

GraphicsEngine

Generated by Doxygen 1.9.4

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Data Structure Index	5
3.1 Data Structures	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	11
5.1 std Namespace Reference	11
6 Data Structure Documentation	13
6.1 AABB Class Reference	13
6.1.1 Detailed Description	13
6.1.2 Constructor & Destructor Documentation	14
6.1.2.1 AABB()	14
6.1.2.2 ~AABB()	14
6.1.3 Member Function Documentation	14
6.1.3.1 get_corners()	14
6.1.3.2 get_radius()	14
6.1.3.3 init()	15
6.1.3.4 render()	16
6.1.4 Field Documentation	16
6.1.4.1 corners	16
6.1.4.2 indices	17
6.1.4.3 maxX	17
6.1.4.4 maxY	17
6.1.4.5 maxZ	17
6.1.4.6 mesh	17
6.1.4.7 minX	18
6.1.4.8 minY	18
6.1.4.9 minZ	18
6.1.4.10 vertices	18
6.2 Camera Class Reference	18
6.2.1 Detailed Description	19
6.2.2 Constructor & Destructor Documentation	19
6.2.2.1 Camera() [1/2]	20
6.2.2.2 Camera() [2/2]	20
6.2.2.3 ~Camera()	20
6.2.3 Member Function Documentation	20

6.2.3.1 get_camera_direction()	21
6.2.3.2 get_camera_position()	21
6.2.3.3 get_far_plane()	21
6.2.3.4 get_fov()	21
6.2.3.5 get_near_plane()	21
6.2.3.6 get_right_axis()	22
6.2.3.7 get_up_axis()	22
6.2.3.8 get_viewmatrix()	22
6.2.3.9 get_yaw()	22
6.2.3.10 key_control()	23
6.2.3.11 mouse_control()	23
6.2.3.12 set_camera_position()	24
6.2.3.13 set_far_plane()	24
6.2.3.14 set_fov()	24
6.2.3.15 set_near_plane()	24
6.2.3.16 update()	25
6.2.4 Field Documentation	25
6.2.4.1 far_plane	25
6.2.4.2 fov	25
6.2.4.3 front	26
6.2.4.4 movement_speed	26
6.2.4.5 near_plane	26
6.2.4.6 pitch	26
6.2.4.7 position	26
6.2.4.8 right	27
6.2.4.9 turn_speed	27
6.2.4.10 up	27
6.2.4.11 world_up	27
6.2.4.12 yaw	27
6.3 CascadedShadowMap Class Reference	28
6.3.1 Detailed Description	28
6.3.2 Constructor & Destructor Documentation	28
6.3.2.1 CascadedShadowMap()	28
6.3.2.2 ~CascadedShadowMap()	29
6.3.3 Member Function Documentation	29
6.3.3.1 get_id()	29
6.3.3.2 get_intensity()	29
6.3.3.3 get_num_active_cascades()	29
6.3.3.4 get_pcf_radius()	30
6.3.3.5 get_shadow_height()	30
6.3.3.6 get_shadow_width()	30
6.3.3.7 init()	30

6.3.3.8 <code>read()</code>	31
6.3.3.9 <code>set_intensity()</code>	31
6.3.3.10 <code>set_pcf_radius()</code>	31
6.3.3.11 <code>write()</code>	32
6.3.3.12 <code>write_light_matrices()</code>	32
6.3.4 Field Documentation	32
6.3.4.1 <code>FBO</code>	32
6.3.4.2 <code>intensity</code>	32
6.3.4.3 <code>matrices_UBO</code>	33
6.3.4.4 <code>num_active_cascades</code>	33
6.3.4.5 <code>pcf_radius</code>	33
6.3.4.6 <code>shadow_height</code>	33
6.3.4.7 <code>shadow_maps</code>	33
6.3.4.8 <code>shadow_width</code>	34
6.4 ClampToEdgeMode Class Reference	34
6.4.1 Detailed Description	34
6.4.2 Constructor & Destructor Documentation	34
6.4.2.1 <code>ClampToEdgeMode()</code>	34
6.4.2.2 <code>~ClampToEdgeMode()</code>	35
6.4.3 Member Function Documentation	35
6.4.3.1 <code>activate()</code>	35
6.5 Clouds Class Reference	35
6.5.1 Detailed Description	36
6.5.2 Constructor & Destructor Documentation	36
6.5.2.1 <code>Clouds()</code>	37
6.5.2.2 <code>~Clouds()</code>	37
6.5.3 Member Function Documentation	37
6.5.3.1 <code>create_noise_textures()</code>	37
6.5.3.2 <code>get_cirrus_effect()</code>	38
6.5.3.3 <code>get_density()</code>	38
6.5.3.4 <code>get_mesh_scale()</code>	38
6.5.3.5 <code>get_model()</code>	38
6.5.3.6 <code>get_movement_direction()</code>	39
6.5.3.7 <code>get_movement_speed()</code>	39
6.5.3.8 <code>get_normal_model()</code>	39
6.5.3.9 <code>get_num_march_steps()</code>	39
6.5.3.10 <code>get_num_march_steps_to_light()</code>	39
6.5.3.11 <code>get_pillowness()</code>	40
6.5.3.12 <code>get_powder_effect()</code>	40
6.5.3.13 <code>get_radius()</code>	40
6.5.3.14 <code>get_scale()</code>	40
6.5.3.15 <code>get_shader_program()</code>	40

6.5.3.16 <code>read()</code>	41
6.5.3.17 <code>render()</code>	41
6.5.3.18 <code>set_cirrus_effect()</code>	41
6.5.3.19 <code>set_density()</code>	42
6.5.3.20 <code>set_movement_direction()</code>	42
6.5.3.21 <code>set_movement_speed()</code>	42
6.5.3.22 <code>set_num_march_steps()</code>	42
6.5.3.23 <code>set_num_march_steps_to_light()</code>	43
6.5.3.24 <code>set_pillowness()</code>	43
6.5.3.25 <code>set_powder_effect()</code>	43
6.5.3.26 <code>set_scale() [1/2]</code>	43
6.5.3.27 <code>set_scale() [2/2]</code>	44
6.5.3.28 <code>set_translation()</code>	44
6.5.4 Field Documentation	44
6.5.4.1 <code>aabb</code>	44
6.5.4.2 <code>cirrus_effect</code>	44
6.5.4.3 <code>density</code>	45
6.5.4.4 <code>maxX</code>	45
6.5.4.5 <code>maxY</code>	45
6.5.4.6 <code>maxZ</code>	45
6.5.4.7 <code>minX</code>	45
6.5.4.8 <code>minY</code>	46
6.5.4.9 <code>minZ</code>	46
6.5.4.10 <code>model</code>	46
6.5.4.11 <code>movement_direction</code>	46
6.5.4.12 <code>movement_speed</code>	46
6.5.4.13 <code>noise</code>	47
6.5.4.14 <code>num_march_steps</code>	47
6.5.4.15 <code>num_march_steps_to_light</code>	47
6.5.4.16 <code>pillowness</code>	47
6.5.4.17 <code>powder_effect</code>	47
6.5.4.18 <code>random_numbers</code>	48
6.5.4.19 <code>scale</code>	48
6.5.4.20 <code>scale_factor</code>	48
6.5.4.21 <code>shader_program</code>	48
6.5.4.22 <code>translation</code>	48
6.6 ComputeShaderProgram Class Reference	49
6.6.1 Detailed Description	49
6.6.2 Constructor & Destructor Documentation	49
6.6.2.1 <code>ComputeShaderProgram()</code>	49
6.6.2.2 <code>~ComputeShaderProgram()</code>	49
6.6.3 Member Function Documentation	49

6.6.3.1 <code>reload()</code>	50
6.7 <code>DebugApp</code> Class Reference	50
6.7.1 Detailed Description	50
6.7.2 Constructor & Destructor Documentation	50
6.7.2.1 <code>DebugApp()</code>	51
6.7.2.2 <code>~DebugApp()</code>	51
6.7.3 Member Function Documentation	51
6.7.3.1 <code>areErrorPrintAll()</code>	51
6.7.3.2 <code>arePreError()</code>	52
6.8 <code>DirectionalLight</code> Class Reference	53
6.8.1 Detailed Description	53
6.8.2 Constructor & Destructor Documentation	54
6.8.2.1 <code>DirectionalLight()</code> [1/2]	54
6.8.2.2 <code>DirectionalLight()</code> [2/2]	54
6.8.2.3 <code>~DirectionalLight()</code>	55
6.8.3 Member Function Documentation	55
6.8.3.1 <code>calc_cascaded_slots()</code>	55
6.8.3.2 <code>calc_orthogonal_projections()</code>	55
6.8.3.3 <code>calculate_light_transform()</code>	56
6.8.3.4 <code>get_cascaded_light_matrices()</code>	57
6.8.3.5 <code>get_cascaded_slots()</code>	57
6.8.3.6 <code>get_color()</code>	57
6.8.3.7 <code>get_direction()</code>	57
6.8.3.8 <code>get_frustum_corners_world_space()</code>	58
6.8.3.9 <code>get_light_view_matrix()</code>	58
6.8.3.10 <code>get_radiance()</code>	58
6.8.3.11 <code>get_shadow_map()</code>	58
6.8.3.12 <code>set_color()</code>	59
6.8.3.13 <code>set_direction()</code>	59
6.8.3.14 <code>set_radiance()</code>	59
6.8.3.15 <code>update_shadow_map()</code>	59
6.8.4 Field Documentation	59
6.8.4.1 <code>cascade_light_matrices</code>	60
6.8.4.2 <code>cascade_slots</code>	60
6.8.4.3 <code>direction</code>	60
6.8.4.4 <code>shadow_far_plane</code>	60
6.8.4.5 <code>shadow_map</code>	60
6.8.4.6 <code>shadow_near_plane</code>	61
6.9 <code>DirectionalShadowMapPass</code> Class Reference	61
6.9.1 Detailed Description	61
6.9.2 Constructor & Destructor Documentation	61
6.9.2.1 <code>DirectionalShadowMapPass()</code>	62

6.9.2.2 ~DirectionalShadowMapPass()	62
6.9.3 Member Function Documentation	62
6.9.3.1 create_shader_program()	62
6.9.3.2 execute()	63
6.9.3.3 set_game_object_uniforms()	63
6.9.4 Field Documentation	64
6.9.4.1 shader_program	64
6.10 File Class Reference	64
6.10.1 Detailed Description	64
6.10.2 Constructor & Destructor Documentation	64
6.10.2.1 File()	65
6.10.2.2 ~File()	65
6.10.3 Member Function Documentation	65
6.10.3.1 read()	65
6.10.4 Field Documentation	65
6.10.4.1 file_location	66
6.11 ViewFrustumCulling::frustum_plane Struct Reference	66
6.11.1 Detailed Description	66
6.11.2 Field Documentation	66
6.11.2.1 normal	66
6.11.2.2 position	67
6.12 GameObject Class Reference	67
6.12.1 Detailed Description	67
6.12.2 Constructor & Destructor Documentation	67
6.12.2.1 GameObject() [1/2]	68
6.12.2.2 GameObject() [2/2]	68
6.12.2.3 ~GameObject()	68
6.12.3 Member Function Documentation	68
6.12.3.1 get_aabb()	68
6.12.3.2 get_model()	69
6.12.3.3 get_normal_world_trafo()	69
6.12.3.4 get_world_trafo()	69
6.12.3.5 init()	69
6.12.3.6 render()	70
6.12.3.7 rotate()	70
6.12.3.8 scale()	70
6.12.3.9 translate()	70
6.12.4 Field Documentation	71
6.12.4.1 model	71
6.12.4.2 rot	71
6.12.4.3 scale_factor	71
6.12.4.4 translation	71

6.13 GBuffer Class Reference	72
6.13.1 Detailed Description	72
6.13.2 Constructor & Destructor Documentation	72
6.13.2.1 GBuffer() [1/2]	72
6.13.2.2 GBuffer() [2/2]	73
6.13.2.3 ~GBuffer()	73
6.13.3 Member Function Documentation	73
6.13.3.1 create()	73
6.13.3.2 get_id()	74
6.13.3.3 read()	74
6.13.3.4 update_window_params()	75
6.13.4 Field Documentation	75
6.13.4.1 g_albedo	75
6.13.4.2 g_buffer	75
6.13.4.3 g_buffer_attachment	75
6.13.4.4 g_depth	76
6.13.4.5 g_material_id	76
6.13.4.6 g_normal	76
6.13.4.7 g_position	76
6.13.4.8 window_height	76
6.13.4.9 window_width	77
6.14 GeometryPass Class Reference	77
6.14.1 Detailed Description	77
6.14.2 Constructor & Destructor Documentation	77
6.14.2.1 GeometryPass()	78
6.14.2.2 ~GeometryPass()	78
6.14.3 Member Function Documentation	78
6.14.3.1 create_shader_program()	78
6.14.3.2 execute()	79
6.14.3.3 set_game_object_uniforms()	80
6.14.4 Field Documentation	81
6.14.4.1 shader_program	81
6.14.4.2 skybox	81
6.15 GeometryPassShaderProgram Class Reference	81
6.15.1 Detailed Description	81
6.15.2 Constructor & Destructor Documentation	82
6.15.2.1 GeometryPassShaderProgram()	82
6.15.2.2 ~GeometryPassShaderProgram()	82
6.15.3 Member Function Documentation	82
6.15.3.1 get_program_id()	82
6.16 GUI Class Reference	82
6.16.1 Detailed Description	83

6.16.2 Constructor & Destructor Documentation	83
6.16.2.1 GUI()	84
6.16.2.2 ~GUI()	84
6.16.3 Member Function Documentation	85
6.16.3.1 init()	85
6.16.3.2 render()	85
6.16.3.3 update_user_input()	87
6.16.4 Field Documentation	88
6.16.4.1 available_shadow_map_resolutions	88
6.16.4.2 cascaded_shadow_intensity	88
6.16.4.3 cloud_cirrus_effect	88
6.16.4.4 cloud_density	89
6.16.4.5 cloud_mesh_offset	89
6.16.4.6 cloud_mesh_scale	89
6.16.4.7 cloud_movement_direction	89
6.16.4.8 cloud_num_march_steps	89
6.16.4.9 cloud_num_march_steps_to_light	90
6.16.4.10 cloud_pillowness	90
6.16.4.11 cloud_powder_effect	90
6.16.4.12 cloud_scale	90
6.16.4.13 cloud_speed	90
6.16.4.14 direccional_light_radiance	91
6.16.4.15 directional_light_color	91
6.16.4.16 directional_light_direction	91
6.16.4.17 logo_tex	91
6.16.4.18 num_shadow_cascades	91
6.16.4.19 pcf_radius	92
6.16.4.20 shadow_map_res_index	92
6.16.4.21 shadow_resolution_changed	92
6.17 std::hash< Vertex > Struct Reference	92
6.17.1 Detailed Description	92
6.17.2 Member Function Documentation	93
6.17.2.1 operator()()	93
6.18 Light Class Reference	93
6.18.1 Detailed Description	93
6.18.2 Constructor & Destructor Documentation	94
6.18.2.1 Light() [1/2]	94
6.18.2.2 Light() [2/2]	94
6.18.2.3 ~Light()	94
6.18.3 Member Function Documentation	94
6.18.3.1 get_color()	94
6.18.3.2 get_radiance()	95

6.18.4 Field Documentation	95
6.18.4.1 color	95
6.18.4.2 light_proj	95
6.18.4.3 radiance	95
6.19 LightingPass Class Reference	96
6.19.1 Detailed Description	96
6.19.2 Constructor & Destructor Documentation	96
6.19.2.1 LightingPass()	96
6.19.2.2 ~LightingPass()	97
6.19.3 Member Function Documentation	97
6.19.3.1 create_shader_program()	97
6.19.3.2 execute()	97
6.19.3.3 set_uniforms()	98
6.19.4 Field Documentation	100
6.19.4.1 current_offset	100
6.19.4.2 quad	100
6.19.4.3 shader_program	100
6.20 LightingPassShaderProgram Class Reference	101
6.20.1 Detailed Description	101
6.20.2 Constructor & Destructor Documentation	101
6.20.2.1 LightingPassShaderProgram()	101
6.20.2.2 ~LightingPassShaderProgram()	101
6.21 LoadingScreen Class Reference	102
6.21.1 Detailed Description	102
6.21.2 Constructor & Destructor Documentation	102
6.21.2.1 LoadingScreen()	102
6.21.2.2 ~LoadingScreen()	103
6.21.3 Member Function Documentation	103
6.21.3.1 create_shader_program()	103
6.21.3.2 init()	103
6.21.3.3 render()	104
6.21.4 Field Documentation	104
6.21.4.1 loading_screen_quad	104
6.21.4.2 loading_screen_shader_program	104
6.21.4.3 loading_screen_tex	104
6.21.4.4 logo_tex	105
6.22 Mesh Class Reference	105
6.22.1 Detailed Description	105
6.22.2 Member Enumeration Documentation	105
6.22.2.1 anonymous enum	105
6.22.2.2 anonymous enum	106
6.22.3 Constructor & Destructor Documentation	106

6.22.3.1 Mesh() [1/2]	106
6.22.3.2 Mesh() [2/2]	107
6.22.3.3 ~Mesh()	107
6.22.4 Member Function Documentation	107
6.22.4.1 getIndices()	108
6.22.4.2 getVertices()	108
6.22.4.3 render()	108
6.22.5 Field Documentation	108
6.22.5.1 indices	108
6.22.5.2 m_drawCount	109
6.22.5.3 m_ibl	109
6.22.5.4 m_vab	109
6.22.5.5 m_vao	109
6.22.5.6 vertices	109
6.23 MirroredRepeatMode Class Reference	110
6.23.1 Detailed Description	110
6.23.2 Constructor & Destructor Documentation	110
6.23.2.1 MirroredRepeatMode()	110
6.23.2.2 ~MirroredRepeatMode()	110
6.23.3 Member Function Documentation	110
6.23.3.1 activate()	111
6.24 Model Class Reference	111
6.24.1 Detailed Description	112
6.24.2 Constructor & Destructor Documentation	112
6.24.2.1 Model()	112
6.24.2.2 ~Model()	112
6.24.3 Member Function Documentation	112
6.24.3.1 bind_ressources()	112
6.24.3.2 create_render_context()	113
6.24.3.3 get_aabb()	113
6.24.3.4 get_materials()	113
6.24.3.5 get_texture_count()	113
6.24.3.6 load_model_in_ram()	114
6.24.3.7 render()	114
6.24.3.8 unbind_resources()	114
6.24.4 Field Documentation	114
6.24.4.1 aabb	114
6.24.4.2 indices	115
6.24.4.3 loader	115
6.24.4.4 materialIndex	115
6.24.4.5 materials	115
6.24.4.6 mesh	115

6.24.4.7 ssbo	116
6.24.4.8 texture_list	116
6.24.4.9 textures	116
6.24.4.10 vertices	116
6.25 Noise Class Reference	116
6.25.1 Detailed Description	117
6.25.2 Constructor & Destructor Documentation	117
6.25.2.1 Noise()	118
6.25.2.2 ~Noise()	118
6.25.3 Member Function Documentation	118
6.25.3.1 create_res128_noise()	118
6.25.3.2 create_res32_noise()	119
6.25.3.3 create_shader_programs()	120
6.25.3.4 delete_textures()	120
6.25.3.5 generate_cells()	120
6.25.3.6 generate_num_cells_textures()	121
6.25.3.7 generate_res128_noise_texture()	122
6.25.3.8 generate_res32_noise_texture()	122
6.25.3.9 generate_textures()	123
6.25.3.10 print_comp_shader_capabilities()	123
6.25.3.11 read_res128_noise()	123
6.25.3.12 read_res32_noise()	124
6.25.3.13 set_num_cells()	124
6.25.3.14 update()	124
6.25.4 Field Documentation	124
6.25.4.1 cell_data	125
6.25.4.2 cell_ids	125
6.25.4.3 num_cells_per_axis	125
6.25.4.4 texture_1_id	125
6.25.4.5 texture_1_shader_program	125
6.25.4.6 texture_2_id	126
6.25.4.7 texture_2_shader_program	126
6.25.4.8 texture_dim_1	126
6.25.4.9 texture_dim_2	126
6.26 ObjLoader Class Reference	126
6.26.1 Detailed Description	127
6.26.2 Constructor & Destructor Documentation	127
6.26.2.1 ObjLoader()	127
6.26.2.2 ~ObjLoader()	127
6.26.3 Member Function Documentation	127
6.26.3.1 get_base_dir()	128
6.26.3.2 load()	128

6.26.4 Field Documentation	130
6.26.4.1 maxX	130
6.26.4.2 maxY	130
6.26.4.3 maxZ	131
6.26.4.4 minX	131
6.26.4.5 minY	131
6.26.4.6 minZ	131
6.27 ObjMaterial Class Reference	131
6.27.1 Detailed Description	132
6.27.2 Constructor & Destructor Documentation	132
6.27.2.1 ObjMaterial() [1/2]	132
6.27.2.2 ObjMaterial() [2/2]	133
6.27.2.3 ~ObjMaterial()	133
6.27.3 Member Function Documentation	133
6.27.3.1 get_ambient()	133
6.27.3.2 get_diffuse()	134
6.27.3.3 get_dissolve()	134
6.27.3.4 get_emission()	134
6.27.3.5 get_illum()	134
6.27.3.6 get_ior()	134
6.27.3.7 get_shininess()	135
6.27.3.8 get_specular()	135
6.27.3.9 get_textureID()	135
6.27.3.10 get_transmittance()	135
6.27.4 Field Documentation	135
6.27.4.1 ambient	136
6.27.4.2 diffuse	136
6.27.4.3 dissolve	136
6.27.4.4 emission	136
6.27.4.5 illum	136
6.27.4.6 ior	137
6.27.4.7 shininess	137
6.27.4.8 specular	137
6.27.4.9 textureID	137
6.27.4.10 transmittance	137
6.28 OmniDirShadowMap Class Reference	138
6.28.1 Detailed Description	138
6.28.2 Constructor & Destructor Documentation	138
6.28.2.1 OmniDirShadowMap()	138
6.28.2.2 ~OmniDirShadowMap()	138
6.28.3 Member Function Documentation	138
6.28.3.1 init()	139

6.28.3.2 <code>read()</code>	139
6.28.3.3 <code>write()</code>	140
6.29 <code>OmniDirShadowShaderProgram</code> Class Reference	140
6.29.1 Detailed Description	140
6.29.2 Constructor & Destructor Documentation	140
6.29.2.1 <code>OmniDirShadowShaderProgram()</code>	140
6.29.2.2 <code>~OmniDirShadowShaderProgram()</code>	141
6.29.3 Member Function Documentation	141
6.29.3.1 <code>reload()</code>	141
6.30 <code>OmniShadowMapPass</code> Class Reference	141
6.30.1 Detailed Description	142
6.30.2 Constructor & Destructor Documentation	142
6.30.2.1 <code>OmniShadowMapPass()</code>	142
6.30.2.2 <code>~OmniShadowMapPass()</code>	142
6.30.3 Member Function Documentation	142
6.30.3.1 <code>create_shader_program()</code>	142
6.30.3.2 <code>execute()</code>	143
6.30.3.3 <code>set_game_object_uniforms()</code>	143
6.30.4 Field Documentation	143
6.30.4.1 <code>shader_program</code>	144
6.31 <code>PointLight</code> Class Reference	144
6.31.1 Detailed Description	144
6.31.2 Constructor & Destructor Documentation	145
6.31.2.1 <code>PointLight() [1/2]</code>	145
6.31.2.2 <code>PointLight() [2/2]</code>	145
6.31.2.3 <code>~PointLight()</code>	145
6.31.3 Member Function Documentation	146
6.31.3.1 <code>calculate_light_transform()</code>	146
6.31.3.2 <code>get_constant_factor()</code>	146
6.31.3.3 <code>get_exponent_factor()</code>	146
6.31.3.4 <code>get_far_plane()</code>	147
6.31.3.5 <code>get_linear_factor()</code>	147
6.31.3.6 <code>get_omni_shadow_map()</code>	147
6.31.3.7 <code>get_position()</code>	147
6.31.3.8 <code>set_position()</code>	147
6.31.4 Field Documentation	148
6.31.4.1 <code>constant</code>	148
6.31.4.2 <code>exponent</code>	148
6.31.4.3 <code>far_plane</code>	148
6.31.4.4 <code>linear</code>	148
6.31.4.5 <code>omni_dir_shadow_map</code>	148
6.31.4.6 <code>position</code>	149

6.32 Quad Class Reference	149
6.32.1 Detailed Description	149
6.32.2 Constructor & Destructor Documentation	149
6.32.2.1 Quad()	150
6.32.2.2 ~Quad()	150
6.32.3 Member Function Documentation	150
6.32.3.1 render()	150
6.32.4 Field Documentation	150
6.32.4.1 q_vao	151
6.32.4.2 q_vbo	151
6.32.4.3 vertices	151
6.33 RandomNumbers Class Reference	152
6.33.1 Detailed Description	152
6.33.2 Constructor & Destructor Documentation	152
6.33.2.1 RandomNumbers()	152
6.33.2.2 ~RandomNumbers()	153
6.33.3 Member Function Documentation	153
6.33.3.1 generate_random_numbers()	153
6.33.3.2 read()	153
6.33.4 Field Documentation	154
6.33.4.1 random_number_data	154
6.33.4.2 random_number_id	154
6.34 Renderer Class Reference	154
6.34.1 Detailed Description	155
6.34.2 Constructor & Destructor Documentation	155
6.34.2.1 Renderer()	155
6.34.2.2 ~Renderer()	155
6.34.3 Member Function Documentation	156
6.34.3.1 drawFrame()	156
6.34.3.2 reload_shader_programs()	156
6.34.3.3 update_window_params()	157
6.34.4 Field Documentation	157
6.34.4.1 directional_shadow_map_pass	157
6.34.4.2 gbuffer	157
6.34.4.3 geometry_pass	157
6.34.4.4 lighting_pass	158
6.34.4.5 omni_shadow_map_pass	158
6.34.4.6 render_passes	158
6.34.4.7 shader_includes	158
6.34.4.8 window_height	158
6.34.4.9 window_width	159
6.35 RenderPass Class Reference	159

6.35.1 Detailed Description	159
6.35.2 Member Function Documentation	159
6.35.2.1 <code>create_shader_program()</code>	159
6.36 RenderPassSceneDependend Class Reference	160
6.36.1 Detailed Description	160
6.36.2 Constructor & Destructor Documentation	160
6.36.2.1 <code>RenderPassSceneDependend()</code>	160
6.36.2.2 <code>~RenderPassSceneDependend()</code>	160
6.36.3 Member Function Documentation	160
6.36.3.1 <code>create_shader_program()</code>	161
6.36.3.2 <code>set_game_object_uniforms()</code>	161
6.37 RepeatMode Class Reference	161
6.37.1 Detailed Description	161
6.37.2 Constructor & Destructor Documentation	161
6.37.2.1 <code>RepeatMode()</code>	162
6.37.2.2 <code>~RepeatMode()</code>	162
6.37.3 Member Function Documentation	162
6.37.3.1 <code>activate()</code>	162
6.38 Rotation Struct Reference	162
6.38.1 Detailed Description	163
6.38.2 Field Documentation	163
6.38.2.1 <code>axis</code>	163
6.38.2.2 <code>degrees</code>	163
6.39 Scene Class Reference	163
6.39.1 Detailed Description	164
6.39.2 Constructor & Destructor Documentation	164
6.39.2.1 <code>Scene() [1/2]</code>	164
6.39.2.2 <code>Scene() [2/2]</code>	165
6.39.2.3 <code>~Scene()</code>	165
6.39.3 Member Function Documentation	165
6.39.3.1 <code>add_game_object()</code>	165
6.39.3.2 <code>bind_textures_and_buffer()</code>	166
6.39.3.3 <code>get_clouds()</code>	166
6.39.3.4 <code>get_context_setup()</code>	166
6.39.3.5 <code>get_game_objects()</code>	166
6.39.3.6 <code>get_materials()</code>	166
6.39.3.7 <code>get_point_light_count()</code>	167
6.39.3.8 <code>get_point_lights()</code>	167
6.39.3.9 <code>get_progress()</code>	167
6.39.3.10 <code>get_sun()</code>	167
6.39.3.11 <code>get_texture_count()</code>	167
6.39.3.12 <code>is_loaded()</code>	168

6.39.3.13 load_models()	168
6.39.3.14 object_is_visible()	168
6.39.3.15 set_context_setup()	169
6.39.3.16 setup_game_object_context()	169
6.39.3.17 spwan()	169
6.39.3.18 unbind_textures_and_buffer()	169
6.39.4 Field Documentation	169
6.39.4.1 clouds	170
6.39.4.2 context_setup	170
6.39.4.3 game_objects	170
6.39.4.4 loaded_scene	170
6.39.4.5 main_camera	170
6.39.4.6 main_window	171
6.39.4.7 mx_isLoaded	171
6.39.4.8 mx_progress	171
6.39.4.9 mx_space_ship	171
6.39.4.10 point_lights	171
6.39.4.11 progress	172
6.39.4.12 sun	172
6.39.4.13 view_frustum_culling	172
6.40 ShaderIncludes Class Reference	172
6.40.1 Detailed Description	173
6.40.2 Constructor & Destructor Documentation	173
6.40.2.1 ShaderIncludes()	173
6.40.2.2 ~ShaderIncludes()	173
6.40.3 Field Documentation	173
6.40.3.1 file_locations_relative	174
6.40.3.2 includeNames	174
6.41 ShaderProgram Class Reference	175
6.41.1 Detailed Description	175
6.41.2 Constructor & Destructor Documentation	176
6.41.2.1 ShaderProgram() [1/2]	176
6.41.2.2 ShaderProgram() [2/2]	176
6.41.2.3 ~ShaderProgram()	176
6.41.3 Member Function Documentation	176
6.41.3.1 add_shader()	177
6.41.3.2 clear_shader_program()	177
6.41.3.3 compile_compute_shader_program()	178
6.41.3.4 compile_program()	178
6.41.3.5 compile_shader_program() [1/2]	178
6.41.3.6 compile_shader_program() [2/2]	179
6.41.3.7 create_computer_shader_program_from_file()	179

