

3D Graphics Engine for ESP32

Generated by Doxygen 1.13.2

1 Todo List	1
2 Namespace Index	3
2.1 Namespace List	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Namespace Documentation	13
6.1 basics Namespace Reference	13
6.1.1 Typedef Documentation	13
6.1.1.1 Id	13
6.1.2 Function Documentation	13
6.1.2.1 fnv32() [1/2]	13
6.1.2.2 fnv32() [2/2]	14
6.1.2.3 static_fnv()	14
6.1.2.4 static_fnv32()	14
6.1.2.5 static_fnv64()	15
6.2 basics::internal Namespace Reference	15
6.2.1 Function Documentation	16
6.2.1.1 static_fnv32()	16
6.2.1.2 static_fnv32< 1 >()	16
6.2.1.3 static_fnv64()	17
6.2.1.4 static_fnv64< 1 >()	17
6.2.2 Variable Documentation	17
6.2.2.1 fnv_basis_32	17
6.2.2.2 fnv_basis_64	17
6.2.2.3 fnv_prime_32	18
6.2.2.4 fnv_prime_64	18
6.3 Ragot Namespace Reference	18
6.3.1 Typedef Documentation	20
6.3.1.1 Matrix4x4	20
6.3.1.2 RGB565	20
6.3.1.3 RGB8	20
6.3.1.4 RGB888	21
6.3.1.5 RGBA8888	21
6.3.2 Enumeration Type Documentation	21
6.3.2.1 Buffer	21

6.3.2.2 render_flag_t	21
6.3.3 Function Documentation	22
6.3.3.1 clipAgainstPlane()	22
6.3.3.2 operator!=(())	22
6.3.3.3 operator==(())	22
6.3.3.4 panel_refresh_callback() [1/2]	23
6.3.3.5 panel_refresh_callback() [2/2]	24
6.3.4 Variable Documentation	24
6.3.4.1 assets	24
6.3.4.2 logger	24
6.3.4.3 MAIN_TAG	24
6.3.4.4 PI	24
6.3.4.5 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0	24
6.3.4.6 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1	25
6.3.4.7 Rasterizer< COLOR_BUFFER_TYPE >::z_cache0	25
6.3.4.8 Rasterizer< COLOR_BUFFER_TYPE >::z_cache1	25
6.3.4.9 RENDERER_TAG	25
6.3.4.10 serializer	25
6.3.4.11 TAG	25
6.3.4.12 thread_pool	25
7 Class Documentation	27
7.1 Ragot::Assets Class Reference	27
7.1.1 Detailed Description	28
7.1.2 Constructor & Destructor Documentation	29
7.1.2.1 Assets() [1/3]	29
7.1.2.2 Assets() [2/3]	29
7.1.2.3 Assets() [3/3]	30
7.1.3 Member Function Documentation	30
7.1.3.1 get_asset_path()	30
7.1.3.2 initialize()	30
7.1.3.3 instance()	31
7.1.3.4 operator=(()) [1/2]	31
7.1.3.5 operator=(()) [2/2]	31
7.1.4 Member Data Documentation	32
7.1.4.1 base_path	32
7.2 Ragot::Camera Class Reference	32
7.2.1 Detailed Description	38
7.2.2 Member Typedef Documentation	38
7.2.2.1 Matrix4x4	38
7.2.3 Constructor & Destructor Documentation	39
7.2.3.1 Camera() [1/2]	39

7.2.3.2 Camera() [2 / 2]	39
7.2.3.3 ~Camera()	39
7.2.4 Member Function Documentation	39
7.2.4.1 calculate_normal()	39
7.2.4.2 get_aspect_ratio()	40
7.2.4.3 get_far_plane()	40
7.2.4.4 get_fov()	40
7.2.4.5 get_location()	41
7.2.4.6 get_near_plane()	41
7.2.4.7 get_projection_matrix()	41
7.2.4.8 get_right_direction()	42
7.2.4.9 get_target()	42
7.2.4.10 get_up_direction()	43
7.2.4.11 get_view_direction()	43
7.2.4.12 get_view_matrix()	44
7.2.4.13 get_vp_matrix()	44
7.2.4.14 is_dirty()	45
7.2.4.15 is_face_visible()	45
7.2.4.16 log_camera_info()	46
7.2.4.17 project_to_ndc()	46
7.2.4.18 set_aspect_ratio()	47
7.2.4.19 set_far_plane()	47
7.2.4.20 set_fov()	47
7.2.4.21 set_location()	47
7.2.4.22 set_near_plane()	48
7.2.4.23 set_target()	48
7.2.5 Member Data Documentation	49
7.2.5.1 aspect_ratio	49
7.2.5.2 CAMERA_TAG	49
7.2.5.3 far_plane	49
7.2.5.4 fov	49
7.2.5.5 near_plane	50
7.2.5.6 projDirty	50
7.2.5.7 projection_matrix	50
7.2.5.8 target	50
7.2.5.9 view_matrix	50
7.2.5.10 viewDirty	50
7.2.5.11 vp_matrix	50
7.2.5.12 vpDirty	51
7.3 Ragot::Camera_Transform Struct Reference	51
7.3.1 Detailed Description	52
7.3.2 Member Data Documentation	52

7.3.2.1 dir_x	52
7.3.2.2 dir_y	52
7.3.2.3 dir_z	52
7.3.2.4 x	52
7.3.2.5 y	52
7.3.2.6 z	52
7.4 Ragot::Component Class Reference	53
7.4.1 Detailed Description	56
7.4.2 Constructor & Destructor Documentation	57
7.4.2.1 Component() [1/3]	57
7.4.2.2 ~Component()	57
7.4.2.3 Component() [2/3]	57
7.4.2.4 Component() [3/3]	58
7.4.3 Member Function Documentation	58
7.4.3.1 add_component()	58
7.4.3.2 get_components()	58
7.4.3.3 operator=() [1/2]	59
7.4.3.4 operator=() [2/2]	59
7.4.3.5 remove_component()	59
7.4.4 Member Data Documentation	60
7.4.4.1 components	60
7.5 Ragot::coordinates_t Struct Reference	60
7.5.1 Detailed Description	60
7.5.2 Member Data Documentation	61
7.5.2.1 x	61
7.5.2.2 y	61
7.6 Ragot::Driver_ST7789 Class Reference	61
7.6.1 Detailed Description	65
7.6.2 Constructor & Destructor Documentation	65
7.6.2.1 Driver_ST7789()	65
7.6.2.2 ~Driver_ST7789()	65
7.6.3 Member Function Documentation	65
7.6.3.1 bsp_init_lcd_backlight()	65
7.6.3.2 deinit()	66
7.6.3.3 init()	66
7.6.3.4 refresh_frame_buffer()	67
7.6.3.5 send_frame_buffer()	67
7.6.3.6 set_pixel()	68
7.6.4 Member Data Documentation	68
7.6.4.1 refresh_semaphore	68
7.6.4.2 TAG	68
7.7 Ragot::DriverEK79007 Class Reference	69

7.7.1 Detailed Description	72
7.7.2 Constructor & Destructor Documentation	72
7.7.2.1 DriverEK79007()	72
7.7.2.2 ~DriverEK79007()	73
7.7.3 Member Function Documentation	73
7.7.3.1 bsp_enable_dsi_phy_power()	73
7.7.3.2 bsp_init_lcd_backlight()	73
7.7.3.3 deinit()	74
7.7.3.4 init()	74
7.7.3.5 refresh_frame_buffer()	75
7.7.3.6 send_frame_buffer()	76
7.7.3.7 set_pixel()	76
7.7.4 Member Data Documentation	77
7.7.4.1 hsync_back_porch	77
7.7.4.2 hsync_front_porch	77
7.7.4.3 hsync_pulse_width	77
7.7.4.4 mipi_dsi_max_data_rate_mbps	77
7.7.4.5 mipi_lane_num	77
7.7.4.6 panel_clk_freq_mhz	78
7.7.4.7 refresh_semaphore	78
7.7.4.8 vsync_back_porch	78
7.7.4.9 vsync_front_porch	78
7.7.4.10 vsync_pulse_width	78
7.8 Ragot::DriverLCD Class Reference	79
7.8.1 Detailed Description	81
7.8.2 Constructor & Destructor Documentation	81
7.8.2.1 DriverLCD()	81
7.8.2.2 ~DriverLCD()	82
7.8.3 Member Function Documentation	82
7.8.3.1 deinit()	82
7.8.3.2 get_handler() [1/2]	82
7.8.3.3 get_handler() [2/2]	82
7.8.3.4 get_height() [1/2]	83
7.8.3.5 get_height() [2/2]	83
7.8.3.6 get_pixel_format() [1/2]	83
7.8.3.7 get_pixel_format() [2/2]	83
7.8.3.8 get_width() [1/2]	83
7.8.3.9 get_width() [2/2]	84
7.8.3.10 init()	84
7.8.3.11 is_initialized() [1/2]	84
7.8.3.12 is_initialized() [2/2]	84
7.8.3.13 send_frame_buffer()	84

7.8.3.14 set_pixel()	85
7.8.4 Member Data Documentation	85
7.8.4.1 handler	85
7.8.4.2 height	85
7.8.4.3 initialized	86
7.8.4.4 pixel_format	86
7.8.4.5 width	86
7.9 Ragot::ExtrudeMesh Class Reference	86
7.9.1 Detailed Description	92
7.9.2 Constructor & Destructor Documentation	92
7.9.2.1 ExtrudeMesh()	92
7.9.2.2 ~ExtrudeMesh()	92
7.9.3 Member Function Documentation	93
7.9.3.1 are_vertices_coplanar()	93
7.9.3.2 generate_faces()	94
7.9.3.3 generate_vertices()	94
7.9.3.4 log_mesh_info()	95
7.9.4 Member Data Documentation	95
7.9.4.1 cam	95
7.9.4.2 camPos	95
7.9.4.3 EXTRUDE_TAG	96
7.9.4.4 faces_can_be_quads	96
7.9.4.5 height	96
7.9.4.6 planes	96
7.10 Ragot::face_t Struct Reference	97
7.10.1 Detailed Description	97
7.10.2 Member Data Documentation	97
7.10.2.1 is_quad	97
7.10.2.2 v1	98
7.10.2.3 v2	98
7.10.2.4 v3	98
7.10.2.5 v4	98
7.11 Ragot::Fragment_Shader Class Reference	98
7.11.1 Detailed Description	101
7.11.2 Constructor & Destructor Documentation	101
7.11.2.1 Fragment_Shader()	101
7.12 Ragot::FrameBuffer< Color > Class Template Reference	102
7.12.1 Detailed Description	105
7.12.2 Member Typedef Documentation	105
7.12.2.1 ColorVector	105
7.12.2.2 TYPE	105
7.12.3 Constructor & Destructor Documentation	106

7.12.3.1 FrameBuffer() [1/4]	106
7.12.3.2 FrameBuffer() [2/4]	107
7.12.3.3 ~FrameBuffer()	108
7.12.3.4 FrameBuffer() [3/4]	108
7.12.3.5 FrameBuffer() [4/4]	108
7.12.4 Member Function Documentation	109
7.12.4.1 blit_to_window()	109
7.12.4.2 clear_buffer()	109
7.12.4.3 fill()	109
7.12.4.4 get_buffer() [1/2]	110
7.12.4.5 get_buffer() [2/2]	110
7.12.4.6 get_height() [1/2]	110
7.12.4.7 get_height() [2/2]	111
7.12.4.8 get_pixel()	111
7.12.4.9 get_width() [1/2]	111
7.12.4.10 get_width() [2/2]	112
7.12.4.11 getGLFormat()	112
7.12.4.12 getGLTex()	112
7.12.4.13 getGLType()	112
7.12.4.14 initGLTexture()	113
7.12.4.15 operator=() [1/2]	113
7.12.4.16 operator=() [2/2]	114
7.12.4.17 sendGL()	114
7.12.4.18 set_color()	114
7.12.4.19 set_pixel() [1/3]	114
7.12.4.20 set_pixel() [2/3]	115
7.12.4.21 set_pixel() [3/3]	115
7.12.4.22 swap_buffers()	115
7.12.5 Member Data Documentation	115
7.12.5.1 buffer_1	115
7.12.5.2 buffer_2	116
7.12.5.3 color	116
7.12.5.4 current_buffer	116
7.12.5.5 double_buffer	116
7.12.5.6 gl_tex	116
7.12.5.7 height	116
7.12.5.8 next_buffer	117
7.12.5.9 width	117
7.13 Ragot::Logger Class Reference	117
7.13.1 Detailed Description	118
7.13.2 Constructor & Destructor Documentation	119
7.13.2.1 Logger() [1/3]	119

7.13.2.2 ~Logger()	119
7.13.2.3 Logger() [2/3]	119
7.13.2.4 Logger() [3/3]	120
7.13.3 Member Function Documentation	120
7.13.3.1 getLogLevel()	120
7.13.3.2 instance()	120
7.13.3.3 Log()	121
7.13.3.4 operator=() [1/2]	122
7.13.3.5 operator=() [2/2]	122
7.13.3.6 setLogLevel()	122
7.13.4 Member Data Documentation	123
7.13.4.1 logLevel	123
7.14 Ragot::Mesh Class Reference	123
7.14.1 Detailed Description	128
7.14.2 Constructor & Destructor Documentation	128
7.14.2.1 Mesh() [1/2]	128
7.14.2.2 ~Mesh()	129
7.14.2.3 Mesh() [2/2]	129
7.14.3 Member Function Documentation	129
7.14.3.1 apply_transform_to_vertices()	129
7.14.3.2 generate_faces()	130
7.14.3.3 generate_vertices()	130
7.14.3.4 get_color()	131
7.14.3.5 get_faces()	131
7.14.3.6 get_total_vertices() [1/2]	131
7.14.3.7 get_total_vertices() [2/2]	132
7.14.3.8 get_vertices()	132
7.14.3.9 recalculate()	133
7.14.3.10 set_color()	133
7.14.4 Member Data Documentation	133
7.14.4.1 color	133
7.14.4.2 faces	133
7.14.4.3 mesh_info	134
7.14.4.4 slices	134
7.14.4.5 vertices	134
7.15 Ragot::mesh_info_t Struct Reference	134
7.15.1 Detailed Description	135
7.15.2 Constructor & Destructor Documentation	135
7.15.2.1 mesh_info_t() [1/2]	135
7.15.2.2 mesh_info_t() [2/2]	135
7.15.3 Member Data Documentation	135
7.15.3.1 coordinates	135

7.15.3.2 render_flag	135
7.15.3.3 vertex_amount	135
7.16 Ragot::MeshSerializer Class Reference	136
7.16.1 Detailed Description	137
7.16.2 Constructor & Destructor Documentation	137
7.16.2.1 MeshSerializer() [1/3]	137
7.16.2.2 MeshSerializer() [2/3]	137
7.16.2.3 MeshSerializer() [3/3]	138
7.16.3 Member Function Documentation	138
7.16.3.1 instance()	138
7.16.3.2 operator=() [1/2]	139
7.16.3.3 operator=() [2/2]	139
7.16.3.4 save_to_obj()	139
7.17 Ragot::Node Class Reference	140
7.17.1 Detailed Description	144
7.17.2 Constructor & Destructor Documentation	144
7.17.2.1 Node() [1/3]	144
7.17.2.2 ~Node()	144
7.17.2.3 Node() [2/3]	145
7.17.2.4 Node() [3/3]	145
7.17.3 Member Function Documentation	145
7.17.3.1 add_child()	145
7.17.3.2 get_children()	146
7.17.3.3 get_transform_matrix()	146
7.17.3.4 operator=() [1/2]	147
7.17.3.5 operator=() [2/2]	147
7.17.3.6 remove_child()	147
7.17.4 Member Data Documentation	148
7.17.4.1 children	148
7.17.4.2 parent	148
7.18 Ragot::Window::OpenGL_Context_Settings Struct Reference	148
7.18.1 Detailed Description	149
7.18.2 Member Data Documentation	149
7.18.2.1 core_profile	149
7.18.2.2 depth_buffer_size	149
7.18.2.3 enable_vsync	149
7.18.2.4 stencil_buffer_size	149
7.18.2.5 version_major	150
7.18.2.6 version_minor	150
7.19 Ragot::PSRAMAllocator< T, Flag > Class Template Reference	150
7.19.1 Detailed Description	151
7.19.2 Member Typedef Documentation	152

7.19.2.1 pointer	152
7.19.2.2 size_type	152
7.19.2.3 value_type	152
7.19.3 Constructor & Destructor Documentation	152
7.19.3.1 PSRAMAllocator() [1/2]	152
7.19.3.2 PSRAMAllocator() [2/2]	152
7.19.4 Member Function Documentation	153
7.19.4.1 allocate()	153
7.19.4.2 deallocate()	153
7.20 Ragot::Rasterizer< Color > Class Template Reference	154
7.20.1 Detailed Description	156
7.20.2 Constructor & Destructor Documentation	157
7.20.2.1 Rasterizer() [1/2]	157
7.20.2.2 Rasterizer() [2/2]	157
7.20.2.3 ~Rasterizer()	157
7.20.3 Member Function Documentation	157
7.20.3.1 clear()	157
7.20.3.2 fill_convex_polygon() [1/2]	158
7.20.3.3 fill_convex_polygon() [2/2]	158
7.20.3.4 fill_convex_polygon_z_buffer() [1/2]	158
7.20.3.5 fill_convex_polygon_z_buffer() [2/2]	159
7.20.3.6 fill_row() [1/2]	159
7.20.3.7 fill_row() [2/2]	160
7.20.3.8 fill_row_zbuffer() [1/2]	160
7.20.3.9 fill_row_zbuffer() [2/2]	161
7.20.3.10 get_frame_buffer()	161
7.20.3.11 interpolate()	161
7.20.3.12 set_color()	162
7.20.4 Member Data Documentation	162
7.20.4.1 clear_color	162
7.20.4.2 color	162
7.20.4.3 debug_enabled	163
7.20.4.4 frame_buffer	163
7.20.4.5 offset_cache0	163
7.20.4.6 offset_cache1	163
7.20.4.7 RASTER_TAG	163
7.20.4.8 z_buffer	163
7.20.4.9 z_cache0	163
7.20.4.10 z_cache1	164
7.21 Ragot::PSRAMAllocator< T, Flag >::rebind< U > Struct Template Reference	164
7.21.1 Detailed Description	164
7.21.2 Member Typedef Documentation	165

7.21.2.1 other	165
7.22 Ragot::Renderer Class Reference	165
7.22.1 Detailed Description	168
7.22.2 Constructor & Destructor Documentation	168
7.22.2.1 Renderer() [1/2]	168
7.22.2.2 Renderer() [2/2]	168
7.22.2.3 ~Renderer()	169
7.22.3 Member Function Documentation	169
7.22.3.1 init()	169
7.22.3.2 initFullScreenQuad()	169
7.22.3.3 is_frontface()	170
7.22.3.4 render()	170
7.22.3.5 set_scene()	171
7.22.3.6 start()	171
7.22.3.7 stop()	171
7.22.3.8 task_render()	172
7.22.4 Member Data Documentation	172
7.22.4.1 accumulated_time	172
7.22.4.2 current_scene	172
7.22.4.3 display_vertices	173
7.22.4.4 fragment_shader_code	173
7.22.4.5 frame_buffer	173
7.22.4.6 height	173
7.22.4.7 initialized	173
7.22.4.8 iterations	173
7.22.4.9 number_of_iterations	174
7.22.4.10 quadEBO	174
7.22.4.11 quadShader	174
7.22.4.12 quadVAO	174
7.22.4.13 quadVBO	174
7.22.4.14 rasterizer	174
7.22.4.15 running	174
7.22.4.16 transformed_vertices	175
7.22.4.17 vertex_shader_code	175
7.22.4.18 width	175
7.23 Ragot::RevolutionMesh Class Reference	175
7.23.1 Detailed Description	180
7.23.2 Constructor & Destructor Documentation	181
7.23.2.1 RevolutionMesh()	181
7.23.2.2 ~RevolutionMesh()	181
7.23.3 Member Function Documentation	182
7.23.3.1 generate_faces()	182

7.23.3.2 generate_vertices()	182
7.23.4 Member Data Documentation	183
7.23.4.1 cam	183
7.23.4.2 faces_can_be_quads	183
7.23.4.3 PI	183
7.24 Ragot::Scene Class Reference	183
7.24.1 Detailed Description	185
7.24.2 Constructor & Destructor Documentation	186
7.24.2.1 Scene() [1/2]	186
7.24.2.2 ~Scene()	186
7.24.2.3 Scene() [2/2]	186
7.24.3 Member Function Documentation	187
7.24.3.1 add_node()	187
7.24.3.2 collect_components()	187
7.24.3.3 find_node()	188
7.24.3.4 get_main_camera()	188
7.24.3.5 get_root() [1/2]	189
7.24.3.6 get_root() [2/2]	189
7.24.3.7 remove_node()	189
7.24.3.8 set_main_camera()	189
7.24.3.9 start()	190
7.24.3.10 stop()	190
7.24.3.11 task_update()	190
7.24.3.12 traverse()	191
7.24.3.13 update()	192
7.24.4 Member Data Documentation	192
7.24.4.1 main_camera	192
7.24.4.2 named_nodes	192
7.24.4.3 root_node	192
7.24.4.4 running	193
7.25 Ragot::Shader Class Reference	193
7.25.1 Detailed Description	195
7.25.2 Constructor & Destructor Documentation	195
7.25.2.1 Shader() [1/2]	195
7.25.2.2 Shader() [2/2]	196
7.25.2.3 ~Shader()	196
7.25.3 Member Function Documentation	196
7.25.3.1 compile_shader()	196
7.25.3.2 get_error()	196
7.25.3.3 get_id()	196
7.25.3.4 is_ok()	197
7.25.3.5 show_compilation_error()	197

7.25.4 Member Data Documentation	197
7.25.4.1 compilation_succeeded	197
7.25.4.2 error	197
7.25.4.3 id	197
7.26 Ragot::Shader_Program Class Reference	198
7.26.1 Detailed Description	199
7.26.2 Constructor & Destructor Documentation	199
7.26.2.1 Shader_Program() [1/3]	199
7.26.2.2 Shader_Program() [2/3]	200
7.26.2.3 ~Shader_Program()	200
7.26.2.4 Shader_Program() [3/3]	200
7.26.3 Member Function Documentation	201
7.26.3.1 get_id()	201
7.26.3.2 get_uniform_location()	201
7.26.3.3 initialize()	201
7.26.3.4 operator=()	202
7.26.3.5 show_linkage_error()	202
7.26.3.6 use()	202
7.26.4 Member Data Documentation	203
7.26.4.1 program_id	203
7.27 Ragot::Sync_Queue< T > Class Template Reference	203
7.27.1 Detailed Description	206
7.27.2 Member Typedef Documentation	206
7.27.2.1 value_type	206
7.27.3 Constructor & Destructor Documentation	207
7.27.3.1 Sync_Queue() [1/2]	207
7.27.3.2 ~Sync_Queue()	207
7.27.3.3 Sync_Queue() [2/2]	208
7.27.4 Member Function Documentation	208
7.27.4.1 back()	208
7.27.4.2 clear()	208
7.27.4.3 close()	209
7.27.4.4 emplace()	209
7.27.4.5 empty()	209
7.27.4.6 get()	210
7.27.4.7 operator=()	210
7.27.4.8 push() [1/2]	211
7.27.4.9 push() [2/2]	211
7.27.4.10 size()	211
7.27.4.11 swap()	211
7.27.5 Member Data Documentation	212
7.27.5.1 closed	212

7.27.5.2 condition	212
7.27.5.3 mutex	212
7.27.5.4 queue	212
7.28 Ragot::Thread_Pool Class Reference	213
7.28.1 Detailed Description	215
7.28.2 Member Typedef Documentation	215
7.28.2.1 Task	215
7.28.3 Constructor & Destructor Documentation	215
7.28.3.1 Thread_Pool() [1/3]	215
7.28.3.2 ~Thread_Pool()	216
7.28.3.3 Thread_Pool() [2/3]	217
7.28.3.4 Thread_Pool() [3/3]	217
7.28.4 Member Function Documentation	217
7.28.4.1 instance()	217
7.28.4.2 operator=() [1/2]	218
7.28.4.3 operator=() [2/2]	219
7.28.4.4 start()	219
7.28.4.5 stop()	219
7.28.4.6 submit()	220
7.28.4.7 submit_with_stop()	220
7.28.4.8 thread_function()	221
7.28.5 Member Data Documentation	221
7.28.5.1 sem_mesh_ready	221
7.28.5.2 sem_render_done	222
7.28.5.3 started	222
7.28.5.4 tasks	222
7.28.5.5 threads	222
7.29 Ragot::Transform Class Reference	223
7.29.1 Detailed Description	225
7.29.2 Constructor & Destructor Documentation	225
7.29.2.1 Transform()	225
7.29.2.2 ~Transform()	225
7.29.3 Member Function Documentation	226
7.29.3.1 get_position()	226
7.29.3.2 get_rotation()	226
7.29.3.3 get_scale()	226
7.29.3.4 get_transform_matrix()	227
7.29.3.5 is_dirty()	227
7.29.3.6 rotate()	227
7.29.3.7 set_position()	228
7.29.3.8 set_rotation()	228
7.29.3.9 set_scale()	228

7.29.4 Member Data Documentation	229
7.29.4.1 dirty	229
7.29.4.2 position	229
7.29.4.3 rotation	229
7.29.4.4 scale	229
7.29.4.5 transform_matrix	229
7.30 Ragot::transform_t Struct Reference	230
7.30.1 Detailed Description	230
7.30.2 Member Data Documentation	231
7.30.2.1 alpha	231
7.30.2.2 beta	231
7.30.2.3 gamma	231
7.30.2.4 scale	231
7.30.2.5 x	231
7.30.2.6 y	231
7.30.2.7 z	231
7.31 Ragot::Vertex_Shader Class Reference	232
7.31.1 Detailed Description	234
7.31.2 Constructor & Destructor Documentation	234
7.31.2.1 Vertex_Shader()	234
7.32 Ragot::vertex_t Struct Reference	235
7.32.1 Detailed Description	235
7.32.2 Member Data Documentation	235
7.32.2.1 x	235
7.32.2.2 y	235
7.32.2.3 z	236
7.33 Ragot::Window Class Reference	236
7.33.1 Detailed Description	237
7.33.2 Member Enumeration Documentation	237
7.33.2.1 Position	237
7.33.3 Constructor & Destructor Documentation	238
7.33.3.1 Window() [1/4]	238
7.33.3.2 Window() [2/4]	239
7.33.3.3 ~Window()	239
7.33.3.4 Window() [3/4]	240
7.33.3.5 Window() [4/4]	240
7.33.4 Member Function Documentation	240
7.33.4.1 get_height()	240
7.33.4.2 get_width()	241
7.33.4.3 operator=() [1/2]	241
7.33.4.4 operator=() [2/2]	241
7.33.4.5 swap_buffers()	242

7.33.5 Member Data Documentation	242
7.33.5.1 height	242
7.33.5.2 opengl_context	242
7.33.5.3 width	242
7.33.5.4 window_handle	242
8 File Documentation	243
8.1 main/Assets.cpp File Reference	243
8.1.1 Detailed Description	244
8.2 main/Assets.hpp File Reference	244
8.2.1 Detailed Description	245
8.3 Assets.hpp	246
8.4 main/atomic_stubs.c File Reference	247
8.4.1 Detailed Description	247
8.4.2 Function Documentation	248
8.4.2.1 __atomic_clear()	248
8.4.2.2 __atomic_test_and_set()	248
8.5 main/Camera.cpp File Reference	249
8.5.1 Detailed Description	249
8.5.2 Macro Definition Documentation	250
8.5.2.1 GLM_ENABLE_EXPERIMENTAL	250
8.6 main/Camera.hpp File Reference	250
8.6.1 Detailed Description	251
8.7 Camera.hpp	252
8.8 main/CommonTypes.hpp File Reference	253
8.8.1 Detailed Description	255
8.9 CommonTypes.hpp	256
8.10 main/Components.hpp File Reference	256
8.10.1 Detailed Description	257
8.11 Components.hpp	258
8.12 main/driver_ek79007.cpp File Reference	259
8.12.1 Detailed Description	260
8.13 main/driver_ek79007.hpp File Reference	260
8.13.1 Detailed Description	261
8.14 driver_ek79007.hpp	262
8.15 main/driver_lcd.hpp File Reference	263
8.15.1 Detailed Description	264
8.16 driver_lcd.hpp	265
8.17 main/Driver_ST7789.cpp File Reference	265
8.17.1 Detailed Description	266
8.18 main/Driver_ST7789.hpp File Reference	267
8.18.1 Detailed Description	267

8.19 Driver_ST7789.hpp	268
8.20 main/ExtrudeMesh.cpp File Reference	269
8.20.1 Detailed Description	269
8.21 main/ExtrudeMesh.hpp File Reference	270
8.21.1 Detailed Description	271
8.22 ExtrudeMesh.hpp	272
8.23 main/fnv.hpp File Reference	273
8.23.1 Macro Definition Documentation	275
8.23.1.1 FNV	275
8.23.1.2 FNV32	275
8.23.1.3 FNV64	275
8.23.2 Function Documentation	275
8.23.2.1 operator""_fnv()	275
8.24 fnv.hpp	276
8.25 main/FrameBuffer.cpp File Reference	277
8.25.1 Detailed Description	278
8.26 main/FrameBuffer.hpp File Reference	279
8.27 FrameBuffer.hpp	280
8.28 main/Id.hpp File Reference	282
8.28.1 Macro Definition Documentation	283
8.28.1.1 ID	283
8.29 Id.hpp	283
8.30 main/Logger.cpp File Reference	283
8.30.1 Detailed Description	284
8.31 main/Logger.hpp File Reference	285
8.31.1 Detailed Description	286
8.32 Logger.hpp	287
8.33 main/main.cpp File Reference	288
8.33.1 Detailed Description	288
8.33.2 Function Documentation	289
8.33.2.1 main()	289
8.33.2.2 main_loop()	290
8.33.3 Variable Documentation	290
8.33.3.1 MAIN_TAG	290
8.34 main/Mesh.cpp File Reference	291
8.34.1 Detailed Description	291
8.35 main/Mesh.hpp File Reference	292
8.35.1 Detailed Description	293
8.36 Mesh.hpp	294
8.37 main/MeshSerializer.cpp File Reference	295
8.37.1 Detailed Description	295
8.38 main/MeshSerializer.hpp File Reference	296

8.38.1 Detailed Description	297
8.39 MeshSerializer.hpp	298
8.40 main/Node.hpp File Reference	299
8.40.1 Detailed Description	300
8.41 Node.hpp	301
8.42 main/RamAllocator.hpp File Reference	301
8.42.1 Detailed Description	302
8.43 RamAllocator.hpp	303
8.44 main/Rasterizer.cpp File Reference	304
8.44.1 Detailed Description	305
8.45 main/Rasterizer.hpp File Reference	305
8.45.1 Detailed Description	306
8.46 Rasterizer.hpp	307
8.47 main/Renderer.cpp File Reference	309
8.47.1 Detailed Description	310
8.48 main/Renderer.hpp File Reference	311
8.48.1 Detailed Description	312
8.49 Renderer.hpp	312
8.50 main/RevolutionMesh.cpp File Reference	313
8.50.1 Detailed Description	314
8.51 main/RevolutionMesh.hpp File Reference	315
8.51.1 Detailed Description	316
8.52 RevolutionMesh.hpp	317
8.53 main/Scene.cpp File Reference	318
8.53.1 Detailed Description	318
8.54 main/Scene.hpp File Reference	319
8.54.1 Detailed Description	320
8.55 Scene.hpp	321
8.56 main/Shader_Program.cpp File Reference	322
8.57 main/Shader_Program.hpp File Reference	322
8.58 Shader_Program.hpp	324
8.59 main/Sync_Queue.hpp File Reference	325
8.59.1 Detailed Description	326
8.60 Sync_Queue.hpp	327
8.61 main/Thread_Pool.cpp File Reference	329
8.61.1 Detailed Description	329
8.62 main/Thread_Pool.hpp File Reference	330
8.63 Thread_Pool.hpp	331
8.64 main/Transform.hpp File Reference	333
8.64.1 Detailed Description	334
8.65 Transform.hpp	335
8.66 main/Window.cpp File Reference	336

8.67 main/Window.hpp File Reference	337
8.68 Window.hpp	338

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

basics	13
basics::internal	15
Ragot	18

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Ragot::Assets	27
Ragot::Camera_Transform	51
Ragot::coordinates_t	60
Ragot::DriverLCD	79
Ragot::DriverEK79007	69
Ragot::Driver_ST7789	61
Ragot::face_t	97
Ragot::FrameBuffer< Color >	102
Ragot::Logger	117
Ragot::mesh_info_t	134
Ragot::MeshSerializer	136
Ragot::Window::OpenGL_Context_Settings	148
Ragot::PSRAMAllocator< T, Flag >	150
Ragot::Rasterizer< Color >	154
Ragot::PSRAMAllocator< T, Flag >::rebind< U >	164
Ragot::Renderer	165
Ragot::Scene	183
Ragot::Shader	193
Ragot::Fragment_Shader	98
Ragot::Vertex_Shader	232
Ragot::Shader_Program	198
Ragot::Sync_Queue< T >	203
Ragot::Thread_Pool	213
Ragot::Transform	223
Ragot::Node	140
Ragot::Component	53
Ragot::Camera	32
Ragot::Mesh	123
Ragot::ExtrudeMesh	86
Ragot::RevolutionMesh	175
Ragot::transform_t	230
Ragot::vertex_t	235
Ragot::Window	236
queue< Task >	??
vector< RGB565 >	??

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Ragot::Assets	Manages the asset paths for the application	27
Ragot::Camera	Represents a camera in a 3D space, managing its properties and transformations	32
Ragot::Camera_Transform	Represents the transformation of a camera in 3D space	51
Ragot::Component	Base class for components in the Ragot engine	53
Ragot::coordinates_t	Represents 2D coordinates	60
Ragot::Driver_ST7789	Driver for the ST7789 LCD panel	61
Ragot::DriverEK79007	Driver for the EK79007 LCD panel	69
Ragot::DriverLCD	Base class for LCD drivers	79
Ragot::ExtrudeMesh	Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information	86
Ragot::face_t	Represents a face in a 3D mesh	97
Ragot::Fragment_Shader	Class for managing an OpenGL fragment shader	98
Ragot::FrameBuffer< Color >	Class to manage a frame buffer for rendering graphics	102
Ragot::Logger	Singleton logger class for the Ragot engine	117
Ragot::Mesh	Represents a 3D mesh in the Ragot engine	123
Ragot::mesh_info_t	Represents information about a mesh	134
Ragot::MeshSerializer	Singleton class to serialize Mesh objects to OBJ file format	136

Ragot::Node	Represents a node in a scene graph for 3D rendering	140
Ragot::Window::OpenGL_Context_Settings	Struct for OpenGL context settings	148
Ragot::PSRAMAllocator< T, Flag >	Custom memory allocator for PSRAM	150
Ragot::Rasterizer< Color >	Class for rasterizing polygons in a frame buffer	154
Ragot::PSRAMAllocator< T, Flag >::rebind< U >	Rebinds the allocator to a different type. This struct allows the PSRAMAllocator to be used with different types while maintaining the same allocation flags	164
Ragot::Renderer	Class for rendering scenes in the Ragot engine	165
Ragot::RevolutionMesh	Class for generating revolution meshes	175
Ragot::Scene	Class for managing a 3D scene	183
Ragot::Shader	Class for managing an OpenGL shader	193
Ragot::Shader_Program	Class for managing an OpenGL shader program	198
Ragot::Sync_Queue< T >	A thread-safe queue implementation	203
Ragot::Thread_Pool	A thread pool for managing concurrent tasks	213
Ragot::Transform	A class representing a 3D transformation with position, rotation, and scale	223
Ragot::transform_t	Represents a transformation in 3D space	230
Ragot::Vertex_Shader	Class for managing an OpenGL vertex shader	232
Ragot::vertex_t	Represents a vertex in 3D space	235
Ragot::Window	Class for managing an SDL window with OpenGL context	236

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

main/ Assets.cpp	This file implements the Assets class, which manages the asset paths for the application . . .	243
main/ Assets.hpp	This file defines the Assets class, which manages the asset paths for the application. It provides a singleton instance to access asset paths based on the executable's location. The class initializes the base path for assets depending on whether the build is in debug or release mode. It also provides a method to retrieve the full path of an asset by its name	244
main/ atomic_stubs.c	This file provides atomic operations using GCC built-in functions	247
main/ Camera.cpp	This file implements the Camera class, which manages camera properties and operations in a 3D space	249
main/ Camera.hpp	This file implements the Camera class, which manages camera properties and operations in a 3D space	250
main/ CommonTypes.hpp	This file defines common types and structures used in the Ragot engine, including camera transformations, mesh information, and rendering flags	253
main/ Components.hpp	This file defines the Component class, which serves as a base class for components in the Ragot engine	256
main/ driver_ek79007.cpp	This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel	259
main/ driver_ek79007.hpp	This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel	260
main/ driver_lcd.hpp	This file defines the DriverLCD class, which serves as a base class for LCD drivers in the Ragot engine	263
main/ Driver_ST7789.cpp	This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel	265

main/ Driver_ST7789.hpp	
This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel	267
main/ ExtrudeMesh.cpp	
This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information	269
main/ ExtrudeMesh.hpp	
This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information	270
main/ fnv.hpp	273
main/ FrameBuffer.cpp	
This file implements the FrameBuffer class, which manages a frame buffer for rendering graphics	277
main/ FrameBuffer.hpp	279
main/ Id.hpp	282
main/ Logger.cpp	
This file implements the Logger class, which provides a singleton logger for the Ragot engine .	283
main/ Logger.hpp	
This file implements the Logger class, which provides a singleton logger for the Ragot engine .	285
main/ main.cpp	
This file contains the main function for the Ragot engine, which initializes the renderer and scene, and starts the main rendering loop	288
main/ Mesh.cpp	
This file implements the Mesh class, which represents a 3D mesh in the Ragot engine	291
main/ Mesh.hpp	
This file implements the Mesh class, which represents a 3D mesh in the Ragot engine	292
main/ MeshSerializer.cpp	
This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format	295
main/ MeshSerializer.hpp	
This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format	296
main/ Node.hpp	
This file implements the Node class, which represents a node in a scene graph for 3D rendering	299
main/ RamAllocator.hpp	
This file implements a custom memory allocator for PSRAM in the Ragot engine	301
main/ Rasterizer.cpp	
Implementation of the Rasterizer class for rendering polygons in a frame buffer	304
main/ Rasterizer.hpp	
Implementation of the Rasterizer class for rendering polygons in a frame buffer	305
main/ Renderer.cpp	
Implementation of the Renderer class for rendering scenes in the Ragot engine	309
main/ Renderer.hpp	
Implementation of the Renderer class for rendering scenes in the Ragot engine	311
main/ RevolutionMesh.cpp	
Implementation of the RevolutionMesh class for generating revolution meshes	313
main/ RevolutionMesh.hpp	
Implementation of the RevolutionMesh class for generating revolution meshes	315
main/ Scene.cpp	
Implementation of the Scene class for managing 3D scenes	318
main/ Scene.hpp	
Implementation of the Scene class for managing 3D scenes	319
main/ Shader_Program.cpp	322
main/ Shader_Program.hpp	322

main/ Sync_Queue.hpp	
Implementation of a synchronized queue for thread-safe operations	325
main/ Thread_Pool.cpp	
Implementation of the Thread_Pool class for managing a pool of threads	329
main/ Thread_Pool.hpp	330
main/ Transform.hpp	
Implementation of the Transform class for 3D transformations	333
main/ Window.cpp	336
main/ Window.hpp	337

Chapter 5

Namespace Documentation

5.1 basics Namespace Reference

Namespaces

- namespace [internal](#)

Typedefs

- typedef unsigned int [ld](#)

Functions

- `template<size_t LENGTH>`
`constexpr uint32_t static_fnv32 (const char(&chars)[LENGTH])`
- `template<size_t LENGTH>`
`constexpr uint64_t static_fnv64 (const char(&chars)[LENGTH])`
- `template<size_t LENGTH>`
`constexpr unsigned static_fnv (const char(&chars)[LENGTH])`
- `template<size_t LENGTH>`
`uint32_t fnv32 (const char(&chars)[LENGTH])`
- `uint32_t fnv32 (const std::string &s)`

5.1.1 Typedef Documentation

5.1.1.1 ld

```
typedef unsigned int basics::ld
```

5.1.2 Function Documentation

5.1.2.1 fnv32() [1/2]

```
template<size_t LENGTH>  
uint32_t basics::fnv32 (  
    const char(&) chars[LENGTH])
```

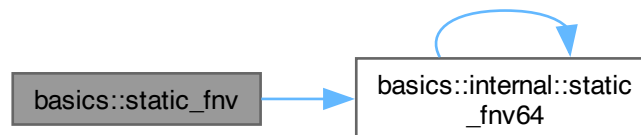
5.1.2.2 fnv32() [2/2]

```
uint32_t basics::fnv32 (
    const std::string & s) [inline]
```

5.1.2.3 static_fnv()

```
template<size_t LENGTH>
unsigned basics::static_fnv (
    const char(&) chars[LENGTH]) [constexpr]
```

Here is the call graph for this function:



5.1.2.4 static_fnv32()

```
template<size_t LENGTH>
uint32_t basics::static_fnv32 (
    const char(&) chars[LENGTH]) [constexpr]
```

Implements the Fowler–Noll–Vo hash function (FNV-1a) which calculates a 32 bit hash code from a C string literal (or array of chars) at compile time.

Template Parameters

<i>LENGTH</i>	Length of the string literal or array of chars.
---------------	---

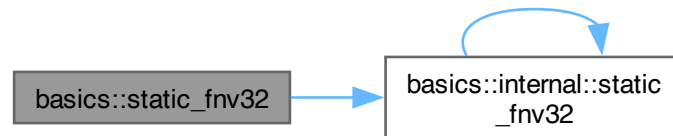
Parameters

<i>chars</i>	The string literal or array of chars to hash.
--------------	---

Returns

The hash code.

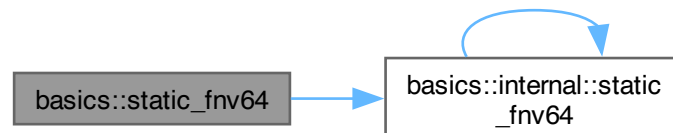
Here is the call graph for this function:

**5.1.2.5 static_fnv64()**

```

template<size_t LENGTH>
uint64_t basics::static_fnv64 (
    const char(&) chars[LENGTH]) [constexpr]
  
```

Here is the call graph for this function:

**5.2 basics::internal Namespace Reference****Functions**

- `template<size_t LENGTH>`
`constexpr uint32_t static_fnv32 (const char *chars)`
- `template<> constexpr uint32_t static_fnv32< 1 > (const char *)`
- `template<size_t LENGTH>`
`constexpr uint64_t static_fnv64 (const char *chars)`
- `template<> constexpr uint64_t static_fnv64< 1 > (const char *)`

Variables

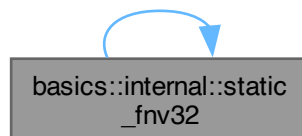
- constexpr uint32_t [fnv_basis_32](#) = 0x811c9dc5u
- constexpr uint32_t [fnv_prime_32](#) = 0x01000193u
- constexpr uint64_t [fnv_basis_64](#) = 0xcbf29ce484222325u
- constexpr uint64_t [fnv_prime_64](#) = 0x000001000000001b3u

5.2.1 Function Documentation

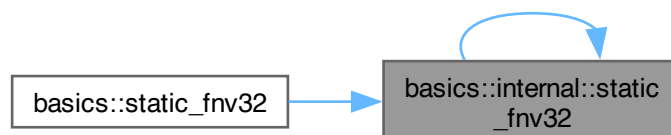
5.2.1.1 static_fnv32()

```
template<size_t LENGTH>
uint32_t basics::internal::static_fnv32 (
    const char * chars) [constexpr]
```

Here is the call graph for this function:



Here is the caller graph for this function:



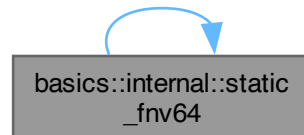
5.2.1.2 static_fnv32< 1 >()

```
template<>
uint32_t basics::internal::static_fnv32< 1 > (
    const char * ) [constexpr]
```

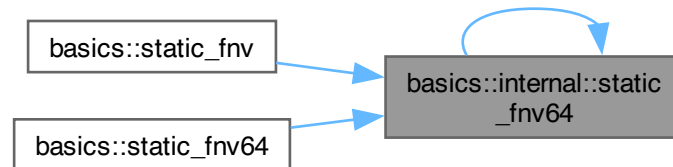
5.2.1.3 static_fnv64()

```
template<size_t LENGTH>
uint64_t basics::internal::static_fnv64 (
    const char * chars) [constexpr]
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.1.4 static_fnv64< 1 >()

```
template<>
uint64_t basics::internal::static_fnv64< 1 > (
    const char * ) [constexpr]
```

5.2.2 Variable Documentation

5.2.2.1 fnv_basis_32

```
uint32_t basics::internal::fnv_basis_32 = 0x811c9dc5u [constexpr]
```

5.2.2.2 fnv_basis_64

```
uint64_t basics::internal::fnv_basis_64 = 0xcbf29ce484222325u [constexpr]
```

5.2.2.3 fnv_prime_32

```
uint32_t basics::internal::fnv_prime_32 = 0x01000193u [constexpr]
```

5.2.2.4 fnv_prime_64

```
uint64_t basics::internal::fnv_prime_64 = 0x000001000000001b3u [constexpr]
```

5.3 Ragot Namespace Reference

Classes

- class [Assets](#)
Manages the asset paths for the application.
- class [Camera](#)
Represents a camera in a 3D space, managing its properties and transformations.
- struct [Camera_Transform](#)
Represents the transformation of a camera in 3D space.
- class [Component](#)
Base class for components in the [Ragot](#) engine.
- struct [coordinates_t](#)
Represents 2D coordinates.
- class [Driver_ST7789](#)
Driver for the ST7789 LCD panel.
- class [DriverEK79007](#)
Driver for the EK79007 LCD panel.
- class [DriverLCD](#)
Base class for LCD drivers.
- class [ExtrudeMesh](#)
Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the [Mesh](#) class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.
- struct [face_t](#)
Represents a face in a 3D mesh.
- class [Fragment_Shader](#)
Class for managing an OpenGL fragment shader.
- class [FrameBuffer](#)
Class to manage a frame buffer for rendering graphics.
- class [Logger](#)
Singleton logger class for the [Ragot](#) engine.
- class [Mesh](#)
Represents a 3D mesh in the [Ragot](#) engine.
- struct [mesh_info_t](#)
Represents information about a mesh.
- class [MeshSerializer](#)
Singleton class to serialize [Mesh](#) objects to OBJ file format.
- class [Node](#)
Represents a node in a scene graph for 3D rendering.

- class [PSRAMAllocator](#)
Custom memory allocator for PSRAM.
- class [Rasterizer](#)
Class for rasterizing polygons in a frame buffer.
- class [Renderer](#)
Class for rendering scenes in the [Ragot](#) engine.
- class [RevolutionMesh](#)
Class for generating revolution meshes.
- class [Scene](#)
Class for managing a 3D scene.
- class [Shader](#)
Class for managing an OpenGL shader.
- class [Shader_Program](#)
Class for managing an OpenGL shader program.
- class [Sync_Queue](#)
A thread-safe queue implementation.
- class [Thread_Pool](#)
A thread pool for managing concurrent tasks.
- class [Transform](#)
A class representing a 3D transformation with position, rotation, and scale.
- struct [transform_t](#)
Represents a transformation in 3D space.
- class [Vertex_Shader](#)
Class for managing an OpenGL vertex shader.
- struct [vertex_t](#)
Represents a vertex in 3D space.
- class [Window](#)
Class for managing an SDL window with OpenGL context.

Typedefs

- using [RGB565](#) = uint16_t
- using [RGB888](#) = uint32_t
- using [RGBA8888](#) = uint32_t
- using [RGB8](#) = uint8_t
Color Index.
- using [Matrix4x4](#) = glm::mat4

Enumerations

- enum [render_flag_t](#) : uint8_t { [RENDER_NONE](#) , [RENDER_REVOLUTION](#) , [RENDER_EXTRUDE](#) , [RENDER_MAX](#) }
Flags for rendering types.
- enum [Buffer](#) : uint8_t { [CURRENT_BUFFER](#) = (1 << 0) , [NEXT_BUFFER](#) = (1 << 1) , [MAX_BUFFER](#) = (1 << 2) }
Enum to represent the different buffers in a frame buffer.

Functions

- static bool [panel_refresh_callback](#) (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data_t *edata, void *user_ctx)
- static bool [panel_refresh_callback](#) (esp_lcd_panel_io_handle_t panel, esp_lcd_panel_io_event_data_t *edata, void *user_ctx)
- template<typename T, uint16_t F1, typename U, uint16_t F2>
bool [operator==](#) (const [PSRAMAllocator](#)< T, F1 > &, const [PSRAMAllocator](#)< U, F2 > &)
Equality operator for [PSRAMAllocator](#).
- template<typename T, uint16_t F1, typename U, uint16_t F2>
bool [operator!=](#) (const [PSRAMAllocator](#)< T, F1 > &a, const [PSRAMAllocator](#)< U, F2 > &b)
Inequality operator for [PSRAMAllocator](#).
- template<typename Inside, typename Intersect>
static std::vector< glm::fvec4 > [clipAgainstPlane](#) (const std::vector< glm::fvec4 > &in, Inside inside, Intersect intersect)

Variables

- [Assets](#) & [assets](#) = [Assets::instance](#)()
- constexpr float [PI](#) = 3.141592653f
Mathematical constant PI.
- static const char * [TAG](#) = "DriverEK79007"
- [Logger](#) & [logger](#) = [Logger::instance](#)()
- [MeshSerializer](#) & [serializer](#) = [MeshSerializer::instance](#)()
- template<class COLOR_BUFFER_TYPE>
int [Rasterizer](#)< [COLOR_BUFFER_TYPE](#) >::offset_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
int [Rasterizer](#)< [COLOR_BUFFER_TYPE](#) >::offset_cache1 [1024]
- template<class COLOR_BUFFER_TYPE>
int [Rasterizer](#)< [COLOR_BUFFER_TYPE](#) >::z_cache0 [1024]
- template<class COLOR_BUFFER_TYPE>
int [Rasterizer](#)< [COLOR_BUFFER_TYPE](#) >::z_cache1 [1024]
- static const char * [RENDERER_TAG](#) = "Renderer"
- static const char * [MAIN_TAG](#) = "Main"
- [Thread_Pool](#) & [thread_pool](#) = [Thread_Pool::instance](#) ()

5.3.1 Typedef Documentation

5.3.1.1 Matrix4x4

```
using Ragot::Matrix4x4 = glm::mat4
```

5.3.1.2 RGB565

```
using Ragot::RGB565 = uint16_t
```

5.3.1.3 RGB8

```
using Ragot::RGB8 = uint8_t
```

Color Index.

5.3.1.4 RGB888

```
using Ragot::RGB888 = uint32_t
```

5.3.1.5 RGBA8888

```
using Ragot::RGBA8888 = uint32_t
```

5.3.2 Enumeration Type Documentation

5.3.2.1 Buffer

```
enum Ragot::Buffer : uint8_t
```

Enum to represent the different buffers in a frame buffer.

This enum defines the constants for the current buffer, next buffer, and maximum buffer.

Enumerator

CURRENT_BUFFER	
NEXT_BUFFER	
MAX_BUFFER	

5.3.2.2 render_flag_t

```
enum Ragot::render_flag_t : uint8_t
```

Flags for rendering types.

This enumeration defines different rendering flags that can be used to specify how a mesh should be rendered.

Enumerator

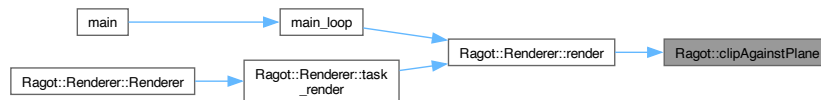
RENDER_NONE	
RENDER_REVOLUTION	
RENDER_EXTRUDE	
RENDER_MAX	

5.3.3 Function Documentation

5.3.3.1 clipAgainstPlane()

```
template<typename Inside, typename Intersect>
static std::vector< glm::fvec4 > Ragot::clipAgainstPlane (
    const std::vector< glm::fvec4 > & in,
    Inside inside,
    Intersect intersect) [static]
```

Here is the caller graph for this function:



5.3.3.2 operator!=(())

```
template<typename T, uint16_t F1, typename U, uint16_t F2>
bool Ragot::operator!=(
    const PSRAMAllocator< T, F1 > & a,
    const PSRAMAllocator< U, F2 > & b)
```

Inequality operator for [PSRAMAllocator](#).

This operator checks if two PSRAMAllocators are not equal based on their flags.

Template Parameters

<i>T</i>	The type of the first allocator.
<i>F1</i>	The flag of the first allocator.
<i>U</i>	The type of the second allocator.
<i>F2</i>	The flag of the second allocator.

Parameters

<i>a</i>	The first PSRAMAllocator .
<i>b</i>	The second PSRAMAllocator .

Returns

true if the flags are not equal, false otherwise.

5.3.3.3 operator==(())

```
template<typename T, uint16_t F1, typename U, uint16_t F2>
bool Ragot::operator==(
    const PSRAMAllocator< T, F1 > & ,
    const PSRAMAllocator< U, F2 > & )
```

Equality operator for [PSRAMAllocator](#).

This operator checks if two PSRAMAllocators are equal based on their flags.

Template Parameters

<i>T</i>	The type of the first allocator.
<i>F1</i>	The flag of the first allocator.
<i>U</i>	The type of the second allocator.
<i>F2</i>	The flag of the second allocator.

Parameters

<i>a</i>	The first PSRAMAllocator .
<i>b</i>	The second PSRAMAllocator .

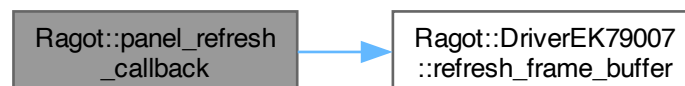
Returns

true if the flags are equal, false otherwise.

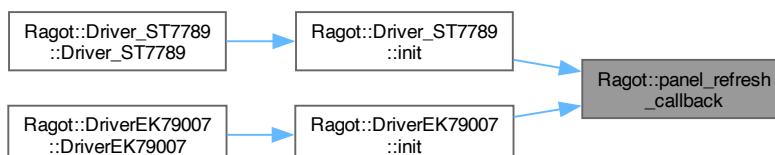
5.3.3.4 `panel_refresh_callback()` [1/2]

```
static bool Ragot::panel_refresh_callback (
    esp_lcd_panel_handle_t panel,
    esp_lcd_dpi_panel_event_data_t * edata,
    void * user_ctx) [static]
```

Here is the call graph for this function:



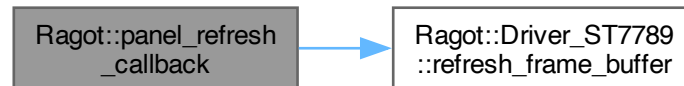
Here is the caller graph for this function:



5.3.3.5 panel_refresh_callback() [2/2]

```
static bool Ragot::panel_refresh_callback (
    esp_lcd_panel_io_handle_t panel,
    esp_lcd_panel_io_event_data_t * edata,
    void * user_ctx) [static]
```

Here is the call graph for this function:



5.3.4 Variable Documentation

5.3.4.1 assets

```
Assets & Ragot::assets = Assets::instance()
```

5.3.4.2 logger

```
Logger & Ragot::logger = Logger::instance()
```

5.3.4.3 MAIN_TAG

```
const char* Ragot::MAIN_TAG = "Main" [static]
```

5.3.4.4 PI

```
float Ragot::PI = 3.141592653f [constexpr]
```

Mathematical constant PI.

This constant represents the value of (pi), which is approximately 3.141592653.

5.3.4.5 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0[1024]
```

5.3.4.6 Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1[1024]
```

5.3.4.7 Rasterizer< COLOR_BUFFER_TYPE >::z_cache0

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache0[1024]
```

5.3.4.8 Rasterizer< COLOR_BUFFER_TYPE >::z_cache1

```
template<class COLOR_BUFFER_TYPE>
int Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache1[1024]
```

5.3.4.9 RENDERER_TAG

```
const char* Ragot::RENDERER_TAG = "Renderer" [static]
```

5.3.4.10 serializer

```
MeshSerializer & Ragot::serializer = MeshSerializer::instance()
```

5.3.4.11 TAG

```
const char* Ragot::TAG = "DriverEK79007" [static]
```

5.3.4.12 thread_pool

```
Thread_Pool & Ragot::thread_pool = Thread_Pool::instance ()
```


Chapter 6

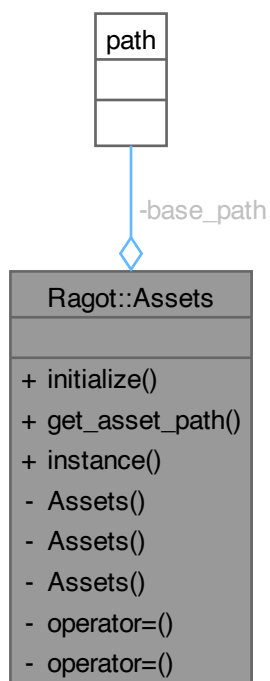
Class Documentation

6.1 Ragot::Assets Class Reference

Manages the asset paths for the application.

```
#include <Assets.hpp>
```

Collaboration diagram for Ragot::Assets:



Public Member Functions

- void `initialize` (const string &executable_file_path)
Initializes the base path for assets based on the executable file path.
- path `get_asset_path` (const string &asset_name)
Gets the full path of an asset by its name.

Static Public Member Functions

- static `Assets` & `instance` ()
Returns the singleton instance of the `Assets` class.

Private Member Functions

- `Assets` ()=default
Construct a new `Assets` object.
- `Assets` (const `Assets` &)=delete
Destroy the `Assets` object.
- `Assets` (const `Assets` &&)=delete
Move constructor is deleted to prevent moving the `Assets` instance.
- `Assets` & `operator=` (const `Assets` &)=delete
Assignment operator is deleted to prevent copying the `Assets` instance.
- `Assets` & `operator=` (const `Assets` &&)=delete
Move assignment operator is deleted to prevent moving the `Assets` instance.

Private Attributes

- path `base_path` = `"./"`

6.1.1 Detailed Description

Manages the asset paths for the application.

The `Assets` class provides a singleton instance to manage asset paths, allowing for easy retrieval of asset files. It initializes the base path based on the executable file path, and provides a method to get the full path of an asset by its name.

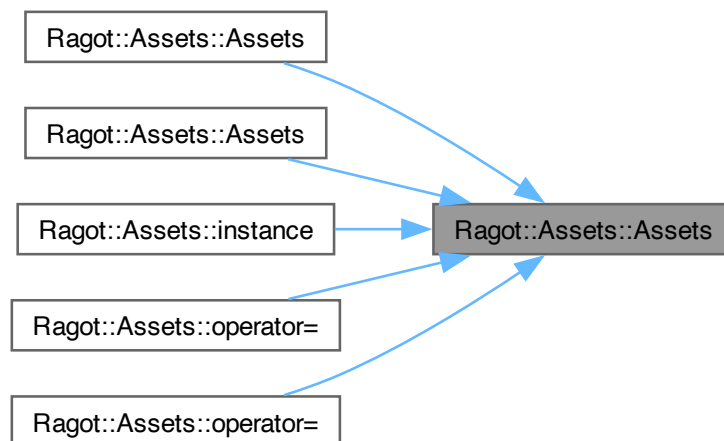
6.1.2 Constructor & Destructor Documentation

6.1.2.1 Assets() [1/3]

```
Ragot::Assets::Assets () [private], [default]
```

Construct a new [Assets](#) object.

Here is the caller graph for this function:



6.1.2.2 Assets() [2/3]

```
Ragot::Assets::Assets (
    const Assets & ) [private], [delete]
```

Destroy the [Assets](#) object.

Here is the call graph for this function:



6.1.2.3 Assets() [3/3]

```
Ragot::Assets::Assets (
    const Assets && ) [private], [delete]
```

Move constructor is deleted to prevent moving the [Assets](#) instance.

Here is the call graph for this function:



6.1.3 Member Function Documentation

6.1.3.1 get_asset_path()

```
path Ragot::Assets::get_asset_path (
    const string & asset_name) [inline]
```

Gets the full path of an asset by its name.

This method constructs the full path to an asset file by appending the asset name to the base path.

Parameters

<i>asset_name</i>	The name of the asset file.
-------------------	-----------------------------

Returns

path The full path to the asset file.

6.1.3.2 initialize()

```
void Ragot::Assets::initialize (
    const string & executable_file_path) [inline]
```

Initializes the base path for assets based on the executable file path.

This method sets the base path to the "assets" directory relative to the executable's location. In debug mode, it uses a relative path to the assets directory. In release mode, it sets the base path to a specific directory.

Parameters

<i>executable_file_path</i>	The path of the executable file.
-----------------------------	----------------------------------

6.1.3.3 instance()

```
static Assets & Ragot::Assets::instance () [inline], [static]
```

Returns the singleton instance of the [Assets](#) class.

This method ensures that only one instance of the [Assets](#) class exists throughout the application.

Returns

[Assets](#)& Reference to the singleton instance of [Assets](#).

