6 hal_gfx_transition_cube_ver

表 3-164 hal_gfx_transition_cube_ver接口

函数原型	void hal_gfx_transition_cube_ver(hal_gfx_tex_t up, hal_gfx_tex_t down, uint32_t blending_mode, float step, int width, int height)
功能说明	立方（纹理映射在立方体的外表面上）垂直过渡，0->1从上向下，1->0从下向上
输入参数	• hal_gfx_tex_t up：顶部的纹理 • hal_gfx_tex_t down：底部的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.7 hal_gfx_transition_innercube_hor

表 3-165 hal_gfx_transition_innercube_hor接口

函数原型	void hal_gfx_transition_innercube_hor(hal_gfx_tex_t left, hal_gfx_tex_t right, uint32_t blending_mode, float step, int width, int height)
功能说明	内部立方体（纹理映射到立方体的内表面）水平过渡，0->1从左向右，1->0从右向左
输入参数	• hal_gfx_tex_t left：左侧的纹理 • hal_gfx_tex_t right：右侧的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.8 hal_gfx_transition_innercube_ver

表 3-166 hal_gfx_transition_innercube_ver接口

函数原型	void hal_gfx_transition_innercube_ver(hal_gfx_tex_t up, hal_gfx_tex_t down, uint32_t blending_mode, float step, int width, int height)
功能说明	内部立方体（纹理映射到立方体的内表面）垂直过渡，0->1从上向下，1->0从下向上
输入参数	• hal_gfx_tex_t up：顶部的纹理 • hal_gfx_tex_t down：底部的纹理 • uint32_t blending_mode：混合模式

	• float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.9 hal_gfx_transition_stack_hor

表 3-167 hal_gfx_transition_stack_hor接口

函数原型	void hal_gfx_transition_stack_hor(hal_gfx_tex_t left, hal_gfx_tex_t right, float step, int width, int height)
功能说明	水平堆叠过渡，0->1从左向右，1->0从右向左
输入参数	• hal_gfx_tex_t left：左侧的纹理 • hal_gfx_tex_t right：右侧的纹理 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.10 hal_gfx_transition_stack_ver

表 3-168 hal_gfx_transition_stack_ver接口

函数原型	void hal_gfx_transition_stack_ver(hal_gfx_tex_t up, hal_gfx_tex_t down, float step, int width, int height)
功能说明	垂直堆叠过渡，0->1从上向下，1->0从下向上
输入参数	• hal_gfx_tex_t up：顶部的纹理 • hal_gfx_tex_t down：底部的纹理 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.11 hal_gfx_transition_fade

表 3-169 hal_gfx_transition_fade接口

函数原型	void hal_gfx_transition_fade(hal_gfx_tex_t initial, hal_gfx_tex_t final, uint32_t blending_mode, float step, int width, int height)
功能说明	淡化过渡，初始纹理正在淡出，最终纹理正在淡入
输入参数	• hal_gfx_tex_t initial：初始的纹理

	• hal_gfx_tex_t final：最终的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.10.12 hal_gfx_transition_fade_zoom

表 3-170 hal_gfx_transition_fade_zoom接口

函数原型	void hal_gfx_transition_fade_zoom(hal_gfx_tex_t initial, hal_gfx_tex_t final, uint32_t blending_mode, float step, int width, int height)
功能说明	淡化-缩放过渡，初始纹理正在被缩放淡出，最终纹理正在被放大淡入
输入参数	• hal_gfx_tex_t initial：初始的纹理 • hal_gfx_tex_t final：最终的纹理 • uint32_t blending_mode：混合模式 • float step：[0.f , 1.f]范围内的当前步长 • int width：纹理宽度 • int height：纹理高度
返回值	无
备注	

3.7.11 hal_gfx_matrix4x4.h

3.7.11.1 hal_gfx_mat4x4_load_identity

表 3-171 hal_gfx_mat4x4_load_identity接口

函数原型	void hal_gfx_mat4x4_load_identity(hal_gfx_matrix4x4_t m)
功能说明	加载4 x 4特征矩阵
输入参数	hal_gfx_matrix4x4_t m：待加载矩阵
返回值	无
备注	

3.7.11.2 hal_gfx_mat4x4_mul

表 3-172 hal_gfx_mat4x4_mul接口

函数原型	void hal_gfx_mat4x4_mul(hal_gfx_matrix4x4_t m, hal_gfx_matrix4x4_t m_l, hal_gfx_matrix4x4_t m_r)
功能说明	将两个4 x 4矩阵相乘