6.41.3.8 create_from_files() [1/2]	180
6.41.3.9 create_from_files() [2/2]	180
6.41.3.10 get_id()	181
6.41.3.11 getUniformLocation()	181
6.41.3.12 setUniformBlockBinding()	181
6.41.3.13 setUniformFloat()	182
6.41.3.14 setUniformInt()	182
6.41.3.15 setUniformMatrix4fv()	182
6.41.3.16 setUniformVec3()	183
6.41.3.17 use_shader_program()	183
6.41.3.18 validate_program()	183
6.41.3.19 validateUniformLocation()	184
6.41.4 Field Documentation	184
6.41.4.1 compute_location	184
6.41.4.2 fragment_location	184
6.41.4.3 geometry_location	184
6.41.4.4 program_id	185
6.41.4.5 shader_base_dir	185
6.41.4.6 vertex_location	185
6.42 ShadowMap Class Reference	185
6.42.1 Detailed Description	186
6.42.2 Constructor & Destructor Documentation	186
6.42.2.1 ShadowMap()	186
6.42.2.2 ~ShadowMap()	186
6.42.3 Member Function Documentation	186
6.42.3.1 get_id()	187
6.42.3.2 get_shadow_height()	187
6.42.3.3 get_shadow_width()	187
6.42.3.4 init()	187
6.42.3.5 read()	188
6.42.3.6 write()	188
6.42.4 Field Documentation	188
6.42.4.1 FBO	189
6.42.4.2 shadow_height	189
6.42.4.3 shadow_map	189
6.42.4.4 shadow_width	189
6.43 SkyBox Class Reference	189
6.43.1 Detailed Description	190
6.43.2 Constructor & Destructor Documentation	190
6.43.2.1 SkyBox()	190
6.43.2.2 ~SkyBox()	192
6.43.3 Member Function Documentation	192

6.43.3.1 draw_sky_box()	192
6.43.3.2 reload()	193
6.43.4 Field Documentation	193
6.43.4.1 movement_speed	193
6.43.4.2 shader_playback_time	193
6.43.4.3 shader_program	193
6.43.4.4 sky_mesh	194
6.43.4.5 texture_id	194
6.43.4.6 uniform_projection	194
6.43.4.7 uniform_view	194
6.44 Texture Class Reference	194
6.44.1 Detailed Description	195
6.44.2 Constructor & Destructor Documentation	195
6.44.2.1 Texture() [1/2]	195
6.44.2.2 Texture() [2/2]	196
6.44.2.3 ~Texture()	196
6.44.3 Member Function Documentation	196
6.44.3.1 clear_texture_context()	196
6.44.3.2 get_filename()	197
6.44.3.3 get_id()	197
6.44.3.4 load_SRGB_texture_with_alpha_channel()	197
6.44.3.5 load_SRGB_texture_without_alpha_channel()	198
6.44.3.6 load_texture_with_alpha_channel()	198
6.44.3.7 load_texture_without_alpha_channel()	199
6.44.3.8 unbind_texture()	199
6.44.3.9 use_texture()	199
6.44.4 Field Documentation	200
6.44.4.1 bit_depth	200
6.44.4.2 file_location	200
6.44.4.3 height	200
6.44.4.4 textureID	200
6.44.4.5 width	201
6.44.4.6 wrapping_mode	201
6.45 TextureWrappingMode Class Reference	201
6.45.1 Detailed Description	201
6.45.2 Member Function Documentation	201
6.45.2.1 activate()	202
6.46 Vertex Struct Reference	202
6.46.1 Detailed Description	202
6.46.2 Constructor & Destructor Documentation	202
6.46.2.1 Vertex() [1/2]	203
6.46.2.2 Vertex() [2/2]	203

6.46.3 Member Function Documentation	203
6.46.3.1 <code>get_color()</code>	203
6.46.3.2 <code>get_normal()</code>	203
6.46.3.3 <code>get_position()</code>	204
6.46.3.4 <code>get_tex_coors()</code>	204
6.46.3.5 <code>operator==()</code>	204
6.46.4 Field Documentation	204
6.46.4.1 <code>color</code>	204
6.46.4.2 <code>normal</code>	205
6.46.4.3 <code>position</code>	205
6.46.4.4 <code>texture_coords</code>	205
6.47 ViewFrustumCulling Class Reference	205
6.47.1 Detailed Description	206
6.47.2 Constructor & Destructor Documentation	206
6.47.2.1 <code>ViewFrustumCulling()</code>	207
6.47.2.2 <code>~ViewFrustumCulling()</code>	207
6.47.3 Member Function Documentation	207
6.47.3.1 <code>corners_outside_plane()</code>	207
6.47.3.2 <code>init()</code>	208
6.47.3.3 <code>is_inside()</code>	209
6.47.3.4 <code>plane_point_distance()</code>	210
6.47.3.5 <code>render_view_frustum()</code>	211
6.47.3.6 <code>update_frustum_param()</code>	211
6.47.4 Field Documentation	212
6.47.4.1 <code>dir</code>	212
6.47.4.2 <code>EBO</code>	213
6.47.4.3 <code>far_bottom_left</code>	213
6.47.4.4 <code>far_bottom_right</code>	213
6.47.4.5 <code>far_center</code>	213
6.47.4.6 <code>far_height</code>	213
6.47.4.7 <code>far_plane</code>	214
6.47.4.8 <code>far_top_left</code>	214
6.47.4.9 <code>far_top_right</code>	214
6.47.4.10 <code>far_width</code>	214
6.47.4.11 <code>fov</code>	214
6.47.4.12 <code>frustum_planes</code>	215
6.47.4.13 <code>m_drawCount</code>	215
6.47.4.14 <code>main_camera</code>	215
6.47.4.15 <code>near_bottom_left</code>	215
6.47.4.16 <code>near_bottom_right</code>	215
6.47.4.17 <code>near_center</code>	216
6.47.4.18 <code>near_height</code>	216

6.47.4.19 near_plane	216
6.47.4.20 near_top_left	216
6.47.4.21 near_top_right	216
6.47.4.22 near_width	217
6.47.4.23 ratio	217
6.47.4.24 tan	217
6.47.4.25 VAO	217
6.47.4.26 VBO	217
6.48 Window Class Reference	218
6.48.1 Detailed Description	219
6.48.2 Constructor & Destructor Documentation	219
6.48.2.1 Window() [1/2]	219
6.48.2.2 Window() [2/2]	219
6.48.2.3 ~Window()	219
6.48.3 Member Function Documentation	220
6.48.3.1 framebuffer_size_callback()	220
6.48.3.2 get_buffer_height()	220
6.48.3.3 get_buffer_width()	220
6.48.3.4 get_keys()	220
6.48.3.5 get_should_close()	221
6.48.3.6 get_window()	221
6.48.3.7 get_x_change()	221
6.48.3.8 get_y_change()	221
6.48.3.9 init_callbacks()	222
6.48.3.10 initialize()	222
6.48.3.11 key_callback()	223
6.48.3.12 mouse_button_callback()	224
6.48.3.13 mouse_callback()	224
6.48.3.14 swap_buffers()	224
6.48.3.15 update_viewport()	225
6.48.4 Field Documentation	225
6.48.4.1 keys	225
6.48.4.2 last_x	225
6.48.4.3 last_y	225
6.48.4.4 main_window	226
6.48.4.5 mouse_first_moved	226
6.48.4.6 window_buffer_height	226
6.48.4.7 window_buffer_width	226
6.48.4.8 window_height	226
6.48.4.9 window_width	227
6.48.4.10 x_change	227
6.48.4.11 y_change	227

7 File Documentation	229
7.1 C:/Users/jonas/Desktop/GraphicEngine/Src/app/App.cpp File Reference	229
7.1.1 Function Documentation	229
7.1.1.1 main()	230
7.2 App.cpp	231
7.3 C:/Users/jonas/Desktop/GraphicEngine/Src/bindings.h File Reference	232
7.3.1 Macro Definition Documentation	233
7.3.1.1 D_LIGHT_SHADOW_TEXTURES_SLOT	233
7.3.1.2 GBUFFER_TEXTURES_SLOT	233
7.3.1.3 MODEL_TEXTURES_SLOT	233
7.3.1.4 NOISE_128D_IMAGE_SLOT	233
7.3.1.5 NOISE_128D_TEXTURES_SLOT	234
7.3.1.6 NOISE_32D_IMAGE_SLOT	234
7.3.1.7 NOISE_32D_TEXTURES_SLOT	234
7.3.1.8 NOISE_CELL_POSITIONS_SLOT	234
7.3.1.9 P_LIGHT_SHADOW_TEXTURES_SLOT	234
7.3.1.10 RANDOM_NUMBERS_SLOT	234
7.3.1.11 SKYBOX_TEXTURES_SLOT	235
7.3.1.12 STORAGE_BUFFER_MATERIAL_ID_BINDING	235
7.3.1.13 UNIFORM_LIGHT_MATRICES_BINDING	235
7.4 bindings.h	235
7.5 C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.cpp File Reference	235
7.6 Camera.cpp	236
7.7 C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.h File Reference	237
7.8 Camera.h	237
7.9 C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProgram.cpp File Reference	238
7.10 ComputeShaderProgram.cpp	238
7.11 C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProgram.h File Reference	238
7.12 ComputeShaderProgram.h	239
7.13 C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.cpp File Reference	239
7.13.1 Function Documentation	239
7.13.1.1 gIDebugOutput()	240
7.14 DebugApp.cpp	241
7.15 C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.h File Reference	243
7.16 DebugApp.h	243
7.17 C:/Users/jonas/Desktop/GraphicEngine/Src/GlobalValues.h File Reference	244
7.17.1 Macro Definition Documentation	244
7.17.1.1 MAX_RESOLUTION_X	244
7.17.1.2 MAX_RESOLUTION_Y	244
7.17.1.3 NUM_MIN_CASCADES	245
7.17.2 Variable Documentation	245
7.17.2.1 G_BUFFER_SIZE	245

7.17.2.2 NUM_CLOUDS	245
7.17.2.3 NUM_FRUSTUM_PLANES	245
7.18 GlobalValues.h	245
7.19 C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.cpp File Reference	246
7.20 GUI.cpp	246
7.21 C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.h File Reference	249
7.22 GUI.h	249
7.23 C:/Users/jonas/Desktop/GraphicEngine/Src/host_device_shared.h File Reference	250
7.23.1 Variable Documentation	250
7.23.1.1 CLOUDS_MATERIAL_ID	251
7.23.1.2 MAX_MATERIALS	251
7.23.1.3 MAX_POINT_LIGHTS	251
7.23.1.4 MAX_TEXTURE_COUNT	251
7.23.1.5 NUM_CASCADES	251
7.23.1.6 NUM_CELL_POSITIONS	252
7.23.1.7 PI	252
7.23.1.8 SKYBOX_MATERIAL_ID	252
7.24 host_device_shared.h	252
7.25 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GBuffer.cpp File Reference	252
7.26 GBuffer.cpp	253
7.27 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GBuffer.h File Reference	254
7.28 GBuffer.h	254
7.29 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryPass.cpp File Reference	255
7.30 GeometryPass.cpp	255
7.31 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryPass.h File Reference	257
7.32 GeometryPass.h	257
7.33 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryPassShaderProgram.cpp File Reference	257
7.34 GeometryPassShaderProgram.cpp	258
7.35 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryPassShaderProgram.h File Reference	258
7.36 GeometryPassShaderProgram.h	258
7.37 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPass.cpp File Reference	258
7.38 LightingPass.cpp	259
7.39 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPass.h File Reference	261
7.40 LightingPass.h	262
7.41 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPassShaderProgram.cpp File Reference	262
7.42 LightingPassShaderProgram.cpp	263
7.43 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPassShaderProgram.h File Reference	263
7.44 LightingPassShaderProgram.h	263
7.45 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/RenderPass.h File Reference	263

7.46 RenderPass.h	264
7.47 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen>LoadingScreen.cpp File Reference	264
7.48 LoadingScreen.cpp	264
7.49 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen>LoadingScreen.h File Reference	265
7.50 LoadingScreen.h	265
7.51 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.cpp File Reference	265
7.52 Renderer.cpp	265
7.53 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.h File Reference	266
7.54 Renderer.h	267
7.55 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneDependend.cpp File Reference	267
7.56 RenderPassSceneDependend.cpp	267
7.57 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneDependend.h File Reference	267
7.58 RenderPassSceneDependend.h	268
7.59 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.cpp File Reference	268
7.60 ShaderIncludes.cpp	268
7.61 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.h File Reference	269
7.62 ShaderIncludes.h	269
7.63 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderProgram.cpp File Reference	270
7.64 ShaderProgram.cpp	270
7.65 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderProgram.h File Reference	274
7.66 ShaderProgram.h	274
7.67 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.cpp File Reference	275
7.68 AABB.cpp	275
7.69 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.h File Reference	277
7.70 AABB.h	277
7.71 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Clouds.cpp File Reference	278
7.72 Clouds.cpp	278
7.73 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Clouds.h File Reference	279
7.74 Clouds.h	279
7.75 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Noise.cpp File Reference	280
7.76 Noise.cpp	280
7.77 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Noise.h File Reference	284
7.78 Noise.h	284
7.79 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Perlin.cpp File Reference	285
7.80 Perlin.cpp	285
7.81 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Perlin.h File Reference	287
7.81.1 Function Documentation	287
7.81.1.1 fade()	287

7.81.1.2 getPermutationVector()	287
7.81.1.3 grad()	290
7.81.1.4 lerp()	291
7.81.1.5 perlin_noise()	291
7.82 Perlin.h	292
7.83 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.cpp File Reference	296
7.84 GameObject.cpp	296
7.85 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.h File Reference	297
7.86 GameObject.h	297
7.87 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/CascadedShadowMap.cpp File Reference	298
7.88 CascadedShadowMap.cpp	298
7.89 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/CascadedShadowMap.h File Reference	299
7.90 CascadedShadowMap.h	299
7.91 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalLight.cpp File Reference	300
7.92 DirectionalLight.cpp	300
7.93 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalLight.h File Reference	303
7.94 DirectionalLight.h	303
7.95 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalShadowMapPass.cpp File Reference	304
7.96 DirectionalShadowMapPass.cpp	304
7.97 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalShadowMapPass.h File Reference	305
7.98 DirectionalShadowMapPass.h	305
7.99 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.cpp File Reference	305
7.100 Light.cpp	305
7.101 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.h File Reference	306
7.102 Light.h	306
7.103 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniDirShadowMap.cpp File Reference	306
7.104 OmniDirShadowMap.cpp	307
7.105 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniDirShadowMap.h File Reference	307
7.106 OmniDirShadowMap.h	308
7.107 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniDirShadowShaderProgram.cpp File Reference	308
7.108 OmniDirShadowShaderProgram.cpp	308
7.109 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniDirShadowShaderProgram.h File Reference	308
7.110 OmniDirShadowShaderProgram.h	309
7.111 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniShadowMapPass.cpp File Reference	309

7.112 OmniShadowMapPass.cpp	309
7.113 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniShadowMapPass.h File Reference	310
7.114 OmniShadowMapPass.h	310
7.115 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/PointLight.cpp File Reference	310
7.116 PointLight.cpp	311
7.117 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/PointLight.h File Reference	311
7.118 PointLight.h	312
7.119 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.cpp File Reference	312
7.120 Mesh.cpp	312
7.121 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.h File Reference	313
7.122 Mesh.h	314
7.123 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.cpp File Reference	314
7.124 Model.cpp	314
7.125 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.h File Reference	315
7.126 Model.h	316
7.127 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.cpp File Reference	316
7.127.1 Macro Definition Documentation	317
7.127.1.1 TINYOBJLOADER_IMPLEMENTATION	317
7.128 ObjLoader.cpp	317
7.129 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.h File Reference	319
7.130 ObjLoader.h	319
7.131 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.cpp File Reference	320
7.132 ObjMaterial.cpp	320
7.133 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.h File Reference	321
7.134 ObjMaterial.h	321
7.135 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.cpp File Reference	322
7.136 Quad.cpp	322
7.137 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.h File Reference	322
7.138 Quad.h	323
7.139 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Rotation.h File Reference	323
7.140 Rotation.h	323
7.141 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.cpp File Reference	324
7.142 Scene.cpp	324
7.143 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.h File Reference	325
7.144 Scene.h	326
7.145 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.cpp File Reference	327
7.146 ShadowMap.cpp	327
7.147 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.h File Reference	328
7.148 ShadowMap.h	328
7.149 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.cpp File Reference	328
7.150 SkyBox.cpp	328

7.151 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.h File Reference	331
7.152 SkyBox.h	331
7.153 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.cpp File Reference	332
7.154 ClampToEdgeMode.cpp	332
7.155 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.h File Reference	332
7.156 ClampToEdgeMode.h	332
7.157 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/MirroredRepeatMode.cpp File Reference	332
7.158 MirroredRepeatMode.cpp	333
7.159 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/MirroredRepeatMode.h File Reference	333
7.160 MirroredRepeatMode.h	333
7.161 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/RepeatMode.cpp File Reference	333
7.162 RepeatMode.cpp	334
7.163 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/RepeatMode.h File Reference	334
7.164 RepeatMode.h	334
7.165 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Texture.cpp File Reference	334
7.165.1 Macro Definition Documentation	335
7.165.1.1 STB_IMAGE_IMPLEMENTATION	335
7.166 Texture.cpp	335
7.167 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Texture.h File Reference	337
7.168 Texture.h	337
7.169 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/TextureWrappingMode.h File Reference	338
7.170 TextureWrappingMode.h	338
7.171 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Vertex.h File Reference	338
7.172 Vertex.h	339
7.173 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.cpp File Reference	340
7.174 ViewFrustumCulling.cpp	340
7.175 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.h File Reference	344
7.176 ViewFrustumCulling.h	344
7.177 C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.cpp File Reference	345
7.178 File.cpp	345
7.179 C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.h File Reference	346
7.180 File.h	346
7.181 C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumbers.cpp File Reference	346
7.182 RandomNumbers.cpp	346
7.183 C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumbers.h File Reference	347
7.184 RandomNumbers.h	347
7.185 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.cpp File Reference	348
7.186 Window.cpp	348
7.187 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.h File Reference	350
7.188 Window.h	350
Index	353

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

std	11
-------------------------------	--------------------

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AABB	13
Camera	18
CascadedShadowMap	28
Clouds	35
DebugApp	50
File	64
ViewFrustumCulling::frustum_plane	66
GameObject	67
GBuffer	72
GUI	82
std::hash< Vertex >	92
Light	93
DirectionalLight	53
PointLight	144
LoadingScreen	102
Mesh	105
Model	111
Noise	116
ObjLoader	126
ObjMaterial	131
Quad	149
RandomNumbers	152
Renderer	154
RenderPass	159
LightingPass	96
RenderPassSceneDependend	160
DirectionalShadowMapPass	61
GeometryPass	77
OmniShadowMapPass	141
Rotation	162
Scene	163
ShaderIncludes	172
ShaderProgram	175
ComputeShaderProgram	49

GeometryPassShaderProgram	81
LightingPassShaderProgram	101
OmniDirShadowShaderProgram	140
ShadowMap	185
OmniDirShadowMap	138
SkyBox	189
Texture	194
TextureWrappingMode	201
ClampToEdgeMode	34
MirroredRepeatMode	110
RepeatMode	161
Vertex	202
ViewFrustumCulling	205
Window	218

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

AABB	13
Camera	18
CascadedShadowMap	28
ClampToEdgeMode	34
Clouds	35
ComputeShaderProgram	49
DebugApp	50
DirectionalLight	53
DirectionalShadowMapPass	61
File	64
ViewFrustumCulling::frustum_plane	66
GameObject	67
GBuffer	72
GeometryPass	77
GeometryPassShaderProgram	81
GUI	82
std::hash< Vertex >	92
Light	93
LightingPass	96
LightingPassShaderProgram	101
LoadingScreen	102
Mesh	105
MirroredRepeatMode	110
Model	111
Noise	116
ObjLoader	126
ObjMaterial	131
OmniDirShadowMap	138
OmniDirShadowShaderProgram	140
OmniShadowMapPass	141
PointLight	144
Quad	149
RandomNumbers	152
Renderer	154
RenderPass	159

RenderPassSceneDependend	160
RepeatMode	161
Rotation	162
Scene	163
ShaderIncludes	172
ShaderProgram	175
ShadowMap	185
SkyBox	189
Texture	194
TextureWrappingMode	201
Vertex	202
ViewFrustumCulling	205
Window	218

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

C:/Users/jonas/Desktop/GraphicEngine/Src/ bindings.h	232
C:/Users/jonas/Desktop/GraphicEngine/Src/ GlobalValues.h	244
C:/Users/jonas/Desktop/GraphicEngine/Src/ host_device_shared.h	250
C:/Users/jonas/Desktop/GraphicEngine/Src/app/ App.cpp	229
C:/Users/jonas/Desktop/GraphicEngine/Src/camera/ Camera.cpp	235
C:/Users/jonas/Desktop/GraphicEngine/Src/camera/ Camera.h	237
C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ ComputeShaderProgram.cpp	238
C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ ComputeShaderProgram.h	238
C:/Users/jonas/Desktop/GraphicEngine/Src/debug/ DebugApp.cpp	239
C:/Users/jonas/Desktop/GraphicEngine/Src/debug/ DebugApp.h	243
C:/Users/jonas/Desktop/GraphicEngine/Src/gui/ GUI.cpp	246
C:/Users/jonas/Desktop/GraphicEngine/Src/gui/ GUI.h	249
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ Renderer.cpp	265
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ Renderer.h	266
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ RenderPassSceneDependend.cpp	267
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ RenderPassSceneDependend.h	267
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ ShaderIncludes.cpp	268
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ ShaderIncludes.h	269
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ ShaderProgram.cpp	270
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ ShaderProgram.h	274
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GBuffer.cpp	252
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GBuffer.h	254
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GeometryPass.cpp	255
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GeometryPass.h	257
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GeometryPassShaderProgram.cpp	257
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GeometryPassShaderProgram.h	258
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ LightingPass.cpp	258
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ LightingPass.h	261
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ LightingPassShaderProgram.cpp	262
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ LightingPassShaderProgram.h	263
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ RenderPass.h	263
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/ LoadingScreen.cpp	264
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/ LoadingScreen.h	265
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ AABB.cpp	275
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ AABB.h	277

C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ GameObject.cpp	296
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ GameObject.h	297
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Mesh.cpp	312
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Mesh.h	313
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Model.cpp	314
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Model.h	315
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ObjLoader.cpp	316
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ObjLoader.h	319
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ObjMaterial.cpp	320
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ObjMaterial.h	321
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Quad.cpp	322
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Quad.h	322
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Rotation.h	323
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Scene.cpp	324
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Scene.h	325
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ Vertex.h	338
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ViewFrustumCulling.cpp	340
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ ViewFrustumCulling.h	344
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Clouds.cpp	278
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Clouds.h	279
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Noise.cpp	280
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Noise.h	284
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Perlin.cpp	285
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/ Perlin.h	287
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/ Light.cpp	305
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/ Light.h	306
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ CascadedShadowMap.cpp	298
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ CascadedShadowMap.h	299
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ DirectionalLight.cpp	300
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ DirectionalLight.h	303
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ DirectionalShadowMapPass.cpp	304
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/ DirectionalShadowMapPass.h	305
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniDirShadowMap.cpp	306
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniDirShadowMap.h	307
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniDirShadowShaderProgram.cpp	308
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniDirShadowShaderProgram.h	308
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniShadowMapPass.cpp	309
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ OmniShadowMapPass.h	310
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ PointLight.cpp	310
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/ PointLight.h	311
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ ShadowMap.cpp	327
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ ShadowMap.h	328
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/ SkyBox.cpp	328
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/ SkyBox.h	331
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ ClampToEdgeMode.cpp	332
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ ClampToEdgeMode.h	332
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ MirroredRepeatMode.cpp	332
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ MirroredRepeatMode.h	333
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ RepeatMode.cpp	333
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ RepeatMode.h	334
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ Texture.cpp	334
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ Texture.h	337
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ TextureWrappingMode.h	338
C:/Users/jonas/Desktop/GraphicEngine/Src/util/ File.cpp	345
C:/Users/jonas/Desktop/GraphicEngine/Src/util/ File.h	346
C:/Users/jonas/Desktop/GraphicEngine/Src/util/ RandomNumbers.cpp	346
C:/Users/jonas/Desktop/GraphicEngine/Src/util/ RandomNumbers.h	347
C:/Users/jonas/Desktop/GraphicEngine/Src/window/ Window.cpp	348

Chapter 5

Namespace Documentation

5.1 std Namespace Reference

Data Structures

- struct `hash< Vertex >`

Chapter 6

Data Structure Documentation

6.1 AABB Class Reference

```
#include <AABB.h>
```

Collaboration diagram for AABB:

Public Member Functions

- [AABB \(\)](#)
- [std::vector< glm::vec3 > get_corners \(glm::mat4 model\)](#)
- [void init \(GLfloat minX, GLfloat maxX, GLfloat minY, GLfloat maxY, GLfloat minZ, GLfloat maxZ\)](#)
- [glm::vec3 get_radius \(\)](#)
- [void render \(\)](#)
- [~AABB \(\)](#)

Private Attributes

- [std::vector< Vertex > vertices](#)
- [std::vector< unsigned int > indices](#)
- [std::shared_ptr< Mesh > mesh](#)
- [std::vector< glm::vec3 > corners](#)
- [GLfloat minX](#)
- [GLfloat maxX](#)
- [GLfloat minY](#)
- [GLfloat maxY](#)
- [GLfloat minZ](#)
- [GLfloat maxZ](#)

6.1.1 Detailed Description

Definition at line 13 of file [AABB.h](#).

6.1.2 Constructor & Destructor Documentation

6.1.2.1 **AABB()**

```
AABB::AABB ( )
```

Definition at line 4 of file [AABB.cpp](#).
00004 { }

6.1.2.2 **~AABB()**

```
AABB::~AABB ( )
```

Definition at line 114 of file [AABB.cpp](#).
00114 { }

6.1.3 Member Function Documentation

6.1.3.1 **get_corners()**

```
std::vector< glm::vec3 > AABB::get_corners (
    glm::mat4 model )
```

Definition at line 6 of file [AABB.cpp](#).

```
00007 {
00008     std::vector<glm::vec3> corners_world_space;
00009
0010     for (glm::vec3 corner : corners) {
0011
0012         corners_world_space.push_back(glm::vec3(model * glm::vec4(corner, 1.0f)));
0013     }
0014
0015     return corners_world_space;
0016 }
```

References [corners](#).

6.1.3.2 **get_radius()**

```
glm::vec3 AABB::get_radius ( )
```

Definition at line 110 of file [AABB.cpp](#).

```
00110 { return glm::vec3(std::abs(maxX - minX), std::abs(maxY - minY), std::abs(maxZ - minZ)); }
```

References [maxX](#), [maxY](#), [maxZ](#), [minX](#), [minY](#), and [minZ](#).

6.1.3.3 init()

```
void AABB::init (
    GLfloat minX,
    GLfloat maxX,
    GLfloat minY,
    GLfloat maxY,
    GLfloat minZ,
    GLfloat maxZ )
```

Definition at line 18 of file [AABB.cpp](#).

```
00019 {
00020
00021
00022     this->minX = minX;
00023     this->maxX = maxX;
00024     this->minY = minY;
00025     this->maxY = maxY;
00026     this->minZ = minZ;
00027     this->maxZ = maxZ;
00028
00029     corners.push_back(glm::vec3(minX, minY, minZ));
00030     corners.push_back(glm::vec3(minX, minY, maxZ));
00031     corners.push_back(glm::vec3(minX, maxY, minZ));
00032     corners.push_back(glm::vec3(minX, maxY, maxZ));
00033     corners.push_back(glm::vec3(maxX, minY, minZ));
00034     corners.push_back(glm::vec3(maxX, minY, maxZ));
00035     corners.push_back(glm::vec3(maxX, maxY, minZ));
00036     corners.push_back(glm::vec3(maxX, maxY, maxZ));
00037
00038     // 0: left bottom front
00039     vertices.push_back(Vertex(glm::vec3(minX, minY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00040                               glm::vec2(0.f)));
00040     // 1: right bottom front
00041     vertices.push_back(Vertex(glm::vec3(maxX, minY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00042                               glm::vec2(0.f)));
00042     // 2: left top front
00043     vertices.push_back(Vertex(glm::vec3(minX, maxY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00044                               glm::vec2(0.f)));
00044     // 3: right top front
00045     vertices.push_back(Vertex(glm::vec3(maxX, maxY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00046                               glm::vec2(0.f)));
00046     // 4: left bottom far
00047     vertices.push_back(Vertex(glm::vec3(minX, minY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00048                               glm::vec2(0.f)));
00048     // 5: right bottom far
00049     vertices.push_back(Vertex(glm::vec3(maxX, minY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00050                               glm::vec2(0.f)));
00050     // 6: left top far
00051     vertices.push_back(Vertex(glm::vec3(minX, maxY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00052                               glm::vec2(0.f)));
00052     // 7: right top far
00053     vertices.push_back(Vertex(glm::vec3(maxX, maxY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00054                               glm::vec2(0.f)));
00054
00055     indices = { // note that we start from 0!
00056
00057         //left
00058         4,
00059         2,
00060         6,
00061         4,
00062         0,
00063         2,
00064
00065         //right
00066         3,
00067         5,
00068         7,
00069         5,
00070         3,
00071         1,
00072
00073         //top
00074         2,
00075         3,
00076         6,
00077         6,
00078         3,
00079         7,
00080
00081         //bottom
```

```
00082     4,  
00083     1,  
00084     0,  
00085     5,  
00086     1,  
00087     4,  
00088  
00089 //back  
00090     7,  
00091     4,  
00092     6,  
00093     5,  
00094     4,  
00095     7,  
00096  
00097 //front  
00098     0,  
00099     3,  
00100     2,  
00101     0,  
00102     1,  
00103     3  
00104  
00105 };  
00106  
00107 mesh = std::make_shared<Mesh>(vertices, indices);  
00108 }
```

References [corners](#), [indices](#), [maxX](#), [maxY](#), [maxZ](#), [mesh](#), [minX](#), [minY](#), [minZ](#), and [vertices](#).

Referenced by [Clouds::Clouds\(\)](#).

Here is the caller graph for this function:

6.1.3.4 render()

```
void AABB::render ( )
```

Definition at line 112 of file [AABB.cpp](#).
00112 { mesh->render(); }

References [mesh](#).

Referenced by [Clouds::render\(\)](#).

Here is the caller graph for this function:

6.1.4 Field Documentation

6.1.4.1 corners

```
std::vector<glm::vec3> AABB::corners [private]
```

Definition at line 33 of file [AABB.h](#).

Referenced by [get_corners\(\)](#), and [init\(\)](#).

6.1.4.2 indices

```
std::vector<unsigned int> AABB::indices [private]
```

Definition at line 29 of file [AABB.h](#).

Referenced by [init\(\)](#).