Here is the call graph for this function:



6.1.3.4 operator=() [1/2]

```
Assets & Ragot::Assets::operator= (   
    const Assets && ) [private], [delete]
```

Move assignment operator is deleted to prevent moving the [Assets](#) instance.

Here is the call graph for this function:



6.1.3.5 operator=() [2/2]

```
Assets & Ragot::Assets::operator= (   
    const Assets & ) [private], [delete]
```

Assignment operator is deleted to prevent copying the [Assets](#) instance.

Here is the call graph for this function:



6.1.4 Member Data Documentation

6.1.4.1 base_path

```
path Ragot::Assets::base_path = "./" [private]
```

The documentation for this class was generated from the following file:

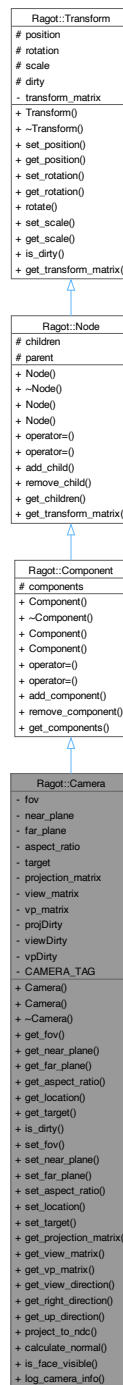
- main/[Assets.hpp](#)

6.2 Ragot::Camera Class Reference

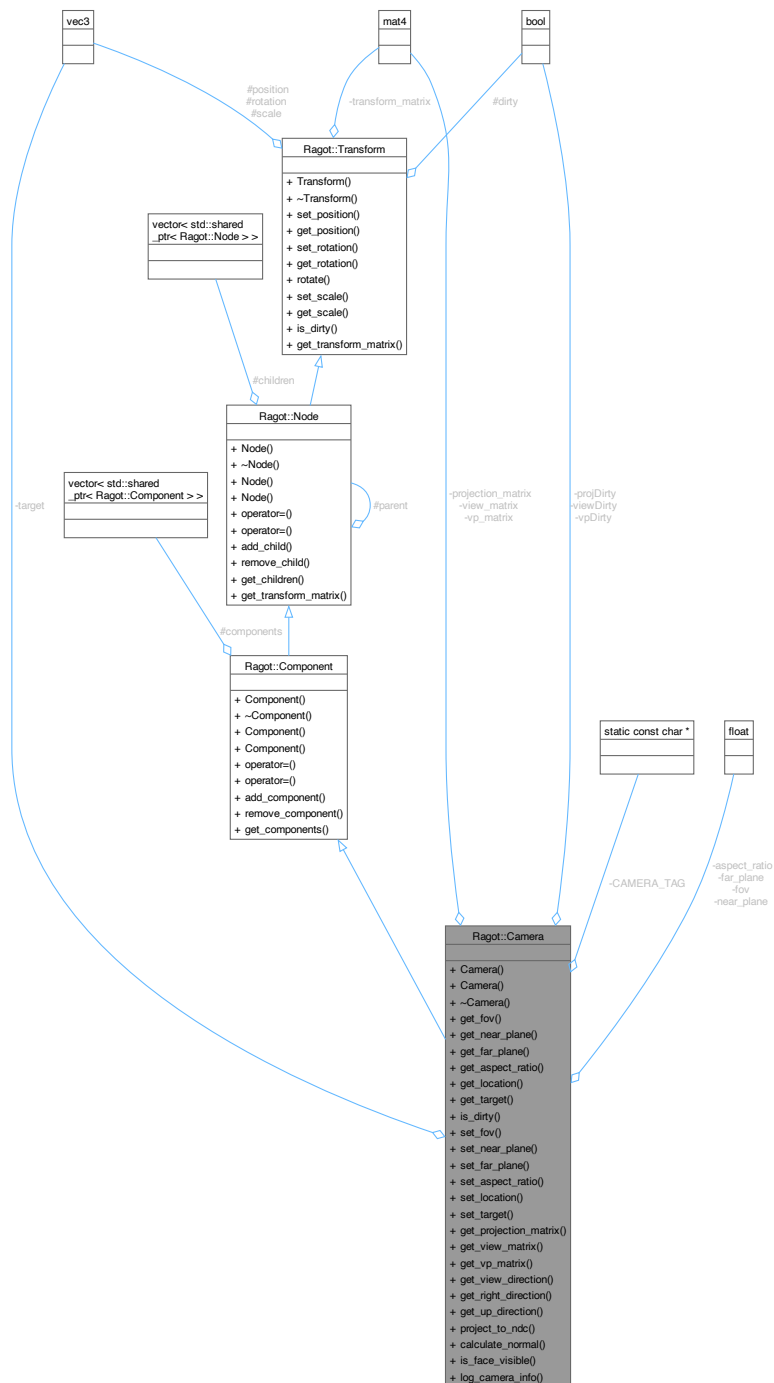
Represents a camera in a 3D space, managing its properties and transformations.

```
#include <Camera.hpp>
```

Inheritance diagram for Ragot::Camera:



Collaboration diagram for Ragot::Camera:



Public Types

- using `Matrix4x4` = `glm::mat4`

Public Member Functions

- `Camera()`=delete

- Delete default constructor to prevent instantiation without parameters.*

 - `Camera` (float `aspect_ratio`=1.f, float `near_plane`=1.f, float `far_plane`=100.f, float `fov_deg`=60.f)

Constructs a `Camera` object with specified parameters.
- `~Camera` ()=default

Default destructor for `Camera` class.
- float `get_fov` () const

Returns the camera's field of view in degrees.
- float `get_near_plane` () const

Returns the camera's near clipping plane distance.
- float `get_far_plane` () const

Returns the camera's far clipping plane distance.
- float `get_aspect_ratio` () const

Returns the camera's aspect ratio (width/height).
- glm::vec3 `get_location` () const

Returns the camera's position in world space.
- glm::vec3 `get_target` () const

Returns the camera's target position in world space.
- bool `is_dirty` () const

Returns the camera's position in world space.
- void `set_fov` (float deg)

Set the fov object.
- void `set_near_plane` (float np)

Set the near clipping plane distance.
- void `set_far_plane` (float fp)

Set the far clipping plane distance.
- void `set_aspect_ratio` (float ar)

Set the aspect ratio of the camera.
- void `set_location` (const glm::vec3 &p)

Set the camera's position in world space.
- void `set_target` (const glm::vec3 &t)

Set the camera's target position in world space.
- const `Matrix4x4` & `get_projection_matrix` () const

Get the projection matrix object.
- const `Matrix4x4` & `get_view_matrix` () const

Get the view matrix object.
- const `Matrix4x4` & `get_vp_matrix` () const

Get the vp matrix object.
- glm::vec3 `get_view_direction` () const

Get the view direction object.
- glm::vec3 `get_right_direction` () const

Get the right direction object.
- glm::vec3 `get_up_direction` () const

Get the up direction object.
- glm::vec3 `project_to_ndc` (const glm::vec4 &worldPos) const
- `vertex_t` `calculate_normal` (const `vertex_t` &v1, const `vertex_t` &v2, const `vertex_t` &v3)

Calculate the normal vector of a face defined by three vertices.
- bool `is_face_visible` (const `vertex_t` &v1, const `vertex_t` &v2, const `vertex_t` &v3)

Check if a face defined by three vertices is visible from the camera's perspective.
- void `log_camera_info` () const

Logs the camera's properties for debugging purposes.

Public Member Functions inherited from Ragot::Component

- [Component](#) ()=default
Default constructor for the [Component](#) class.
- virtual [~Component](#) ()=default
Default virtual destructor for the [Component](#) class.
- [Component](#) (const [Component](#) &)=delete
Deleted copy constructor for the [Component](#) class.
- [Component](#) (const [Component](#) &&)=delete
Deleted move constructor for the [Component](#) class.
- [Component](#) & [operator=](#) (const [Component](#) &)=delete
Deleted assignment operator for the [Component](#) class.
- [Component](#) & [operator=](#) (const [Component](#) &&)=delete
Deleted move assignment operator for the [Component](#) class.
- void [add_component](#) (std::shared_ptr< [Component](#) > component)
Adds a component to the collection.
- void [remove_component](#) (std::shared_ptr< [Component](#) > component)
Removes a component from the collection.
- const std::vector< std::shared_ptr< [Component](#) > > [get_components](#) () const
Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- [Node](#) ()=default
Default constructor for [Node](#). Initializes an empty node with no parent and no children.
- virtual [~Node](#) ()=default
Default destructor for [Node](#). Cleans up the node and its children.
- [Node](#) (const [Node](#) &)=delete
Deleted copy constructor for [Node](#). Prevents copying of [Node](#) instances.
- [Node](#) (const [Node](#) &&)=delete
Deleted move constructor for [Node](#). Prevents moving of [Node](#) instances.
- [Node](#) & [operator=](#) (const [Node](#) &)=delete
Deleted assignment operator for [Node](#). Prevents assignment of [Node](#) instances.
- [Node](#) & [operator=](#) (const [Node](#) &&)=delete
Deleted move assignment operator for [Node](#). Prevents moving of [Node](#) instances.
- void [add_child](#) (std::shared_ptr< [Node](#) > child)
Get the parent node.
- void [remove_child](#) (std::shared_ptr< [Node](#) > child)
Remove a child node.
- const std::vector< std::shared_ptr< [Node](#) > > & [get_children](#) () const
Get the parent node.
- mat4 [get_transform_matrix](#) () override
Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- [Transform](#) ()
Default constructor for the [Transform](#) class.
- virtual [~Transform](#) ()=default
Virtual destructor for the [Transform](#) class.
- void [set_position](#) (const vec3 &pos)
Sets the position of the object.
- vec3 [get_position](#) () const
Gets the current position of the object.
- void [set_rotation](#) (const vec3 &rot)
Moves the object by a specified vector.
- vec3 [get_rotation](#) () const
Gets the current rotation of the object.
- void [rotate](#) (const float angle, const vec3 &axis)
Rotates the object by a specified angle around a given axis.
- void [set_scale](#) (const vec3 &scale)
Sets the scale of the object.
- vec3 [get_scale](#) () const
Sets the scale of the object uniformly.
- bool [is_dirty](#) () const
Checks if the transformation matrix is dirty (needs recalculation).

Private Attributes

- float [fov](#)
vertical field of view in degrees
- float [near_plane](#)
near clipping plane distance
- float [far_plane](#)
far clipping plane distance
- float [aspect_ratio](#)
aspect ratio (width/height)
- glm::vec3 [target](#)
world space look-at target
- [Matrix4x4](#) [projection_matrix](#)
cached projection matrix
- [Matrix4x4](#) [view_matrix](#)
cached view matrix
- [Matrix4x4](#) [vp_matrix](#)
cached view-projection matrix
- bool [projDirty](#) = true
flag to indicate if projection matrix is dirty
- bool [viewDirty](#) = true
flag to indicate if view matrix is dirty
- bool [vpDirty](#) = true
flag to indicate if view-projection matrix is dirty

Static Private Attributes

- static const char * [CAMERA_TAG](#) = "Camera"
Tag for logging camera-related messages.

Additional Inherited Members

Protected Attributes inherited from [Ragot::Component](#)

- std::vector< std::shared_ptr< [Component](#) > > [components](#)
Collection of components managed by this [Component](#) instance.

Protected Attributes inherited from [Ragot::Node](#)

- std::vector< std::shared_ptr< [Node](#) > > [children](#)
List of child nodes.
- [Node](#) * [parent](#) = nullptr
Pointer to the parent node.

Protected Attributes inherited from [Ragot::Transform](#)

- vec3 [position](#)
The position of the object in 3D space.
- vec3 [rotation](#)
The rotation of the object in degrees around each axis (x, y, z).
- vec3 [scale](#)
The scale of the object in 3D space, default is (1, 1, 1).
- bool [dirty](#) = true
Flag indicating whether the transformation matrix needs to be recalculated.

6.2.1 Detailed Description

Represents a camera in a 3D space, managing its properties and transformations.

The [Camera](#) class provides functionality to set and get camera properties such as field of view, near and far planes, aspect ratio, and target position. It also computes projection, view, and combined view-projection matrices, and provides methods for projecting world coordinates to normalized device coordinates (NDC).

6.2.2 Member Typedef Documentation

6.2.2.1 Matrix4x4

```
using Ragot::Camera::Matrix4x4 = glm::mat4
```

6.2.3 Constructor & Destructor Documentation

6.2.3.1 Camera() [1/2]

```
Ragot::Camera::Camera () [delete]
```

Delete default constructor to prevent instantiation without parameters.

6.2.3.2 Camera() [2/2]

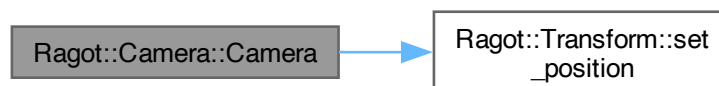
```
Ragot::Camera::Camera (
    float aspect_ratio = 1.f,
    float near_plane = 1.f,
    float far_plane = 100.f,
    float fov_deg = 60.f) [inline]
```

Constructs a [Camera](#) object with specified parameters.

Parameters

<i>aspect_ratio</i>	Aspect ratio of the camera (default is 1.0).
<i>near_plane</i>	Distance to the near clipping plane (default is 1.0).
<i>far_plane</i>	Distance to the far clipping plane (default is 100.0).
<i>fov_deg</i>	Vertical field of view in degrees (default is 60.0).

Here is the call graph for this function:



6.2.3.3 ~Camera()

```
Ragot::Camera::~~Camera () [default]
```

Default destructor for [Camera](#) class.

6.2.4 Member Function Documentation

6.2.4.1 calculate_normal()

```
vertex_t Ragot::Camera::calculate_normal (
    const vertex_t & v1,
    const vertex_t & v2,
    const vertex_t & v3)
```

Calculate the normal vector of a face defined by three vertices.

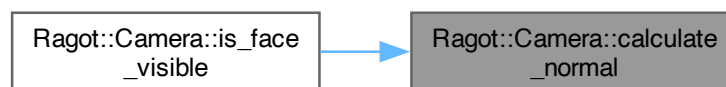
Parameters

<code>v1</code>	First vertex of the face.
<code>v2</code>	Second vertex of the face.
<code>v3</code>	Third vertex of the face.

Returns

[vertex_t](#)

Here is the caller graph for this function:

**6.2.4.2 get_aspect_ratio()**

```
float Ragot::Camera::get_aspect_ratio () const [inline]
```

Returns the camera's aspect ratio (width/height).

Returns

float The aspect ratio of the camera.

6.2.4.3 get_far_plane()

```
float Ragot::Camera::get_far_plane () const [inline]
```

Returns the camera's far clipping plane distance.

Returns

float The distance to the far clipping plane.

6.2.4.4 get_fov()

```
float Ragot::Camera::get_fov () const [inline]
```

Returns the camera's field of view in degrees.

Returns

float The field of view in degrees.

6.2.4.5 get_location()

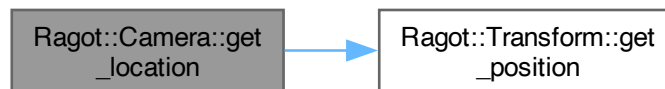
```
glm::vec3 Ragot::Camera::get_location () const [inline]
```

Returns the camera's position in world space.

Returns

glm::vec3 The position of the camera.

Here is the call graph for this function:



6.2.4.6 get_near_plane()

```
float Ragot::Camera::get_near_plane () const [inline]
```

Returns the camera's near clipping plane distance.

Returns

float The distance to the near clipping plane.

6.2.4.7 get_projection_matrix()

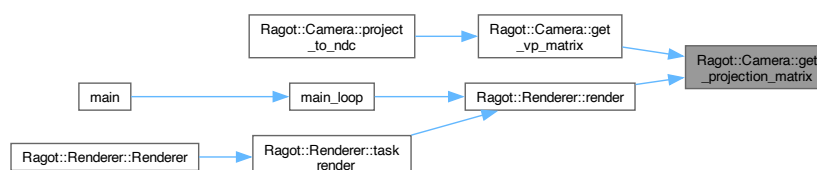
```
const Matrix4x4 & Ragot::Camera::get_projection_matrix () const [inline]
```

Get the projection matrix object.

Returns

const Matrix4x4&

Here is the caller graph for this function:



6.2.4.8 get_right_direction()

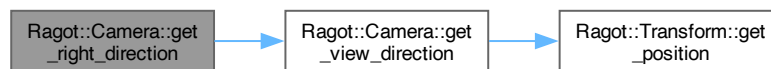
```
glm::vec3 Ragot::Camera::get_right_direction () const [inline]
```

Get the right direction object.

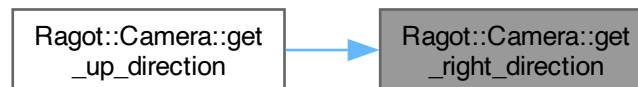
Returns

glm::vec3

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.9 get_target()

```
glm::vec3 Ragot::Camera::get_target () const [inline]
```

Returns the camera's target position in world space.

Returns

glm::vec3 The target position of the camera.

6.2.4.10 get_up_direction()

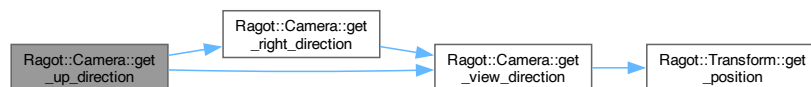
```
glm::vec3 Ragot::Camera::get_up_direction () const [inline]
```

Get the up direction object.

Returns

glm::vec3

Here is the call graph for this function:



6.2.4.11 get_view_direction()

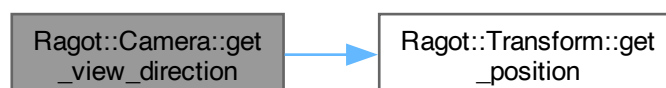
```
glm::vec3 Ragot::Camera::get_view_direction () const [inline]
```

Get the view direction object.

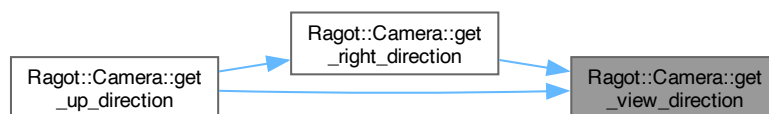
Returns

glm::vec3

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.12 get_view_matrix()

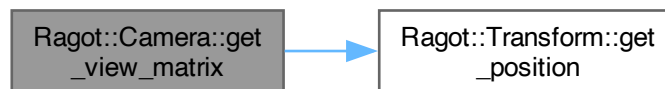
```
const Matrix4x4 & Ragot::Camera::get_view_matrix () const [inline]
```

Get the view matrix object.

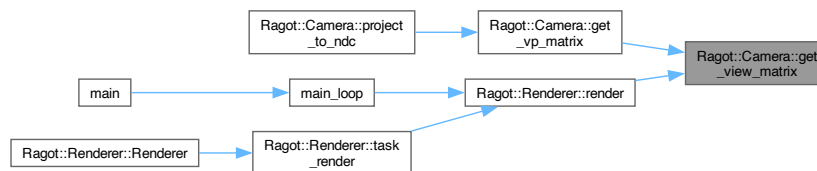
Returns

const Matrix4x4&

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.13 get_vp_matrix()

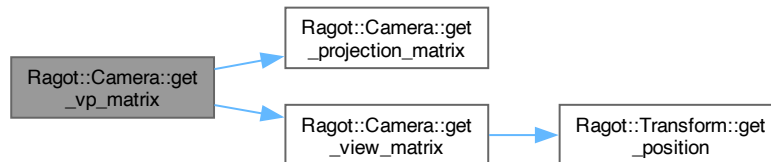
```
const Matrix4x4 & Ragot::Camera::get_vp_matrix () const [inline]
```

Get the vp matrix object.

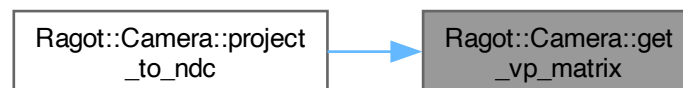
Returns

const [Matrix4x4](#)&

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.14 is_dirty()

```
bool Ragot::Camera::is_dirty () const [inline]
```

Returns the camera's position in world space.

Returns

`glm::vec3` The position of the camera.

6.2.4.15 is_face_visible()

```
bool Ragot::Camera::is_face_visible (
    const vertex\_t & v1,
    const vertex\_t & v2,
    const vertex\_t & v3)
```

Check if a face defined by three vertices is visible from the camera's perspective.

A face is considered visible if its normal vector points towards the camera.

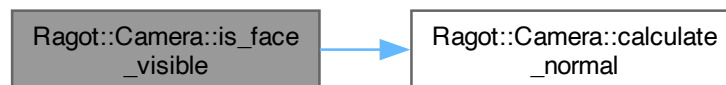
Parameters

<i>v1</i>	First vertex of the face.
<i>v2</i>	Second vertex of the face.
<i>v3</i>	Third vertex of the face.

Returns

true if the face is visible, false otherwise.

Here is the call graph for this function:

6.2.4.16 `log_camera_info()`

```
void Ragot::Camera::log_camera_info () const
```

Logs the camera's properties for debugging purposes.

This method logs the camera's position, target, field of view, near and far planes, and aspect ratio.

6.2.4.17 `project_to_ndc()`

```
glm::vec3 Ragot::Camera::project_to_ndc (
    const glm::vec4 & worldPos) const [inline]
```

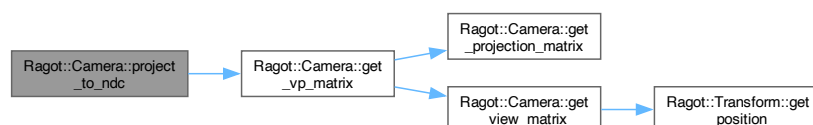
Parameters

<i>worldPos</i>	
-----------------	--

Returns

glm::vec3

Here is the call graph for this function:



6.2.4.18 set_aspect_ratio()

```
void Ragot::Camera::set_aspect_ratio (
    float ar) [inline]
```

Set the aspect ratio of the camera.

Parameters

<i>ar</i>	Aspect ratio (width/height).
-----------	------------------------------

6.2.4.19 set_far_plane()

```
void Ragot::Camera::set_far_plane (
    float fp) [inline]
```

Set the far clipping plane distance.

Parameters

<i>fp</i>	Distance to the far clipping plane.
-----------	-------------------------------------

6.2.4.20 set_fov()

```
void Ragot::Camera::set_fov (
    float deg) [inline]
```

Set the fov object.

Parameters

<i>deg</i>	Field of view in degrees.
------------	---------------------------

6.2.4.21 set_location()

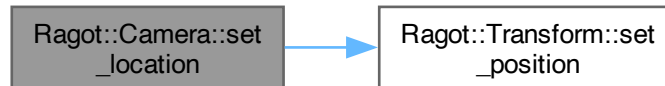
```
void Ragot::Camera::set_location (
    const glm::vec3 & p) [inline]
```

Set the camera's position in world space.

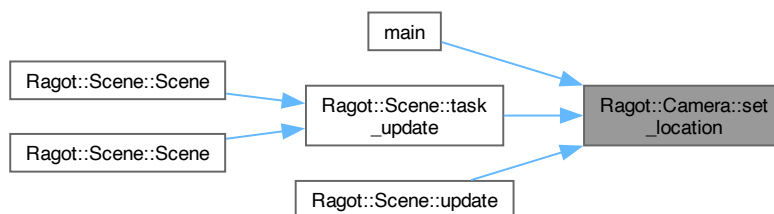
Parameters

<i>p</i>	Position of the camera.
----------	-------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4.22 set_near_plane()

```
void Ragot::Camera::set_near_plane (
    float np) [inline]
```

Set the near clipping plane distance.

Parameters

<i>np</i>	Distance to the near clipping plane.
-----------	--------------------------------------

6.2.4.23 set_target()

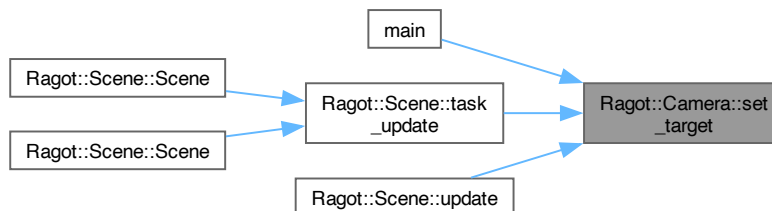
```
void Ragot::Camera::set_target (
    const glm::vec3 & t) [inline]
```

Set the camera's target position in world space.

Parameters

<i>t</i>	Target position of the camera.
----------	--------------------------------

Here is the caller graph for this function:

**6.2.5 Member Data Documentation****6.2.5.1 aspect_ratio**

```
float Ragot::Camera::aspect_ratio [private]
```

aspect ratio (width/height)

6.2.5.2 CAMERA_TAG

```
const char * Ragot::Camera::CAMERA_TAG = "Camera" [static], [private]
```

Tag for logging camera-related messages.

6.2.5.3 far_plane

```
float Ragot::Camera::far_plane [private]
```

far clipping plane distance

6.2.5.4 fov

```
float Ragot::Camera::fov [private]
```

vertical field of view in degrees

6.2.5.5 near_plane

```
float Ragot::Camera::near_plane [private]
```

near clipping plane distance

6.2.5.6 projDirty

```
bool Ragot::Camera::projDirty = true [mutable], [private]
```

flag to indicate if projection matrix is dirty

6.2.5.7 projection_matrix

```
Matrix4x4 Ragot::Camera::projection_matrix [mutable], [private]
```

cached projection matrix

6.2.5.8 target

```
glm::vec3 Ragot::Camera::target [private]
```

world space look-at target

6.2.5.9 view_matrix

```
Matrix4x4 Ragot::Camera::view_matrix [mutable], [private]
```

cached view matrix

6.2.5.10 viewDirty

```
bool Ragot::Camera::viewDirty = true [mutable], [private]
```

flag to indicate if view matrix is dirty

6.2.5.11 vp_matrix

```
Matrix4x4 Ragot::Camera::vp_matrix [mutable], [private]
```

cached view-projection matrix

6.2.5.12 vpDirty

```
bool Ragot::Camera::vpDirty = true [mutable], [private]
```

flag to indicate if view-projection matrix is dirty

The documentation for this class was generated from the following files:

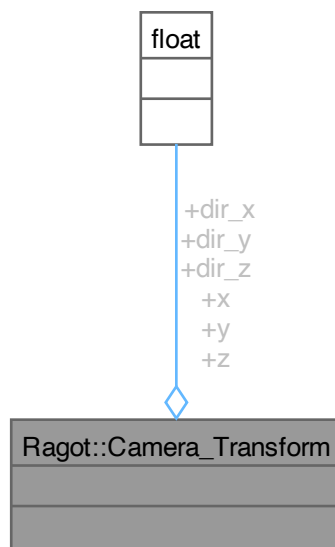
- main/[Camera.hpp](#)
- main/[Camera.cpp](#)

6.3 Ragot::Camera_Transform Struct Reference

Represents the transformation of a camera in 3D space.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::Camera_Transform:



Public Attributes

- float [x](#)
- float [y](#)
- float [z](#)
- float [dir_x](#)
- float [dir_y](#)
- float [dir_z](#)

6.3.1 Detailed Description

Represents the transformation of a camera in 3D space.

This structure holds the position and direction of the camera, allowing for transformations in a 3D environment.

6.3.2 Member Data Documentation

6.3.2.1 dir_x

```
float Ragot::Camera_Transform::dir_x
```

6.3.2.2 dir_y

```
float Ragot::Camera_Transform::dir_y
```

6.3.2.3 dir_z

```
float Ragot::Camera_Transform::dir_z
```

6.3.2.4 x

```
float Ragot::Camera_Transform::x
```

6.3.2.5 y

```
float Ragot::Camera_Transform::y
```

6.3.2.6 z

```
float Ragot::Camera_Transform::z
```

The documentation for this struct was generated from the following file:

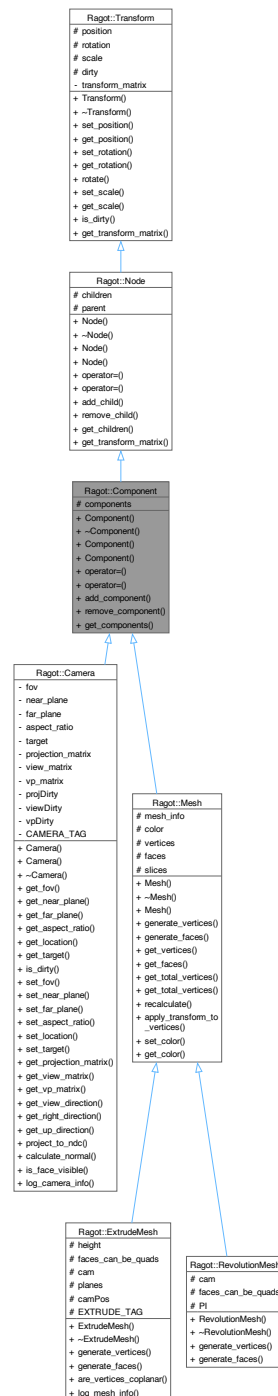
- [main/CommonTypes.hpp](#)

6.4 Ragot::Component Class Reference

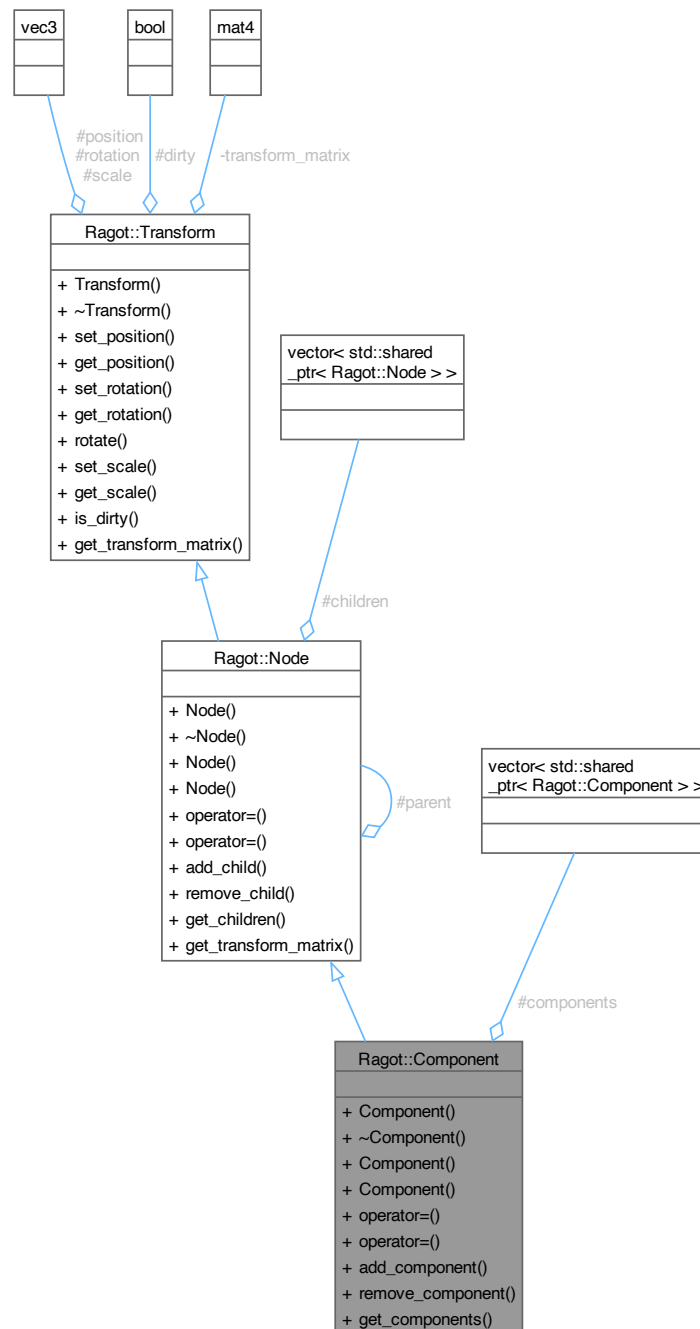
Base class for components in the [Ragot](#) engine.

```
#include <Components.hpp>
```

Inheritance diagram for Ragot::Component:



Collaboration diagram for Ragot::Component:



Public Member Functions

- `Component()`=default
Default constructor for the `Component` class.
- `virtual ~Component()`=default
Default virtual destructor for the `Component` class.
- `Component (const Component &)=delete`

- Deleted copy constructor for the [Component](#) class.*

 - [Component](#) (const [Component](#) &&)=delete
- Deleted move constructor for the [Component](#) class.*

 - [Component](#) & [operator=](#) (const [Component](#) &)=delete
- Deleted assignment operator for the [Component](#) class.*

 - [Component](#) & [operator=](#) (const [Component](#) &&)=delete
- Deleted move assignment operator for the [Component](#) class.*

 - void [add_component](#) (std::shared_ptr< [Component](#) > component)

Adds a component to the collection.
- void [remove_component](#) (std::shared_ptr< [Component](#) > component)

Removes a component from the collection.
- const std::vector< std::shared_ptr< [Component](#) > > [get_components](#) () const

Gets the collection of components.

Public Member Functions inherited from [Ragot::Node](#)

- [Node](#) ()=default
- Default constructor for [Node](#). Initializes an empty node with no parent and no children.*
- virtual [~Node](#) ()=default
- Default destructor for [Node](#). Cleans up the node and its children.*
- [Node](#) (const [Node](#) &)=delete
- Deleted copy constructor for [Node](#). Prevents copying of [Node](#) instances.*
- [Node](#) (const [Node](#) &&)=delete
- Deleted move constructor for [Node](#). Prevents moving of [Node](#) instances.*
- [Node](#) & [operator=](#) (const [Node](#) &)=delete
- Deleted assignment operator for [Node](#). Prevents assignment of [Node](#) instances.*
- [Node](#) & [operator=](#) (const [Node](#) &&)=delete
- Deleted move assignment operator for [Node](#). Prevents moving of [Node](#) instances.*
- void [add_child](#) (std::shared_ptr< [Node](#) > child)
- Get the parent node.*
- void [remove_child](#) (std::shared_ptr< [Node](#) > child)
- Remove a child node.*
- const std::vector< std::shared_ptr< [Node](#) > > & [get_children](#) () const
- Get the parent node.*
- mat4 [get_transform_matrix](#) () override
- Get the transform matrix object.*

Public Member Functions inherited from [Ragot::Transform](#)

- [Transform](#) ()
- Default constructor for the [Transform](#) class.*
- virtual [~Transform](#) ()=default
- Virtual destructor for the [Transform](#) class.*
- void [set_position](#) (const vec3 &pos)
- Sets the position of the object.*
- vec3 [get_position](#) () const
- Gets the current position of the object.*
- void [set_rotation](#) (const vec3 &rot)
- Moves the object by a specified vector.*

- `vec3 get_rotation () const`
Gets the current rotation of the object.
- `void rotate (const float angle, const vec3 &axis)`
Rotates the object by a specified angle around a given axis.
- `void set_scale (const vec3 &scale)`
Sets the scale of the object.
- `vec3 get_scale () const`
Sets the scale of the object uniformly.
- `bool is_dirty () const`
Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- `std::vector< std::shared_ptr< Component > > components`
Collection of components managed by this Component instance.

Protected Attributes inherited from `Ragot::Node`

- `std::vector< std::shared_ptr< Node > > children`
List of child nodes.
- `Node * parent = nullptr`
Pointer to the parent node.

Protected Attributes inherited from `Ragot::Transform`

- `vec3 position`
The position of the object in 3D space.
- `vec3 rotation`
The rotation of the object in degrees around each axis (x, y, z).
- `vec3 scale`
The scale of the object in 3D space, default is (1, 1, 1).
- `bool dirty = true`
Flag indicating whether the transformation matrix needs to be recalculated.

6.4.1 Detailed Description

Base class for components in the `Ragot` engine.

The `Component` class serves as a base class for all components in the `Ragot` engine. It allows for the management of a collection of components, providing methods to add and remove components, and access the list of components.

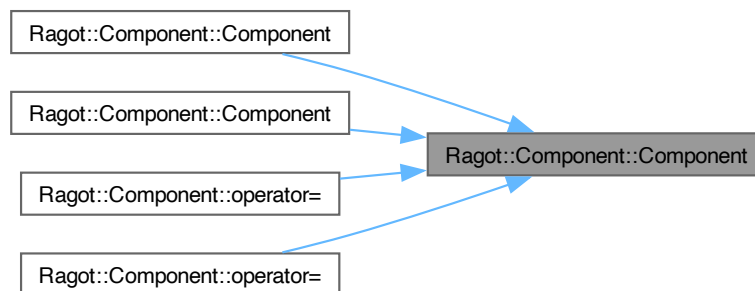
6.4.2 Constructor & Destructor Documentation

6.4.2.1 Component() [1/3]

```
Ragot::Component::Component () [default]
```

Default constructor for the [Component](#) class.

Initializes an empty component with no parent. Here is the caller graph for this function:



6.4.2.2 ~Component()

```
virtual Ragot::Component::~~Component () [virtual], [default]
```

Default virtual destructor for the [Component](#) class.

Cleans up the component and its resources.

6.4.2.3 Component() [2/3]

```
Ragot::Component::Component (  
    const Component & ) [delete]
```

Deleted copy constructor for the [Component](#) class.

Here is the call graph for this function:



6.4.2.4 Component() [3/3]

```
Ragot::Component::Component (
    const Component && ) [delete]
```

Deleted move constructor for the [Component](#) class.

Prevents moving a [Component](#) instance. Here is the call graph for this function:



6.4.3 Member Function Documentation

6.4.3.1 add_component()

```
void Ragot::Component::add_component (
    std::shared_ptr< Component > component) [inline]
```

Adds a component to the collection.

This method adds a shared pointer to a component to the collection and sets its parent to this [Component](#) instance.

Parameters

<i>component</i>	Shared pointer to the component to be added.
------------------	--

6.4.3.2 get_components()

```
const std::vector< std::shared_ptr< Component > > Ragot::Component::get_components () const
[inline]
```

Gets the collection of components.

This method returns a constant reference to the vector of components managed by this [Component](#) instance.

Returns

const std::vector<std::shared_ptr<Component>>& Reference to the vector of components.

6.4.3.3 operator=() [1/2]

```
Component & Ragot::Component::operator= (  
    const Component && ) [delete]
```

Deleted move assignment operator for the [Component](#) class.

Prevents moving a [Component](#) instance. Here is the call graph for this function:



6.4.3.4 operator=() [2/2]

```
Component & Ragot::Component::operator= (  
    const Component & ) [delete]
```

Deleted assignment operator for the [Component](#) class.

Prevents assignment of a [Component](#) instance. Here is the call graph for this function:



6.4.3.5 remove_component()

```
void Ragot::Component::remove_component (  
    std::shared_ptr< Component > component) [inline]
```

Removes a component from the collection.

This method removes a shared pointer to a component from the collection and sets its parent to nullptr.

Parameters

<i>component</i>	Shared pointer to the component to be removed.
------------------	--

6.4.4 Member Data Documentation

6.4.4.1 components

```
std::vector< std::shared_ptr < Component > > Ragot::Component::components [protected]
```

Collection of components managed by this [Component](#) instance.

This vector holds shared pointers to the components that are part of this [Component](#) instance.

The documentation for this class was generated from the following file:

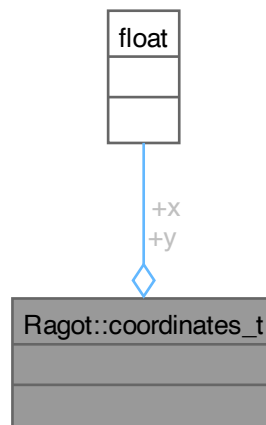
- main/[Components.hpp](#)

6.5 Ragot::coordinates_t Struct Reference

Represents 2D coordinates.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::coordinates_t:



Public Attributes

- float [x](#)
- float [y](#)

6.5.1 Detailed Description

Represents 2D coordinates.

This structure holds the x and y coordinates of a point in 2D space, typically used for mesh vertices.

6.5.2 Member Data Documentation

6.5.2.1 x

```
float Ragot::coordinates_t::x
```

6.5.2.2 y

```
float Ragot::coordinates_t::y
```

The documentation for this struct was generated from the following file:

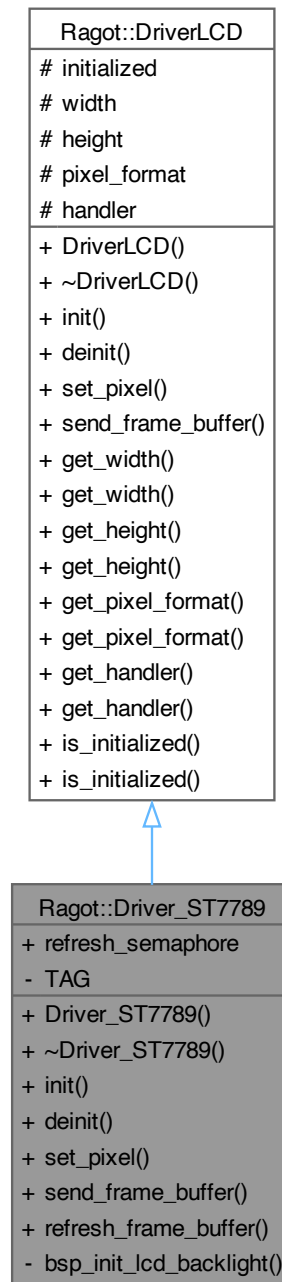
- [main/CommonTypes.hpp](#)

6.6 Ragot::Driver_ST7789 Class Reference

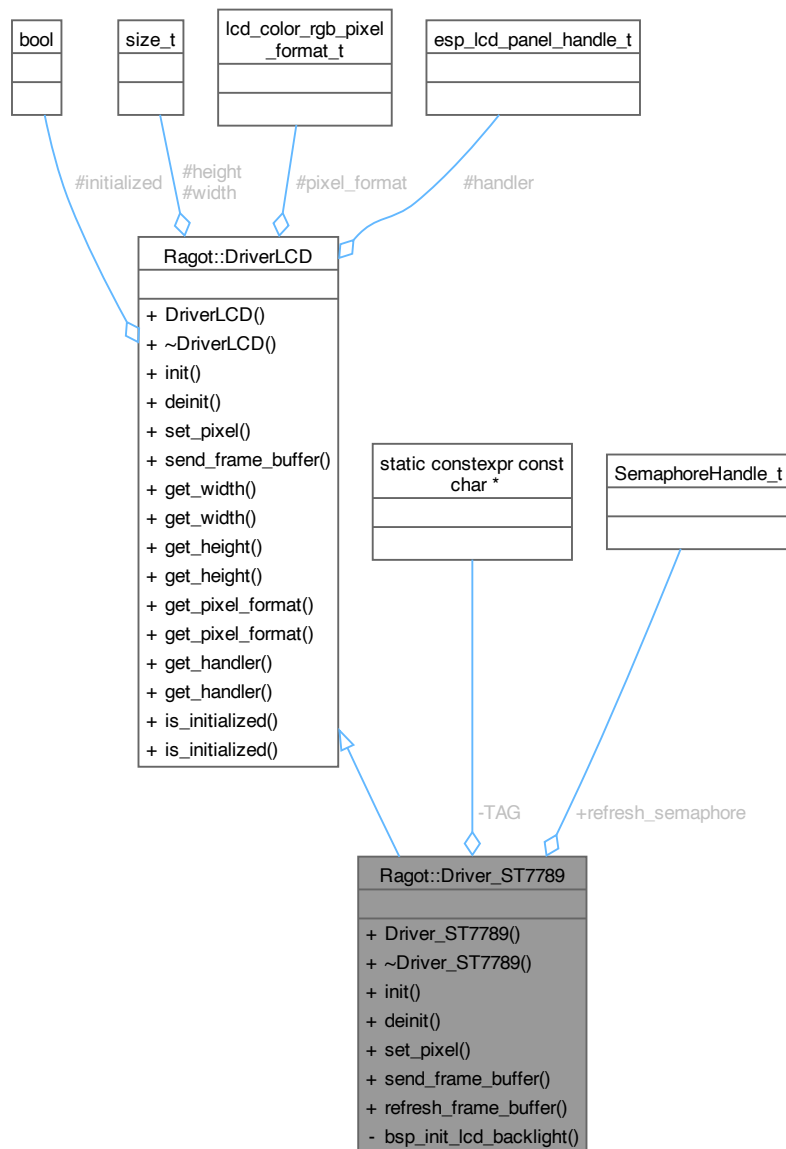
Driver for the ST7789 LCD panel.

```
#include <Driver_ST7789.hpp>
```

Inheritance diagram for Ragot::Driver_ST7789:



Collaboration diagram for Ragot::Driver_ST7789:



Public Member Functions

- [Driver_ST7789](#) ()
Default constructor for the [Driver_ST7789](#) class.
- virtual [~Driver_ST7789](#) ()=default
Default virtual destructor for the [Driver_ST7789](#) class.
- `esp_err_t` [init](#) (`gpio_num_t` reset_pin, `gpio_num_t` bk_pin) override
Initializes the ST7789 LCD panel driver.
- `esp_err_t` [deinit](#) () override
Deinitializes the ST7789 LCD panel driver.
- `esp_err_t` [set_pixel](#) (`uint32_t` x, `uint32_t` y, `uint32_t` color) override

- Sets a pixel at the specified coordinates to the given color.*
- esp_err_t [send_frame_buffer](#) (const void *frame_buffer) override
Sends a frame buffer to the ST7789 LCD panel.
- IRAM_ATTR bool [refresh_frame_buffer](#) (void *user_ctx)
Refreshes the frame buffer for the ST7789 LCD panel.

Public Member Functions inherited from [Ragot::DriverLCD](#)

- [DriverLCD](#) ()=default
Default constructor for the [DriverLCD](#) class.
- virtual [~DriverLCD](#) ()=default
Default virtual destructor for the [DriverLCD](#) class.
- const size_t [get_width](#) () const
Gets the width of the LCD panel.
- size_t [get_width](#) ()
Gets the height of the LCD panel.
- const size_t [get_height](#) () const
Gets the height of the LCD panel.
- size_t [get_height](#) ()
Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t [get_pixel_format](#) () const
Get the pixel format object.
- lcd_color_rgb_pixel_format_t [get_pixel_format](#) ()
Get the pixel format object.
- const esp_lcd_panel_handle_t [get_handler](#) () const
Get the handler object.
- esp_lcd_panel_handle_t [get_handler](#) ()
Get the handler object.
- const bool [is_initialized](#) () const
Checks if the LCD driver is initialized.
- bool [is_initialized](#) ()
Checks if the LCD driver is initialized.

Public Attributes

- SemaphoreHandle_t [refresh_semaphore](#)
Refresh semaphore for synchronizing frame buffer updates.

Private Member Functions

- void [bsp_init_lcd_backlight](#) (gpio_num_t bk_pin)
Initializes the LCD backlight GPIO pin.

Static Private Attributes

- static constexpr const char * [TAG](#) = "[Driver_ST7789]..."
Tag for logging messages related to the ST7789 driver.

Additional Inherited Members

Protected Attributes inherited from [Ragot::DriverLCD](#)

- bool [initialized](#) = false
Flag indicating if the LCD driver is initialized.
- size_t [width](#)
Width of the LCD panel in pixels.
- size_t [height](#)
Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t [pixel_format](#)
Pixel format of the LCD panel.
- esp_lcd_panel_handle_t [handler](#)
Handle to the LCD panel.

6.6.1 Detailed Description

Driver for the ST7789 LCD panel.

This class provides methods to initialize, deinitialize, and send frame buffers to the ST7789 LCD panel. It inherits from the [DriverLCD](#) class and implements the necessary methods for LCD operations.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Driver_ST7789()

```
Ragot::Driver_ST7789::Driver_ST7789 ()
```

Default constructor for the [Driver_ST7789](#) class.

Initializes the LCD driver with default values. Here is the call graph for this function:



6.6.2.2 ~Driver_ST7789()

```
virtual Ragot::Driver_ST7789::~~Driver_ST7789 () [virtual], [default]
```

Default virtual destructor for the [Driver_ST7789](#) class.

Cleans up resources used by the LCD driver.

6.6.3 Member Function Documentation

6.6.3.1 bsp_init_lcd_backlight()

```
void Ragot::Driver_ST7789::bsp_init_lcd_backlight (
    gpio_num_t bk_pin) [private]
```

Initializes the LCD backlight GPIO pin.

This method sets up the specified GPIO pin for controlling the LCD backlight.

Parameters

<i>bk_pin</i>	GPIO pin for backlight control.
---------------	---------------------------------

6.6.3.2 deinit()

```
esp_err_t Ragot::Driver_ST7789::deinit () [override], [virtual]
```

Deinitializes the ST7789 LCD panel driver.

This method cleans up resources used by the driver, including deleting the panel handler and freeing the refresh semaphore.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

6.6.3.3 init()

```
esp_err_t Ragot::Driver_ST7789::init (
    gpio_num_t reset_pin,
    gpio_num_t bk_pin) [override], [virtual]
```

Initializes the ST7789 LCD panel driver.

This method sets up the LCD panel with the specified reset and backlight GPIO pins. It configures the panel's pixel format, width, height, and other parameters.

Parameters

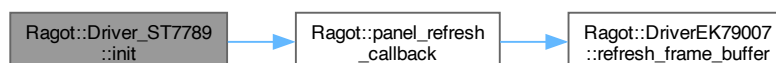
<i>reset_pin</i>	GPIO pin for panel reset.
<i>bk_pin</i>	GPIO pin for backlight control.

Returns

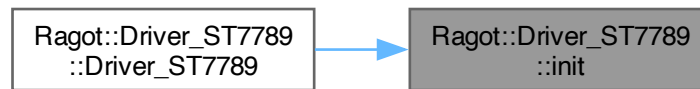
esp_err_t ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.4 refresh_frame_buffer()

```
IRAM_ATTR bool Ragot::Driver_ST7789::refresh_frame_buffer (
    void * user_ctx)
```

Refreshes the frame buffer for the ST7789 LCD panel.

Note

This method is called from the ISR context and should be used to signal that the frame buffer has been updated.

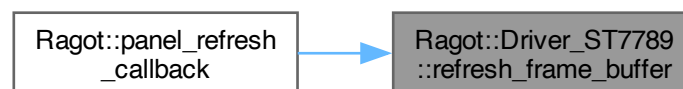
Parameters

<code>user_ctx</code>	
-----------------------	--

Returns

bool True if the frame buffer was refreshed successfully, false otherwise.

Here is the caller graph for this function:



6.6.3.5 send_frame_buffer()

```
esp_err_t Ragot::Driver_ST7789::send_frame_buffer (
    const void * frame_buffer) [override], [virtual]
```

Sends a frame buffer to the ST7789 LCD panel.

This method sends the provided frame buffer to the panel for display. It waits for the refresh semaphore to be available before sending the frame buffer.

Parameters

<i>frame_buffer</i>	Pointer to the frame buffer data.
---------------------	-----------------------------------

Returns

`esp_err_t` ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

6.6.3.6 set_pixel()

```
esp_err_t Ragot::Driver_ST7789::set_pixel (
    uint32_t x,
    uint32_t y,
    uint32_t color) [inline], [override], [virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method is not implemented for the ST7789 driver and will return ESP_FAIL.

Parameters

<i>x</i>	X coordinate of the pixel.
<i>y</i>	Y coordinate of the pixel.
<i>color</i>	Color value for the pixel.

Returns

`esp_err_t` ESP_FAIL as this method is not implemented.

Implements [Ragot::DriverLCD](#).

6.6.4 Member Data Documentation**6.6.4.1 refresh_semaphore**

```
SemaphoreHandle_t Ragot::Driver_ST7789::refresh_semaphore
```

Refresh semaphore for synchronizing frame buffer updates.

6.6.4.2 TAG

```
const char* Ragot::Driver_ST7789::TAG = "[Driver_ST7789]..." [static], [constexpr], [private]
```

Tag for logging messages related to the ST7789 driver.

The documentation for this class was generated from the following files:

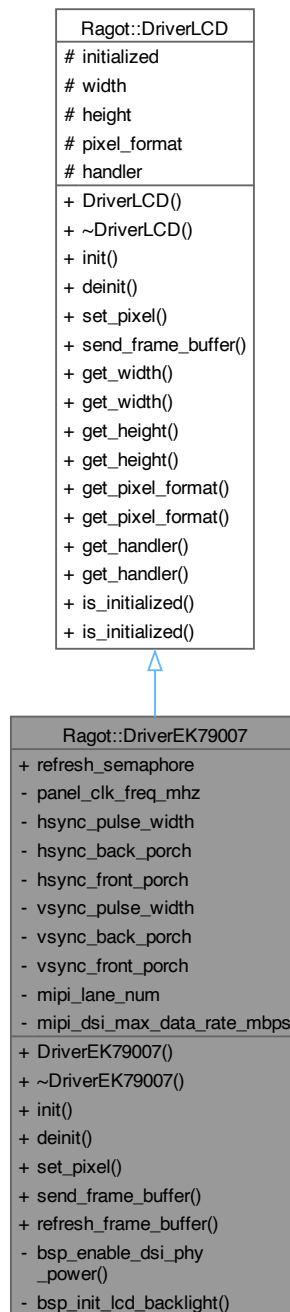
- main/[Driver_ST7789.hpp](#)
- main/[Driver_ST7789.cpp](#)

6.7 Ragot::DriverEK79007 Class Reference

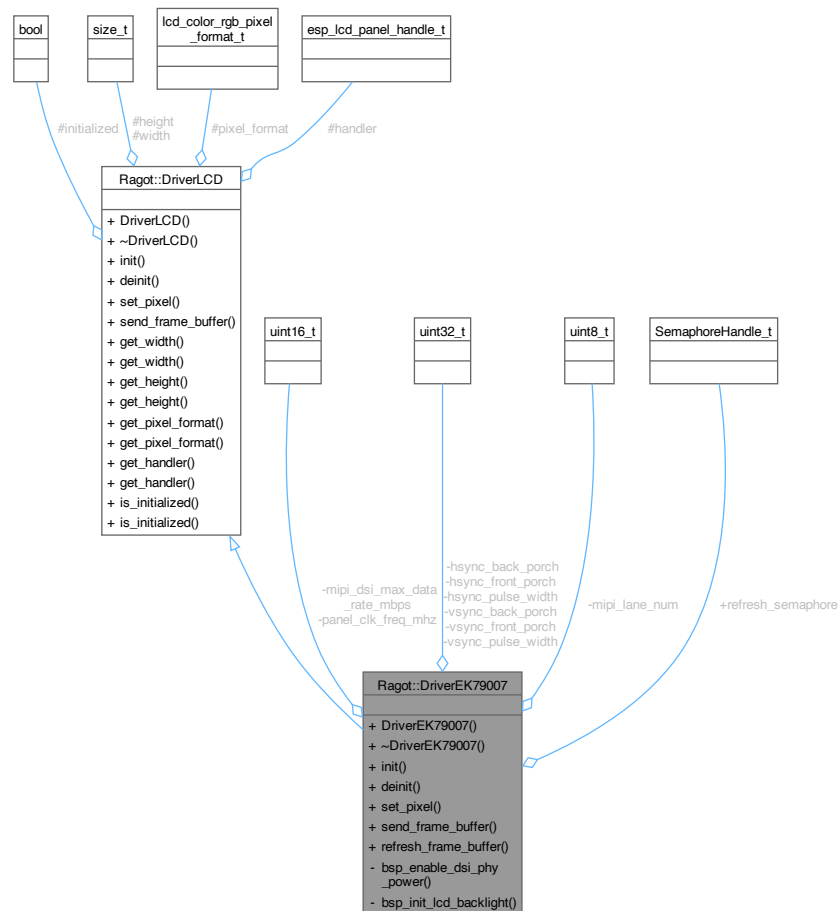
Driver for the EK79007 LCD panel.

```
#include <driver_ek79007.hpp>
```

Inheritance diagram for Ragot::DriverEK79007:



Collaboration diagram for Ragot::DriverEK79007:



Public Member Functions

- [DriverEK79007](#) ()
Default constructor for the [DriverEK79007](#) class.
- [~DriverEK79007](#) () override
Destructor for the [DriverEK79007](#) class.
- [esp_err_t init](#) (gpio_num_t reset_pin, gpio_num_t bk_pin) override
Initializes the EK79007 LCD panel driver.
- [esp_err_t deinit](#) () override
Deinitializes the EK79007 LCD panel driver.
- [esp_err_t set_pixel](#) (uint32_t x, uint32_t y, uint32_t color) override
Sets a pixel at the specified coordinates to the given color.
- [esp_err_t send_frame_buffer](#) (const void *frame_buffer) override
Sends a frame buffer to the EK79007 LCD panel.
- [IRAM_ATTR bool refresh_frame_buffer](#) (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data_t *edata, void *user_ctx)
Refreshes the frame buffer for the EK79007 LCD panel.

Public Member Functions inherited from Ragot::DriverLCD

- [DriverLCD](#) ()=default
Default constructor for the [DriverLCD](#) class.
- virtual [~DriverLCD](#) ()=default
Default virtual destructor for the [DriverLCD](#) class.
- const size_t [get_width](#) () const
Gets the width of the LCD panel.
- size_t [get_width](#) ()
Gets the height of the LCD panel.
- const size_t [get_height](#) () const
Gets the height of the LCD panel.
- size_t [get_height](#) ()
Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t [get_pixel_format](#) () const
Get the pixel format object.
- lcd_color_rgb_pixel_format_t [get_pixel_format](#) ()
Get the pixel format object.
- const esp_lcd_panel_handle_t [get_handler](#) () const
Get the handler object.
- esp_lcd_panel_handle_t [get_handler](#) ()
Get the handler object.
- const bool [is_initialized](#) () const
Checks if the LCD driver is initialized.
- bool [is_initialized](#) ()
Checks if the LCD driver is initialized.

Public Attributes

- SemaphoreHandle_t [refresh_semaphore](#)
Refresh semaphore for synchronizing frame buffer updates.

Private Member Functions

- void [bsp_enable_dsi_phy_power](#) ()
Enables the MIPI DSI PHY power.
- void [bsp_init_lcd_backlight](#) (gpio_num_t bk_pin)
Initializes the LCD backlight.

Private Attributes

- uint16_t [panel_clk_freq_mhz](#)
Horizontal pixel clock frequency in MHz.
- uint32_t [hsync_pulse_width](#)
Horizontal sync width, in pixel clock.
- uint32_t [hsync_back_porch](#)
Horizontal back porch, number of pixel clock between hsync and start of line active data.
- uint32_t [hsync_front_porch](#)
Horizontal front porch, number of pixel clock between the end of active data and the next hsync.

- uint32_t [vsync_pulse_width](#)
Vertical sync width, in number of lines.
- uint32_t [vsync_back_porch](#)
Vertical back porch, number of invalid lines between vsync and start of frame.
- uint32_t [vsync_front_porch](#)
Vertical front porch, number of invalid lines between the end of frame and the next vsync.
- uint8_t [mipi_lane_num](#)
Number of MIPI DSI lanes used for the panel.
- uint16_t [mipi_dsi_max_data_rate_mbps](#)
Maximum data rate of MIPI DSI in Mbps.

Additional Inherited Members

Protected Attributes inherited from [Ragot::DriverLCD](#)

- bool [initialized](#) = false
Flag indicating if the LCD driver is initialized.
- size_t [width](#)
Width of the LCD panel in pixels.
- size_t [height](#)
Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t [pixel_format](#)
Pixel format of the LCD panel.
- esp_lcd_panel_handle_t [handler](#)
Handle to the LCD panel.

6.7.1 Detailed Description

Driver for the EK79007 LCD panel.

This class provides methods to initialize, deinitialize, and send frame buffers to the EK79007 LCD panel. It inherits from the [DriverLCD](#) class and implements the necessary methods for LCD operations.

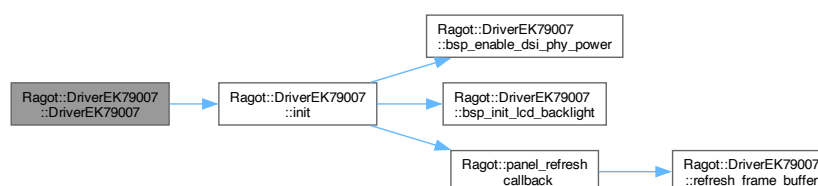
6.7.2 Constructor & Destructor Documentation

6.7.2.1 DriverEK79007()

```
Ragot::DriverEK79007::DriverEK79007 ()
```

Default constructor for the [DriverEK79007](#) class.

Here is the call graph for this function:

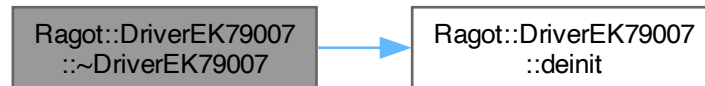


6.7.2.2 ~DriverEK79007()

```
Ragot::DriverEK79007::~~DriverEK79007 () [override]
```

Destructor for the [DriverEK79007](#) class.

Cleans up resources used by the driver. Here is the call graph for this function:



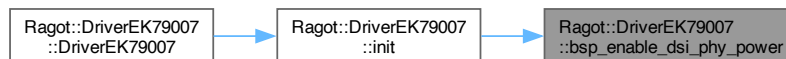
6.7.3 Member Function Documentation

6.7.3.1 bsp_enable_dsi_phy_power()

```
void Ragot::DriverEK79007::bsp_enable_dsi_phy_power () [private]
```

Enables the MIPI DSI PHY power.

This method powers on the MIPI DSI PHY by acquiring the appropriate LDO channel. Here is the caller graph for this function:



6.7.3.2 bsp_init_lcd_backlight()

```
void Ragot::DriverEK79007::bsp_init_lcd_backlight (
    gpio_num_t bk_pin) [private]
```

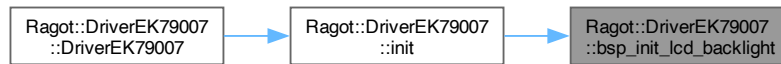
Initializes the LCD backlight.

This method configures the GPIO pin for the backlight and sets it to high to turn on the backlight.

Parameters

<i>bk_pin</i>	GPIO pin number for the backlight control.
---------------	--

Here is the caller graph for this function:



6.7.3.3 deinit()

```
esp_err_t Ragot::DriverEK79007::deinit () [override], [virtual]
```

Deinitializes the EK79007 LCD panel driver.

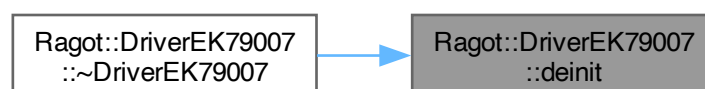
This method cleans up resources used by the driver, including deleting the panel handler and freeing the refresh semaphore.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

Here is the caller graph for this function:



6.7.3.4 init()

```
esp_err_t Ragot::DriverEK79007::init (
    gpio_num_t reset_pin,
    gpio_num_t bk_pin) [override], [virtual]
```

Initializes the EK79007 LCD panel driver.

This method sets up the panel with the specified reset and backlight GPIO pins, configures the panel parameters, and registers the refresh callback.

Parameters

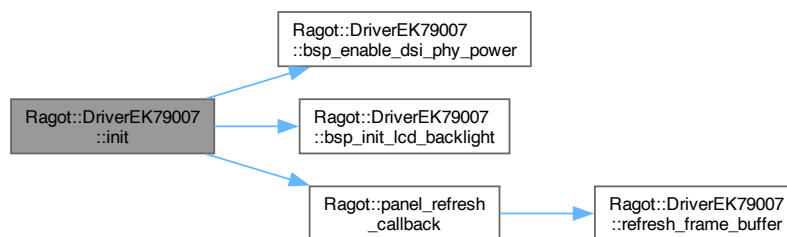
<i>reset_pin</i>	GPIO pin for panel reset.
<i>bk_pin</i>	GPIO pin for backlight control.

Returns

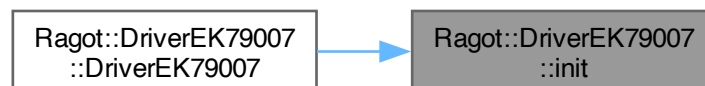
esp_err_t ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.3.5 refresh_frame_buffer()

```

IRAM_ATTR bool Ragot::DriverEK79007::refresh_frame_buffer (
    esp_lcd_panel_handle_t panel,
    esp_lcd_dpi_panel_event_data_t * edata,
    void * user_ctx)
  
```

Refreshes the frame buffer for the EK79007 LCD panel.

This method is called when the panel refresh is done. It handles the actual drawing of the frame buffer to the panel and releases the refresh semaphore.

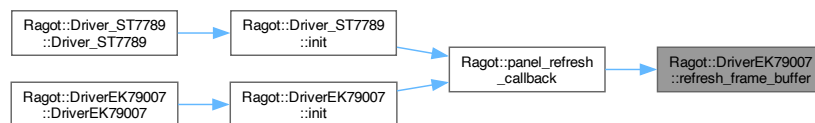
Parameters

<i>panel</i>	Pointer to the LCD panel handle.
<i>edata</i>	Pointer to the event data for the DPI panel.
<i>user_ctx</i>	User context pointer, which is this driver instance.

Returns

true if successful, false otherwise.

Here is the caller graph for this function:



6.7.3.6 send_frame_buffer()

```
esp_err_t Ragot::DriverEK79007::send_frame_buffer (
    const void * frame_buffer) [override], [virtual]
```

Sends a frame buffer to the EK79007 LCD panel.

This method sends the provided frame buffer to the panel for display. It waits for the refresh semaphore to be available before sending the frame buffer.

Parameters

<i>frame_buffer</i>	Pointer to the frame buffer data.
---------------------	-----------------------------------

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implements [Ragot::DriverLCD](#).

6.7.3.7 set_pixel()

```
esp_err_t Ragot::DriverEK79007::set_pixel (
    uint32_t x,
    uint32_t y,
    uint32_t color) [inline], [override], [virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method is not implemented for the EK79007 driver and will return ESP_FAIL.