输入参数	• hal_gfx_matrix4x4_t m：结果矩阵 • hal_gfx_matrix4x4_t m_l：左操作数 • hal_gfx_matrix4x4_t m_r：右操作数
返回值	无
备注	

3.7.11.3 hal_gfx_mat4x4_mul_vec

表 3-173 hal_gfx_mat4x4_mul_vec接口

函数原型	void hal_gfx_mat4x4_mul_vec(hal_gfx_matrix4x4_t m, float *x, float *y, float *z, float *w)
功能说明	将4 x 1向量与4 x 4矩阵相乘
输入参数	• hal_gfx_matrix4x4_t m：待相乘矩阵 • float *x：向量第一个元素 • float *y：向量第二个元素 • float *z：向量第三个元素 • float *w：向量第四个元素
返回值	无
备注	

3.7.11.4 hal_gfx_mat4x4_translate

表 3-174 hal_gfx_mat4x4_translate接口

函数原型	void hal_gfx_mat4x4_translate(hal_gfx_matrix4x4_t m, float tx, float ty, float tz)
功能说明	实现矩阵平移变换
输入参数	• hal_gfx_matrix4x4_t m：应用变换的矩阵 • float tx：X平移因子 • float ty：Y平移因子 • float tz：Z平移因子
返回值	无
备注	

3.7.11.5 hal_gfx_mat4x4_scale

表 3-175 hal_gfx_mat4x4_scale接口

函数原型	void hal_gfx_mat4x4_scale(hal_gfx_matrix4x4_t m, float sx, float sy, float sz)
功能说明	实现矩阵缩放变换
输入参数	• hal_gfx_matrix4x4_t m：应用变换的矩阵 • float sx：X比例因子 • float sy：Y比例因子 • float sz：Z比例因子
返回值	无

备注	

3.7.11.6 hal_gfx_mat4x4_rotate_X

表 3-176 hal_gfx_mat4x4_rotate_X接口

函数原型	void hal_gfx_mat4x4_rotate_X(hal_gfx_matrix4x4_t m, float angle_degrees)
功能说明	围绕X轴应用旋转变换
输入参数	• hal_gfx_matrix4x4_t m：应用变换的矩阵 • float angle_degrees：以度为单位旋转的角度
返回值	无
备注	

3.7.11.7 hal_gfx_mat4x4_rotate_Y

表 3-177 hal_gfx_mat4x4_rotate_Y接口

函数原型	void hal_gfx_mat4x4_rotate_Y(hal_gfx_matrix4x4_t m, float angle_degrees)
功能说明	围绕Y轴应用旋转变换
输入参数	• hal_gfx_matrix4x4_t m：应用变换的矩阵 • float angle_degrees：以度为单位旋转的角度
返回值	无
备注	

3.7.11.8 hal_gfx_mat4x4_rotate_Z

表 3-178 hal_gfx_mat4x4_rotate_Z接口

函数原型	void hal_gfx_mat4x4_rotate_Z(hal_gfx_matrix4x4_t m, float angle_degrees)
功能说明	围绕Z轴应用旋转变换
输入参数	• hal_gfx_matrix4x4_t m：应用变换的矩阵 • float angle_degrees：以度为单位旋转的角度
返回值	无
备注	

3.7.11.9 hal_gfx_mat4x4_load_perspective

表 3-179 hal_gfx_mat4x4_load_perspective接口

函数原型	void hal_gfx_mat4x4_load_perspective(hal_gfx_matrix4x4_t m, float fovy_degrees, float aspect, float nearVal, float farVal)
功能说明	设置透视投影矩阵
输入参数	• hal_gfx_matrix4x4_t m：4x4矩阵 • float fovy_degrees：以度为单位的视野

	• float aspect：宽高比 • float nearVal：从观察者到近剪裁平面的距离（始终为正） • float farVal：从观察者到远剪裁平面的距离（始终为正）
返回值	无
备注	