6.1.4.3 maxX

```
GLfloat AABB::maxX [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.4 maxY

```
GLfloat AABB::maxY [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.5 maxZ

```
GLfloat AABB::maxZ [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.6 mesh

```
std::shared_ptr<Mesh> AABB::mesh [private]
```

Definition at line 31 of file [AABB.h](#).

Referenced by [init\(\)](#), and [render\(\)](#).

6.1.4.7 minX

```
GLfloat AABB::minX [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.8 minY

```
GLfloat AABB::minY [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.9 minZ

```
GLfloat AABB::minZ [private]
```

Definition at line 35 of file [AABB.h](#).

Referenced by [get_radius\(\)](#), and [init\(\)](#).

6.1.4.10 vertices

```
std::vector<Vertex> AABB::vertices [private]
```

Definition at line 28 of file [AABB.h](#).

Referenced by [init\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[AABB.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[AABB.cpp](#)

6.2 Camera Class Reference

```
#include <Camera.h>
```

Collaboration diagram for Camera:

Public Member Functions

- `Camera ()`
- `Camera (glm::vec3 start_position, glm::vec3 start_up, GLfloat start_yaw, GLfloat start_pitch, GLfloat start_near_plane, GLfloat start_far_plane, GLfloat start_fov)`
- `void key_control (bool *keys, GLfloat delta_time)`
- `void mouse_control (GLfloat x_change, GLfloat y_change)`
- `glm::vec3 get_camera_position () const`
- `glm::vec3 get_camera_direction () const`
- `glm::vec3 get_up_axis () const`
- `glm::vec3 get_right_axis () const`
- `GLfloat get_near_plane () const`
- `GLfloat get_far_plane () const`
- `GLfloat get_fov () const`
- `GLfloat get_yaw () const`
- `glm::mat4 get_viewmatrix () const`
- `void set_near_plane (GLfloat near_plane)`
- `void set_far_plane (GLfloat far_plane)`
- `void set_fov (GLfloat fov)`
- `void set_camera_position (glm::vec3 new_camera_position)`
- `~Camera ()`

Private Member Functions

- `void update ()`

Private Attributes

- `glm::vec3 position`
- `glm::vec3 front`
- `glm::vec3 world_up`
- `glm::vec3 right`
- `glm::vec3 up`
- `GLfloat yaw`
- `GLfloat pitch`
- `GLfloat movement_speed`
- `GLfloat turn_speed`
- `GLfloat near_plane`
- `GLfloat far_plane`
- `GLfloat fov`

6.2.1 Detailed Description

Definition at line 8 of file [Camera.h](#).

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Camera() [1/2]

```
Camera::Camera ( )
```

Definition at line 3 of file Camera.cpp.

```
00003      :
00004
00005     position(glm::vec3(0.0f, 50.0f, 0.0f)),
00006     //here we want the normal coord. axis z is showing to us !!
00007     front(glm::vec3(0.0f, 0.0f, -1.0f)), world_up(glm::vec3(0.0f, 1.0f, 0.0f)),
00008     right(glm::normalize(glm::cross(front, world_up))),
00009     up(glm::normalize(glm::cross(right, front))), yaw(-60.0f), pitch(0.0f), movement_speed(35.0f),
00010     turn_speed(0.25f), near_plane(0.1f), far_plane(1000.f),
00011     fov(45.f)
00012 }
```

6.2.2.2 Camera() [2/2]

```
Camera::Camera (
    glm::vec3 start_position,
    glm::vec3 start_up,
    GLfloat start_yaw,
    GLfloat start_pitch,
    GLfloat start_move_speed,
    GLfloat start_turn_speed,
    GLfloat near_plane,
    GLfloat far_plane,
    GLfloat fov )
```

Definition at line 14 of file Camera.cpp.

```
00015      :
00016
00017     position(start_position),
00018     //here we want the normal coord. axis z is showing to us !!
00019     front(glm::vec3(0.0f, 0.0f, -1.0f)), world_up(start_up), right(glm::normalize(glm::cross(front,
00020     world_up))), up(glm::normalize(glm::cross(right, front))), yaw(start_yaw), pitch(start_pitch),
00021     movement_speed(start_move_speed), turn_speed(start_turn_speed), near_plane(near_plane), far_plane(far_plane),
00022     fov(fov)
00023 }
```

6.2.2.3 ~Camera()

```
Camera::~Camera ( )
```

Definition at line 98 of file Camera.cpp.

```
00098 { }
```

6.2.3 Member Function Documentation

6.2.3.1 get_camera_direction()

```
glm::vec3 Camera::get_camera_direction() const [inline]
```

Definition at line 20 of file [Camera.h](#).

```
00020 { return glm::normalize(front); };
```

References [front](#).

6.2.3.2 get_camera_position()

```
glm::vec3 Camera::get_camera_position() const [inline]
```

Definition at line 19 of file [Camera.h](#).

```
00019 { return position; };
```

References [position](#).

6.2.3.3 get_far_plane()

```
GLfloat Camera::get_far_plane() const [inline]
```

Definition at line 24 of file [Camera.h](#).

```
00024 { return far_plane; };
```

References [far_plane](#).

6.2.3.4 get_fov()

```
GLfloat Camera::get_fov() const [inline]
```

Definition at line 25 of file [Camera.h](#).

```
00025 { return fov; };
```

References [fov](#).

6.2.3.5 get_near_plane()

```
GLfloat Camera::get_near_plane() const [inline]
```

Definition at line 23 of file [Camera.h](#).

```
00023 { return near_plane; };
```

References [near_plane](#).

6.2.3.6 get_right_axis()

```
glm::vec3 Camera::get_right_axis ( ) const [inline]
```

Definition at line 22 of file [Camera.h](#).
00022 { **return right;** };

References [right](#).

6.2.3.7 get_up_axis()

```
glm::vec3 Camera::get_up_axis ( ) const [inline]
```

Definition at line 21 of file [Camera.h](#).
00021 { **return up;** };

References [up](#).

6.2.3.8 get_viewmatrix()

```
glm::mat4 Camera::get_viewmatrix ( ) const
```

Definition at line 92 of file [Camera.cpp](#).
00093 {
00094 //very necessary for further calc
00095 **return** glm::lookAt([position](#), [position](#) + [front](#), [up](#));
00096 }

References [front](#), [position](#), and [up](#).

6.2.3.9 get_yaw()

```
GLfloat Camera::get_yaw ( ) const [inline]
```

Definition at line 26 of file [Camera.h](#).
00026 { **return yaw;** };

References [yaw](#).

6.2.3.10 key_control()

```
void Camera::key_control (
    bool * keys,
    GLfloat delta_time )
```

Definition at line 25 of file [Camera.cpp](#).

```
00026 {
00027     GLfloat velocity = movement_speed * delta_time;
00028
00029     if (keys[GLFW_KEY_W]) {
00030         position += front * velocity;
00031     }
00032
00033     if (keys[GLFW_KEY_D]) {
00034         position += right * velocity;
00035     }
00036
00037     if (keys[GLFW_KEY_A]) {
00038         position += -right * velocity;
00039     }
00040
00041     if (keys[GLFW_KEY_S]) {
00042         position += -front * velocity;
00043     }
00044
00045     if (keys[GLFW_KEY_Q]) {
00046         yaw += -velocity;
00047     }
00048
00049     if (keys[GLFW_KEY_E]) {
00050         yaw += velocity;
00051     }
00052
00053 }
```

References [front](#), [movement_speed](#), [position](#), [right](#), and [yaw](#).

6.2.3.11 mouse_control()

```
void Camera::mouse_control (
    GLfloat x_change,
    GLfloat y_change )
```

Definition at line 60 of file [Camera.cpp](#).

```
00061 {
00062
00063     //here we only want to support views 90 degrees to each side
00064     //again choose turn speed well in respect to its ordinal scale
00065     x_change *= turn_speed;
00066     y_change *= turn_speed;
00067
00068     yaw += x_change;
00069     pitch += y_change;
00070
00071     if (pitch > 89.0f) {
00072         pitch = 89.0f;
00073     }
00074
00075     if (pitch < -89.0f) {
00076         pitch = -89.0f;
00077     }
00078
00079     // by changing the rotations you need to update all parameters
00080     // for we retrieve them later for further calculations!
00081     update();
00082 }
```

References [pitch](#), [turn_speed](#), [update\(\)](#), and [yaw](#).

Here is the call graph for this function:

6.2.3.12 set_camera_position()

```
void Camera::set_camera_position (
    glm::vec3 new_camera_position )
```

Definition at line 90 of file [Camera.cpp](#).

```
00090 { this->position = new_camera_position; }
```

References [position](#).

6.2.3.13 set_far_plane()

```
void Camera::set_far_plane (
    GLfloat far_plane )
```

Definition at line 86 of file [Camera.cpp](#).

```
00086 { this->far_plane = far_plane; }
```

References [far_plane](#).

6.2.3.14 set_fov()

```
void Camera::set_fov (
    GLfloat fov )
```

Definition at line 88 of file [Camera.cpp](#).

```
00088 { this->fov = fov; }
```

References [fov](#).

6.2.3.15 set_near_plane()

```
void Camera::set_near_plane (
    GLfloat near_plane )
```

Definition at line 84 of file [Camera.cpp](#).

```
00084 { this->near_plane = near_plane; }
```

References [near_plane](#).

6.2.3.16 update()

```
void Camera::update () [private]
```

Definition at line 100 of file [Camera.cpp](#).

```
00101 {  
00102     //https://learnopengl.com/Getting-started/Camera?fbclid=IwAR1WEr4jt6IyWC52s_WKYHtaFoeug37pG5YqbDPifgn5F1UXPbUjWbJWiq  
00103     // thats a bit tricky; have a look to link above if there a questions :)  
00104     // but simple geometrical analysis  
00105     // consider yaw you are turnig to the side; pitch as you move the head forward and back; roll  
00106     // rotations around z-axis will make you dizzy :))  
00107     front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));  
00108     front.y = sin(glm::radians(pitch));  
00109     front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));  
00110     front = glm::normalize(front);  
00111  
00112     //retrieve the right vector with some world_up  
00113     right = glm::normalize(glm::cross(front, world_up));  
00114  
00115     // but this means the up vector must again be calculated with right vector calculated!!!  
00116     up = glm::normalize(glm::cross(right, front));  
00117 }
```

References [front](#), [pitch](#), [right](#), [up](#), [world_up](#), and [yaw](#).

Referenced by [mouse_control\(\)](#).

Here is the caller graph for this function:

6.2.4 Field Documentation

6.2.4.1 far_plane

```
GLfloat Camera::far_plane [private]
```

Definition at line 50 of file [Camera.h](#).

Referenced by [get_far_plane\(\)](#), and [set_far_plane\(\)](#).

6.2.4.2 fov

```
GLfloat Camera::fov [private]
```

Definition at line 50 of file [Camera.h](#).

Referenced by [get_fov\(\)](#), and [set_fov\(\)](#).

6.2.4.3 front

```
glm::vec3 Camera::front [private]
```

Definition at line 39 of file [Camera.h](#).

Referenced by [get_camera_direction\(\)](#), [get_viewmatrix\(\)](#), [key_control\(\)](#), and [update\(\)](#).

6.2.4.4 movement_speed

```
GLfloat Camera::movement_speed [private]
```

Definition at line 47 of file [Camera.h](#).

Referenced by [key_control\(\)](#).

6.2.4.5 near_plane

```
GLfloat Camera::near_plane [private]
```

Definition at line 50 of file [Camera.h](#).

Referenced by [get_near_plane\(\)](#), and [set_near_plane\(\)](#).

6.2.4.6 pitch

```
GLfloat Camera::pitch [private]
```

Definition at line 45 of file [Camera.h](#).

Referenced by [mouse_control\(\)](#), and [update\(\)](#).

6.2.4.7 position

```
glm::vec3 Camera::position [private]
```

Definition at line 38 of file [Camera.h](#).

Referenced by [get_camera_position\(\)](#), [get_viewmatrix\(\)](#), [key_control\(\)](#), and [set_camera_position\(\)](#).

6.2.4.8 right

```
glm::vec3 Camera::right [private]
```

Definition at line 41 of file [Camera.h](#).

Referenced by [get_right_axis\(\)](#), [key_control\(\)](#), and [update\(\)](#).

6.2.4.9 turn_speed

```
GLfloat Camera::turn_speed [private]
```

Definition at line 48 of file [Camera.h](#).

Referenced by [mouse_control\(\)](#).

6.2.4.10 up

```
glm::vec3 Camera::up [private]
```

Definition at line 42 of file [Camera.h](#).

Referenced by [get_up_axis\(\)](#), [get_viewmatrix\(\)](#), and [update\(\)](#).

6.2.4.11 world_up

```
glm::vec3 Camera::world_up [private]
```

Definition at line 40 of file [Camera.h](#).

Referenced by [update\(\)](#).

6.2.4.12 yaw

```
GLfloat Camera::yaw [private]
```

Definition at line 44 of file [Camera.h](#).

Referenced by [get_yaw\(\)](#), [key_control\(\)](#), [mouse_control\(\)](#), and [update\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/camera/[Camera.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/camera/[Camera.cpp](#)

6.3 CascadedShadowMap Class Reference

```
#include <CascadedShadowMap.h>
```

Collaboration diagram for CascadedShadowMap:

Public Member Functions

- `CascadedShadowMap ()`
- `bool init (GLuint width, GLuint height, GLuint num_cascades)`
- `void write ()`
- `void read (GLenum texture_unit)`
- `void write_light_matrices (std::vector< glm::mat4x4 > &lightMatrices)`
- `void set_pcf_radius (GLuint radius)`
- `void set_intensity (GLfloat intensity)`
- `GLfloat get_intensity () const`
- `GLuint get_shadow_width () const`
- `GLuint get_shadow_height () const`
- `GLuint get_id () const`
- `GLuint get_num_active_cascades () const`
- `GLuint get_pcf_radius () const`
- `~CascadedShadowMap ()`

Protected Attributes

- `GLuint FBO`
- `GLuint shadow_maps`
- `GLuint shadow_width`
- `GLuint shadow_height`
- `GLuint matrices_UBO`
- `GLuint num_active_cascades`
- `GLuint pcf_radius`
- `GLfloat intensity`

6.3.1 Detailed Description

Definition at line 11 of file [CascadedShadowMap.h](#).

6.3.2 Constructor & Destructor Documentation

6.3.2.1 CascadedShadowMap()

```
CascadedShadowMap::CascadedShadowMap ( )
```

Definition at line 5 of file [CascadedShadowMap.cpp](#).

```
00005 :
00006 :
00007     FBO(0), shadow_maps(0), shadow_width(0), shadow_height(0), matrices_UBO(0),
00008     num_active_cascades(0), pcf_radius(1), intensity(1)
00009 {
00010 }
```

6.3.2.2 ~CascadedShadowMap()

```
CascadedShadowMap::~CascadedShadowMap ( )
```

Definition at line 79 of file [CascadedShadowMap.cpp](#).

```
00080 {  
00081     if (FBO) {  
00082         glDeleteFramebuffers(1, &FBO);  
00083     }  
00084  
00085     if (shadow_maps) {  
00086         glDeleteTextures(1, &shadow_maps);  
00087     }  
00088 }
```

References [FBO](#), and [shadow_maps](#).

6.3.3 Member Function Documentation

6.3.3.1 get_id()

```
GLuint CascadedShadowMap::get_id ( ) const [inline]
```

Definition at line 26 of file [CascadedShadowMap.h](#).

```
00026 { return shadow_maps; };
```

References [shadow_maps](#).

6.3.3.2 get_intensity()

```
GLfloat CascadedShadowMap::get_intensity ( ) const [inline]
```

Definition at line 23 of file [CascadedShadowMap.h](#).

```
00023 { return intensity; };
```

References [intensity](#).

6.3.3.3 get_num_active_cascades()

```
GLuint CascadedShadowMap::get_num_active_cascades ( ) const [inline]
```

Definition at line 27 of file [CascadedShadowMap.h](#).

```
00027 { return num_active_cascades; };
```

References [num_active_cascades](#).

6.3.3.4 get_pcf_radius()

```
GLuint CascadedShadowMap::get_pcf_radius () const [inline]
```

Definition at line 28 of file [CascadedShadowMap.h](#).

```
00028 { return pcf_radius; }
```

References [pcf_radius](#).

6.3.3.5 get_shadow_height()

```
GLuint CascadedShadowMap::get_shadow_height () const [inline]
```

Definition at line 25 of file [CascadedShadowMap.h](#).

```
00025 { return shadow_height; }
```

References [shadow_height](#).

6.3.3.6 get_shadow_width()

```
GLuint CascadedShadowMap::get_shadow_width () const [inline]
```

Definition at line 24 of file [CascadedShadowMap.h](#).

```
00024 { return shadow_width; }
```

References [shadow_width](#).

6.3.3.7 init()

```
bool CascadedShadowMap::init (
    GLuint width,
    GLuint height,
    GLuint num_cascades )
```

Definition at line 12 of file [CascadedShadowMap.cpp](#).

```
00013 {
00014
00015     shadow_width = width;
00016     shadow_height = height;
00017
00018     num_active_cascades = num_cascades;
00019
00020     glGenFramebuffers(1, &FBO);
00021     glGenTextures(1, &shadow_maps);
00022     glBindTexture(GL_TEXTURE_2D_ARRAY, shadow_maps);
00023     glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_DEPTH_COMPONENT32F, shadow_width, shadow_height,
00024                 NUM_CASCADES, 0, GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00025
00026     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00027     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00028     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
00029     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
00030
00031     constexpr float bordercolor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
00032     glTexParameterfv(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_BORDER_COLOR, bordercolor);
```

```

00032     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00033     glBindFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, shadow_maps, 0);
00034     glDrawBuffer(GL_NONE);
00035     glReadBuffer(GL_NONE);
00036
00037
00038     int status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00039     if (status != GL_FRAMEBUFFER_COMPLETE) {
00040         std::cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!";
00041         throw 0;
00042     }
00043
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045
00046     // setting up our buffer for the light matrices
00047     // for every cascade we will have 1 matrix in the geometry shader
00048     glGenBuffers(1, &matrices_UBO);
00049     glBindBuffer(GL_UNIFORM_BUFFER, matrices_UBO);
00050     glBufferData(GL_UNIFORM_BUFFER, sizeof(glm::mat4) * NUM_CASCADES, nullptr, GL_DYNAMIC_DRAW);
00051     glBindBufferBase(GL_UNIFORM_BUFFER, UNIFORM_LIGHT_MATRICES_BINDING, matrices_UBO);
00052     glBindBuffer(GL_UNIFORM_BUFFER, 0);
00053
00054     return true;
00055 }
```

References `FBO`, `matrices_UBO`, `num_active_cascades`, `NUM_CASCADES`, `shadow_height`, `shadow_maps`, `shadow_width`, and `UNIFORM_LIGHT_MATRICES_BINDING`.

6.3.3.8 `read()`

```
void CascadedShadowMap::read (
    GLenum texture_unit )
```

Definition at line 69 of file `CascadedShadowMap.cpp`.

```

00070 {
00071     glBindTexture(GL_TEXTURE_2D_ARRAY, shadow_maps);
00072 }
```

References `shadow_maps`.

6.3.3.9 `set_intensity()`

```
void CascadedShadowMap::set_intensity (
    GLfloat intensity )
```

Definition at line 77 of file `CascadedShadowMap.cpp`.

```
00077 { this->intensity = intensity; }
```

References `intensity`.

6.3.3.10 `set_pcf_radius()`

```
void CascadedShadowMap::set_pcf_radius (
    GLuint radius )
```

Definition at line 75 of file `CascadedShadowMap.cpp`.

```
00075 { pcf_radius = radius; }
```

References `pcf_radius`.

6.3.3.11 write()

```
void CascadedShadowMap::write ( )
```

Definition at line 67 of file [CascadedShadowMap.cpp](#).
 00067 { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }

References [FBO](#).

6.3.3.12 write_light_matrices()

```
void CascadedShadowMap::write_light_matrices (
    std::vector< glm::mat4x4 > & lightMatrices )
```

Definition at line 57 of file [CascadedShadowMap.cpp](#).

```
00058 {
00059
00060     glBindBuffer(GL_UNIFORM_BUFFER, matrices_UBO);
00061     for (size_t i = 0; i < lightMatrices.size(); ++i) {
00062         glBufferSubData(GL_UNIFORM_BUFFER, i * sizeof(glm::mat4x4), sizeof(glm::mat4x4),
00063                         &lightMatrices[i]);
00064     glBindBuffer(GL_UNIFORM_BUFFER, 0);
00065 }
```

References [matrices_UBO](#).

6.3.4 Field Documentation

6.3.4.1 FBO

```
GLuint CascadedShadowMap::FBO [protected]
```

Definition at line 33 of file [CascadedShadowMap.h](#).

Referenced by [init\(\)](#), [write\(\)](#), and [~CascadedShadowMap\(\)](#).

6.3.4.2 intensity

```
GLfloat CascadedShadowMap::intensity [protected]
```

Definition at line 43 of file [CascadedShadowMap.h](#).

Referenced by [get_intensity\(\)](#), and [set_intensity\(\)](#).

6.3.4.3 `matrices_UBO`

```
GLuint CascadedShadowMap::matrices_UBO [protected]
```

Definition at line 37 of file [CascadedShadowMap.h](#).

Referenced by [init\(\)](#), and [write_light_matrices\(\)](#).

6.3.4.4 `num_active_cascades`

```
GLuint CascadedShadowMap::num_active_cascades [protected]
```

Definition at line 39 of file [CascadedShadowMap.h](#).

Referenced by [get_num_active_cascades\(\)](#), and [init\(\)](#).

6.3.4.5 `pcf_radius`

```
GLuint CascadedShadowMap::pcf_radius [protected]
```

Definition at line 41 of file [CascadedShadowMap.h](#).

Referenced by [get_pcf_radius\(\)](#), and [set_pcf_radius\(\)](#).

6.3.4.6 `shadow_height`

```
GLuint CascadedShadowMap::shadow_height [protected]
```

Definition at line 35 of file [CascadedShadowMap.h](#).

Referenced by [get_shadow_height\(\)](#), and [init\(\)](#).

6.3.4.7 `shadow_maps`

```
GLuint CascadedShadowMap::shadow_maps [protected]
```

Definition at line 33 of file [CascadedShadowMap.h](#).

Referenced by [get_id\(\)](#), [init\(\)](#), [read\(\)](#), and [~CascadedShadowMap\(\)](#).

6.3.4.8 shadow_width

```
GLuint CascadedShadowMap::shadow_width [protected]
```

Definition at line 35 of file [CascadedShadowMap.h](#).

Referenced by [get_shadow_width\(\)](#), and [init\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[CascadedShadowMap.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[CascadedShadowMap.cpp](#)

6.4 ClampToEdgeMode Class Reference

```
#include <ClampToEdgeMode.h>
```

Inheritance diagram for ClampToEdgeMode:

Collaboration diagram for ClampToEdgeMode:

Public Member Functions

- [ClampToEdgeMode \(\)](#)
- void [activate \(\)](#) override
- [~ClampToEdgeMode \(\)](#)

6.4.1 Detailed Description

Definition at line 4 of file [ClampToEdgeMode.h](#).

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ClampToEdgeMode()

```
ClampToEdgeMode::ClampToEdgeMode ( )
```

Definition at line 3 of file [ClampToEdgeMode.cpp](#).
00003 { }

6.4.2.2 ~ClampToEdgeMode()

```
ClampToEdgeMode::~ClampToEdgeMode ( )
```

Definition at line 11 of file [ClampToEdgeMode.cpp](#).
00011 { }

6.4.3 Member Function Documentation

6.4.3.1 activate()

```
void ClampToEdgeMode::activate ( ) [override], [virtual]
```

Implements [TextureWrappingMode](#).

Definition at line 5 of file [ClampToEdgeMode.cpp](#).

```
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00009 }
```

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[ClampToEdgeMode.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[ClampToEdgeMode.cpp](#)

6.5 Clouds Class Reference

```
#include <Clouds.h>
```

Collaboration diagram for Clouds:

Public Member Functions

- [Clouds \(\)](#)
- void [render](#) (glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width, GLuint window_height)
- void [read \(\)](#)
- void [create_noise_textures \(\)](#)
- glm::mat4 [get_model \(\) const](#)
- glm::vec3 [get_movement_direction \(\) const](#)
- glm::vec3 [get_radius \(\) const](#)
- glm::mat4 [get_normal_model \(\) const](#)
- glm::vec3 [get_mesh_scale \(\) const](#)
- GLfloat [get_movement_speed \(\) const](#)
- GLfloat [get_density \(\) const](#)
- GLfloat [get_scale \(\) const](#)
- GLfloat [get_pillowness \(\) const](#)

- GLfloat `get_cirrus_effect () const`
- GLuint `get_num_march_steps () const`
- GLuint `get_num_march_steps_to_light () const`
- bool `get_powder_effect () const`
- std::shared_ptr< `ShaderProgram` > `get_shader_program () const`
- void `set_powder_effect (bool cloud_powder_effect)`
- void `set_cirrus_effect (GLfloat cirrus_effect)`
- void `set_pillowness (GLfloat cloud_pillowness)`
- void `set_scale (GLfloat scale)`
- void `set_density (GLfloat density)`
- void `set_movement_speed (GLfloat speed)`
- void `set_scale (glm::vec3 scale)`
- void `set_translation (glm::vec3 translation)`
- void `set_movement_direction (glm::vec3 movement_dir)`
- void `set_num_march_steps (GLuint num_march_steps)`
- void `set_num_march_steps_to_light (GLuint num_march_steps_to_light)`
- `~Clouds ()`

Private Attributes

- std::shared_ptr< `ShaderProgram` > `shader_program`
- std::shared_ptr< `Noise` > `noise`
- std::shared_ptr< `RandomNumbers` > `random_numbers`
- glm::mat4 `model`
- `AABB aabb`
- GLfloat `minX`
- GLfloat `maxX`
- GLfloat `minY`
- GLfloat `maxY`
- GLfloat `minZ`
- GLfloat `maxZ`
- glm::vec3 `movement_direction`
- glm::vec3 `scale_factor`
- glm::vec3 `translation`
- GLfloat `movement_speed`
- GLfloat `density`
- GLfloat `scale`
- GLfloat `pillowness`
- GLfloat `cirrus_effect`
- GLuint `num_march_steps`
- GLuint `num_march_steps_to_light`
- bool `powder_effect`

6.5.1 Detailed Description

Definition at line 9 of file `Clouds.h`.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Clouds()

Clouds::Clouds ()

Definition at line 3 of file [Clouds.cpp](#).

```
00003 :
00004
00005     shader_program(std::make_shared<ShaderProgram>(ShaderProgram{})),
00006     noise(std::make_shared<Noise>()), random_numbers(std::make_shared<RandomNumbers>()),
00007     model(glm::mat4(1.f)), aabb(AABB()),
00008     minX(-1.f), maxX(1.f), minY(-1.f), maxY(1.f), minZ(-1.f), maxZ(1.f),
00009     movement_direction(glm::vec3(0.0f, 0.0f, 1.0f)), scale_factor(glm::vec3(1.f)),
00010     translation(glm::vec3(0.0f)),
00011     movement_speed(0.65f), density(0.7f), scale(0.63f), pillowness(1.0f), cirrus_effect(0.0f),
00012     num_march_steps(8), num_march_steps_to_light(3),
00013     powder_effect(true)
00014 {
00015
00016     aabb.init(minX, maxX, minY, maxY, minZ, maxZ);
00017 }
```

References [aabb](#), [create_noise_textures\(\)](#), [AABB::init\(\)](#), [maxX](#), [maxY](#), [maxZ](#), [minX](#), [minY](#), [minZ](#), and [shader_program](#).

Here is the call graph for this function:

6.5.2.2 ~Clouds()

Clouds::~Clouds ()

Definition at line 86 of file [Clouds.cpp](#).

```
00086 { }
```

6.5.3 Member Function Documentation

6.5.3.1 create_noise_textures()

void Clouds::create_noise_textures ()

Definition at line 50 of file [Clouds.cpp](#).

```
00051 {
00052     noise->create_res128_noise();
00053     noise->create_res32_noise();
00054 }
```

References [noise](#).

Referenced by [Clouds\(\)](#).

Here is the caller graph for this function:

6.5.3.2 get_cirrus_effect()

```
GLfloat Clouds::get_cirrus_effect () const [inline]
```

Definition at line 28 of file [Clouds.h](#).
00028 { **return** cirrus_effect; };

References [cirrus_effect](#).

6.5.3.3 get_density()

```
GLfloat Clouds::get_density () const [inline]
```

Definition at line 25 of file [Clouds.h](#).
00025 { **return** density; };

References [density](#).

6.5.3.4 get_mesh_scale()

```
glm::vec3 Clouds::get_mesh_scale () const [inline]
```

Definition at line 23 of file [Clouds.h](#).
00023 { **return** scale_factor; };

References [scale_factor](#).

6.5.3.5 get_model()

```
glm::mat4 Clouds::get_model () const
```

Definition at line 72 of file [Clouds.cpp](#).
00073 {
00074 glm::mat4 model = glm::mat4(1.f);
00075 model = glm::translate(model, translation);
00076 model = glm::scale(model, scale_factor);
00077 **return** model;
00078 }

References [model](#), [scale_factor](#), and [translation](#).

Referenced by [render\(\)](#).

Here is the caller graph for this function:

6.5.3.6 get_movement_direction()

```
glm::vec3 Clouds::get_movement_direction () const [inline]
```

Definition at line 20 of file [Clouds.h](#).

```
00020 { return movement_direction; };
```

References [movement_direction](#).

6.5.3.7 get_movement_speed()

```
GLfloat Clouds::get_movement_speed () const [inline]
```

Definition at line 24 of file [Clouds.h](#).

```
00024 { return movement_speed; };
```

References [movement_speed](#).

6.5.3.8 get_normal_model()

```
glm::mat4 Clouds::get_normal_model () const [inline]
```

Definition at line 22 of file [Clouds.h](#).

```
00022 { return glm::transpose(glm::inverse(model)); };
```

References [model](#).

6.5.3.9 get_num_march_steps()

```
GLuint Clouds::get_num_march_steps () const [inline]
```

Definition at line 29 of file [Clouds.h](#).

```
00029 { return num_march_steps; };
```

References [num_march_steps](#).

6.5.3.10 get_num_march_steps_to_light()

```
GLuint Clouds::get_num_march_steps_to_light () const [inline]
```

Definition at line 30 of file [Clouds.h](#).