Parameters

<i>x</i>	X coordinate of the pixel.
<i>y</i>	Y coordinate of the pixel.
<i>color</i>	Color value to set the pixel to.

Returns

esp_err_t ESP_FAIL as this method is not implemented.

Implements [Ragot::DriverLCD](#).

6.7.4 Member Data Documentation

6.7.4.1 hsync_back_porch

```
uint32_t Ragot::DriverEK79007::hsync_back_porch [private]
```

Horizontal back porch, number of pixel clock between hsync and start of line active data.

6.7.4.2 hsync_front_porch

```
uint32_t Ragot::DriverEK79007::hsync_front_porch [private]
```

Horizontal front porch, number of pixel clock between the end of active data and the next hsync.

6.7.4.3 hsync_pulse_width

```
uint32_t Ragot::DriverEK79007::hsync_pulse_width [private]
```

Horizontal sync width, in pixel clock.

6.7.4.4 mipi_dsi_max_data_rate_mbps

```
uint16_t Ragot::DriverEK79007::mipi_dsi_max_data_rate_mbps [private]
```

Maximum data rate of MIPI DSI in Mbps.

6.7.4.5 mipi_lane_num

```
uint8_t Ragot::DriverEK79007::mipi_lane_num [private]
```

Number of MIPI DSI lanes used for the panel.

6.7.4.6 panel_clk_freq_mhz

```
uint16_t Ragot::DriverEK79007::panel_clk_freq_mhz [private]
```

Horizontal pixel clock frequency in MHz.

6.7.4.7 refresh_semaphore

```
SemaphoreHandle_t Ragot::DriverEK79007::refresh_semaphore
```

Refresh semaphore for synchronizing frame buffer updates.

This semaphore is used to ensure that the frame buffer is updated only when it is safe to do so.

6.7.4.8 vsync_back_porch

```
uint32_t Ragot::DriverEK79007::vsync_back_porch [private]
```

Vertical back porch, number of invalid lines between vsync and start of frame.

6.7.4.9 vsync_front_porch

```
uint32_t Ragot::DriverEK79007::vsync_front_porch [private]
```

Vertical front porch, number of invalid lines between the end of frame and the next vsync.

6.7.4.10 vsync_pulse_width

```
uint32_t Ragot::DriverEK79007::vsync_pulse_width [private]
```

Vertical sync width, in number of lines.

The documentation for this class was generated from the following files:

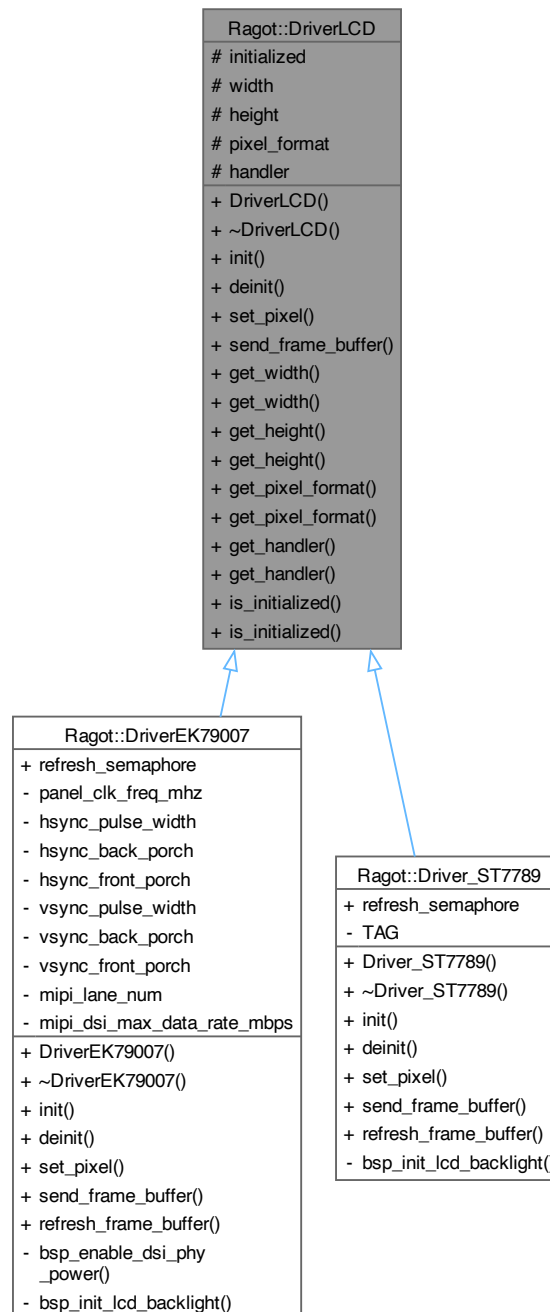
- [main/driver_ek79007.hpp](#)
- [main/driver_ek79007.cpp](#)

6.8 Ragot::DriverLCD Class Reference

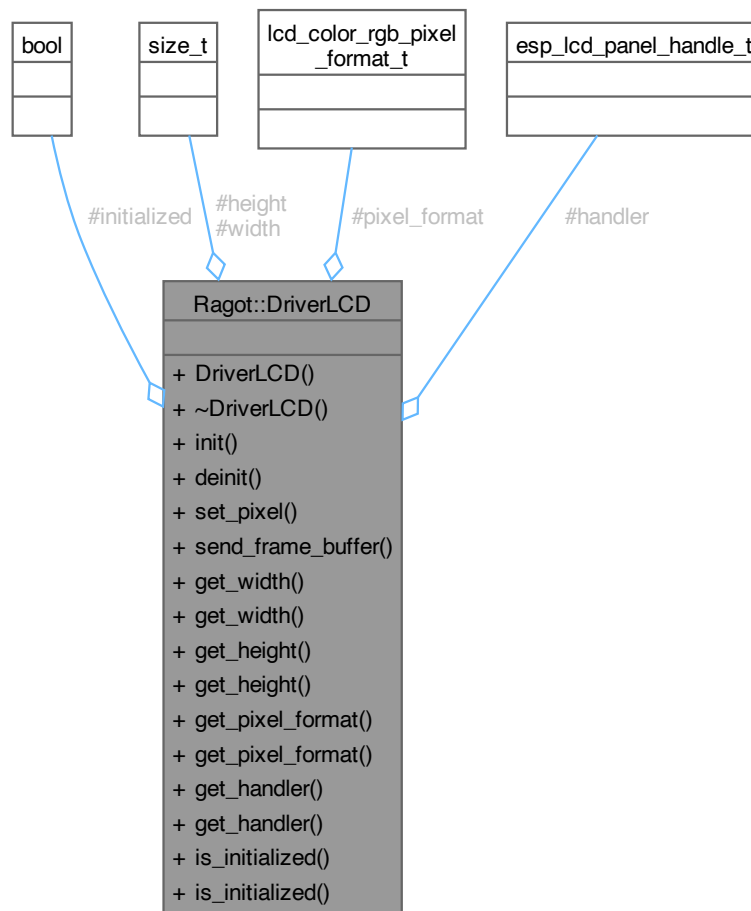
Base class for LCD drivers.

```
#include <driver_lcd.hpp>
```

Inheritance diagram for Ragot::DriverLCD:



Collaboration diagram for Ragot::DriverLCD:



Public Member Functions

- `DriverLCD ()`=default
Default constructor for the `DriverLCD` class.
- virtual `~DriverLCD ()`=default
Default virtual destructor for the `DriverLCD` class.
- virtual `esp_err_t init (gpio_num_t reset_pin, gpio_num_t bk_pin)=0`
Initializes the LCD driver with the specified reset and backlight GPIO pins.
- virtual `esp_err_t deinit ()=0`
Deinitializes the LCD driver.
- virtual `esp_err_t set_pixel (uint32_t x, uint32_t y, uint32_t color)=0`
Sets a pixel at the specified coordinates to the given color.
- virtual `esp_err_t send_frame_buffer (const void *frame_buffer)=0`
Sends a frame buffer to the LCD panel.
- const `size_t get_width ()` const
Gets the width of the LCD panel.
- `size_t get_width ()`

- Gets the height of the LCD panel.*
- const size_t [get_height](#) () const
Gets the height of the LCD panel.
- size_t [get_height](#) ()
Gets the height of the LCD panel.
- const lcd_color_rgb_pixel_format_t [get_pixel_format](#) () const
Get the pixel format object.
- lcd_color_rgb_pixel_format_t [get_pixel_format](#) ()
Get the pixel format object.
- const esp_lcd_panel_handle_t [get_handler](#) () const
Get the handler object.
- esp_lcd_panel_handle_t [get_handler](#) ()
Get the handler object.
- const bool [is_initialized](#) () const
Checks if the LCD driver is initialized.
- bool [is_initialized](#) ()
Checks if the LCD driver is initialized.

Protected Attributes

- bool [initialized](#) = false
Flag indicating if the LCD driver is initialized.
- size_t [width](#)
Width of the LCD panel in pixels.
- size_t [height](#)
Height of the LCD panel in pixels.
- lcd_color_rgb_pixel_format_t [pixel_format](#)
Pixel format of the LCD panel.
- esp_lcd_panel_handle_t [handler](#)
Handle to the LCD panel.

6.8.1 Detailed Description

Base class for LCD drivers.

This class provides an interface for LCD drivers, including methods for initialization, deinitialization, setting pixels, and sending frame buffers. It also provides access to the LCD panel's width, height, pixel format, and handler.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DriverLCD()

```
Ragot::DriverLCD::DriverLCD () [default]
```

Default constructor for the [DriverLCD](#) class.

Initializes the LCD driver with default values.

6.8.2.2 ~DriverLCD()

```
virtual Ragot::DriverLCD::~DriverLCD () [virtual], [default]
```

Default virtual destructor for the [DriverLCD](#) class.

Cleans up resources used by the LCD driver.

6.8.3 Member Function Documentation

6.8.3.1 deinit()

```
virtual esp_err_t Ragot::DriverLCD::deinit () [pure virtual]
```

Deinitializes the LCD driver.

This method should be implemented by derived classes to clean up resources used by the LCD panel.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in [Ragot::Driver_ST7789](#), and [Ragot::DriverEK79007](#).

6.8.3.2 get_handler() [1/2]

```
esp_lcd_panel_handle_t Ragot::DriverLCD::get_handler () [inline]
```

Get the handler object.

Returns

esp_lcd_panel_handle_t

6.8.3.3 get_handler() [2/2]

```
const esp_lcd_panel_handle_t Ragot::DriverLCD::get_handler () const [inline]
```

Get the handler object.

Returns

const esp_lcd_panel_handle_t

6.8.3.4 get_height() [1/2]

```
size_t Ragot::DriverLCD::get_height () [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.5 get_height() [2/2]

```
const size_t Ragot::DriverLCD::get_height () const [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.6 get_pixel_format() [1/2]

```
lcd_color_rgb_pixel_format_t Ragot::DriverLCD::get_pixel_format () [inline]
```

Get the pixel format object.

Returns

lcd_color_rgb_pixel_format_t

6.8.3.7 get_pixel_format() [2/2]

```
const lcd_color_rgb_pixel_format_t Ragot::DriverLCD::get_pixel_format () const [inline]
```

Get the pixel format object.

Returns

const lcd_color_rgb_pixel_format_t

6.8.3.8 get_width() [1/2]

```
size_t Ragot::DriverLCD::get_width () [inline]
```

Gets the height of the LCD panel.

Returns

size_t Height of the LCD panel in pixels.

6.8.3.9 get_width() [2/2]

```
const size_t Ragot::DriverLCD::get_width () const [inline]
```

Gets the width of the LCD panel.

Returns

size_t Width of the LCD panel in pixels.

6.8.3.10 init()

```
virtual esp_err_t Ragot::DriverLCD::init (
    gpio_num_t reset_pin,
    gpio_num_t bk_pin) [pure virtual]
```

Initializes the LCD driver with the specified reset and backlight GPIO pins.

This method should be implemented by derived classes to set up the LCD panel.

Parameters

<i>reset_pin</i>	GPIO pin for panel reset.
<i>bk_pin</i>	GPIO pin for backlight control.

Returns

esp_err_t ESP_OK on success, or an error code on failure.

Implemented in [Ragot::Driver_ST7789](#), and [Ragot::DriverEK79007](#).

6.8.3.11 is_initialized() [1/2]

```
bool Ragot::DriverLCD::is_initialized () [inline]
```

Checks if the LCD driver is initialized.

Returns

true
false

6.8.3.12 is_initialized() [2/2]

```
const bool Ragot::DriverLCD::is_initialized () const [inline]
```

Checks if the LCD driver is initialized.

Returns

true
false

6.8.3.13 send_frame_buffer()

```
virtual esp_err_t Ragot::DriverLCD::send_frame_buffer (
    const void * frame_buffer) [pure virtual]
```

Sends a frame buffer to the LCD panel.

This method should be implemented by derived classes to send the provided frame buffer to the panel for display.

Parameters

<i>frame_buffer</i>	Pointer to the frame buffer data.
---------------------	-----------------------------------

Returns

`esp_err_t` ESP_OK on success, or an error code on failure.

Implemented in [Ragot::Driver_ST7789](#), and [Ragot::DriverEK79007](#).

6.8.3.14 set_pixel()

```
virtual esp_err_t Ragot::DriverLCD::set_pixel (
    uint32_t x,
    uint32_t y,
    uint32_t color) [pure virtual]
```

Sets a pixel at the specified coordinates to the given color.

This method should be implemented by derived classes to set a pixel on the LCD panel.

Parameters

<i>x</i>	X coordinate of the pixel.
<i>y</i>	Y coordinate of the pixel.
<i>color</i>	Color value for the pixel.

Returns

`esp_err_t` ESP_OK on success, or an error code on failure.

Implemented in [Ragot::Driver_ST7789](#), and [Ragot::DriverEK79007](#).

6.8.4 Member Data Documentation**6.8.4.1 handler**

```
esp_lcd_panel_handle_t Ragot::DriverLCD::handler [protected]
```

Handle to the LCD panel.

6.8.4.2 height

```
size_t Ragot::DriverLCD::height [protected]
```

Height of the LCD panel in pixels.

6.8.4.3 initialized

```
bool Ragot::DriverLCD::initialized = false [protected]
```

Flag indicating if the LCD driver is initialized.

6.8.4.4 pixel_format

```
lcd_color_rgb_pixel_format_t Ragot::DriverLCD::pixel_format [protected]
```

Pixel format of the LCD panel.

6.8.4.5 width

```
size_t Ragot::DriverLCD::width [protected]
```

Width of the LCD panel in pixels.

The documentation for this class was generated from the following file:

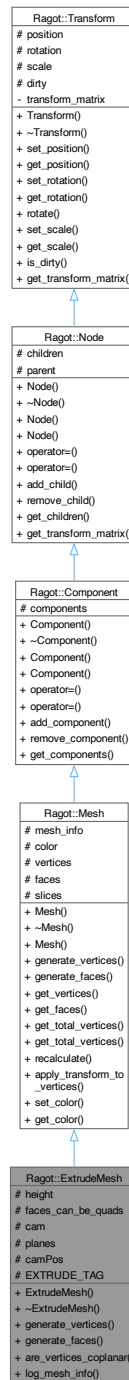
- [main/driver_lcd.hpp](#)

6.9 Ragot::ExtrudeMesh Class Reference

Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the [Mesh](#) class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include <ExtrudeMesh.hpp>
```

Inheritance diagram for Ragot::ExtrudeMesh:



Public Member Functions inherited from Ragot::Mesh

- [Mesh](#) ()=delete
Construct a new [Mesh](#) object (deleted constructor).
- virtual [~Mesh](#) ()=default
Default virtual destructor for the [Mesh](#) class.
- [Mesh](#) (mesh_info_t &mesh_info)
Construct a new [Mesh](#) object with mesh information.
- const std::vector< glm::fvec4 > & [get_vertices](#) () const
Get the vertices object.
- const std::vector< [face_t](#) > & [get_faces](#) () const
Get the faces object.
- const size_t [get_total_vertices](#) () const
Get the total vertices object.
- size_t [get_total_vertices](#) ()
Get the total vertices object.
- void [recalculate](#) ()
Recalculate the mesh vertices and faces.
- void [apply_transform_to_vertices](#) ()
Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the [Transform](#) class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- void [set_color](#) (uint16_t new_color)
Set the color of the mesh.
- uint16_t [get_color](#) () const
Get the color of the mesh.

Public Member Functions inherited from Ragot::Component

- [Component](#) ()=default
Default constructor for the [Component](#) class.
- virtual [~Component](#) ()=default
Default virtual destructor for the [Component](#) class.
- [Component](#) (const [Component](#) &)=delete
Deleted copy constructor for the [Component](#) class.
- [Component](#) (const [Component](#) &&)=delete
Deleted move constructor for the [Component](#) class.
- [Component](#) & [operator=](#) (const [Component](#) &)=delete
Deleted assignment operator for the [Component](#) class.
- [Component](#) & [operator=](#) (const [Component](#) &&)=delete
Deleted move assignment operator for the [Component](#) class.
- void [add_component](#) (std::shared_ptr< [Component](#) > component)
Adds a component to the collection.
- void [remove_component](#) (std::shared_ptr< [Component](#) > component)
Removes a component from the collection.
- const std::vector< std::shared_ptr< [Component](#) > > [get_components](#) () const
Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- [Node](#) ()=default
Default constructor for [Node](#). Initializes an empty node with no parent and no children.
- virtual [~Node](#) ()=default
Default destructor for [Node](#). Cleans up the node and its children.
- [Node](#) (const [Node](#) &)=delete
Deleted copy constructor for [Node](#). Prevents copying of [Node](#) instances.
- [Node](#) (const [Node](#) &&)=delete
Deleted move constructor for [Node](#). Prevents moving of [Node](#) instances.
- [Node](#) & operator= (const [Node](#) &)=delete
Deleted assignment operator for [Node](#). Prevents assignment of [Node](#) instances.
- [Node](#) & operator= (const [Node](#) &&)=delete
Deleted move assignment operator for [Node](#). Prevents moving of [Node](#) instances.
- void [add_child](#) (std::shared_ptr< [Node](#) > child)
Get the parent node.
- void [remove_child](#) (std::shared_ptr< [Node](#) > child)
Remove a child node.
- const std::vector< std::shared_ptr< [Node](#) > > & [get_children](#) () const
Get the parent node.
- mat4 [get_transform_matrix](#) () override
Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- [Transform](#) ()
Default constructor for the [Transform](#) class.
- virtual [~Transform](#) ()=default
Virtual destructor for the [Transform](#) class.
- void [set_position](#) (const vec3 &pos)
Sets the position of the object.
- vec3 [get_position](#) () const
Gets the current position of the object.
- void [set_rotation](#) (const vec3 &rot)
Moves the object by a specified vector.
- vec3 [get_rotation](#) () const
Gets the current rotation of the object.
- void [rotate](#) (const float angle, const vec3 &axis)
Rotates the object by a specified angle around a given axis.
- void [set_scale](#) (const vec3 &scale)
Sets the scale of the object.
- vec3 [get_scale](#) () const
Sets the scale of the object uniformly.
- bool [is_dirty](#) () const
Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- float `height` = 1.0f
Height of the extrusion.
- bool `faces_can_be_quads` = false
Flag indicating whether the faces can be quads.
- const `Camera` & `cam`
`Camera` reference for culling faces based on the camera's view direction.
- glm::vec4 `planes` [4]
Array of planes used for culling faces.
- glm::vec3 `camPos`
Position of the camera in world space.

Protected Attributes inherited from `Ragot::Mesh`

- `mesh_info_t` `mesh_info`
Information about the mesh, including coordinates and rendering type.
- uint16_t `color` = 0xFFFF
Color of the mesh, default is white (0xFFFF).
- std::vector< glm::fvec4 > `vertices`
Vector of vertices representing the mesh in 3D space.
- std::vector< `face_t` > `faces`
Vector of faces representing the mesh, each face can be a triangle or a quad.
- int `slices` = 16
Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from `Ragot::Component`

- std::vector< std::shared_ptr< `Component` > > `components`
Collection of components managed by this `Component` instance.

Protected Attributes inherited from `Ragot::Node`

- std::vector< std::shared_ptr< `Node` > > `children`
List of child nodes.
- `Node` * `parent` = nullptr
Pointer to the parent node.

Protected Attributes inherited from `Ragot::Transform`

- vec3 `position`
The position of the object in 3D space.
- vec3 `rotation`
The rotation of the object in degrees around each axis (x, y, z).
- vec3 `scale`
The scale of the object in 3D space, default is (1, 1, 1).
- bool `dirty` = true
Flag indicating whether the transformation matrix needs to be recalculated.

Static Protected Attributes

- static const char * [EXTRUDE_TAG](#) = "ExtrudeMesh"
Tag for logging messages related to the [ExtrudeMesh](#) class.

6.9.1 Detailed Description

Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the [Mesh](#) class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

The [ExtrudeMesh](#) class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ExtrudeMesh()

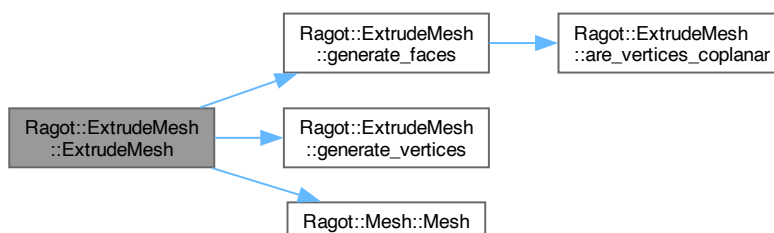
```
Ragot::ExtrudeMesh::ExtrudeMesh (
    mesh_info_t & mesh_info,
    const Camera & cam) [inline]
```

Constructs an [ExtrudeMesh](#) object with the specified mesh information and camera reference.

Parameters

<i>mesh_info</i>	Information about the mesh to be extruded.
<i>cam</i>	Reference to the camera used for culling faces.

Here is the call graph for this function:



6.9.2.2 ~ExtrudeMesh()

```
Ragot::ExtrudeMesh::~~ExtrudeMesh () [default]
```

Default destructor for the [ExtrudeMesh](#) class.

This destructor is used to clean up resources when the [ExtrudeMesh](#) object is destroyed.

6.9.3 Member Function Documentation

6.9.3.1 are_vertices_coplanar()

```
bool Ragot::ExtrudeMesh::are_vertices_coplanar (
    const glm::fvec4 & v1,
    const glm::fvec4 & v2,
    const glm::fvec4 & v3,
    const glm::fvec4 & v4,
    float tolerance = 0.1f)
```

Verifies if four vertices are coplanar.

This method checks if the four vertices v1, v2, v3, and v4 are coplanar within a specified tolerance. It uses the scalar triple product to determine coplanarity.

Parameters

<i>v1</i>	First vertex in homogeneous coordinates.
<i>v2</i>	Second vertex in homogeneous coordinates.
<i>v3</i>	Third vertex in homogeneous coordinates.
<i>v4</i>	Fourth vertex in homogeneous coordinates.
<i>tolerance</i>	Tolerance value for coplanarity check (default is 0.1).

Returns

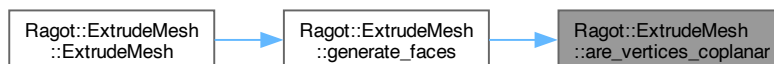
true if the vertices are coplanar, false otherwise.

The method calculates the scalar triple product of the vectors formed by the vertices and checks if it is close to zero within the specified tolerance. The scalar triple product is computed as the dot product of the first vector with the cross product of the other two vectors. This method is useful for determining if a set of vertices can form a valid face in the mesh.

Note

This method assumes that the vertices are provided in homogeneous coordinates (4D vectors).

Here is the caller graph for this function:



6.9.3.2 generate_faces()

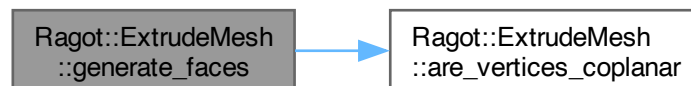
```
void Ragot::ExtrudeMesh::generate_faces () [override], [virtual]
```

Generates the faces of the extruded mesh.

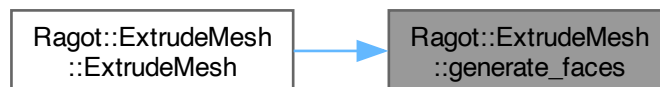
This method creates the faces of the mesh based on the vertices generated by the `generate_vertices` method. It checks if the vertices are coplanar and creates either quads or triangles accordingly.

Implements [Ragot::Mesh](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.3.3 generate_vertices()

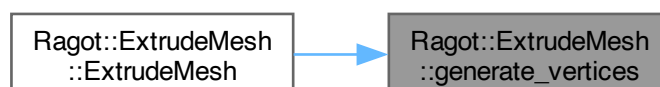
```
void Ragot::ExtrudeMesh::generate_vertices () [override], [virtual]
```

Deleted copy constructor for the [ExtrudeMesh](#) class.

This constructor is deleted to prevent copying of [ExtrudeMesh](#) objects.

Implements [Ragot::Mesh](#).

Here is the caller graph for this function:

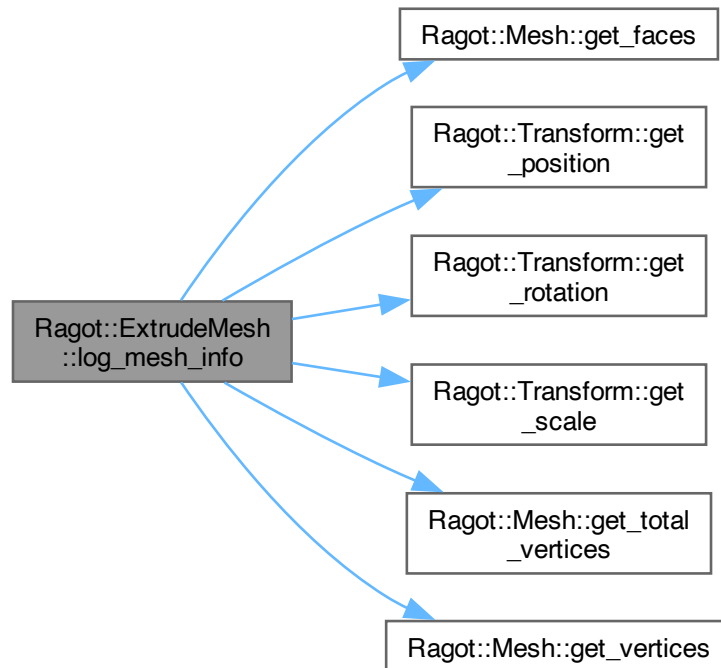


6.9.3.4 log_mesh_info()

```
void Ragot::ExtrudeMesh::log_mesh_info () const
```

Logs detailed information about the mesh.

Here is the call graph for this function:



6.9.4 Member Data Documentation

6.9.4.1 cam

```
const Camera& Ragot::ExtrudeMesh::cam [protected]
```

[Camera](#) reference for culling faces based on the camera's view direction.

This reference is used to determine which faces of the mesh are visible from the camera's perspective.

6.9.4.2 camPos

```
glm::vec3 Ragot::ExtrudeMesh::camPos [protected]
```

Position of the camera in world space.

This vector represents the position of the camera in the 3D world. It is used to calculate the visibility of faces based on the camera's position.

6.9.4.3 EXTRUDE_TAG

```
const char * Ragot::ExtrudeMesh::EXTRUDE_TAG = "ExtrudeMesh" [static], [protected]
```

Tag for logging messages related to the [ExtrudeMesh](#) class.

6.9.4.4 faces_can_be_quads

```
bool Ragot::ExtrudeMesh::faces_can_be_quads = false [protected]
```

Flag indicating whether the faces can be quads.

This flag is set to true if the number of vertices is a multiple of 8 or if it is exactly 4. It determines how faces are generated in the mesh.

6.9.4.5 height

```
float Ragot::ExtrudeMesh::height = 1.0f [protected]
```

Height of the extrusion.

This value determines how far the 2D shape is extruded in the Z direction.

6.9.4.6 planes

```
glm::vec4 Ragot::ExtrudeMesh::planes[4] [protected]
```

Array of planes used for culling faces.

This array contains the planes that define the view frustum of the camera. It is used to determine which faces are visible and which can be culled.

The documentation for this class was generated from the following files:

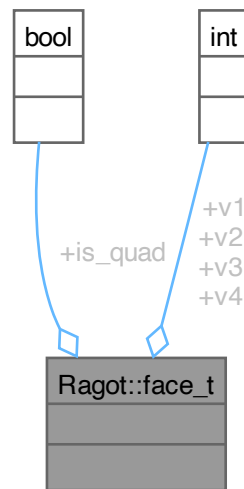
- main/[ExtrudeMesh.hpp](#)
- main/[ExtrudeMesh.cpp](#)

6.10 Ragot::face_t Struct Reference

Represents a face in a 3D mesh.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::face_t:



Public Attributes

- bool `is_quad`
- int `v1`
- int `v2`
- int `v3`
- int `v4`

6.10.1 Detailed Description

Represents a face in a 3D mesh.

This structure can represent either a triangle or a quadrilateral face, depending on the `is_quad` flag.

6.10.2 Member Data Documentation

6.10.2.1 `is_quad`

```
bool Ragot::face_t::is_quad
```

6.10.2.2 v1

```
int Ragot::face_t::v1
```

6.10.2.3 v2

```
int Ragot::face_t::v2
```

6.10.2.4 v3

```
int Ragot::face_t::v3
```

6.10.2.5 v4

```
int Ragot::face_t::v4
```

The documentation for this struct was generated from the following file:

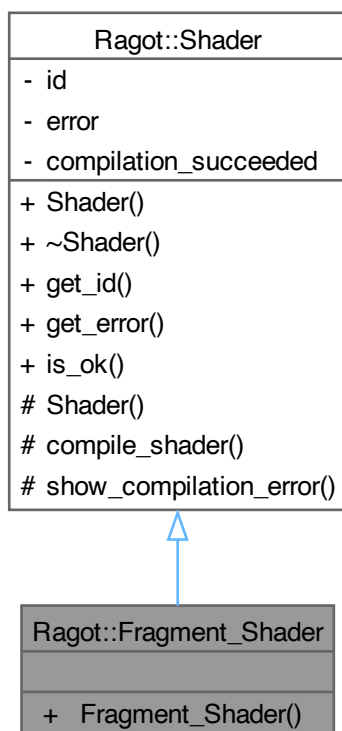
- main/[CommonTypes.hpp](#)

6.11 Ragot::Fragment_Shader Class Reference

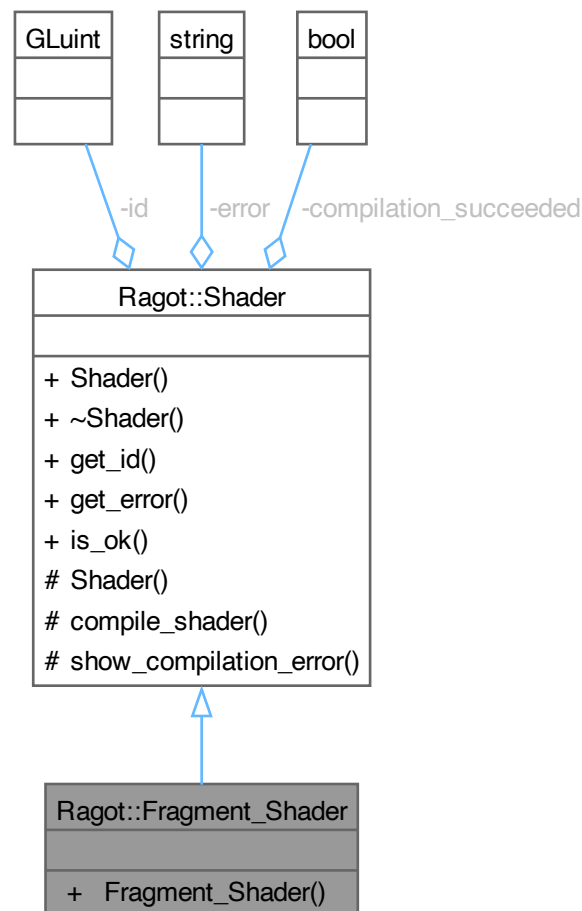
Class for managing an OpenGL fragment shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Fragment_Shader:



Collaboration diagram for Ragot::Fragment_Shader:



Public Member Functions

- [Fragment_Shader](#) (const vector< string > &source_code)
Constructor for the [Fragment_Shader](#) class.

Public Member Functions inherited from [Ragot::Shader](#)

- [Shader](#) ()=delete
Deleted default constructor.
- [~Shader](#) ()
Destructor for the [Shader](#) class.
- GLuint [get_id](#) () const
Gets the shader ID.
- string * [get_error](#) ()
Gets the compilation error message.
- bool [is_ok](#) () const
Checks if the shader is compiled successfully.

Additional Inherited Members

Protected Member Functions inherited from Ragot::Shader

- [Shader](#) (const vector< string > &source_code, GLenum type)
Constructor for the [Shader](#) class.
- GLuint [compile_shader](#) ()
Compiles the shader.
- void [show_compilation_error](#) ()
Displays compilation errors.

6.11.1 Detailed Description

Class for managing an OpenGL fragment shader.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Fragment_Shader()

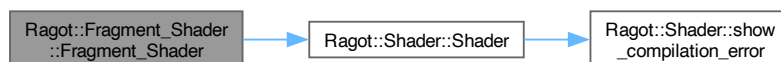
```
Ragot::Fragment_Shader::Fragment_Shader (
    const vector< string > & source_code) [inline]
```

Constructor for the [Fragment_Shader](#) class.

Parameters

<i>source_code</i>	Vector of fragment shader source code.
--------------------	--

Here is the call graph for this function:



The documentation for this class was generated from the following file:

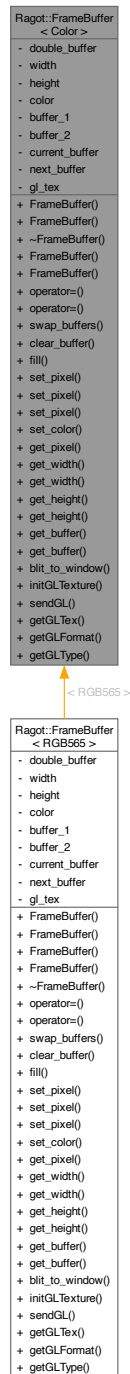
- main/[Shader_Program.hpp](#)

6.12 Ragot::FrameBuffer< Color > Class Template Reference

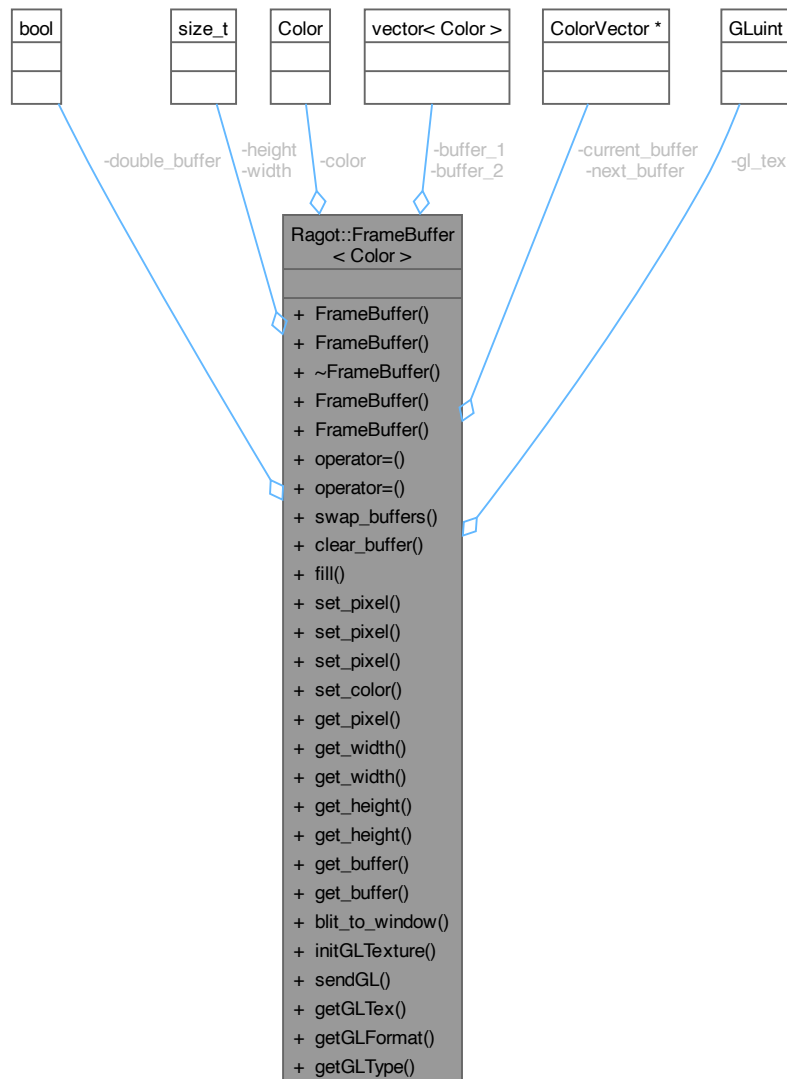
Class to manage a frame buffer for rendering graphics.

```
#include <FrameBuffer.hpp>
```

Inheritance diagram for Ragot::FrameBuffer< Color >:



Collaboration diagram for Ragot::FrameBuffer< Color >:



Public Types

- using `TYPE` = `Color`
- using `ColorVector` = `std::vector < Color >`

Public Member Functions

- `FrameBuffer` (`size_t width`, `size_t height`, `bool double_buffer`)
Constructor for the `FrameBuffer` class.
- `FrameBuffer` ()=delete
Default constructor for the `FrameBuffer` class (Deleted).
- `~FrameBuffer` ()=default

- Default destructor for the *FrameBuffer* class.
- *FrameBuffer* (const *FrameBuffer* &)=delete

Construct a new *Frame Buffer* object (Deleted).
- *FrameBuffer* (const *FrameBuffer* &&)=delete

Construct a new *Frame Buffer* object (Deleted).
- *FrameBuffer* & operator= (const *FrameBuffer* &)=delete

Assignment operator for the *FrameBuffer* class (Deleted).
- *FrameBuffer* & operator= (const *FrameBuffer* &&)=delete

Assignment operator for the *FrameBuffer* class (Deleted).
- void *swap_buffers* ()

Swaps the current buffer with the next buffer.
- void *clear_buffer* (*Buffer* buffer_to_clear=NEXT_BUFFER)

Clears the specified buffer by filling it with the default color.
- void *fill* (Color color=0, *Buffer* buffer_to_fill=NEXT_BUFFER)

Fills the specified buffer with the given color.
- void *set_pixel* (size_t x, size_t y, Color color)

Sets a pixel at the specified coordinates to the given color.
- void *set_pixel* (size_t offset, Color color)

Sets a pixel at the specified offset in the buffer to the given color.
- void *set_pixel* (size_t offset)

Sets a pixel at the specified offset in the buffer to the default color.
- void *set_color* (Color color)

Sets the default color for the frame buffer.
- Color *get_pixel* (size_t x, size_t y) const

Gets the color of a pixel at the specified coordinates.
- size_t *get_width* ()

Gets the color of a pixel at the specified offset in the buffer.
- size_t *get_width* () const

Gets the width of the frame buffer.
- size_t *get_height* ()

Gets the height of the frame buffer.
- size_t *get_height* () const

Gets the height of the frame buffer.
- const Color * *get_buffer* () const

Gets the current buffer being used.
- Color * *get_buffer* ()

Get the buffer object.
- void *blit_to_window* () const

Blits the current buffer to the window. This method copies the contents of the current buffer to the next buffer, effectively preparing the next frame for rendering.
- void *initGLTexture* ()

Initializes the OpenGL texture for the frame buffer.
- void *sendGL* () const

Clears the OpenGL texture associated with the frame buffer.
- GLuint *getGLTex* () const

Gets the OpenGL texture ID for the frame buffer.

Static Public Member Functions

- static GLenum *getGLFormat* ()

Sets the OpenGL texture for the frame buffer.
- static GLenum *getGLType* ()

Gets the OpenGL type for the frame buffer.

Private Attributes

- bool [double_buffer](#)
Flag to indicate if double buffering is enabled.
- size_t [width](#)
Width of the frame buffer in pixels.
- size_t [height](#)
Height of the frame buffer in pixels.
- Color [color](#)
Default color for filling the buffer.
- [ColorVector](#) [buffer_1](#)
First buffer for single or double buffering.
- [ColorVector](#) [buffer_2](#)
Second buffer for double buffering (if enabled)
- [ColorVector](#) * [current_buffer](#)
Pointer to the current buffer being used.
- [ColorVector](#) * [next_buffer](#)
Pointer to the next buffer to be used (for double buffering)
- GLuint [gl_tex](#) = 0
OpenGL texture ID for the frame buffer.

6.12.1 Detailed Description

template<typename Color>
class Ragot::FrameBuffer< Color >

Class to manage a frame buffer for rendering graphics.

This class provides methods to create a frame buffer, swap buffers, clear the buffer, fill it with a color, set and get pixels, and manage OpenGL textures. It supports both single and double buffering modes.

6.12.2 Member Typedef Documentation

6.12.2.1 ColorVector

```
template<typename Color>
using Ragot::FrameBuffer< Color >::ColorVector = std::vector < Color >
```

6.12.2.2 TYPE

```
template<typename Color>
using Ragot::FrameBuffer< Color >::TYPE = Color
```

6.12.3 Constructor & Destructor Documentation

6.12.3.1 `FrameBuffer()` [1/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer (
    size_t width,
    size_t height,
    bool double_buffer)
```

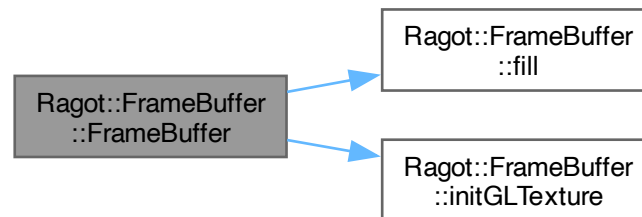
Constructor for the `FrameBuffer` class.

Initializes the frame buffer with the specified width, height, and double buffering option. Allocates memory for the buffers and fills them with the default color.

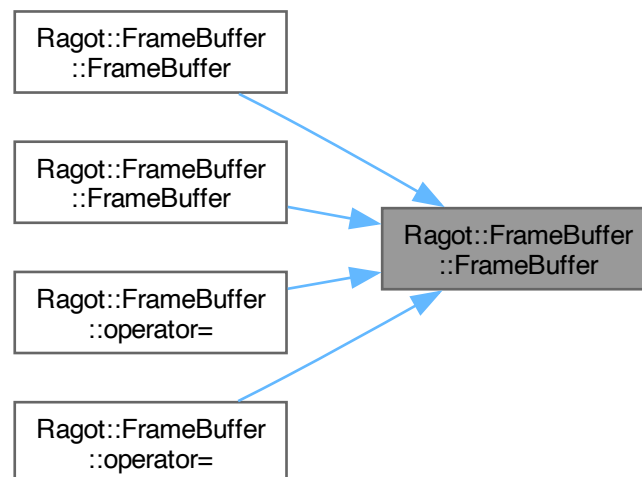
Parameters

<i>width</i>	Width of the frame buffer in pixels.
<i>height</i>	Height of the frame buffer in pixels.
<i>double_buffer</i>	Flag to indicate if double buffering is enabled.

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.3.2 FrameBuffer() [2/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::FrameBuffer () [delete]
```

Default constructor for the `FrameBuffer` class (Deleted).

6.12.3.3 ~FrameBuffer()

```
template<typename Color>
Ragot::FrameBuffer< Color >::~~FrameBuffer () [default]
```

Default destructor for the [FrameBuffer](#) class.

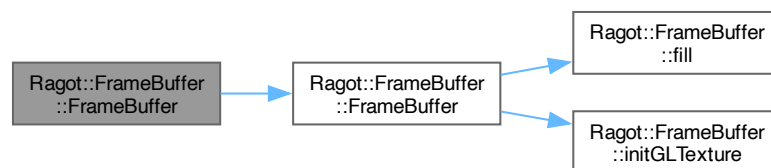
Cleans up resources used by the frame buffer.

6.12.3.4 FrameBuffer() [3/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::~FrameBuffer (
    const FrameBuffer< Color > & ) [delete]
```

Construct a new [Frame Buffer](#) object (Deleted).

Here is the call graph for this function:

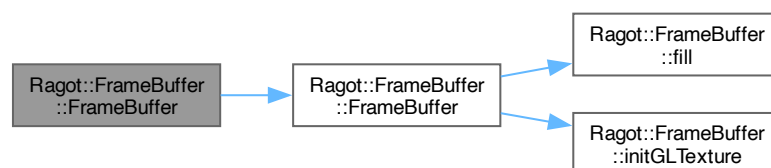


6.12.3.5 FrameBuffer() [4/4]

```
template<typename Color>
Ragot::FrameBuffer< Color >::~FrameBuffer (
    const FrameBuffer< Color > && ) [delete]
```

Construct a new [Frame Buffer](#) object (Deleted).

Here is the call graph for this function:



6.12.4 Member Function Documentation

6.12.4.1 blit_to_window()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::blit_to_window () const [inline]
```

Blits the current buffer to the window. This method copies the contents of the current buffer to the next buffer, effectively preparing the next frame for rendering.

6.12.4.2 clear_buffer()

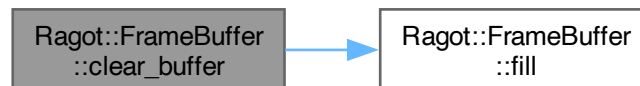
```
template<typename Color>
void Ragot::FrameBuffer< Color >::clear_buffer (
    Buffer buffer_to_clear = NEXT_BUFFER)
```

Clears the specified buffer by filling it with the default color.

Parameters

<i>buffer_to_clear</i>	The buffer to clear (CURRENT_BUFFER, NEXT_BUFFER, or MAX_BUFFER).
------------------------	---

Here is the call graph for this function:



6.12.4.3 fill()

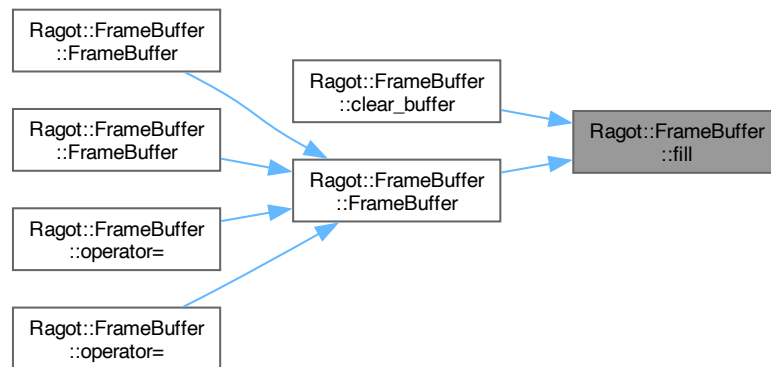
```
template<typename Color>
void Ragot::FrameBuffer< Color >::fill (
    Color color = 0,
    Buffer buffer_to_fill = NEXT_BUFFER)
```

Fills the specified buffer with the given color.

Parameters

<i>color</i>	The color to fill the buffer with (default is 0).
<i>buffer_to_fill</i>	The buffer to fill (CURRENT_BUFFER, NEXT_BUFFER, or MAX_BUFFER).

Here is the caller graph for this function:



6.12.4.4 `get_buffer()` [1/2]

```
template<typename Color>
Color * Ragot::FrameBuffer< Color >::get_buffer () [inline]
```

Get the buffer object.

Returns

Color*

6.12.4.5 `get_buffer()` [2/2]

```
template<typename Color>
const Color * Ragot::FrameBuffer< Color >::get_buffer () const [inline]
```

Gets the current buffer being used.

Returns

const Color* Pointer to the current buffer data.

6.12.4.6 `get_height()` [1/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_height () [inline]
```

Gets the height of the frame buffer.

Returns

size_t The height of the frame buffer in pixels.

6.12.4.7 get_height() [2/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_height () const [inline]
```

Gets the height of the frame buffer.

Returns

size_t The height of the frame buffer in pixels.

6.12.4.8 get_pixel()

```
template<typename Color>
Color Ragot::FrameBuffer< Color >::get_pixel (
    size_t x,
    size_t y) const
```

Gets the color of a pixel at the specified coordinates.

Parameters

<i>x</i>	X coordinate of the pixel.
<i>y</i>	Y coordinate of the pixel.

Returns

Color The color of the pixel at the specified coordinates.

6.12.4.9 get_width() [1/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_width () [inline]
```

Gets the color of a pixel at the specified offset in the buffer.

Parameters

<i>offset</i>	Offset in the buffer (calculated as y * width + x).
---------------	---

Returns

Color The color of the pixel at the specified offset.

6.12.4.10 get_width() [2/2]

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::get_width () const [inline]
```

Gets the width of the frame buffer.

Returns

size_t The width of the frame buffer in pixels.

6.12.4.11 getGLFormat()

```
template<typename Color>
GLenum Ragot::FrameBuffer< Color >::getGLFormat () [static]
```

Sets the OpenGL texture for the frame buffer.

This method binds the OpenGL texture to the current context.

6.12.4.12 getGLTex()

```
template<typename Color>
GLuint Ragot::FrameBuffer< Color >::getGLTex () const [inline]
```

Gets the OpenGL texture ID for the frame buffer.

Returns

GLuint The OpenGL texture ID.

6.12.4.13 getGLType()

```
template<typename Color>
GLenum Ragot::FrameBuffer< Color >::getGLType () [static]
```

Gets the OpenGL type for the frame buffer.

This method returns the OpenGL type corresponding to the color format used in the frame buffer.

Returns

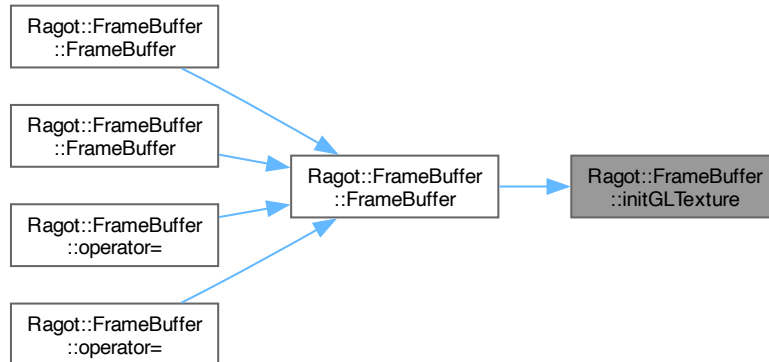
GLenum The OpenGL type for the frame buffer.

6.12.4.14 initGLTexture()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::initGLTexture ()
```

Initializes the OpenGL texture for the frame buffer.

This method creates an OpenGL texture and binds it to the frame buffer. Here is the caller graph for this function:



6.12.4.15 operator=() [1/2]

```
template<typename Color>
FrameBuffer & Ragot::FrameBuffer< Color >::operator= (
    const FrameBuffer< Color > && ) [delete]
```

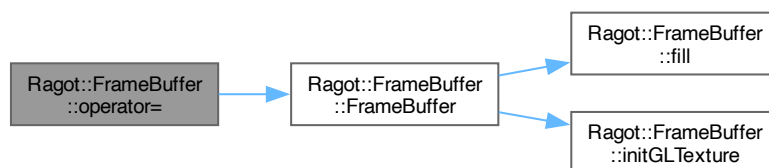
Assignment operator for the `FrameBuffer` class (Deleted).

Prevents assignment of `FrameBuffer` objects.

Returns

`FrameBuffer&` Reference to the current object.

Here is the call graph for this function:



6.12.4.16 operator=() [2/2]

```
template<typename Color>
Ragot::FrameBuffer< Color >::operator= (
    const FrameBuffer< Color > & ) [delete]
```

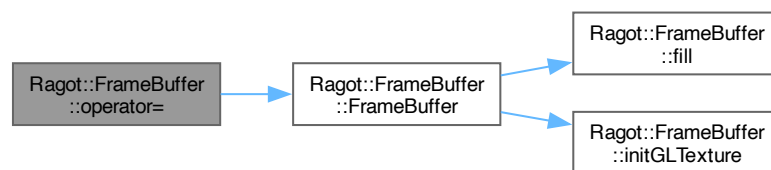
Assignment operator for the [FrameBuffer](#) class (Deleted).

Prevents assignment of [FrameBuffer](#) objects.

Returns

[FrameBuffer](#)& Reference to the current object.

Here is the call graph for this function:



6.12.4.17 sendGL()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::sendGL () const
```

Clears the OpenGL texture associated with the frame buffer.

This method deletes the OpenGL texture to free up resources.

6.12.4.18 set_color()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_color (
    Color color)
```

Sets the default color for the frame buffer.

Parameters

<i>color</i>	The color to set as the default.
--------------	----------------------------------

6.12.4.19 set_pixel() [1/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
    size_t offset)
```

Sets a pixel at the specified offset in the buffer to the default color.

Parameters

<i>offset</i>	Offset in the buffer (calculated as $y * \text{width} + x$).
---------------	---

6.12.4.20 set_pixel() [2/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
    size_t offset,
    Color color)
```

Sets a pixel at the specified offset in the buffer to the given color.

Parameters

<i>offset</i>	Offset in the buffer (calculated as $y * \text{width} + x$).
<i>color</i>	Color value for the pixel.

6.12.4.21 set_pixel() [3/3]

```
template<typename Color>
void Ragot::FrameBuffer< Color >::set_pixel (
    size_t x,
    size_t y,
    Color color)
```

Sets a pixel at the specified coordinates to the given color.

Parameters

<i>x</i>	X coordinate of the pixel.
<i>y</i>	Y coordinate of the pixel.
<i>color</i>	Color value for the pixel.

6.12.4.22 swap_buffers()

```
template<typename Color>
void Ragot::FrameBuffer< Color >::swap_buffers ()
```

Swaps the current buffer with the next buffer.

This method is used in double buffering to switch between the buffers for rendering.

6.12.5 Member Data Documentation**6.12.5.1 buffer_1**

```
template<typename Color>
ColorVector Ragot::FrameBuffer< Color >::buffer_1 [private]
```

First buffer for single or double buffering.

6.12.5.2 buffer_2

```
template<typename Color>
ColorVector Ragot::FrameBuffer< Color >::buffer_2 [private]
```

Second buffer for double buffering (if enabled)

6.12.5.3 color

```
template<typename Color>
Color Ragot::FrameBuffer< Color >::color [private]
```

Default color for filling the buffer.

6.12.5.4 current_buffer

```
template<typename Color>
ColorVector* Ragot::FrameBuffer< Color >::current_buffer [private]
```

Pointer to the current buffer being used.

6.12.5.5 double_buffer

```
template<typename Color>
bool Ragot::FrameBuffer< Color >::double_buffer [private]
```

Flag to indicate if double buffering is enabled.

6.12.5.6 gl_tex

```
template<typename Color>
GLuint Ragot::FrameBuffer< Color >::gl_tex = 0 [private]
```

OpenGL texture ID for the frame buffer.

This variable holds the OpenGL texture ID used for rendering the frame buffer.

6.12.5.7 height

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::height [private]
```

Height of the frame buffer in pixels.

6.12.5.8 next_buffer

```
template<typename Color>
ColorVector* Ragot::FrameBuffer< Color >::next_buffer [private]
```

Pointer to the next buffer to be used (for double buffering)

6.12.5.9 width

```
template<typename Color>
size_t Ragot::FrameBuffer< Color >::width [private]
```

Width of the frame buffer in pixels.

The documentation for this class was generated from the following files:

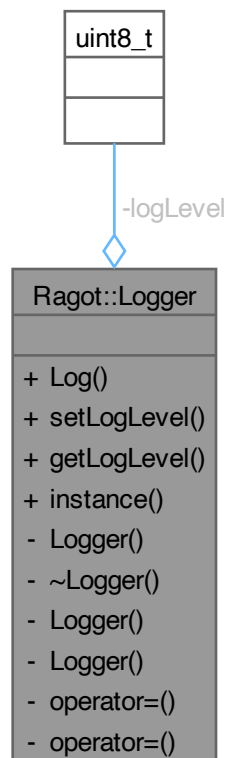
- main/[FrameBuffer.hpp](#)
- main/[FrameBuffer.cpp](#)

6.13 Ragot::Logger Class Reference

Singleton logger class for the [Ragot](#) engine.