3.7.11.10 hal_gfx_mat4x4_load_ortho

表 3-180 hal_gfx_mat4x4_load_ortho接口

函数原型	void hal_gfx_mat4x4_load_ortho(hal_gfx_matrix4x4_t m, float left, float right, float bottom, float top, float nearVal, float farVal)
功能说明	设置正交投影矩阵
输入参数	• hal_gfx_matrix4x4_t m：4x4矩阵 • float left：左垂直剪裁平面 • float right：右垂直剪裁平面 • float bottom：底部水平剪裁平面 • float top：顶部水平剪裁平面 • float nearVal：从观察者到近剪裁平面的距离（始终为正） • float farVal：从观察者到远剪裁平面的距离（始终为正）
返回值	无
备注	

3.7.11.11 hal_gfx_mat4x4_load_ortho_2d

表 3-181 hal_gfx_mat4x4_load_ortho_2d接口

函数原型	void hal_gfx_mat4x4_load_ortho_2d(hal_gfx_matrix4x4_t m, float left, float right, float bottom, float top)
功能说明	设置二维正交投影矩阵
输入参数	• hal_gfx_matrix4x4_t m：4x4矩阵 • float left：左垂直剪裁平面 • float right：右垂直剪裁平面 • float bottom：底部水平剪裁平面 • float top：顶部水平剪裁平面
返回值	无
备注	

3.7.11.12 hal_gfx_mat4x4_obj_to_win_coords

表 3-182 hal_gfx_mat4x4_obj_to_win_coords接口

函数原型	int hal_gfx_mat4x4_obj_to_win_coords(hal_gfx_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)
功能说明	从对象坐标计算窗口坐标的便捷函数
输入参数	• hal_gfx_matrix4x4_t mvp：模型、视图和投影矩阵 • float x_orig：窗口左上角X坐标 • float y_orig：窗口左上角Y坐标 • int width：窗口宽度 • int height：窗口高度 • float nearVal：从观察者到近剪裁平面的距离（始终为正） • float farVal：从观察者到远剪裁平面的距离（始终为正） • float *x：X对象坐标 • float *y：Y对象坐标 • float *z：Z对象坐标 • float *w：W对象坐标
返回值	无
备注	

4 教程

本章介绍GPU应用典型的代码结构，并结合相关示例工程演示如何在GR5526上应用GPU模块实现图像处理。

4.1 GPU应用典型代码结构

GPU应用典型的代码结构主要分为四个部分：Graphics外设初始化、缓冲区配置、渲染任务与屏幕刷新、外设反初始化（可选）。

- Graphics外设初始化
  1. 初始化OSPI、PSRAM外设模块（默认使用OSPI访问内部PSRAM，若不使用内部PSRAM可不初始化相关外设）
  2. 初始化GPU模块
  3. 初始化DC模块
- 缓冲区配置
  1. Graphics内存管理初始化
  2. 根据需求，初始化合适的帧缓冲区
     (1) 为GPU配置全局Framebuffer参数
     (2) 为DC配置全局Framebuffer参数
  3. 根据需求，申请适当的Texture Memory存储纹理图案
  4. 初始化环形缓冲区，为命令列表提供缓冲空间
- 渲染任务与屏幕刷新
  1. 构建GPU的命令列表，并提交至GPU进行渲染
  2. 进入核心的渲染逻辑执行渲染任务，并实时将渲染结果刷新至屏幕上显示
     (1) 根据GPU中断获取渲染结果
     (2) 根据DC中断获取刷屏结果
- 外设反初始化（可选）
  1. 任务结束时反初始化Graphics相关外设模块
  2. 释放任务过程中申请的内存

4.2 控制宏参数

4.2.1 graphics_defs.h

*graphics_defs.h*头文件属于工程配置文件，每个示例工程独立，其位于：SDK_Folder\projects\peripheral\graphics\示例工程名\Src\application。

```
/* Set current screen configuration */
#define LCD_RES_CURRENT             LCD_RES_454

/* Set display enable */
#define HAL_GDC_DISPLAY_ENABLE      1u

/* Set TSCx compressed format enable */
#define TSCx_ENABLE                 1u

/* Set image show area */
#define SHOW_AREA_X                   240u
#define SHOW_AREA_Y                   240u

/* Set pixel depth size in byte (Determined by color format) */
#define PIXEL_DEPTH_BYTES             4u

/* Set pixel show size in byte (Determined by color format) */
#define PIXEL_SHOW_BYTES              3u

/* Set screen display area */
#define DISPLAY_LCD_RES_X             454u
#define DISPLAY_LCD_RES_Y             454u
```