```
00030 { return num_march_steps_to_light; };
```

References [num_march_steps_to_light](#).

6.5.3.11 get_pillowness()

```
GLfloat Clouds::get_pillowness () const [inline]
```

Definition at line 27 of file [Clouds.h](#).
00027 { **return** pillowness; };

References [pillowness](#).

6.5.3.12 get_powder_effect()

```
bool Clouds::get_powder_effect () const [inline]
```

Definition at line 31 of file [Clouds.h](#).
00031 { **return** powder_effect; };

References [powder_effect](#).

6.5.3.13 get_radius()

```
glm::vec3 Clouds::get_radius () const [inline]
```

Definition at line 21 of file [Clouds.h](#).
00021 { **return** scale_factor / 2.f; };

References [scale_factor](#).

6.5.3.14 get_scale()

```
GLfloat Clouds::get_scale () const [inline]
```

Definition at line 26 of file [Clouds.h](#).
00026 { **return** scale; };

References [scale](#).

6.5.3.15 get_shader_program()

```
std::shared_ptr< ShaderProgram > Clouds::get_shader_program () const [inline]
```

Definition at line 33 of file [Clouds.h](#).
00033 { **return** shader_program; };

References [shader_program](#).

6.5.3.16 read()

```
void Clouds::read ( )
```

Definition at line 42 of file [Clouds.cpp](#).

```
00043 {
00044
00045     random_numbers->read();
00046     noise->read_res128_noise();
00047     noise->read_res32_noise();
00048 }
```

References [noise](#), and [random_numbers](#).

6.5.3.17 render()

```
void Clouds::render (
    glm::mat4 projection_matrix,
    glm::mat4 view_matrix,
    GLuint window_width,
    GLuint window_height )
```

Definition at line 27 of file [Clouds.cpp](#).

```
00028 {
00029
00030     shader_program->use_shader_program();
00031     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00032     shader_program->setUniformMatrix4fv(view_matrix, "view");
00033     shader_program->setUniformMatrix4fv(get_model(), "model");
00034
00035     shader_program->validate_program();
00036
00037 //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
00038 aabb.render();
00039 //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
00040 }
```

References [aabb](#), [get_model\(\)](#), [AABB::render\(\)](#), and [shader_program](#).

Here is the call graph for this function:

6.5.3.18 set_cirrus_effect()

```
void Clouds::set_cirrus_effect (
    GLfloat cirrus_effect )
```

Definition at line 58 of file [Clouds.cpp](#).

```
00058 { this->cirrus_effect = cirrus_effect; }
```

References [cirrus_effect](#).

6.5.3.19 set_density()

```
void Clouds::set_density (
    GLfloat density )
```

Definition at line 64 of file [Clouds.cpp](#).
00064 { this->density = density; }

References [density](#).

6.5.3.20 set_movement_direction()

```
void Clouds::set_movement_direction (
    glm::vec3 movement_dir )
```

Definition at line 80 of file [Clouds.cpp](#).
00080 { this->movement_direction = movement_dir; }

References [movement_direction](#).

6.5.3.21 set_movement_speed()

```
void Clouds::set_movement_speed (
    GLfloat speed )
```

Definition at line 66 of file [Clouds.cpp](#).
00066 { movement_speed = speed; }

References [movement_speed](#).

6.5.3.22 set_num_march_steps()

```
void Clouds::set_num_march_steps (
    GLuint num_march_steps )
```

Definition at line 82 of file [Clouds.cpp](#).
00082 { this->num_march_steps = num_march_steps; }

References [num_march_steps](#).

6.5.3.23 set_num_march_steps_to_light()

```
void Clouds::set_num_march_steps_to_light (
    GLuint num_march_steps_to_light )
```

Definition at line 84 of file [Clouds.cpp](#).

```
00084 { this->num_march_steps_to_light = num_march_steps_to_light; }
```

References [num_march_steps_to_light](#).

6.5.3.24 set_pillowness()

```
void Clouds::set_pillowness (
    GLfloat cloud_pillowness )
```

Definition at line 60 of file [Clouds.cpp](#).

```
00060 { this->pillowness = cloud_pillowness; }
```

References [pillowness](#).

6.5.3.25 set_powder_effect()

```
void Clouds::set_powder_effect (
    bool cloud_powder_effect )
```

Definition at line 56 of file [Clouds.cpp](#).

```
00056 { this->powder_effect = cloud_powder_effect; }
```

References [powder_effect](#).

6.5.3.26 set_scale() [1/2]

```
void Clouds::set_scale (
    GLfloat scale )
```

Definition at line 62 of file [Clouds.cpp](#).

```
00062 { this->scale = scale; }
```

References [scale](#).

6.5.3.27 set_scale() [2/2]

```
void Clouds::set_scale (
    glm::vec3 scale )
```

Definition at line 68 of file [Clouds.cpp](#).
00068 { scale_factor = scale; }

References [scale](#), and [scale_factor](#).

6.5.3.28 set_translation()

```
void Clouds::set_translation (
    glm::vec3 translation )
```

Definition at line 70 of file [Clouds.cpp](#).
00070 { this->translation = translation; }

References [translation](#).

6.5.4 Field Documentation

6.5.4.1 aabb

[AABB](#) Clouds::aabb [private]

Definition at line 55 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#), and [render\(\)](#).

6.5.4.2 cirrus_effect

GLfloat Clouds::cirrus_effect [private]

Definition at line 62 of file [Clouds.h](#).

Referenced by [get_cirrus_effect\(\)](#), and [set_cirrus_effect\(\)](#).

6.5.4.3 density

```
GLfloat Clouds::density [private]
```

Definition at line 62 of file [Clouds.h](#).

Referenced by [get_density\(\)](#), and [set_density\(\)](#).

6.5.4.4 maxX

```
GLfloat Clouds::maxX [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.5 maxY

```
GLfloat Clouds::maxY [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.6 maxZ

```
GLfloat Clouds::maxZ [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.7 minX

```
GLfloat Clouds::minX [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.8 minY

```
GLfloat Clouds::minY [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.9 minZ

```
GLfloat Clouds::minZ [private]
```

Definition at line 57 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#).

6.5.4.10 model

```
glm::mat4 Clouds::model [private]
```

Definition at line 54 of file [Clouds.h](#).

Referenced by [get_model\(\)](#), and [get_normal_model\(\)](#).

6.5.4.11 movement_direction

```
glm::vec3 Clouds::movement_direction [private]
```

Definition at line 59 of file [Clouds.h](#).

Referenced by [get_movement_direction\(\)](#), and [set_movement_direction\(\)](#).

6.5.4.12 movement_speed

```
GLfloat Clouds::movement_speed [private]
```

Definition at line 62 of file [Clouds.h](#).

Referenced by [get_movement_speed\(\)](#), and [set_movement_speed\(\)](#).

6.5.4.13 noise

```
std::shared_ptr<Noise> Clouds::noise [private]
```

Definition at line 51 of file [Clouds.h](#).

Referenced by [create_noise_textures\(\)](#), and [read\(\)](#).

6.5.4.14 num_march_steps

```
GLuint Clouds::num_march_steps [private]
```

Definition at line 64 of file [Clouds.h](#).

Referenced by [get_num_march_steps\(\)](#), and [set_num_march_steps\(\)](#).

6.5.4.15 num_march_steps_to_light

```
GLuint Clouds::num_march_steps_to_light [private]
```

Definition at line 64 of file [Clouds.h](#).

Referenced by [get_num_march_steps_to_light\(\)](#), and [set_num_march_steps_to_light\(\)](#).

6.5.4.16 pillowness

```
GLfloat Clouds::pillowness [private]
```

Definition at line 62 of file [Clouds.h](#).

Referenced by [get_pillowness\(\)](#), and [set_pillowness\(\)](#).

6.5.4.17 powder_effect

```
bool Clouds::powder_effect [private]
```

Definition at line 66 of file [Clouds.h](#).

Referenced by [get_powder_effect\(\)](#), and [set_powder_effect\(\)](#).

6.5.4.18 random_numbers

```
std::shared_ptr<RandomNumbers> Clouds::random_numbers [private]
```

Definition at line 52 of file [Clouds.h](#).

Referenced by [read\(\)](#).

6.5.4.19 scale

```
GLfloat Clouds::scale [private]
```

Definition at line 62 of file [Clouds.h](#).

Referenced by [get_scale\(\)](#), and [set_scale\(\)](#).

6.5.4.20 scale_factor

```
glm::vec3 Clouds::scale_factor [private]
```

Definition at line 60 of file [Clouds.h](#).

Referenced by [get_mesh_scale\(\)](#), [get_model\(\)](#), [get_radius\(\)](#), and [set_scale\(\)](#).

6.5.4.21 shader_program

```
std::shared_ptr<ShaderProgram> Clouds::shader_program [private]
```

Definition at line 50 of file [Clouds.h](#).

Referenced by [Clouds\(\)](#), [get_shader_program\(\)](#), and [render\(\)](#).

6.5.4.22 translation

```
glm::vec3 Clouds::translation [private]
```

Definition at line 60 of file [Clouds.h](#).

Referenced by [get_model\(\)](#), and [set_translation\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/[Clouds.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/[Clouds.cpp](#)

6.6 ComputeShaderProgram Class Reference

```
#include <ComputeShaderProgram.h>
```

Inheritance diagram for ComputeShaderProgram:

Collaboration diagram for ComputeShaderProgram:

Public Member Functions

- [ComputeShaderProgram \(\)](#)
- [void reload \(\)](#)
- [~ComputeShaderProgram \(\)](#)

Additional Inherited Members

6.6.1 Detailed Description

Definition at line [7](#) of file [ComputeShaderProgram.h](#).

6.6.2 Constructor & Destructor Documentation

6.6.2.1 ComputeShaderProgram()

```
ComputeShaderProgram::ComputeShaderProgram ( )
```

Definition at line [3](#) of file [ComputeShaderProgram.cpp](#).
00003 { }

6.6.2.2 ~ComputeShaderProgram()

```
ComputeShaderProgram::~ComputeShaderProgram ( )
```

Definition at line [7](#) of file [ComputeShaderProgram.cpp](#).
00007 { }

6.6.3 Member Function Documentation

6.6.3.1 reload()

```
void ComputeShaderProgram::reload ( )
```

Definition at line 5 of file [ComputeShaderProgram.cpp](#).

```
00005 { create_computer_shader_program_from_file(compute_location); }
```

References [ShaderProgram::compute_location](#), and [ShaderProgram::create_computer_shader_program_from_file\(\)](#).

Here is the call graph for this function:

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/compute/[ComputeShaderProgram.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/compute/[ComputeShaderProgram.cpp](#)

6.7 DebugApp Class Reference

```
#include <DebugApp.h>
```

Collaboration diagram for DebugApp:

Public Member Functions

- [DebugApp \(\)](#)
- bool [areErrorPrintAll](#) (const std::string &AdditionalArrayMessage="Empty", const char *file=__FILE__, int line=__LINE__)
- bool [arePreError](#) (const std::string &AdditionalArrayMessage="Empty", const char *file=__FILE__, int line=__LINE__)
- [~DebugApp \(\)](#)

6.7.1 Detailed Description

Definition at line 8 of file [DebugApp.h](#).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DebugApp()

```
DebugApp::DebugApp ( )
```

Definition at line 82 of file [DebugApp.cpp](#).

```
00083 {
00084 #ifdef NDEBUG
00085 // nondebug
00086 #else
00087     int flags;
00088     glGetIntegerv(GL_CONTEXT_FLAGS, &flags);
00089     if (flags & GL_CONTEXT_FLAG_DEBUG_BIT) {
00090         // initialize debug output
00091         glEnable(GL_DEBUG_OUTPUT);
00092         glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS);
00093         glDebugMessageCallback(glDebugOutput, nullptr);
00094         glDebugMessageControl(GL_DONT_CARE, GL_DONT_CARE, GL_DONT_CARE, 0, nullptr, GL_TRUE);
00095     }
00096 #endif
00097 }
```

References [glDebugOutput\(\)](#).

Here is the call graph for this function:

6.7.2.2 ~DebugApp()

```
DebugApp::~DebugApp ( )
```

Definition at line 189 of file [DebugApp.cpp](#).

```
00189 { }
```

6.7.3 Member Function Documentation

6.7.3.1 areErrorPrintAll()

```
bool DebugApp::areErrorPrintAll (
    const std::string & AdditionalArrayMessage = "Empty",
    const char * file = __FILE__,
    int line = __LINE__ )
```

Definition at line 99 of file [DebugApp.cpp](#).

```
00100 {
00101 #ifdef NDEBUG
00102 // nondebug
00103     return false;
00104 #else
00105     GLenum err;
00106     bool isError = false;
00107     while ((err = glGetError()) != GL_NO_ERROR) {
00108         std::string errorCode;
00109         switch (err) {
00110             case GL_INVALID_ENUM:
00111                 errorCode = "INVALID_ENUM";
00112                 break;
00113             case GL_INVALID_VALUE:
00114                 errorCode = "INVALID_VALUE";
00115                 break;
00116             case GL_INVALID_OPERATION:
00117                 errorCode = "INVALID_OPERATION";
00118                 break;
00119             case GL_STACK_OVERFLOW:
00120                 errorCode = "STACK_OVERFLOW";
```

```

00121     break;
00122 case GL_STACK_UNDERFLOW:
00123     errorCode = "STACK_UNDERFLOW";
00124     break;
00125 case GL_OUT_OF_MEMORY:
00126     errorCode = "OUT_OF_MEMORY";
00127     break;
00128 case GL_INVALID_FRAMEBUFFER_OPERATION:
00129     errorCode = "INVALID_FRAMEBUFFER_OPERATION";
00130     break;
00131 }
00132 std::cout << "Error, " << errorCode << " | "
00133             << "\nAdditional Error Message: " << AdditionalArrayMessage << " "
00134             << "In File: " << file << ", Line: " << line << std::endl;
00135 isError = true;
00136 }
00137
00138 return isError;
00139 // debug code
00140 #endif
00141 }

```

6.7.3.2 arePreError()

```

bool DebugApp::arePreError (
    const std::string & AdditionalArrayMessage = "Empty",
    const char * file = __FILE__,
    int line = __LINE__ )

```

Definition at line 143 of file [DebugApp.cpp](#).

```

00144 {
00145 #ifdef NDEBUG
00146 // nondebug
00147 return false;
00148 #else
00149 // debug code
00150 GLenum err;
00151 bool isError = false;
00152 while ((err = glGetError()) != GL_NO_ERROR) {
00153     std::string errorCode;
00154     switch (err) {
00155     case GL_INVALID_ENUM:
00156         errorCode = "INVALID_ENUM";
00157         break;
00158     case GL_INVALID_VALUE:
00159         errorCode = "INVALID_VALUE";
00160         break;
00161     case GL_INVALID_OPERATION:
00162         errorCode = "INVALID_OPERATION";
00163         break;
00164     case GL_STACK_OVERFLOW:
00165         errorCode = "STACK_OVERFLOW";
00166         break;
00167     case GL_STACK_UNDERFLOW:
00168         errorCode = "STACK_UNDERFLOW";
00169         break;
00170     case GL_OUT_OF_MEMORY:
00171         errorCode = "OUT_OF_MEMORY";
00172         break;
00173     case GL_INVALID_FRAMEBUFFER_OPERATION:
00174         errorCode = "INVALID_FRAMEBUFFER_OPERATION";
00175         break;
00176 }
00177 std::cout << errorCode << " Error appears before executing the function, "
00178             << " | "
00179             << "\nAdditional Error Message: " << AdditionalArrayMessage << " "
00180             << "In File: " << file << ", Line: " << line << std::endl;
00181 isError = true;
00182 }
00183
00184 return isError;
00185
00186 #endif
00187 }

```

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/debug/[DebugApp.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/debug/[DebugApp.cpp](#)

6.8 DirectionalLight Class Reference

```
#include <DirectionalLight.h>
```

Inheritance diagram for DirectionalLight:

Collaboration diagram for DirectionalLight:

Public Member Functions

- `DirectionalLight ()`
- `DirectionalLight (GLuint shadow_width, GLuint shadow_height, GLfloat red, GLfloat green, GLfloat blue, GLfloat radiance, GLfloat x_dir, GLfloat y_dir, GLfloat z_dir, GLfloat near_plane, GLfloat far_plane, int num_cascades)`
- `glm::mat4 calculate_light_transform ()`
- `std::shared_ptr< CascadedShadowMap > get_shadow_map () const`
- `glm::vec3 get_direction () const`
- `glm::vec3 get_color () const`
- `float get_radiance () const`
- `glm::mat4 get_light_view_matrix () const`
- `std::vector< GLfloat > get_cascaded_slots () const`
- `std::vector< glm::mat4 > & get_cascaded_light_matrices ()`
- `void set_direction (glm::vec3 direction)`
- `void set_radiance (float radiance)`
- `void set_color (glm::vec3 color)`
- `void update_shadow_map (GLfloat shadow_width, GLfloat shadow_height, GLuint num_cascades)`
- `void calc_orthogonal_projections (glm::mat4 camera_view_matrix, GLfloat fov, GLuint window_width, GLuint window_height, GLuint current_num_cascades)`
- `~DirectionalLight ()`

Private Member Functions

- `std::vector< glm::vec4 > get_frustum_corners_world_space (const glm::mat4 &proj, const glm::mat4 &view)`
- `void calc_cascaded_slots ()`

Private Attributes

- `std::shared_ptr< CascadedShadowMap > shadow_map`
- `glm::vec3 direction`
- `GLfloat shadow_near_plane`
- `GLfloat shadow_far_plane`
- `std::array< GLfloat, NUM_CASCADES+1 > cascade_slots`
- `std::vector< glm::mat4 > cascade_light_matrices`

Additional Inherited Members

6.8.1 Detailed Description

Definition at line 10 of file [DirectionalLight.h](#).

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DirectionalLight() [1/2]

DirectionalLight::DirectionalLight ()

Definition at line 3 of file [DirectionalLight.cpp](#).

```
00003   :
00004
00005     Light (), shadow_map(std::make_shared<CascadedShadowMap>()),
00006
00007     direction(glm::vec3{ 0, 0, 0 }),
00008
00009     shadow_near_plane(0.f), shadow_far_plane(0.f),
00010
00011     cascade_light_matrices(NUM_CASCADES, glm::mat4(0.f))
00012
00013 {
00014   light_proj = glm::ortho(-20.0f, 20.0f, -20.0f, 20.0f, 0.1f, 100.f);
00015 }
00016 }
```

References [Light::light_proj](#).

6.8.2.2 DirectionalLight() [2/2]

DirectionalLight::DirectionalLight (
 GLuint shadow_width,
 GLuint shadow_height,
 GLfloat red,
 GLfloat green,
 GLfloat blue,
 GLfloat radiance,
 GLfloat x_dir,
 GLfloat y_dir,
 GLfloat z_dir,
 GLfloat near_plane,
 GLfloat far_plane,
 int num_cascades)

Definition at line 19 of file [DirectionalLight.cpp](#).

```
00020   :
00021
00022     Light(red, green, blue, radiance),
00023     shadow_map(std::make_shared<CascadedShadowMap>()), direction(glm::vec3{ x_dir, y_dir, z_dir }),
00024
00025     shadow_near_plane(near_plane), shadow_far_plane(far_plane),
00026
00027     cascade_light_matrices(NUM_CASCADES, glm::mat4(0.f))
00028
00029 {
00030
00031   light_proj = glm::ortho(-20.0f, 20.0f, -20.0f, 20.0f, 0.1f, 100.f);
00032
00033   shadow_map->init(shadow_width, shadow_height, num_cascades);
00034 }
```

References [Light::light_proj](#), and [shadow_map](#).

6.8.2.3 ~DirectionalLight()

```
DirectionalLight::~DirectionalLight ( )
```

Definition at line 178 of file [DirectionalLight.cpp](#).
00178 { }

6.8.3 Member Function Documentation

6.8.3.1 calc_cascaded_slots()

```
void DirectionalLight::calc_cascaded_slots ( ) [private]
```

Definition at line 81 of file [DirectionalLight.cpp](#).

```
00082 {
00083     GLuint number_of_elements = shadow_map->get_num_active_cascades();
00084
00085     for (int i = 0; i < NUM_CASCADES + 1; i++) {
00086
00087         cascade_slots[i] = 100000.f;
00088     }
00089
00090     for (int i = 0; i < static_cast<int>(number_of_elements + 1); i++) {
00091         if (i == 0) {
00092
00093             (cascade_slots)[i] = shadow_near_plane;
00094         } else {
00095
00096             (cascade_slots)[i] = (shadow_far_plane) * ((GLfloat)i / (GLfloat)(number_of_elements));
00097         }
00098     }
00099 }
00100
00101 //cascade_slots = { shadow_near_plane, shadow_far_plane / 50.f, shadow_far_plane / 25.f,
00102 shadow_far_plane };
00103
00104 }
```

References [cascade_slots](#), [NUM_CASCADES](#), [shadow_far_plane](#), [shadow_map](#), and [shadow_near_plane](#).

Referenced by [calc_orthogonal_projections\(\)](#).

Here is the caller graph for this function:

6.8.3.2 calc_orthogonal_projections()

```
void DirectionalLight::calc_orthogonal_projections (
    glm::mat4 camera_view_matrix,
    GLfloat fov,
    GLuint window_width,
    GLuint window_height,
    GLuint current_num_cascades )
```

Definition at line 106 of file [DirectionalLight.cpp](#).

```
00108 {
00109     //calc the start and end point for our cascaded shadow maps
00110     calc_cascaded_slots();
00111
00112     for (int i = 0; i < static_cast<int>(current_num_cascades); i++) {
```

```

00114     glm::mat4 curr_cascade_proj = glm::perspective(glm::radians(fov), (float)window_width /
00115     (float)window_height, cascade_slots[i], cascade_slots[i + 1]);
00116
00117     std::vector<glm::vec4> frustumCornerWorldSpace =
00118     get_frustum_corners_world_space(curr_cascade_proj, camera_view_matrix);
00119
00120     glm::vec3 center = glm::vec3(0, 0, 0);
00121     for (const auto& v : frustumCornerWorldSpace) {
00122         center += glm::vec3(v);
00123     }
00124
00125     center /= frustumCornerWorldSpace.size();
00126
00127     glm::mat4 light_view_matrix = glm::lookAt(center - get_direction(), center, glm::vec3(0.0f, 1.0f,
0.0f));
00128
00129     // the # of frustum corners = 8
00130     GLfloat minX = std::numeric_limits<float>::max();
00131     GLfloat maxX = std::numeric_limits<float>::min();
00132     GLfloat minY = std::numeric_limits<float>::max();
00133     GLfloat maxY = std::numeric_limits<float>::min();
00134     GLfloat minZ = std::numeric_limits<float>::max();
00135     GLfloat maxZ = std::numeric_limits<float>::min();
00136
00137     for (unsigned int m = 0; m < frustumCornerWorldSpace.size(); m++) {
00138         //transform each corner from view to world space
00139         glm::vec4 v_light_view = light_view_matrix * frustumCornerWorldSpace[m];
00140
00141         //now go to light space
00142         minX = std::min(minX, v_light_view.x);
00143         maxX = std::max(maxX, v_light_view.x);
00144         // we always have negativ y values
00145         minY = std::min(minY, v_light_view.y);
00146         maxY = std::max(maxY, v_light_view.y);
00147
00148         minZ = std::min(minZ, v_light_view.z);
00149         maxZ = std::max(maxZ, v_light_view.z);
00150     }
00151
00152     // Tune this parameter according to the scene
00153     // for having objects casting shadows that are actually not in the frustum :
00154     constexpr float zMult = 10.0f;
00155     if (minZ < 0) {
00156         minZ *= zMult;
00157     } else {
00158         minZ /= zMult;
00159     }
00160     if (maxZ < 0) {
00161         maxZ /= zMult;
00162     } else {
00163         maxZ *= zMult;
00164     }
00165
00166     glm::mat4 light_projection = glm::ortho(minX, maxX, minY, maxY, minZ, maxZ);
00167     cascade_light_matrices[i] = light_projection * light_view_matrix;
00168 }
```

References [calc_cascaded_slots\(\)](#), [cascade_light_matrices](#), [cascade_slots](#), [get_direction\(\)](#), and [get_frustum_corners_world_space\(\)](#).

Here is the call graph for this function:

6.8.3.3 calculate_light_transform()

```
glm::mat4 DirectionalLight::calculate_light_transform ( )
```

Definition at line 170 of file [DirectionalLight.cpp](#).

```
00170 { return light_proj * get_light_view_matrix(); }
```

References [get_light_view_matrix\(\)](#), and [Light::light_proj](#).

Here is the call graph for this function:

6.8.3.4 get_cascaded_light_matrices()

```
std::vector< glm::mat4 > & DirectionalLight::get_cascaded_light_matrices ( )
```

Definition at line 56 of file [DirectionalLight.cpp](#).
00056 { **return** cascade_light_matrices; }

References [cascade_light_matrices](#).

6.8.3.5 get_cascaded_slots()

```
std::vector< GLfloat > DirectionalLight::get_cascaded_slots ( ) const
```

Definition at line 44 of file [DirectionalLight.cpp](#).

```
00045 {  
00046     std::vector<GLfloat> result;  
00047  
00048     for (int i = 0; i < NUM_CASCADES + 1; i++) {  
00049         result.push_back(cascade_slots[i]);  
00050     }  
00051     return result;  
00052 }
```

References [cascade_slots](#), and [NUM_CASCADES](#).

6.8.3.6 get_color()

```
glm::vec3 DirectionalLight::get_color ( ) const
```

Definition at line 40 of file [DirectionalLight.cpp](#).
00040 { **return** color; }

References [Light::color](#).

6.8.3.7 get_direction()

```
glm::vec3 DirectionalLight::get_direction ( ) const
```

Definition at line 38 of file [DirectionalLight.cpp](#).
00038 { **return** direction; }

References [direction](#).

Referenced by [calc_orthogonal_projections\(\)](#).

Here is the caller graph for this function:

6.8.3.8 get_frustum_corners_world_space()

```
std::vector< glm::vec4 > DirectionalLight::get_frustum_corners_world_space (
    const glm::mat4 & proj,
    const glm::mat4 & view ) [private]
```

Definition at line 64 of file [DirectionalLight.cpp](#).

```
00065 {
00066     const auto inv = glm::inverse(proj * view);
00067
00068     std::vector<glm::vec4> frustumCorners;
00069     for (unsigned int x = 0; x < 2; ++x) {
00070         for (unsigned int y = 0; y < 2; ++y) {
00071             for (unsigned int z = 0; z < 2; ++z) {
00072                 const glm::vec4 pt = inv * glm::vec4(2.0f * x - 1.0f, 2.0f * y - 1.0f, 2.0f * z - 1.0f, 1.0f);
00073                 frustumCorners.push_back(pt / pt.w);
00074             }
00075         }
00076     }
00077     return frustumCorners;
00079 }
```

Referenced by [calc_orthogonal_projections\(\)](#).

Here is the caller graph for this function:

6.8.3.9 get_light_view_matrix()

```
glm::mat4 DirectionalLight::get_light_view_matrix ( ) const
```

Definition at line 36 of file [DirectionalLight.cpp](#).

```
00036 { return glm::lookAt(direction, glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f)); }
```

References [direction](#).

Referenced by [calculate_light_transform\(\)](#).

Here is the caller graph for this function:

6.8.3.10 get_radiance()

```
float DirectionalLight::get_radiance ( ) const
```

Definition at line 42 of file [DirectionalLight.cpp](#).

```
00042 { return radiance; }
```

References [Light::radiance](#).

6.8.3.11 get_shadow_map()

```
std::shared_ptr< CascadedShadowMap > DirectionalLight::get_shadow_map ( ) const [inline]
```

Definition at line 19 of file [DirectionalLight.h](#).

```
00019 { return shadow_map; }
```

References [shadow_map](#).

6.8.3.12 set_color()

```
void DirectionalLight::set_color (
    glm::vec3 color )
```

Definition at line 176 of file [DirectionalLight.cpp](#).
00176 { this->color = color; }

References [Light::color](#).

6.8.3.13 set_direction()

```
void DirectionalLight::set_direction (
    glm::vec3 direction )
```

Definition at line 172 of file [DirectionalLight.cpp](#).
00172 { this->direction = direction; }

References [direction](#).

6.8.3.14 set_radiance()

```
void DirectionalLight::set_radiance (
    float radiance )
```

Definition at line 174 of file [DirectionalLight.cpp](#).
00174 { this->radiance = radiance; }

References [Light::radiance](#).

6.8.3.15 update_shadow_map()

```
void DirectionalLight::update_shadow_map (
    GLfloat shadow_width,
    GLfloat shadow_height,
    GLuint num_cascades )
```

Definition at line 58 of file [DirectionalLight.cpp](#).
00059 {
00060 shadow_map.reset(new CascadedShadowMap);
00061 shadow_map->init((GLuint)shadow_width, (GLuint)shadow_height, num_cascades);
00062 }

References [shadow_map](#).

6.8.4 Field Documentation

6.8.4.1 cascade_light_matrices

```
std::vector<glm::mat4> DirectionalLight::cascade_light_matrices [private]
```

Definition at line 49 of file [DirectionalLight.h](#).

Referenced by [calc_orthogonal_projections\(\)](#), and [get_cascaded_light_matrices\(\)](#).

6.8.4.2 cascade_slots

```
std::array<GLfloat, NUM_CASCADES + 1> DirectionalLight::cascade_slots [private]
```

Definition at line 48 of file [DirectionalLight.h](#).

Referenced by [calc_cascaded_slots\(\)](#), [calc_orthogonal_projections\(\)](#), and [get_cascaded_slots\(\)](#).

6.8.4.3 direction

```
glm::vec3 DirectionalLight::direction [private]
```

Definition at line 45 of file [DirectionalLight.h](#).

Referenced by [get_direction\(\)](#), [get_light_view_matrix\(\)](#), and [set_direction\(\)](#).

6.8.4.4 shadow_far_plane

```
GLfloat DirectionalLight::shadow_far_plane [private]
```

Definition at line 46 of file [DirectionalLight.h](#).

Referenced by [calc_cascaded_slots\(\)](#).

6.8.4.5 shadow_map

```
std::shared_ptr<CascadedShadowMap> DirectionalLight::shadow_map [private]
```

Definition at line 43 of file [DirectionalLight.h](#).

Referenced by [calc_cascaded_slots\(\)](#), [DirectionalLight\(\)](#), [get_shadow_map\(\)](#), and [update_shadow_map\(\)](#).

6.8.4.6 shadow_near_plane

```
GLfloat DirectionalLight::shadow_near_plane [private]
```

Definition at line 46 of file [DirectionalLight.h](#).