```
#include <Logger.hpp>
```

Collaboration diagram for Ragot::Logger:



Public Member Functions

- `template<typename... Args>`
`void Log (const char *TAG, uint8_t level, const char *fmt, Args... args)`
Logs a message with the specified tag and level.
- `void setLogLevel (uint8_t level)`
Sets the log level for the logger.
- `uint8_t getLogLevel () const`
Gets the current log level of the logger.

Static Public Member Functions

- `static Logger & instance ()`
Gets the singleton instance of the `Logger` class.

Private Member Functions

- `Logger ()=default`
Default constructor for the `Logger` class.
- `~Logger ()=default`
Default destructor for the `Logger` class.
- `Logger (const Logger &)=delete`
Construct a new `Logger` object (Deleted).
- `Logger (const Logger &&)=delete`
Construct a new `Logger` object (Deleted).
- `Logger & operator= (const Logger &)=delete`
Assignment operator for the `Logger` class (Deleted).
- `Logger & operator= (const Logger &&)=delete`
Assignment operator for the `Logger` class (Deleted).

Private Attributes

- `uint8_t logLevel = 0`
Current log level for the logger.

6.13.1 Detailed Description

Singleton logger class for the `Ragot` engine.

This class provides a singleton logger that allows logging messages with different severity levels (INFO, WARNING, ERROR). It supports formatted logging using printf-style format strings and can be used across different platforms.

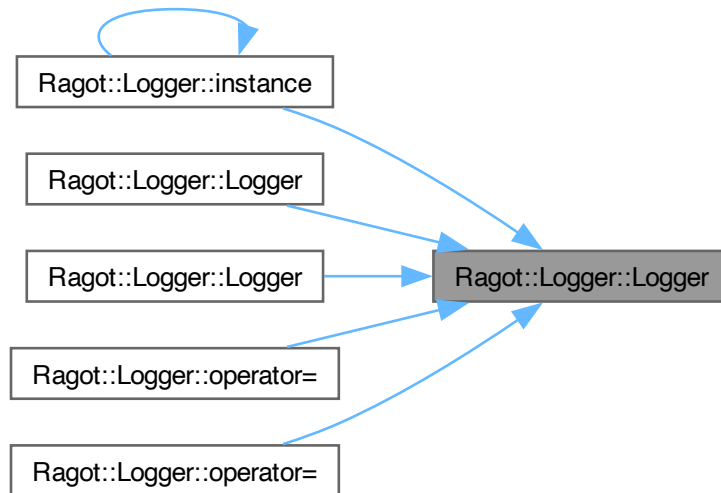
6.13.2 Constructor & Destructor Documentation

6.13.2.1 Logger() [1/3]

```
Ragot::Logger::Logger () [private], [default]
```

Default constructor for the [Logger](#) class.

This constructor is private to enforce the singleton pattern. Here is the caller graph for this function:



6.13.2.2 ~Logger()

```
Ragot::Logger::~~Logger () [private], [default]
```

Default destructor for the [Logger](#) class.

This destructor is defaulted and does not perform any special cleanup.

6.13.2.3 Logger() [2/3]

```
Ragot::Logger::Logger (
    const Logger & ) [private], [delete]
```

Construct a new [Logger](#) object (Deleted).

Here is the call graph for this function:



6.13.2.4 `Logger()` [3/3]

```
Ragot::Logger::Logger (
    const Logger && ) [private], [delete]
```

Construct a new [Logger](#) object (Deleted).

This constructor is deleted to prevent moving the [Logger](#) instance. Here is the call graph for this function:



6.13.3 Member Function Documentation

6.13.3.1 `getLogLevel()`

```
uint8_t Ragot::Logger::getLogLevel () const [inline]
```

Gets the current log level of the logger.

This method returns the current log level, which determines the severity of messages that will be logged.

Returns

`uint8_t` The current log level (0 = INFO, 1 = WARNING, 2 = ERROR).

6.13.3.2 `instance()`

```
static Logger & Ragot::Logger::instance () [inline], [static]
```

Gets the singleton instance of the [Logger](#) class.

This method returns a reference to the singleton [Logger](#) instance.

Returns

[Logger&](#) Reference to the singleton [Logger](#) instance.

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.3 Log()

```

template<typename... Args>
void Ragot::Logger::Log (
    const char * TAG,
    uint8_t level,
    const char * fmt,
    Args... args) [inline]
  
```

Logs a message with the specified tag and level.

This method logs a message with the given tag and severity level. The message can be formatted using printf-style format strings.

Parameters

<i>TAG</i>	The tag for the log message.
<i>level</i>	The severity level of the log message (0 = INFO, 1 = WARNING, 2 = ERROR).
<i>fmt</i>	The format string for the log message.
<i>args</i>	The arguments to format the log message.

6.13.3.4 operator=() [1/2]

```
Logger & Ragot::Logger::operator= (
    const Logger && ) [private], [delete]
```

Assignment operator for the [Logger](#) class (Deleted).

This operator is deleted to prevent moving [Logger](#) instances.

Returns

[Logger](#)& Reference to the current object.

Here is the call graph for this function:



6.13.3.5 operator=() [2/2]

```
Logger & Ragot::Logger::operator= (
    const Logger & ) [private], [delete]
```

Assignment operator for the [Logger](#) class (Deleted).

This operator is deleted to prevent assignment of [Logger](#) instances.

Returns

[Logger](#)& Reference to the current object.

Here is the call graph for this function:



6.13.3.6 setLogLevel()

```
void Ragot::Logger::setLogLevel (
    uint8_t level) [inline]
```

Sets the log level for the logger.

This method sets the log level for the logger, which determines the severity of messages that will be logged. It also configures the ESP-IDF logging system if running on an ESP platform.

Parameters

<i>level</i>	The new log level to set (0 = INFO, 1 = WARNING, 2 = ERROR).
--------------	--

6.13.4 Member Data Documentation

6.13.4.1 logLevel

```
uint8_t Ragot::Logger::logLevel = 0 [private]
```

Current log level for the logger.

This variable stores the current log level, which determines the severity of messages that will be logged. 0 = INFO, 1 = WARNING, 2 = ERROR.

The documentation for this class was generated from the following file:

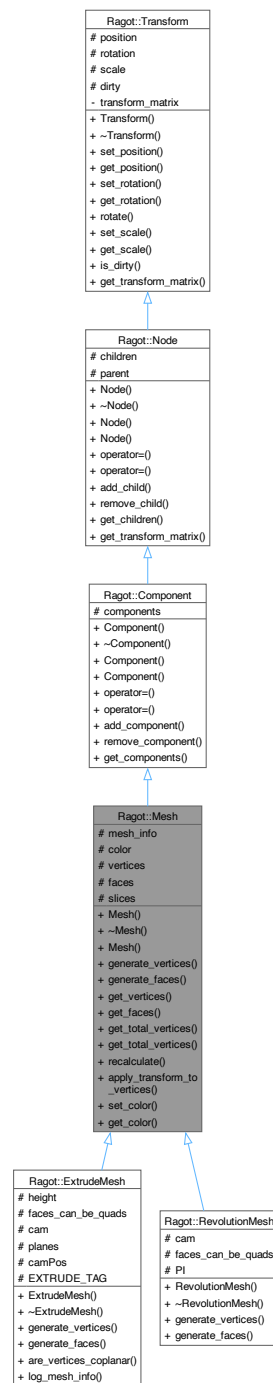
- main/[Logger.hpp](#)

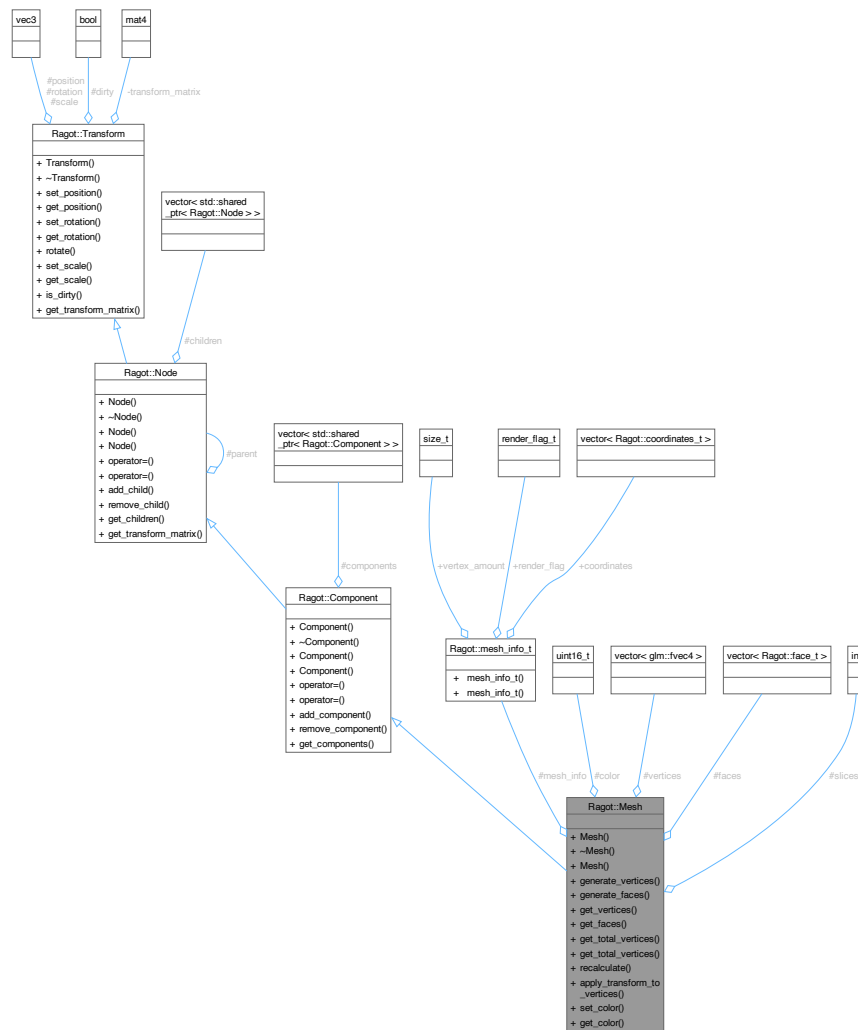
6.14 Ragot::Mesh Class Reference

Represents a 3D mesh in the [Ragot](#) engine.

```
#include <Mesh.hpp>
```

Inheritance diagram for Ragot::Mesh:





- `Mesh ()=delete`
Construct a new `Mesh` object (deleted constructor).
- `virtual ~Mesh ()=default`
Default virtual destructor for the `Mesh` class.
- `Mesh (mesh_info_t &mesh_info)`
Construct a new `Mesh` object with mesh information.
- `virtual void generate_vertices ()=0`
Generate vertices for the mesh.
- `virtual void generate_faces ()=0`
Generate faces for the mesh.
- `const std::vector< glm::fvec4 > &get_vertices () const`
Get the vertices object.
- `const std::vector< face_t > &get_faces () const`
Get the faces object.

- `const size_t get_total_vertices () const`
Get the total vertices object.
- `size_t get_total_vertices ()`
Get the total vertices object.
- `void recalculate ()`
Recalculate the mesh vertices and faces.
- `void apply_transform_to_vertices ()`
Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the [Transform](#) class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- `void set_color (uint16_t new_color)`
Set the color of the mesh.
- `uint16_t get_color () const`
Get the color of the mesh.

Public Member Functions inherited from [Ragot::Component](#)

- `Component ()=default`
Default constructor for the [Component](#) class.
- `virtual ~Component ()=default`
Default virtual destructor for the [Component](#) class.
- `Component (const Component &)=delete`
Deleted copy constructor for the [Component](#) class.
- `Component (const Component &&)=delete`
Deleted move constructor for the [Component](#) class.
- `Component & operator= (const Component &)=delete`
Deleted assignment operator for the [Component](#) class.
- `Component & operator= (const Component &&)=delete`
Deleted move assignment operator for the [Component](#) class.
- `void add_component (std::shared_ptr< Component > component)`
Adds a component to the collection.
- `void remove_component (std::shared_ptr< Component > component)`
Removes a component from the collection.
- `const std::vector< std::shared_ptr< Component > > get_components () const`
Gets the collection of components.

Public Member Functions inherited from [Ragot::Node](#)

- `Node ()=default`
Default constructor for [Node](#). Initializes an empty node with no parent and no children.
- `virtual ~Node ()=default`
Default destructor for [Node](#). Cleans up the node and its children.
- `Node (const Node &)=delete`
Deleted copy constructor for [Node](#). Prevents copying of [Node](#) instances.
- `Node (const Node &&)=delete`
Deleted move constructor for [Node](#). Prevents moving of [Node](#) instances.
- `Node & operator= (const Node &)=delete`
Deleted assignment operator for [Node](#). Prevents assignment of [Node](#) instances.
- `Node & operator= (const Node &&)=delete`

- Deleted move assignment operator for [Node](#). Prevents moving of [Node](#) instances.
- void [add_child](#) (std::shared_ptr< [Node](#) > child)
Get the parent node.
- void [remove_child](#) (std::shared_ptr< [Node](#) > child)
Remove a child node.
- const std::vector< std::shared_ptr< [Node](#) > > & [get_children](#) () const
Get the parent node.
- mat4 [get_transform_matrix](#) () override
Get the transform matrix object.

Public Member Functions inherited from [Ragot::Transform](#)

- [Transform](#) ()
Default constructor for the [Transform](#) class.
- virtual [~Transform](#) ()=default
Virtual destructor for the [Transform](#) class.
- void [set_position](#) (const vec3 &pos)
Sets the position of the object.
- vec3 [get_position](#) () const
Gets the current position of the object.
- void [set_rotation](#) (const vec3 &rot)
Moves the object by a specified vector.
- vec3 [get_rotation](#) () const
Gets the current rotation of the object.
- void [rotate](#) (const float angle, const vec3 &axis)
Rotates the object by a specified angle around a given axis.
- void [set_scale](#) (const vec3 &scale)
Sets the scale of the object.
- vec3 [get_scale](#) () const
Sets the scale of the object uniformly.
- bool [is_dirty](#) () const
Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- [mesh_info_t](#) [mesh_info](#)
Information about the mesh, including coordinates and rendering type.
- uint16_t [color](#) = 0xFFFF
Color of the mesh, default is white (0xFFFF).
- std::vector< glm::fvec4 > [vertices](#)
Vector of vertices representing the mesh in 3D space.
- std::vector< [face_t](#) > [faces](#)
Vector of faces representing the mesh, each face can be a triangle or a quad.
- int [slices](#) = 16
Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from [Ragot::Component](#)

- std::vector< std::shared_ptr< [Component](#) > > [components](#)
Collection of components managed by this [Component](#) instance.

Protected Attributes inherited from [Ragot::Node](#)

- `std::vector< std::shared_ptr< Node > > children`

List of child nodes.

- `Node * parent = nullptr`

Pointer to the parent node.

Protected Attributes inherited from [Ragot::Transform](#)

- `vec3 position`

The position of the object in 3D space.

- `vec3 rotation`

The rotation of the object in degrees around each axis (x, y, z).

- `vec3 scale`

The scale of the object in 3D space, default is (1, 1, 1).

- `bool dirty = true`

Flag indicating whether the transformation matrix needs to be recalculated.

6.14.1 Detailed Description

Represents a 3D mesh in the [Ragot](#) engine.

The [Mesh](#) class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

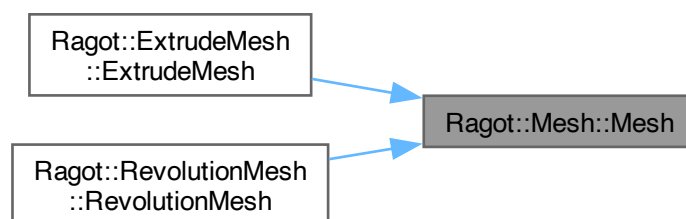
6.14.2 Constructor & Destructor Documentation

6.14.2.1 `Mesh()` [1/2]

```
Ragot::Mesh::Mesh () [delete]
```

Construct a new [Mesh](#) object (deleted constructor).

Here is the caller graph for this function:



6.14.2.2 ~Mesh()

```
virtual Ragot::Mesh::~~Mesh () [virtual], [default]
```

Default virtual destructor for the [Mesh](#) class.

Cleans up the mesh and its resources.

6.14.2.3 Mesh() [2/2]

```
Ragot::Mesh::Mesh (
    mesh_info_t & mesh_info)
```

Construct a new [Mesh](#) object with mesh information.

Initializes the mesh with the provided mesh information.

Parameters

<i>mesh_info</i>	Information about the mesh, including coordinates and rendering type.
------------------	---

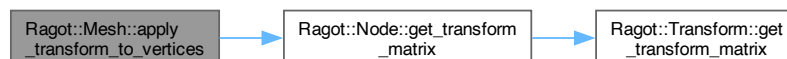
6.14.3 Member Function Documentation

6.14.3.1 apply_transform_to_vertices()

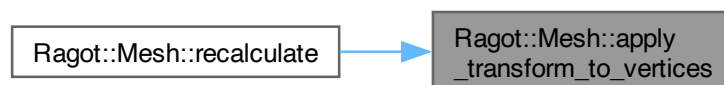
```
void Ragot::Mesh::apply_transform_to_vertices () [inline]
```

Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the [Transform](#) class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.

Here is the call graph for this function:



Here is the caller graph for this function:



6.14.3.2 generate_faces()

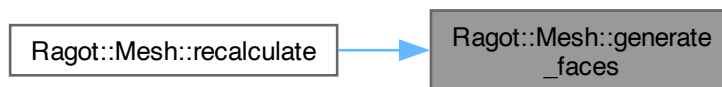
```
virtual void Ragot::Mesh::generate_faces () [pure virtual]
```

Generate faces for the mesh.

This method is pure virtual and must be implemented by derived classes. It is responsible for generating the faces of the mesh based on the vertices generated by generate_vertices.

Implemented in [Ragot::ExtrudeMesh](#), and [Ragot::RevolutionMesh](#).

Here is the caller graph for this function:



6.14.3.3 generate_vertices()

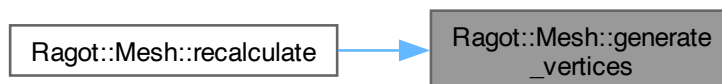
```
virtual void Ragot::Mesh::generate_vertices () [pure virtual]
```

Generate vertices for the mesh.

This method is pure virtual and must be implemented by derived classes. It is responsible for generating the vertices of the mesh based on the mesh information.

Implemented in [Ragot::ExtrudeMesh](#), and [Ragot::RevolutionMesh](#).

Here is the caller graph for this function:



6.14.3.4 get_color()

```
uint16_t Ragot::Mesh::get_color () const [inline]
```

Get the color of the mesh.

This method returns the current color of the mesh.

Returns

uint16_t The color of the mesh.

6.14.3.5 get_faces()

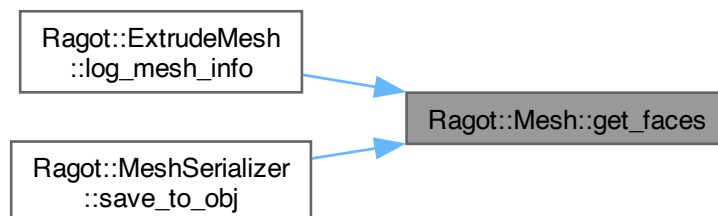
```
const std::vector< face_t > & Ragot::Mesh::get_faces () const [inline]
```

Get the faces object.

Returns

const std::vector < face_t >&

Here is the caller graph for this function:



6.14.3.6 get_total_vertices() [1/2]

```
size_t Ragot::Mesh::get_total_vertices () [inline]
```

Get the total vertices object.

Returns

size_t

6.14.3.7 get_total_vertices() [2/2]

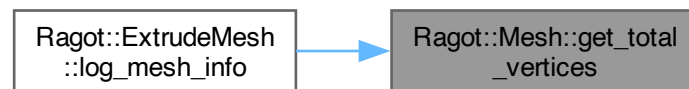
```
const size_t Ragot::Mesh::get_total_vertices () const [inline]
```

Get the total vertices object.

Returns

const size_t

Here is the caller graph for this function:



6.14.3.8 get_vertices()

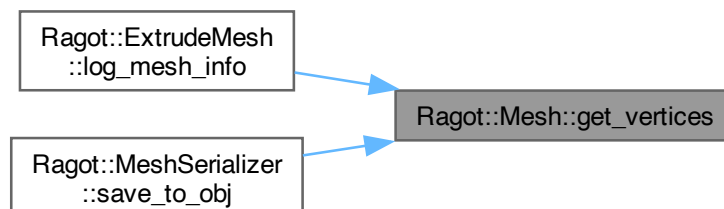
```
const std::vector< glm::fvec4 > & Ragot::Mesh::get_vertices () const [inline]
```

Get the vertices object.

Returns

const std::vector < glm::fvec4 >&

Here is the caller graph for this function:

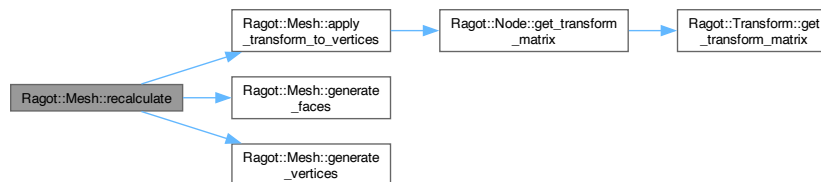


6.14.3.9 recalculate()

```
void Ragot::Mesh::recalculate () [inline]
```

Recalculate the mesh vertices and faces.

Here is the call graph for this function:



6.14.3.10 set_color()

```
void Ragot::Mesh::set_color (
    uint16_t new_color) [inline]
```

Set the color of the mesh.

This method sets the color of the mesh to the specified new color.

Parameters

<code>new_color</code>	The new color to set for the mesh.
------------------------	------------------------------------

6.14.4 Member Data Documentation

6.14.4.1 color

```
uint16_t Ragot::Mesh::color = 0xFFFF [protected]
```

Color of the mesh, default is white (0xFFFF).

6.14.4.2 faces

```
std::vector< face_t > Ragot::Mesh::faces [protected]
```

Vector of faces representing the mesh, each face can be a triangle or a quad.

6.14.4.3 mesh_info

```
mesh_info_t Ragot::Mesh::mesh_info [protected]
```

Information about the mesh, including coordinates and rendering type.

6.14.4.4 slices

```
int Ragot::Mesh::slices = 16 [protected]
```

Number of slices for generating the mesh, default is 16.

6.14.4.5 vertices

```
std::vector< glm::fvec4 > Ragot::Mesh::vertices [protected]
```

Vector of vertices representing the mesh in 3D space.

The documentation for this class was generated from the following files:

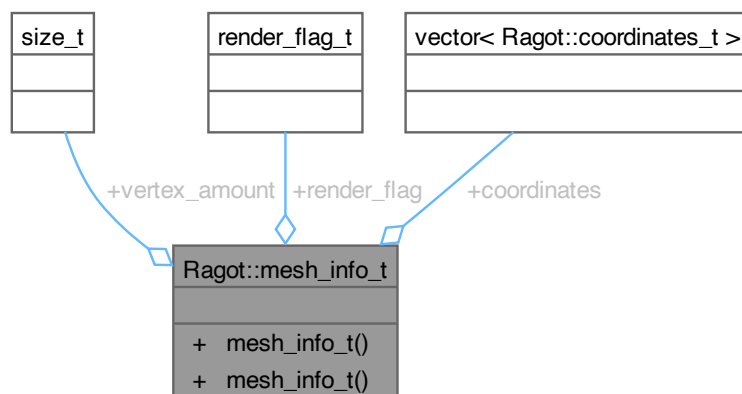
- [main/Mesh.hpp](#)
- [main/Mesh.cpp](#)

6.15 Ragot::mesh_info_t Struct Reference

Represents information about a mesh.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::mesh_info_t:



Public Member Functions

- [mesh_info_t](#) ()=default
- [mesh_info_t](#) (std::vector< [coordinates_t](#) > &coords, [render_flag_t](#) flag)

Public Attributes

- size_t [vertex_amount](#) = 0
- [render_flag_t](#) [render_flag](#) = [RENDER_NONE](#)
- std::vector< [coordinates_t](#) > [coordinates](#)

6.15.1 Detailed Description

Represents information about a mesh.

This structure holds the number of vertices, rendering flags, and a vector of coordinates for a mesh.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 [mesh_info_t](#)() [1/2]

```
Ragot::mesh_info_t::mesh_info_t () [default]
```

6.15.2.2 [mesh_info_t](#)() [2/2]

```
Ragot::mesh_info_t::mesh_info_t (  
    std::vector< coordinates\_t > & coords,  
    render\_flag\_t flag) [inline]
```

6.15.3 Member Data Documentation

6.15.3.1 [coordinates](#)

```
std::vector< coordinates\_t > Ragot::mesh_info_t::coordinates
```

6.15.3.2 [render_flag](#)

```
render\_flag\_t Ragot::mesh_info_t::render_flag = RENDER\_NONE
```

6.15.3.3 [vertex_amount](#)

```
size_t Ragot::mesh_info_t::vertex_amount = 0
```

The documentation for this struct was generated from the following file:

- main/[CommonTypes.hpp](#)

6.16 Ragot::MeshSerializer Class Reference

Singleton class to serialize [Mesh](#) objects to OBJ file format.

```
#include <MeshSerializer.hpp>
```

Collaboration diagram for Ragot::MeshSerializer:

Ragot::MeshSerializer
<ul style="list-style-type: none"> + save_to_obj() + instance() - MeshSerializer() - MeshSerializer() - MeshSerializer() - operator=() - operator=()

Public Member Functions

- bool [save_to_obj](#) (const [Mesh](#) &mesh, const std::filesystem::path &path)
Saves a [Mesh](#) object to an OBJ file.

Static Public Member Functions

- static [MeshSerializer](#) & [instance](#) ()
Gets the singleton instance of the [MeshSerializer](#) class.

Private Member Functions

- [MeshSerializer](#) ()=default
Private constructor to prevent instantiation from outside the class.
- [MeshSerializer](#) (const [MeshSerializer](#) &)=delete
Construct a new [Mesh](#) Serializer object (deleted).
- [MeshSerializer](#) (const [MeshSerializer](#) &&)=delete
Construct a new [Mesh](#) Serializer object (deleted).
- [MeshSerializer](#) & [operator=](#) (const [MeshSerializer](#) &)=delete
Assignment operator for the [MeshSerializer](#) class (deleted).
- [MeshSerializer](#) & [operator=](#) (const [MeshSerializer](#) &&)=delete
Assignment operator for the [MeshSerializer](#) class (deleted).

6.16.1 Detailed Description

Singleton class to serialize [Mesh](#) objects to OBJ file format.

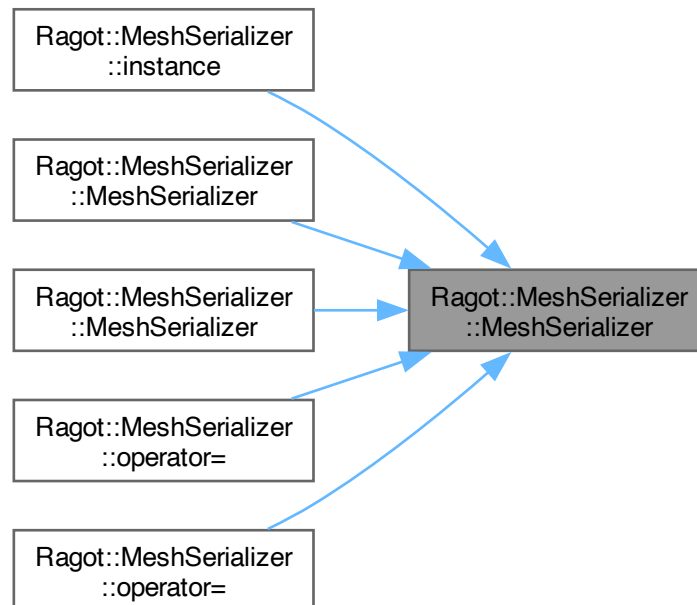
6.16.2 Constructor & Destructor Documentation

6.16.2.1 MeshSerializer() [1/3]

```
Ragot::MeshSerializer::MeshSerializer () [private], [default]
```

Private constructor to prevent instantiation from outside the class.

This constructor is private to enforce the singleton pattern, ensuring that only one instance of [MeshSerializer](#) exists. Here is the caller graph for this function:

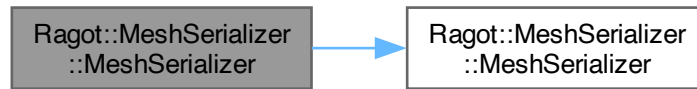


6.16.2.2 MeshSerializer() [2/3]

```
Ragot::MeshSerializer::MeshSerializer (
    const MeshSerializer & ) [private], [delete]
```

Construct a new [Mesh](#) Serializer object (deleted).

Here is the call graph for this function:

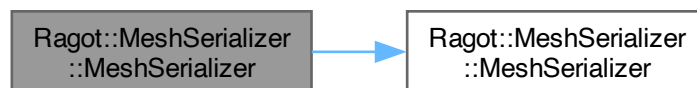


6.16.2.3 MeshSerializer() [3/3]

```
Ragot::MeshSerializer::MeshSerializer (
    const MeshSerializer && ) [private], [delete]
```

Construct a new [Mesh](#) Serializer object (deleted).

This constructor is deleted to prevent moving the [MeshSerializer](#) instance. Here is the call graph for this function:



6.16.3 Member Function Documentation

6.16.3.1 instance()

```
static MeshSerializer & Ragot::MeshSerializer::instance () [inline], [static]
```

Gets the singleton instance of the [MeshSerializer](#) class.

This method ensures that only one instance of [MeshSerializer](#) exists throughout the application.

Returns

[MeshSerializer](#)& Reference to the singleton instance of [MeshSerializer](#).

Here is the call graph for this function:



6.16.3.2 operator=() [1/2]

```
MeshSerializer & Ragot::MeshSerializer::operator= (
    const MeshSerializer && ) [private], [delete]
```

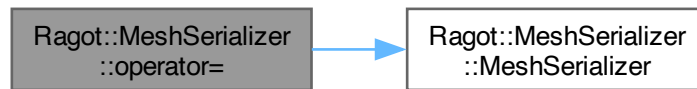
Assignment operator for the [MeshSerializer](#) class (deleted).

This operator is deleted to prevent moving [MeshSerializer](#) instances.

Returns

[MeshSerializer](#)& Reference to the current object.

Here is the call graph for this function:

**6.16.3.3 operator=()** [2/2]

```
MeshSerializer & Ragot::MeshSerializer::operator= (
    const MeshSerializer & ) [private], [delete]
```

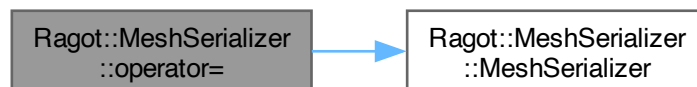
Assignment operator for the [MeshSerializer](#) class (deleted).

This operator is deleted to prevent assignment of [MeshSerializer](#) instances.

Returns

[MeshSerializer](#)& Reference to the current object.

Here is the call graph for this function:

**6.16.3.4 save_to_obj()**

```
bool Ragot::MeshSerializer::save_to_obj (
    const Mesh & mesh,
    const std::filesystem::path & path)
```

Saves a [Mesh](#) object to an OBJ file.

This method serializes the vertices and faces of the [Mesh](#) object and writes them to the specified OBJ file path.

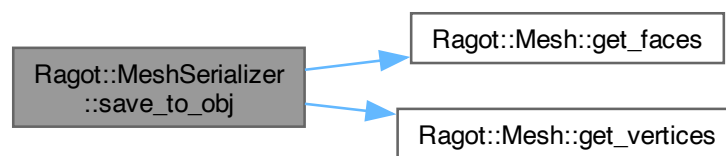
Parameters

<i>mesh</i>	The Mesh object to serialize.
<i>path</i>	The filesystem path where the OBJ file will be saved.

Returns

true if the serialization was successful, false otherwise.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

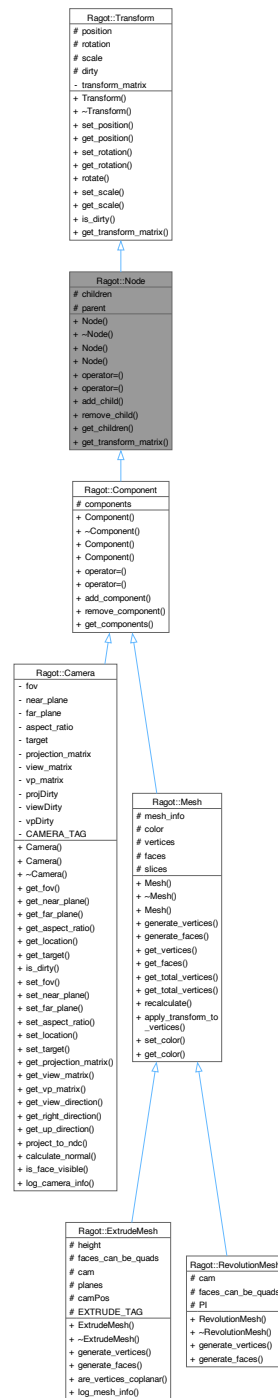
- [main/MeshSerializer.hpp](#)
- [main/MeshSerializer.cpp](#)

6.17 Ragot::Node Class Reference

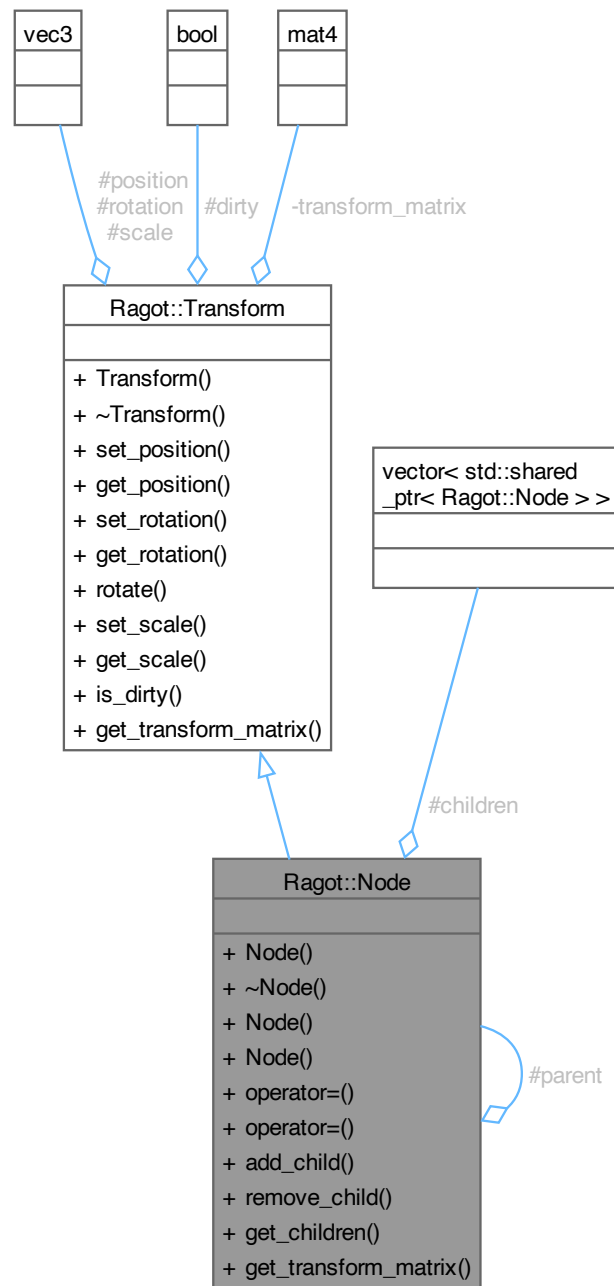
Represents a node in a scene graph for 3D rendering.

```
#include <Node.hpp>
```


Inheritance diagram for Ragot::Node:



Collaboration diagram for Ragot::Node:



Public Member Functions

- `Node()`=default
Default constructor for `Node`. Initializes an empty node with no parent and no children.
- virtual `~Node()`=default
Default destructor for `Node`. Cleans up the node and its children.
- `Node (const Node &)=delete`

- Deleted copy constructor for Node. Prevents copying of Node instances.*

 - `Node (const Node &&)=delete`
- Deleted move constructor for Node. Prevents moving of Node instances.*

 - `Node & operator= (const Node &)=delete`
- Deleted assignment operator for Node. Prevents assignment of Node instances.*

 - `Node & operator= (const Node &&)=delete`
- Deleted move assignment operator for Node. Prevents moving of Node instances.*

 - `void add_child (std::shared_ptr< Node > child)`
- Get the parent node.*

 - `void remove_child (std::shared_ptr< Node > child)`
- Remove a child node.*

 - `const std::vector< std::shared_ptr< Node > > & get_children () const`
- Get the parent node.*

 - `mat4 get_transform_matrix () override`
- Get the transform matrix object.*

Public Member Functions inherited from Ragot::Transform

- `Transform ()`
- Default constructor for the Transform class.*
- `virtual ~Transform ()=default`
- Virtual destructor for the Transform class.*
- `void set_position (const vec3 &pos)`
- Sets the position of the object.*
- `vec3 get_position () const`
- Gets the current position of the object.*
- `void set_rotation (const vec3 &rot)`
- Moves the object by a specified vector.*
- `vec3 get_rotation () const`
- Gets the current rotation of the object.*
- `void rotate (const float angle, const vec3 &axis)`
- Rotates the object by a specified angle around a given axis.*
- `void set_scale (const vec3 &scale)`
- Sets the scale of the object.*
- `vec3 get_scale () const`
- Sets the scale of the object uniformly.*
- `bool is_dirty () const`
- Checks if the transformation matrix is dirty (needs recalculation).*

Protected Attributes

- `std::vector< std::shared_ptr< Node > > children`
- List of child nodes.*
- `Node * parent = nullptr`
- Pointer to the parent node.*

Protected Attributes inherited from [Ragot::Transform](#)

- `vec3 position`
The position of the object in 3D space.
- `vec3 rotation`
The rotation of the object in degrees around each axis (x, y, z).
- `vec3 scale`
The scale of the object in 3D space, default is (1, 1, 1).
- `bool dirty = true`
Flag indicating whether the transformation matrix needs to be recalculated.

6.17.1 Detailed Description

Represents a node in a scene graph for 3D rendering.

The [Node](#) class extends the [Transform](#) class to include child nodes, allowing for hierarchical transformations and management of child nodes.

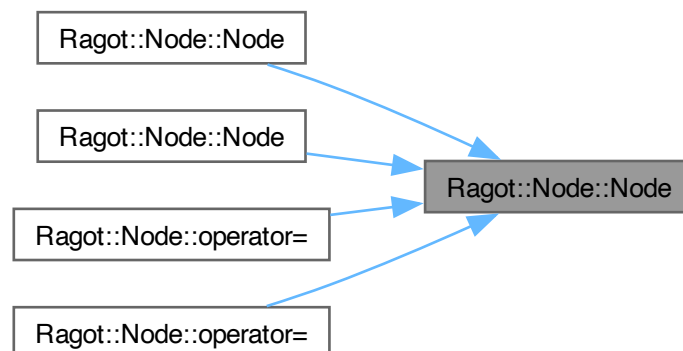
6.17.2 Constructor & Destructor Documentation

6.17.2.1 `Node()` [1/3]

```
Ragot::Node::Node () [default]
```

Default constructor for [Node](#). Initializes an empty node with no parent and no children.

Here is the caller graph for this function:



6.17.2.2 `~Node()`

```
virtual Ragot::Node::~~Node () [virtual], [default]
```

Default destructor for [Node](#). Cleans up the node and its children.

6.17.2.3 Node() [2/3]

```
Ragot::Node::Node (
    const Node & ) [delete]
```

Deleted copy constructor for `Node`. Prevents copying of `Node` instances.

Here is the call graph for this function:



6.17.2.4 Node() [3/3]

```
Ragot::Node::Node (
    const Node && ) [delete]
```

Deleted move constructor for `Node`. Prevents moving of `Node` instances.

Here is the call graph for this function:



6.17.3 Member Function Documentation

6.17.3.1 add_child()

```
void Ragot::Node::add_child (
    std::shared_ptr< Node > child) [inline]
```

Get the parent node.

Returns

`Node*` Pointer to the parent node, or `nullptr` if no parent exists.

6.17.3.2 get_children()

```
const std::vector< std::shared_ptr< Node > > & Ragot::Node::get_children () const [inline]
```

Get the parent node.

Returns

Node* Pointer to the parent node, or nullptr if no parent exists.

6.17.3.3 get_transform_matrix()

```
mat4 Ragot::Node::get_transform_matrix () [inline], [override], [virtual]
```

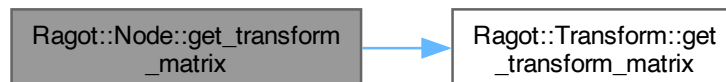
Get the transform matrix object.

Returns

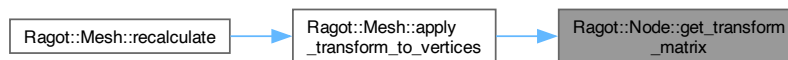
mat4 The transformation matrix of the node, including its parent's transformation.

Reimplemented from [Ragot::Transform](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.17.3.4 operator=() [1/2]

```
Node & Ragot::Node::operator= (
    const Node && ) [delete]
```

Deleted move assignment operator for `Node`. Prevents moving of `Node` instances.

Returns

`Node&` Reference to the current object.

Here is the call graph for this function:



6.17.3.5 operator=() [2/2]

```
Node & Ragot::Node::operator= (
    const Node & ) [delete]
```

Deleted assignment operator for `Node`. Prevents assignment of `Node` instances.

Here is the call graph for this function:



6.17.3.6 remove_child()

```
void Ragot::Node::remove_child (
    std::shared_ptr< Node > child) [inline]
```

Remove a child node.

This method removes the specified child node from the list of children. If the child exists, it is removed and its parent pointer is set to nullptr.

Parameters

<i>child</i>	The child node to remove.
--------------	---------------------------

6.17.4 Member Data Documentation

6.17.4.1 children

```
std::vector< std::shared_ptr < Node > > Ragot::Node::children [protected]
```

List of child nodes.

6.17.4.2 parent

```
Node* Ragot::Node::parent = nullptr [protected]
```

Pointer to the parent node.

The documentation for this class was generated from the following file:

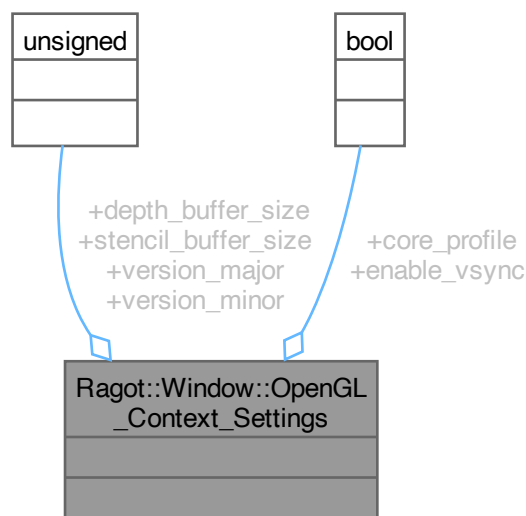
- main/[Node.hpp](#)

6.18 Ragot::Window::OpenGL_Context_Settings Struct Reference

Struct for OpenGL context settings.

```
#include <Window.hpp>
```

Collaboration diagram for Ragot::Window::OpenGL_Context_Settings:



Public Attributes

- unsigned `version_major` = 3
Major version of OpenGL.
- unsigned `version_minor` = 3
Minor version of OpenGL.
- bool `core_profile` = true
Core profile flag.
- unsigned `depth_buffer_size` = 24
Depth buffer size.
- unsigned `stencil_buffer_size` = 0
Stencil buffer size.
- bool `enable_vsync` = true
V-Sync enable flag.

6.18.1 Detailed Description

Struct for OpenGL context settings.

6.18.2 Member Data Documentation

6.18.2.1 `core_profile`

```
bool Ragot::Window::OpenGL_Context_Settings::core_profile = true
```

Core profile flag.

6.18.2.2 `depth_buffer_size`

```
unsigned Ragot::Window::OpenGL_Context_Settings::depth_buffer_size = 24
```

Depth buffer size.

6.18.2.3 `enable_vsync`

```
bool Ragot::Window::OpenGL_Context_Settings::enable_vsync = true
```

V-Sync enable flag.

6.18.2.4 `stencil_buffer_size`

```
unsigned Ragot::Window::OpenGL_Context_Settings::stencil_buffer_size = 0
```

Stencil buffer size.

6.18.2.5 version_major

```
unsigned Ragot::Window::OpenGL_Context_Settings::version_major = 3
```

Major version of OpenGL.

6.18.2.6 version_minor

```
unsigned Ragot::Window::OpenGL_Context_Settings::version_minor = 3
```

Minor version of OpenGL.

The documentation for this struct was generated from the following file:

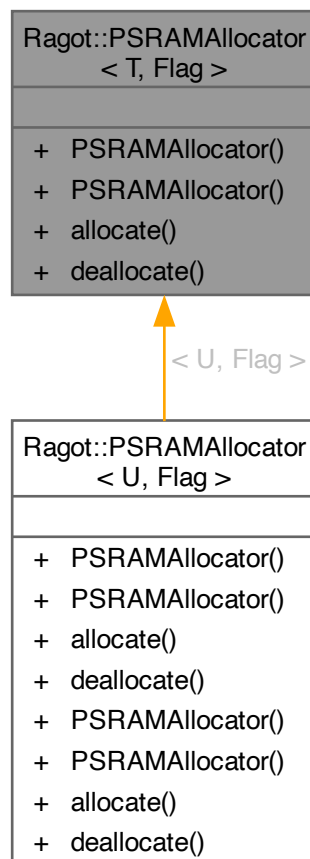
- [main/Window.hpp](#)

6.19 Ragot::PSRAMAllocator< T, Flag > Class Template Reference

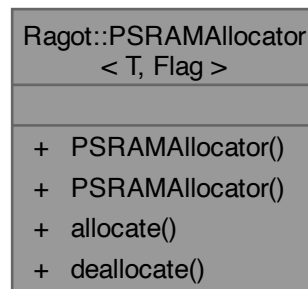
Custom memory allocator for PSRAM.

```
#include <RamAllocator.hpp>
```

Inheritance diagram for Ragot::PSRAMAllocator< T, Flag >:



Collaboration diagram for Ragot::PSRAMAllocator< T, Flag >:



Classes

- struct [rebind](#)

Rebinds the allocator to a different type. This struct allows the [PSRAMAllocator](#) to be used with different types while maintaining the same allocation flags.

Public Types

- using [value_type](#) = T
- using [pointer](#) = T*
- using [size_type](#) = std::size_t

Public Member Functions

- [PSRAMAllocator](#) () noexcept
Default constructor for [PSRAMAllocator](#).
- template<typename U, uint16_t F2>
[PSRAMAllocator](#) (const [PSRAMAllocator](#)< U, F2 > &) noexcept
Copy constructor for [PSRAMAllocator](#).
- T * [allocate](#) ([size_type](#) n)
Allocates memory for n objects of type T in PSRAM.
- void [deallocate](#) (T *p, [size_type](#)) noexcept
Deallocates memory for n objects of type T in PSRAM.

6.19.1 Detailed Description

```
template<typename T, uint16_t Flag>
class Ragot::PSRAMAllocator< T, Flag >
```

Custom memory allocator for PSRAM.

This class provides a custom memory allocator that uses PSRAM with specific flags. It can be used with standard containers like `std::vector` to manage memory in embedded systems.

6.19.2 Member Typedef Documentation

6.19.2.1 pointer

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::pointer = T*
```

6.19.2.2 size_type

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::size_type = std::size_t
```

6.19.2.3 value_type

```
template<typename T, uint16_t Flag>
using Ragot::PSRAMAllocator< T, Flag >::value_type = T
```

6.19.3 Constructor & Destructor Documentation

6.19.3.1 PSRAMAllocator() [1/2]

```
template<typename T, uint16_t Flag>
Ragot::PSRAMAllocator< T, Flag >::PSRAMAllocator () [inline], [noexcept]
```

Default constructor for [PSRAMAllocator](#).

This constructor initializes the [PSRAMAllocator](#) without any specific parameters.

6.19.3.2 PSRAMAllocator() [2/2]

```
template<typename T, uint16_t Flag>
template<typename U, uint16_t F2>
Ragot::PSRAMAllocator< T, Flag >::PSRAMAllocator (
    const PSRAMAllocator< U, F2 > & ) [inline], [noexcept]
```

Copy constructor for [PSRAMAllocator](#).

This constructor allows copying of the [PSRAMAllocator](#), but it does not perform any specific actions. It is designed to be used with standard containers that require copyable allocators.

Template Parameters

<i>U</i>	The type to rebind to.
----------	------------------------

Parameters

<i>other</i>	The allocator to copy from.
--------------	-----------------------------

6.19.4 Member Function Documentation

6.19.4.1 allocate()

```
template<typename T, uint16_t Flag>
T * Ragot::PSRAMAllocator< T, Flag >::allocate (
    size_type n)    [inline]
```

Allocates memory for n objects of type T in PSRAM.

This method allocates memory for n objects of type T using heap_caps_malloc with the specified flags. If the allocation fails, it throws std::bad_alloc.

Parameters

<i>n</i>	The number of objects to allocate memory for.
----------	---

Returns

T* Pointer to the allocated memory.

6.19.4.2 deallocate()

```
template<typename T, uint16_t Flag>
void Ragot::PSRAMAllocator< T, Flag >::deallocate (
    T * p,
    size_type )    [inline], [noexcept]
```

Deallocates memory for n objects of type T in PSRAM.

This method deallocates memory for n objects of type T using heap_caps_free. It does not throw any exceptions.

Parameters

<i>p</i>	Pointer to the memory to deallocate.
<i>n</i>	The number of objects to deallocate (not used).

The documentation for this class was generated from the following file:

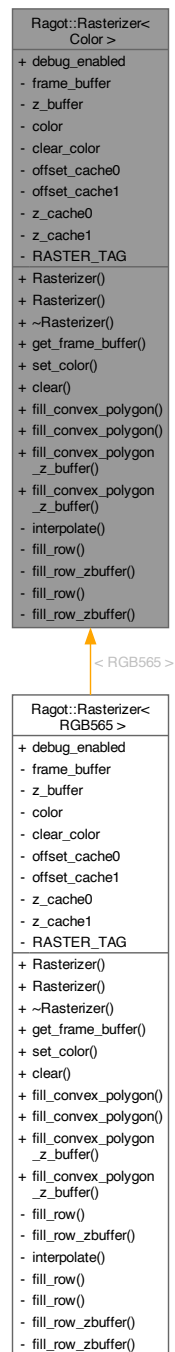
- main/[RamAllocator.hpp](#)

6.20 Ragot::Rasterizer< Color > Class Template Reference

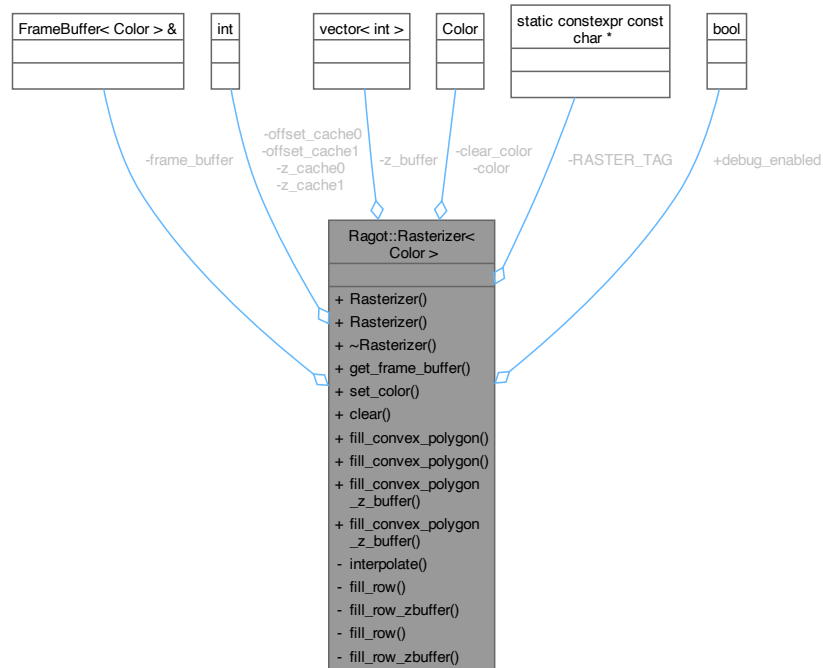
Class for rasterizing polygons in a frame buffer.

```
#include <Rasterizer.hpp>
```

Inheritance diagram for Ragot::Rasterizer< Color >:



Collaboration diagram for Ragot::Rasterizer< Color >:



Public Member Functions

- **Rasterizer** (**FrameBuffer**< **Color** > &frame)
Constructs a **Rasterizer** with a given frame buffer.
- **Rasterizer** ()=default
Default constructor for the **Rasterizer** class (Default).
- **~Rasterizer** ()=default
Default destructor for the **Rasterizer** class.
- const **FrameBuffer**< **Color** > &**get_frame_buffer** () const
Gets the frame buffer associated with this rasterizer.
- void **set_color** (const **Color** &new_color)
Gets the current color used for drawing polygons.
- void **clear** ()
Gets the current color used for clearing the frame buffer.
- void **fill_convex_polygon** (const glm::ivec4 *const vertices, const int *const indices_begin, const int *const indices_end)
Fills a convex polygon defined by its vertices and indices.
- void **fill_convex_polygon** (const glm::ivec4 *const vertices, const **face_t** *const face)
Fills a convex polygon defined by its vertices and face structure.
- void **fill_convex_polygon_z_buffer** (const glm::ivec4 *const vertices, const **face_t** *const face)
Fills a convex polygon in the Z-buffer.
- void **fill_convex_polygon_z_buffer** (const glm::ivec4 *const vertices, const int *const indices_begin, const int *const indices_end)
Fills a convex polygon in the Z-buffer using vertex indices.

Public Attributes

- bool `debug_enabled` = true

Private Member Functions

- template<typename VALUE_TYPE, size_t SHIFT>
void `interpolate` (int *cache, int v0, int v1, int y_min, int y_max)
< Enable or disable debug logging for rasterization operations.
- template<unsigned COLOR_SIZE>
void `fill_row` (Color *start, unsigned left_offset, unsigned right_offset, const Color &color)
Fills a row of pixels in the frame buffer with a specified color.
- template<unsigned COLOR_SIZE>
void `fill_row_zbuffer` (Color *start, int *zbuffer, unsigned left_offset, unsigned right_offset, int z_start, int dz, const Color &color)
Fills a row of pixels in the frame buffer with a specified color using Z-buffering.
- void `fill_row` (RGB565 *start, unsigned left_offset, unsigned right_offset, const RGB565 &color)
- void `fill_row_zbuffer` (RGB565 *start, int *zbuffer, unsigned left_offset, unsigned right_offset, int z_start, int dz, const RGB565 &color)

Private Attributes

- `FrameBuffer`< Color > & `frame_buffer`
Reference to the frame buffer where polygons will be drawn.
- std::vector< int > `z_buffer`
Z-buffer for depth testing, used when painter's algorithm is disabled.
- Color `color`
Current color to be used for drawing polygons.
- Color `clear_color`
Color used to clear the frame buffer, default is black.

Static Private Attributes

- static int `offset_cache0` [1024]
Cache for left offsets of scanlines >
- static int `offset_cache1` [1024]
Cache for right offsets of scanlines.
- static int `z_cache0` [1024]
Cache for Z-buffer values for left offsets.
- static int `z_cache1` [1024]
Cache for Z-buffer values for right offsets.
- static constexpr const char * `RASTER_TAG` = "Rasterizer"

6.20.1 Detailed Description

```
template<typename Color>
class Ragot::Rasterizer< Color >
```

Class for rasterizing polygons in a frame buffer.

This class provides methods to fill convex polygons in a frame buffer with a specified color. It supports both standard rasterization and Z-buffering techniques.

Template Parameters

<i>Color</i>	The color type used for the frame buffer (e.g., RGB565).
--------------	---

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Rasterizer() [1/2]

```
template<typename Color>
Ragot::Rasterizer< Color >::Rasterizer (
    FrameBuffer< Color > & frame) [inline]
```

Constructs a [Rasterizer](#) with a given frame buffer.

Initializes the rasterizer with the specified frame buffer and prepares the Z-buffer if needed.

Parameters

<i>frame</i>	Reference to the frame buffer where polygons will be drawn.
--------------	---

6.20.2.2 Rasterizer() [2/2]

```
template<typename Color>
Ragot::Rasterizer< Color >::Rasterizer () [default]
```

Default constructor for the [Rasterizer](#) class (Default).

6.20.2.3 ~Rasterizer()

```
template<typename Color>
Ragot::Rasterizer< Color >::~~Rasterizer () [default]
```

Default destructor for the [Rasterizer](#) class.

Cleans up resources used by the rasterizer.

6.20.3 Member Function Documentation

6.20.3.1 clear()

```
template<typename Color>
void Ragot::Rasterizer< Color >::clear () [inline]
```

Gets the current color used for clearing the frame buffer.

This method returns the color that will be used to clear the frame buffer.

Returns

const Color& Reference to the clear color.

6.20.3.2 fill_convex_polygon() [1/2]

```
template<typename COLOR>
void Ragot::Rasterizer< COLOR >::fill_convex_polygon (
    const glm::ivec4 *const vertices,
    const face_t *const face)
```

Fills a convex polygon defined by its vertices and face structure.

This method fills a convex polygon in the frame buffer using the specified vertices and face structure.

Parameters

<i>vertices</i>	Pointer to an array of vertices defining the polygon.
<i>face</i>	Pointer to the face structure containing vertex indices.

6.20.3.3 fill_convex_polygon() [2/2]

```
template<typename COLOR>
void Ragot::Rasterizer< COLOR >::fill_convex_polygon (
    const glm::ivec4 *const vertices,
    const int *const indices_begin,
    const int *const indices_end)
```

Fills a convex polygon defined by its vertices and indices.

This method fills a convex polygon in the frame buffer using the specified vertices and indices.

Parameters

<i>vertices</i>	Pointer to an array of vertices defining the polygon.
<i>indices_begin</i>	Pointer to the beginning of the indices array.
<i>indices_end</i>	Pointer to the end of the indices array.

Here is the call graph for this function:



6.20.3.4 fill_convex_polygon_z_buffer() [1/2]

```
template<typename COLOR>
template void Ragot::Rasterizer< Color >::fill_convex_polygon_z_buffer (
    const glm::ivec4 *const vertices,
    const face_t *const face)
```

Fills a convex polygon in the Z-buffer.

This method fills a convex polygon in the Z-buffer using the specified vertices and face structure. It performs depth testing to ensure correct rendering order.

Parameters

<i>vertices</i>	Pointer to an array of vertices defining the polygon.
<i>face</i>	Pointer to the face structure containing vertex indices.

6.20.3.5 fill_convex_polygon_z_buffer() [2/2]

```
template<typename COLOR>
template void Ragot::Rasterizer< Color >::fill_convex_polygon_z_buffer (
    const glm::ivec4 *const vertices,
    const int *const indices_begin,
    const int *const indices_end)
```

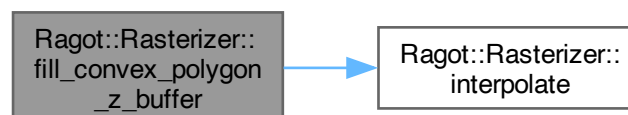
Fills a convex polygon in the Z-buffer using vertex indices.

- This method fills a convex polygon in the Z-buffer using the specified vertices and indices. It performs depth testing to ensure correct rendering order.

Parameters

<i>vertices</i>	Pointer to an array of vertices defining the polygon.
<i>indices_begin</i>	Pointer to the beginning of the indices array.
<i>indices_end</i>	Pointer to the end of the indices array.

Here is the call graph for this function:



6.20.3.6 fill_row() [1/2]

```
template<typename Color>
template<unsigned COLOR_SIZE>
void Ragot::Rasterizer< Color >::fill_row (
    Color * start,
    unsigned left_offset,
    unsigned right_offset,
    const Color & color) [inline], [private]
```

Fills a row of pixels in the frame buffer with a specified color.

This method fills a row of pixels in the frame buffer with the specified color, from left_offset to right_offset.

Template Parameters

<i>COLOR_SIZE</i>	The size of the color type in bytes.
-------------------	--------------------------------------

Parameters

<i>start</i>	Pointer to the first pixel of the scanline.
<i>left_offset</i>	The starting offset (inclusive).
<i>right_offset</i>	The ending offset (exclusive).
<i>color</i>	The color to fill the row with.

6.20.3.7 fill_row() [2/2]

```
void Ragot::Rasterizer< RGB565 >::fill_row< 2 > (
    RGB565 * start,
    unsigned left_offset,
    unsigned right_offset,
    const RGB565 & color) [private]
```

6.20.3.8 fill_row_zbuffer() [1/2]

```
template<typename Color>
template<unsigned COLOR_SIZE>
void Ragot::Rasterizer< Color >::fill_row_zbuffer (
    Color * start,
    int * zbuffer,
    unsigned left_offset,
    unsigned right_offset,
    int z_start,
    int dz,
    const Color & color) [inline], [private]
```

Fills a row of pixels in the frame buffer with a specified color using Z-buffering.

This method fills a row of pixels in the frame buffer with the specified color, from *left_offset* to *right_offset*, while performing depth testing using the Z-buffer.

Template Parameters

<i>COLOR_SIZE</i>	The size of the color type in bytes.
-------------------	--------------------------------------

Parameters

<i>start</i>	Pointer to the first pixel of the scanline.
<i>zbuffer</i>	Pointer to the first element of the Z-buffer.
<i>left_offset</i>	The starting offset (inclusive).
<i>right_offset</i>	The ending offset (exclusive).
<i>z_start</i>	The initial depth value at <i>left_offset</i> .
<i>dz</i>	The increment of depth per pixel.
<i>color</i>	The color to fill the row with.

6.20.3.9 fill_row_zbuffer() [2/2]

```
void Ragot::Rasterizer< RGB565 >::fill_row_zbuffer< 2 > (
    RGB565 * start,
    int * zbuffer,
    unsigned left_offset,
    unsigned right_offset,
    int z_start,
    int dz,
    const RGB565 & color) [private]
```

6.20.3.10 get_frame_buffer()

```
template<typename Color>
const FrameBuffer< Color > & Ragot::Rasterizer< Color >::get_frame_buffer () const [inline]
```

Gets the frame buffer associated with this rasterizer.

This method returns a reference to the frame buffer where polygons will be drawn.

Returns

const [FrameBuffer<Color>](#) & Reference to the frame buffer.

6.20.3.11 interpolate()

```
template<typename COLOR>
template<typename VALUE_TYPE, size_t SHIFT>
void Ragot::Rasterizer< COLOR >::interpolate (
    int * cache,
    int v0,
    int v1,
    int y_min,
    int y_max) [private]
```

< Enable or disable debug logging for rasterization operations.

Interpolates pixel offsets for a scanline.

This method interpolates pixel offsets for a scanline based on the provided vertex values and Y range. It fills the offset cache with calculated pixel offsets.

Template Parameters

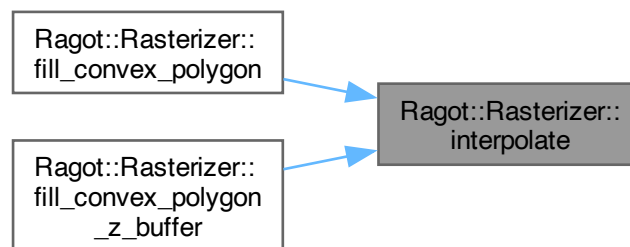
<i>VALUE_TYPE</i>	The type of value used for interpolation (e.g., int32_t).
<i>SHIFT</i>	The bit shift value used for scaling the interpolation.

Parameters

<i>cache</i>	Pointer to the cache where interpolated offsets will be stored.
<i>v0</i>	The starting vertex value.
<i>v1</i>	The ending vertex value.

<i>y_min</i>	The minimum Y coordinate of the scanline.
<i>y_max</i>	The maximum Y coordinate of the scanline.

Here is the caller graph for this function:



6.20.3.12 set_color()

```

template<typename Color>
void Ragot::Rasterizer< Color >::set_color (
    const Color & new_color) [inline]
  
```

Gets the current color used for drawing polygons.

This method returns the current color that will be used to fill polygons.

Returns

const Color& Reference to the current color.

6.20.4 Member Data Documentation

6.20.4.1 clear_color

```

template<typename Color>
Color Ragot::Rasterizer< Color >::clear_color [private]
  
```

Color used to clear the frame buffer, default is black.

6.20.4.2 color

```

template<typename Color>
Color Ragot::Rasterizer< Color >::color [private]
  
```

Current color to be used for drawing polygons.

6.20.4.3 debug_enabled

```
template<typename Color>
bool Ragot::Rasterizer< Color >::debug_enabled = true
```

6.20.4.4 frame_buffer

```
template<typename Color>
FrameBuffer< Color >& Ragot::Rasterizer< Color >::frame_buffer [private]
```

Reference to the frame buffer where polygons will be drawn.

6.20.4.5 offset_cache0

```
template<typename Color>
int Ragot::Rasterizer< Color >::offset_cache0[1024] [static], [private]
```

Cache for left offsets of scanlines >

6.20.4.6 offset_cache1

```
template<typename Color>
int Ragot::Rasterizer< Color >::offset_cache1[1024] [static], [private]
```

Cache for right offsets of scanlines.

6.20.4.7 RASTER_TAG

```
template<typename Color>
const char* Ragot::Rasterizer< Color >::RASTER_TAG = "Rasterizer" [static], [constexpr],
[private]
```

6.20.4.8 z_buffer

```
template<typename Color>
std::vector< int > Ragot::Rasterizer< Color >::z_buffer [private]
```

Z-buffer for depth testing, used when painter's algorithm is disabled.

6.20.4.9 z_cache0

```
template<typename Color>
int Ragot::Rasterizer< Color >::z_cache0[1024] [static], [private]
```

Cache for Z-buffer values for left offsets.

6.20.4.10 z_cache1

```
template<typename Color>
int Ragot::Rasterizer< Color >::z_cache1[1024] [static], [private]
```

Cache for Z-buffer values for right offsets.

The documentation for this class was generated from the following files:

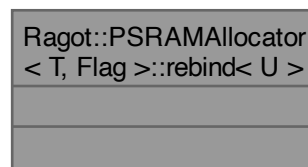
- main/[Rasterizer.hpp](#)
- main/[Rasterizer.cpp](#)

6.21 Ragot::PSRAMAllocator< T, Flag >::rebind< U > Struct Template Reference

Rebinds the allocator to a different type. This struct allows the [PSRAMAllocator](#) to be used with different types while maintaining the same allocation flags.