4.2.2 graphics_sys_defs.h

*graphics_sys_defs.h*头文件属于graphics系统配置文件，其位于：SDK_Folder\components\graphics\gfx\porting。

```
/**< The GPU RING BUFFER SIZE. */
#define HAL_GFX_RING_BUFFER_SIZE            1024u

/**< The same to Pool id */
#define HAL_GFX_MEM_POOL_ASSETS             0

/**< Pool id, only set to 0 currently */
#define HAL_GFX_MEM_POOL_FB                 0

/**< the graphics (video) memory base address. */
#define VMEM_BASEADDR         ((uint32_t)(&s_graphics_memory_buffer[0]))

/**< The GPU max memory size for frame buffer. */
#ifndef VMEM_SIZE
```

```
    #define VMEM_SIZE                          (20*1024u)
#endif

/**< If enable, use the memory management of GPU. */
#ifndef USE_TSI_MALLOC
    #define USE_TSI_MALLOC
#endif

/**< if HAL_GFX_MULTI_MEM_POOLS is defined, use HAL_GFX_MULTI_MEM_POOLS_CNT
pools must be equal or less than 4. */
#ifndef HAL_GFX_MULTI_MEM_POOLS_CNT
    #define HAL_GFX_MULTI_MEM_POOLS_CNT     1
#endif
```

4.3 示例工程graphics_animation_effects

4.3.1 工程目录

graphics_animation_effects示例的源代码和工程文件位于：SDK_Folder\projects\peripheral\graphics\graphics_animation_effects，其中工程文件位于Keil_5文件夹。

双击打开*graphics_animation_effects.uvprojx*工程文件，在Keil中查看示例graphics_animation_effects工程目录结构，如图 4-1所示。

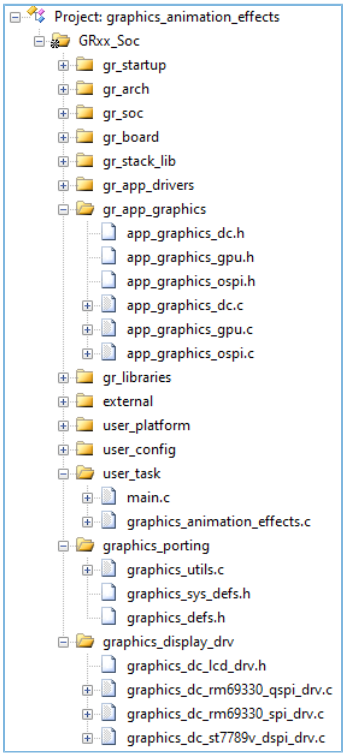


图 4-1 graphics_animation_effects工程目录

4.3.2 应用代码介绍

graphics_animation_effects示例工程主要展现GPU的图像过渡效果。

该工程已适配FreeRTOS，通过FreeRTOS的系统心跳可计算图像的FPS。开发者可以通过禁用ENV_USE_FREERTOS宏来禁用FreeRTOS，通过自定义函数来计算图像的FPS。

路径：工程目录下的Src\application\graphics_animation_effects.c

名称：int graphics_animation_effects_main(void)

该函数为graphics_animation_effects示例工程GPU应用程序的主体代码入口，该函数完成外设初始化、缓冲区配置、渲染任务与屏幕刷新等。

```
int graphics_animation_effects_main(void)
{
    /* Peripheral Initialization and Buffer Configuration */
    int ret = init();
    if (ret)
        {
        app_graphics_mem_free(s_fb.bo.base_virt);
        return ret;
        }

    /* Rendering Tasks and Screen Refresh */
    animation_effects_loop();

    /* Destroy Command List and Memory Release */
    hal_gfx_cl_destroy(&cl);
    app_graphics_mem_free(s_fb.bo.base_virt);
    return 0;
}
```

4.3.2.1 外设初始化与缓冲区配置

路径：工程目录下的Src\application\graphics_animation_effects.c

名称：static int init(void)