Referenced by [calc_cascaded_slots\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[DirectionalLight.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[DirectionalLight.cpp](#)

6.9 DirectionalShadowMapPass Class Reference

```
#include <DirectionalShadowMapPass.h>
```

Inheritance diagram for DirectionalShadowMapPass:

Collaboration diagram for DirectionalShadowMapPass:

Public Member Functions

- [DirectionalShadowMapPass \(\)](#)
- void [execute \(glm::mat4 projection, std::shared_ptr< Camera > main_camera, GLuint window_width, GLuint window_height, std::shared_ptr< Scene > scene\)](#)
- void [create_shader_program \(\)](#)
- void [set_game_object_uniforms \(glm::mat4 model, glm::mat4 normal_model\)](#)
- [~DirectionalShadowMapPass \(\)](#)

Private Attributes

- std::shared_ptr< [ShaderProgram](#) > [shader_program](#)

6.9.1 Detailed Description

Definition at line 9 of file [DirectionalShadowMapPass.h](#).

6.9.2 Constructor & Destructor Documentation

6.9.2.1 DirectionalShadowMapPass()

```
DirectionalShadowMapPass::DirectionalShadowMapPass ( )
```

Definition at line 4 of file [DirectionalShadowMapPass.cpp](#).
00004 { [create_shader_program\(\)](#); }

References [create_shader_program\(\)](#).

Here is the call graph for this function:

6.9.2.2 ~DirectionalShadowMapPass()

```
DirectionalShadowMapPass::~DirectionalShadowMapPass ( )
```

Definition at line 61 of file [DirectionalShadowMapPass.cpp](#).
00061 { }

6.9.3 Member Function Documentation

6.9.3.1 create_shader_program()

```
void DirectionalShadowMapPass::create_shader_program ( ) [virtual]
```

Implements [RenderPassSceneDependend](#).

Definition at line 48 of file [DirectionalShadowMapPass.cpp](#).

```
00049 {
00050     shader\_program = std::make_shared<ShaderProgram>(ShaderProgram{});
00051     shader\_program->create_from_files(
00052         "rasterizer/shadows/directional_shadow_map.vert",
00053         "rasterizer/shadows/directional_shadow_map.geom", "rasterizer/shadows/directional_shadow_map.frag");
00053 }
```

References [shader_program](#).

Referenced by [DirectionalShadowMapPass\(\)](#).

Here is the caller graph for this function:

6.9.3.2 execute()

```
void DirectionalShadowMapPass::execute (
    glm::mat4 projection,
    std::shared_ptr< Camera > main_camera,
    GLuint window_width,
    GLuint window_height,
    std::shared_ptr< Scene > scene )
```

Definition at line 6 of file [DirectionalShadowMapPass.cpp](#).

```
00008 {
00009
00010     std::shared_ptr<DirectionalLight> sun = scene->get_sun();
00011     //retreive shadow map before our geometry pass
00012     sun->calc_orthogonal_projections(main_camera->get_viewmatrix(), main_camera->get_fov(),
00013     window_width, window_height, NUM_CASCADES);
00014     shader_program->use_shader_program();
00015
00016     sun->get_shadow_map()->write();
00017
00018     glViewport(0, 0, sun->get_shadow_map()->get_shadow_width(),
00019     sun->get_shadow_map()->get_shadow_height());
00020     glClear(GL_DEPTH_BUFFER_BIT);
00021
00022     glEnable(GL_CULL_FACE);
00023     glCullFace(GL_BACK);
00024     glFrontFace(GL_CCW);
00025
00026     //glCullFace(GL_FRONT); // avoid peter panning
00027     sun->get_shadow_map()->write_light_matrices(sun->get_cascaded_light_matrices());
00028
00029     shader_program->setUniformBlockBinding(UNIFORM_LIGHT_MATRICES_BINDING, "LightSpaceMatrices");
00030
00031     std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00032
00033     for (std::shared_ptr<GameObject> object : game_objects) {
00034
00035         /* if (object_is_visible(object)) */
00036
00037         set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00038
00039         object->render();
00040         //}
00041     }
00042
00043     //glCullFace(GL_BACK);
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045     shader_program->validate_program();
00046 }
```

References [NUM_CASCADES](#), [set_game_object_uniforms\(\)](#), [shader_program](#), and [UNIFORM_LIGHT_MATRICES_BINDING](#).

Here is the call graph for this function:

6.9.3.3 set_game_object_uniforms()

```
void DirectionalShadowMapPass::set_game_object_uniforms (
    glm::mat4 model,
    glm::mat4 normal_model ) [virtual]
```

Implements [RenderPassSceneDependend](#).

Definition at line 55 of file [DirectionalShadowMapPass.cpp](#).

```
00056 {
00057     // DO NOT set neither normal model nor material_id hence we didn't need it
00058     shader_program->setUniformMatrix4fv(model, "model");
00059 }
```

References [shader_program](#).

Referenced by [execute\(\)](#).

Here is the caller graph for this function:

6.9.4 Field Documentation

6.9.4.1 shader_program

```
std::shared_ptr<ShaderProgram> DirectionalShadowMapPass::shader_program [private]
```

Definition at line 22 of file [DirectionalShadowMapPass.h](#).

Referenced by [create_shader_program\(\)](#), [execute\(\)](#), and [set_game_object_uniforms\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[DirectionalShadowMapPass.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/[DirectionalShadowMapPass.cpp](#)

6.10 File Class Reference

```
#include <File.h>
```

Collaboration diagram for File:

Public Member Functions

- [File](#) (const std::string &[file_location](#))
- std::string [read](#) ()
- [~File](#) ()

Private Attributes

- std::string [file_location](#)

6.10.1 Detailed Description

Definition at line 4 of file [File.h](#).

6.10.2 Constructor & Destructor Documentation

6.10.2.1 File()

```
File::File (
    const std::string & file_location ) [explicit]
```

Definition at line 5 of file [File.cpp](#).

```
00005 { this->file_location = file_location; }
```

References [file_location](#).

6.10.2.2 ~File()

```
File::~File ( )
```

Definition at line 27 of file [File.cpp](#).

```
00027 { }
```

6.10.3 Member Function Documentation

6.10.3.1 read()

```
std::string File::read ( )
```

Definition at line 7 of file [File.cpp](#).

```
00008 {
00009     std::string content;
00010     std::ifstream file_stream(file_location, std::ios::in);
00011
00012     if (!file_stream.is_open()) {
00013         printf("Failed to read %. File does not exist.", file_location.c_str());
00014         return "";
00015     }
00016
00017     std::string line = "";
00018     while (!file_stream.eof()) {
00019         std::getline(file_stream, line);
00020         content.append(line + "\n");
00021     }
00022
00023     file_stream.close();
00024     return content;
00025 }
```

References [file_location](#).

Referenced by [ShaderProgram::create_computer_shader_program_from_file\(\)](#), [ShaderProgram::create_from_files\(\)](#), and [ShaderIncludes::ShaderIncludes\(\)](#).

Here is the caller graph for this function:

6.10.4 Field Documentation

6.10.4.1 file_location

```
std::string File::file_location [private]
```

Definition at line 14 of file [File.h](#).

Referenced by [File\(\)](#), and [read\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/util/[File.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/util/[File.cpp](#)

6.11 ViewFrustumCulling::frustum_plane Struct Reference

Collaboration diagram for ViewFrustumCulling::frustum_plane:

Data Fields

- `glm::vec3 normal`
- `glm::vec3 position`

6.11.1 Detailed Description

Definition at line 49 of file [ViewFrustumCulling.h](#).

6.11.2 Field Documentation

6.11.2.1 normal

```
glm::vec3 ViewFrustumCulling::frustum_plane::normal
```

Definition at line 51 of file [ViewFrustumCulling.h](#).

Referenced by [ViewFrustumCulling::plane_point_distance\(\)](#), and [ViewFrustumCulling::update_frustum_param\(\)](#).

6.11.2.2 position

```
glm::vec3 ViewFrustumCulling::frustum_plane::position
```

Definition at line 52 of file [ViewFrustumCulling.h](#).

Referenced by [ViewFrustumCulling::plane_point_distance\(\)](#), and [ViewFrustumCulling::update_frustum_param\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ViewFrustumCulling.h](#)

6.12 GameObject Class Reference

```
#include <GameObject.h>
```

Collaboration diagram for GameObject:

Public Member Functions

- [GameObject \(\)](#)
- [GameObject \(const std::string &model_path, glm::vec3 translation, GLfloat scale, Rotation rot\)](#)
- void [init \(const std::string &model_path, glm::vec3 translation, GLfloat scale, Rotation rot\)](#)
- glm::mat4 [get_world_trafo \(\)](#)
- glm::mat4 [get_normal_world_trafo \(\)](#)
- std::shared_ptr< [AABB](#) > [get_aabb \(\)](#)
- std::shared_ptr< [Model](#) > [get_model \(\)](#)
- void [translate \(glm::vec3 translate\)](#)
- void [scale \(GLfloat scale_factor\)](#)
- void [rotate \(Rotation rot\)](#)
- void [render \(\)](#)
- [~GameObject \(\)](#)

Private Attributes

- std::shared_ptr< [Model](#) > [model](#)
- GLfloat [scale_factor](#)
- [Rotation rot](#)
- glm::vec3 [translation](#)

6.12.1 Detailed Description

Definition at line 6 of file [GameObject.h](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 GameObject() [1/2]

```
GameObject::GameObject ( )
```

Definition at line 3 of file [GameObject.cpp](#).
 00003 : `model(std::make_shared<Model>(Model())) { }`

6.12.2.2 GameObject() [2/2]

```
GameObject::GameObject (
    const std::string & model_path,
    glm::vec3 translation,
    GLfloat scale,
    Rotation rot )
```

Definition at line 5 of file [GameObject.cpp](#).

```
00005   model(std::make_shared<Model>())
00006 { model->load_model_in_ram(model_path);
00007   this->translation = translation;
00008   this->scale_factor = scale;
00009   this->rot = rot;
00010 }
00011 }
```

References [model](#), [rot](#), [scale\(\)](#), [scale_factor](#), and [translation](#).

Here is the call graph for this function:

6.12.2.3 ~GameObject()

```
GameObject::~GameObject ( )
```

Definition at line 50 of file [GameObject.cpp](#).
 00050 { }

6.12.3 Member Function Documentation

6.12.3.1 get_aabb()

```
std::shared_ptr< AABB > GameObject::get_aabb ( )
```

Definition at line 40 of file [GameObject.cpp](#).
 00040 { `return model->get_aabb();` }

References [model](#).

6.12.3.2 get_model()

```
std::shared_ptr< Model > GameObject::get_model ( )
```

Definition at line 42 of file [GameObject.cpp](#).

```
00042 { return model; }
```

References [model](#).

6.12.3.3 get_normal_world_trafo()

```
glm::mat4 GameObject::get_normal_world_trafo ( )
```

Definition at line 32 of file [GameObject.cpp](#).

```
00033 {
00034     glm::mat4 world_trafo = get_world_trafo();
00035     return glm::transpose(glm::inverse(world_trafo));
00036 }
```

References [get_world_trafo\(\)](#).

Here is the call graph for this function:

6.12.3.4 get_world_trafo()

```
glm::mat4 GameObject::get_world_trafo ( )
```

Definition at line 22 of file [GameObject.cpp](#).

```
00023 {
00024     glm::mat4 model_to_world = glm::mat4(1.0);
00025     model_to_world = glm::translate(model_to_world, translation);
00026     model_to_world = glm::scale(model_to_world, glm::vec3(scale_factor));
00027     model_to_world = glm::rotate(model_to_world, glm::radians(rot.degrees), rot.axis);
00028
00029     return model_to_world;
00030 }
```

References [Rotation::axis](#), [Rotation::degrees](#), [rot](#), [scale_factor](#), and [translation](#).

Referenced by [get_normal_world_trafo\(\)](#).

Here is the caller graph for this function:

6.12.3.5 init()

```
void GameObject::init (
    const std::string & model_path,
    glm::vec3 translation,
    GLfloat scale,
    Rotation rot )
```

Definition at line 13 of file [GameObject.cpp](#).

```
00014 {
00015     model = std::make_shared<Model> (Model ());
00016     model->load_model_in_ram(model_path);
00017     this->translation = translation;
00018     this->scale_factor = scale;
00019     this->rot = rot;
00020 }
```

References [model](#), [rot](#), [scale\(\)](#), [scale_factor](#), and [translation](#).

Here is the call graph for this function:

6.12.3.6 render()

```
void GameObject::render ( )
```

Definition at line 38 of file [GameObject.cpp](#).
00038 { model->render(); }

References [model](#).

6.12.3.7 rotate()

```
void GameObject::rotate (  
    Rotation rot )
```

Definition at line 46 of file [GameObject.cpp](#).
00046 { this->rot = rot; }

References [rot](#).

6.12.3.8 scale()

```
void GameObject::scale (  
    GLfloat scale_factor )
```

Definition at line 48 of file [GameObject.cpp](#).
00048 { this->scale_factor = scale_factor; }

References [scale_factor](#).

Referenced by [GameObject\(\)](#), and [init\(\)](#).

Here is the caller graph for this function:

6.12.3.9 translate()

```
void GameObject::translate (   
    glm::vec3 translate )
```

Definition at line 44 of file [GameObject.cpp](#).
00044 { this->translation = translate; }

References [translate\(\)](#), and [translation](#).

Referenced by [translate\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.12.4 Field Documentation

6.12.4.1 model

```
std::shared_ptr<Model> GameObject::model [private]
```

Definition at line 30 of file [GameObject.h](#).

Referenced by [GameObject\(\)](#), [get_aabb\(\)](#), [get_model\(\)](#), [init\(\)](#), and [render\(\)](#).

6.12.4.2 rot

```
Rotation GameObject::rot [private]
```

Definition at line 33 of file [GameObject.h](#).

Referenced by [GameObject\(\)](#), [get_world_trafo\(\)](#), [init\(\)](#), and [rotate\(\)](#).

6.12.4.3 scale_factor

```
GLfloat GameObject::scale_factor [private]
```

Definition at line 32 of file [GameObject.h](#).

Referenced by [GameObject\(\)](#), [get_world_trafo\(\)](#), [init\(\)](#), and [scale\(\)](#).

6.12.4.4 translation

```
glm::vec3 GameObject::translation [private]
```

Definition at line 34 of file [GameObject.h](#).

Referenced by [GameObject\(\)](#), [get_world_trafo\(\)](#), [init\(\)](#), and [translate\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[GameObject.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[GameObject.cpp](#)

6.13 GBuffer Class Reference

```
#include <GBuffer.h>
```

Collaboration diagram for GBuffer:

Public Member Functions

- [GBuffer \(\)](#)
- [GBuffer \(GLint window_width, GLint window_height\)](#)
- [void create \(\)](#)
- [void read \(std::shared_ptr< ShaderProgram > shader_program\)](#)
- [void update_window_params \(GLuint window_width, GLuint window_height\)](#)
- [GLuint get_id \(\) const](#)
- [~GBuffer \(\)](#)

Private Attributes

- [GLuint g_buffer](#)
- [GLuint g_position](#)
- [GLuint g_normal](#)
- [GLuint g_albedo](#)
- [GLuint g_material_id](#)
- [GLuint g_depth](#)
- [GLuint g_buffer_attachment \[G_BUFFER_SIZE\] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1, GL_COLOR_ATTACHMENT2, GL_COLOR_ATTACHMENT3 }](#)
- [GLuint window_width](#)
- [GLuint window_height](#)

6.13.1 Detailed Description

Definition at line 11 of file [GBuffer.h](#).

6.13.2 Constructor & Destructor Documentation

6.13.2.1 GBuffer() [1/2]

```
GBuffer::GBuffer ( )
```

Definition at line 4 of file [GBuffer.cpp](#).

```
00005 {
00006     this->window_width = 1024;
00007     this->window_height = 768;
00008 }
```

References [window_height](#), and [window_width](#).

6.13.2.2 GBuffer() [2/2]

```
GBuffer::GBuffer (
    GLint window_width,
    GLint window_height )
```

Definition at line 10 of file [GBuffer.cpp](#).

```
00011 {
00012
00013     this->window_width = window_width;
00014     this->window_height = window_height;
00015 }
```

References [window_height](#), and [window_width](#).

6.13.2.3 ~GBuffer()

```
GBuffer::~GBuffer ( )
```

Definition at line 97 of file [GBuffer.cpp](#).

```
00098 {
00099
00100     glDeleteFramebuffers(1, &g_buffer);
00101     glDeleteTextures(1, &g_position);
00102     glDeleteTextures(1, &g_normal);
00103     glDeleteTextures(1, &g_albedo);
00104     glDeleteTextures(1, &g_material_id);
00105 }
```

References [g_albedo](#), [g_buffer](#), [g_material_id](#), [g_normal](#), and [g_position](#).

6.13.3 Member Function Documentation

6.13.3.1 create()

```
void GBuffer::create ( )
```

Definition at line 17 of file [GBuffer.cpp](#).

```
00018 {
00019
00020     glGenFramebuffers(1, &g_buffer);
00021     glBindFramebuffer(GL_FRAMEBUFFER, g_buffer);
00022
00023     glGenTextures(1, &g_position);
00024     glBindTexture(GL_TEXTURE_2D, g_position);
00025     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_FLOAT, NULL);
00026     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00027     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00028     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[0], GL_TEXTURE_2D, g_position, 0);
00029
00030     glGenTextures(1, &g_normal);
00031     glBindTexture(GL_TEXTURE_2D, g_normal);
00032     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_FLOAT, NULL);
00033     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00034     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00035     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[1], GL_TEXTURE_2D, g_normal, 0);
00036
00037     glGenTextures(1, &g_albedo);
00038     glBindTexture(GL_TEXTURE_2D, g_albedo);
00039     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, NULL);
```

```

00040     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00041     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00042     glBindFramebuffer(GL_FRAMEBUFFER, g_buffer_attachment[2], GL_TEXTURE_2D, g_albedo, 0);
00043
00044     glGenTextures(1, &g_material_id);
00045     glBindTexture(GL_TEXTURE_2D, g_material_id);
00046     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_UNSIGNED_INT,
00047                 NULL);
00047     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00048     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00049     glBindFramebuffer(GL_FRAMEBUFFER, g_buffer_attachment[3], GL_TEXTURE_2D, g_material_id, 0);
00050
00051     glDrawBuffers(G_BUFFER_SIZE, g_buffer_attachment);
00052
00053     // create and attach depth buffer (renderbuffer)
00054     // renderbuffers are a bit more performant
00055     glGenRenderbuffers(1, &g_depth);
00056     glBindRenderbuffer(GL_RENDERBUFFER, g_depth);
00057     glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT32F, window_width, window_height);
00058     glBindFramebuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, g_depth);
00059
00060     if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE) {
00061         printf("Framebuffer not complete!");
00062     }
00063
00064     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00065 }
```

References [g_albedo](#), [g_buffer](#), [g_buffer_attachment](#), [G_BUFFER_SIZE](#), [g_depth](#), [g_material_id](#), [g_normal](#), [g_position](#), [window_height](#), and [window_width](#).

6.13.3.2 get_id()

```
GLuint GBUFFER::get_id () const [inline]
```

Definition at line 21 of file [GBUFFER.h](#).

```
00021 { return g_buffer; };
```

References [g_buffer](#).

6.13.3.3 read()

```
void GBUFFER::read (
    std::shared_ptr< ShaderProgram > shader_program )
```

Definition at line 67 of file [GBUFFER.cpp](#).

```

00068 {
00069     // GBUFFER
00070     GLuint texture_index = GBUFFER_TEXTURES_SLOT;
00071     shader_program->setUniformInt(texture_index, "g_position");
00072     glEnableTexture(GL_TEXTURE0 + texture_index);
00073     glBindTexture(GL_TEXTURE_2D, g_position);
00074
00075     texture_index++;
00076     shader_program->setUniformInt(texture_index, "g_normal");
00077     glEnableTexture(GL_TEXTURE0 + texture_index);
00078     glBindTexture(GL_TEXTURE_2D, g_normal);
00079
00080     texture_index++;
00081     shader_program->setUniformInt(texture_index, "g_albedo");
00082     glEnableTexture(GL_TEXTURE0 + texture_index);
00083     glBindTexture(GL_TEXTURE_2D, g_albedo);
00084
00085     texture_index++;
00086     shader_program->setUniformInt(texture_index, "g_material_id");
00087     glEnableTexture(GL_TEXTURE0 + texture_index);
00088     glBindTexture(GL_TEXTURE_2D, g_material_id);
00089 }
```

References [g_albedo](#), [g_material_id](#), [g_normal](#), [g_position](#), and [GBUFFER_TEXTURES_SLOT](#).

6.13.3.4 update_window_params()

```
void GBuffer::update_window_params (
    GLuint window_width,
    GLuint window_height )
```

Definition at line 91 of file [GBuffer.cpp](#).

```
00092 {
00093     this->window_width = window_width;
00094     this->window_height = window_height;
00095 }
```

References [window_height](#), and [window_width](#).

6.13.4 Field Documentation

6.13.4.1 g_albedo

```
GLuint GBuffer::g_albedo [private]
```

Definition at line 28 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [read\(\)](#), and [~GBuffer\(\)](#).

6.13.4.2 g_buffer

```
GLuint GBuffer::g_buffer [private]
```

Definition at line 26 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [get_id\(\)](#), and [~GBuffer\(\)](#).

6.13.4.3 g_buffer_attachment

```
GLuint GBuffer::g_buffer_attachment[G_BUFFER_SIZE] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1,
GL_COLOR_ATTACHMENT2, GL_COLOR_ATTACHMENT3 } [private]
```

Definition at line 30 of file [GBuffer.h](#).

Referenced by [create\(\)](#).

6.13.4.4 g_depth

```
GLuint GBuffer::g_depth [private]
```

Definition at line 28 of file [GBuffer.h](#).

Referenced by [create\(\)](#).

6.13.4.5 g_material_id

```
GLuint GBuffer::g_material_id [private]
```

Definition at line 28 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [read\(\)](#), and [~GBuffer\(\)](#).

6.13.4.6 g_normal

```
GLuint GBuffer::g_normal [private]
```

Definition at line 28 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [read\(\)](#), and [~GBuffer\(\)](#).

6.13.4.7 g_position

```
GLuint GBuffer::g_position [private]
```

Definition at line 28 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [read\(\)](#), and [~GBuffer\(\)](#).

6.13.4.8 window_height

```
GLuint GBuffer::window_height [private]
```

Definition at line 32 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [GBuffer\(\)](#), and [update_window_params\(\)](#).

6.13.4.9 window_width

```
GLuint GBuffer::window_width [private]
```

Definition at line 32 of file [GBuffer.h](#).

Referenced by [create\(\)](#), [GBuffer\(\)](#), and [update_window_params\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GBuffer.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GBuffer.cpp](#)

6.14 GeometryPass Class Reference

```
#include <GeometryPass.h>
```

Inheritance diagram for GeometryPass:

Collaboration diagram for GeometryPass:

Public Member Functions

- [GeometryPass \(\)](#)
- void [execute](#) (glm::mat4 projection_matrix, std::shared_ptr<[Camera](#)> main_camera, GLuint window_width, GLuint window_height, GLuint gbuffer_id, GLfloat delta_time, std::shared_ptr<[Scene](#)>)
- void [create_shader_program \(\)](#)
- void [set_game_object_uniforms](#) (glm::mat4 model, glm::mat4 normal_model)
- [~GeometryPass \(\)](#)

Private Attributes

- std::shared_ptr<[GeometryPassShaderProgram](#)> [shader_program](#)
- [SkyBox skybox](#)

6.14.1 Detailed Description

Definition at line 14 of file [GeometryPass.h](#).

6.14.2 Constructor & Destructor Documentation

6.14.2.1 GeometryPass()

```
GeometryPass::GeometryPass ( )
```

Definition at line 3 of file [GeometryPass.cpp](#).
00003 : skybox() { create_shader_program(); }

References [create_shader_program\(\)](#).

Here is the call graph for this function:

6.14.2.2 ~GeometryPass()

```
GeometryPass::~GeometryPass ( )
```

Definition at line 134 of file [GeometryPass.cpp](#).
00134 { }

6.14.3 Member Function Documentation

6.14.3.1 create_shader_program()

```
void GeometryPass::create_shader_program ( ) [virtual]
```

Implements [RenderPassSceneDependend](#).

Definition at line 121 of file [GeometryPass.cpp](#).

```
00122 {  
00123     this->shader_program = std::make_shared<GeometryPassShaderProgram>(<GeometryPassShaderProgram>());  
00124     this->shader_program->create_from_files("rasterizer/g_buffer_geometry_pass.vert",  
00125         "rasterizer/g_buffer_geometry_pass.frag");  
00126 }
```

References [shader_program](#).

Referenced by [GeometryPass\(\)](#).

Here is the caller graph for this function:

6.14.3.2 execute()

```
void GeometryPass::execute (
    glm::mat4 projection_matrix,
    std::shared_ptr< Camera > main_camera,
    GLuint window_width,
    GLuint window_height,
    GLuint gbuffer_id,
    GLfloat delta_time,
    std::shared_ptr< Scene > scene )
```

Definition at line 5 of file [GeometryPass.cpp](#).

```
00007 {
00008
00009     glBindFramebuffer(GL_FRAMEBUFFER, gbuffer_id);
00010
00011     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00012     glViewport(0, 0, window_width, window_height);
00013
00014     glEnable(GL_CULL_FACE);
00015     glCullFace(GL_BACK);
00016     glFrontFace(GL_CCW);
00017
00018     shader_program->use_shader_program();
00019
00020     glm::mat4 view_matrix = main_camera->get_viewmatrix();
00021     std::vector<ObjMaterial> materials = scene->get_materials();
00022
00023     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00024     shader_program->setUniformMatrix4fv(view_matrix, "view");
00025
00026     std::stringstream ss;
00027     for (uint32_t i = 0; i < static_cast<uint32_t>(scene->get_texture_count(0)); i++) {
00028
00029         ss << "model_textures[" << i << "]";
00030         shader_program->setUniformInt(MODEL_TEXTURES_SLOT + i, ss.str());
00031         ss.clear();
00032         ss.str(std::string());
00033     }
00034
00035     for (uint32_t i = 0; i < static_cast<uint32_t>(materials.size()); i++) {
00036
00037         ss << "materials[" << i << "].ambient";
00038         shader_program->setUniformVec3(materials[i].get_ambient(), ss.str());
00039         ss.clear();
00040         ss.str(std::string());
00041
00042         ss << "materials[" << i << "].diffuse";
00043         shader_program->setUniformVec3(materials[i].get_diffuse(), ss.str());
00044         ss.clear();
00045         ss.str(std::string());
00046
00047         ss << "materials[" << i << "].specular";
00048         shader_program->setUniformVec3(materials[i].get_specular(), ss.str());
00049         ss.clear();
00050         ss.str(std::string());
00051
00052         ss << "materials[" << i << "].transmittance";
00053         shader_program->setUniformVec3(materials[i].get_transmittance(), ss.str());
00054         ss.clear();
00055         ss.str(std::string());
00056
00057         ss << "materials[" << i << "].emission";
00058         shader_program->setUniformVec3(materials[i].get_emission(), ss.str());
00059         ss.clear();
00060         ss.str(std::string());
00061
00062         ss << "materials[" << i << "].shininess";
00063         shader_program->setUniformFloat(materials[i].get_shininess(), ss.str());
00064         ss.clear();
00065         ss.str(std::string());
00066
00067         ss << "materials[" << i << "].ior";
00068         shader_program->setUniformFloat(materials[i].get_ior(), ss.str());
00069         ss.clear();
00070         ss.str(std::string());
00071
00072         ss << "materials[" << i << "].dissolve";
00073         shader_program->setUniformFloat(materials[i].get_dissolve(), ss.str());
00074         ss.clear();
00075         ss.str(std::string());
```

```

00076
00077     ss << "materials[" << i << "].illum";
00078     shader_program->setUniformInt(materials[i].get_illum(), ss.str());
00079     ss.clear();
00080     ss.str(std::string());
00081
00082     ss << "materials[" << i << "].textureID";
00083     shader_program->setUniformInt(materials[i].get_textureID(), ss.str());
00084     ss.clear();
00085     ss.str(std::string());
00086 }
00087
00088 shader_program->validate_program();
00089
00090 scene->bind_textures_and_buffer();
00091
00092 //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
00093 //aabb->render();
00094 //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
00095 //
00096
00097 std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00098
00099 for (std::shared_ptr<GameObject> object : game_objects) {
00100
00101 /* if (object_is_visible(object)) {*/
00102
00103     set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00104
00105     object->render();
00106 //}
00107 }
00108
00109 skybox.draw_sky_box(projection_matrix, view_matrix, window_width, window_height, delta_time);
00110
00111 /*glCullFace(GL_FRONT);
00112 glFrontFace(GL_CCW);*/
00113 // render the AABB for the clouds
00114 glBindable(GL_CULL_FACE);
00115 std::shared_ptr<Clouds> clouds = scene->get_clouds();
00116 clouds->render(projection_matrix, view_matrix, window_width, window_height);
00117
00118 glBindFramebuffer(GL_FRAMEBUFFER, 0);
00119 }

```

References [SkyBox::draw_sky_box\(\)](#), [MODEL_TEXTURES_SLOT](#), [set_game_object_uniforms\(\)](#), [shader_program](#), and [skybox](#).

Here is the call graph for this function:

6.14.3.3 set_game_object_uniforms()

```

void GeometryPass::set_game_object_uniforms (
    glm::mat4 model,
    glm::mat4 normal_model ) [virtual]

```

Implements [RenderPassSceneDependend](#).

Definition at line 127 of file [GeometryPass.cpp](#).

```

00128 {
00129
00130     shader_program->setUniformMatrix4fv(model, "model");
00131     shader_program->setUniformMatrix4fv(normal_model, "normal_model");
00132 }

```

References [shader_program](#).

Referenced by [execute\(\)](#).

Here is the caller graph for this function:

6.14.4 Field Documentation

6.14.4.1 shader_program

```
std::shared_ptr<GeometryPassShaderProgram> GeometryPass::shader_program [private]
```

Definition at line 28 of file [GeometryPass.h](#).

Referenced by [create_shader_program\(\)](#), [execute\(\)](#), and [set_game_object_uniforms\(\)](#).

6.14.4.2 skybox

```
SkyBox GeometryPass::skybox [private]
```

Definition at line 30 of file [GeometryPass.h](#).

Referenced by [execute\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GeometryPass.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GeometryPass.cpp](#)

6.15 GeometryPassShaderProgram Class Reference

```
#include <GeometryPassShaderProgram.h>
```

Inheritance diagram for GeometryPassShaderProgram:

Collaboration diagram for GeometryPassShaderProgram:

Public Member Functions

- [GeometryPassShaderProgram \(\)](#)
- [GLuint get_program_id \(\)](#)
- [~GeometryPassShaderProgram \(\)](#)

Additional Inherited Members

6.15.1 Detailed Description

Definition at line 5 of file [GeometryPassShaderProgram.h](#).