```
#include <RamAllocator.hpp>
```

Collaboration diagram for Ragot::PSRAMAllocator< T, Flag >::rebind< U >:



Public Types

- using [other](#) = [PSRAMAllocator](#)<U, Flag>

6.21.1 Detailed Description

```
template<typename T, uint16_t Flag>
template<typename U>
struct Ragot::PSRAMAllocator< T, Flag >::rebind< U >
```

Rebinds the allocator to a different type. This struct allows the [PSRAMAllocator](#) to be used with different types while maintaining the same allocation flags.

Template Parameters

<i>U</i>	
----------	--

6.21.2 Member Typedef Documentation

6.21.2.1 other

```
template<typename T, uint16_t Flag>
template<typename U>
using Ragot::PSRAMAllocator< T, Flag >::rebind< U >::other = PSRAMAllocator<U, Flag>
```

The documentation for this struct was generated from the following file:

- main/[RamAllocator.hpp](#)

6.22 Ragot::Renderer Class Reference

Class for rendering scenes in the [Ragot](#) engine.

```
#include <Renderer.hpp>
```


- Performs the rendering task in a separate thread.*
- bool `is_frontface` (const glm::fvec4 *const projected_vertices, const `face_t` *const indices)
Checks if the face defined by the indices is front-facing.
- void `start` ()
Starts the rendering process.
- void `stop` ()
Stops the rendering process.

Public Attributes

- std::vector< glm::fvec4 > `transformed_vertices`
Vector to store transformed vertices for rendering, used to hold the vertices after applying transformations such as model, view, and projection matrices.
- std::vector< glm::ivec4 > `display_vertices`
Vector to store display vertices for rendering, used to hold the vertices after applying viewport transformations and clipping.

Private Member Functions

- void `initFullScreenQuad` ()
Initializes the full-screen quad for rendering, used in non-ESP platforms.

Private Attributes

- float `accumulated_time` = 0.f
Accumulated time for rendering frames, used for timing and performance measurement.
- size_t `iterations` = 0
Number of iterations for rendering, used for performance testing and optimization.
- std::unique_ptr< `Shader_Program` > `quadShader` = nullptr
Number of iterations for performance testing, can be adjusted for different scenarios.
- GLuint `quadVAO` = 0
Vertex Array Object for the full-screen quad, used in non-ESP platforms.
- GLuint `quadVBO` = 0
Vertex `Buffer` Object for the full-screen quad, used in non-ESP platforms.
- GLuint `quadEBO` = 0
Element `Buffer` Object for the full-screen quad, used in non-ESP platforms.
- `FrameBuffer`< `RGB565` > `frame_buffer`
Frame buffer for rendering, used to store pixel data for the rendered scene.
- `Scene` * `current_scene` = nullptr
Pointer to the current scene being rendered, allows access to scene data and objects.
- `Rasterizer`< `RGB565` > `rasterizer`
`Rasterizer` for rendering polygons in the frame buffer, responsible for filling polygons with color and handling depth testing.
- unsigned `width`
Width of the rendering area in pixels, used to define the size of the frame buffer and viewport.
- unsigned `height`
Height of the rendering area in pixels, used to define the size of the frame buffer and viewport.
- bool `initialized` = false
Flag to indicate if the renderer has been initialized, used to prevent re-initialization and ensure resources are set up correctly.
- std::atomic< bool > `running` = false
Flag to indicate if the renderer is currently running, used to control rendering tasks and stop them gracefully.

Static Private Attributes

- static constexpr size_t [number_of_iterations](#) = 10000000000000000
- static const std::string [vertex_shader_code](#)
Vertex shader code for rendering, used in non-ESP platforms.
- static const std::string [fragment_shader_code](#)
Fragment shader code for rendering, used in non-ESP platforms.

6.22.1 Detailed Description

Class for rendering scenes in the [Ragot](#) engine.

This class is responsible for rendering 3D scenes using a rasterization approach. It manages the frame buffer, rasterizer, and scene to be rendered.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 [Renderer\(\)](#) [1/2]

```
Ragot::Renderer::Renderer () [delete]
```

Construct a new [Renderer](#) object (Deleted).

This constructor is deleted to prevent default construction of the [Renderer](#) class.

6.22.2.2 [Renderer\(\)](#) [2/2]

```
Ragot::Renderer::Renderer (
    unsigned width,
    unsigned height)
```

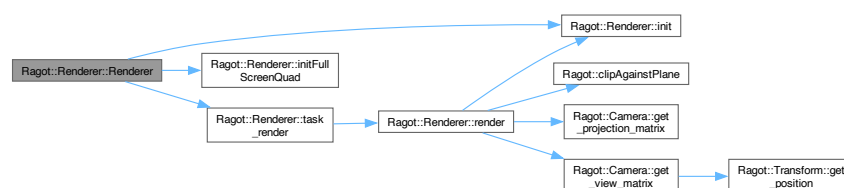
Constructs a [Renderer](#) with the specified width and height.

Initializes the renderer with the given dimensions and prepares the frame buffer and rasterizer.

Parameters

<i>width</i>	The width of the rendering area in pixels.
<i>height</i>	The height of the rendering area in pixels.

Here is the call graph for this function:



6.22.2.3 ~Renderer()

```
Ragot::Renderer::~~Renderer () [default]
```

Default destructor for the [Renderer](#) class.

Cleans up resources used by the renderer.

6.22.3 Member Function Documentation

6.22.3.1 init()

```
void Ragot::Renderer::init ()
```

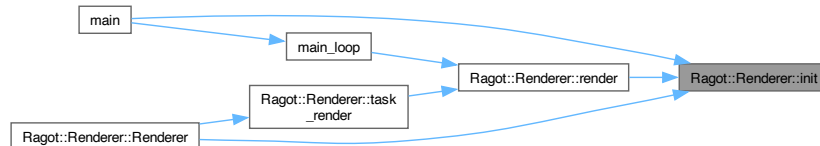
Gets the current scene being rendered.

This method returns a pointer to the current scene being rendered by the renderer. It allows access to the scene's objects and properties for rendering.

Returns

Scene* Pointer to the current [Scene](#) object.

Here is the caller graph for this function:

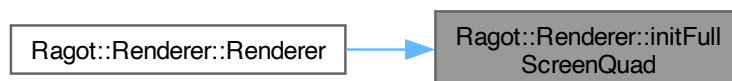


6.22.3.2 initFullScreenQuad()

```
void Ragot::Renderer::initFullScreenQuad () [private]
```

Initializes the full-screen quad for rendering, used in non-ESP platforms.

Here is the caller graph for this function:



6.22.3.3 is_frontface()

```
bool Ragot::Renderer::is_frontface (
    const glm::fvec4 *const projected_vertices,
    const face_t *const indices)
```

Checks if the face defined by the indices is front-facing.

This method checks if the face defined by the indices is front-facing based on the projected vertices. It uses the area of the face to determine its orientation.

Parameters

<i>projected_vertices</i>	Pointer to an array of projected vertices in clip space.
<i>indices</i>	Pointer to the face structure containing vertex indices.

Returns

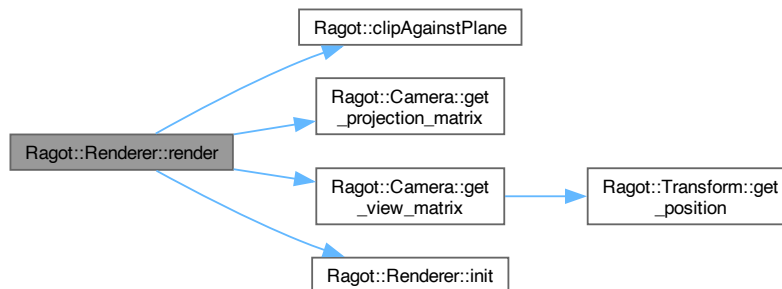
true if the face is front-facing, false otherwise.

6.22.3.4 render()

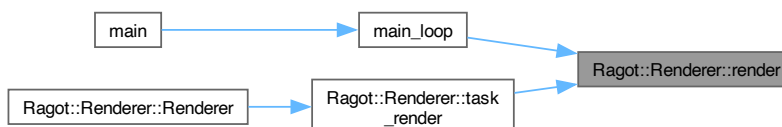
```
void Ragot::Renderer::render ()
```

Renders the current scene.

This method performs the rendering of the current scene by preparing matrices, transforming vertices, and filling polygons in the frame buffer using the rasterizer. It also handles depth testing and color filling for polygons. Here is the call graph for this function:



Here is the caller graph for this function:



6.22.3.5 set_scene()

```
void Ragot::Renderer::set_scene (
    Scene * scene) [inline]
```

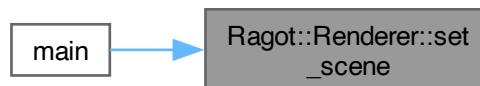
Sets the current scene to be rendered.

This method sets the current scene to be rendered by the renderer. It allows the renderer to access the scene's objects and properties for rendering.

Parameters

<code>scene</code>	Pointer to the <code>Scene</code> object to be set as the current scene.
--------------------	--

Here is the caller graph for this function:



6.22.3.6 start()

```
void Ragot::Renderer::start () [inline]
```

Starts the rendering process.

This method sets the running flag to true, indicating that the renderer is ready to start rendering. Here is the caller graph for this function:



6.22.3.7 stop()

```
void Ragot::Renderer::stop () [inline]
```

Stops the rendering process.

This method sets the running flag to false, indicating that the renderer should stop rendering.

6.22.3.8 task_render()

```
void Ragot::Renderer::task_render (
    std::stop_token stop_token)
```

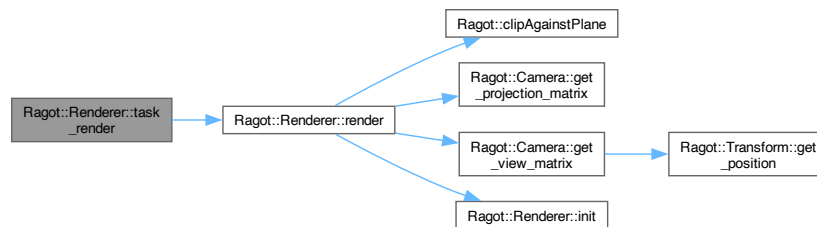
Performs the rendering task in a separate thread.

This method runs the rendering task in a separate thread, allowing for asynchronous rendering.

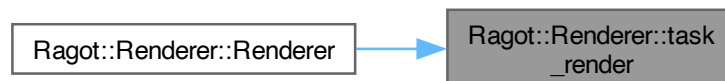
Parameters

<i>stop_token</i>

Here is the call graph for this function:



Here is the caller graph for this function:



6.22.4 Member Data Documentation

6.22.4.1 accumulated_time

```
float Ragot::Renderer::accumulated_time = 0.f [private]
```

Accumulated time for rendering frames, used for timing and performance measurement.

6.22.4.2 current_scene

```
Scene* Ragot::Renderer::current_scene = nullptr [private]
```

Pointer to the current scene being rendered, allows access to scene data and objects.

6.22.4.3 display_vertices

```
std::vector< glm::ivec4 > Ragot::Renderer::display_vertices
```

Vector to store display vertices for rendering, used to hold the vertices after applying viewport transformations and clipping.

6.22.4.4 fragment_shader_code

```
const string Ragot::Renderer::fragment_shader_code [static], [private]
```

Initial value:

```
=
"#version 330\n"
"in vec2 vTex;"
"out vec4 FragColor;"
""
"uniform sampler2D uSampler;"
""
"void main() "
"{ "
"    FragColor = texture(uSampler, vTex);"
"}"
```

Fragment shader code for rendering, used in non-ESP platforms.

6.22.4.5 frame_buffer

```
Framebuffer< RGB565 > Ragot::Renderer::frame_buffer [private]
```

Frame buffer for rendering, used to store pixel data for the rendered scene.

6.22.4.6 height

```
unsigned Ragot::Renderer::height [private]
```

Height of the rendering area in pixels, used to define the size of the frame buffer and viewport.

6.22.4.7 initialized

```
bool Ragot::Renderer::initialized = false [private]
```

Flag to indicate if the renderer has been initialized, used to prevent re-initialization and ensure resources are set up correctly.

6.22.4.8 iterations

```
size_t Ragot::Renderer::iterations = 0 [private]
```

Number of iterations for rendering, used for performance testing and optimization.

6.22.4.9 number_of_iterations

```
size_t Ragot::Renderer::number_of_iterations = 10000000000000000 [static], [constexpr], [private]
```

6.22.4.10 quadEBO

```
GLuint Ragot::Renderer::quadEBO = 0 [private]
```

Element [Buffer](#) Object for the full-screen quad, used in non-ESP platforms.

6.22.4.11 quadShader

```
std::unique_ptr< Shader\_Program > Ragot::Renderer::quadShader = nullptr [private]
```

Number of iterations for performance testing, can be adjusted for different scenarios.

[Shader](#) program for rendering a full-screen quad, used in non-ESP platforms.

6.22.4.12 quadVAO

```
GLuint Ragot::Renderer::quadVAO = 0 [private]
```

Vertex Array Object for the full-screen quad, used in non-ESP platforms.

6.22.4.13 quadVBO

```
GLuint Ragot::Renderer::quadVBO = 0 [private]
```

Vertex [Buffer](#) Object for the full-screen quad, used in non-ESP platforms.

6.22.4.14 rasterizer

```
Rasterizer< RGB565 > Ragot::Renderer::rasterizer [private]
```

[Rasterizer](#) for rendering polygons in the frame buffer, responsible for filling polygons with color and handling depth testing.

6.22.4.15 running

```
std::atomic<bool> Ragot::Renderer::running = false [private]
```

Flag to indicate if the renderer is currently running, used to control rendering tasks and stop them gracefully.

6.22.4.16 transformed_vertices

```
std::vector< glm::fvec4 > Ragot::Renderer::transformed_vertices
```

Vector to store transformed vertices for rendering, used to hold the vertices after applying transformations such as model, view, and projection matrices.

6.22.4.17 vertex_shader_code

```
const string Ragot::Renderer::vertex_shader_code [static], [private]
```

Initial value:

```
=
"#version 330\n"
"layout(location = 0) in vec2 aPos;"
"layout(location = 1) in vec2 aTex;"
""
"out vec2 vTex;"
""
"void main() "
"{ "
"    vTex = aTex;"
"    gl_Position = vec4(aPos, 0.0, 1.0);"
"} "
```

Vertex shader code for rendering, used in non-ESP platforms.

6.22.4.18 width

```
unsigned Ragot::Renderer::width [private]
```

Width of the rendering area in pixels, used to define the size of the frame buffer and viewport.

The documentation for this class was generated from the following files:

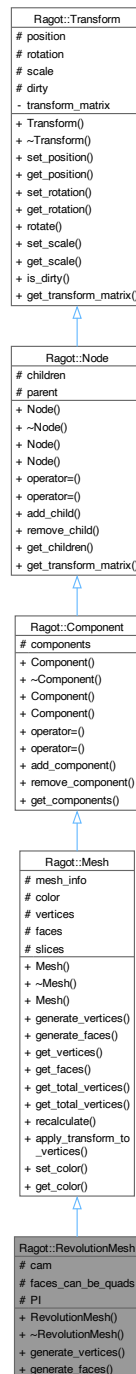
- main/[Renderer.hpp](#)
- main/[Renderer.cpp](#)

6.23 Ragot::RevolutionMesh Class Reference

Class for generating revolution meshes.

```
#include <RevolutionMesh.hpp>
```

Inheritance diagram for Ragot::RevolutionMesh:



The diagram illustrates the architecture of the Ragdoll engine, organized into several functional modules:

- Transform and Node Management:**
 - Ragdoll-Transform** handles transformations with methods like `Transform()`, `set_position()`, `get_position()`, `set_rotation()`, `get_rotation()`, `is_dirty()`, `set_dirty()`, `set_scale()`, `get_scale()`, and `get_transform_matrix()`.
 - Ragdoll-Node** manages a hierarchy of nodes, including `Node()`, `~Node()`, `operator=()`, `operator()`, `add_child()`, `remove_child()`, `get_children()`, and `get_transform_matrix()`.
 - Ragdoll-Component** manages components with methods like `Component()`, `~Component()`, `operator=()`, `operator()`, `add_component()`, `remove_component()`, and `get_components()`.
- Mesh and Geometry:**
 - Ragdoll-Mesh** manages mesh data with methods like `Mesh()`, `~Mesh()`, `generate_vertices()`, `generate_faces()`, `get_vertices()`, `get_faces()`, `get_total_vertices()`, `get_total_faces()`, `recalculate()`, `apply_transform_to_vertices()`, `set_color()`, and `get_color()`.
 - Ragdoll-RevolutionMesh** is a specialized mesh class with methods like `RevolutionMesh()`, `~RevolutionMesh()`, `generate_vertices()`, and `generate_faces()`.
- Camera and Viewport:**
 - Ragdoll-Camera** manages camera parameters with methods like `Camera()`, `~Camera()`, `get_fov()`, `set_near_plane()`, `set_far_plane()`, `set_aspect_ratio()`, `set_location()`, `set_target()`, `is_dirty()`, `set_fov()`, `set_near_plane()`, `set_far_plane()`, `set_aspect_ratio()`, `set_location()`, `set_target()`, `get_projection_matrix()`, `get_view_matrix()`, `get_view_direction()`, `get_right_direction()`, `get_up_direction()`, `project_to_ndc()`, `calculate_normal()`, `is_face_visible()`, and `log_camera_info()`.
- Utility and Constants:**
 - base** and **vec3** classes provide fundamental data types.
 - static const char*** defines camera tags like `CAMERA_TAG`.
 - float** and **int** classes represent numerical values.

Relationships are indicated by directed associations with labels such as `dirty`, `transform_matrix`, `position`, `rotation`, `scale`, `children`, `parent`, `components`, `vertices`, `faces`, `mesh_info`, `projection_matrix`, `view_matrix`, `view_direction`, `right_direction`, `up_direction`, `project_to_ndc`, `calculate_normal`, `is_face_visible`, `log_camera_info`, `aspect_ratio`, `near_plane`, `far_plane`, `target`, `dirty`, `transform_matrix`, `position`, `rotation`, `scale`, `children`, `parent`, `components`, `vertices`, `faces`, `mesh_info`, `projection_matrix`, `view_matrix`, `view_direction`, `right_direction`, `up_direction`, `project_to_ndc`, `calculate_normal`, `is_face_visible`, `log_camera_info`, `aspect_ratio`, `near_plane`, `far_plane`, `target`.

- `RevolutionMesh (mesh_info_t &mesh_info, const Camera &cam)`
Constructor for the `RevolutionMesh` class.
- `~RevolutionMesh ()`=default
Default destructor for the `RevolutionMesh` class.
- `void generate_vertices ()` override
Generates the vertices for the revolution mesh.
- `void generate_faces ()` override
Generates the faces for the revolution mesh.

- `Mesh()`=delete
Construct a new `Mesh` object (deleted constructor).

- virtual `~Mesh()`=default
Default virtual destructor for the `Mesh` class.
- `Mesh(mesh_info_t &mesh_info)`
Construct a new `Mesh` object with mesh information.
- `const std::vector< glm::fvec4 > & get_vertices()` const
Get the vertices object.
- `const std::vector< face_t > & get_faces()` const
Get the faces object.
- `const size_t get_total_vertices()` const
Get the total vertices object.
- `size_t get_total_vertices()`
Get the total vertices object.
- `void recalculate()`
Recalculate the mesh vertices and faces.
- `void apply_transform_to_vertices()`
Apply the current transformation to the vertices of the mesh. This method applies the transformation matrix obtained from the `Transform` class to each vertex in the mesh. This is useful for updating the mesh vertices after any transformation has been applied, such as translation, rotation, or scaling. It modifies the vertices in place, transforming them according to the current transformation matrix.
- `void set_color(uint16_t new_color)`
Set the color of the mesh.
- `uint16_t get_color()` const
Get the color of the mesh.

Public Member Functions inherited from `Ragot::Component`

- `Component()`=default
Default constructor for the `Component` class.
- virtual `~Component()`=default
Default virtual destructor for the `Component` class.
- `Component(const Component &)=delete`
Deleted copy constructor for the `Component` class.
- `Component(const Component &&)=delete`
Deleted move constructor for the `Component` class.
- `Component & operator= (const Component &)=delete`
Deleted assignment operator for the `Component` class.
- `Component & operator= (const Component &&)=delete`
Deleted move assignment operator for the `Component` class.
- `void add_component(std::shared_ptr< Component > component)`
Adds a component to the collection.
- `void remove_component(std::shared_ptr< Component > component)`
Removes a component from the collection.
- `const std::vector< std::shared_ptr< Component > > get_components()` const
Gets the collection of components.

Public Member Functions inherited from Ragot::Node

- [Node](#) ()=default
Default constructor for [Node](#). Initializes an empty node with no parent and no children.
- virtual [~Node](#) ()=default
Default destructor for [Node](#). Cleans up the node and its children.
- [Node](#) (const [Node](#) &)=delete
Deleted copy constructor for [Node](#). Prevents copying of [Node](#) instances.
- [Node](#) (const [Node](#) &&)=delete
Deleted move constructor for [Node](#). Prevents moving of [Node](#) instances.
- [Node](#) & operator= (const [Node](#) &)=delete
Deleted assignment operator for [Node](#). Prevents assignment of [Node](#) instances.
- [Node](#) & operator= (const [Node](#) &&)=delete
Deleted move assignment operator for [Node](#). Prevents moving of [Node](#) instances.
- void [add_child](#) (std::shared_ptr< [Node](#) > child)
Get the parent node.
- void [remove_child](#) (std::shared_ptr< [Node](#) > child)
Remove a child node.
- const std::vector< std::shared_ptr< [Node](#) > > & [get_children](#) () const
Get the parent node.
- mat4 [get_transform_matrix](#) () override
Get the transform matrix object.

Public Member Functions inherited from Ragot::Transform

- [Transform](#) ()
Default constructor for the [Transform](#) class.
- virtual [~Transform](#) ()=default
Virtual destructor for the [Transform](#) class.
- void [set_position](#) (const vec3 &pos)
Sets the position of the object.
- vec3 [get_position](#) () const
Gets the current position of the object.
- void [set_rotation](#) (const vec3 &rot)
Moves the object by a specified vector.
- vec3 [get_rotation](#) () const
Gets the current rotation of the object.
- void [rotate](#) (const float angle, const vec3 &axis)
Rotates the object by a specified angle around a given axis.
- void [set_scale](#) (const vec3 &scale)
Sets the scale of the object.
- vec3 [get_scale](#) () const
Sets the scale of the object uniformly.
- bool [is_dirty](#) () const
Checks if the transformation matrix is dirty (needs recalculation).

Protected Attributes

- const [Camera](#) & [cam](#)
Reference to the camera used for rendering, providing view direction and other properties.
- bool [faces_can_be_quads](#)
Flag indicating whether the faces can be rendered as quads or triangles.

Protected Attributes inherited from [Ragot::Mesh](#)

- [mesh_info_t](#) [mesh_info](#)
Information about the mesh, including coordinates and rendering type.
- [uint16_t](#) [color](#) = 0xFFFF
Color of the mesh, default is white (0xFFFF).
- [std::vector< glm::fvec4 >](#) [vertices](#)
Vector of vertices representing the mesh in 3D space.
- [std::vector< \[face_t\]\(#\) >](#) [faces](#)
Vector of faces representing the mesh, each face can be a triangle or a quad.
- [int](#) [slices](#) = 16
Number of slices for generating the mesh, default is 16.

Protected Attributes inherited from [Ragot::Component](#)

- [std::vector< std::shared_ptr< \[Component\]\(#\) > >](#) [components](#)
Collection of components managed by this [Component](#) instance.

Protected Attributes inherited from [Ragot::Node](#)

- [std::vector< std::shared_ptr< \[Node\]\(#\) > >](#) [children](#)
List of child nodes.
- [Node *](#) [parent](#) = nullptr
Pointer to the parent node.

Protected Attributes inherited from [Ragot::Transform](#)

- [vec3](#) [position](#)
The position of the object in 3D space.
- [vec3](#) [rotation](#)
The rotation of the object in degrees around each axis (x, y, z).
- [vec3](#) [scale](#)
The scale of the object in 3D space, default is (1, 1, 1).
- [bool](#) [dirty](#) = true
Flag indicating whether the transformation matrix needs to be recalculated.

Static Protected Attributes

- [static constexpr float](#) [PI](#) = 3.14159265358979323846f
Constant value for Pi, used in calculations involving angles and rotations.

6.23.1 Detailed Description

Class for generating revolution meshes.

This class generates a mesh by revolving a 2D profile around an axis. It inherits from the [Mesh](#) class and implements the methods to generate vertices and faces.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 RevolutionMesh()

```
Ragot::RevolutionMesh::RevolutionMesh (
    mesh_info_t & mesh_info,
    const Camera & cam) [inline]
```

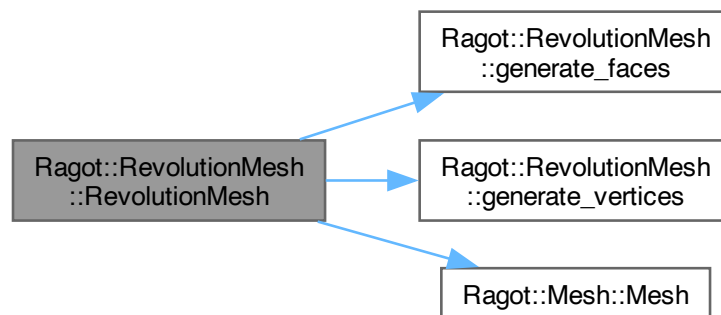
Constructor for the [RevolutionMesh](#) class.

Initializes the mesh with the provided mesh information and camera.

Parameters

<i>mesh_info</i>	Information about the mesh, including coordinates and rendering type.
<i>cam</i>	Reference to the camera used for rendering.

Here is the call graph for this function:



6.23.2.2 ~RevolutionMesh()

```
Ragot::RevolutionMesh::~~RevolutionMesh () [default]
```

Default destructor for the [RevolutionMesh](#) class.

Cleans up resources used by the mesh.

6.23.3 Member Function Documentation

6.23.3.1 generate_faces()

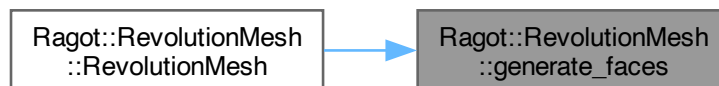
```
void Ragot::RevolutionMesh::generate_faces () [override], [virtual]
```

Generates the faces for the revolution mesh.

This method creates the faces of the mesh by connecting the vertices generated by generate_vertices. It can create either triangles or quads based on the mesh information and the flag faces_can_be_quads.

Implements [Ragot::Mesh](#).

Here is the caller graph for this function:



6.23.3.2 generate_vertices()

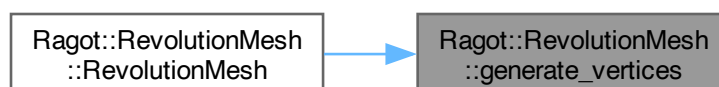
```
void Ragot::RevolutionMesh::generate_vertices () [override], [virtual]
```

Generates the vertices for the revolution mesh.

This method calculates the vertices by revolving the 2D profile around the specified axis. It uses the camera's view direction to determine the local space for the vertices.

Implements [Ragot::Mesh](#).

Here is the caller graph for this function:



6.23.4 Member Data Documentation

6.23.4.1 cam

```
const Camera& Ragot::RevolutionMesh::cam [protected]
```

Reference to the camera used for rendering, providing view direction and other properties.

6.23.4.2 faces_can_be_quads

```
bool Ragot::RevolutionMesh::faces_can_be_quads [protected]
```

Flag indicating whether the faces can be rendered as quads or triangles.

6.23.4.3 PI

```
float Ragot::RevolutionMesh::PI = 3.14159265358979323846f [static], [constexpr], [protected]
```

Constant value for Pi, used in calculations involving angles and rotations.

The documentation for this class was generated from the following files:

- main/[RevolutionMesh.hpp](#)
- main/[RevolutionMesh.cpp](#)

6.24 Ragot::Scene Class Reference

Class for managing a 3D scene.

```
#include <Scene.hpp>
```


- Construct a new [Scene](#) object.
- void [add_node](#) (std::shared_ptr< [Node](#) > node, const [basics::Id](#) name)
 - Adds a node to the scene with a specified name.
- void [remove_node](#) (std::shared_ptr< [Node](#) > node)
 - Removes a node from the scene.
- std::shared_ptr< [Node](#) > [find_node](#) (const [basics::Id](#) name)
 - Finds a node in the scene by its name.
- void [set_main_camera](#) ([Camera](#) *camera)
 - Sets the main camera for the scene.
- [Camera](#) * [get_main_camera](#) () const
 - Gets the main camera of the scene.
- void [traverse](#) (const std::function< void(std::shared_ptr< [Node](#) >) > &callback)
 - Traverses the scene graph and applies a callback function to each node.
- template<typename T>
 - std::vector< std::shared_ptr< T > > [collect_components](#) ()
 - Collects all components of a specified type from the scene.
- void [update](#) (float delta_time)
 - Updates the scene with a specified delta time.
- std::shared_ptr< [Node](#) > [get_root](#) ()
 - Get the root object.
- const std::shared_ptr< [Node](#) > [get_root](#) () const
 - Get the root object.
- void [start](#) ()
 - Starts the scene, setting the running flag to true.
- void [stop](#) ()
 - Stops the scene, setting the running flag to false.

Private Member Functions

- void [task_update](#) (std::stop_token, float delta_time)
 - Task to update the scene in a separate thread.

Private Attributes

- [Camera](#) * [main_camera](#) = nullptr
 - Pointer to the main camera used for rendering the scene.
- std::shared_ptr< [Node](#) > [root_node](#)
 - Shared pointer to the root node of the scene graph.
- std::unordered_map< [basics::Id](#), std::shared_ptr< [Node](#) > > [named_nodes](#)
 - Map of named nodes for quick access by name.
- std::atomic< bool > [running](#) = false
 - Flag indicating whether the scene is currently running or not.

6.24.1 Detailed Description

Class for managing a 3D scene.

The [Scene](#) class provides methods to manage nodes, cameras, and scene traversal. It allows adding and removing nodes, setting the main camera, and traversing the scene graph.

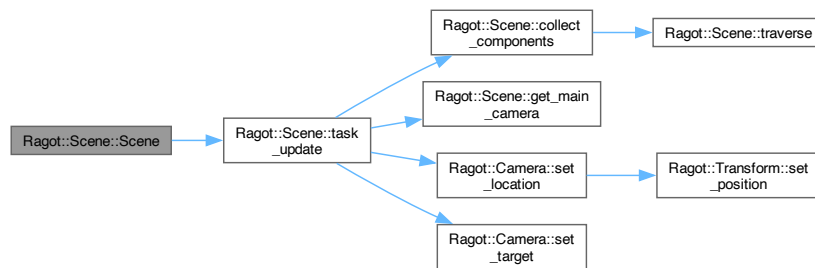
6.24.2 Constructor & Destructor Documentation

6.24.2.1 Scene() [1/2]

```
Ragot::Scene::Scene ()
```

Default constructor for the [Scene](#) class.

Initializes the scene with a root node and prepares it for use. Here is the call graph for this function:



6.24.2.2 ~Scene()

```
Ragot::Scene::~~Scene () [default]
```

Destructor for the [Scene](#) class.

6.24.2.3 Scene() [2/2]

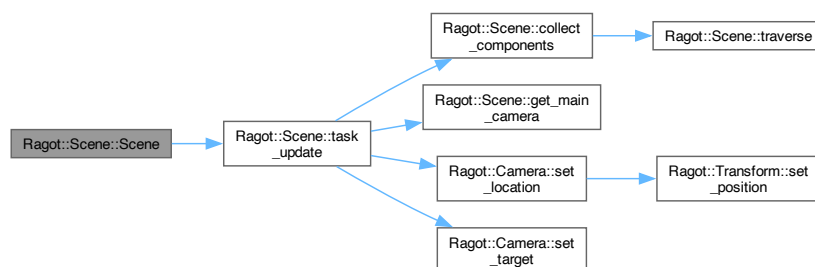
```
Ragot::Scene::Scene (
    Camera * camera)
```

Construct a new [Scene](#) object.

Parameters

<code>camera</code>	Pointer to the Camera object to be used as the main camera for the scene.
---------------------	---

Here is the call graph for this function:



6.24.3 Member Function Documentation

6.24.3.1 add_node()

```
void Ragot::Scene::add_node (
    std::shared_ptr< Node > node,
    const basics::Id name)
```

Adds a node to the scene with a specified name.

Parameters

<i>node</i>	Shared pointer to the Node object to be added to the scene.
<i>name</i>	Unique identifier for the node, used for quick access.

6.24.3.2 collect_components()

```
template<typename T>
template std::vector< std::shared_ptr< Camera > > Ragot::Scene::collect_components< Camera >
()
```

Collects all components of a specified type from the scene.

This method traverses the scene graph and collects all components of the specified type.

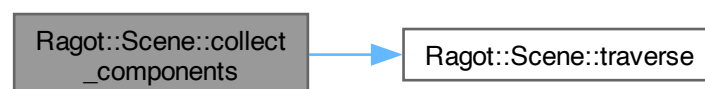
Template Parameters

<i>T</i>	The type of component to collect.
----------	-----------------------------------

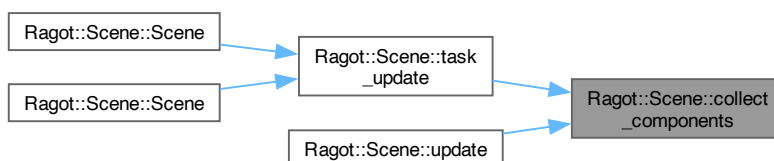
Returns

`std::vector<std::shared_ptr<T>>` A vector containing shared pointers to the collected components.

Here is the call graph for this function:



Here is the caller graph for this function:



6.24.3.3 find_node()

```
std::shared_ptr< Node > Ragot::Scene::find_node (
    const basics::Id name)
```

Finds a node in the scene by its name.

Parameters

<i>name</i>	Unique identifier for the node to be found.
-------------	---

Returns

std::shared_ptr<Node> Shared pointer to the found [Node](#) object, or nullptr if not found.

6.24.3.4 get_main_camera()

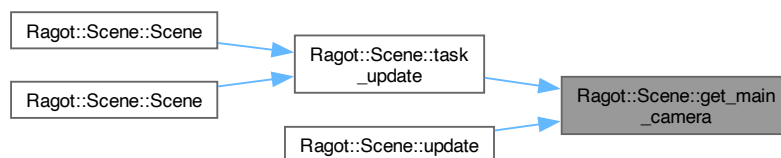
```
Camera * Ragot::Scene::get_main_camera () const [inline]
```

Gets the main camera of the scene.

Returns

Camera* Pointer to the main [Camera](#) object used for rendering the scene.

Here is the caller graph for this function:



6.24.3.5 get_root() [1/2]

```
std::shared_ptr< Node > Ragot::Scene::get_root () [inline]
```

Get the root object.

Returns

std::shared_ptr < [Node](#) >

6.24.3.6 get_root() [2/2]

```
const std::shared_ptr< Node > Ragot::Scene::get_root () const [inline]
```

Get the root object.

Returns

const std::shared_ptr < Node >

6.24.3.7 remove_node()

```
void Ragot::Scene::remove_node (  
    std::shared_ptr< Node > node)
```

Removes a node from the scene.

Parameters

<i>node</i>	Shared pointer to the Node object to be removed from the scene.
-------------	---

6.24.3.8 set_main_camera()

```
void Ragot::Scene::set_main_camera (  
    Camera * camera)
```

Sets the main camera for the scene.

Parameters

<i>camera</i>	Pointer to the Camera object to be set as the main camera for the scene.
---------------	--

6.24.3.9 start()

```
void Ragot::Scene::start () [inline]
```

Starts the scene, setting the running flag to true.

This method is used to indicate that the scene is active and should be updated. Here is the caller graph for this function:



6.24.3.10 stop()

```
void Ragot::Scene::stop () [inline]
```

Stops the scene, setting the running flag to false.

This method is used to indicate that the scene is no longer active and should not be updated.

6.24.3.11 task_update()

```
void Ragot::Scene::task_update (
    std::stop_token stop_token,
    float delta_time) [private]
```

Task to update the scene in a separate thread.

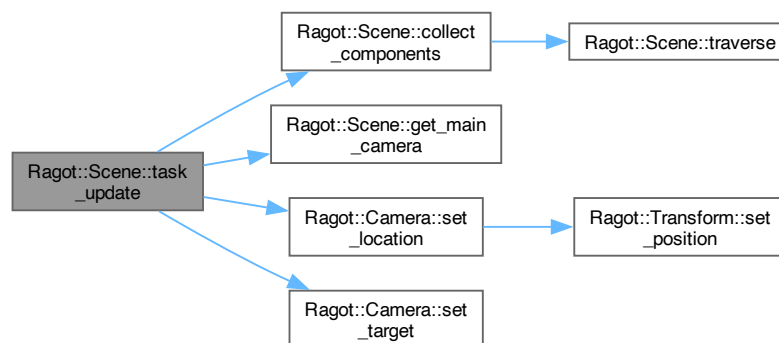
This method runs in a separate thread and updates the scene based on the delta time. It uses a stop token to allow for graceful termination of the task.

Parameters

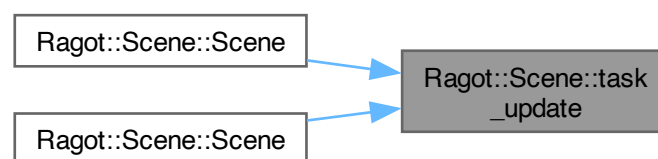
<i>stop_token</i>	Token to signal when the task should stop.
<i>delta_time</i>	The time elapsed since the last update, in seconds.

< Espera inicial para asegurar que la cámara esté lista.

< Espera inicial para asegurar que la cámara esté lista. Here is the call graph for this function:



Here is the caller graph for this function:



6.24.3.12 traverse()

```
void Ragot::Scene::traverse (
    const std::function< void(std::shared_ptr< Node >) > & callback)
```

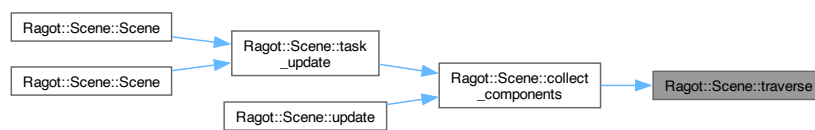
Traverses the scene graph and applies a callback function to each node.

This method allows for custom operations on each node in the scene graph.

Parameters

<i>callback</i>	Function to be called for each node in the scene graph.
-----------------	---

Here is the caller graph for this function:



6.24.3.13 update()

```
void Ragot::Scene::update (
    float delta_time)
```

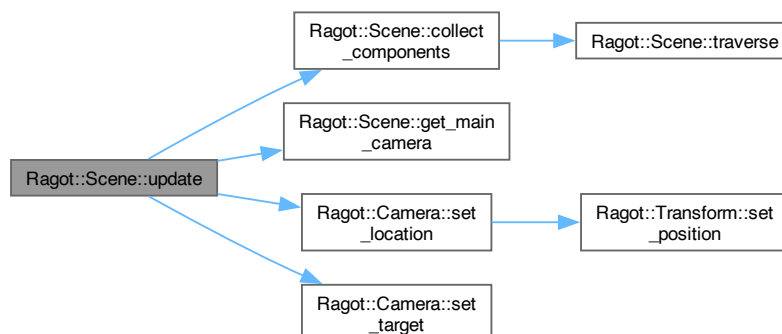
Updates the scene with a specified delta time.

This method updates the scene, allowing for animations or other time-based changes.

Parameters

<i>delta_time</i>	The time elapsed since the last update, in seconds.
-------------------	---

Here is the call graph for this function:



6.24.4 Member Data Documentation

6.24.4.1 main_camera

```
Camera* Ragot::Scene::main_camera = nullptr [private]
```

Pointer to the main camera used for rendering the scene.

6.24.4.2 named_nodes

```
std::unordered_map<basics::Id, std::shared_ptr < Node > > Ragot::Scene::named_nodes [private]
```

Map of named nodes for quick access by name.

6.24.4.3 root_node

```
std::shared_ptr< Node > Ragot::Scene::root_node [private]
```

Shared pointer to the root node of the scene graph.

6.24.4.4 running

```
std::atomic<bool> Ragot::Scene::running = false [private]
```

Flag indicating whether the scene is currently running or not.

The documentation for this class was generated from the following files:

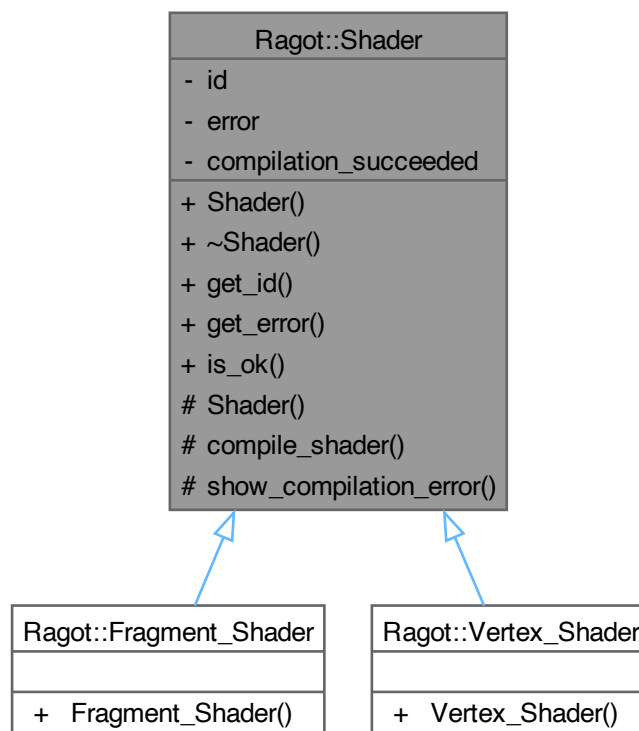
- [main/Scene.hpp](#)
- [main/Scene.cpp](#)

6.25 Ragot::Shader Class Reference

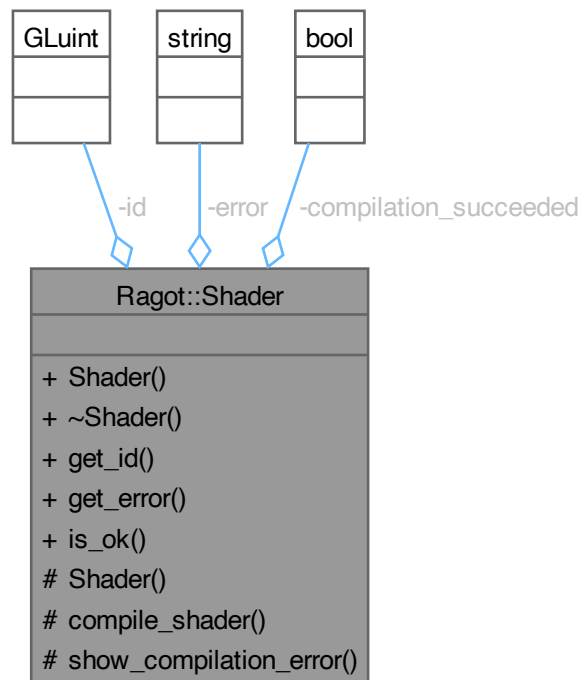
Class for managing an OpenGL shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Shader:



Collaboration diagram for Ragot::Shader:



Public Member Functions

- `Shader ()=delete`
Deleted default constructor.
- `~Shader ()`
Destructor for the `Shader` class.
- `GLuint get_id () const`
Gets the shader ID.
- `string * get_error ()`
Gets the compilation error message.
- `bool is_ok () const`
Checks if the shader is compiled successfully.

Protected Member Functions

- `Shader (const vector< string > &source_code, GLenum type)`
Constructor for the `Shader` class.
- `GLuint compile_shader ()`
Compiles the shader.
- `void show_compilation_error ()`
Displays compilation errors.

Private Attributes

- GLuint [id](#)
Shader ID.
- string [error](#)
Compilation error message.
- bool [compilation_succeeded](#)
Flag indicating if compilation succeeded.

6.25.1 Detailed Description

Class for managing an OpenGL shader.

6.25.2 Constructor & Destructor Documentation**6.25.2.1 Shader() [1/2]**

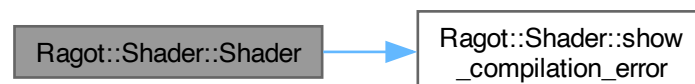
```
Ragot::Shader::Shader (
    const vector< string > & source_code,
    GLenum type) [protected]
```

Constructor for the [Shader](#) class.

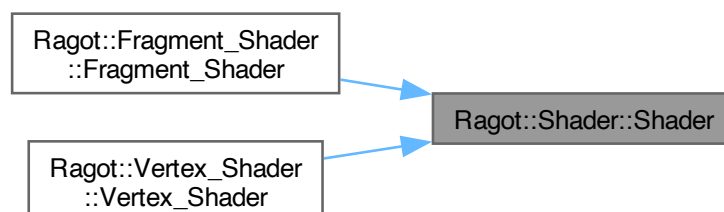
Parameters

<i>source_code</i>	Vector of shader source code.
<i>type</i>	Shader type (e.g., GL_VERTEX_SHADER, GL_FRAGMENT_SHADER).

Here is the call graph for this function:



Here is the caller graph for this function:



6.25.2.2 Shader() [2/2]

```
Ragot::Shader::Shader () [delete]
```

Deleted default constructor.

6.25.2.3 ~Shader()

```
Ragot::Shader::~~Shader () [inline]
```

Destructor for the [Shader](#) class.

6.25.3 Member Function Documentation

6.25.3.1 compile_shader()

```
GLuint Ragot::Shader::compile_shader () [protected]
```

Compiles the shader.

Returns

[Shader](#) ID.

6.25.3.2 get_error()

```
string * Ragot::Shader::get_error () [inline]
```

Gets the compilation error message.

Returns

Pointer to the error message string.

6.25.3.3 get_id()

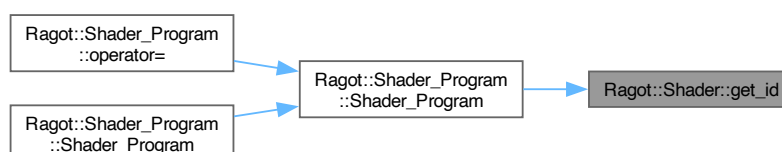
```
GLuint Ragot::Shader::get_id () const [inline]
```

Gets the shader ID.

Returns

[Shader](#) ID.

Here is the caller graph for this function:



6.25.3.4 is_ok()

```
bool Ragot::Shader::is_ok () const [inline]
```

Checks if the shader is compiled successfully.

Returns

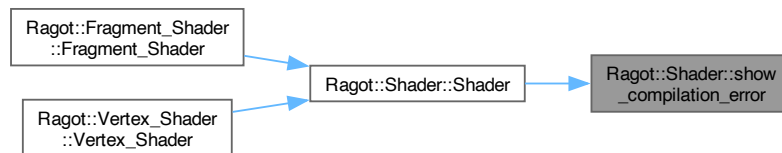
True if compilation succeeded, false otherwise.

6.25.3.5 show_compilation_error()

```
void Ragot::Shader::show_compilation_error () [protected]
```

Displays compilation errors.

Here is the caller graph for this function:



6.25.4 Member Data Documentation

6.25.4.1 compilation_succeeded

```
bool Ragot::Shader::compilation_succeeded [private]
```

Flag indicating if compilation succeeded.

6.25.4.2 error

```
string Ragot::Shader::error [private]
```

Compilation error message.

6.25.4.3 id

```
GLuint Ragot::Shader::id [private]
```

Shader ID.

The documentation for this class was generated from the following files:

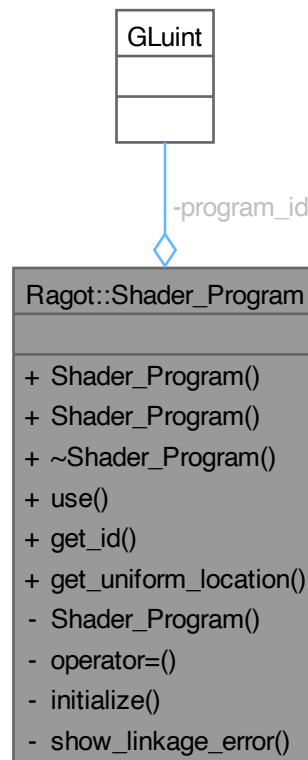
- [main/Shader_Program.hpp](#)
- [main/Shader_Program.cpp](#)

6.26 Ragot::Shader_Program Class Reference

Class for managing an OpenGL shader program.

```
#include <Shader_Program.hpp>
```

Collaboration diagram for Ragot::Shader_Program:



Public Member Functions

- `Shader_Program` (const vector< string > &source_code_vertex, const vector< string > &source_code_↵ fragment)
Constructor for the `Shader_Program` class.
- `Shader_Program` ()=delete
Deleted default constructor.
- `~Shader_Program` ()
Destructor for the `Shader_Program` class.
- void `use` () const
Uses the shader program.
- GLuint `get_id` () const
Gets the shader program ID.
- GLuint `get_uniform_location` (string uniform_name) const
Gets the uniform location in the shader program.

Private Member Functions

- [Shader_Program](#) (const [Shader_Program](#) &)=delete
Deleted copy constructor.
- [Shader_Program](#) & [operator=](#) (const [Shader_Program](#) &)=delete
Deleted copy assignment operator.
- void [initialize](#) (GLuint vertex_shader_id, GLuint fragment_shader_id)
Initializes the shader program.
- void [show_linkage_error](#) ()
Displays linkage errors.

Private Attributes

- GLuint [program_id](#)
Shader program ID.

6.26.1 Detailed Description

Class for managing an OpenGL shader program.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Shader_Program() [1/3]

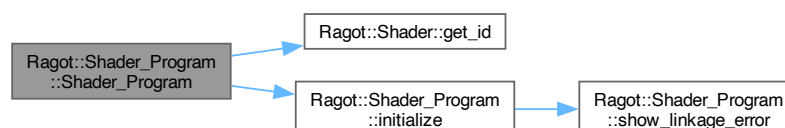
```
Ragot::Shader_Program::Shader_Program (
    const vector< string > & source_code_vertex,
    const vector< string > & source_code_fragment)
```

Constructor for the [Shader_Program](#) class.

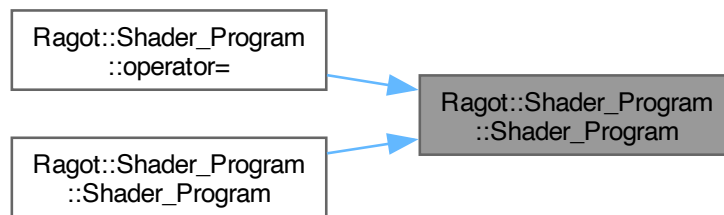
Parameters

<i>source_code_vertex</i>	Vector of vertex shader source code.
<i>source_code_fragment</i>	Vector of fragment shader source code.

Here is the call graph for this function:



Here is the caller graph for this function:



6.26.2.2 Shader_Program() [2/3]

```
Ragot::Shader_Program::Shader_Program () [delete]
```

Deleted default constructor.

6.26.2.3 ~Shader_Program()

```
Ragot::Shader_Program::~~Shader_Program () [inline]
```

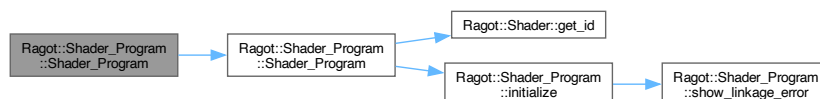
Destructor for the [Shader_Program](#) class.

6.26.2.4 Shader_Program() [3/3]

```
Ragot::Shader_Program::Shader_Program (
    const Shader_Program & ) [private], [delete]
```

Deleted copy constructor.

Here is the call graph for this function:



6.26.3 Member Function Documentation

6.26.3.1 get_id()

```
GLuint Ragot::Shader_Program::get_id () const [inline]
```

Gets the shader program ID.

Returns

Shader program ID.

6.26.3.2 get_uniform_location()

```
GLuint Ragot::Shader_Program::get_uniform_location (
    string uniform_name) const [inline]
```

Gets the uniform location in the shader program.

Parameters

<i>uniform_name</i>	Name of the uniform.
---------------------	----------------------

Returns

Uniform location.

6.26.3.3 initialize()

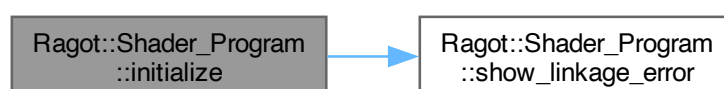
```
void Ragot::Shader_Program::initialize (
    GLuint vertex_shader_id,
    GLuint fragment_shader_id) [private]
```

Initializes the shader program.

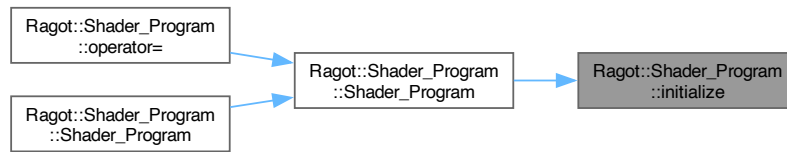
Parameters

<i>vertex_shader_id</i>	Vertex shader ID.
<i>fragment_shader_id</i>	Fragment shader ID.

Here is the call graph for this function:



Here is the caller graph for this function:

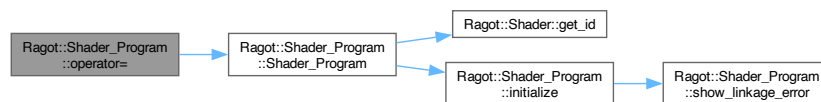


6.26.3.4 operator=()

```
Shader_Program & Ragot::Shader_Program::operator= (
    const Shader_Program & ) [private], [delete]
```

Deleted copy assignment operator.

Here is the call graph for this function:

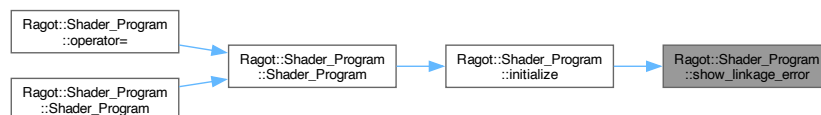


6.26.3.5 show_linkage_error()

```
void Ragot::Shader_Program::show_linkage_error () [private]
```

Displays linkage errors.

Here is the caller graph for this function:



6.26.3.6 use()

```
void Ragot::Shader_Program::use () const [inline]
```

Uses the shader program.

6.26.4 Member Data Documentation

6.26.4.1 program_id

```
GLuint Ragot::Shader_Program::program_id [private]
```

[Shader](#) program ID.

The documentation for this class was generated from the following files:

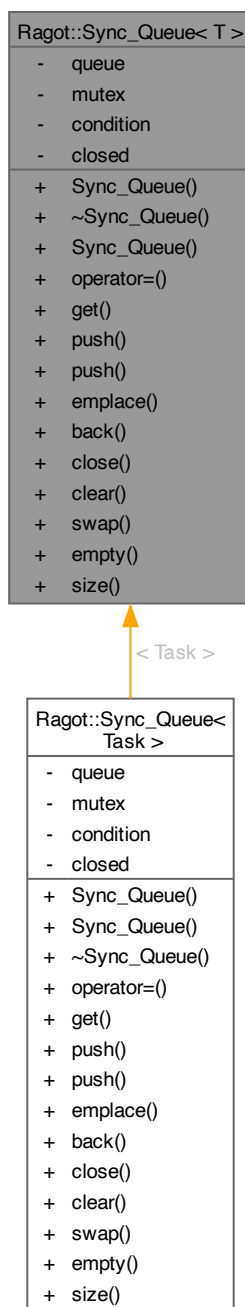
- main/[Shader_Program.hpp](#)
- main/[Shader_Program.cpp](#)

6.27 Ragot::Sync_Queue< T > Class Template Reference

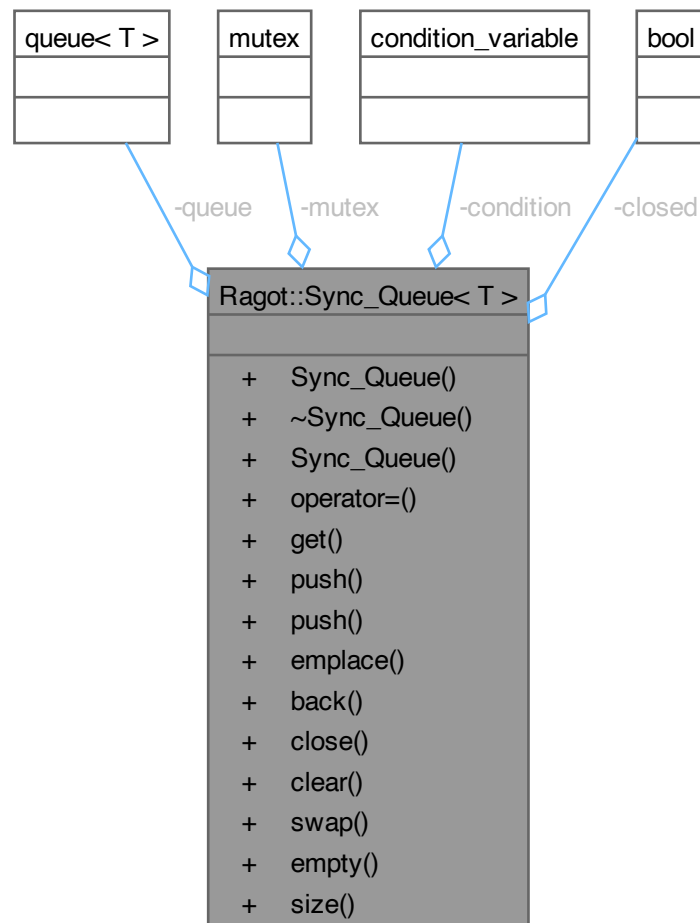
A thread-safe queue implementation.