该函数主要完成外设初始化与缓冲区配置，包括：

- 初始化OSPI、PSRAM外设模块（可选）
- 初始化GPU、DC外设模块
- 初始化帧缓冲区，申请合适的Texture Memory存储Texture Image（可选），并完成绑定工作
- 初始化CL所需的环形缓冲区

```
static int init(void)
{
    /* Initialize OSPI, PSRAM Peripheral Modules */
    app_graphics_ospi_params_t params = PSRAM_INIT_PARAMS_Default;
    app_graphics_ospi_init(&params);

    /* Initialize GPU, DC Peripheral Modules */
    graphics_gpu_init(NULL);
```

```
    graphics_dc_rm69330_qspi_lcd_init(LCD_RES_CURRENT, LCD_PIXEL_mode_24bit,
                                      GDC_MIPICFG_QSPI_RGB888_OPT0);

    /* Initialize Graphics Memory Management */
    app_graphics_mem_init((uint8_t*)GFX_MEM_BASE, GFX_MEM_SIZE);

    /* Buffer Configuration */
    load_objects();

    /* Create Command List */
    cl  = hal_gfx_cl_create();

    /* Bind Command List */
    hal_gfx_cl_bind(&cl);

    /* Set Clipping Rectangle */
    hal_gfx_set_clip(0, 0, SHOW_AREA_X, SHOW_AREA_Y);

    /* Bind Texture to HAL_GFX_TEX1 Slot (Foreground) */
    hal_gfx_bind_src_tex(screen0.bo.base_phys,
                         screen0.w, screen0.h,
                         (hal_gfx_tex_format_t)(screen0.format),
                         screen0.stride,
                         (hal_gfx_tex_mode_t)(HAL_GFX_FILTER_BL |
                         HAL_GFX_TEX_BORDER));

    /* Bind Texture to HAL_GFX_TEX2 Slot (Background) */
    hal_gfx_bind_src2_tex(screen1.bo.base_phys,
                          screen1.w, screen1.h,
                          (hal_gfx_tex_format_t)(screen1.format),
                          screen1.stride,
                          (hal_gfx_tex_mode_t)(HAL_GFX_FILTER_BL |
                           HAL_GFX_TEX_BORDER));

    /* Set Texture Default Color */
    hal_gfx_set_tex_color(0);

    /* Submit Current CL and Wait */
    hal_gfx_cl_submit(&cl);
    hal_gfx_cl_wait(&cl);

    return 0;
}
```

路径：工程目录下的Src\application\graphics_animation_effects.c

名称：static void load_objects(void)

该函数主要完成缓冲区配置，包括初始化帧缓冲区，为GPU和DC配置全局Framebuffer参数，申请合适的Texture Memory存储Texture Image（可选）。

```
static void load_objects(void)
{
    /* Configure Framebuffer for GPU */
    s_fb.bo = hal_gfx_fb_create(s_fb.w*s_fb.h*PIXEL_DEPTH_BYTES);
    hal_gfx_buffer_map(&s_fb.bo);

    /* Configure Framebuffer for DC */
#if HAL_GDC_DISPLAY_ENABLE >0u
    s_dc_layer.resolution_x   = s_fb.w;
    s_dc_layer.resolution_y   = s_fb.h;
#if TSCx_ENABLE > 0u
    s_dc_layer.data_format    = GDC_DATA_FORMAT_RGBA8888 ;
#else
    s_dc_layer.data_format    = GDC_DATA_FORMAT_RGBA8888 ;
#endif
    s_dc_layer.row_stride     = s_dc_layer.resolution_x*PIXEL_DEPTH_BYTES;
    s_dc_layer.blendmode      = HAL_GDC_BL_SIMPLE;
    s_dc_layer.start_x        = 0;
    s_dc_layer.start_y        = 0;
    s_dc_layer.size_x         = s_dc_layer.resolution_x;
    s_dc_layer.size_y         = s_dc_layer.resolution_y;
    s_dc_layer.alpha          = 0x80;
    s_dc_layer.frame_baseaddr = (void*)s_fb.bo.base_phys;
#endif

    printf("FB: V:%p P:0x%08x\n", (void *)s_fb.bo.base_virt,
                                  (uint32_t)s_fb.bo.base_phys);
}
```

4.3.2.2 渲染任务与屏幕刷新

路径：工程目录下的Src\application\graphics_animation_effects.c

名称：static void animation_effects_loop(void)