6.15.2 Constructor & Destructor Documentation

6.15.2.1 GeometryPassShaderProgram()

```
GeometryPassShaderProgram::GeometryPassShaderProgram ( )
```

Definition at line 3 of file [GeometryPassShaderProgram.cpp](#).
00003 { }

6.15.2.2 ~GeometryPassShaderProgram()

```
GeometryPassShaderProgram::~GeometryPassShaderProgram ( )
```

Definition at line 5 of file [GeometryPassShaderProgram.cpp](#).
00005 { }

6.15.3 Member Function Documentation

6.15.3.1 get_program_id()

```
GLuint GeometryPassShaderProgram::get_program_id ( ) [inline]
```

Definition at line 10 of file [GeometryPassShaderProgram.h](#).
00010 { return program_id; }

References [ShaderProgram::program_id](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GeometryPassShaderProgram.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[GeometryPassShaderProgram.cpp](#)

6.16 GUI Class Reference

```
#include <GUI.h>
```

Collaboration diagram for GUI:

Public Member Functions

- `GUI ()`
- void `init (std::shared_ptr< Window > main_window)`
- void `render (bool loading_in_progress, float progress, bool & shader_hot_reload_triggered)`
- void `update_user_input (std::shared_ptr< Scene > scene)`
- `~GUI ()`

Private Attributes

- Texture `logo_tex`
- float `direcional_light_radiance`
- float `directional_light_color [3]`
- float `directional_light_direction [3]`
- int `cloud_speed`
- float `cloud_scale`
- float `cloud_density`
- float `cloud_pillowness`
- float `cloud_cirrus_effect`
- float `cloud_mesh_scale [3]`
- float `cloud_mesh_offset [3]`
- bool `cloud_powder_effect`
- float `cloud_movement_direction [3]`
- int `cloud_num_march_steps`
- int `cloud_num_march_steps_to_light`
- int `shadow_map_res_index`
- bool `shadow_resolution_changed`
- int `num_shadow_cascades`
- int `pcf_radius`
- float `cascaded_shadow_intensity`
- const char * `available_shadow_map_resolutions [4]`

6.16.1 Detailed Description

Definition at line 14 of file [GUI.h](#).

6.16.2 Constructor & Destructor Documentation

6.16.2.1 GUI()

```
GUI::GUI ( )

Definition at line 8 of file GUI.cpp.
00009 {
00010
00011     // give some arbitrary values; we will update these values after 1 frame :
00012     this->direcional_light_radiance = 4.0f;
00013
00014     this->directional_light_color[0] = 1;
00015     this->directional_light_color[1] = 1;
00016     this->directional_light_color[2] = 1;
00017
00018     this->directional_light_direction[0] = -0.1f;
00019     this->directional_light_direction[1] = -1.f;
00020     this->directional_light_direction[2] = -0.1f;
00021
00022     this->cloud_speed = 6;
00023     this->cloud_scale = 0.63f;
00024     this->cloud_density = 0.493f;
00025     this->cloud_pillowness = 0.966f;
00026     this->cloud_cirrus_effect = 0.034f;
00027
00028     this->cloud_mesh_scale[0] = 1000.f;
00029     this->cloud_mesh_scale[1] = 5.f;
00030     this->cloud_mesh_scale[2] = 1000.f;
00031
00032     this->cloud_mesh_offset[0] = -.364f;
00033     this->cloud_mesh_offset[1] = 367.f;
00034     this->cloud_mesh_offset[2] = -18.351f;
00035
00036     this->cloud_powder_effect = true;
00037
00038     this->cloud_movement_direction[0] = 1.f;
00039     this->cloud_movement_direction[1] = 1.f;
00040     this->cloud_movement_direction[2] = 1.f;
00041
00042     this->cloud_num_march_steps = 8;
00043     this->cloud_num_march_steps_to_light = 3;
00044
00045     this->shadow_map_res_index = 3;
00046     this->shadow_resolution_changed = false;
00047     this->num_shadow_cascades = NUM_CASCADES;
00048     this->pcf_radius = 2;
00049     this->cascaded_shadow_intensity = 0.65f;
00050
00051     this->available_shadow_map_resolutions[0] = "512";
00052     this->available_shadow_map_resolutions[1] = "1024";
00053     this->available_shadow_map_resolutions[2] = "2048";
00054     this->available_shadow_map_resolutions[3] = "4096";
00055
00056
00057     std::stringstream texture_base_dir;
00058     texture_base_dir << CMAKELISTS_DIR;
00059     texture_base_dir << "/Resources/Textures/";
00060
00061     std::stringstream texture_logo;
00062     texture_logo << texture_base_dir.str() << "Loading_Screen/Engine_logo.png";
00063     logo_tex = Texture(texture_logo.str().c_str(), std::make_shared<RepeatMode>());
00064     logo_tex.load_texture_with_alpha_channel();
00065 }
```

References `available_shadow_map_resolutions`, `cascaded_shadow_intensity`, `cloud_cirrus_effect`, `cloud_density`, `cloud_mesh_offset`, `cloud_mesh_scale`, `cloud_movement_direction`, `cloud_num_march_steps`, `cloud_num_march_steps_to_light`, `cloud_pillowness`, `cloud_powder_effect`, `cloud_scale`, `cloud_speed`, `direcional_light_radiance`, `directional_light_color`, `directional_light_direction`, `Texture::load_texture_with_alpha_channel()`, `logo_tex`, `NUM_CASCADES`, `num_shadow_cascades`, `pcf_radius`, `shadow_map_res_index`, and `shadow_resolution_changed`.

Here is the call graph for this function:

6.16.2.2 ~GUI()

```
GUI::~GUI ( )

Definition at line 262 of file GUI.cpp.
00263 {
00264     ImGui_ImplOpenGL3_Shutdown();
00265     ImGui_ImplGlfw_Shutdown();
00266     ImGui::DestroyContext();
00267 }
```

6.16.3 Member Function Documentation

6.16.3.1 init()

```
void GUI::init (
    std::shared_ptr< Window > main_window )
```

Definition at line 67 of file [GUI.cpp](#).

```
00068 {
00069     // Setup Dear ImGui context
00070     IMGUI_CHECKVERSION();
00071     ImGui::CreateContext();
00072     //ImGuiIO& io = ImGui::GetIO();
00073     const ImGuiStyle& style = ImGui::GetStyle();
00074     // Setup Platform/Renderer bindings
00075     ImGui_ImplGlfw_InitForOpenGL(main_window->get_window(), true);
00076     const char* glsl_version = "#version 460";
00077     ImGui_ImplOpenGL3_Init(glsl_version);
00078     // Setup Dear ImGui style
00079     ImGui::StyleColorsDark();
00080     ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 10);
00081     ImGui::PushStyleVar(ImGuiStyleVar_FrameRounding, 10);
00082     ImGui::PushStyleVar(ImGuiStyleVar_FrameBorderSize, 1);
00083 }
```

Referenced by [main\(\)](#).

Here is the caller graph for this function:

6.16.3.2 render()

```
void GUI::render (
    bool loading_in_progress,
    float progress,
    bool & shader_hot_reload_triggered )
```

Definition at line 85 of file [GUI.cpp](#).

```
00086 {
00087
00088     // feed inputs to dear imgui, start new frame
00089     //UI.start_new_frame();
00090     ImGui_ImplOpenGL3_NewFrame();
00091     ImGui_ImplGlfw_NewFrame();
00092     ImGui::NewFrame();
00093     // render your GUI
00094     ImGui::Begin("GUI v1.3.3");
00095
00096     if (loading_in_progress) {
00097         ImGui::ProgressBar(progress, ImVec2(0.0f, 0.0f));
00098         ImGui::SameLine(0.0f, ImGui::GetStyle().ItemInnerSpacing.x);
00099         ImGui::Text("Loading scene");
00100         ImGui::Separator();
00101     }
00102
00103
00104     if (ImGui::CollapsingHeader("Hot shader reload")) {
00105
00106         if (ImGui::Button("Hot reload ALL shaders!")) {
00107             shader_hot_reload_triggered = true;
00108         }
00109     }
00110
00111     ImGui::Separator();
00112
00113     if (ImGui::CollapsingHeader("Graphic Settings")) {
00114
00115         if (ImGui::TreeNode("Directional Light"))
00116             ImGui::Separator();
```

```

00117     ImGui::SliderFloat("Radiance", &direccional_light_radiance, 0.0f, 50.0f);
00118     ImGui::Separator();
00119     // Edit a color (stored as ~4 floats)
00120     ImGui::ColorEdit3("Directional Light Color", directional_light_color);
00121     ImGui::Separator();
00122     ImGui::SliderFloat3("Light Direction", directional_light_direction, -1.f, 1.0f);
00123
00124     if (ImGui::TreeNode("Shadows")) {
00125
00126         int shadow_map_res_index_before = shadow_map_res_index;
00127         ImGui::Combo("Shadow Map Resolution", &shadow_map_res_index, available_shadow_map_resolutions,
00128             IM_ARRAYSIZE(available_shadow_map_resolutions));
00129         if (shadow_map_res_index_before != shadow_map_res_index) shadow_resolution_changed = true;
00130
00131         int num_cascades_before = num_shadow_cascades;
00132         ImGui::SliderInt("# cascades", &num_shadow_cascades, NUM_MIN_CASCADES, NUM_CASCADES);
00133         if (num_cascades_before != num_shadow_cascades) shadow_resolution_changed = true;
00134
00135         ImGui::SliderInt("PCF radius", &pcf_radius, 1, 20);
00136         ImGui::SliderFloat("Shadow intensity", &cascaded_shadow_intensity, 0.0f, 1.0f);
00137
00138         ImGui::TreePop();
00139     }
00140
00141     ImGui::TreePop();
00142
00143
00144     if (ImGui::TreeNode("Cloud Settings")) {
00145
00146         ImGui::SliderInt("Speed", &cloud_speed, 0, 30);
00147         ImGui::SliderInt("# march steps", &cloud_num_march_steps, 1, 128);
00148         ImGui::SliderInt("# march steps to light", &cloud_num_march_steps_to_light, 1, 128);
00149         ImGui::SliderFloat3("Movement Direction", cloud_movement_direction, -10.0f, 10.0f);
00150         ImGui::SliderFloat("Illumination intensity", &cloud_scale, 0.0f, 1.0f);
00151         ImGui::SliderFloat("Density", &cloud_density, 0.0f, 1.0f);
00152         ImGui::SliderFloat("Pillowness", &cloud_pillowness, 0.0f, 1.0f);
00153         ImGui::SliderFloat("Cirrus effect", &cloud_cirrus_effect, 0.0f, 1.0f);
00154         ImGui::Checkbox("Powder effect", &cloud_powder_effect);
00155         ImGui::SliderFloat3("Scale", cloud_mesh_scale, 0.0f, 1000.0f);
00156         ImGui::SliderFloat3("Translation", cloud_mesh_offset, -200.0f, 400.0f);
00157
00158         ImGui::TreePop();
00159     }
00160 }
00161
00162     ImGui::Separator();
00163
00164 /*if (ImGui::CollapsingHeader("Audio Settings")) {
00165
00166     ImGui::SliderFloat("Volume", &sound_volume, 0.0f, 1.0f);
00167
00168 }*/
00169
00170     ImGui::Separator();
00171
00172     if (ImGui::CollapsingHeader("GUI Settings")) {
00173         ImGuiStyle& style = ImGui::GetStyle();
00174         if (ImGui::SliderFloat("Frame Rounding", &style.FrameRounding, 0.0f, 12.0f, "%0.0f")) {
00175             style.GrabRounding = style.FrameRounding; // Make GrabRounding always the same value as
00176             FrameRounding
00177         }
00178         {
00179             bool border = (style.FrameBorderSize > 0.0f);
00180             if (ImGui::Checkbox("FrameBorder", &border)) {
00181                 style.FrameBorderSize = border ? 1.0f : 0.0f;
00182             }
00183             ImGui::SliderFloat("WindowRounding", &style.WindowRounding, 0.0f, 12.0f, "%0.0f");
00184         }
00185
00186     ImGui::Separator();
00187
00188     if (ImGui::CollapsingHeader("KEY Bindings")) {
00189
00190         ImGui::Text("WASD for moving Forward, backward and to the side\nQE for rotating ");
00191
00192
00193     ImGui::Separator();
00194
00195     ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate,
00196     ImGui::GetIO().Framerate);
00197     //ImGui::ShowDemoWindow();
00198
00199     ImGui::Image((void*)(intptr_t)logo_tex.get_id(), ImVec2(200, 200), ImVec2(0, 1), ImVec2(1, 0));
00200
00201     ImGui::End();

```

```

00201
00202     // feed inputs to dear imgui, start new frame
00203     ImGui::Render();
00204     ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
00205 }
```

References `available_shadow_map_resolutions`, `cascaded_shadow_intensity`, `cloud_cirrus_effect`, `cloud_density`, `cloud_mesh_offset`, `cloud_mesh_scale`, `cloud_movement_direction`, `cloud_num_march_steps`, `cloud_num_march_steps_to_light`, `cloud_pillowness`, `cloud_powder_effect`, `cloud_scale`, `cloud_speed`, `direcional_light_radiance`, `directional_light_color`, `directional_light_direction`, `Texture::get_id()`, `logo_tex`, `NUM_CASCADES`, `NUM_MIN_CASCADES`, `num_shadow_cascades`, `pcf_radius`, `shadow_map_res_index`, and `shadow_resolution_changed`.

Referenced by `main()`.

Here is the call graph for this function: Here is the caller graph for this function:

6.16.3.3 update_user_input()

```
void GUI::update_user_input (
    std::shared_ptr< Scene > scene )
```

Definition at line 207 of file `GUI.cpp`.

```

00208 {
00209     std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00210     main_light->set_radiance(direcional_light_radiance);
00211     main_light->get_shadow_map()->set_intensity(cascaded_shadow_intensity);
00212     main_light->get_shadow_map()->set_pcf_radius(pcf_radius);
00213
00214     glm::vec3 new_main_light_color(directional_light_color[0], directional_light_color[1],
00215                                     directional_light_color[2]);
00216     main_light->set_color(new_main_light_color);
00217
00218     glm::vec3 new_main_light_pos(directional_light_direction[0], directional_light_direction[1],
00219                                 directional_light_direction[2]);
00220     main_light->set_direction(new_main_light_pos);
00221
00222     glm::vec3 cloud_move(cloud_movement_direction[0], cloud_movement_direction[1],
00223                           cloud_movement_direction[2]);
00224
00225     std::shared_ptr<Clouds> clouds = scene->get_clouds();
00226
00227     clouds->set_movement_direction(cloud_move);
00228     clouds->set_movement_speed(static_cast<float>(cloud_speed));
00229     clouds->set_density(cloud_density);
00230     clouds->set_scale(cloud_scale);
00231     clouds->set_pillowness(cloud_pillowness);
00232     clouds->set_cirrus_effect(cloud_cirrus_effect);
00233     clouds->set_powder_effect(cloud_powder_effect);
00234
00235     clouds->set_scale(glm::vec3(cloud_mesh_scale[0], cloud_mesh_scale[1], cloud_mesh_scale[2]));
00236     clouds->set_translation(glm::vec3(cloud_mesh_offset[0], cloud_mesh_offset[1],
00237                                 cloud_mesh_offset[2]));
00238
00239     GLfloat shadow_map_resolution = 4096.f;
00240
00241     if (shadow_resolution_changed) {
00242
00243         switch (shadow_map_res_index) {
00244             case 0:
00245                 shadow_map_resolution = 512.f;
00246                 break;
00247             case 1:
00248                 shadow_map_resolution = 1024.f;
00249                 break;
00250             case 2:
00251                 shadow_map_resolution = 2048.f;
00252                 break;
00253             case 3:
00254                 shadow_map_resolution = 4096.f;
00255         }
00256
00257     main_light->update_shadow_map(shadow_map_resolution, shadow_map_resolution, NUM_CASCADES);
```

```
00258     shadow_resolution_changed = false;
00259 }
00260 }
```

References [cascaded_shadow_intensity](#), [cloud_cirrus_effect](#), [cloud_density](#), [cloud_mesh_offset](#), [cloud_mesh_scale](#), [cloud_movement_direction](#), [cloud_pillowness](#), [cloud_powder_effect](#), [cloud_scale](#), [cloud_speed](#), [direcional_light_radiance](#), [directional_light_color](#), [directional_light_direction](#), [NUM_CASCADES](#), [pcf_radius](#), [shadow_map_res_index](#), and [shadow_resolution_changed](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:

6.16.4 Field Documentation

6.16.4.1 available_shadow_map_resolutions

```
const char* GUI::available_shadow_map_resolutions[4] [private]
```

Definition at line 50 of file [GUI.h](#).

Referenced by [GUI\(\)](#), and [render\(\)](#).

6.16.4.2 cascaded_shadow_intensity

```
float GUI::cascaded_shadow_intensity [private]
```

Definition at line 49 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.3 cloud_cirrus_effect

```
float GUI::cloud_cirrus_effect [private]
```

Definition at line 37 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.4 `cloud_density`

```
float GUI::cloud_density [private]
```

Definition at line 35 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.5 `cloud_mesh_offset`

```
float GUI::cloud_mesh_offset[3] [private]
```

Definition at line 39 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.6 `cloud_mesh_scale`

```
float GUI::cloud_mesh_scale[3] [private]
```

Definition at line 38 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.7 `cloud_movement_direction`

```
float GUI::cloud_movement_direction[3] [private]
```

Definition at line 41 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.8 `cloud_num_march_steps`

```
int GUI::cloud_num_march_steps [private]
```

Definition at line 42 of file [GUI.h](#).

Referenced by [GUI\(\)](#), and [render\(\)](#).

6.16.4.9 `cloud_num_march_steps_to_light`

```
int GUI::cloud_num_march_steps_to_light [private]
```

Definition at line 43 of file [GUI.h](#).

Referenced by [GUI\(\)](#), and [render\(\)](#).

6.16.4.10 `cloud_pillowness`

```
float GUI::cloud_pillowness [private]
```

Definition at line 36 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.11 `cloud_powder_effect`

```
bool GUI::cloud_powder_effect [private]
```

Definition at line 40 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.12 `cloud_scale`

```
float GUI::cloud_scale [private]
```

Definition at line 34 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.13 `cloud_speed`

```
int GUI::cloud_speed [private]
```

Definition at line 33 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.14 `direcional_light_radiance`

```
float GUI::direcional_light_radiance [private]
```

Definition at line 29 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.15 `directional_light_color`

```
float GUI::directional_light_color[3] [private]
```

Definition at line 30 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.16 `directional_light_direction`

```
float GUI::directional_light_direction[3] [private]
```

Definition at line 31 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.17 `logo_tex`

```
Texture GUI::logo_tex [private]
```

Definition at line 27 of file [GUI.h](#).

Referenced by [GUI\(\)](#), and [render\(\)](#).

6.16.4.18 `num_shadow_cascades`

```
int GUI::num_shadow_cascades [private]
```

Definition at line 47 of file [GUI.h](#).

Referenced by [GUI\(\)](#), and [render\(\)](#).

6.16.4.19 pcf_radius

```
int GUI::pcf_radius [private]
```

Definition at line 48 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.20 shadow_map_res_index

```
int GUI::shadow_map_res_index [private]
```

Definition at line 45 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

6.16.4.21 shadow_resolution_changed

```
bool GUI::shadow_resolution_changed [private]
```

Definition at line 46 of file [GUI.h](#).

Referenced by [GUI\(\)](#), [render\(\)](#), and [update_user_input\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/gui/[GUI.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/gui/[GUI.cpp](#)

6.17 std::hash< Vertex > Struct Reference

```
#include <Vertex.h>
```

Collaboration diagram for std::hash< Vertex >:

Public Member Functions

- size_t [operator\(\)](#) ([Vertex](#) const &vert) const

6.17.1 Detailed Description

Definition at line 48 of file [Vertex.h](#).

6.17.2 Member Function Documentation

6.17.2.1 operator()()

```
size_t std::hash< Vertex >::operator() (
    Vertex const & vert ) const [inline]
```

Definition at line 50 of file [Vertex.h](#).

```
00051 {
00052
00053     size_t h1 = hash<glm::vec3>()(vert.position);
00054     size_t h2 = hash<glm::vec3>()(vert.normal);
00055     size_t h3 = hash<glm::vec2>()(vert.texture_coords);
00056
00057     // combine hashed wonderfully :)))
00058     return ((h1 ^ (h2 << 1)) >> 1) ^ h3;
00059 }
```

References [Vertex::normal](#), [Vertex::position](#), and [Vertex::texture_coords](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Vertex.h](#)

6.18 Light Class Reference

```
#include <Light.h>
```

Inheritance diagram for Light:

Collaboration diagram for Light:

Public Member Functions

- [Light \(\)](#)
- [Light \(GLfloat red, GLfloat green, GLfloat blue, GLfloat radiance\)](#)
- [glm::vec3 get_color \(\) const](#)
- [float get_radiance \(\) const](#)
- [~Light \(\)](#)

Protected Attributes

- [glm::vec3 color](#)
- [float radiance](#)
- [glm::mat4 light_proj](#)

6.18.1 Detailed Description

Definition at line 8 of file [Light.h](#).

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Light() [1/2]

```
Light::Light ( )
```

Definition at line 3 of file [Light.cpp](#).

```
00003     :
00004
00005     color(glm::vec3(1.0f)), radiance(1.0f)
00006
00007 {
00008 }
```

6.18.2.2 Light() [2/2]

```
Light::Light (
    GLfloat red,
    GLfloat green,
    GLfloat blue,
    GLfloat radiance )
```

Definition at line 10 of file [Light.cpp](#).

```
00010     :
00011
00012     color(glm::vec3(red, green, blue)), radiance(radiance)
00013
00014 {
00015 }
```

6.18.2.3 ~Light()

```
Light::~Light ( )
```

Definition at line 17 of file [Light.cpp](#).

```
00017 { }
```

6.18.3 Member Function Documentation

6.18.3.1 get_color()

```
glm::vec3 Light::get_color ( ) const [inline]
```

Definition at line 14 of file [Light.h](#).

```
00014 { return color; };
```

References [color](#).

6.18.3.2 get_radiance()

```
float Light::get_radiance ( ) const [inline]
```

Definition at line 15 of file [Light.h](#).
00015 { **return radiance;** };

References [radiance](#).

6.18.4 Field Documentation

6.18.4.1 color

```
glm::vec3 Light::color [protected]
```

Definition at line 20 of file [Light.h](#).

Referenced by [DirectionalLight::get_color\(\)](#), [get_color\(\)](#), and [DirectionalLight::set_color\(\)](#).

6.18.4.2 light_proj

```
glm::mat4 Light::light_proj [protected]
```

Definition at line 23 of file [Light.h](#).

Referenced by [DirectionalLight::calculate_light_transform\(\)](#), [PointLight::calculate_light_transform\(\)](#), [DirectionalLight::DirectionalLight\(\)](#) and [PointLight::PointLight\(\)](#).

6.18.4.3 radiance

```
float Light::radiance [protected]
```

Definition at line 21 of file [Light.h](#).

Referenced by [DirectionalLight::get_radiance\(\)](#), [get_radiance\(\)](#), and [DirectionalLight::set_radiance\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/[Light.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/[Light.cpp](#)

6.19 LightingPass Class Reference

```
#include <LightingPass.h>
```

Inheritance diagram for LightingPass:

Collaboration diagram for LightingPass:

Public Member Functions

- [LightingPass \(\)](#)
- void [execute \(glm::mat4 projection_matrix, std::shared_ptr< Camera >, std::shared_ptr< Scene > scene, std::shared_ptr< GBuffer > gbuffer, float delta_time\)](#)
- void [create_shader_program \(\)](#)
- [~LightingPass \(\)](#)

Private Member Functions

- void [set_uniforms \(glm::mat4 projection_matrix, std::shared_ptr< Camera > main_camera, std::shared_ptr< Scene > scene, std::shared_ptr< GBuffer > gbuffer, float delta_time\)](#)

Private Attributes

- glm::vec3 [current_offset](#)
- std::shared_ptr< [LightingPassShaderProgram](#) > [shader_program](#)
- Quad [quad](#)

6.19.1 Detailed Description

Definition at line 23 of file [LightingPass.h](#).

6.19.2 Constructor & Destructor Documentation

6.19.2.1 LightingPass()

```
LightingPass::LightingPass ( )
```

Definition at line 3 of file [LightingPass.cpp](#).

```
00003 :
00004
00005     current_offset(glm::vec3(0.0f)), quad()
00006
00007 {
00008
00009     create_shader_program();
0010 }
```

References [create_shader_program\(\)](#).

Here is the call graph for this function:

6.19.2.2 ~LightingPass()

LightingPass::~LightingPass ()

Definition at line 220 of file [LightingPass.cpp](#).
00220 { }

6.19.3 Member Function Documentation

6.19.3.1 create_shader_program()

void LightingPass::create_shader_program () [virtual]

Implements [RenderPass](#).

Definition at line 40 of file [LightingPass.cpp](#).

```
00041 {
00042     shader_program = std::make_shared<LightingPassShaderProgram>(LightingPassShaderProgram{});
00043     shader_program->create_from_files("rasterizer/g_buffer_lighting_pass.vert",
00044         "rasterizer/g_buffer_lighting_pass.frag");
```

References [shader_program](#).

Referenced by [LightingPass\(\)](#).

Here is the caller graph for this function:

6.19.3.2 execute()

```
void LightingPass::execute (
    glm::mat4 projection_matrix,
    std::shared_ptr< Camera > main_camera,
    std::shared_ptr< Scene > scene,
    std::shared_ptr< GBuffer > gbuffer,
    float delta_time )
```

Definition at line 12 of file [LightingPass.cpp](#).

```
00014 {
00015
00016     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00017
00018     std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00019     std::shared_ptr<Clouds> cloud = scene->get_clouds();
00020     std::vector<std::shared_ptr<PointLight>> point_lights = scene->get_point_lights();
00021
00022     shader_program->use_shader_program();
00023
00024     set_uniforms(projection_matrix, main_camera, scene, gbuffer, delta_time);
00025
00026     //bind textures to their units
00027     main_light->get_shadow_map()->read(D_LIGHT_SHADOW_TEXTURES_SLOT);
00028     cloud->read();
00029
00030     for (uint32_t i = 0; i < static_cast<GLuint>(point_lights.size()); i++) {
00031         point_lights[i]->get_omni_shadow_map()->read(P_LIGHT_SHADOW_TEXTURES_SLOT + i);
00032     }
00033
00034     // render screen filling quad
00035     quad.render();
00036
00037     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00038 }
```

References [D_LIGHT_SHADOW_TEXTURES_SLOT](#), [P_LIGHT_SHADOW_TEXTURES_SLOT](#), [quad](#), [Quad::render\(\)](#), [set_uniforms\(\)](#), and [shader_program](#).