```
#include <Sync_Queue.hpp>
```

Inheritance diagram for Ragot::Sync_Queue< T >:



Collaboration diagram for Ragot::Sync_Queue< T >:



Public Types

- using [value_type](#) = T

Public Member Functions

- [Sync_Queue](#) ()=default
Construct a new [Sync_Queue](#) object.
- [~Sync_Queue](#) ()
Destroy the [Sync_Queue](#) object.
- [Sync_Queue](#) (const [Sync_Queue](#) &)=delete
Construct a new [Sync_Queue](#) object.
- [Sync_Queue](#) & [operator=](#) (const [Sync_Queue](#) &)=delete
Assignment operator for [Sync_Queue](#).
- std::optional< [value_type](#) > [get](#) ()

- Retrieves an element from the queue.*
 - void `push` (const `value_type` &value)
 - Pushes an element into the queue.*
 - template<typename ... ARGUMENTS>
 - void `push` (ARGUMENTS &&...arguments)
 - Pushes an element into the queue.*
 - template<typename ... ARGUMENTS>
 - void `emplace` (ARGUMENTS &&...arguments)
 - Emplaces an element into the queue.*
 - `value_type` & `back` ()
 - Retrieves the front element of the queue without removing it.*
 - void `close` ()
 - Closes the queue.*
 - void `clear` ()
 - Clears the queue.*
 - void `swap` (`Sync_Queue` &other)
 - Swaps the contents of this queue with another queue.*
 - bool `empty` () const
 - Checks if the queue is empty.*
 - size_t `size` () const
 - Gets the size of the queue.*

Private Attributes

- std::queue< T > `queue`
 - The underlying queue to store elements, used for thread-safe operations.*
- std::mutex `mutex`
 - Mutex to protect access to the queue, ensuring that only one thread can modify the queue at a time.*
- std::condition_variable `condition`
 - Condition variable to notify waiting threads when elements are available in the queue or when the queue is closed.*
- bool `closed` = false
 - Flag to indicate whether the queue is closed, preventing further pushes and allowing threads to exit gracefully when the queue is empty.*

6.27.1 Detailed Description

```
template<typename T>
class Ragot::Sync_Queue< T >
```

A thread-safe queue implementation.

This class provides a synchronized queue that allows multiple threads to safely push and pop elements. It uses mutexes and condition variables to ensure thread safety.

6.27.2 Member Typedef Documentation

6.27.2.1 value_type

```
template<typename T>
using Ragot::Sync_Queue< T >::value_type = T
```

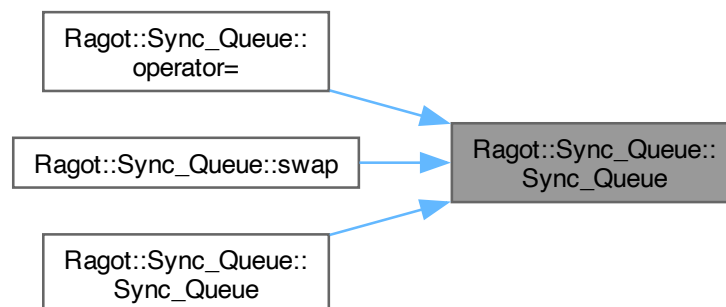
6.27.3 Constructor & Destructor Documentation

6.27.3.1 Sync_Queue() [1/2]

```
template<typename T>  
Ragot::Sync_Queue< T >::Sync_Queue () [default]
```

Construct a new [Sync_Queue](#) object.

Here is the caller graph for this function:



6.27.3.2 ~Sync_Queue()

```
template<typename T>  
Ragot::Sync_Queue< T >::~~Sync_Queue () [inline]
```

Destroy the [Sync_Queue](#) object.

Closes the queue and releases any resources held by it. Here is the call graph for this function:

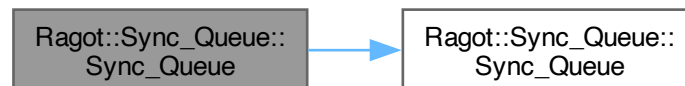


6.27.3.3 Sync_Queue() [2/2]

```
template<typename T>
Ragot::Sync_Queue< T >::Sync_Queue (
    const Sync_Queue< T > & ) [delete]
```

Construct a new [Sync_Queue](#) object.

This constructor is deleted to prevent copying of the [Sync_Queue](#) object. It ensures that the queue cannot be copied, which is important for thread safety. Here is the call graph for this function:



6.27.4 Member Function Documentation

6.27.4.1 back()

```
template<typename T>
value_type & Ragot::Sync_Queue< T >::back () [inline]
```

Retrieves the front element of the queue without removing it.

This method returns a reference to the front element of the queue. It is not thread-safe and should be used with caution.

Returns

[value_type](#)& A reference to the front element of the queue.

6.27.4.2 clear()

```
template<typename T>
void Ragot::Sync_Queue< T >::clear () [inline]
```

Clears the queue.

This method removes all elements from the queue and resets it to an empty state. It is thread-safe and can be called while other threads are accessing the queue. Here is the call graph for this function:

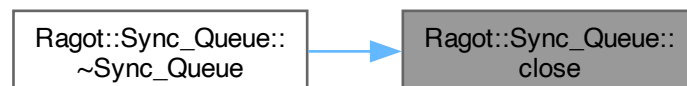


6.27.4.3 close()

```
template<typename T>
void Ragot::Sync_Queue< T >::close () [inline]
```

Closes the queue.

This method sets the closed flag to true and notifies all waiting threads. After closing, no more elements can be pushed into the queue. Here is the caller graph for this function:



6.27.4.4 emplace()

```
template<typename T>
template<typename ... ARGUMENTS>
void Ragot::Sync_Queue< T >::emplace (
    ARGUMENTS &&... arguments) [inline]
```

Emplaces an element into the queue.

This method constructs an element in place and adds it to the queue if it is not closed. It notifies one waiting thread that an element is available.

Template Parameters

<i>ARGUMENTS</i>	The types of arguments used to construct the element.
------------------	---

Parameters

<i>arguments</i>	The arguments used to construct the element.
------------------	--

6.27.4.5 empty()

```
template<typename T>
bool Ragot::Sync_Queue< T >::empty () const [inline]
```

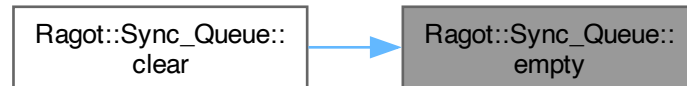
Checks if the queue is empty.

This method checks if the queue is empty by acquiring a lock on the mutex. It returns true if the queue is empty, false otherwise.

Returns

true if the queue is empty, false otherwise.

Here is the caller graph for this function:

**6.27.4.6 get()**

```
template<typename T>
std::optional< value_type > Ragot::Sync_Queue< T >::get () [inline]
```

Retrieves an element from the queue.

This method blocks until an element is available or the queue is closed. If the queue is closed and empty, it returns `std::nullopt`.

Returns

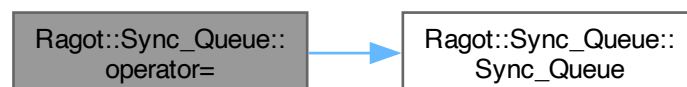
`std::optional<value_type>` The retrieved element or `std::nullopt` if the queue is closed and empty.

6.27.4.7 operator=()

```
template<typename T>
Sync_Queue & Ragot::Sync_Queue< T >::operator= (
    const Sync_Queue< T > & ) [delete]
```

Assignment operator for `Sync_Queue`.

This assignment operator is deleted to prevent copying of the `Sync_Queue` object. It ensures that the queue cannot be assigned, which is important for thread safety. Here is the call graph for this function:



6.27.4.8 push() [1/2]

```
template<typename T>
template<typename ... ARGUMENTS>
void Ragot::Sync_Queue< T >::push (
    ARGUMENTS &&... arguments) [inline]
```

Pushes an element into the queue.

This method adds an element to the queue if it is not closed. It notifies one waiting thread that an element is available.

Parameters

<i>value</i>	The value to be pushed into the queue.
--------------	--

6.27.4.9 push() [2/2]

```
template<typename T>
void Ragot::Sync_Queue< T >::push (
    const value_type & value) [inline]
```

Pushes an element into the queue.

This method adds an element to the queue if it is not closed. It notifies one waiting thread that an element is available.

Parameters

<i>value</i>	The value to be pushed into the queue.
--------------	--

6.27.4.10 size()

```
template<typename T>
size_t Ragot::Sync_Queue< T >::size () const [inline]
```

Gets the size of the queue.

This method returns the number of elements in the queue by acquiring a lock on the mutex. It is thread-safe and can be called while other threads are accessing the queue.

Returns

size_t The number of elements in the queue.

6.27.4.11 swap()

```
template<typename T>
void Ragot::Sync_Queue< T >::swap (
    Sync_Queue< T > & other) [inline]
```

Swaps the contents of this queue with another queue.

This method swaps the contents of this queue with another [Sync_Queue](#). It locks both mutexes to ensure thread safety during the swap operation.

Parameters

<i>other</i>	The other Sync_Queue to swap with.
--------------	--

Here is the call graph for this function:

**6.27.5 Member Data Documentation****6.27.5.1 closed**

```
template<typename T>
bool Ragot::Sync_Queue< T >::closed = false [private]
```

Flag to indicate whether the queue is closed, preventing further pushes and allowing threads to exit gracefully when the queue is empty.

6.27.5.2 condition

```
template<typename T>
std::condition_variable Ragot::Sync_Queue< T >::condition [private]
```

Condition variable to notify waiting threads when elements are available in the queue or when the queue is closed.

6.27.5.3 mutex

```
template<typename T>
std::mutex Ragot::Sync_Queue< T >::mutex [mutable], [private]
```

Mutex to protect access to the queue, ensuring that only one thread can modify the queue at a time.

6.27.5.4 queue

```
template<typename T>
std::queue<T> Ragot::Sync_Queue< T >::queue [private]
```

The underlying queue to store elements, used for thread-safe operations.

The documentation for this class was generated from the following file:

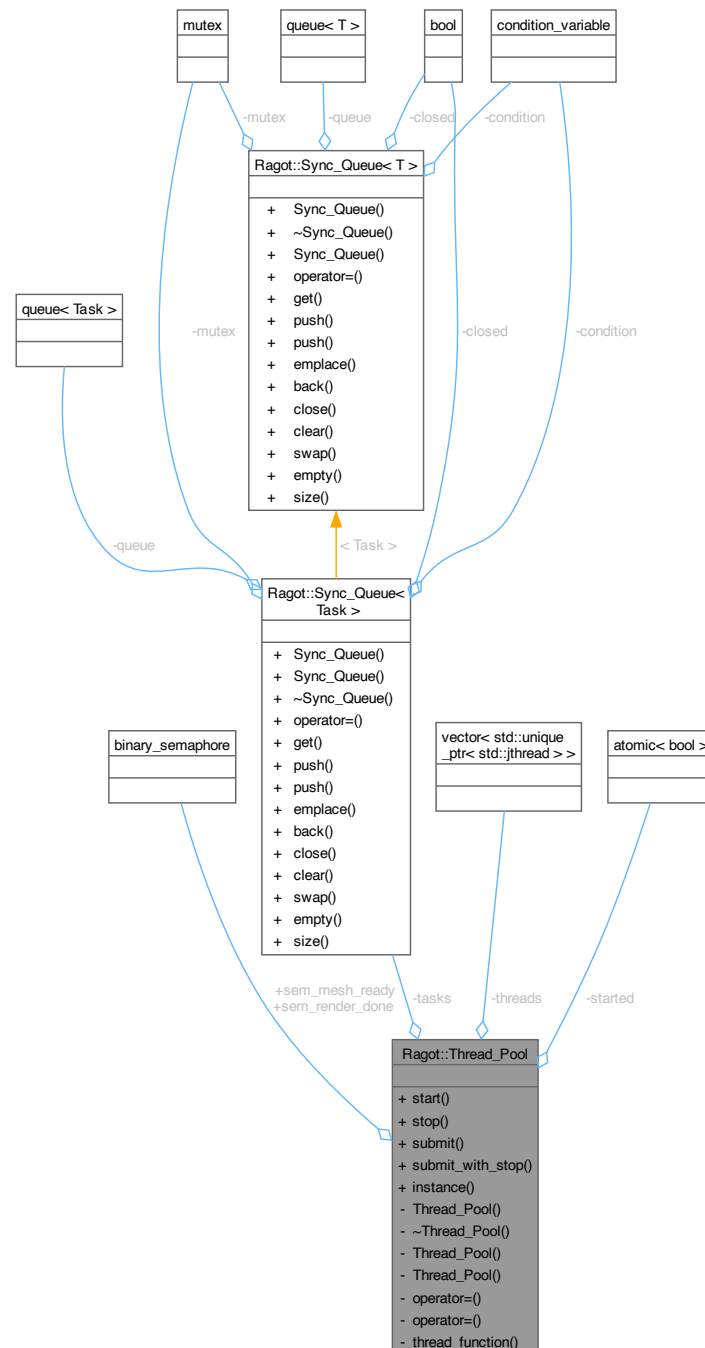
- [main/Sync_Queue.hpp](#)

6.28 Ragot::Thread_Pool Class Reference

A thread pool for managing concurrent tasks.

```
#include <Thread_Pool.hpp>
```

Collaboration diagram for Ragot::Thread_Pool:



Public Types

- using `Task` = `std::function < void (std::stop_token) >`

Public Member Functions

- void [start](#) ()
Starts the thread pool by creating threads and binding them to the `thread_function`.
- void [stop](#) ()
Stops the thread pool by closing the task queue and requesting all threads to stop.
- template<typename F, typename... Args>
std::future< std::invoke_result_t< F, Args... > > [submit](#) (F &&f, Args &&... args)
Submits a task to the thread pool for execution.
- template<typename F, typename... Args>
std::future< std::invoke_result_t< F, std::stop_token, Args... > > [submit_with_stop](#) (F &&f, Args &&... args)
Submits a task to the thread pool for execution with a stop token.

Static Public Member Functions

- static [Thread_Pool](#) & [instance](#) ()
Singleton instance of the [Thread_Pool](#) class.

Public Attributes

- std::binary_semaphore [sem_mesh_ready](#) {0}
Semaphore to signal that the mesh is ready for rendering.
- std::binary_semaphore [sem_render_done](#) {0}
Semaphore to signal that the rendering is done.

Private Member Functions

- [Thread_Pool](#) ()
Private constructor for the [Thread_Pool](#) class.
- [~Thread_Pool](#) ()
Destructor for the [Thread_Pool](#) class.
- [Thread_Pool](#) ([Thread_Pool](#) &)=delete
Construct a new [Thread_Pool](#) object (Deleted). This constructor is deleted to prevent copying of the [Thread_Pool](#) object.
- [Thread_Pool](#) ([Thread_Pool](#) &&)=delete
Move constructor for the [Thread_Pool](#) class (Deleted). This move constructor is deleted to prevent moving of the [Thread_Pool](#) object.
- [Thread_Pool](#) & operator= ([Thread_Pool](#) &)=delete
Assignment operator for the [Thread_Pool](#) class (Deleted). This assignment operator is deleted to prevent copying of the [Thread_Pool](#) object.
- [Thread_Pool](#) & operator= ([Thread_Pool](#) &&)=delete
Move assignment operator for the [Thread_Pool](#) class (Deleted). This move assignment operator is deleted to prevent moving of the [Thread_Pool](#) object.
- void [thread_function](#) (std::stop_token)
The function executed by each thread in the thread pool.

Private Attributes

- [Sync_Queue< Task > tasks](#)
A synchronized queue for managing tasks in the thread pool.
- `std::vector< std::unique_ptr< std::jthread > >` [threads](#)
A vector of unique pointers to jthread objects representing the threads in the pool.
- `std::atomic< bool >` [started](#)
Flag indicating whether the thread pool has been started or not.

6.28.1 Detailed Description

A thread pool for managing concurrent tasks.

This class provides a thread pool that can execute tasks concurrently using a specified number of threads. It allows submitting tasks and handles synchronization using semaphores.

6.28.2 Member Typedef Documentation

6.28.2.1 Task

```
using Ragot::Thread_Pool::Task = std::function < void (std::stop_token) >
```

6.28.3 Constructor & Destructor Documentation

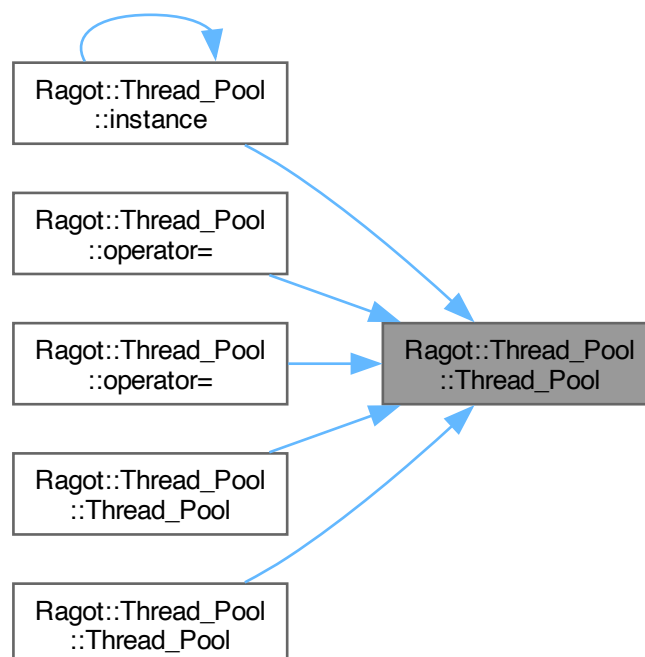
6.28.3.1 Thread_Pool() [1/3]

```
Ragot::Thread_Pool::Thread_Pool () [inline], [private]
```

Private constructor for the [Thread_Pool](#) class.

Initializes the thread pool with the number of threads equal to the number of hardware cores available. If the number

of cores is zero, it defaults to two threads. Here is the caller graph for this function:



6.28.3.2 `~Thread_Pool()`

```
Ragot::Thread_Pool::~~Thread_Pool () [inline], [private]
```

Destructor for the [Thread_Pool](#) class.

Cleans up resources used by the thread pool, releases semaphores, and stops all threads if they are running. Here is the call graph for this function:

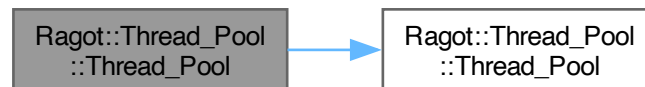


6.28.3.3 Thread_Pool() [2/3]

```
Ragot::Thread_Pool::Thread_Pool (  
    Thread_Pool & ) [private], [delete]
```

Construct a new [Thread_Pool](#) object (Deleted). This constructor is deleted to prevent copying of the [Thread_Pool](#) object.

Here is the call graph for this function:

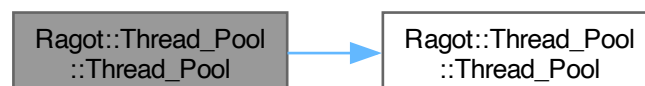


6.28.3.4 Thread_Pool() [3/3]

```
Ragot::Thread_Pool::Thread_Pool (  
    Thread_Pool && ) [private], [delete]
```

Move constructor for the [Thread_Pool](#) class (Deleted). This move constructor is deleted to prevent moving of the [Thread_Pool](#) object.

Here is the call graph for this function:



6.28.4 Member Function Documentation

6.28.4.1 instance()

```
static Thread_Pool & Ragot::Thread_Pool::instance () [inline], [static]
```

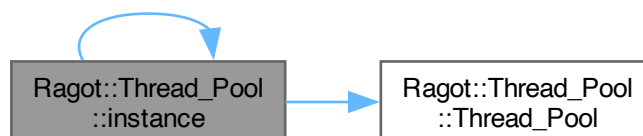
Singleton instance of the [Thread_Pool](#) class.

This method provides access to the singleton instance of the [Thread_Pool](#). It ensures that only one instance of the thread pool exists throughout the application.

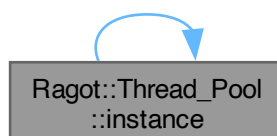
Returns

[Thread_Pool](#)& Reference to the singleton instance of the [Thread_Pool](#).

Here is the call graph for this function:



Here is the caller graph for this function:



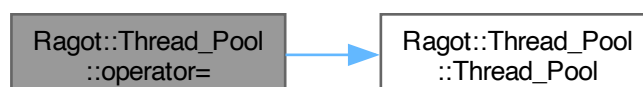
6.28.4.2 operator=() [1/2]

```

Thread_Pool & Ragot::Thread_Pool::operator= (
    Thread_Pool && ) [private], [delete]
  
```

Move assignment operator for the [Thread_Pool](#) class (Deleted). This move assignment operator is deleted to prevent moving of the [Thread_Pool](#) object.

Here is the call graph for this function:

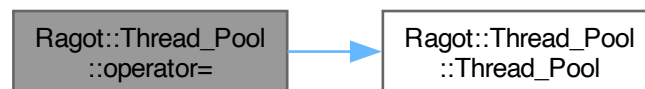


6.28.4.3 operator=() [2/2]

```
Thread_Pool & Ragot::Thread_Pool::operator= (  
    Thread_Pool & ) [private], [delete]
```

Assignment operator for the [Thread_Pool](#) class (Deleted). This assignment operator is deleted to prevent copying of the [Thread_Pool](#) object.

Here is the call graph for this function:

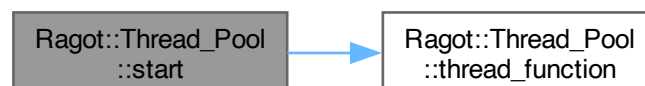


6.28.4.4 start()

```
void Ragot::Thread_Pool::start () [inline]
```

Starts the thread pool by creating threads and binding them to the `thread_function`.

This method initializes the threads in the pool and starts them, allowing them to execute tasks concurrently. It asserts that the thread pool has not already been started to prevent multiple initializations. Here is the call graph for this function:

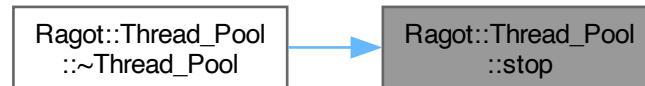


6.28.4.5 stop()

```
void Ragot::Thread_Pool::stop () [inline]
```

Stops the thread pool by closing the task queue and requesting all threads to stop.

This method stops the thread pool, ensuring that all threads are requested to stop gracefully. It asserts that the thread pool has been started before attempting to stop it. Here is the caller graph for this function:



6.28.4.6 submit()

```

template<typename F, typename... Args>
std::future< std::invoke_result_t< F, Args... > > Ragot::Thread_Pool::submit (
    F && f,
    Args &&... args) [inline]
  
```

Submits a task to the thread pool for execution.

This method allows submitting a task to the thread pool, which will be executed by one of the threads. It returns a future that can be used to retrieve the result of the task once it is completed.

Template Parameters

<i>F</i>	The type of the function to be executed.
<i>Args</i>	The types of the arguments to be passed to the function.

Parameters

<i>f</i>	The function to be executed.
<i>args</i>	The arguments to be passed to the function.

Returns

`std::future<std::invoke_result_t<F, Args...>>` A future representing the result of the task.

6.28.4.7 submit_with_stop()

```

template<typename F, typename... Args>
std::future< std::invoke_result_t< F, std::stop_token, Args... > > Ragot::Thread_Pool↵
::submit_with_stop (
    F && f,
    Args &&... args) [inline]
  
```

Submits a task to the thread pool for execution with a stop token.

This method allows submitting a task to the thread pool, which will be executed by one of the threads. It accepts a stop token that can be used to request cancellation of the task. It returns a future that can be used to retrieve the result of the task once it is completed.

Template Parameters

<i>F</i>	The type of the function to be executed.
<i>Args</i>	The types of the arguments to be passed to the function.

Parameters

<i>f</i>	The function to be executed.
<i>args</i>	The arguments to be passed to the function.

Returns

`std::future<std::invoke_result_t<F, std::stop_token, Args...>>` A future representing the result of the task.

6.28.4.8 thread_function()

```
void Ragot::Thread_Pool::thread_function (
    std::stop_token stop_token) [private]
```

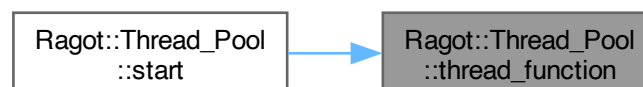
The function executed by each thread in the thread pool.

This function continuously retrieves tasks from the task queue and executes them. It uses a stop token to allow threads to gracefully exit when requested.

Parameters

<i>stop_token</i>	The stop token used to request cancellation of the task.
-------------------	--

Here is the caller graph for this function:

**6.28.5 Member Data Documentation****6.28.5.1 sem_mesh_ready**

```
std::binary_semaphore Ragot::Thread_Pool::sem_mesh_ready {0}
```

Semaphore to signal that the mesh is ready for rendering.

6.28.5.2 sem_render_done

```
std::binary_semaphore Ragot::Thread_Pool::sem_render_done {0}
```

Semaphore to signal that the rendering is done.

6.28.5.3 started

```
std::atomic< bool > Ragot::Thread_Pool::started [private]
```

Flag indicating whether the thread pool has been started or not.

6.28.5.4 tasks

```
Sync_Queue< Task > Ragot::Thread_Pool::tasks [private]
```

A synchronized queue for managing tasks in the thread pool.

This queue is used to store tasks that need to be executed by the threads in the pool. It provides thread-safe operations for pushing and popping tasks.

6.28.5.5 threads

```
std::vector< std::unique_ptr < std::jthread > > Ragot::Thread_Pool::threads [private]
```

A vector of unique pointers to jthread objects representing the threads in the pool.

This vector holds the threads that will execute tasks concurrently. Each thread is managed by a unique pointer to ensure proper resource management.

The documentation for this class was generated from the following files:

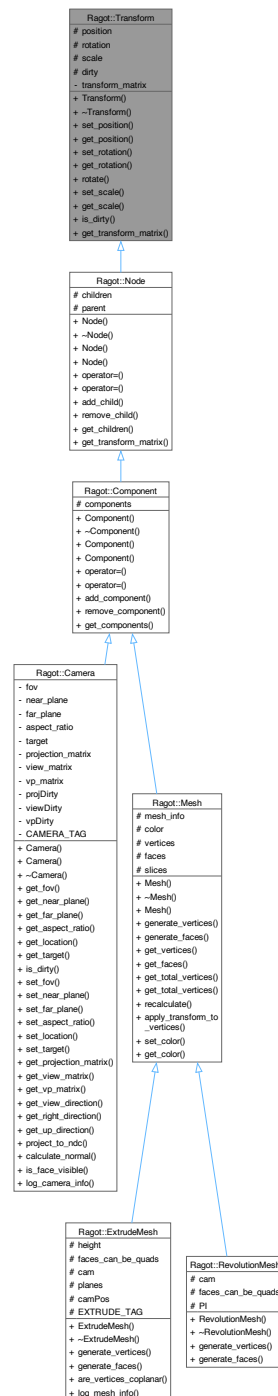
- [main/Thread_Pool.hpp](#)
- [main/Thread_Pool.cpp](#)

6.29 Ragot::Transform Class Reference

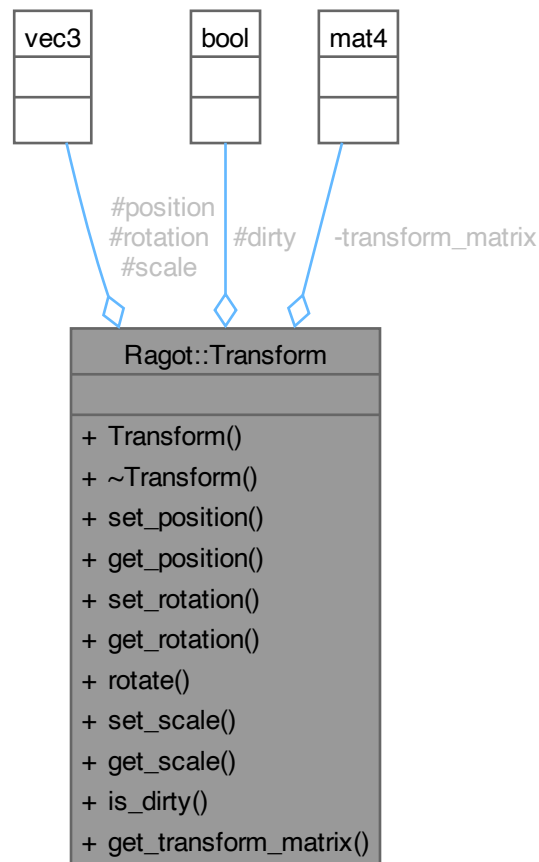
A class representing a 3D transformation with position, rotation, and scale.

```
#include <Transform.hpp>
```

Inheritance diagram for Ragot::Transform:



Collaboration diagram for Ragot::Transform:



Public Member Functions

- [Transform](#) ()
Default constructor for the [Transform](#) class.
- virtual [~Transform](#) ()=default
Virtual destructor for the [Transform](#) class.
- void [set_position](#) (const vec3 &pos)
Sets the position of the object.
- vec3 [get_position](#) () const
Gets the current position of the object.
- void [set_rotation](#) (const vec3 &rot)
Moves the object by a specified vector.
- vec3 [get_rotation](#) () const
Gets the current rotation of the object.
- void [rotate](#) (const float angle, const vec3 &axis)
Rotates the object by a specified angle around a given axis.
- void [set_scale](#) (const vec3 &scale)

- Sets the scale of the object.*
 - `vec3 get_scale () const`
 - Sets the scale of the object uniformly.*
 - `bool is_dirty () const`
 - Checks if the transformation matrix is dirty (needs recalculation).*
 - `virtual mat4 get_transform_matrix ()`
 - Gets the transformation matrix that combines position, rotation, and scale.*

Protected Attributes

- `vec3 position`
 - The position of the object in 3D space.*
- `vec3 rotation`
 - The rotation of the object in degrees around each axis (x, y, z).*
- `vec3 scale`
 - The scale of the object in 3D space, default is (1, 1, 1).*
- `bool dirty = true`
 - Flag indicating whether the transformation matrix needs to be recalculated.*

Private Attributes

- `mat4 transform_matrix`
 - The transformation matrix that combines position, rotation, and scale.*

6.29.1 Detailed Description

A class representing a 3D transformation with position, rotation, and scale.

This class provides methods to set and get the position, rotation, and scale of a 3D object. It also provides a method to get the transformation matrix that combines these transformations.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Transform()

```
Ragot::Transform::Transform () [inline]
```

Default constructor for the [Transform](#) class.

Initializes position to (0, 0, 0), rotation to (0, 0, 0), and scale to (1, 1, 1).

6.29.2.2 ~Transform()

```
virtual Ragot::Transform::~~Transform () [virtual], [default]
```

Virtual destructor for the [Transform](#) class.

Ensures proper cleanup of derived classes.

6.29.3 Member Function Documentation

6.29.3.1 get_position()

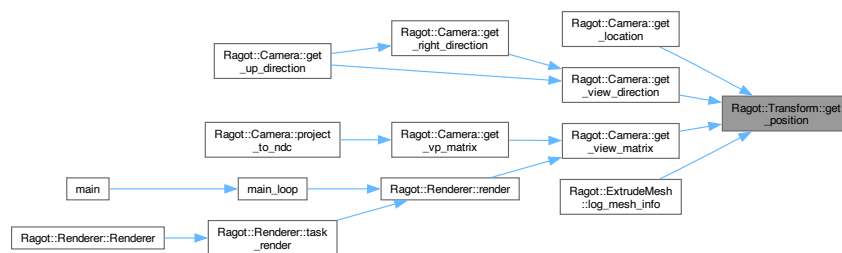
```
vec3 Ragot::Transform::get_position () const [inline]
```

Gets the current position of the object.

Returns

vec3 The current position.

Here is the caller graph for this function:



6.29.3.2 get_rotation()

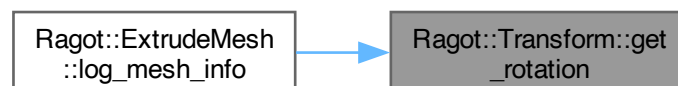
```
vec3 Ragot::Transform::get_rotation () const [inline]
```

Gets the current rotation of the object.

Returns

vec3 The current rotation in degrees around each axis (x, y, z).

Here is the caller graph for this function:



6.29.3.3 get_scale()

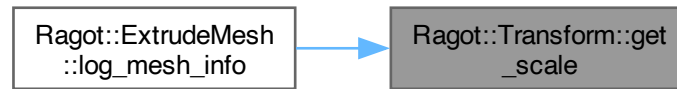
```
vec3 Ragot::Transform::get_scale () const [inline]
```

Sets the scale of the object uniformly.

Parameters

<i>scale</i>	The new uniform scale factor.
--------------	-------------------------------

Here is the caller graph for this function:

6.29.3.4 `get_transform_matrix()`

```
virtual mat4 Ragot::Transform::get_transform_matrix () [inline], [virtual]
```

Gets the transformation matrix that combines position, rotation, and scale.

This method recalculates the transformation matrix if it is dirty.

Returns

mat4 The transformation matrix.

Reimplemented in [Ragot::Node](#).

Here is the caller graph for this function:

6.29.3.5 `is_dirty()`

```
bool Ragot::Transform::is_dirty () const [inline]
```

Checks if the transformation matrix is dirty (needs recalculation).

Returns

true if the transformation matrix is dirty, false otherwise.

6.29.3.6 `rotate()`

```
void Ragot::Transform::rotate (
    const float angle,
    const vec3 & axis) [inline]
```

Rotates the object by a specified angle around a given axis.

Parameters

<i>angle</i>	The angle in degrees to rotate.
<i>axis</i>	The axis around which to rotate, as a vec3.

6.29.3.7 set_position()

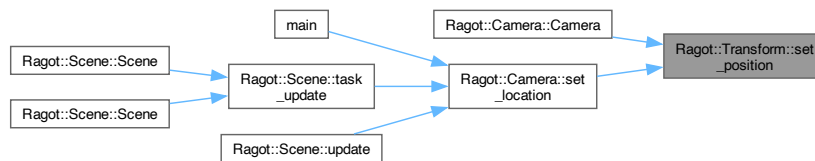
```
void Ragot::Transform::set_position (
    const vec3 & pos) [inline]
```

Sets the position of the object.

Parameters

<i>pos</i>	The new position as a vec3.
------------	-----------------------------

Here is the caller graph for this function:

**6.29.3.8 set_rotation()**

```
void Ragot::Transform::set_rotation (
    const vec3 & rot) [inline]
```

Moves the object by a specified vector.

Parameters

<i>offset</i>	The vector by which to move the object.
---------------	---

6.29.3.9 set_scale()

```
void Ragot::Transform::set_scale (
    const vec3 & scale) [inline]
```

Sets the scale of the object.

Parameters

<i>scale</i>	The new scale as a vec3.
--------------	--------------------------

6.29.4 Member Data Documentation

6.29.4.1 dirty

```
bool Ragot::Transform::dirty = true [protected]
```

Flag indicating whether the transformation matrix needs to be recalculated.

6.29.4.2 position

```
vec3 Ragot::Transform::position [protected]
```

The position of the object in 3D space.

6.29.4.3 rotation

```
vec3 Ragot::Transform::rotation [protected]
```

The rotation of the object in degrees around each axis (x, y, z).

6.29.4.4 scale

```
vec3 Ragot::Transform::scale [protected]
```

The scale of the object in 3D space, default is (1, 1, 1).

6.29.4.5 transform_matrix

```
mat4 Ragot::Transform::transform_matrix [private]
```

The transformation matrix that combines position, rotation, and scale.

The documentation for this class was generated from the following file:

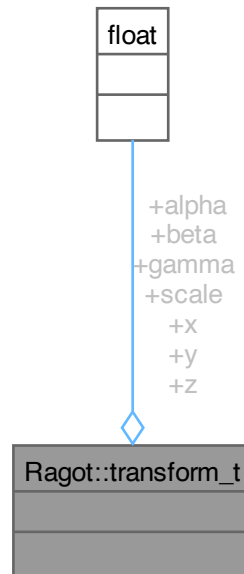
- [main/Transform.hpp](#)

6.30 Ragot::transform_t Struct Reference

Represents a transformation in 3D space.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::transform_t:



Public Attributes

- float `x`
- float `y`
- float `z`
- float `alpha`
- float `beta`
- float `gamma`
- float `scale`

6.30.1 Detailed Description

Represents a transformation in 3D space.

This structure holds the position, rotation (in Euler angles), and scale of an object in 3D space.

6.30.2 Member Data Documentation

6.30.2.1 alpha

`float Ragot::transform_t::alpha`

6.30.2.2 beta

`float Ragot::transform_t::beta`

6.30.2.3 gamma

`float Ragot::transform_t::gamma`

6.30.2.4 scale

`float Ragot::transform_t::scale`

6.30.2.5 x

`float Ragot::transform_t::x`

6.30.2.6 y

`float Ragot::transform_t::y`

6.30.2.7 z

`float Ragot::transform_t::z`

The documentation for this struct was generated from the following file:

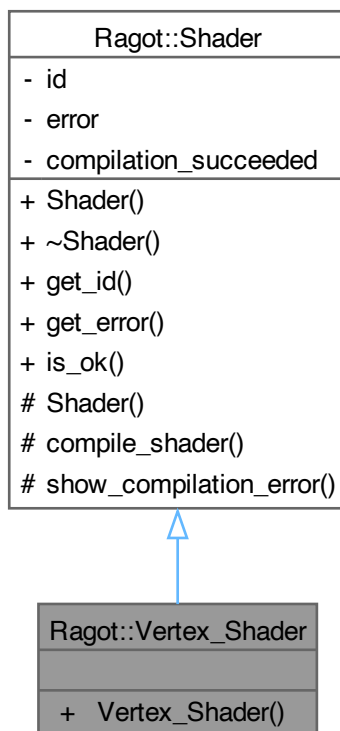
- [main/CommonTypes.hpp](#)

6.31 Ragot::Vertex_Shader Class Reference

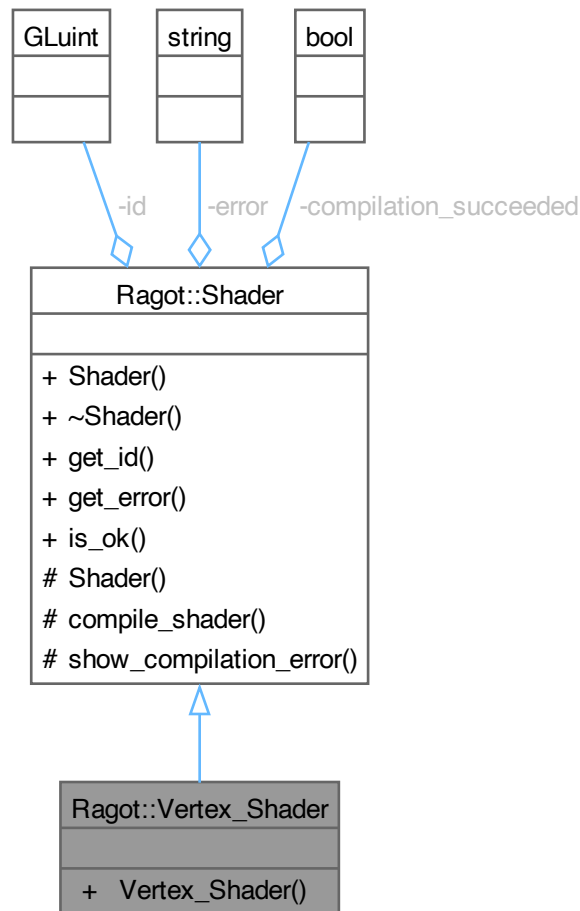
Class for managing an OpenGL vertex shader.

```
#include <Shader_Program.hpp>
```

Inheritance diagram for Ragot::Vertex_Shader:



Collaboration diagram for Ragot::Vertex_Shader:



Public Member Functions

- [Vertex_Shader](#) (const vector< string > &source_code)
Constructor for the [Vertex_Shader](#) class.

Public Member Functions inherited from [Ragot::Shader](#)

- [Shader](#) ()=delete
Deleted default constructor.
- [~Shader](#) ()
Destructor for the [Shader](#) class.
- GLuint [get_id](#) () const
Gets the shader ID.
- string * [get_error](#) ()
Gets the compilation error message.
- bool [is_ok](#) () const
Checks if the shader is compiled successfully.

Additional Inherited Members

Protected Member Functions inherited from [Ragot::Shader](#)

- [Shader](#) (const vector< string > &source_code, GLenum type)
Constructor for the [Shader](#) class.
- GLuint [compile_shader](#) ()
Compiles the shader.
- void [show_compilation_error](#) ()
Displays compilation errors.

6.31.1 Detailed Description

Class for managing an OpenGL vertex shader.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 Vertex_Shader()

```
Ragot::Vertex_Shader::Vertex_Shader (
    const vector< string > & source_code) [inline]
```

Constructor for the [Vertex_Shader](#) class.

Parameters

<i>source_code</i>	Vector of vertex shader source code.
--------------------	--------------------------------------

Here is the call graph for this function:



The documentation for this class was generated from the following file:

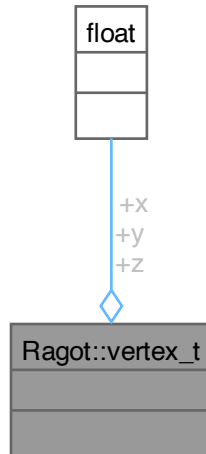
- main/[Shader_Program.hpp](#)

6.32 Ragot::vertex_t Struct Reference

Represents a vertex in 3D space.

```
#include <CommonTypes.hpp>
```

Collaboration diagram for Ragot::vertex_t:



Public Attributes

- float `x`
- float `y`
- float `z`

6.32.1 Detailed Description

Represents a vertex in 3D space.

This structure holds the x, y, and z coordinates of a vertex.

6.32.2 Member Data Documentation

6.32.2.1 x

```
float Ragot::vertex_t::x
```

6.32.2.2 y

```
float Ragot::vertex_t::y
```

6.32.2.3 z

```
float Ragot::vertex_t::z
```

The documentation for this struct was generated from the following file:

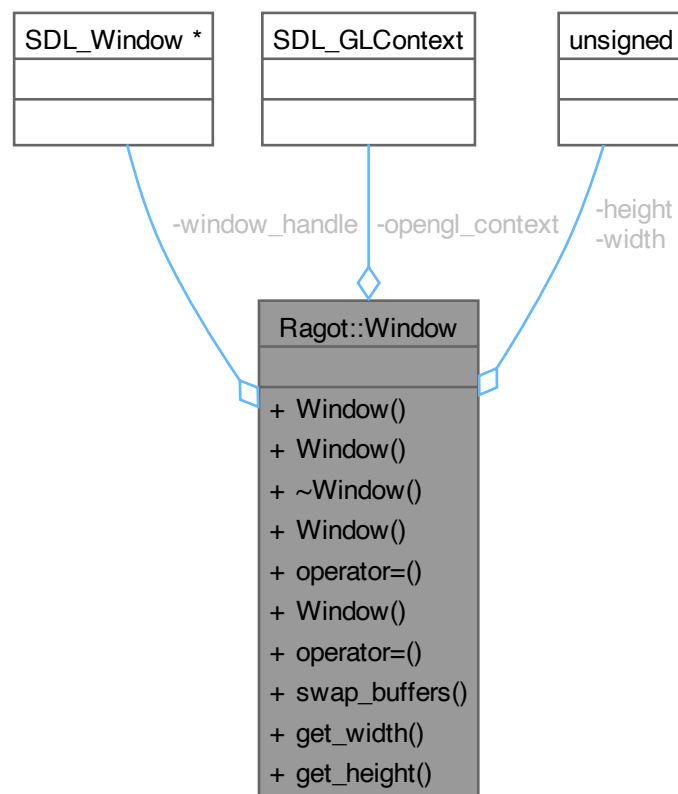
- main/[CommonTypes.hpp](#)

6.33 Ragot::Window Class Reference

Class for managing an SDL window with OpenGL context.

```
#include <Window.hpp>
```

Collaboration diagram for Ragot::Window:



Classes

- struct [OpenGL_Context_Settings](#)
Struct for OpenGL context settings.

Public Types

- enum [Position](#) { [UNDEFINED](#) = SDL_WINDOWPOS_UNDEFINED , [CENTERED](#) = SDL_WINDOWPOS_↵
CENTERED }

Enum for window position.

Public Member Functions

- [Window](#) (const std::string &title, int left_x, int top_y, unsigned [width](#), unsigned [height](#), const [OpenGL_Context_Settings](#) &context_details)
Constructor for the [Window](#) class.
- [Window](#) (const char *title, int left_x, int top_y, unsigned [width](#), unsigned [height](#), const [OpenGL_Context_Settings](#) &context_details)
Constructor for the [Window](#) class.
- [~Window](#) ()
Destructor for the [Window](#) class.
- [Window](#) (const [Window](#) &)=delete
Deleted copy constructor.
- [Window](#) & [operator=](#) (const [Window](#) &)=delete
Deleted copy assignment operator.
- [Window](#) ([Window](#) &&other) noexcept
Move constructor for the [Window](#) class.
- [Window](#) & [operator=](#) ([Window](#) &&other) noexcept
Move assignment operator for the [Window](#) class.
- void [swap_buffers](#) ()
Swaps the OpenGL buffers.
- unsigned [get_width](#) ()
Gets the width of the window.
- unsigned [get_height](#) ()
Gets the height of the window.

Private Attributes

- SDL_Window * [window_handle](#)
Handle to the SDL window.
- SDL_GLContext [opengl_context](#)
OpenGL context.
- unsigned [width](#)
Width of the window.
- unsigned [height](#)
Height of the window.

6.33.1 Detailed Description

Class for managing an SDL window with OpenGL context.

6.33.2 Member Enumeration Documentation

6.33.2.1 Position

```
enum Ragot::Window::Position
```

Enum for window position.

Enumerator

UNDEFINED	Undefined position.
CENTERED	Centered position.

6.33.3 Constructor & Destructor Documentation

6.33.3.1 Window() [1/4]

```
Ragot::Window::Window (
    const std::string & title,
    int left_x,
    int top_y,
    unsigned width,
    unsigned height,
    const OpenGL_Context_Settings & context_details) [inline]
```

Constructor for the [Window](#) class.

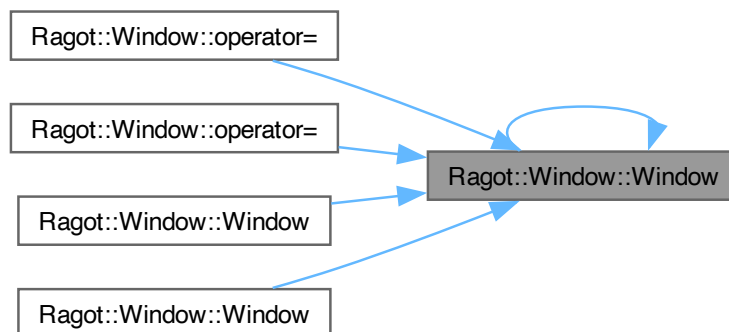
Parameters

<i>title</i>	Title of the window.
<i>left_x</i>	X coordinate of the window position.
<i>top_y</i>	Y coordinate of the window position.
<i>width</i>	Width of the window.
<i>height</i>	Height of the window.
<i>context_details</i>	OpenGL context settings.

Here is the call graph for this function:



Here is the caller graph for this function:



6.33.3.2 Window() [2/4]

```

Ragot::Window::Window (
    const char * title,
    int left_x,
    int top_y,
    unsigned width,
    unsigned height,
    const OpenGL_Context_Settings & context_details)

```

Constructor for the [Window](#) class.

Parameters

<i>title</i>	Title of the window.
<i>left_x</i>	X coordinate of the window position.
<i>top_y</i>	Y coordinate of the window position.
<i>width</i>	Width of the window.
<i>height</i>	Height of the window.
<i>context_details</i>	OpenGL context settings.

6.33.3.3 ~Window()

```

Ragot::Window::~~Window ()

```

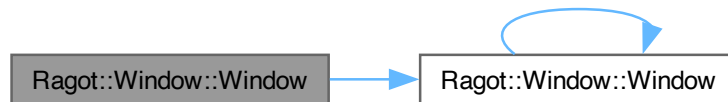
Destructor for the [Window](#) class.

6.33.3.4 Window() [3/4]

```
Ragot::Window::Window (
    const Window & ) [delete]
```

Deleted copy constructor.

Here is the call graph for this function:



6.33.3.5 Window() [4/4]

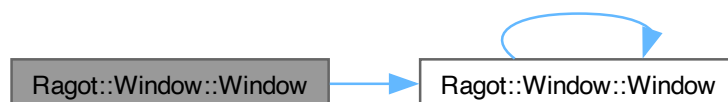
```
Ragot::Window::Window (
    Window && other) [inline], [noexcept]
```

Move constructor for the [Window](#) class.

Parameters

<i>other</i>	Other window to move from.
--------------	----------------------------

Here is the call graph for this function:



6.33.4 Member Function Documentation

6.33.4.1 get_height()

```
unsigned Ragot::Window::get_height () [inline]
```

Gets the height of the window.

Returns

Height of the window.

6.33.4.2 get_width()

```
unsigned Ragot::Window::get_width () [inline]
```

Gets the width of the window.

Returns

Width of the window.

6.33.4.3 operator=() [1/2]

```
Window & Ragot::Window::operator= (
    const Window & ) [delete]
```

Deleted copy assignment operator.

Here is the call graph for this function:

**6.33.4.4 operator=() [2/2]**

```
Window & Ragot::Window::operator= (
    Window && other) [inline], [noexcept]
```

Move assignment operator for the [Window](#) class.

Parameters

<i>other</i>	Other window to move from.
--------------	----------------------------

Returns

Reference to the moved window.

Here is the call graph for this function:



6.33.4.5 swap_buffers()

```
void Ragot::Window::swap_buffers () [inline]
```

Swaps the OpenGL buffers.

Here is the caller graph for this function:



6.33.5 Member Data Documentation

6.33.5.1 height

```
unsigned Ragot::Window::height [private]
```

Height of the window.

6.33.5.2 opengl_context

```
SDL_GLContext Ragot::Window::opengl_context [private]
```

OpenGL context.

6.33.5.3 width

```
unsigned Ragot::Window::width [private]
```

Width of the window.

6.33.5.4 window_handle

```
SDL_Window* Ragot::Window::window_handle [private]
```

Handle to the SDL window.

The documentation for this class was generated from the following files:

- [main/Window.hpp](#)
- [main/Window.cpp](#)

Chapter 7

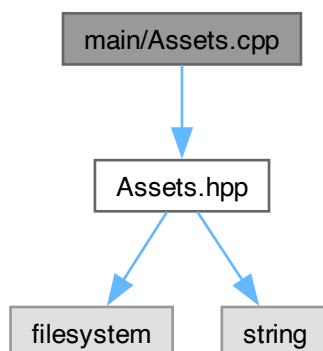
File Documentation

7.1 main/Assets.cpp File Reference

This file implements the Assets class, which manages the asset paths for the application.

```
#include "Assets.hpp"
```

Include dependency graph for Assets.cpp:



Namespaces

- namespace `Ragot`

Variables

- `Assets & Ragot::assets = Assets::instance()`

7.1.1 Detailed Description

This file implements the Assets class, which manages the asset paths for the application.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

The Assets class provides a singleton instance to manage asset paths, allowing for easy retrieval of asset files. It initializes the base path based on the executable file path, and provides a method to get the full path of an asset by its name.

Version

1,0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](https://github.com/andresragot)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

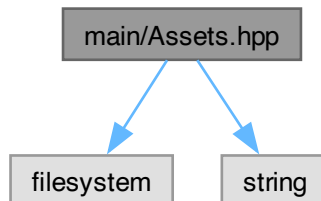
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.2 main/Assets.hpp File Reference

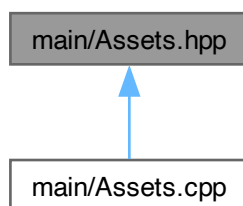
This file defines the Assets class, which manages the asset paths for the application. It provides a singleton instance to access asset paths based on the executable's location. The class initializes the base path for assets depending on whether the build is in debug or release mode. It also provides a method to retrieve the full path of an asset by its name.


```
#include <filesystem>
#include <string>
```

Include dependency graph for Assets.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Assets](#)
Manages the asset paths for the application.

Namespaces

- namespace [Ragot](#)

7.2.1 Detailed Description

This file defines the Assets class, which manages the asset paths for the application. It provides a singleton instance to access asset paths based on the executable's location. The class initializes the base path for assets depending on whether the build is in debug or release mode. It also provides a method to retrieve the full path of an asset by its name.

Author

Andrés Ragot (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.3 Assets.hpp

[Go to the documentation of this file.](#)

```
00001
00034
00035 #pragma once
00036
00037 #include <filesystem>
00038 #include <string>
00039
00040 namespace Ragot
00041 {
00042     using std::string;
00043     using std::filesystem::path;
00044
00052     class Assets
00053     {
00054     public:
00056         static Assets & instance()
00057         {
00065             static Assets assets;
00066             return assets;
00067         }
00068     private:
00071
00072
```

```

00076     Assets() = default;
00080     Assets(const Assets&) = delete;
00084     Assets(const Assets&&) = delete;
00088     Assets& operator = (const Assets&) = delete;
00092     Assets& operator = (const Assets&&) = delete;
00093
00094     public:
00095
00105     void initialize(const string& executable_file_path)
00106     {
00107         #if defined NDEBUG
00108             base_path = path{ executable_file_path }.parent_path() / "assets";
00109         #else
00110             base_path = path{ "../..../assets/" };
00111         #endif
00112     }
00113
00122     path get_asset_path(const string & asset_name)
00123     {
00124         return base_path/asset_name;
00125     }
00126
00127 };
00128
00129 extern Assets & assets;
00130 }

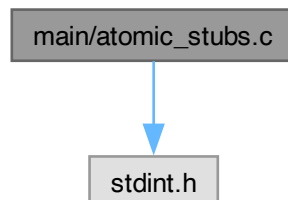
```

7.4 main/atomic_stubs.c File Reference

This file provides atomic operations using GCC built-in functions.

```
#include <stdint.h>
```

Include dependency graph for atomic_stubs.c:



Functions

- `int __atomic_test_and_set` (volatile void *ptr, int memorder)
- `void __atomic_clear` (volatile void *ptr, int memorder)

7.4.1 Detailed Description

This file provides atomic operations using GCC built-in functions.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.4.2 Function Documentation

7.4.2.1 `__atomic_clear()`

```
void __atomic_clear (
    volatile void * ptr,
    int memorder)
```

7.4.2.2 `__atomic_test_and_set()`

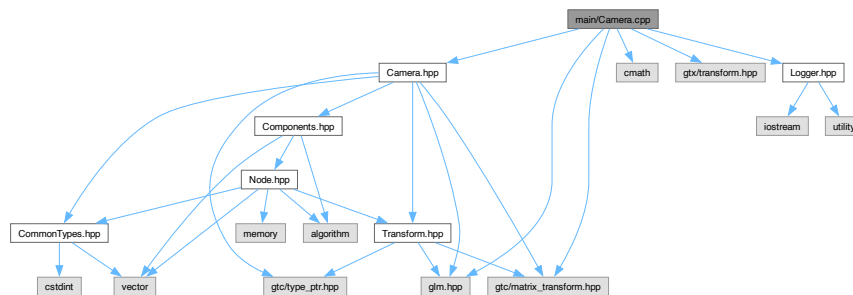
```
int __atomic_test_and_set (
    volatile void * ptr,
    int memorder)
```

7.5 main/Camera.cpp File Reference

This file implements the Camera class, which manages camera properties and operations in a 3D space.

```
#include "Camera.hpp"
#include <cmath>
#include <gtx/transform.hpp>
#include <gtc/matrix_transform.hpp>
#include <glm.hpp>
#include "Logger.hpp"
```

Include dependency graph for Camera.cpp:



Namespaces

- namespace [Ragot](#)

Macros

- `#define` [GLM_ENABLE_EXPERIMENTAL](#)

7.5.1 Detailed Description

This file implements the Camera class, which manages camera properties and operations in a 3D space.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Camera class provides functionality to log camera information, calculate normals for faces, and determine visibility of faces based on their orientation relative to the camera. It uses GLM (OpenGL Mathematics) for vector and matrix operations, and includes methods for calculating normals and checking face visibility.

Version

0.1

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.5.2 Macro Definition Documentation

7.5.2.1 GLM_ENABLE_EXPERIMENTAL

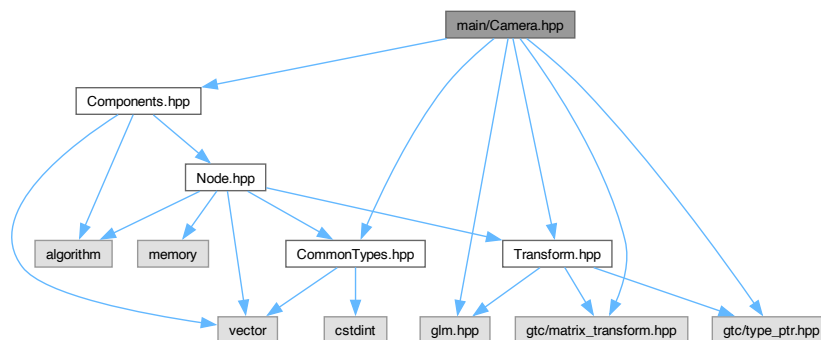
```
#define GLM_ENABLE_EXPERIMENTAL
```

7.6 main/Camera.hpp File Reference

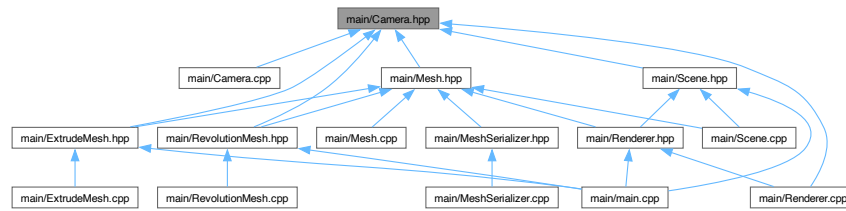
This file implements the Camera class, which manages camera properties and operations in a 3D space.