该函数主要构建GPU渲染任务的命令列表，包含GPU的各种图像过渡效果。最终通过DC外设模块，实时将图像刷新至屏幕。

```
static void animation_effects_loop(void)
{
    float step = 0.f;
    float step_step = ANIMATION_STEP_0_1;
    float start_time = hal_gfx_get_time();
    int frame = 0;

    while (1)
    {
        /* Build GPU Rendering Tasks and Refresh Screen */
        display(step, HAL_GFX_BL_SRC);
```

```
        /* Realize Image Transition Effect of GPU */
        if (step <= 0.f)
        {
            step = 0.f;
            step_step = ANIMATION_STEP_0_1;
            next_effect();
        }
        else if (step >= 1.f)
        {
            step = 1.f;
            step_step = -ANIMATION_STEP_0_1;
            next_effect();
        }
        step += step_step;
        frame++;

        /* Calculate FPS */
        hal_gfx_calculate_fps_ext(start_time, frame);
    }
}
```

路径：工程目录下的Src\application\graphics_animation_effects.c

名称：static void display(float step, uint32_t blendmode)

该函数主要构建GPU渲染任务的命令列表和通过DC外设模块实时将图像刷新至屏幕。

```
static void display(float step, uint32_t blendmode)
{

    /* Reset Position of Next Command to be Written to the Beginning */
    hal_gfx_cl_rewind(&cl);

    /* Bind Framebuffer to HAL_GFX_TEX0 Slot */
    hal_gfx_bind_dst_tex((uint32_t)s_fb.bo.base_phys,
                         s_fb.w, s_fb.h,
                         (hal_gfx_tex_format_t)(s_fb.format),
                         s_fb.stride);

    /* GPU Image Transition Effect */
    hal_gfx_transition(s_effect, HAL_GFX_TEX1, HAL_GFX_TEX2,
                       blendmode, step, s_fb.w, s_fb.h);

    /* Submit Current CL and Wait */
    hal_gfx_cl_submit(&cl);
    (void)hal_gfx_cl_wait(&cl);

    /* Refresh Image to Screen via DC */
#if HAL_GDC_DISPLAY_ENABLE > 0u
    app_graphics_dc_set_power_state(GDC_POWER_STATE_ACTIVE);
    graphics_dc_rm69330_qspi_send_frame(s_dc_layer, DISPLAY_LCD_RES_X,
```

```
                                        DISPLAY_LCD_RES_Y);
#endif
}
```

5 常见问题

5.1 GPU初始化失败或无法访问GPU寄存器

- 问题描述

  使用GR5526，在Keil下编译graphics_animation_effects示例工程没有错误，且正常下载至芯片中。程序GPU初始化失败，且无法访问GPU寄存器。

- 问题分析

  GR5526分为标准版本和GPU版本，标准版本不包含GPU功能模块，因此程序GPU初始化失败，且无法访问GPU寄存器。

- 处理方法

  使用GR5526VGBIP或GR5526RGNIP芯片。

5.2 使用GPU处理渲染任务，屏幕无显示或显示不正确

- 问题描述

  使用GR5526VGBIP或GR5526RGNIP正确构建GPU的命令列表，工程编译成功，且正常下载至芯片中。程序正常运行，但是屏幕上没有显示图像，或显示图像不符合预期。

- 问题分析

  1. 屏幕有损坏，或者屏幕与DC的适配存在问题。
  2. 当设置Framebuffer全部位于RAM时，需检查当前申请的Framebuffer大小是否已超出*graphics_sys_defs.h*中分配给Framebuffer的内存最大值（不同大小的显示区域和颜色格式所需的Framebuffer大小不同）。如申请的Framebuffer超过设置的最大值，则Framebuffer参数将设置失败。
  3. GPU的命令列表储存于环形缓冲区中，*graphics_sys_defs.h*中有定义环形缓冲区的最大值。如果单个命令列表过长或命令列表过多，会导致部分命令列表异常。

- 处理方法

  1. 通过SDK内部示例工程测试屏幕能否正常刷新。
  2. 检查Framebuffer的基地址。
     - 在运行过程中，如果Framebuffer申请内存成功，则基地址应位于RAM区域。
     - 如果Framebuffer申请内存失败，则基地址为0x00000000。
  3. 如命令列表出现异常，可适当增大设置的环形缓冲区的最大值。

5.3 使用GPU处理纹理图像，屏幕图像失真或FPS低

- 问题描述

  使用GR5526VGBIP或GR5526RGNIP正确构建GPU的命令列表，工程编译成功，且正常下载至芯片中。程序正常运行，且屏幕上有图像显示，但是图像存在失真，或图像的FPS低。