Here is the call graph for this function:

6.19.3.3 set_uniforms()

```
void LightingPass::set_uniforms (
    glm::mat4 projection_matrix,
    std::shared_ptr< Camera > main_camera,
    std::shared_ptr< Scene > scene,
    std::shared_ptr< GBuffer > gbuffer,
    float delta_time ) [private]
```

Definition at line 46 of file [LightingPass.cpp](#).

```
00048 {
00049
00050 // VP
00051 glm::mat4 view_matrix = main_camera->get_viewmatrix();
00052 shader_program->setUniformMatrix4fv(view_matrix, "view");
00053 shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00054
00055 // SUN UNIFORMS
00056 std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00057 shader_program->setUniformFloat(main_light->get_radiance(), "directional_light.base.radiance");
00058 shader_program->setUniformVec3(main_light->get_color(), "directional_light.base.color");
00059 shader_program->setUniformVec3(main_light->get_direction(), "directional_light.direction");
00060
00061 // EVERYTHING REGARDING THE SHADOW CASCADE
00062 shader_program->setUniformInt(D_LIGHT_SHADOW_TEXTURES_SLOT, "directional_shadow_maps");
00063
00064 std::vector<GLfloat> cascade_slots = main_light->get_cascaded_slots();
00065
00066 std::stringstream ss;
00067 for (uint32_t i = 0; i < NUM_CASCADES; i++) {
00068
00069     glm::vec4 clip_end_slot = projection_matrix * glm::vec4(0.0f, 0.0f, -cascade_slots[i + 1], 1.0f);
00070     ss << "cascade_endpoints[" << i << "]";
00071     shader_program->setUniformFloat(clip_end_slot.z, ss.str());
00072     ss.clear();
00073     ss.str(std::string());
00074 }
00075
00076 shader_program->setUniformInt(main_light->get_shadow_map()->get_pcf_radius(), "pcf_radius");
00077
00078 // READ GBUFFER
00079 gbuffer->read(shader_program);
00080
00081 // POINT LIGHTS
00082 std::vector<std::shared_ptr<PointLight>> point_lights = scene->get_point_lights();
00083
00084 shader_program->setUniformInt(static_cast<uint32_t>(point_lights.size()), "point_light_count");
00085
00086 for (uint32_t i = 0; i < static_cast<uint32_t>(point_lights.size()); i++) {
00087
00088     ss << "point_lights[" << i << "].base.color";
00089     shader_program->setUniformVec3(point_lights[i]->get_color(), ss.str());
00090     ss.clear();
00091     ss.str(std::string());
00092
00093     ss << "point_lights[" << i << "].base.radiance";
00094     shader_program->setUniformFloat(point_lights[i]->get_radiance(), ss.str());
00095     ss.clear();
00096     ss.str(std::string());
00097
00098     ss << "point_lights[" << i << "].position";
00099     shader_program->setUniformVec3(point_lights[i]->get_position(), ss.str());
00100     ss.clear();
00101     ss.str(std::string());
00102
00103     ss << "point_lights[" << i << "].base.constant";
00104     shader_program->setUniformFloat(point_lights[i]->get_constant_factor(), ss.str());
00105     ss.clear();
00106     ss.str(std::string());
00107
00108     ss << "point_lights[" << i << "].linear";
00109     shader_program->setUniformFloat(point_lights[i]->get_linear_factor(), ss.str());
00110     ss.clear();
00111     ss.str(std::string());
00112
00113     ss << "point_lights[" << i << "].exponent";
00114     shader_program->setUniformFloat(point_lights[i]->get_exponent_factor(), ss.str());
00115     ss.clear();
00116     ss.str(std::string());
00117
00118     ss << "omni_shadow_maps[" << i << "].shadow_map";
00119     shader_program->setUniformInt((GLint)(P_LIGHT_SHADOW_TEXTURES_SLOT + i), ss.str());
```

```

00120     ss.clear();
00121     ss.str(std::string());
00122
00123     ss << "omni_shadow_maps[" << i << "]far_plane";
00124     shader_program->setUniformFloat(point_lights[i]->get_far_plane(), ss.str());
00125     ss.clear();
00126     ss.str(std::string());
00127 }
00128
00129 // CAMERA
00130 glm::vec3 camera_position = main_camera->get_camera_position();
00131 shader_program->setUniformVec3(camera_position, "eye_position");
00132
00133 // MATERIALS
00134 std::vector<ObjMaterial> materials = scene->get_materials();
00135 for (uint32_t i = 0; i < static_cast<uint32_t>(materials.size()); i++) {
00136
00137     ss << "materials[" << i << "].ambient";
00138     shader_program->setUniformVec3(materials[i].get_ambient(), ss.str());
00139     ss.clear();
00140     ss.str(std::string());
00141
00142     ss << "materials[" << i << "].diffuse";
00143     shader_program->setUniformVec3(materials[i].get_diffuse(), ss.str());
00144     ss.clear();
00145     ss.str(std::string());
00146
00147     ss << "materials[" << i << "].specular";
00148     shader_program->setUniformVec3(materials[i].get_specular(), ss.str());
00149     ss.clear();
00150     ss.str(std::string());
00151
00152     ss << "materials[" << i << "].transmittance";
00153     shader_program->setUniformVec3(materials[i].get_transmittance(), ss.str());
00154     ss.clear();
00155     ss.str(std::string());
00156
00157     ss << "materials[" << i << "].emission";
00158     shader_program->setUniformVec3(materials[i].get_emission(), ss.str());
00159     ss.clear();
00160     ss.str(std::string());
00161
00162     ss << "materials[" << i << "].shininess";
00163     shader_program->setUniformFloat(materials[i].get_shininess(), ss.str());
00164     ss.clear();
00165     ss.str(std::string());
00166
00167     ss << "materials[" << i << "].ior";
00168     shader_program->setUniformFloat(materials[i].get_ior(), ss.str());
00169     ss.clear();
00170     ss.str(std::string());
00171
00172     ss << "materials[" << i << "].dissolve";
00173     shader_program->setUniformFloat(materials[i].get_dissolve(), ss.str());
00174     ss.clear();
00175     ss.str(std::string());
00176
00177     ss << "materials[" << i << "].illum";
00178     shader_program->setUniformInt(materials[i].get_illum(), ss.str());
00179     ss.clear();
00180     ss.str(std::string());
00181
00182     ss << "materials[" << i << "].textureID";
00183     shader_program->setUniformInt(materials[i].get_textureID(), ss.str());
00184     ss.clear();
00185     ss.str(std::string());
00186 }
00187
00188 // CLOUDS
00189
00190 std::shared_ptr<Clouds> cloud = scene->get_clouds();
00191
00192 shader_program->setUniformVec3(cloud->get_radius(), "cloud.radius");
00193 GLfloat velocity = cloud->get_movement_speed() * delta_time;
00194 current_offset = current_offset + cloud->get_movement_direction() * velocity;
00195 shader_program->setUniformVec3(current_offset, "cloud.offset");
00196 shader_program->setUniformMatrix4fv(cloud->get_model(), "cloud.model_to_world");
00197 shader_program->setUniformFloat(cloud->get_scale(), "cloud.scale");
00198 shader_program->setUniformFloat(1.f - cloud->get_density(), "cloud.threshold");
00199 shader_program->setUniformFloat(cloud->get_pillowness(), "cloud.pillowness");
00200 shader_program->setUniformFloat(cloud->get_cirrus_effect(), "cloud.cirrus_effect");
00201 shader_program->setUniformInt(cloud->get_num_march_steps(), "cloud.num_march_steps");
00202 shader_program->setUniformInt(cloud->get_num_march_steps_to_light(),
00203 "cloud.num_steps_to_light");
00204 if (cloud->get_powder_effect()) {
00205     shader_program->setUniformInt(true, "cloud.powder_effect");

```

```

00206 } else {
00207     shader_program->setUniformInt(false, "cloud.powder_effect");
00208 }
00209
00210     shader_program->setUniformBlockBinding(UNIFORM_LIGHT_MATRICES_BINDING, "LightSpaceMatrices");
00211
00212     shader_program->setUniformInt(RANDOM_NUMBERS_SLOT, "random_number");
00213
00214     shader_program->setUniformInt(NOISE_128D_TEXTURES_SLOT, "noise_texture_1");
00215     shader_program->setUniformInt(NOISE_32D_TEXTURES_SLOT, "noise_texture_2");
00216
00217     shader_program->validate_program();
00218 }
```

References `current_offset`, `D_LIGHT_SHADOW_TEXTURES_SLOT`, `NOISE_128D_TEXTURES_SLOT`, `NOISE_32D_TEXTURES_SLOT`, `NUM_CASCADES`, `P_LIGHT_SHADOW_TEXTURES_SLOT`, `RANDOM_NUMBERS_SLOT`, `shader_program`, and `UNIFORM_LIGHT_MATRICES_BINDING`.

Referenced by [execute\(\)](#).

Here is the caller graph for this function:

6.19.4 Field Documentation

6.19.4.1 current_offset

```
glm::vec3 LightingPass::current_offset [private]
```

Definition at line 34 of file [LightingPass.h](#).

Referenced by [set_uniforms\(\)](#).

6.19.4.2 quad

```
Quad LightingPass::quad [private]
```

Definition at line 41 of file [LightingPass.h](#).

Referenced by [execute\(\)](#).

6.19.4.3 shader_program

```
std::shared_ptr<LightingPassShaderProgram> LightingPass::shader_program [private]
```

Definition at line 39 of file [LightingPass.h](#).

Referenced by [create_shader_program\(\)](#), [execute\(\)](#), and [set_uniforms\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[LightingPass.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[LightingPass.cpp](#)

6.20 LightingPassShaderProgram Class Reference

```
#include <LightingPassShaderProgram.h>
```

Inheritance diagram for LightingPassShaderProgram:

Collaboration diagram for LightingPassShaderProgram:

Public Member Functions

- [LightingPassShaderProgram \(\)](#)
- [~LightingPassShaderProgram \(\)](#)

Additional Inherited Members

6.20.1 Detailed Description

Definition at line [6](#) of file [LightingPassShaderProgram.h](#).

6.20.2 Constructor & Destructor Documentation

6.20.2.1 [LightingPassShaderProgram\(\)](#)

```
LightingPassShaderProgram::LightingPassShaderProgram ( )
```

Definition at line [4](#) of file [LightingPassShaderProgram.cpp](#).
00004 { }

6.20.2.2 [~LightingPassShaderProgram\(\)](#)

```
LightingPassShaderProgram::~LightingPassShaderProgram ( )
```

Definition at line [6](#) of file [LightingPassShaderProgram.cpp](#).
00006 { }

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[LightingPassShaderProgram.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[LightingPassShaderProgram.cpp](#)

6.21 LoadingScreen Class Reference

```
#include <LoadingScreen.h>
```

Collaboration diagram for LoadingScreen:

Public Member Functions

- [LoadingScreen \(\)](#)
- void [init \(\)](#)
- void [render \(\)](#)
- [~LoadingScreen \(\)](#)

Private Member Functions

- void [create_shader_program \(\)](#)

Private Attributes

- Quad [loading_screen_quad](#)
- Texture [loading_screen_tex](#)
- Texture [logo_tex](#)
- std::shared_ptr< ShaderProgram > [loading_screen_shader_program](#)

6.21.1 Detailed Description

Definition at line [6](#) of file [LoadingScreen.h](#).

6.21.2 Constructor & Destructor Documentation

6.21.2.1 LoadingScreen()

```
LoadingScreen::LoadingScreen ( )
```

Definition at line [5](#) of file [LoadingScreen.cpp](#).
00005 { [create_shader_program\(\)](#); }

References [create_shader_program\(\)](#).

Here is the call graph for this function:

6.21.2.2 ~LoadingScreen()

```
LoadingScreen::~LoadingScreen ( )
```

Definition at line 48 of file [LoadingScreen.cpp](#).
00048 { }

6.21.3 Member Function Documentation

6.21.3.1 create_shader_program()

```
void LoadingScreen::create_shader_program ( ) [private]
```

Definition at line 40 of file [LoadingScreen.cpp](#).

```
00041 {
00042
00043     loading_screen_shader_program = std::make_shared<ShaderProgram>(ShaderProgram{});
00044
00045     loading_screen_shader_program->create_from_files("loading_screen/loading_screen.vert",
00046         "loading_screen/loading_screen.frag");
```

References [loading_screen_shader_program](#).

Referenced by [LoadingScreen\(\)](#).

Here is the caller graph for this function:

6.21.3.2 init()

```
void LoadingScreen::init ( )
```

Definition at line 7 of file [LoadingScreen.cpp](#).

```
00008 {
00009
00010     std::stringstream texture_base_dir;
00011     texture_base_dir << CMAKELISTS_DIR;
00012     texture_base_dir << "/Resources/Textures/";
00013
00014     std::stringstream texture_loading_screen;
00015     texture_loading_screen << texture_base_dir.str() << "Loading_Screen/ukraine.jpg";
00016
00017     loading_screen_tex = Texture(texture_loading_screen.str().c_str(), std::make_shared<RepeatMode>());
00018     loading_screen_tex.load_texture_without_alpha_channel();
00019
00020     std::stringstream texture_logo;
00021     texture_logo << texture_base_dir.str() << "Loading_Screen/Engine_logo.png";
00022     logo_tex = Texture(texture_logo.str().c_str(), std::make_shared<RepeatMode>());
00023     logo_tex.load_texture_with_alpha_channel();
00024 }
```

References [Texture::load_texture_with_alpha_channel\(\)](#), [Texture::load_texture_without_alpha_channel\(\)](#), [loading_screen_tex](#), and [logo_tex](#).

Referenced by [main\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.21.3.3 render()

```
void LoadingScreen::render ( )
```

Definition at line 26 of file [LoadingScreen.cpp](#).

```
00027 {
00028     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00029
00030     loading_screen_tex.use_texture(0);
00031
00032     loading_screen_shader_program->use_shader_program();
00033
00034     loading_screen_quad.render();
00035
00036     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00037
00038 }
```

References [loading_screen_quad](#), [loading_screen_shader_program](#), [loading_screen_tex](#), [Quad::render\(\)](#), and [Texture::use_texture\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.21.4 Field Documentation

6.21.4.1 loading_screen_quad

```
Quad LoadingScreen::loading_screen_quad [private]
```

Definition at line 17 of file [LoadingScreen.h](#).

Referenced by [render\(\)](#).

6.21.4.2 loading_screen_shader_program

```
std::shared_ptr<ShaderProgram> LoadingScreen::loading_screen_shader_program [private]
```

Definition at line 21 of file [LoadingScreen.h](#).

Referenced by [create_shader_program\(\)](#), and [render\(\)](#).

6.21.4.3 loading_screen_tex

```
Texture LoadingScreen::loading_screen_tex [private]
```

Definition at line 18 of file [LoadingScreen.h](#).

Referenced by [init\(\)](#), and [render\(\)](#).

6.21.4.4 logo_tex

`Texture LoadingScreen::logo_tex [private]`

Definition at line 19 of file [LoadingScreen.h](#).

Referenced by [init\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/[LoadingScreen.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/[LoadingScreen.cpp](#)

6.22 Mesh Class Reference

#include <Mesh.h>

Collaboration diagram for Mesh:

Public Member Functions

- `Mesh (std::vector< Vertex > &vertices, std::vector< unsigned int > &indices)`
- `Mesh ()`
- `void render ()`
- `std::vector< Vertex > getVertices () const`
- `std::vector< unsigned int > getIndices () const`
- `~Mesh ()`

Private Types

- `enum { POSITION = 0, NORMAL = 1, COLOR = 2, TEXTURECOORD = 3 }`
- `enum { POSITION_VB, NUM_BUFFERS }`

Private Attributes

- `GLuint m_vao`
- `GLuint m_ibo`
- `GLuint m_vab [NUM_BUFFERS]`
- `uint32_t m_drawCount`
- `std::vector< Vertex > vertices`
- `std::vector< uint32_t > indices`

6.22.1 Detailed Description

Definition at line 11 of file [Mesh.h](#).

6.22.2 Member Enumeration Documentation

6.22.2.1 anonymous enum

`anonymous enum [private]`

Enumerator

POSITION	
NORMAL	
COLOR	
TEXTURECOORD	

Definition at line 27 of file Mesh.h.

```
00027     {
00028     POSITION = 0,
00029     NORMAL = 1,
00030     COLOR = 2,
00031     TEXTURECOORD = 3
00032
00033 };
```

6.22.2.2 anonymous enum

anonymous enum [private]

Enumerator

POSITION_VB	
NUM_BUFFERS	

Definition at line 35 of file Mesh.h.

```
00035 { POSITION_VB, NUM_BUFFERS };
```

6.22.3 Constructor & Destructor Documentation**6.22.3.1 Mesh() [1/2]**

```
Mesh::Mesh (
    std::vector< Vertex > & vertices,
    std::vector< unsigned int > & indices )
```

Definition at line 10 of file Mesh.cpp.

```
00010
00011
00012     vertices(vertices), indices(indices)
00013 {
00014
00015     uint32_t numVertices = static_cast<uint32_t>(vertices.size());
00016     uint32_t num_indices = static_cast<uint32_t>(indices.size());
00017
00018     m_drawCount = num_indices;
00019     glGenVertexArrays(1, &m_vao);
00020     glBindVertexArray(m_vao);
00021
00022     glGenBuffers(1, &m_ibo);
00023     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_ibo);
00024     // Dynamic Draw = lower Performance.
00025     glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(this->indices[0]) * num_indices, &(this->indices[0]),
GL_DYNAMIC_DRAW);
```

```

00026     glGenBuffers(NUM_BUFFERS, m_vab);
00027     glBindBuffer(GL_ARRAY_BUFFER, m_vab[POSITION_VB]);
00028     glBufferData(GL_ARRAY_BUFFER, numVertices * sizeof(this->vertices[0]), &(this->vertices[0]),
00029                 GL_DYNAMIC_DRAW);
00030
00031     //enable Vertex Attribs for Pos, Norm, Textcood
00032     // Vertex Position
00033     //Stride bytes: just the size of Vertex, offset = use offsetof funktion
00034     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, position));
00035     glEnableVertexAttribArray(0);
00036     //Vertex Normal
00037     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, normal));
00038     glEnableVertexAttribArray(1);
00039     //Vertex Normal
00040     glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, color));
00041     glEnableVertexAttribArray(2);
00042     // Vertex Texture Cood
00043     glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)(offsetof(Vertex,
00044         texture_coords)));
00044     glEnableVertexAttribArray(3);
00045
00046     //unbind everything after setting the attrs
00047     glBindBuffer(GL_ARRAY_BUFFER, 0);
00048     glBindVertexArray(0);
00049     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00050 }

```

References [indices](#), [m_drawCount](#), [m_ib0](#), [m_vab](#), [m_vao](#), [NUM_BUFFERS](#), [POSITION_VB](#), and [vertices](#).

6.22.3.2 Mesh() [2/2]

```
Mesh::Mesh ( )
```

Definition at line 8 of file [Mesh.cpp](#).

```
00008 : m_vao(-1), m_ib0(-1), m_drawCount(0), vertices(std::vector<Vertex>()),
  indices(std::vector<uint32_t>()) { }
```

6.22.3.3 ~Mesh()

```
Mesh::~Mesh ( )
```

Definition at line 66 of file [Mesh.cpp](#).

```
00067 {
00068
00069     glDeleteVertexArrays(1, &m_vao);
00070     glDeleteBuffers(1, &m_ib0);
00071     glDeleteBuffers(NUM_BUFFERS, m_vab);
00072 }
```

References [m_ib0](#), [m_vab](#), [m_vao](#), and [NUM_BUFFERS](#).

6.22.4 Member Function Documentation

6.22.4.1 getIndices()

```
std::vector< unsigned int > Mesh::getIndices ( ) const [inline]
```

Definition at line 20 of file [Mesh.h](#).
 00020 { **return** this->[indices](#); }

References [indices](#).

6.22.4.2 getVertices()

```
std::vector< Vertex > Mesh::getVertices ( ) const [inline]
```

Definition at line 19 of file [Mesh.h](#).
 00019 { **return** this->[vertices](#); }

References [vertices](#).

6.22.4.3 render()

```
void Mesh::render ( )
```

Definition at line 53 of file [Mesh.cpp](#).

```
00054 {
00055
00056     glBindVertexArray(m\_vao);
00057     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m\_ib0);
00058     //Draw Triangles
00059     glDrawElements(GL_TRIANGLES, m\_drawCount, GL_UNSIGNED_INT, 0);
00060
00061     //unbind all again
00062     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00063     glBindVertexArray(0);
00064 }
```

References [m_drawCount](#), [m_ib0](#), and [m_vao](#).

6.22.5 Field Documentation

6.22.5.1 indices

```
std::vector<uint32_t> Mesh::indices [private]
```

Definition at line 44 of file [Mesh.h](#).

Referenced by [getIndices\(\)](#), and [Mesh\(\)](#).

6.22.5.2 m_drawCount

```
uint32_t Mesh::m_drawCount [private]
```

Definition at line 42 of file [Mesh.h](#).

Referenced by [Mesh\(\)](#), and [render\(\)](#).

6.22.5.3 m_ib0

```
GLuint Mesh::m_ib0 [private]
```

Definition at line 38 of file [Mesh.h](#).

Referenced by [Mesh\(\)](#), [render\(\)](#), and [~Mesh\(\)](#).

6.22.5.4 m_vab

```
GLuint Mesh::m_vab[NUM_BUFFERS] [private]
```

Definition at line 40 of file [Mesh.h](#).

Referenced by [Mesh\(\)](#), and [~Mesh\(\)](#).

6.22.5.5 m_vao

```
GLuint Mesh::m_vao [private]
```

Definition at line 38 of file [Mesh.h](#).

Referenced by [Mesh\(\)](#), [render\(\)](#), and [~Mesh\(\)](#).

6.22.5.6 vertices

```
std::vector<Vertex> Mesh::vertices [private]
```

Definition at line 43 of file [Mesh.h](#).

Referenced by [getVertices\(\)](#), and [Mesh\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Mesh.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Mesh.cpp](#)

6.23 MirroredRepeatMode Class Reference

```
#include <MirroredRepeatMode.h>
```

Inheritance diagram for MirroredRepeatMode:

Collaboration diagram for MirroredRepeatMode:

Public Member Functions

- [MirroredRepeatMode \(\)](#)
- void [activate \(\)](#) override
- [~MirroredRepeatMode \(\)](#)

6.23.1 Detailed Description

Definition at line [4](#) of file [MirroredRepeatMode.h](#).

6.23.2 Constructor & Destructor Documentation

6.23.2.1 [MirroredRepeatMode\(\)](#)

```
MirroredRepeatMode::MirroredRepeatMode ( )
```

Definition at line [3](#) of file [MirroredRepeatMode.cpp](#).
00003 { }

6.23.2.2 [~MirroredRepeatMode\(\)](#)

```
MirroredRepeatMode::~MirroredRepeatMode ( )
```

Definition at line [11](#) of file [MirroredRepeatMode.cpp](#).
00011 { }

6.23.3 Member Function Documentation

6.23.3.1 activate()

```
void MirroredRepeatMode::activate ( ) [override], [virtual]
```

Implements [TextureWrappingMode](#).

Definition at line 5 of file [MirroredRepeatMode.cpp](#).

```
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
00009 }
```

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[MirroredRepeatMode.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[MirroredRepeatMode.cpp](#)

6.24 Model Class Reference

```
#include <Model.h>
```

Collaboration diagram for Model:

Public Member Functions

- [Model \(\)](#)
- void [load_model_in_ram](#) (const std::string &model_path)
- void [create_render_context \(\)](#)
- void [bind_ressources \(\)](#)
- void [unbind_resources \(\)](#)
- std::shared_ptr< [AABB](#) > [get_aabb \(\)](#)
- std::vector< [ObjMaterial](#) > [get_materials \(\) const](#)
- int [get_texture_count \(\) const](#)
- void [render \(\)](#)
- [~Model \(\)](#)

Private Attributes

- GLuint ssbo
- [ObjLoader](#) loader
- std::shared_ptr< [AABB](#) > aabb
- std::shared_ptr< [Mesh](#) > mesh
- std::vector< [Vertex](#) > vertices
- std::vector< unsigned int > indices
- std::vector< std::shared_ptr< [Texture](#) > > texture_list
- std::vector< [ObjMaterial](#) > materials
- std::vector< glm::vec4 > materialIndex
- std::vector< std::string > textures

6.24.1 Detailed Description

Definition at line 16 of file [Model.h](#).

6.24.2 Constructor & Destructor Documentation

6.24.2.1 Model()

```
Model::Model ( )
```

Definition at line 3 of file [Model.cpp](#).

```
00003 : aabb(std::make_shared<AABB>()) { }
```

6.24.2.2 ~Model()

```
Model::~Model ( )
```

Definition at line 63 of file [Model.cpp](#).

```
00064 {
00065     // unbind material index buffer
00066 }
```

6.24.3 Member Function Documentation

6.24.3.1 bind_ressources()

```
void Model::bind_ressources ( )
```

Definition at line 45 of file [Model.cpp](#).

```
00046 {
00047     glBindBufferBase(GL_SHADER_STORAGE_BUFFER, STORAGE_BUFFER_MATERIAL_ID_BINDING, ssbo);
00048     for (int i = 0; i < static_cast<int>(texture_list.size()); i++) {
00049         texture_list[i]->use_texture(i + MODEL_TEXTURES_SLOT);
00050     }
00051 }
```

References [MODEL_TEXTURES_SLOT](#), [ssbo](#), [STORAGE_BUFFER_MATERIAL_ID_BINDING](#), and [texture_list](#).

6.24.3.2 create_render_context()

```
void Model::create_render_context ( )
```

Definition at line 20 of file [Model.cpp](#).

```
00021 {
00022     texture_list.resize(textures.size());
00023
00024     for (uint32_t i = 0; i < static_cast<uint32_t>(textures.size()); i++) {
00025
00026         texture_list[i] = std::make_shared<Texture>(textures[i].c_str(), std::make_shared<RepeatMode>());
00027
00028         if (!texture_list[i]->load_SRGB_texture_without_alpha_channel()) {
00029             printf("Failed to load texture at: %s\n", textures[i].c_str());
00030             texture_list[i].reset();
00031         }
00032     }
00033
00034     mesh = std::make_shared<Mesh>(vertices, indices);
00035
00036     // https://www.khronos.org/opengl/wiki/Shader\_Storage\_Buffer\_Object
00037     glGenBuffers(1, &ssbo);
00038     glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);
00039     glBufferData(GL_SHADER_STORAGE_BUFFER,
00040         materialIndex.size() * sizeof(glm::vec4),
00041         materialIndex.data(),
00042         GL_STREAM_READ); //sizeof(data) only works for statically sized C/C++ arrays.
00043 }
```

References [indices](#), [materialIndex](#), [mesh](#), [ssbo](#), [texture_list](#), [textures](#), and [vertices](#).

6.24.3.3 get_aabb()

```
std::shared_ptr< AABB > Model::get_aabb ( )
```

Definition at line 5 of file [Model.cpp](#).

```
00005 { return aabb; }
```

References [aabb](#).

6.24.3.4 get_materials()

```
std::vector< ObjMaterial > Model::get_materials ( ) const
```

Definition at line 7 of file [Model.cpp](#).

```
00007 { return materials; }
```

References [materials](#).

6.24.3.5 get_texture_count()

```
int Model::get_texture_count ( ) const
```

Definition at line 9 of file [Model.cpp](#).

```
00009 { return static_cast<uint32_t>(texture_list.size()); }
```

References [texture_list](#).

6.24.3.6 load_model_in_ram()

```
void Model::load_model_in_ram (
    const std::string & model_path )
```

Definition at line 11 of file [Model.cpp](#).

```
00012 {
00013
00014     loader = ObjLoader();
00015     loader.load(model_path, vertices, indices, textures, materials, materialIndex);
00016 }
```

References [indices](#), [ObjLoader::load\(\)](#), [loader](#), [materialIndex](#), [materials](#), [textures](#), and [vertices](#).

Here is the call graph for this function:

6.24.3.7 render()

```
void Model::render ( )
```

Definition at line 61 of file [Model.cpp](#).

```
00061 { mesh->render(); }
```

References [mesh](#).

6.24.3.8 unbind_resources()

```
void Model::unbind_resources ( )
```

Definition at line 53 of file [Model.cpp](#).

```
00054 {
00055     glBindBuffer(GL_SHADER_STORAGE_BUFFER, 0);
00056     for (int i = 0; i < static_cast<int>(texture_list.size()); i++) {
00057         texture_list[i]->unbind_texture(i + MODEL_TEXTURES_SLOT);
00058     }
00059 }
```

References [MODEL_TEXTURES_SLOT](#), and [texture_list](#).

6.24.4 Field Documentation

6.24.4.1 aabb

```
std::shared_ptr<AABB> Model::aabb [private]
```

Definition at line 42 of file [Model.h](#).

Referenced by [get_aabb\(\)](#).

6.24.4.2 indices

```
std::vector<unsigned int> Model::indices [private]
```

Definition at line 46 of file [Model.h](#).

Referenced by [create_render_context\(\)](#), and [load_model_in_ram\(\)](#).

6.24.4.3 loader

```
ObjLoader Model::loader [private]
```

Definition at line 40 of file [Model.h](#).

Referenced by [load_model_in_ram\(\)](#).

6.24.4.4 materialIndex

```
std::vector<glm::vec4> Model::materialIndex [private]
```

Definition at line 49 of file [Model.h](#).

Referenced by [create_render_context\(\)](#), and [load_model_in_ram\(\)](#).

6.24.4.5 materials

```
std::vector<ObjMaterial> Model::materials [private]
```

Definition at line 48 of file [Model.h](#).

Referenced by [get_materials\(\)](#), and [load_model_in_ram\(\)](#).

6.24.4.6 mesh

```
std::shared_ptr<Mesh> Model::mesh [private]
```

Definition at line 44 of file [Model.h](#).

Referenced by [create_render_context\(\)](#), and [render\(\)](#).

6.24.4.7 ssbo

```
GLuint Model::ssbo [private]
```

Definition at line 38 of file [Model.h](#).

Referenced by [bind_ressources\(\)](#), and [create_render_context\(\)](#).

6.24.4.8 texture_list

```
std::vector<std::shared_ptr<Texture>> Model::texture_list [private]
```

Definition at line 47 of file [Model.h](#).

Referenced by [bind_ressources\(\)](#), [create_render_context\(\)](#), [get_texture_count\(\)](#), and [unbind_resources\(\)](#).

6.24.4.9 textures

```
std::vector<std::string> Model::textures [private]
```

Definition at line 50 of file [Model.h](#).

Referenced by [create_render_context\(\)](#), and [load_model_in_ram\(\)](#).

6.24.4.10 vertices

```
std::vector<Vertex> Model::vertices [private]
```

Definition at line 45 of file [Model.h](#).

Referenced by [create_render_context\(\)](#), and [load_model_in_ram\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Model.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Model.cpp](#)

6.25 Noise Class Reference

```
#include <Noise.h>
```

Collaboration diagram for Noise:

Public Member Functions

- `Noise ()`
- `void create_res128_noise ()`
- `void create_res32_noise ()`
- `void read_res128_noise ()`
- `void read_res32_noise ()`
- `void update ()`
- `void set_num_cells (GLuint num_cells_per_axis, GLuint index)`
- `~Noise ()`

Private Member Functions

- `void create_shader_programs ()`
- `void generate_cells (GLuint num_cells_per_axis, GLuint cell_index)`
 @ num_cells_per_axis: current voxel grid dimension @ cell_index: index into global array for all voxel grids
- `void generate_textures ()`
- `void generate_num_cells_textures ()`
- `void generate_res128_noise_texture ()`
- `void generate_res32_noise_texture ()`
- `void delete_textures ()`
- `void print_comp_shader_capabilities ()`

Private Attributes

- `GLuint texture_1_id`
- `GLuint texture_dim_1`
- `std::shared_ptr< ComputeShaderProgram > texture_1_shader_program`
- `GLuint texture_2_id`
- `GLuint texture_dim_2`
- `std::shared_ptr< ComputeShaderProgram > texture_2_shader_program`
- `GLuint cell_ids [NUM_CELL_POSITIONS]`
- `GLuint num_cells_per_axis [NUM_CELL_POSITIONS]`
- `std::array< std::vector< GLfloat >, NUM_CELL_POSITIONS > cell_data`

6.25.1 Detailed Description

Definition at line 15 of file [Noise.h](#).

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Noise()

Noise::Noise ()

Definition at line 5 of file [Noise.cpp](#).

```
00005      :
00006
00007     texture_dim_1(128), texture_dim_2(32)
00008
00009 {
00010
00011     create_shader_programs();
00012
00013 // we need 3d-voxel grids with different sizes for
00014 // creating different worley frequencies
00015     for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00016
00017     num_cells_per_axis[i] = static_cast<GLuint>(pow(2, i + 1));
00018     generate_cells(num_cells_per_axis[i], i);
00019   }
00020
00021     generate_textures();
00022 }
```

References [create_shader_programs\(\)](#), [generate_cells\(\)](#), [generate_textures\(\)](#), [NUM_CELL_POSITIONS](#), and [num_cells_per_axis](#).

Here is the call graph for this function:

6.25.2.2 ~Noise()

Noise::~Noise ()

Definition at line 287 of file [Noise.cpp](#).

```
00287 { delete_textures(); }
```

References [delete_textures\(\)](#).