```
#include "CommonTypes.hpp"
#include "Components.hpp"
#include "Transform.hpp"
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
```

Include dependency graph for Camera.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Camera](#)
Represents a camera in a 3D space, managing its properties and transformations.

Namespaces

- namespace [Ragot](#)

7.6.1 Detailed Description

This file implements the Camera class, which manages camera properties and operations in a 3D space.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Camera class provides functionality to log camera information, calculate normals for faces, and determine visibility of faces based on their orientation relative to the camera. It uses GLM (OpenGL Mathematics) for vector and matrix operations, and includes methods for calculating normals and checking face visibility.

Version

0.1

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.7 Camera.hpp

[Go to the documentation of this file.](#)

```

00001
00033
00034
00035 #pragma once
00036
00037 #include "CommonTypes.hpp"
00038 #include "Components.hpp"
00039 #include "Transform.hpp"
00040 #include <glm.hpp>
00041 #include <gtc/matrix_transform.hpp>
00042 #include <gtc/type_ptr.hpp>
00043
00044 namespace Ragot
00045 {
00054     class Camera : public Component
00055     {
00059         static const char* CAMERA_TAG;
00060
00061     public:
00062         using Matrix4x4 = glm::mat4;
00063
00064     private:
00065         float fov;
00066         float near_plane;
00067         float far_plane;
00068         float aspect_ratio;
00069
00070         glm::vec3 target;
00071
00072         mutable Matrix4x4 projection_matrix;
00073         mutable Matrix4x4 view_matrix;
00074         mutable Matrix4x4 vp_matrix;
00075         mutable bool projDirty = true;
00076         mutable bool viewDirty = true;
00077         mutable bool vpDirty = true;
00078
00079     public:
00083         Camera() = delete;
00092         Camera(float aspect_ratio = 1.f,
00093                float near_plane = 1.f,
00094                float far_plane = 100.f,
00095                float fov_deg = 60.f)
00096             : fov(fov_deg), near_plane(near_plane), far_plane(far_plane), aspect_ratio(aspect_ratio)
00097         {
00098             set_position(glm::vec3(0.f));
00099             target = glm::vec3(0.f, 0.f, -1.f);
00100         }
00104         ~Camera() = default;
00105
00106         // --- Getters ---
00112         float get_fov() const { return fov; }
00113
00119         float get_near_plane() const { return near_plane; }
00120
00126         float get_far_plane() const { return far_plane; }
00127
00133         float get_aspect_ratio() const { return aspect_ratio; }
00134
00140         glm::vec3 get_location() const { return get_position(); }
00141
00147         glm::vec3 get_target() const { return target; }
00148
00154         bool is_dirty() const { return projDirty || viewDirty || vpDirty; }
00155
00156         // --- Setters (mark dirty) ---
00157
00163         void set_fov(float deg) { fov = deg; projDirty = true; vpDirty = true; }
00164
00170         void set_near_plane(float np) { near_plane = np; projDirty = true; vpDirty = true; }
00171
00177         void set_far_plane(float fp) { far_plane = fp; projDirty = true; vpDirty = true; }
00178
00184         void set_aspect_ratio(float ar) { aspect_ratio = ar; projDirty = true; vpDirty = true; }
00185
00191         void set_location(const glm::vec3 &p) { set_position(p); viewDirty = true; vpDirty = true; }
00192
00198         void set_target(const glm::vec3 &t) { target = t; viewDirty = true; vpDirty = true; }
00199
00205         const Matrix4x4& get_projection_matrix() const
00206         {
00207             if (projDirty)
00208                 {

```



```

00209         projection_matrix = glm::perspective(glm::radians(fov), aspect_ratio, near_plane,
00210         far_plane);
00210         projDirty = false;
00211     }
00212     return projection_matrix;
00213 }
00214
00220     const Matrix4x4& get_view_matrix() const
00221     {
00222         if (viewDirty)
00223         {
00224             view_matrix = glm::lookAt(
00225                 get_position(),
00226                 target,
00227                 glm::vec3(0.f, 1.f, 0.f)
00228             );
00229             viewDirty = false;
00230         }
00231         return view_matrix;
00232     }
00233
00239     const Matrix4x4& get_vp_matrix() const
00240     {
00241         if (vpDirty)
00242         {
00243             vp_matrix = get_projection_matrix() * get_view_matrix();
00244             vpDirty = false;
00245         }
00246         return vp_matrix;
00247     }
00248
00254     glm::vec3 get_view_direction() const
00255     {
00256         return glm::normalize(target - get_position());
00257     }
00258
00264     glm::vec3 get_right_direction() const
00265     {
00266         return glm::normalize(glm::cross(get_view_direction(), glm::vec3(0.f, 1.f, 0.f)));
00267     }
00268
00274     glm::vec3 get_up_direction() const
00275     {
00276         return glm::normalize(glm::cross(get_right_direction(), get_view_direction()));
00277     }
00278
00285     glm::vec3 project_to_ndc(const glm::vec4 &worldPos) const
00286     {
00287         glm::vec4 clip = get_vp_matrix() * worldPos;
00288         return (clip.w == 0.f) ? glm::vec3(0.f) : glm::vec3(clip) / clip.w;
00289     }
00290
00299     vertex_t calculate_normal(const vertex_t &v1, const vertex_t &v2, const vertex_t &v3);
00300
00311     bool is_face_visible(const vertex_t &v1, const vertex_t &v2, const vertex_t &v3);
00312
00318     void log_camera_info() const;
00319
00320 };
00321 }

```

7.8 main/CommonTypes.hpp File Reference

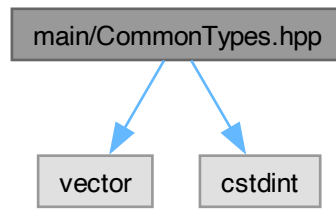
This file defines common types and structures used in the [Ragot](#) engine, including camera transformations, mesh information, and rendering flags.

```

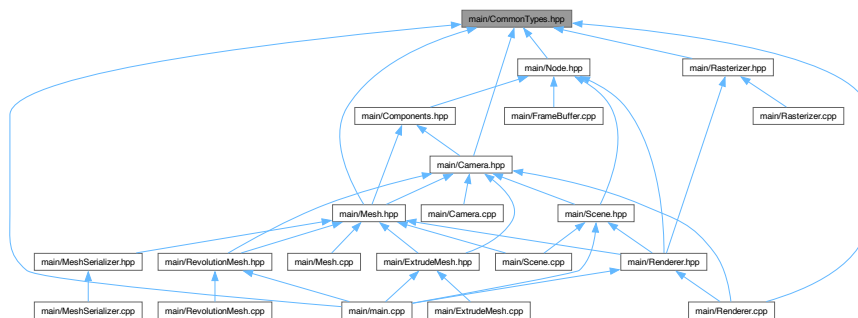
#include <vector>
#include <cstdint>

```

Include dependency graph for CommonTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Ragot::Camera_Transform](#)
Represents the transformation of a camera in 3D space.
- struct [Ragot::transform_t](#)
Represents a transformation in 3D space.
- struct [Ragot::vertex_t](#)
Represents a vertex in 3D space.
- struct [Ragot::face_t](#)
Represents a face in a 3D mesh.
- struct [Ragot::coordinates_t](#)
Represents 2D coordinates.
- struct [Ragot::mesh_info_t](#)
Represents information about a mesh.

Namespaces

- namespace [Ragot](#)

Enumerations

- enum [Ragot::render_flag_t](#) : uint8_t { [Ragot::RENDER_NONE](#) , [Ragot::RENDER_REVOLUTION](#) , [Ragot::RENDER_EXTRUDE](#) , [Ragot::RENDER_MAX](#) }

Flags for rendering types.

Variables

- constexpr float [Ragot::PI](#) = 3.141592653f

Mathematical constant PI.

7.8.1 Detailed Description

This file defines common types and structures used in the [Ragot](#) engine, including camera transformations, mesh information, and rendering flags.

Author

Andrés [Ragot](#) (github.com/andresragot)

The [CommonTypes.hpp](#) file provides essential data structures for representing camera transformations, vertex and face definitions, rendering flags, and mesh information. It includes structures for camera position and direction, vertex coordinates, face definitions (triangles and quads), and rendering flags for different mesh types. The file also defines a constant for the mathematical constant PI.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.9 CommonTypes.hpp

[Go to the documentation of this file.](#)

```

00001
00033
00034 #pragma once
00035 #include <vector>
00036 #include <stdint>
00037
00038 namespace Ragot
00039 {
00046     struct Camera_Transform
00047     {
00048         float x, y, z;           // Posición de la cámara
00049         float dir_x, dir_y, dir_z; // Dirección de la cámara (vector normalizado)
00050     };
00051
00058     struct transform_t
00059     {
00060         float x, y, z;
00061         float alpha, beta, gamma;
00062         float scale;
00063     };
00064
00071     struct vertex_t
00072     {
00073         float x;
00074         float y;
00075         float z;
00076     };
00077
00084     struct face_t
00085     {
00086         bool is_quad;
00087         int v1;
00088         int v2;
00089         int v3;
00090         int v4;
00091     };
00092
00093
00100     enum render_flag_t : uint8_t
00101     {
00102         RENDER_NONE,
00103         RENDER_REVOLUTION,
00104         RENDER_EXTRUDE,
00105         RENDER_MAX
00106     };
00107
00108
00115     struct coordinates_t
00116     {
00117         float x;
00118         float y;
00119     };
00120
00121
00128     struct mesh_info_t
00129     {
00130         size_t vertex_amount = 0;
00131         render_flag_t render_flag = RENDER_NONE;
00132         std::vector < coordinates_t > coordinates;
00133
00134         mesh_info_t () = default;
00135         mesh_info_t (std::vector < coordinates_t > & coords, render_flag_t flag)
00136             : vertex_amount (coords.size ()), render_flag (flag), coordinates (coords)
00137         {
00138         }
00139     };
00140
00146     constexpr float PI = 3.141592653f;
00147 }

```

7.10 main/Components.hpp File Reference

This file defines the Component class, which serves as a base class for components in the [Ragot](#) engine.

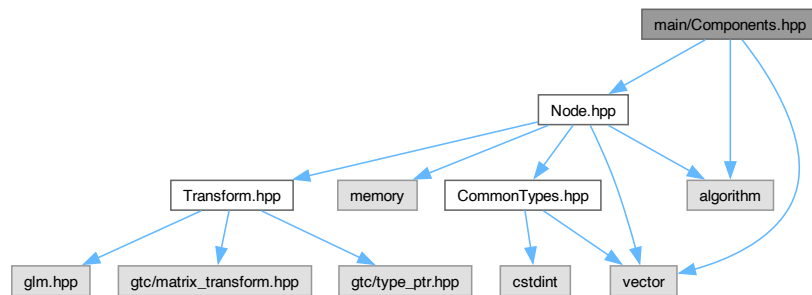
```

#include "Node.hpp"
#include <vector>

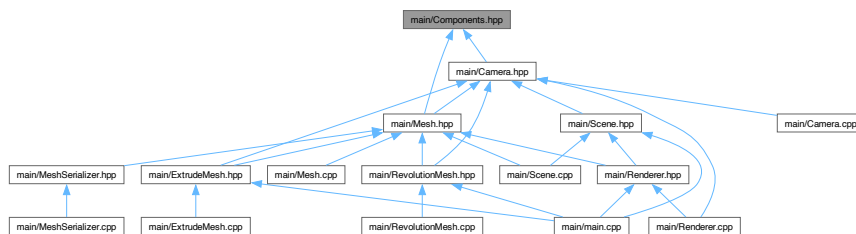
```

```
#include <algorithm>
```

Include dependency graph for Components.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Component](#)
Base class for components in the [Ragot](#) engine.

Namespaces

- namespace [Ragot](#)

7.10.1 Detailed Description

This file defines the Component class, which serves as a base class for components in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Component class inherits from Node and provides functionality to manage a collection of components. It allows adding and removing components, and provides access to the list of components.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.11 Components.hpp

[Go to the documentation of this file.](#)

```

00001
00033
00034 #pragma once
00035
00036 #include "Node.hpp"
00037 #include <vector>
00038 #include <algorithm>
00039
00040
00041 namespace Ragot
00042 {
00051     class Component : public Node
00052     {
00053     public:
00054
00060         Component() = default;
00061
00067         virtual ~Component() = default;
00068
00072         Component(const Component & ) = delete;
00073
00079         Component(const Component &&) = delete;
00080
00086         Component & operator = (const Component & ) = delete;
00087
00093         Component & operator = (const Component &&) = delete;
00094
00095     protected:
00096
00102         std::vector < std::shared_ptr < Component > > components;
00103
00104     public:
00105

```

```

00113     void add_component(std::shared_ptr < Component > component)
00114     {
00115         if (component)
00116         {
00117             components.emplace_back(component);
00118             component->parent = this;
00119         }
00120     }
00121
00129     void remove_component (std::shared_ptr < Component > component)
00130     {
00131         if (component)
00132         {
00133             auto it = std::remove(components.begin(), components.end(), component);
00134             if (it != components.end())
00135             {
00136                 components.erase(it, components.end());
00137                 component->parent = nullptr;
00138             }
00139         }
00140     }
00141
00149     const std::vector<std::shared_ptr < Component > > get_components() const { return components;
00150 }
00151 };
00152 }

```

7.12 main/driver_ek79007.cpp File Reference

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_ek79007.hpp"
```

```
#include "esp_lcd_ek79007.h"
```

Include dependency graph for driver_ek79007.cpp:



Namespaces

- namespace [Ragot](#)

Functions

- static bool [Ragot::panel_refresh_callback](#) (esp_lcd_panel_handle_t panel, esp_lcd_dpi_panel_event_data_t *edata, void *user_ctx)

Variables

- static const char * [Ragot::TAG](#) = "DriverEK79007"

7.12.1 Detailed Description

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

Version

1.0

Date

2025-04-17

Copyright

Copyright (c) 2025 Andrés [Ragot](https://github.com/andresragot) MIT License

Copyright (c) 2025 Andrés [Ragot](https://github.com/andresragot)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.13 main/driver_ek79007.hpp File Reference

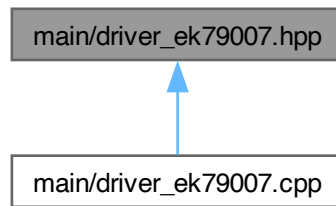
This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_lcd.hpp"
```

Include dependency graph for driver_ek79007.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::DriverEK79007](#)
Driver for the EK79007 LCD panel.

Namespaces

- namespace [Ragot](#)

7.13.1 Detailed Description

This file implements the DriverEK79007 class, which manages the initialization and operation of the EK79007 LCD panel driver. The DriverEK79007 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés [Ragot](#) (github.com/andresragot)

Version

1.0

Date

2025-04-17

Copyright

Copyright (c) 2025 Andrés [Ragot](#) MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.14 driver_ek79007.hpp

[Go to the documentation of this file.](#)

```

00001
00033
00034 #pragma once
00035
00036 #include "driver_lcd.hpp"
00037
00038
00039 namespace Ragot
00040 {
00048     class DriverEK79007 : public DriverLCD
00049     {
00050     public:
00051
00055         DriverEK79007();
00056
00062         ~DriverEK79007() override;
00063
00074         esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) override;
00075
00084         esp_err_t deinit() override;
00085
00096         esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) override {return ESP_FAIL;};
00097
00107         esp_err_t send_frame_buffer(const void * frame_buffer) override;
00108
00120         IRAM_ATTR bool refresh_frame_buffer( esp_lcd_panel_handle_t panel,
00121         esp_lcd_dpi_panel_event_data_t * edata, void * user_ctx);
00121
00122     private:
00123         uint16_t panel_clk_freq_mhz;
00124         uint32_t hsync_pulse_width;
00125         uint32_t hsync_back_porch;
00126         uint32_t hsync_front_porch;
00127         uint32_t vsync_pulse_width;
00128         uint32_t vsync_back_porch;
00129         uint32_t vsync_front_porch;
00130         uint8_t mipi_lane_num;
00131         uint16_t mipi_dsi_max_data_rate_mbps;
00132
00133     private:
00139         void bsp_enable_dsi_phy_power();
00140
00148         void bsp_init_lcd_backlight(gpio_num_t bk_pin);
00149
00150     public:
00156         SemaphoreHandle_t refresh_semaphore;
00157     };
00158 }

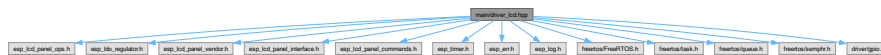
```

7.15 main/driver_lcd.hpp File Reference

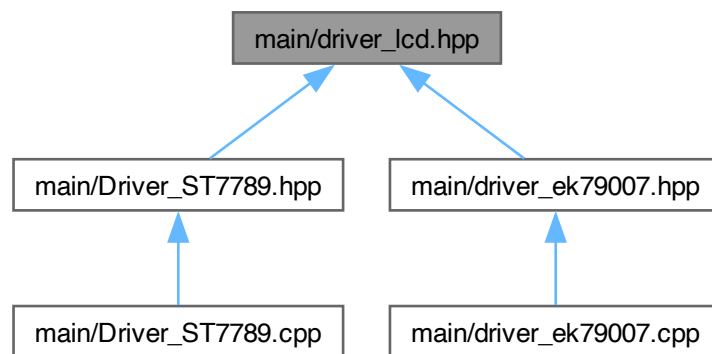
This file defines the DriverLCD class, which serves as a base class for LCD drivers in the [Ragot](#) engine.

```
#include "esp_lcd_panel_ops.h"
#include "esp_ldo_regulator.h"
#include "esp_lcd_panel_vendor.h"
#include "esp_lcd_panel_interface.h"
#include "esp_lcd_panel_commands.h"
#include "esp_timer.h"
#include "esp_err.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "driver/gpio.h"
```

Include dependency graph for driver_lcd.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::DriverLCD](#)
Base class for LCD drivers.

Namespaces

- namespace [Ragot](#)

7.15.1 Detailed Description

This file defines the DriverLCD class, which serves as a base class for LCD drivers in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The DriverLCD class provides an interface for initializing, deinitializing, setting pixels, and sending frame buffers to an LCD panel. It includes methods to get the width, height, pixel format, and handler of the LCD panel. The class is designed to be inherited by specific LCD driver implementations, such as the EK79007 driver.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025

MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.16 driver_lcd.hpp

[Go to the documentation of this file.](#)

```

00001
00035
00036 #pragma once
00037
00038 #include "esp_lcd_panel_ops.h"
00039 #include "esp_ldo_regulator.h"
00040 #include "esp_lcd_panel_vendor.h"
00041 #include "esp_lcd_panel_interface.h"
00042 #include "esp_lcd_panel_commands.h"
00043 #include "esp_timer.h"
00044 #include "esp_err.h"
00045 #include "esp_log.h"
00046 #include "freertos/FreeRTOS.h"
00047 #include "freertos/task.h"
00048 #include "freertos/queue.h"
00049 #include "freertos/semphr.h"
00050 #include "driver/gpio.h"
00051 #ifdef CONFIG_IDF_TARGET_ESP32P4
00052 #include "esp_lcd_mipi_dsi.h"
00053 #endif
00054
00055 namespace Ragot
00056 {
00065     class DriverLCD
00066     {
00067     public:
00068
00074         DriverLCD() = default;
00075
00081         virtual ~DriverLCD() = default;
00082
00092         virtual esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) = 0;
00093
00101         virtual esp_err_t deinit() = 0;
00102
00113         virtual esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) = 0;
00114
00123         virtual esp_err_t send_frame_buffer( const void * frame_buffer) = 0;
00124
00130         const size_t get_width() const { return width; }
00131
00137         size_t get_width() { return width; }
00138
00144         const size_t get_height() const { return height; }
00145
00151         size_t get_height() { return height; }
00152
00153
00159         const lcd_color_rgb_pixel_format_t get_pixel_format() const { return pixel_format; }
00160
00166         lcd_color_rgb_pixel_format_t get_pixel_format() { return pixel_format; }
00167
00173         const esp_lcd_panel_handle_t get_handler() const { return handler; }
00174
00180         esp_lcd_panel_handle_t get_handler() { return handler; }
00181
00188         const bool is_initialized() const { return initialized; }
00189
00196         bool is_initialized() { return initialized; }
00197
00198     protected:
00199         bool initialized = false;
00200         size_t width;
00201         size_t height;
00202         lcd_color_rgb_pixel_format_t pixel_format;
00203         esp_lcd_panel_handle_t handler;
00204     };
00205 }

```

7.17 main/Driver_ST7789.cpp File Reference

This file implements the Driver_ST7789 class, which manages the initialization and operation of the ST7789 LCD panel driver. The Driver_ST7789 class inherits from the DriverLCD class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

7.18 main/Driver_ST7789.hpp File Reference

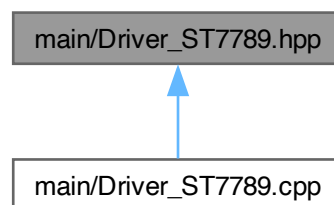
This file implements the `Driver_ST7789` class, which manages the initialization and operation of the ST7789 LCD panel driver. The `Driver_ST7789` class inherits from the `DriverLCD` class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

```
#include "driver_lcd.hpp"
```

Include dependency graph for `Driver_ST7789.hpp`:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Driver_ST7789](#)
Driver for the ST7789 LCD panel.

Namespaces

- namespace [Ragot](#)

7.18.1 Detailed Description

This file implements the `Driver_ST7789` class, which manages the initialization and operation of the ST7789 LCD panel driver. The `Driver_ST7789` class inherits from the `DriverLCD` class and provides methods to initialize, deinitialize, and send frame buffers to the LCD panel.

Author

Andrés [Ragot](#) (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.19 Driver_ST7789.hpp

[Go to the documentation of this file.](#)

```

00001
00032
00033 #pragma once
00034 #include "driver_lcd.hpp"
00035
00036 namespace Ragot
00037 {
00045     class Driver_ST7789 : public DriverLCD
00046     {
00047     private:
00051         static constexpr const char * TAG = "[Driver_ST7789]...";
00052
00053     public:
00059         Driver_ST7789 ();
00060
00066         virtual ~Driver_ST7789() = default;
00067
00078         esp_err_t init(gpio_num_t reset_pin, gpio_num_t bk_pin) override;
00079
00088         esp_err_t deinit() override;
00089
00100         esp_err_t set_pixel(uint32_t x, uint32_t y, uint32_t color) override {return ESP_FAIL;};
00101
00111         esp_err_t send_frame_buffer( const void * frame_buffer) override;
00112
00121         IRAM_ATTR bool refresh_frame_buffer(void * user_ctx);
00122
00123     private:
00131         void bsp_init_lcd_backlight(gpio_num_t bk_pin);
00132
00133     public:
00137         SemaphoreHandle_t refresh_semaphore;
00138     };
00139 }

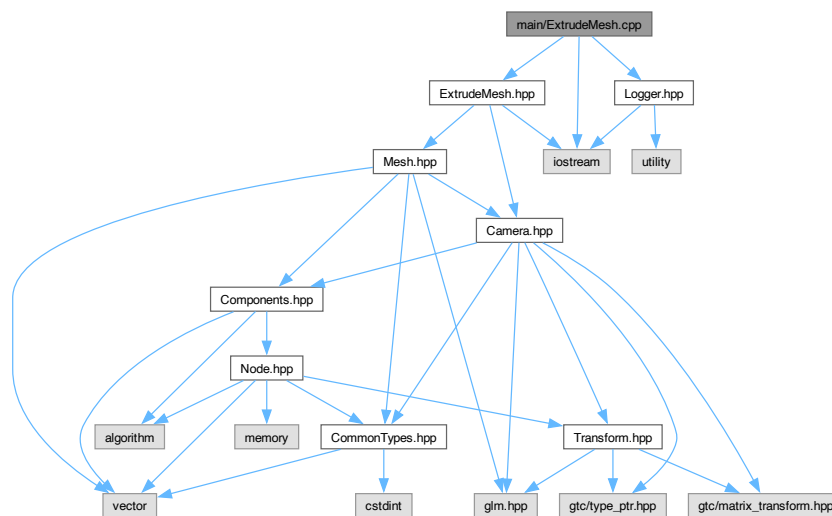
```


7.20 main/ExtrudeMesh.cpp File Reference

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include "ExtrudeMesh.hpp"
#include <iostream>
#include "Logger.hpp"
```

Include dependency graph for ExtrudeMesh.cpp:



Namespaces

- namespace [Ragot](#)

7.20.1 Detailed Description

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Author

Andrés [Ragot](#) (github.com/andresragot)

The ExtrudeMesh class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

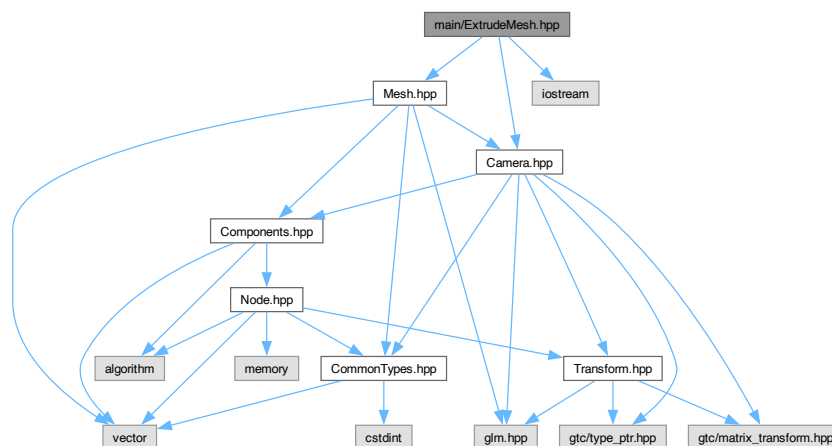
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

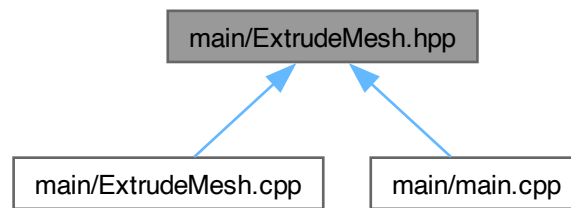
7.21 main/ExtrudeMesh.hpp File Reference

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

```
#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>
Include dependency graph for ExtrudeMesh.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::ExtrudeMesh](#)

Represents a 3D mesh created by extruding a 2D shape along a specified height. This class inherits from the [Mesh](#) class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Namespaces

- namespace [Ragot](#)

7.21.1 Detailed Description

This file implements the ExtrudeMesh class, which manages the extrusion of a mesh in 3D space. The ExtrudeMesh class inherits from the Mesh class and provides methods to generate vertices and faces for the extruded mesh. It also includes methods for culling faces based on the camera's view direction and logging mesh information.

Author

Andrés [Ragot](#) (github.com/andresragot)

The ExtrudeMesh class is designed to create a 3D mesh by extruding a 2D shape along a specified height. It uses the GLM library for vector and matrix operations, and includes functionality for face culling based on the camera's view direction. The class also provides a method to log detailed information about the mesh, including its position, rotation, scale, and vertex data.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.22 ExtrudeMesh.hpp

[Go to the documentation of this file.](#)

```

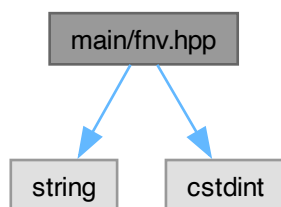
00001
00036
00037 #pragma once
00038
00039 #include "Mesh.hpp"
00040 #include "Camera.hpp"
00041 #include <iostream>
00042
00043 namespace Ragot
00044 {
00054     class ExtrudeMesh : public Mesh
00055     {
00056     protected:
00060         static const char* EXTRUDE_TAG;
00061
00067         float height = 1.0f;
00068
00075         bool faces_can_be_quads = false;
00076
00082         const Camera & cam;
00083
00090         glm::vec4 planes[4];
00091
00098         glm::vec3 camPos;
00099
00100     public:
00107         ExtrudeMesh (mesh_info_t & mesh_info, const Camera & cam) : Mesh (mesh_info), cam (cam)
00108         {
00109             vertices.reserve (mesh_info.coordinates.size() * 2 );
00110             faces.reserve (mesh_info.coordinates.size() * 3 - 3);
00111             // Si son 14 vertices -> 39 - 28 = 11
00112             // Si son 12 vertices -> 33 - 24 = 9
00113             // Si son 10 vertices -> 27 - 20 = 7
00114             // Si son 8 vertices -> 9 - 16 = -7
00115             // Si son 6 vertices -> 15 - 12 = 3
00116             // Si son 4 vertices -> 9 - 5 = 4
00117
00118             // % 8 porque como están las coordenadas duplicadas...
00119             faces_can_be_quads = (mesh_info.vertex_amount % 8 == 0 || mesh_info.vertex_amount == 4);
00120             generate_vertices();
00121             generate_faces();
00122
00123             std::cout << "Etrude Vertices: " << vertices.size() << std::endl;
00124             std::cout << "Extrude Faces: " << faces.size() << std::endl;
00125         }
00126
00132         ~ExtrudeMesh() = default;

```

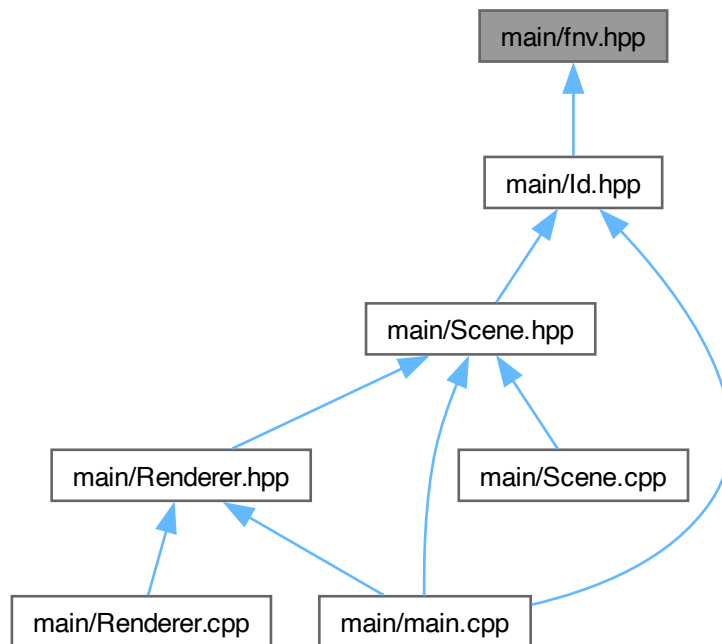
```
00133
00139     void generate_vertices () override;
00140
00147     void generate_faces     () override;
00148
00165     bool are_vertices_coplanar (const glm::fvec4 & v1, const glm::fvec4 & v2, const glm::fvec4 &
v3, const glm::fvec4 & v4, float tolerance = 0.1f);
00166
00170     void log_mesh_info() const;
00171 };
00172 }
```

7.23 main/fnv.hpp File Reference

```
#include <string>
#include <stdint>
Include dependency graph for fnv.hpp:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [basics](#)
- namespace [basics::internal](#)

Macros

- `#define FNV(X)`
- `#define FNV32(X)`
- `#define FNV64(X)`

Functions

- `template<size_t LENGTH>`
`constexpr uint32_t basics::internal::static_fnv32 (const char *chars)`
- `template<> constexpr uint32_t basics::internal::static_fnv32< 1 > (const char *)`
- `template<size_t LENGTH>`
`constexpr uint64_t basics::internal::static_fnv64 (const char *chars)`
- `template<> constexpr uint64_t basics::internal::static_fnv64< 1 > (const char *)`
- `template<size_t LENGTH>`
`constexpr uint32_t basics::static_fnv32 (const char(&chars)[LENGTH])`
- `template<size_t LENGTH>`
`constexpr uint64_t basics::static_fnv64 (const char(&chars)[LENGTH])`

- template<size_t LENGTH>
constexpr unsigned [basics::static_fnv](#) (const char(&chars)[LENGTH])
- template<size_t LENGTH>
uint32_t [basics::fnv32](#) (const char(&chars)[LENGTH])
- uint32_t [basics::fnv32](#) (const std::string &s)
- constexpr unsigned [operator""_fnv](#) (const char *c)

Variables

- constexpr uint32_t [basics::internal::fnv_basis_32](#) = 0x811c9dc5u
- constexpr uint32_t [basics::internal::fnv_prime_32](#) = 0x01000193u
- constexpr uint64_t [basics::internal::fnv_basis_64](#) = 0xcbf29ce484222325u
- constexpr uint64_t [basics::internal::fnv_prime_64](#) = 0x00000100000001b3u

7.23.1 Macro Definition Documentation

7.23.1.1 FNV

```
#define FNV(  
    X)
```

Value:

[basics::static_fnv](#) (#X)

7.23.1.2 FNV32

```
#define FNV32(  
    X)
```

Value:

[basics::static_fnv32](#) (#X)

7.23.1.3 FNV64

```
#define FNV64(  
    X)
```

Value:

[basics::static_fnv64](#) (#X)

7.23.2 Function Documentation

7.23.2.1 operator""_fnv()

```
unsigned operator""_fnv (  
    const char * c) [constexpr]
```

7.24 fnv.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * FNV
00003  * Copyright © 2017+ Ángel Rodríguez Ballesteros
00004  *
00005  * Distributed under the Boost Software License, version 1.0
00006  * See documents/LICENSE.TXT or www.boost.org/LICENSE_1_0.txt
00007  *
00008  * angel.rodriguez@esne.edu
00009  *
00010  * C1712171830
00011  */
00012
00013 #ifndef BASICS_FNV_HEADER
00014 #define BASICS_FNV_HEADER
00015
00016     #include <string>
00017     #include <stdint>
00018
00019     namespace basics
00020     {
00021
00022         namespace internal
00023         {
00024
00025             constexpr uint32_t fnv_basis_32 = 0x811c9dc5u;
00026             constexpr uint32_t fnv_prime_32 = 0x01000193u;
00027             constexpr uint64_t fnv_basis_64 = 0xcbf29ce484222325u;
00028             constexpr uint64_t fnv_prime_64 = 0x00000100000001b3u;
00029
00030             template< size_t LENGTH >
00031             constexpr uint32_t static_fnv32 (const char * chars)
00032             {
00033                 return (static_fnv32< LENGTH - 1 > (chars) ^ chars[LENGTH - 2]) * fnv_prime_32;
00034             }
00035
00036             template< >
00037             constexpr uint32_t static_fnv32< 1 > (const char * )
00038             {
00039                 return fnv_basis_32;
00040             }
00041
00042             template< size_t LENGTH >
00043             constexpr uint64_t static_fnv64 (const char * chars)
00044             {
00045                 return (static_fnv64< LENGTH - 1 > (chars) ^ chars[LENGTH - 2]) * fnv_prime_64;
00046             }
00047
00048             template< >
00049             constexpr uint64_t static_fnv64< 1 > (const char * )
00050             {
00051                 return fnv_basis_64;
00052             }
00053
00054         }
00055
00056         // -----
00057
00065         template< size_t LENGTH >
00066         constexpr uint32_t static_fnv32 (const char (& chars)[LENGTH])
00067         {
00068             return internal::static_fnv32< LENGTH > (chars);
00069         }
00070
00071         // -----
00072
00073         template< size_t LENGTH >
00074         constexpr uint64_t static_fnv64 (const char (& chars)[LENGTH])
00075         {
00076             return internal::static_fnv64< LENGTH > (chars);
00077         }
00078
00079         // -----
00080
00081         #if BASICS_INT_SIZE == 4
00082
00083             template< size_t LENGTH >
00084             constexpr unsigned static_fnv (const char (& chars)[LENGTH])
00085             {
00086                 return internal::static_fnv32< LENGTH > (chars);
00087             }
00088
00089         #else

```



```

00090
00091     template< size_t LENGTH >
00092     constexpr unsigned static_fnv (const char (& chars)[LENGTH])
00093     {
00094         return static_cast < unsigned > (internal::static_fnv64< LENGTH > (chars));
00095     }
00096
00097 #endif
00098
00099 // -----
00100
00101     template< size_t LENGTH >
00102     uint32_t fnv32 (const char (& chars)[LENGTH])
00103     {
00104         uint32_t hash = internal::fnv_basis_32;
00105
00106         for (size_t index = 0; index < LENGTH; ++index)
00107         {
00108             hash ^= chars[index]; // Use array indexing instead
00109             hash *= internal::fnv_prime_32;
00110         }
00111
00112         return hash;
00113     }
00114
00115     inline uint32_t fnv32 (const std::string & s)
00116     {
00117         uint32_t hash = internal::fnv_basis_32;
00118
00119         for (auto c : s)
00120         {
00121             hash ^= c;
00122             hash *= internal::fnv_prime_32;
00123         }
00124
00125         return hash;
00126     }
00127 }
00128
00129 constexpr unsigned operator "" _fnv (const char * c)
00130 {
00131     return c ? 1 : operator "" _fnv ("2");
00132 }
00133
00134 // -----
00135
00136 #define FNV(X)    basics::static_fnv    (#X)
00137 #define FNV32(X)  basics::static_fnv32  (#X)
00138 #define FNV64(X)  basics::static_fnv64  (#X)
00139
00140 #endif

```

7.25 main/FrameBuffer.cpp File Reference

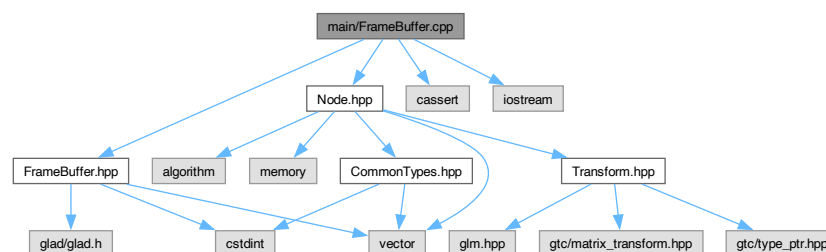
This file implements the FrameBuffer class, which manages a frame buffer for rendering graphics.

```

#include "FrameBuffer.hpp"
#include <cassert>
#include "Node.hpp"
#include <iostream>

```

Include dependency graph for FrameBuffer.cpp:



Namespaces

- namespace [Ragot](#)

7.25.1 Detailed Description

This file implements the `FrameBuffer` class, which manages a frame buffer for rendering graphics.

Author

Andrés [Ragot](#) (github.com/andresragot)

The `FrameBuffer` class provides methods to create a frame buffer, swap buffers, clear the buffer, fill it with a color, set and get pixels, and manage OpenGL textures. It supports both single and double buffering modes. The class is designed to be used in graphics applications where rendering performance and buffer management are crucial.

Version

0.1

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

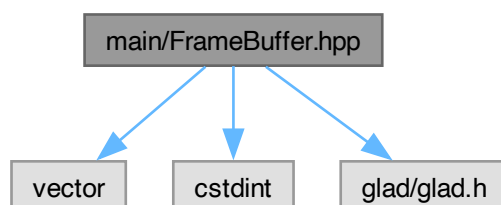
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

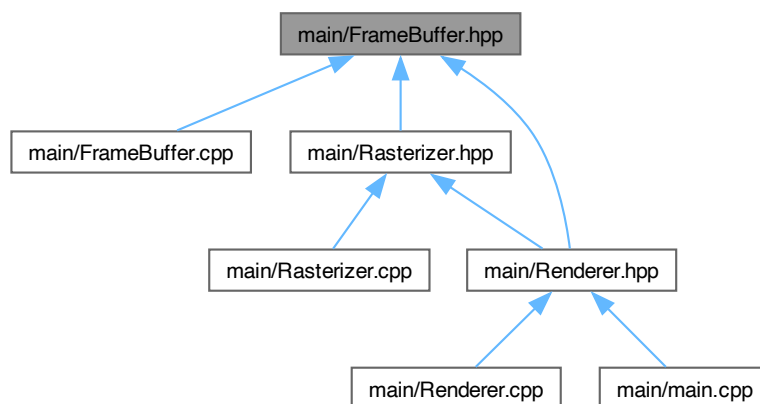
7.26 main/Framebuffer.hpp File Reference

```
#include <vector>
#include <stdint>
#include <glad/glad.h>
```

Include dependency graph for FrameBuffer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Framebuffer< Color >](#)
Class to manage a frame buffer for rendering graphics.

Namespaces

- namespace [Ragot](#)

Typedefs

- using [Ragot::RGB565](#) = uint16_t
- using [Ragot::RGB888](#) = uint32_t
- using [Ragot::RGBA8888](#) = uint32_t
- using [Ragot::RGB8](#) = uint8_t

Color Index.

Enumerations

- enum [Ragot::Buffer](#) : uint8_t { [Ragot::CURRENT_BUFFER](#) = (1 << 0) , [Ragot::NEXT_BUFFER](#) = (1 << 1) , [Ragot::MAX_BUFFER](#) = (1 << 2) }

Enum to represent the different buffers in a frame buffer.

7.27 FrameBuffer.hpp

[Go to the documentation of this file.](#)

```

00001
00035
00036 #pragma once
00037 #include <vector>
00038 #include <stdint>
00039 #if ESP_PLATFORM == 1
00040 #include "RamAllocator.hpp"
00041 #else
00042 #include <glad/glad.h>
00043 #endif
00044
00045 namespace Ragot
00046 {
00047     using RGB565    = uint16_t;
00048     using RGB888    = uint32_t;
00049     using RGBA8888 = uint32_t;
00050     using RGB8      = uint8_t ;
00051
00052     enum Buffer : uint8_t
00053     {
00054         CURRENT_BUFFER = ( 1 << 0 ),
00055         NEXT_BUFFER    = ( 1 << 1 ),
00056         MAX_BUFFER     = ( 1 << 2 )
00057     };
00058
00059     template <typename Color>
00060     class FrameBuffer
00061     {
00062     public:
00063         using TYPE = Color;
00064         #if ESP_PLATFORM == 1
00065         using ColorVector = std::vector < Color, PSRAMAllocator< Color, MALLOC_CAP_8BIT > >;
00066         #else
00067         using ColorVector = std::vector < Color >;
00068         #endif
00069
00070     private:
00071         bool double_buffer;
00072         size_t width;
00073         size_t height;
00074         Color color;
00075         ColorVector buffer_1;
00076         ColorVector buffer_2;
00077         ColorVector * current_buffer;
00078         ColorVector * next_buffer;
00079
00080     public:
00081         FrameBuffer (size_t width, size_t height, bool double_buffer);
00082
00083         FrameBuffer () = delete;
00084
00085         ~FrameBuffer () = default;

```

```

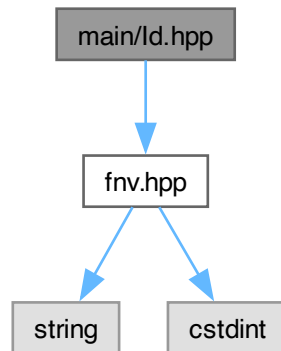
00120
00121 // Nada más queremos que haya un FrameBuffer por ahora.
00122
00126 FrameBuffer (const FrameBuffer &) = delete;
00127
00131 FrameBuffer (const FrameBuffer &&) = delete;
00132
00140 FrameBuffer & operator = (const FrameBuffer &) = delete;
00141
00149 FrameBuffer & operator = (const FrameBuffer &&) = delete;
00150
00156 void swap_buffers();
00157
00163 void clear_buffer( Buffer buffer_to_clear = NEXT_BUFFER );
00164
00171 void fill (Color color = 0, Buffer buffer_to_fill = NEXT_BUFFER);
00172
00180 void set_pixel (size_t x, size_t y, Color color);
00181
00188 void set_pixel (size_t offset, Color color);
00189
00195 void set_pixel (size_t offset);
00196
00202 void set_color (Color color);
00203
00211 Color get_pixel (size_t x, size_t y) const;
00212
00219 size_t get_width () { return width; }
00220
00226 size_t get_width () const { return width; }
00227
00233 size_t get_height () { return height; }
00234
00240 size_t get_height () const { return height; }
00241
00247 const Color * get_buffer() const { return current_buffer->data(); }
00248
00254 Color * get_buffer() { return current_buffer->data(); }
00255
00261 inline void blit_to_window () const
00262 {
00263 // Implementar la función para blit a la ventana
00264 std::copy(current_buffer->begin(), current_buffer->end(), next_buffer->begin());
00265 }
00266
00267
00268 #if ESP_PLATFORM != 1
00269
00275 void initGLTexture();
00276
00282 void sendGL() const;
00283
00289 static GLenum getGLFormat();
00290
00298 static GLenum getGLType();
00299
00305 GLuint getGLTex () const { return gl_tex; }
00306
00307 private:
00313 GLuint gl_tex = 0;
00314 #endif
00315 };
00316 }

```

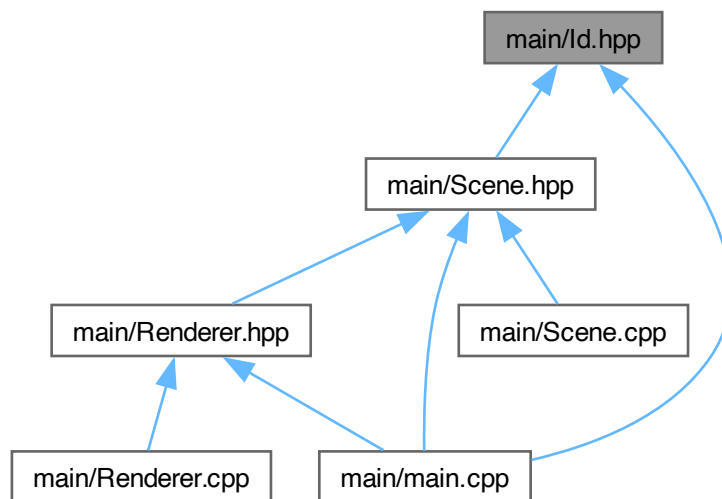
7.28 main/Id.hpp File Reference

```
#include "fnv.hpp"
```

Include dependency graph for Id.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [basics](#)

Macros

- `#define ID(X)`

Typedefs

- `typedef unsigned int basics::Id`

7.28.1 Macro Definition Documentation**7.28.1.1 ID**

```
#define ID(  
    X)
```

Value:

`FNV(X)`

7.29 Id.hpp

[Go to the documentation of this file.](#)

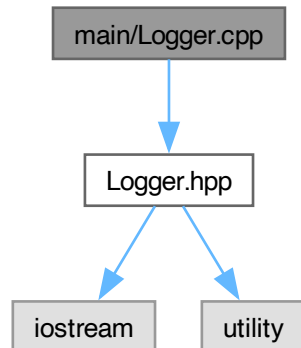
```
00001 /*
00002  * ID
00003  * Copyright © 2017+ Ángel Rodríguez Ballesteros
00004  *
00005  * Distributed under the Boost Software License, version 1.0
00006  * See documents/LICENSE.TXT or www.boost.org/LICENSE_1_0.txt
00007  *
00008  * angel.rodriguez@esne.edu
00009  *
00010  * C1712211447
00011  */
00012
00013 #ifndef BASICS_ID_HEADER
00014 #define BASICS_ID_HEADER
00015
00016     #include "fnv.hpp"
00017
00018     #define ID(X) FNV(X)
00019
00020     namespace basics
00021     {
00022
00023         typedef unsigned int Id;
00024
00025     }
00026
00027 #endif
```

7.30 main/Logger.cpp File Reference

This file implements the Logger class, which provides a singleton logger for the [Ragot](#) engine.

```
#include "Logger.hpp"
```

Include dependency graph for Logger.cpp:



Namespaces

- namespace [Ragot](#)

Variables

- [Logger](#) & [Ragot::logger](#) = [Logger::instance\(\)](#)

7.30.1 Detailed Description

This file implements the `Logger` class, which provides a singleton logger for the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The `Logger` class allows logging messages with different severity levels (`INFO`, `WARNING`, `ERROR`). It supports formatted logging using printf-style format strings and can be used across different platforms. The logger can be configured to set the log level, and it uses the ESP-IDF logging system on ESP platforms.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

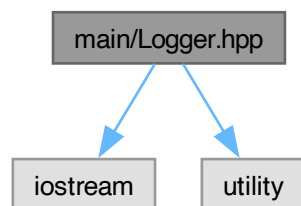
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

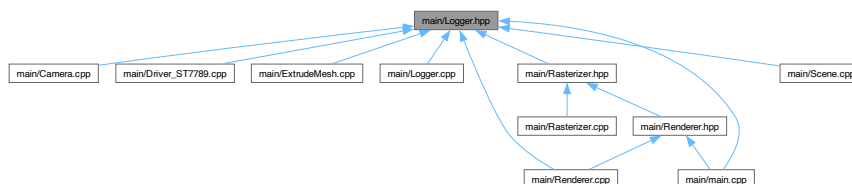
7.31 main/Logger.hpp File Reference

This file implements the Logger class, which provides a singleton logger for the [Ragot](#) engine.

```
#include <iostream>
#include <utility>
Include dependency graph for Logger.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Logger](#)
Singleton logger class for the [Ragot](#) engine.

Namespaces

- namespace [Ragot](#)

7.31.1 Detailed Description

This file implements the Logger class, which provides a singleton logger for the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Logger class allows logging messages with different severity levels (INFO, WARNING, ERROR). It supports formatted logging using printf-style format strings and can be used across different platforms. The logger can be configured to set the log level, and it uses the ESP-IDF logging system on ESP platforms.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.32 Logger.hpp

[Go to the documentation of this file.](#)

```

00001
00034
00035
00036 #pragma once
00037
00038 #include <iostream>
00039 #include <utility>
00040
00041 #if ESP_PLATFORM == 1
00042 #include "esp_log.h"
00043 #endif
00044
00045 namespace Ragot
00046 {
00054     class Logger
00055     {
00056     public:
00057
00065         static Logger &instance()
00066         {
00067             static Logger instance;
00068             return instance;
00069         }
00070
00081         template < typename... Args >
00082         void Log (const char * TAG, uint8_t level, const char * fmt, Args... args)
00083         {
00084             if (level > logLevel)
00085                 return;
00086
00087             #if ESP_PLATFORM == 1
00088                 esp_log_write ((esp_log_level_t)level, TAG, fmt, std::forward<Args>(args)...);
00089             #else
00090                 // 1) Calculamos el tamaño del buffer necesario
00091                 int needed = std::snprintf(nullptr, 0, fmt, std::forward<Args>(args)... ) + 1;
00092                 std::vector<char> buffer(needed);
00093
00094                 // 2) Rellenamos el buffer con el texto formateado
00095                 std::snprintf(buffer.data(), buffer.size(), fmt, std::forward<Args>(args)...);
00096
00097                 // 3) Lo imprimimos
00098                 std::cout << "[" << TAG << "]: " << buffer.data() << std::endl;
00099             #endif
00100         }
00101
00102     private:
00103
00109         Logger () = default;
00110
00116         ~Logger () = default;
00117
00121         Logger (const Logger & ) = delete;
00122
00128         Logger (const Logger &&) = delete;
00129
00137         Logger & operator = (const Logger & ) = delete;
00138
00146         Logger & operator = (const Logger &&) = delete;
00147
00148     private:
00149
00156         uint8_t logLevel = 0; // 0 = INFO, 1 = WARNING, 2 = ERROR
00157
00158     public:
00159
00168         void setLogLevel (uint8_t level)
00169         {
00170             logLevel = level;
00171             #if ESP_PLATFORM == 1
00172                 esp_log_level_set("...", (esp_log_level_t)level);
00173             #endif
00174         }
00175
00183         uint8_t getLogLevel () const
00184         {
00185             return logLevel;
00186         }
00187
00188     };
00189
00190     extern Logger & logger;
00191 }

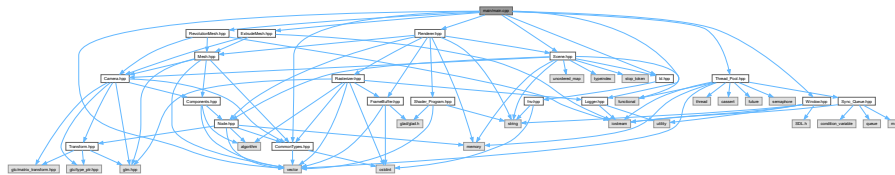
```

7.33 main/main.cpp File Reference

This file contains the main function for the [Ragot](#) engine, which initializes the renderer and scene, and starts the main rendering loop.

```
#include "Renderer.hpp"
#include <vector>
#include "CommonTypes.hpp"
#include "Scene.hpp"
#include "RevolutionMesh.hpp"
#include "ExtrudeMesh.hpp"
#include "Id.hpp"
#include "Logger.hpp"
#include "Thread_Pool.hpp"
#include "Window.hpp"
```

Include dependency graph for main.cpp:



Functions

- void [main_loop](#) ([Renderer](#) &renderer, [Scene](#) &scene, [Window](#) &window)
Main loop for the [Ragot](#) engine on non-ESP platforms.
- int [main](#) (int argc, char *argv[])
Main function for the [Ragot](#) engine.

Variables

- static const char * [MAIN_TAG](#) = "Main"

7.33.1 Detailed Description

This file contains the main function for the [Ragot](#) engine, which initializes the renderer and scene, and starts the main rendering loop.

Author

Andrés [Ragot](#) (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.33.2 Function Documentation**7.33.2.1 main()**

```
int main (
    int argc,
    char * argv[])
```

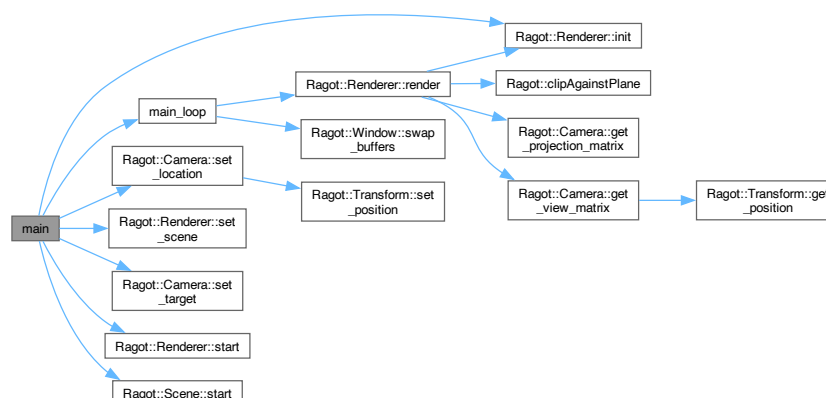
Main function for the [Ragot](#) engine.

This function initializes the logger, sets up the scene, creates the renderer, and enters the main rendering loop.

Returns

int Exit status of the program.

Here is the call graph for this function:



7.33.2.2 main_loop()

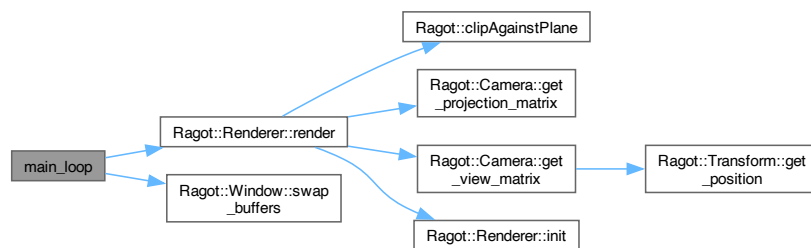
```
void main_loop (
    Renderer & renderer,
    Scene & scene,
    Window & window)
```

Main loop for the [Ragot](#) engine on non-ESP platforms.

Parameters

<i>renderer</i>	Renderer instance to handle rendering.
<i>scene</i>	Scene instance containing the 3D objects.
<i>window</i>	Window instance for displaying the rendered output.

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.3 Variable Documentation

7.33.3.1 MAIN_TAG

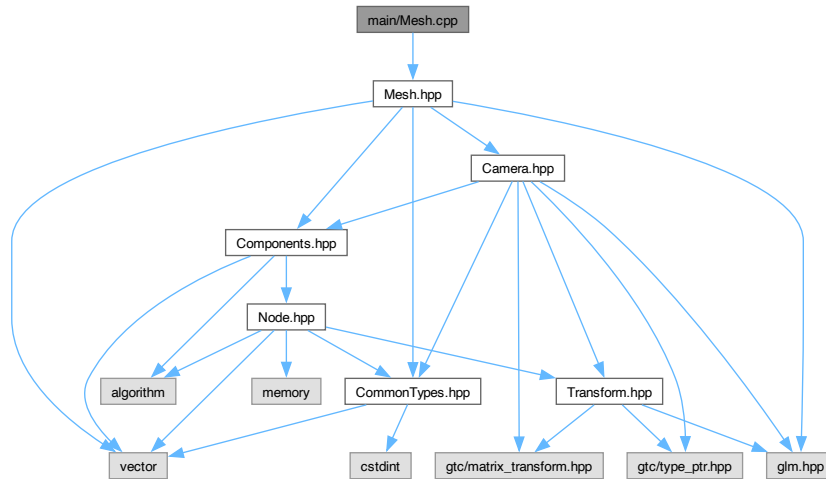
```
const char* MAIN_TAG = "Main" [static]
```

7.34 main/Mesh.cpp File Reference

This file implements the Mesh class, which represents a 3D mesh in the [Ragot](#) engine.

```
#include "Mesh.hpp"
```

Include dependency graph for Mesh.cpp:



Namespaces

- namespace [Ragot](#)

7.34.1 Detailed Description

This file implements the Mesh class, which represents a 3D mesh in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Mesh class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

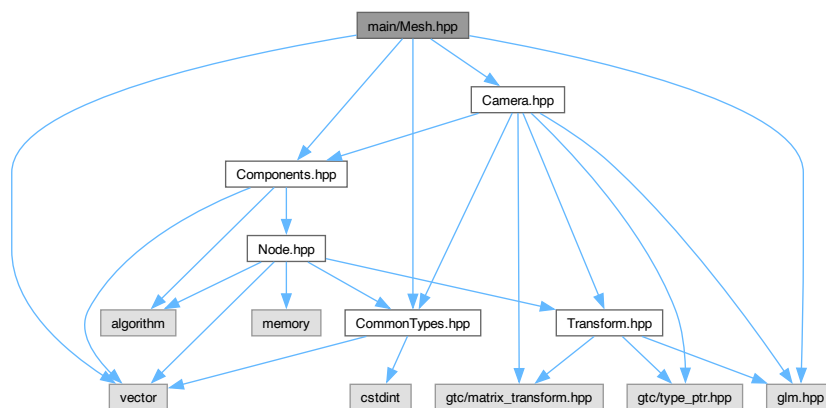
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

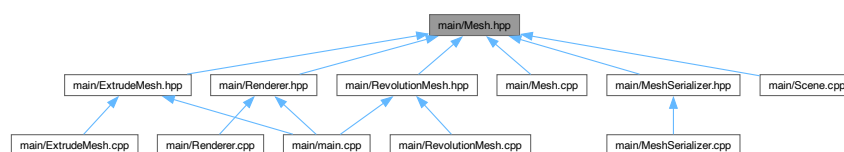
7.35 main/Mesh.hpp File Reference

This file implements the Mesh class, which represents a 3D mesh in the [Ragot](#) engine.

```
#include "CommonTypes.hpp"
#include "Camera.hpp"
#include <vector>
#include "Components.hpp"
#include <glm.hpp>
Include dependency graph for Mesh.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Mesh](#)
Represents a 3D mesh in the [Ragot](#) engine.

Namespaces

- namespace [Ragot](#)

7.35.1 Detailed Description

This file implements the Mesh class, which represents a 3D mesh in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Mesh class is a base class for creating 3D meshes with vertices and faces. It provides methods to generate vertices and faces, apply transformations, and manage mesh information. The class also includes methods for setting and getting the color of the mesh.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.36 Mesh.hpp

[Go to the documentation of this file.](#)

```

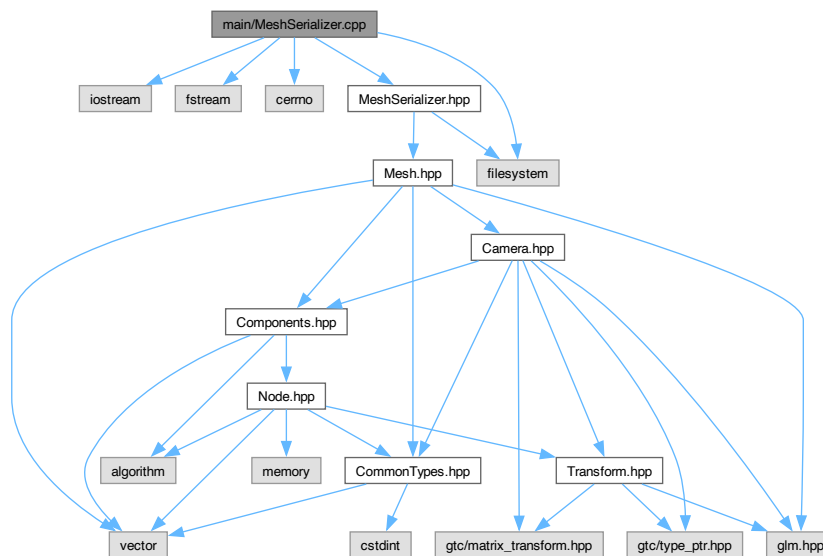
00001
00034
00035 #pragma once
00036
00037 #include "CommonTypes.hpp"
00038 #include "Camera.hpp"
00039 #include <vector>
00040 #include "Components.hpp"
00041 #include <glm.hpp>
00042
00043 namespace Ragot
00044 {
00053     class Mesh : public Component
00054     {
00055     protected:
00056
00057         mesh_info_t mesh_info;
00058         uint16_t color = 0xFFFF;
00059
00060         std::vector < glm::fvec4 > vertices;
00061         std::vector < face_t > faces;
00062
00063         int slices = 16;
00064
00065     public:
00066         Mesh() = delete;
00070
00076         virtual ~Mesh() = default;
00077
00085         Mesh(mesh_info_t & mesh_info);
00086
00093         virtual void generate_vertices() = 0;
00094
00101         virtual void generate_faces() = 0;
00102
00108         const std::vector < glm::fvec4 > & get_vertices() const { return vertices; }
00109
00115         const std::vector < face_t > & get_faces() const { return faces; }
00116
00117
00123         const size_t get_total_vertices() const { return vertices.size(); }
00124
00130         size_t get_total_vertices() { return vertices.size(); }
00131
00135         void recalculate()
00136         {
00137             vertices.clear();
00138             faces.clear();
00139
00140             generate_vertices();
00141             generate_faces();
00142
00143             apply_transform_to_vertices();
00144         }
00145
00152         void apply_transform_to_vertices()
00153         {
00154             glm::mat4 M = get_transform_matrix();
00155             for (auto & v : vertices)
00156             {
00157                 v = M * v;
00158             }
00159         }
00160
00168         void set_color(uint16_t new_color)
00169         {
00170             color = new_color;
00171         }
00172
00180         uint16_t get_color() const
00181         {
00182             return color;
00183         }
00184     };
00185 }
```

7.37 main/MeshSerializer.cpp File Reference

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

```
#include <iostream>
#include <fstream>
#include <cerrno>
#include "MeshSerializer.hpp"
#include <filesystem>

Include dependency graph for MeshSerializer.cpp:
```



Namespaces

- namespace [Ragot](#)

Variables

- [MeshSerializer](#) & [Ragot::serializer](#) = [MeshSerializer::instance\(\)](#)

7.37.1 Detailed Description

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

Author

Andrés [Ragot](#) (github.com/andresragot)

The MeshSerializer class allows saving a Mesh object to an OBJ file, which is a common format for 3D models. It handles the serialization of vertices and faces, ensuring that the data is written in a format compatible with OBJ files. The class is designed to be used in graphics applications where exporting 3D models is required.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

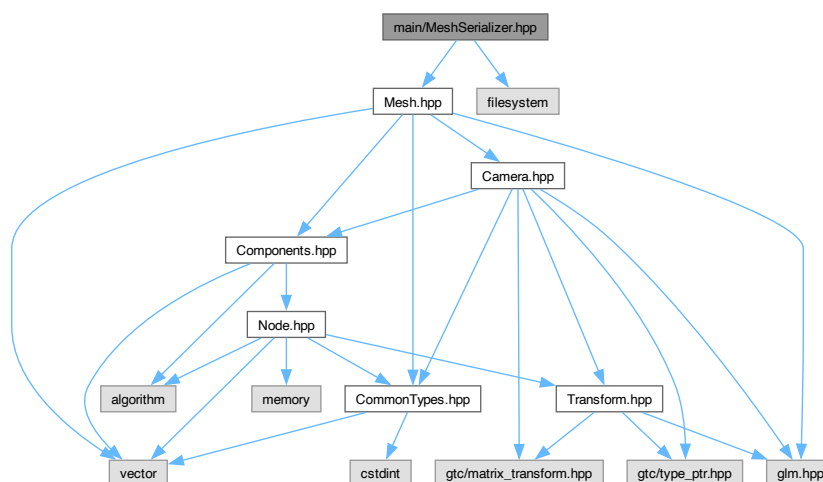
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.38 main/MeshSerializer.hpp File Reference

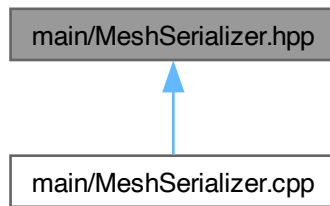
This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

```
#include "Mesh.hpp"
#include <filesystem>
```

Include dependency graph for MeshSerializer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::MeshSerializer](#)
Singleton class to serialize [Mesh](#) objects to OBJ file format.

Namespaces

- namespace [Ragot](#)

7.38.1 Detailed Description

This file implements the MeshSerializer class, which provides methods to serialize a Mesh object to an OBJ file format.

Author

Andrés [Ragot](#) (github.com/andresragot)

The MeshSerializer class allows saving a Mesh object to an OBJ file, which is a common format for 3D models. It handles the serialization of vertices and faces, ensuring that the data is written in a format compatible with OBJ files. The class is designed to be used in graphics applications where exporting 3D models is required.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.39 MeshSerializer.hpp

[Go to the documentation of this file.](#)

```

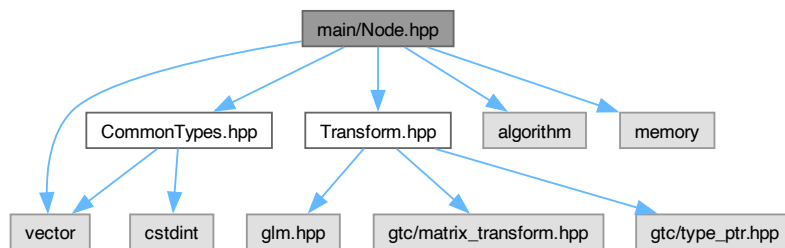
00001
00034 #pragma once
00035
00036 #include "Mesh.hpp"
00037 #include <filesystem>
00038
00039 namespace Ragot
00040 {
00041     // Vamos a hacer un Singleton.
00042     class MeshSerializer
00043     {
00044     public:
00056         static MeshSerializer & instance()
00057         {
00058             static MeshSerializer serializer;
00059             return serializer;
00060         }
00061
00062     private:
00063
00069         MeshSerializer () = default;
00070
00074         MeshSerializer (const MeshSerializer &) = delete;
00075
00081         MeshSerializer (const MeshSerializer &&) = delete;
00082
00090         MeshSerializer & operator = (const MeshSerializer &) = delete;
00091
00099         MeshSerializer & operator = (const MeshSerializer &&) = delete;
00100
00101     public:
00102
00112         bool save_to_obj (const Mesh & mesh, const std::filesystem::path & path);
00113
00114     };
00115
00116     extern MeshSerializer & serializer;
00117 }

```

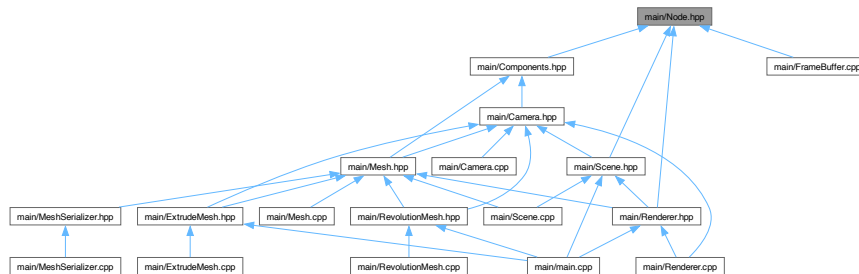
7.40 main/Node.hpp File Reference

This file implements the Node class, which represents a node in a scene graph for 3D rendering.