- 问题分析

  大部分图像失真或图像FPS偏低的原因和纹理图像、帧缓冲区位于的存储介质有关。由于不同存储介质的吞吐率不同，屏幕的帧率和显示效果也会受到相应的影响。

- 处理方法

  1. 将纹理图像从Flash中拷贝至PSRAM或RAM运行。
  2. 使用GPU的压缩格式纹理图像。

5.4 如何支持GPU应用实现低功耗管理

- 问题描述

  如何支持GPU应用实现低功耗电源管理。

- 问题分析

  GPU应用与其他应用相比，新增GPU、DC、OSPI三个模块需要进行低功耗管理，因此针对GPU、DC、OSPI分别实现低功耗管理即可。

- 处理方法

  1. GPU低功耗管理

     应用层使用hal_gfx_cl_le_create和hal_gfx_cl_le_destroy接口，两个接口每次渲染任务开始和结束成对使用，即可完成GPU低功耗管理。

  2. DC低功耗管理

     应用层在开始刷屏时调用如下接口将DC配置为ACTIVE状态，关闭DC电源逻辑由app_graphics_dc驱动模块在刷屏完成后自动完成。

```
app_graphics_dc_set_power_state(GDC_POWER_STATE_ACTIVE);
```

使用示例如下，即在每次刷屏时，将DC配置为ACTIVE状态：

```
static void display_flush(gw_area_t * area)
{
    app_graphics_dc_set_power_state(GDC_POWER_STATE_ACTIVE);
    uint32_t dis_buf_addr =(uint32_t)gw_disp_get_buf_act();
    app_graphics_dc_cmd_t   dc_cmd;
    dc_cmd.command       = 0x12;
    dc_cmd.address       = 0x002C00;
    dc_cmd.address_width = GDC_FRAME_ADDRESS_WIDTH_16BIT;
    dc_cmd.frame_timing  = GDC_QSPI_FRAME_TIMING_1;
```

```
    s_dc_layer.resolution_y    = area->y2 - area->y1;
    s_dc_layer.size_y          = s_dc_layer.resolution_y;
    s_dc_layer.frame_baseaddr = (void*)(dis_buf_addr + area->y1 * GR_HOR_RES_MAX);
    graphics_dc_rm69330_qspi_lcd_set_show_area(area->x1, area->x2-1,
                                               area->y1, area->y2-1);
    app_graphics_dc_send_single_frame(GRAPHICS_DC_LAYER_0, &s_dc_layer,
                                      &dc_cmd, GDC_ACCESS_TYPE_ASYNC);
}
```

3. OSPI低功耗管理

   OSPI支持HALF SLEEP和DEEP SLEEP两种低功耗模式，其中HALF SLEEP下PSRAM数据不会丢失，DEEP SLEEP下PSRAM数据会丢失：

   - OSPI HALF SLEEP模式

     OSPI驱动默认支持HALF SLEEP模式，该模式无需用户管理OSPI电源。

   - DEEP SLEEP模式

     OSPI DEEP SLEEP模式下PSRAM数据会丢失，因此该模式下需要用户通过app_graphics_ospi_set_sleep_state接口配置OSPI为DEEP SLEEP模式，并在下次唤醒页面重载需要的资源到PSRAM。

   另外提供如下接口注册函数，该接口在PSRAM掉电后，再次初始化时调用，可用于通知应用层PSRAM是否丢失数据，或者直接在回调中恢复PSRAM数据，建议采用通知方式根据需要仅在PSRAM掉电发生数据丢失后，在应用层页面创建逻辑重载资源：

```
/*
 * @brief  Register the OSPI reload function for PSRAM recovery after deep sleep.
 *
 * @param[in]  psram_reload_func_t: Reload the resources to PSRAM after PSRAM deep sleep.
 */
void app_graphics_ospi_register_psram_reload_func(psram_reload_func_t psram_reload_func);
```

6 术语和缩略语

表 6-1 术语和缩略语

名称	描述
CG	Computer Graphics，计算机图形学
CL	Command List，命令列表
CLP	Command List Processor，命令列表处理器
CRF	Configuration Register File，配置寄存器文件
DC	Display Controller，显示控制器
DMA	Direct Memory Access，直接存储器访问
GPOS	Glyph Positioning，字形定位
GPU	Graphics Processing Unit，图形处理器
ISA	Instruction Set Architecture，指令集架构
MSAA	Multi-Sampling Anti-Aliasing，多重采样抗锯齿
SoC	System-on-Chip，片上系统