```
#include "CommonTypes.hpp"
#include <vector>
#include "Transform.hpp"
#include <algorithm>
#include <memory>
Include dependency graph for Node.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Node](#)
Represents a node in a scene graph for 3D rendering.

Namespaces

- namespace [Ragot](#)

7.40.1 Detailed Description

This file implements the Node class, which represents a node in a scene graph for 3D rendering.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

The Node class is a part of the [Ragot](#) engine and extends the Transform class to include child nodes. It allows for hierarchical transformations and management of child nodes, enabling complex scene structures. The class provides methods to add and remove child nodes, retrieve the list of children, and compute the transformation matrix.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.41 Node.hpp

[Go to the documentation of this file.](#)

```

00001
00035
00036 #pragma once
00037 #include "CommonTypes.hpp"
00038 #include <vector>
00039 #include "Transform.hpp"
00040 #include <algorithm>
00041 #include <memory>
00042
00043 namespace Ragot
00044 {
00045     using glm::mat4;
00046
00054     class Node : public Transform
00055     {
00056
00057     protected:
00058
00059         std::vector < std::shared_ptr < Node > > children;
00060         Node * parent = nullptr;
00061
00062     public:
00063         Node () = default;
00064
00065         virtual ~Node () = default;
00066
00067         Node (const Node &) = delete;
00068
00069         Node (const Node &&) = delete;
00070
00071         Node & operator = (const Node &) = delete;
00072
00073         Node & operator = (const Node &&) = delete;
00074
00075     public:
00076
00077         void add_child (std::shared_ptr < Node > child)
00078         {
00079             if (child)
00080             {
00081                 children.emplace_back(child);
00082                 child->parent = this;
00083                 child->dirty = true;
00084                 dirty = true;
00085             }
00086         }
00087
00088         void remove_child (std::shared_ptr < Node > child)
00089         {
00090             if (child)
00091             {
00092                 auto it = std::remove(children.begin(), children.end(), child);
00093                 if (it != children.end())
00094                 {
00095                     children.erase(it, children.end());
00096                     child->parent = nullptr;
00097                     child->dirty = true;
00098                 }
00099             }
00100         }
00101
00102         const std::vector< std::shared_ptr < Node > > & get_children() const { return children; }
00103
00104         mat4 get_transform_matrix() override
00105         {
00106             mat4 transform_matrix = Transform::get_transform_matrix();
00107             if (parent)
00108                 transform_matrix = parent->get_transform_matrix() * transform_matrix;
00109
00110             return transform_matrix;
00111         }
00112     };
00113 }
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162

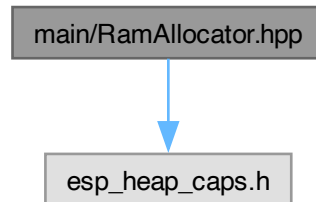
```

7.42 main/RamAllocator.hpp File Reference

This file implements a custom memory allocator for PSRAM in the [Ragot](#) engine.

```
#include "esp_heap_caps.h"
```

Include dependency graph for RamAllocator.hpp:



Classes

- class [Ragot::PSRAMAllocator< T, Flag >](#)
Custom memory allocator for PSRAM.
- struct [Ragot::PSRAMAllocator< T, Flag >::rebind< U >](#)
Rebinds the allocator to a different type. This struct allows the [PSRAMAllocator](#) to be used with different types while maintaining the same allocation flags.

Namespaces

- namespace [Ragot](#)

Functions

- template<typename T, uint16_t F1, typename U, uint16_t F2>
bool [Ragot::operator==](#) (const [PSRAMAllocator](#)< T, F1 > &, const [PSRAMAllocator](#)< U, F2 > &)
Equality operator for [PSRAMAllocator](#).
- template<typename T, uint16_t F1, typename U, uint16_t F2>
bool [Ragot::operator!=](#) (const [PSRAMAllocator](#)< T, F1 > &a, const [PSRAMAllocator](#)< U, F2 > &b)
Inequality operator for [PSRAMAllocator](#).

7.42.1 Detailed Description

This file implements a custom memory allocator for PSRAM in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The [PSRAMAllocator](#) class provides a way to allocate and deallocate memory in PSRAM with specific flags. It is designed to be used with standard containers like `std::vector`, allowing for efficient memory management in embedded systems. The allocator uses ESP-IDF's `heap_caps_malloc` and `heap_caps_free` functions to manage memory.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.43 RamAllocator.hpp

[Go to the documentation of this file.](#)

```

00001
00034
00035 #pragma once
00036 #include "esp_heap_caps.h"
00037
00038 namespace Ragot
00039 {
00040
00048     template <typename T, uint16_t Flag>
00049     class PSRAMAllocator
00050     {
00051     public:
00052         using value_type      = T;
00053         using pointer          = T*;
00054         using size_type        = std::size_t;
00055
00063         template <typename U>
00064         struct rebind { using other = PSRAMAllocator<U, Flag>; };
00065
00071         PSRAMAllocator() noexcept {}
00072
00082         template <typename U, uint16_t F2>
00083         PSRAMAllocator(const PSRAMAllocator<U, F2>&) noexcept {}
00084
00094         T* allocate(size_type n)
00095         {
00096             T* p = static_cast<T*>(heap_caps_malloc(n * sizeof(T), Flag));
00097             if (not p) throw std::bad_alloc();
00098             return p;
00099         }
00100
00110         void deallocate(T* p, size_type) noexcept
00111         {

```

```

00112         heap_caps_free(p);
00113     }
00114 };
00115
00129 template <typename T, uint16_t F1, typename U, uint16_t F2>
00130 bool operator==(const PSRAMAllocator<T, F1>&, const PSRAMAllocator<U, F2>&)
00131 {
00132     return F1 == F2;
00133 }
00134
00148 template <typename T, uint16_t F1, typename U, uint16_t F2>
00149 bool operator!=(const PSRAMAllocator<T, F1>& a, const PSRAMAllocator<U, F2>& b)
00150 {
00151     return !(a == b);
00152 }
00153
00154 } // namespace Ragot

```

7.44 main/Rasterizer.cpp File Reference

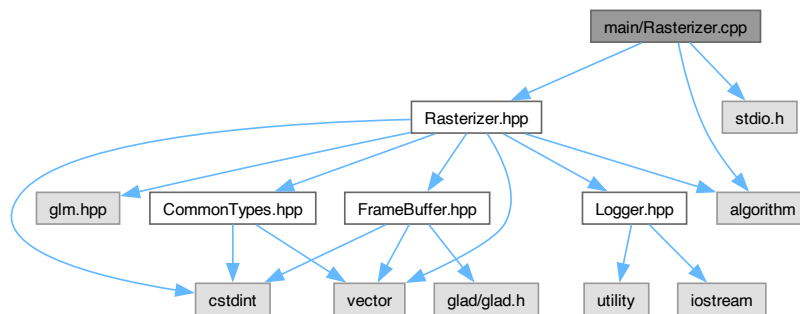
Implementation of the Rasterizer class for rendering polygons in a frame buffer.

```

#include "Rasterizer.hpp"
#include <stdio.h>
#include <algorithm>

```

Include dependency graph for Rasterizer.cpp:



Namespaces

- namespace [Ragot](#)

Variables

- template<class COLOR_BUFFER_TYPE>
int [Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache0](#) [1024]
- template<class COLOR_BUFFER_TYPE>
int [Ragot::Rasterizer< COLOR_BUFFER_TYPE >::offset_cache1](#) [1024]
- template<class COLOR_BUFFER_TYPE>
int [Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache0](#) [1024]
- template<class COLOR_BUFFER_TYPE>
int [Ragot::Rasterizer< COLOR_BUFFER_TYPE >::z_cache1](#) [1024]

7.44.1 Detailed Description

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](https://github.com/andresragot)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

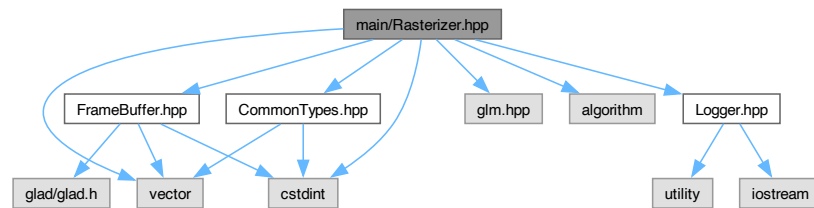
7.45 main/Rasterizer.hpp File Reference

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

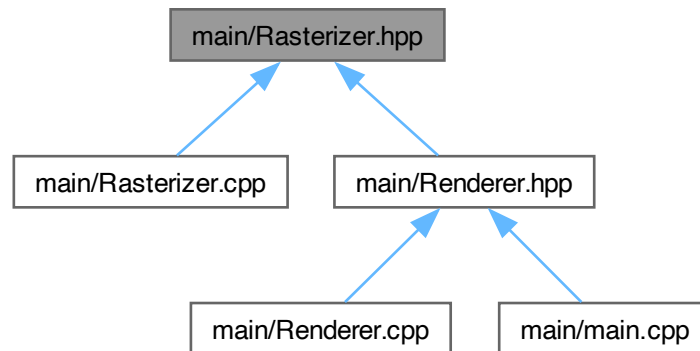
```
#include "FrameBuffer.hpp"
#include "CommonTypes.hpp"
#include <glm.hpp>
#include <cstdint>
#include <vector>
#include <algorithm>
```

```
#include "Logger.hpp"
```

Include dependency graph for Rasterizer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Rasterizer< Color >](#)
Class for rasterizing polygons in a frame buffer.

Namespaces

- namespace [Ragot](#)

7.45.1 Detailed Description

Implementation of the Rasterizer class for rendering polygons in a frame buffer.

Author

Andrés [Ragot](#) (github.com/andresragot)

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.46 Rasterizer.hpp

[Go to the documentation of this file.](#)

```

00001
00032
00033 #pragma once
00034 #include "FrameBuffer.hpp"
00035 #include "CommonTypes.hpp"
00036 #include <glm.hpp>
00037 #include <stdint>
00038 #include <vector>
00039 #include <algorithm>
00040 #include "Logger.hpp"
00041
00042 namespace Ragot
00043 {
00053     template <typename Color>
00054     class Rasterizer
00055     {
00056
00057     private:
00058         FrameBuffer < Color > & frame_buffer;
00059
00060         static int offset_cache0 [1024];
00061         static int offset_cache1 [1024];
00062 #ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00063
00064         static int z_cache0 [1024];
00065         static int z_cache1 [1024];
00066
00067         std::vector < int > z_buffer;
00068 #endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00069
00070
00071         Color color;
00072         Color clear_color;

```

```

00073
00074     static constexpr const char * RASTER_TAG = "Rasterizer";
00075
00076     public:
00077
00085         Rasterizer (FrameBuffer < Color > & frame) : frame_buffer (frame)
00086 #ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00087     , z_buffer(frame.get_width () * frame.get_height ())
00088 #endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00089     {
00090     }
00091
00095         Rasterizer () = default;
00096
00102         ~Rasterizer () = default;
00103
00104     public:
00112         const FrameBuffer < Color > & get_frame_buffer() const
00113         {
00114             return (frame_buffer);
00115         }
00116
00124         void set_color (const Color & new_color)
00125         {
00126             color = new_color;
00127             // frame_buffer.set_color (new_color);
00128         }
00129
00137         void clear ()
00138         {
00139             logger.Log (RASTER_TAG, 3, "Limpiando framebuffer");
00140             frame_buffer.clear_buffer();
00141
00142 #ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00143             for (int * z = z_buffer.data(), * end = z + z_buffer.size(); z != end; ++z)
00144             {
00145                 * z = std::numeric_limits< int >::max ();
00146             }
00147 #endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00148         }
00149
00159         void fill_convex_polygon ( const glm::ivec4 * const vertices,
00160                                 const int          * const indices_begin,
00161                                 const int          * const indices_end
00162                                 );
00163
00172         void fill_convex_polygon ( const glm::ivec4 * const vertices,
00173                                 const face_t      * const face
00174                                 );
00175
00176 #ifndef CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00177
00187         void fill_convex_polygon_z_buffer ( const glm::ivec4 * const vertices,
00188                                           const face_t      * const face
00189                                           );
00190
00199         void fill_convex_polygon_z_buffer (
00200                                           const glm::ivec4 * const vertices,
00201                                           const int          * const indices_begin,
00202                                           const int          * const indices_end
00203                                           );
00204 #endif // CONFIG_GRAPHICS_PAINTER_ALGO_ENABLED
00205
00206         // Logs debug para rasterizado
00207         bool debug_enabled = true;
00208
00209     private:
00210
00225         template < typename VALUE_TYPE, size_t SHIFT >
00226         void interpolate (int * cache, int v0, int v1, int y_min, int y_max);
00227
00239         template < unsigned COLOR_SIZE >
00240         void fill_row (Color * start, unsigned left_offset, unsigned right_offset, const Color &
00241 color)
00242         {
00243             std::fill_n (start + left_offset, right_offset - left_offset, color);
00244         }
00245         // dentro de Rasterizer<COLOR> o en un header común
00261         template <unsigned COLOR_SIZE>
00262         void fill_row_zbuffer (
00263             Color *      start,          // puntero al primer píxel de la scanline
00264             int *        zbuffer,        // puntero al primer elemento del Z-buffer
00265             unsigned     left_offset,    // offset inicial (inclusive)
00266             unsigned     right_offset,   // offset final (exclusive)
00267             int          z_start,        // profundidad en left_offset
00268             int          dz,             // incremento de z por píxel

```



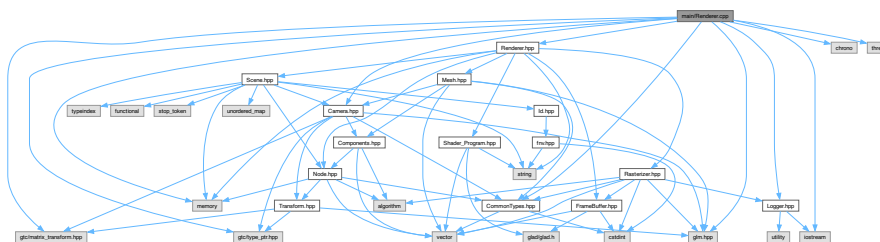
```
00269         const Color & color // color a pintar
00270     )
00271     {
00272         unsigned length = right_offset - left_offset;
00273         Color * pix = start + left_offset;
00274         int * zb = zbuffer + left_offset;
00275
00276         // Recorremos de 0 a length-1
00277         for (unsigned i = 0; i < length; ++i, ++pix, ++zb, z_start += dz)
00278         {
00279             if (z_start < *zb)
00280             {
00281                 *pix = color;
00282                 *zb = z_start;
00283             }
00284         }
00285     }
00286 };
00287
00288 template class Rasterizer<RGB565>;
00289 }
```

7.47 main/Renderer.cpp File Reference

Implementation of the Renderer class for rendering scenes in the [Ragot](#) engine.

```
#include "Renderer.hpp"
#include "Camera.hpp"
#include <iostream>
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
#include "CommonTypes.hpp"
#include "Logger.hpp"
#include <memory>
#include <chrono>
#include <thread>
```

Include dependency graph for Renderer.cpp:



Namespaces

- namespace Ragot

Typedefs

- using Ragot::Matrix4x4 = glm::mat4

Functions

- `template<typename Inside, typename Intersect>`
`static std::vector< glm::fvec4 > Ragot::clipAgainstPlane (const std::vector< glm::fvec4 > &in, Inside inside, Intersect intersect)`

Variables

- `static const char * Ragot::RENDERER_TAG = "Renderer"`
- `static const char * Ragot::MAIN_TAG = "Main"`

7.47.1 Detailed Description

Implementation of the `Renderer` class for rendering scenes in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The `Renderer` class is responsible for rendering 3D scenes using a rasterization approach.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.48.1 Detailed Description

Implementation of the `Renderer` class for rendering scenes in the [Ragot](#) engine.

Author

Andrés [Ragot](#) (github.com/andresragot)

The `Renderer` class is responsible for rendering 3D scenes using a rasterization approach.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.49 `Renderer.hpp`

[Go to the documentation of this file.](#)

```
00001
00032
00033 #pragma once
00034
00035 #include "Mesh.hpp"
00036 #include "FrameBuffer.hpp"
00037 #include "Rasterizer.hpp"
00038 #include "Node.hpp"
00039 #include "Scene.hpp"
00040 #include <memory>
00041 #include <string>
00042
00043 #if ESP_PLATFORM == 1
00044 #ifdef CONFIG_IDF_TARGET_ESP32P4
00045 #include "driver_ek79007.hpp"
```

```

00046 #elif CONFIG_IDF_TARGET_ESP32S3
00047 #include "Driver_ST7789.hpp"
00048 #endif
00049 #else
00050 #include "Shader_Program.hpp"
00051 #endif
00052
00053 namespace Ragot
00054 {
00062     class Renderer
00063     {
00064     private:
00065
00066         float accumulated_time = 0.f;
00067         size_t iterations = 0;
00068         static constexpr size_t number_of_iterations = 10000000000000000;
00069
00070         #if ESP_PLATFORM == 1
00071         #ifdef CONFIG_IDF_TARGET_ESP32P4
00072         DriverEK79007 driver;
00073         #elif CONFIG_IDF_TARGET_ESP32S3
00074         Driver_ST7789 driver;
00075         #endif
00076         #else
00077         std::unique_ptr < Shader_Program > quadShader = nullptr;
00078         GLuint quadVAO = 0;
00079         GLuint quadVBO = 0;
00080         GLuint quadEBO = 0;
00081
00082         void initFullScreenQuad();
00083
00084         static const std::string vertex_shader_code;
00085         static const std::string fragment_shader_code;
00086         #endif
00087         FrameBuffer < RGB565 > frame_buffer;
00088         Scene * current_scene = nullptr;
00089         Rasterizer < RGB565 > rasterizer;
00090
00091         unsigned width;
00092         unsigned height;
00093
00094         bool initialized = false;
00095
00096         std::atomic<bool> running = false;
00097
00098     public:
00104         Renderer () = delete;
00105
00114         Renderer (unsigned width, unsigned height);
00115
00121         ~Renderer () = default;
00122
00123         std::vector < glm::fvec4 > transformed_vertices;
00124         std::vector < glm::ivec4 > display_vertices;
00125
00134         void set_scene (Scene * scene) { current_scene = scene; }
00135
00144         void init ();
00145
00153         void render ();
00154
00162         void task_render (std::stop_token stop_token);
00163
00174         bool is_frontface (const glm::fvec4 * const projected_vertices, const face_t * const indices);
00175
00181         void start() { running = true; }
00182
00188         void stop() { running = false; }
00189     };
00190 }
00191
00192

```

7.50 main/RevolutionMesh.cpp File Reference

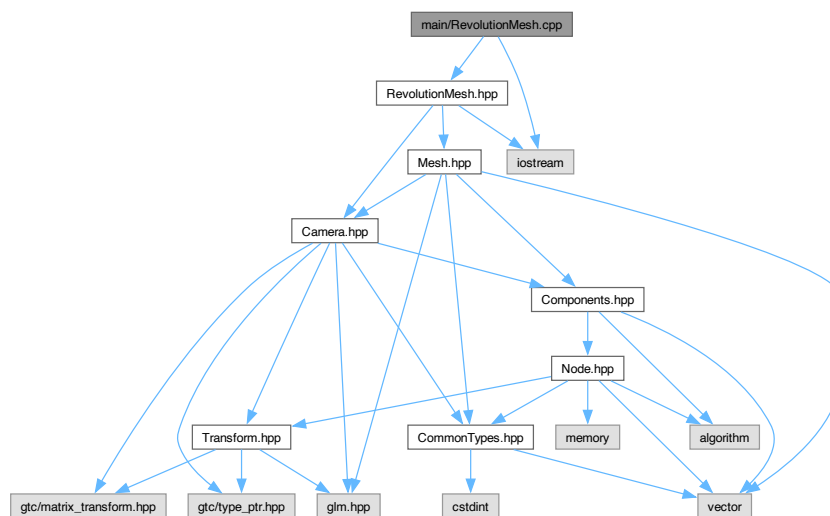
Implementation of the RevolutionMesh class for generating revolution meshes.

```

#include "RevolutionMesh.hpp"
#include <iostream>

```

Include dependency graph for RevolutionMesh.cpp:



Namespaces

- namespace [Ragot](#)

7.50.1 Detailed Description

Implementation of the RevolutionMesh class for generating revolution meshes.

Author

Andrés [Ragot](#) (github.com/andresragot)

The RevolutionMesh class generates a mesh by revolving a 2D profile around an axis.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

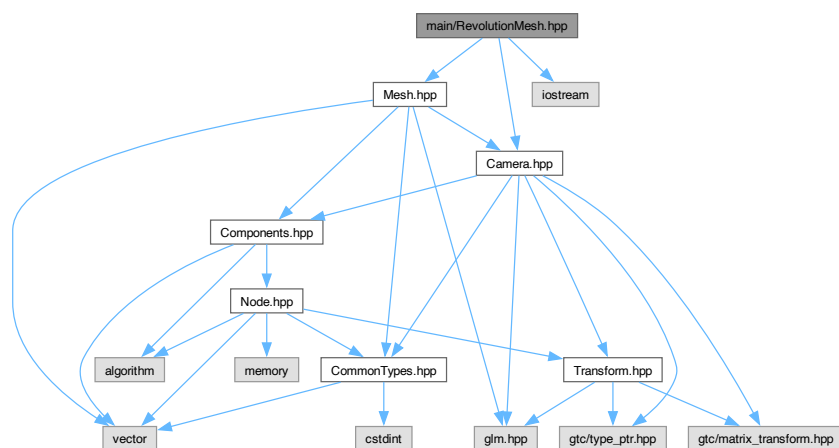
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.51 main/RevolutionMesh.hpp File Reference

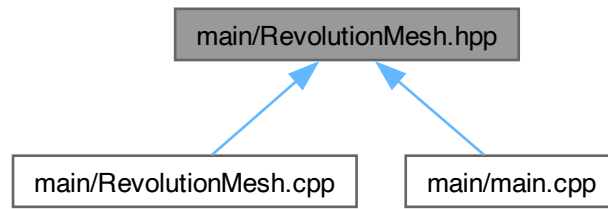
Implementation of the RevolutionMesh class for generating revolution meshes.

```
#include "Mesh.hpp"
#include "Camera.hpp"
#include <iostream>
```

Include dependency graph for RevolutionMesh.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::RevolutionMesh](#)
Class for generating revolution meshes.

Namespaces

- namespace [Ragot](#)

7.51.1 Detailed Description

Implementation of the RevolutionMesh class for generating revolution meshes.

Author

Andrés [Ragot](#) (github.com/andresragot)

The RevolutionMesh class generates a mesh by revolving a 2D profile around an axis.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.52 RevolutionMesh.hpp

[Go to the documentation of this file.](#)

```

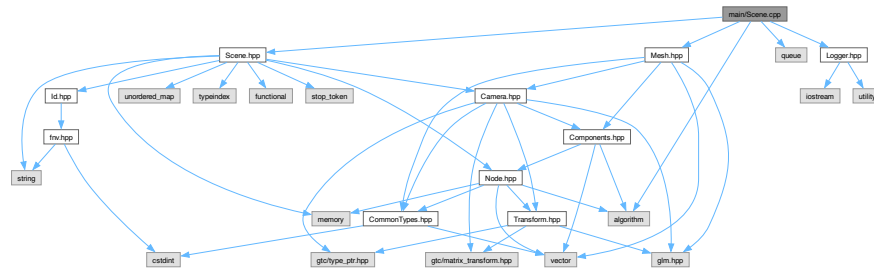
00001
00032
00033 #pragma once
00034
00035 #include "Mesh.hpp"
00036 #include "Camera.hpp"
00037 #include <iostream>
00038
00039 namespace Ragot
00040 {
00041     class RevolutionMesh : public Mesh
00042     {
00043     protected:
00044         const Camera & cam;
00045         bool faces_can_be_quads;
00046         static constexpr float PI = 3.14159265358979323846f;
00047     public:
00048         RevolutionMesh (mesh_info_t & mesh_info, const Camera & cam) : Mesh (mesh_info), cam (cam)
00049         {
00050             faces_can_be_quads = (mesh_info.vertex_amount % 8 == 0 || mesh_info.vertex_amount == 4);
00051             vertices.reserve (mesh_info.coordinates.size() * (slices + 1));
00052             faces.reserve (mesh_info.coordinates.size() * slices);
00053
00054             generate_vertices();
00055             generate_faces();
00056
00057             std::cout << "Revolution Vertices: " << vertices.size() << std::endl;
00058             std::cout << "Revolution Faces: " << faces.size() << std::endl;
00059         }
00060
00061         ~RevolutionMesh() = default;
00062
00063         void generate_vertices () override;
00064
00065         void generate_faces () override;
00066     };
00067 }
00068
00069 }
```

7.53 main/Scene.cpp File Reference

Implementation of the Scene class for managing 3D scenes.

```
#include "Scene.hpp"
#include "Mesh.hpp"
#include <queue>
#include <algorithm>
#include "Logger.hpp"
```

Include dependency graph for Scene.cpp:



Namespaces

- namespace [Ragot](#)

7.53.1 Detailed Description

Implementation of the Scene class for managing 3D scenes.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Scene class provides methods to manage nodes, cameras, and scene traversal.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

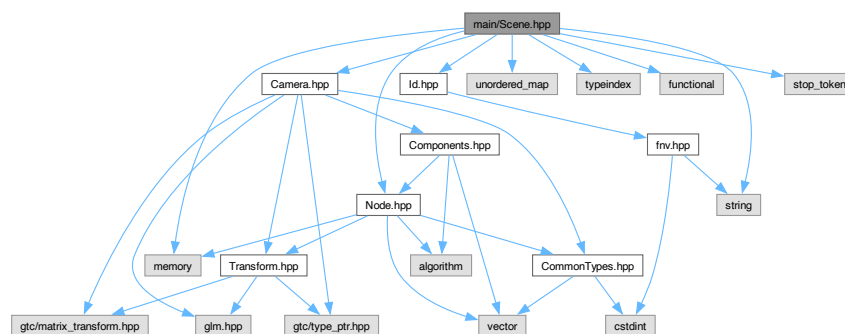
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.54 main/Scene.hpp File Reference

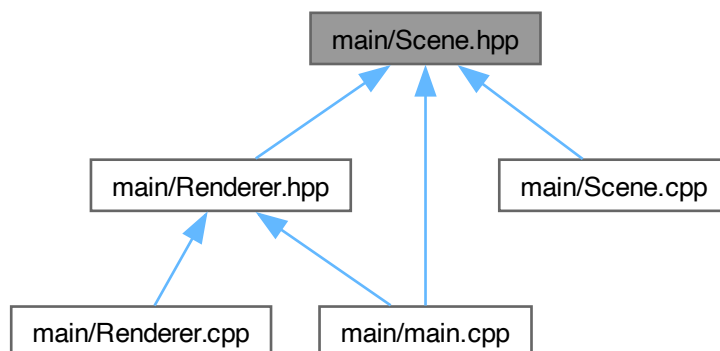
Implementation of the Scene class for managing 3D scenes.

```
#include "Node.hpp"
#include "Camera.hpp"
#include <string>
#include <unordered_map>
#include <typeindex>
#include <functional>
#include <memory>
#include "Id.hpp"
#include <stop_token>
```

Include dependency graph for Scene.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Scene](#)
Class for managing a 3D scene.

Namespaces

- namespace [Ragot](#)

7.54.1 Detailed Description

Implementation of the Scene class for managing 3D scenes.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Scene class provides methods to manage nodes, cameras, and scene traversal.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.55 Scene.hpp

[Go to the documentation of this file.](#)

```

00001
00002
00003 #pragma once
00004
00005 #include "Node.hpp"
00006 #include "Camera.hpp"
00007 #include <string>
00008 #include <unordered_map>
00009 #include <typeindex>
00010 #include <functional>
00011 #include <memory>
00012 #include "Id.hpp"
00013 #include <stop_token>
00014
00015 namespace Ragot
00016 {
00017     class Scene
00018     {
00019     private:
00020         Camera * main_camera = nullptr;
00021         std::shared_ptr < Node > root_node;
00022         std::unordered_map<basics::Id, std::shared_ptr < Node > > named_nodes;
00023         std::atomic<bool> running = false;
00024
00025     public:
00026         Scene();
00027
00028         ~Scene() = default;
00029
00030         Scene (Camera * camera);
00031
00032         void add_node(std::shared_ptr < Node > node, const basics::Id name);
00033
00034         void remove_node(std::shared_ptr < Node > node);
00035
00036         std::shared_ptr < Node > find_node(const basics::Id name);
00037
00038         void set_main_camera(Camera * camera);
00039
00040         Camera * get_main_camera() const { return main_camera; }
00041
00042         void traverse(const std::function<void(std::shared_ptr < Node >) >& callback);
00043
00044         template<typename T>
00045         std::vector<std::shared_ptr < T > > collect_components();
00046
00047         void update(float delta_time);

```

```

00147
00153         std::shared_ptr < Node > get_root() { return root_node; }
00159     const std::shared_ptr < Node > get_root() const { return root_node; }
00160
00166     void start() { running = true; }
00167
00173     void stop() { running = false; }
00174
00175 private:
00185     void task_update (std::stop_token, float delta_time);
00186 };
00187 }

```

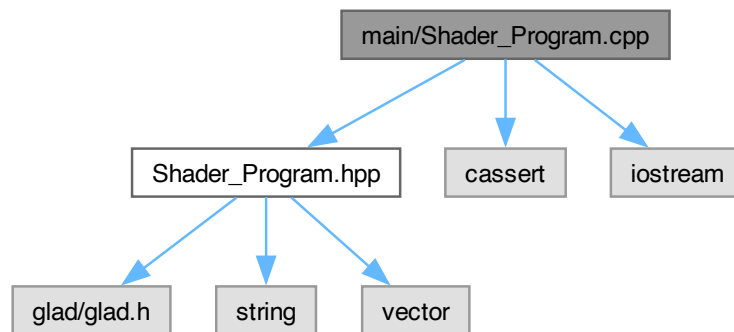
7.56 main/Shader_Program.cpp File Reference

```

#include "Shader_Program.hpp"
#include <cassert>
#include <iostream>

```

Include dependency graph for Shader_Program.cpp:



Namespaces

- namespace [Ragot](#)

7.57 main/Shader_Program.hpp File Reference

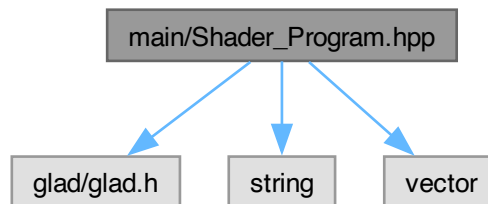
```

#include <glad/glad.h>
#include <string>

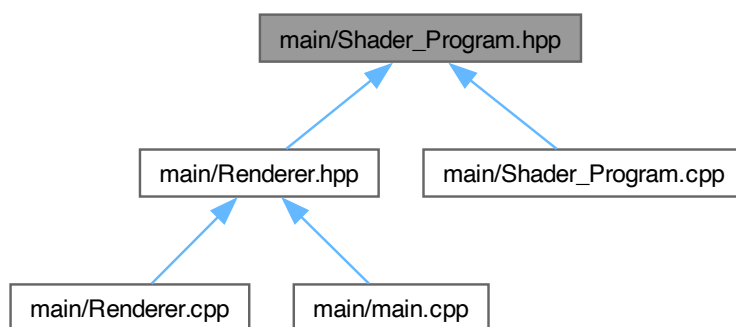
```

```
#include <vector>
```

Include dependency graph for Shader_Program.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Shader](#)
Class for managing an OpenGL shader.
- class [Ragot::Vertex_Shader](#)
Class for managing an OpenGL vertex shader.
- class [Ragot::Fragment_Shader](#)
Class for managing an OpenGL fragment shader.
- class [Ragot::Shader_Program](#)
Class for managing an OpenGL shader program.

Namespaces

- namespace [Ragot](#)

7.58 Shader_Program.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of OpenGL-FinalProject
00003  *
00004  * Developed by Andrés Ragot - github.com/andresragot
00005  *
00006  * MIT License
00007  *
00008  * Copyright (c) 2025 Andrés Ragot
00009  *
00010  * Permission is hereby granted, free of charge, to any person obtaining a copy
00011  * of this software and associated documentation files (the "Software"), to deal
00012  * in the Software without restriction, including without limitation the rights
00013  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00014  * copies of the Software, and to permit persons to whom the Software is
00015  * furnished to do so, subject to the following conditions:
00016  *
00017  * The above copyright notice and this permission notice shall be included in all
00018  * copies or substantial portions of the Software.
00019  *
00020  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00021  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00022  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00023  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00024  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00025  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00026  * SOFTWARE.a
00027  */
00028
00029 #pragma once
00030
00031 #include <glad/glad.h>
00032
00033 #include <string>
00034 #include <vector>
00035
00036 namespace Ragot
00037 {
00038     using namespace std;
00039
00040     class Shader
00041     {
00042     private:
00043         GLuint id;
00044         string error;
00045         bool compilation_succeeded;
00046
00047     protected:
00048         Shader(const vector<string>& source_code, GLenum type);
00049
00050         GLuint compile_shader();
00051
00052         void show_compilation_error();
00053
00054     public:
00055         Shader() = delete;
00056
00057         ~Shader()
00058         {
00059             glDeleteShader(id);
00060         }
00061
00062         GLuint get_id() const
00063         {
00064             return id;
00065         }
00066
00067         string* get_error()
00068         {
00069             return error.empty() ? nullptr : &error;
00070         }
00071
00072         bool is_ok() const
00073         {
00074             return compilation_succeeded;
00075         }
00076     };
00077
00078     class Vertex_Shader : public Shader
00079     {
00080     public:
00081         Vertex_Shader(const vector<string>& source_code) : Shader(source_code, GL_VERTEX_SHADER)
00082         {
00083         }
00084     };

```



```

00120     }
00121 };
00122
00126 class Fragment_Shader : public Shader
00127 {
00128 public:
00133     Fragment_Shader(const vector<string>& source_code) : Shader(source_code, GL_FRAGMENT_SHADER)
00134     {
00135     }
00136 };
00137
00141 class Shader_Program
00142 {
00143 private:
00144     GLuint program_id;
00145
00146 public:
00152     Shader_Program(const vector<string>& source_code_vertex, const vector<string>&
source_code_fragment);
00153
00154     Shader_Program() = delete;
00155
00159     ~Shader_Program()
00160     {
00161         glDeleteProgram(program_id);
00162     }
00163
00167     void use() const
00168     {
00169         glUseProgram(program_id);
00170     }
00171
00176     GLuint get_id() const
00177     {
00178         return program_id;
00179     }
00180
00186     GLuint get_uniform_location(string uniform_name) const
00187     {
00188         return glGetUniformLocation(program_id, uniform_name.c_str());
00189     }
00190
00191 private:
00192     Shader_Program(const Shader_Program&) = delete;
00193     Shader_Program& operator=(const Shader_Program&) = delete;
00194
00200     void initialize(GLuint vertex_shader_id, GLuint fragment_shader_id);
00201
00205     void show_linkage_error();
00206 };
00207 }

```

7.59 main/Sync_Queue.hpp File Reference

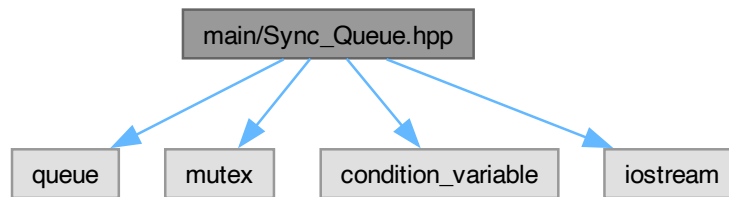
Implementation of a synchronized queue for thread-safe operations.

```

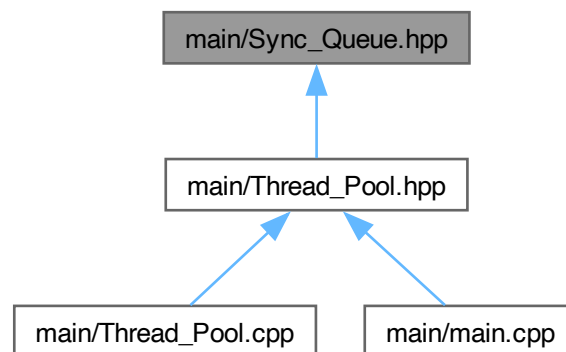
#include <queue>
#include <mutex>
#include <condition_variable>
#include <iostream>

```

Include dependency graph for Sync_Queue.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Sync_Queue< T >](#)
A thread-safe queue implementation.

Namespaces

- namespace [Ragot](#)

7.59.1 Detailed Description

Implementation of a synchronized queue for thread-safe operations.

Author

Andrés [Ragot](https://github.com/andresragot) (github.com/andresragot)

The Sync_Queue class provides a thread-safe queue implementation using mutexes and condition variables.

Version

1.0

Date

2025-06-02

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](https://github.com/andresragot)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.60 Sync_Queue.hpp

[Go to the documentation of this file.](#)

```
00001
00002
00003 #pragma once
00004
00005 #include <queue>
00006 #include <mutex>
00007 #include <condition_variable>
00008 #include <iostream>
00009
00010 namespace Ragot
00011 {
00012     template<typename T>
00013     class Sync_Queue
00014     {
00015     public:
00016         using value_type = T;
00017
00018     public:
00019         Sync_Queue () = default;
00020
00021         ~Sync_Queue ()
00022         {
00023             close ();
00024         }
00025     };
00026 }
```

```

00069     }
00070
00071     Sync_Queue (const Sync_Queue &) = delete;
00072
00073     Sync_Queue & operator=(const Sync_Queue &) = delete;
00074
00075 private:
00076     std::queue<T> queue;
00077     mutable std::mutex mutex;
00078     std::condition_variable condition;
00079     bool closed = false;
00080
00081 public:
00082     std::optional < value_type > get()
00083     {
00084         std::unique_lock<std::mutex> lock(mutex);
00085         condition.wait(lock, [this] { return closed || !queue.empty(); });
00086         if (not closed)
00087         {
00088             value_type value = queue.front();
00089             queue.pop();
00090             return value;
00091         }
00092         return std::nullopt;
00093     }
00094
00095     void push (const value_type & value)
00096     {
00097         std::lock_guard<std::mutex> lock(mutex);
00098         if (not closed)
00099         {
00100             queue.push(value);
00101             condition.notify_one ();
00102         }
00103     }
00104
00105     template < typename ...ARGUMENTS >
00106     void push (ARGUMENTS && ...arguments)
00107     {
00108         std::lock_guard<std::mutex> lock(mutex);
00109         if (not closed)
00110         {
00111             queue.push (std::forward < ARGUMENTS > (arguments)...);
00112             condition.notify_one ();
00113         }
00114     }
00115
00116     template < typename ...ARGUMENTS >
00117     void emplace (ARGUMENTS && ...arguments)
00118     {
00119         std::lock_guard<std::mutex> lock(mutex);
00120         if (not closed)
00121         {
00122             queue.emplace (std::forward<ARGUMENTS> (arguments)...);
00123             condition.notify_one ();
00124         }
00125     }
00126
00127     value_type & back ()
00128     {
00129         std::lock_guard<std::mutex> lock(mutex);
00130         return queue.back();
00131     }
00132
00133     void close ()
00134     {
00135         {
00136             std::lock_guard<std::mutex> lock(mutex);
00137             closed = true;
00138         }
00139         condition.notify_all();
00140     }
00141
00142     void clear ()
00143     {
00144         std::queue < value_type > empty;
00145         std::lock_guard < std::mutex > lock(mutex);
00146         queue.swap(empty);
00147     }
00148
00149     void swap (Sync_Queue & other)
00150     {
00151         std::lock(mutex, other.mutex);
00152         std::lock_guard<std::mutex> lock1(this->mutex, std::adopt_lock);
00153         std::lock_guard<std::mutex> lock2(other.mutex, std::adopt_lock);
00154         queue.swap(other.queue);
00155     }

```

```

00229
00238     bool empty () const
00239     {
00240         std::lock_guard<std::mutex> lock (mutex);
00241         return queue.empty ();
00242     }
00243
00252     size_t size () const
00253     {
00254         std::lock_guard<std::mutex> lock (mutex);
00255         return queue.size ();
00256     }
00257 };
00258 }

```

7.61 main/Thread_Pool.cpp File Reference

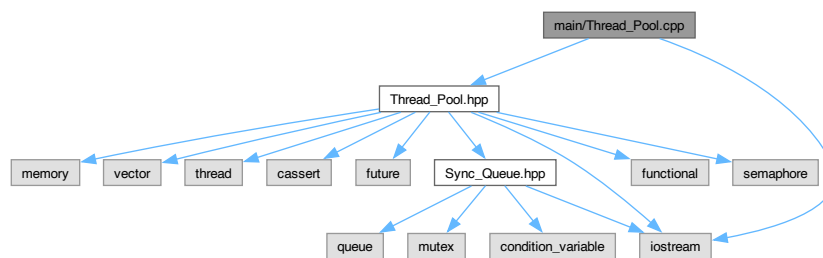
Implementation of the Thread_Pool class for managing a pool of threads.

```

#include "Thread_Pool.hpp"
#include <iostream>

```

Include dependency graph for Thread_Pool.cpp:



Namespaces

- namespace [Ragot](#)

Variables

- [Thread_Pool](#) & [Ragot::thread_pool](#) = [Thread_Pool::instance](#) ()

7.61.1 Detailed Description

Implementation of the Thread_Pool class for managing a pool of threads.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Thread_Pool class provides a way to manage a pool of threads that can execute tasks concurrently.

Version

1.0

Date

2025-06-02

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

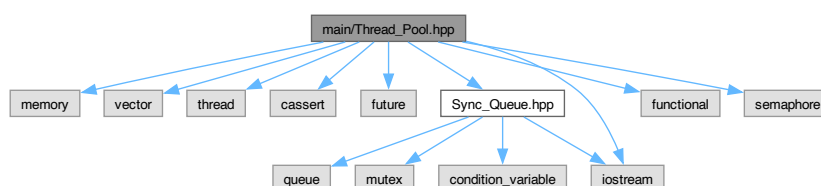
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

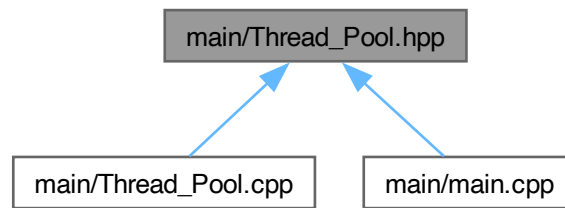
7.62 main/Thread_Pool.hpp File Reference

```
#include <memory>
#include <vector>
#include <thread>
#include <cassert>
#include <future>
#include "Sync_Queue.hpp"
#include <iostream>
#include <functional>
#include <semaphore>
```

Include dependency graph for Thread_Pool.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Thread_Pool](#)
A thread pool for managing concurrent tasks.

Namespaces

- namespace [Ragot](#)

7.63 Thread_Pool.hpp

[Go to the documentation of this file.](#)

```

00001
00032
00033 #pragma once
00034
00035 #include <memory>
00036 #include <vector>
00037 #include <thread>
00038 #include <cassert>
00039 #include <future>
00040 #include "Sync_Queue.hpp"
00041 #include <iostream>
00042 #include <functional>
00043 #include <semaphore>
00044
00045 namespace Ragot
00046 {
00054     class Thread_Pool
00055     {
00056     public:
00057         using Task = std::function < void (std::stop_token) >;
00058         std::binary_semaphore sem_mesh_ready {0};
00059         std::binary_semaphore sem_render_done {0};
00060
00061     private:
00062         Sync_Queue < Task > tasks;
00063
00076         std::vector < std::unique_ptr < std::jthread > > threads;
00077
00078         std::atomic < bool > started;
00079
00080     public:
00089         static Thread_Pool & instance ()
00090         {
00091             static Thread_Pool instance;
00092             return instance;
00093         }
00093
  
```

```

00094
00095     private:
00102         Thread_Pool()
00103         {
00104             auto cores = std::thread::hardware_concurrency();
00105
00106             std::cout << "Thread pool initialized with " << cores << " threads." << std::endl;
00107
00108             if (cores == 0)
00109             {
00110                 cores = 2; // Default to 2 threads if hardware concurrency is not available
00111             }
00112
00113             threads.resize (cores);
00114
00115             started = false;
00116         }
00117
00123     ~Thread_Pool()
00124     {
00125         sem_mesh_ready.release();
00126         sem_render_done.release();
00127
00128         if (started)
00129         {
00130             stop();
00131         }
00132     }
00133
00134     Thread_Pool (Thread_Pool & ) = delete;
00135
00136     Thread_Pool (Thread_Pool &&) = delete;
00137
00138     Thread_Pool & operator = (Thread_Pool & ) = delete;
00139
00140     Thread_Pool & operator = (Thread_Pool &&) = delete;
00141
00142     public:
00143
00144     void start ()
00145     {
00146         assert (not started);
00147
00148         std::cout << "Starting thread pool..." << std::endl;
00149
00150         for (auto & thread : threads)
00151         {
00152             thread = std::make_unique < std::jthread > ( std::bind (&Thread_Pool::thread_function,
00153 this, std::placeholders::_1) );
00154         }
00155
00156         started = true;
00157     }
00158
00159     void stop ()
00160     {
00161         assert (started == true);
00162         started = false;
00163
00164         tasks.close ();
00165         for (auto & thread : threads)
00166         {
00167             thread->request_stop();
00168         }
00169
00170         threads.clear();
00171     }
00172
00173     template<typename F, typename... Args>
00174     std::future < std::invoke_result_t < F, Args... > > submit (F && f, Args && ... args)
00175     {
00176         std::cout << "submit" << std::endl;
00177
00178         using ReturnType = std::invoke_result_t < F, Args... >;
00179
00180         auto task_ptr = std::make_shared < std::packaged_task < ReturnType() > > (
00181             std::bind (std::forward < F > (f), std::forward < Args > (args)...)
00182         );
00183
00184         std::future<ReturnType> res = task_ptr->get_future();
00185
00186         tasks.push ([task_ptr] (std::stop_token) {
00187             (*task_ptr) ();
00188         });
00189     }

```



```

00231
00232         return res;
00233     }
00234
00248     template<typename F, typename... Args>
00249     std::future<std::invoke_result_t<F, std::stop_token, Args...>>
00250     submit_with_stop(F&& f, Args&&... args)
00251     {
00252         std::cout << "submit_with_stop" << std::endl;
00253
00254         using ReturnType = std::invoke_result_t<F, std::stop_token, Args...>;
00255
00256         auto task_ptr = std::make_shared<std::packaged_task<ReturnType (std::stop_token)>>(
00257             std::bind(std::forward<F>(f), std::placeholders::_1, std::forward<Args>(args)...)
00258         );
00259         std::future<ReturnType> res = task_ptr->get_future();
00260
00261         tasks.push([task_ptr](std::stop_token tok) {
00262             (*task_ptr)(tok);
00263         });
00264
00265         return res;
00266     }
00267
00268 private:
00269     void thread_function (std::stop_token);
00270
00271 };
00281
00282 extern Thread_Pool & thread_pool;
00283 }

```

7.64 main/Transform.hpp File Reference

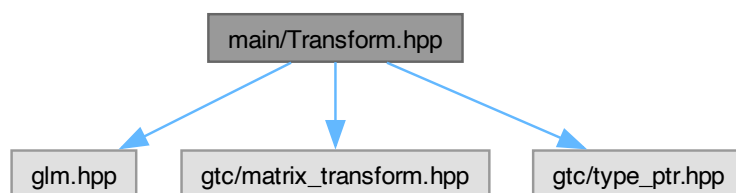
Implementation of the Transform class for 3D transformations.

```

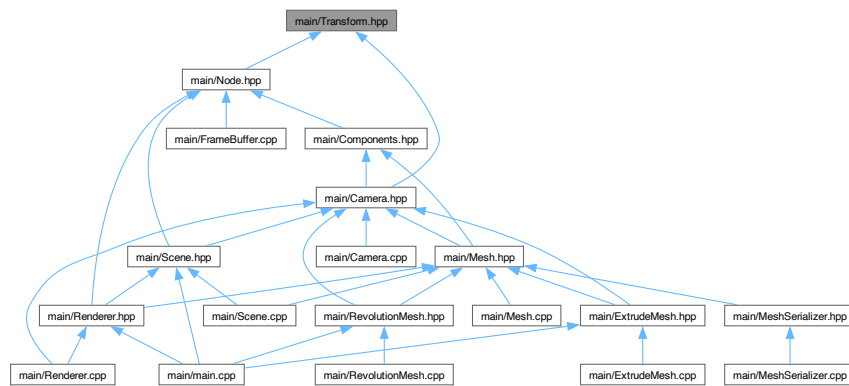
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>

```

Include dependency graph for Transform.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Transform](#)
A class representing a 3D transformation with position, rotation, and scale.

Namespaces

- namespace [Ragot](#)

7.64.1 Detailed Description

Implementation of the Transform class for 3D transformations.

Author

Andrés [Ragot](#) (github.com/andresragot)

The Transform class provides methods to manage position, rotation, and scale of 3D objects.

Version

1.0

Date

2025-06-01

Copyright

Copyright (c) 2025 MIT License

Copyright (c) 2025 Andrés [Ragot](#)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.65 Transform.hpp

[Go to the documentation of this file.](#)

```

00001
00002
00003 #pragma once
00004
00005 #include <glm.hpp>
00006 #include <gtc/matrix_transform.hpp>           // translate, rotate, scale, perspective
00007 #include <gtc/type_ptr.hpp>                 // value_ptr, quat
00008
00009 namespace Ragot
00010 {
00011     using glm::vec3;
00012     using glm::mat4;
00013
00014     class Transform
00015     {
00016     protected:
00017         vec3 position;
00018         vec3 rotation;
00019         vec3 scale;
00020         bool dirty = true;
00021
00022     private:
00023         mat4 transform_matrix;
00024
00025     public:
00026         Transform () : position (0.f), rotation (0.f), scale (1.f) {}
00027
00028         virtual ~Transform () = default;
00029
00030     public:
00031         void set_position (const vec3 & pos)
00032         {
00033             position = pos;
00034             dirty = true;
00035         }
00036
00037         vec3 get_position () const { return position; }
00038
00039         void set_rotation (const vec3 & rot)
00040         {
00041             rotation = rot;
00042             dirty = true;
00043         }
00044
00045         vec3 get_rotation () const { return rotation; }
00046
00047         void rotate (const float angle, const vec3 & axis)

```

```

00121     {
00122         rotation += angle * axis;
00123         if (rotation.x > 360.f) rotation.x -= 360.f;
00124         dirty = true;
00125     }
00126
00132     void set_scale (const vec3 & scale)
00133     {
00134         this->scale = scale;
00135         dirty = true;
00136     }
00137
00143     vec3 get_scale () const { return scale; }
00144
00150     bool is_dirty () const { return dirty; }
00151
00152 public:
00153
00161     virtual mat4 get_transform_matrix ()
00162     {
00163         if (dirty)
00164         {
00165             dirty = false;
00166
00167             mat4 identity(1);
00168             identity = glm::translate(identity, position);
00169             identity = glm::scale (identity, scale);
00170
00171             glm::quat quaternion_rotation = glm::quat (glm::radians (rotation));
00172             identity *= glm::mat4_cast (quaternion_rotation);
00173
00174             transform_matrix = identity;
00175         }
00176
00177         return transform_matrix;
00178     }
00179 };
00180 }

```

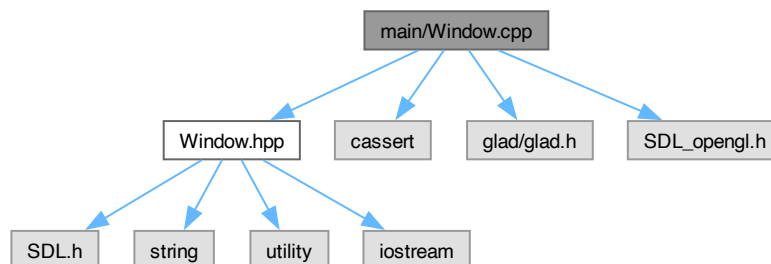
7.66 main/Window.cpp File Reference

```

#include "Window.hpp"
#include <cassert>
#include <glad/glad.h>
#include <SDL_opengl.h>

```

Include dependency graph for Window.cpp:



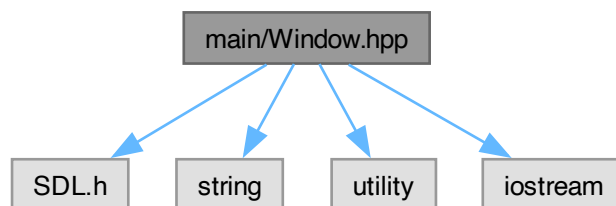
Namespaces

- namespace [Ragot](#)

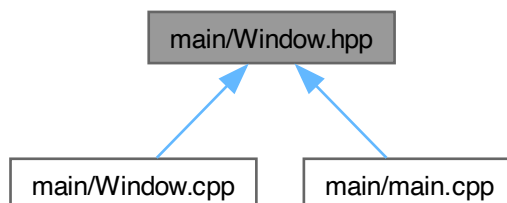
7.67 main/Window.hpp File Reference

```
#include <SDL.h>
#include <string>
#include <utility>
#include <iostream>
```

Include dependency graph for Window.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Ragot::Window](#)
Class for managing an SDL window with OpenGL context.
- struct [Ragot::Window::OpenGL_Context_Settings](#)
Struct for OpenGL context settings.

Namespaces

- namespace [Ragot](#)

7.68 Window.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of OpenGL-FinalProject
00003  *
00004  * Developed by Andrés Ragot - github.com/andresragot
00005  *
00006  * MIT License
00007  *
00008  * Copyright (c) 2025 Andrés Ragot
00009  *
00010  * Permission is hereby granted, free of charge, to any person obtaining a copy
00011  * of this software and associated documentation files (the "Software"), to deal
00012  * in the Software without restriction, including without limitation the rights
00013  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00014  * copies of the Software, and to permit persons to whom the Software is
00015  * furnished to do so, subject to the following conditions:
00016  *
00017  * The above copyright notice and this permission notice shall be included in all
00018  * copies or substantial portions of the Software.
00019  *
00020  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00021  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00022  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00023  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00024  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00025  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00026  * SOFTWARE.
00027  */
00028
00029 #pragma once
00030
00031 #include <SDL.h>
00032 #include <string>
00033 #include <utility>
00034
00035 #include <iostream>
00036
00037 namespace Ragot
00038 {
00039     class Window
00040     {
00041     public:
00042         enum Position
00043         {
00044             UNDEFINED = SDL_WINDOWPOS_UNDEFINED,
00045             CENTERED = SDL_WINDOWPOS_CENTERED,
00046         };
00047
00048         struct OpenGL_Context_Settings
00049         {
00050             unsigned version_major = 3;
00051             unsigned version_minor = 3;
00052             bool core_profile = true;
00053             unsigned depth_buffer_size = 24;
00054             unsigned stencil_buffer_size = 0;
00055             bool enable_vsync = true;
00056         };
00057
00058     private:
00059         SDL_Window* window_handle;
00060         SDL_GLContext opengl_context;
00061
00062         unsigned width;
00063         unsigned height;
00064
00065     public:
00066         Window(const std::string& title, int left_x, int top_y, unsigned width, unsigned height, const
OpenGL_Context_Settings& context_details)
00067             : Window(title.c_str(), left_x, top_y, width, height, context_details)
00068         {
00069         }
00070
00071         Window(const char* title, int left_x, int top_y, unsigned width, unsigned height, const
OpenGL_Context_Settings& context_details);
00072
00073         ~Window();
00074
00075     public:
00076         Window(const Window&) = delete;
00077         Window& operator=(const Window&) = delete;
00078
00079         Window(Window&& other) noexcept
00080         {

```

```
00115         this->window_handle = std::exchange(other.window_handle, nullptr);
00116         this->opengl_context = std::exchange(other.opengl_context, nullptr);
00117     }
00118
00124     Window& operator=(Window&& other) noexcept
00125     {
00126         this->window_handle = std::exchange(other.window_handle, nullptr);
00127         this->opengl_context = std::exchange(other.opengl_context, nullptr);
00128
00129         return *this;
00130     }
00131
00135     void swap_buffers()
00136     {
00137         SDL_GL_SwapWindow(window_handle);
00138     }
00139
00144     unsigned get_width() { return width; }
00145
00150     unsigned get_height() { return height; }
00151 };
00152 }
```