Here is the call graph for this function:

6.25.3 Member Function Documentation

6.25.3.1 create_res128_noise()

void Noise::create_res128_noise ()

Definition at line 197 of file [Noise.cpp](#).

```
00198 {
00199
00200     texture_1_shader_program->use_shader_program();
00201
00202     texture_1_shader_program->setUniformInt(NOISE_128D_IMAGE_SLOT, "noise");
00203
00204     std::stringstream ss;
00205
00206     for (uint32_t i = 0; i < NUM_CELL_POSITIONS; i++) {
00207
00208         ss << "cell_positions[" << i << "]";
00209         texture_1_shader_program->setUniformInt(NOISE_CELL_POSITIONS_SLOT + i, ss.str());
00210         ss.clear();
00211         ss.str(std::string());
00212 }
```

```

00213     ss << "num_cells[" << i << "]";
00214     texture_1_shader_program->setUniformInt(num_cells_per_axis[i], ss.str());
00215     ss.clear();
00216     ss.str(std::string());
00217
00218     glActiveTexture(GL_TEXTURE0 + NOISE_CELL_POSITIONS_SLOT + i);
00219     glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00220 }
00221
00222 glDispatchCompute((GLuint)texture_dim_1, (GLuint)texture_dim_1, (GLuint)texture_dim_1);
00223
00224 // make sure writing to image has finished before read
00225 glMemoryBarrier(GL_SHADER_IMAGE_ACCESS_BARRIER_BIT);
00226 //glMemoryBarrier(GL_ALL_BARRIER_BITS);
00227
00228 glBindTexture(GL_TEXTURE_3D, 0);
00229 }
```

References `cell_ids`, `NOISE_128D_IMAGE_SLOT`, `NOISE_CELL_POSITIONS_SLOT`, `NUM_CELL_POSITIONS`, `num_cells_per_axis`, `texture_1_shader_program`, and `texture_dim_1`.

Referenced by [update\(\)](#).

Here is the caller graph for this function:

6.25.3.2 create_res32_noise()

```
void Noise::create_res32_noise ( )
```

Definition at line 231 of file `Noise.cpp`.

```

00232 {
00233     texture_2_shader_program->use_shader_program();
00235
00236     texture_2_shader_program->setUniformInt(NOISE_32D_IMAGE_SLOT, "noise");
00237
00238     std::stringstream ss;
00239
00240     for (uint32_t i = 0; i < NUM_CELL_POSITIONS; i++) {
00241
00242         ss << "cell_positions[" << i << "]";
00243         texture_2_shader_program->setUniformInt(NOISE_CELL_POSITIONS_SLOT + i, ss.str());
00244         ss.clear();
00245         ss.str(std::string());
00246
00247         ss << "num_cells[" << i << "]";
00248         texture_2_shader_program->setUniformInt(num_cells_per_axis[i], ss.str());
00249         ss.clear();
00250         ss.str(std::string());
00251
00252         glActiveTexture(GL_TEXTURE0 + NOISE_CELL_POSITIONS_SLOT + i);
00253         glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00254     }
00255
00256     glDispatchCompute((GLuint)texture_dim_2, (GLuint)texture_dim_2, (GLuint)texture_dim_2);
00257
00258 // make sure writing to image has finished before read
00259 glMemoryBarrier(GL_SHADER_IMAGE_ACCESS_BARRIER_BIT);
00260 //glMemoryBarrier(GL_ALL_BARRIER_BITS);
00261 glBindTexture(GL_TEXTURE_3D, 0);
00262 }
```

References `cell_ids`, `NOISE_32D_IMAGE_SLOT`, `NOISE_CELL_POSITIONS_SLOT`, `NUM_CELL_POSITIONS`, `num_cells_per_axis`, `texture_2_shader_program`, and `texture_dim_2`.

Referenced by [update\(\)](#).

Here is the caller graph for this function:

6.25.3.3 create_shader_programs()

```
void Noise::create_shader_programs ( ) [private]
```

Definition at line 24 of file [Noise.cpp](#).

```
00025 {
00026     texture_1_shader_program = std::make_shared<ComputeShaderProgram>();
00027     texture_2_shader_program = std::make_shared<ComputeShaderProgram>();
00028
00029     texture_1_shader_program->create_computer_shader_program_from_file("clouds/noise_texture_128_res.comp");
00030     texture_2_shader_program->create_computer_shader_program_from_file("clouds/noise_texture_32_res.comp");
00031 }
```

References [texture_1_shader_program](#), and [texture_2_shader_program](#).

Referenced by [Noise\(\)](#).

Here is the caller graph for this function:

6.25.3.4 delete_textures()

```
void Noise::delete_textures ( ) [private]
```

Definition at line 278 of file [Noise.cpp](#).

```
00279 {
00280
00281     glDeleteTextures(1, &texture_1_id);
00282     glDeleteTextures(1, &texture_2_id);
00283
00284     glDeleteTextures(NUM_CELL_POSITIONS, cell_ids);
00285 }
```

References [cell_ids](#), [NUM_CELL_POSITIONS](#), [texture_1_id](#), and [texture_2_id](#).

Referenced by [update\(\)](#), and [~Noise\(\)](#).

Here is the caller graph for this function:

6.25.3.5 generate_cells()

```
void Noise::generate_cells (
    GLuint num_cells_per_axis,
    GLuint cell_index ) [private]
```

@ num_cells_per_axis: current voxel grid dimension @ cell_index: index into global array for all voxel grids

needed for generating different worley frequencies later on

Definition at line 150 of file [Noise.cpp](#).

```
00151 {
00152
00153     cell_data[cell_index].reserve(num_cells_per_axis * num_cells_per_axis * num_cells_per_axis * 4);
00154
00155     // guess which birthday this is ;
00156     std::mt19937_64 gen64(25121995);
00157     std::uniform_real_distribution<float> dis(0, 1);
00158
00159     //depth
00160     for (int i = 0; i < static_cast<int>(num_cells_per_axis); i++) {
00161         //height
00162         for (int k = 0; k < static_cast<int>(num_cells_per_axis); k++) {
00163             //width
00164             for (int m = 0; m < static_cast<int>(num_cells_per_axis); m++) {
```

```

00165     // from: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexImage3D.xhtml
00166     // "The first element corresponds to the lower left corner of the texture image.
00167     // Subsequent elements progress left-to-right through the remaining texels in the lowest row
00168     // of the texture image, and then in successively higher rows of the texture image.
00169     // The final element corresponds to the upper right corner of the texture image."
00170
00171     const GLfloat random_offset[3] = { dis(gen64), dis(gen64), dis(gen64) };
00172
00173     GLfloat position[3] = { (m + random_offset[0]), (k + random_offset[1]), (i + random_offset[2]) }
00174   };
00175
00176   cell_data[cell_index].push_back(position[0]);
00177   cell_data[cell_index].push_back(position[1]);
00178   cell_data[cell_index].push_back(position[2]);
00179   cell_data[cell_index].push_back(1.0f);
00180
00181   // i leave this more c-style approach for my further me
00182   // to clarify things :
00183   //GLuint index = (i + num_cells_per_axis * (k + m * num_cells_per_axis)) * 4;
00184   /*GLfloat position[3] = {
00185     (i + random_offset[0]),
00186     (k +
00187     random_offset[1]),
00188     (m +
00189     random_offset[2]));*/
00190
00191   /*cell_data[cell_index][index] = position[0];
00192   cell_data[cell_index][index + 1] = position[1];
00193   cell_data[cell_index][index + 2] = position[2];
00194   cell_data[cell_index][index + 3] = 1.0f; */
00195 }
```

References [cell_data](#), and [num_cells_per_axis](#).

Referenced by [Noise\(\)](#), and [update\(\)](#).

Here is the caller graph for this function:

6.25.3.6 generate_num_cells_textures()

```
void Noise::generate_num_cells_textures ( ) [private]
```

Definition at line 41 of file [Noise.cpp](#).

```

00042 {
00043   glGenTextures(NUM_CELL_POSITIONS, cell_ids);
00044
00045   for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00046
00047     glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00048
00049     glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, num_cells_per_axis[i], num_cells_per_axis[i],
00050     num_cells_per_axis[i], 0, GL_RGBA, GL_FLOAT, cell_data[i].data());
00051
00052     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00053     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00054     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00055     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00056     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00057
00058   }
00059 }
```

References [cell_data](#), [cell_ids](#), [NUM_CELL_POSITIONS](#), and [num_cells_per_axis](#).

Referenced by [generate_textures\(\)](#).

Here is the caller graph for this function:

6.25.3.7 generate_res128_noise_texture()

```
void Noise::generate_res128_noise_texture ( ) [private]
```

Definition at line 61 of file [Noise.cpp](#).

```
00062 {
00063     glGenTextures(1, &texture_1_id);
00064
00065     glBindTexture(GL_TEXTURE_3D, texture_1_id);
00066
00067     glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, texture_dim_1, texture_dim_1, texture_dim_1, 0, GL_RGBA,
00068     GL_FLOAT, NULL);
00069
00070     glBindImageTexture(NOISE_128D_IMAGE_SLOT, texture_1_id, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA32F);
00071
00072     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00073     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00074     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
00075     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00076     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00077
00078     glBindTexture(GL_TEXTURE_3D, 0);
00079 }
```

References [NOISE_128D_IMAGE_SLOT](#), [NOISE_128D_TEXTURES_SLOT](#), [texture_1_id](#), and [texture_dim_1](#).

Referenced by [generate_textures\(\)](#).

Here is the caller graph for this function:

6.25.3.8 generate_res32_noise_texture()

```
void Noise::generate_res32_noise_texture ( ) [private]
```

Definition at line 81 of file [Noise.cpp](#).

```
00082 {
00083
00084     glGenTextures(1, &texture_2_id);
00085
00086     glBindTexture(GL_TEXTURE_3D, texture_2_id);
00087
00088     glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, texture_dim_2, texture_dim_2, texture_dim_2, 0, GL_RGBA,
00089     GL_FLOAT, NULL);
00090
00091     glBindImageTexture(NOISE_32D_IMAGE_SLOT, texture_2_id, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA32F);
00092
00093     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00094     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00095     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
00096     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00097     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00098
00099     glBindTexture(GL_TEXTURE_3D, 0);
00100 }
```

References [NOISE_32D_IMAGE_SLOT](#), [NOISE_32D_TEXTURES_SLOT](#), [texture_2_id](#), and [texture_dim_2](#).

Referenced by [generate_textures\(\)](#).

Here is the caller graph for this function:

6.25.3.9 generate_textures()

```
void Noise::generate_textures ( ) [private]
```

Definition at line 33 of file [Noise.cpp](#).

```
00034 {
00035
00036     generate_num_cells_textures ();
00037     generate_res128_noise_texture ();
00038     generate_res32_noise_texture ();
00039 }
```

References [generate_num_cells_textures\(\)](#), [generate_res128_noise_texture\(\)](#), and [generate_res32_noise_texture\(\)](#).

Referenced by [Noise\(\)](#), and [update\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.25.3.10 print_comp_shader_capabilities()

```
void Noise::print_comp_shader_capabilities ( ) [private]
```

Definition at line 102 of file [Noise.cpp](#).

```
00103 {
00104
00105     int work_grp_cnt[3];
00106
00107     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 0, &work_grp_cnt[0]);
00108     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 1, &work_grp_cnt[1]);
00109     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 2, &work_grp_cnt[2]);
00110
00111     printf("max global (total) work group counts x:%i y:%i z:%i\n", work_grp_cnt[0], work_grp_cnt[1],
00112           work_grp_cnt[2]);
00113
00114     int work_grp_size[3];
00115
00116     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 0, &work_grp_size[0]);
00117     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 1, &work_grp_size[1]);
00118     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 2, &work_grp_size[2]);
00119
00120     printf("max local (in one shader) work group sizes x:%i y:%i z:%i\n", work_grp_size[0],
           work_grp_size[1], work_grp_size[2]);
```

6.25.3.11 read_res128_noise()

```
void Noise::read_res128_noise ( )
```

Definition at line 264 of file [Noise.cpp](#).

```
00265 {
00266     GLuint texture_index = GL_TEXTURE0 + NOISE_128D_TEXTURES_SLOT;
00267     glBindTexture(GL_TEXTURE_3D, texture_1_id);
00268 }
00269 }
```

References [NOISE_128D_TEXTURES_SLOT](#), and [texture_1_id](#).

6.25.3.12 `read_res32_noise()`

```
void Noise::read_res32_noise ( )
```

Definition at line 271 of file [Noise.cpp](#).

```
00272 {
00273     GLuint texture_index = GL_TEXTURE0 + NOISE_32D_TEXTURES_SLOT;
00274     glBindTexture(GL_TEXTURE_3D, texture_2_id);
00275     glBindTexture(GL_TEXTURE_3D, texture_2_id);
00276 }
```

References [NOISE_32D_TEXTURES_SLOT](#), and [texture_2_id](#).

6.25.3.13 `set_num_cells()`

```
void Noise::set_num_cells (
    GLuint num_cells_per_axis,
    GLuint index )
```

Definition at line 141 of file [Noise.cpp](#).

```
00141 { this->num_cells_per_axis[index] = num_cells_per_axis; }
```

References [num_cells_per_axis](#).

6.25.3.14 `update()`

```
void Noise::update ( )
```

Definition at line 122 of file [Noise.cpp](#).

```
00123 {
00124
00125     delete_textures();
00126
00127     texture_1_shader_program->reload();
00128     texture_2_shader_program->reload();
00129
00130     for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00131
00132         generate_cells(num_cells_per_axis[i], i);
00133     }
00134
00135     generate_textures();
00136
00137     create_res128_noise();
00138     create_res32_noise();
00139 }
```

References [create_res128_noise\(\)](#), [create_res32_noise\(\)](#), [delete_textures\(\)](#), [generate_cells\(\)](#), [generate_textures\(\)](#), [NUM_CELL_POSITIONS](#), [num_cells_per_axis](#), [texture_1_shader_program](#), and [texture_2_shader_program](#).

Here is the call graph for this function:

6.25.4 Field Documentation

6.25.4.1 cell_data

```
std::array<std::vector<GLfloat>, NUM_CELL_POSITIONS> Noise::cell_data [private]
```

Definition at line 56 of file [Noise.h](#).

Referenced by [generate_cells\(\)](#), and [generate_num_cells_textures\(\)](#).

6.25.4.2 cell_ids

```
GLuint Noise::cell_ids[NUM_CELL_POSITIONS] [private]
```

Definition at line 54 of file [Noise.h](#).

Referenced by [create_res128_noise\(\)](#), [create_res32_noise\(\)](#), [delete_textures\(\)](#), and [generate_num_cells_textures\(\)](#).

6.25.4.3 num_cells_per_axis

```
GLuint Noise::num_cells_per_axis[NUM_CELL_POSITIONS] [private]
```

Definition at line 55 of file [Noise.h](#).

Referenced by [create_res128_noise\(\)](#), [create_res32_noise\(\)](#), [generate_cells\(\)](#), [generate_num_cells_textures\(\)](#), [Noise\(\)](#), [set_num_cells\(\)](#), and [update\(\)](#).

6.25.4.4 texture_1_id

```
GLuint Noise::texture_1_id [private]
```

Definition at line 45 of file [Noise.h](#).

Referenced by [delete_textures\(\)](#), [generate_res128_noise_texture\(\)](#), and [read_res128_noise\(\)](#).

6.25.4.5 texture_1_shader_program

```
std::shared_ptr<ComputeShaderProgram> Noise::texture_1_shader_program [private]
```

Definition at line 47 of file [Noise.h](#).

Referenced by [create_res128_noise\(\)](#), [create_shader_programs\(\)](#), and [update\(\)](#).

6.25.4.6 texture_2_id

```
GLuint Noise::texture_2_id [private]
```

Definition at line 50 of file [Noise.h](#).

Referenced by [delete_textures\(\)](#), [generate_res32_noise_texture\(\)](#), and [read_res32_noise\(\)](#).

6.25.4.7 texture_2_shader_program

```
std::shared_ptr<ComputeShaderProgram> Noise::texture_2_shader_program [private]
```

Definition at line 52 of file [Noise.h](#).

Referenced by [create_res32_noise\(\)](#), [create_shader_programs\(\)](#), and [update\(\)](#).

6.25.4.8 texture_dim_1

```
GLuint Noise::texture_dim_1 [private]
```

Definition at line 46 of file [Noise.h](#).

Referenced by [create_res128_noise\(\)](#), and [generate_res128_noise_texture\(\)](#).

6.25.4.9 texture_dim_2

```
GLuint Noise::texture_dim_2 [private]
```

Definition at line 51 of file [Noise.h](#).

Referenced by [create_res32_noise\(\)](#), and [generate_res32_noise_texture\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/[Noise.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/[Noise.cpp](#)

6.26 ObjLoader Class Reference

```
#include <ObjLoader.h>
```

Collaboration diagram for ObjLoader:

Public Member Functions

- `ObjLoader ()`
- `void load (std::string modelFile, std::vector< Vertex > &vertices, std::vector< unsigned int > &indices, std::vector< std::string > &texture_list, std::vector< ObjMaterial > &materials, std::vector< glm::vec4 > &materialIndex)`
- `~ObjLoader ()`

Static Private Member Functions

- `static std::string get_base_dir (const std::string &filepath)`

Private Attributes

- `GLfloat minX = std::numeric_limits<float>::max()`
- `GLfloat maxX = std::numeric_limits<float>::min()`
- `GLfloat minY = std::numeric_limits<float>::max()`
- `GLfloat maxY = std::numeric_limits<float>::min()`
- `GLfloat minZ = std::numeric_limits<float>::max()`
- `GLfloat maxZ = std::numeric_limits<float>::min()`

6.26.1 Detailed Description

Definition at line 11 of file [ObjLoader.h](#).

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ObjLoader()

```
ObjLoader::ObjLoader ( )
```

Definition at line 6 of file [ObjLoader.cpp](#).
00006 { }

6.26.2.2 ~ObjLoader()

```
ObjLoader::~ObjLoader ( )
```

Definition at line 173 of file [ObjLoader.cpp](#).
00173 { }

6.26.3 Member Function Documentation

6.26.3.1 get_base_dir()

```
static std::string ObjLoader::get_base_dir (
    const std::string & filepath ) [inline], [static], [private]
```

Definition at line 28 of file [ObjLoader.h](#).

```
00029 {
00030
00031     if (filepath.find_last_of("/\\") != std::string::npos) return filepath.substr(0,
00032         filepath.find_last_of("/\\"));
00033     return "";
00034 }
```

Referenced by [load\(\)](#).

Here is the caller graph for this function:

6.26.3.2 load()

```
void ObjLoader::load (
    std::string modelFile,
    std::vector< Vertex > & vertices,
    std::vector< unsigned int > & indices,
    std::vector< std::string > & texture_list,
    std::vector< ObjMaterial > & materials,
    std::vector< glm::vec4 > & materialIndex )
```

Definition at line 8 of file [ObjLoader.cpp](#).

```
00010 {
00011
00012     tinyobj::ObjReaderConfig reader_config;
00013     tinyobj::ObjReader reader;
00014
00015     if (!reader.ParseFromFile(modelFile, reader_config)) {
00016         if (!reader.Error().empty()) {
00017             std::cerr << "TinyObjReader: " << reader.Error();
00018         }
00019         exit(EXIT_FAILURE);
00020     }
00021
00022     if (!reader.Warning().empty()) {
00023         std::cout << "TinyObjReader: " << reader.Warning();
00024     }
00025
00026     auto& tol_materials = reader.GetMaterials();
00027 //texture_list.reserve(tol_materials.size());
00028
00029     if (static_cast<GLuint>(tol_materials.size() > MAX_MATERIALS)) std::runtime_error("ObjLoader: We try
00030     to load more materials than MAX_MATERIALS is defined!");
00031
00032 // texture at position 0 is plain texture to handle non existing materials
00033     int texture_id = 1;
00034
00035     std::stringstream texture_base_dir;
00036     texture_base_dir << CMAKELISTS_DIR << "/Resources/Textures/plain.png";
00037     texture_list.push_back(texture_base_dir.str());
00038
00039 // we now iterate over all materials to get diffuse textures
00040     for (size_t i = 0; i < tol_materials.size(); i++) {
00041
00042         const tinyobj::material_t* mp = &tol_materials[i];
00043         ObjMaterial material;
00044         material.ambient = glm::vec3(mp->ambient[0], mp->ambient[1], mp->ambient[2]);
00045         material.diffuse = glm::vec3(mp->diffuse[0], mp->diffuse[1], mp->diffuse[2]);
00046         material.specular = glm::vec3(mp->specular[0], mp->specular[1], mp->specular[2]);
00047         material.emission = glm::vec3(mp->emission[0], mp->emission[1], mp->emission[2]);
00048         material.transmittance = glm::vec3(mp->transmittance[0], mp->transmittance[1],
00049                                         mp->transmittance[2]);
00050         material.dissolve = mp->dissolve;
00051         material.ior = mp->ior;
00052         material.shininess = mp->shininess;
00053         material.illum = mp->illum;
```

```

00053     if (mp->diffuse_texname.length() > 0) {
00054
00055         std::string relative_texture_filename = mp->diffuse_texname;
00056         std::string texture_filename = get_base_dir(modelFile) + "/" + relative_texture_filename;
00057
00058         texture_list.push_back(texture_filename);
00059
00060         material.textureID = texture_id;
00061         texture_id++;
00062
00063     } else {
00064
00065         // this means no texture was assigned; we catch it here and assign our plain texture
00066         // at position 0
00067         material.textureID = 0;
00068     }
00069
00070     materials.push_back(material);
00071 }
00072
00073 // for the case no .mtl file is given place some random standard material ...
00074 if (tol_materials.empty()) {
00075     materials.emplace_back(ObjMaterial());
00076 }
00077
00078 auto& attrib = reader.GetAttrib();
00079 auto& shapes = reader.GetShapes();
00080
00081 std::unordered_map<Vertex, uint32_t> vertices_map{};
00082
00083 // Loop over shapes
00084 for (size_t s = 0; s < shapes.size(); s++) {
00085     // prepare for enlargement
00086     vertices.reserve(shapes[s].mesh.indices.size() + vertices.size());
00087     indices.reserve(shapes[s].mesh.indices.size() + indices.size());
00088
00089     // Loop over faces(polygon)
00090     size_t index_offset = 0;
00091     for (size_t f = 0; f < shapes[s].mesh.num_face_vertices.size(); f++) {
00092
00093         size_t fv = size_t(shapes[s].mesh.num_face_vertices[f]);
00094
00095         // Loop over vertices in the face.
00096         for (size_t v = 0; v < fv; v++) {
00097
00098             // access to vertex
00099             tinyobj::index_t idx = shapes[s].mesh.indices[index_offset + v];
00100             tinyobj::real_t vx = attrib.vertices[3 * size_t(idx.vertex_index) + 0];
00101             tinyobj::real_t vy = attrib.vertices[3 * size_t(idx.vertex_index) + 1];
00102             tinyobj::real_t vz = attrib.vertices[3 * size_t(idx.vertex_index) + 2];
00103             glm::vec3 pos = { vx, vy, vz };
00104
00105             minX = std::min(minX, pos.x);
00106             maxX = std::max(maxX, pos.x);
00107             minY = std::min(minY, pos.y);
00108             maxY = std::max(maxY, pos.y);
00109             minZ = std::min(minZ, pos.z);
00110             maxZ = std::max(maxZ, pos.z);
00111
00112             glm::vec3 normals(0.0f);
00113             // Check if 'normal_index' is zero or positive. negative = no normal data
00114             if (idx.normal_index >= 0 && !attrib.normals.empty()) {
00115                 tinyobj::real_t nx = attrib.normals[3 * size_t(idx.normal_index) + 0];
00116                 tinyobj::real_t ny = attrib.normals[3 * size_t(idx.normal_index) + 1];
00117                 tinyobj::real_t nz = attrib.normals[3 * size_t(idx.normal_index) + 2];
00118                 normals = glm::vec3(nx, ny, nz);
00119             }
00120
00121             glm::vec3 color(-1.f);
00122             if (!attrib.colors.empty()) {
00123                 tinyobj::real_t red = attrib.colors[3 * size_t(idx.vertex_index) + 0];
00124                 tinyobj::real_t green = attrib.colors[3 * size_t(idx.vertex_index) + 1];
00125                 tinyobj::real_t blue = attrib.colors[3 * size_t(idx.vertex_index) + 2];
00126                 color = glm::vec3(red, green, blue);
00127             }
00128
00129             glm::vec2 tex_coords(0.0f);
00130             // Check if 'texcoord_index' is zero or positive. negative = no texcoord data
00131             if (idx.texcoord_index >= 0 && !attrib.texcoords.empty()) {
00132                 tinyobj::real_t tx = attrib.texcoords[2 * size_t(idx.texcoord_index) + 0];
00133                 // flip y coordinate !!
00134                 tinyobj::real_t ty = 1.f - attrib.texcoords[2 * size_t(idx.texcoord_index) + 1];
00135                 tex_coords = glm::vec2(tx, ty);
00136             }
00137
00138             Vertex vert{ pos, normals, color, tex_coords };
00139

```

```

00140     if (vertices_map.count(vert) == 0) {
00141         vertices_map[vert] = static_cast<uint32_t>(vertices.size());
00142         vertices.push_back(vert);
00143     }
00145
00146     indices.push_back(vertices_map[vert]);
00147 }
00148
00149     index_offset += fv;
00150
00151 // per-face material; face usually is triangle
00152 // matToTex[shapes[s].mesh.material_ids[f]]
00153     materialIndex.push_back(glm::vec4(shapes[s].mesh.material_ids[f], 0.0f, 0.0f, 0.0f));
00154 }
00155 }
00156
00157 // precompute normals if no provided
00158 if (attrib.normals.empty()) {
00159
00160     for (size_t i = 0; i < indices.size(); i += 3) {
00161         Vertex& v0 = vertices[indices[i + 0]];
00162         Vertex& v1 = vertices[indices[i + 1]];
00163         Vertex& v2 = vertices[indices[i + 2]];
00164
00165         glm::vec3 n = glm::normalize(glm::cross((v1.position - v0.position),
00166                                             v0.position));
00166         v0.normal = n;
00167         v1.normal = n;
00168         v2.normal = n;
00169     }
00170 }
00171 }
```

References [ObjMaterial::ambient](#), [ObjMaterial::diffuse](#), [ObjMaterial::dissolve](#), [ObjMaterial::emission](#), [get_base_dir\(\)](#), [ObjMaterial::illum](#), [ObjMaterial::ior](#), [MAX_MATERIALS](#), [maxX](#), [maxY](#), [maxZ](#), [minX](#), [minY](#), [minZ](#), [Vertex::normal](#), [Vertex::position](#), [ObjMaterial::shininess](#), [ObjMaterial::specular](#), [ObjMaterial::textureID](#), and [ObjMaterial::transmittance](#).

Referenced by [Model::load_model_in_ram\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.26.4 Field Documentation

6.26.4.1 maxX

```
GLfloat ObjLoader::maxX = std::numeric_limits<float>::min() [private]
```

Definition at line 22 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

6.26.4.2 maxY

```
GLfloat ObjLoader::maxY = std::numeric_limits<float>::min() [private]
```

Definition at line 24 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

6.26.4.3 maxZ

```
GLfloat ObjLoader::maxZ = std::numeric_limits<float>::min() [private]
```

Definition at line 26 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

6.26.4.4 minX

```
GLfloat ObjLoader::minX = std::numeric_limits<float>::max() [private]
```

Definition at line 21 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

6.26.4.5 minY

```
GLfloat ObjLoader::minY = std::numeric_limits<float>::max() [private]
```

Definition at line 23 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

6.26.4.6 minZ

```
GLfloat ObjLoader::minZ = std::numeric_limits<float>::max() [private]
```

Definition at line 25 of file [ObjLoader.h](#).

Referenced by [load\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ObjLoader.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ObjLoader.cpp](#)

6.27 ObjMaterial Class Reference

```
#include <ObjMaterial.h>
```

Collaboration diagram for ObjMaterial:

Public Member Functions

- `ObjMaterial ()`
- `ObjMaterial (glm::vec3 ambient, glm::vec3 diffuse, glm::vec3 specular, glm::vec3 transmittance, glm::vec3 emission, float shininess, float ior, float dissolve, int illum, int textureID)`
- `glm::vec3 get_ambient () const`
- `glm::vec3 get_diffuse () const`
- `glm::vec3 get_specular () const`
- `glm::vec3 get_transmittance () const`
- `glm::vec3 get_emission () const`
- `float get_shininess () const`
- `float get_iор () const`
- `float get_dissolve () const`
- `int get_illum () const`
- `int get_textureID () const`
- `~ObjMaterial ()`

Data Fields

- `glm::vec3 ambient = glm::vec3(0.1f, 0.1f, 0.1f)`
- `glm::vec3 diffuse = glm::vec3(0.7f, 0.7f, 0.7f)`
- `glm::vec3 specular = glm::vec3(1.0f, 1.0f, 1.0f)`
- `glm::vec3 transmittance = glm::vec3(0.0f, 0.0f, 0.0f)`
- `glm::vec3 emission = glm::vec3(0.0f, 0.0f, 0.10)`
- `float shininess = 0.f`
- `float ior = 1.0f`
- `float dissolve = 1.f`
- `int illum = 0`
- `int textureID = -1`

6.27.1 Detailed Description

Definition at line 12 of file [ObjMaterial.h](#).

6.27.2 Constructor & Destructor Documentation

6.27.2.1 ObjMaterial() [1/2]

```
ObjMaterial::ObjMaterial ( )
```

Definition at line 3 of file [ObjMaterial.cpp](#).

```
00004 {
00005
00006     this->ambient = glm::vec3(0.1f, 0.1f, 0.1f);
00007     this->diffuse = glm::vec3(0.7f, 0.7f, 0.7f);
00008     this->specular = glm::vec3(1.0f, 1.0f, 1.0f);
00009     this->transmittance = glm::vec3(0.0f, 0.0f, 0.0f);
00010     this->emission = glm::vec3(0.0f, 0.0f, 0.10);
00011     this->shininess = 0.f;
00012     this->iор = 1.0f;
00013     this->dissolve = 1.f;
00014     this->illum = 0;
00015     this->textureID = -1;
00016 }
```

References `ambient`, `diffuse`, `dissolve`, `emission`, `illum`, `ior`, `shininess`, `specular`, `textureID`, and `transmittance`.

6.27.2.2 ObjMaterial() [2/2]

```
ObjMaterial::ObjMaterial (
    glm::vec3 ambient,
    glm::vec3 diffuse,
    glm::vec3 specular,
    glm::vec3 transmittance,
    glm::vec3 emission,
    float shininess,
    float ior,
    float dissolve,
    int illum,
    int textureID )
```

Definition at line 18 of file [ObjMaterial.cpp](#).

```
00020 {
00021     this->ambient = ambient;
00022     this->diffuse = diffuse;
00023     this->specular = specular;
00024     this->transmittance = transmittance;
00025     this->emission = emission;
00026     this->shininess = shininess;
00027     this->ior = ior;
00028     this->dissolve = dissolve;
00029     this->illum = illum;
00030     this->textureID = textureID;
00031 }
```

References [ambient](#), [diffuse](#), [dissolve](#), [emission](#), [illum](#), [ior](#), [shininess](#), [specular](#), [textureID](#), and [transmittance](#).

6.27.2.3 ~ObjMaterial()

```
ObjMaterial::~ObjMaterial ( )
```

Definition at line 33 of file [ObjMaterial.cpp](#).

```
00033 { }
```

6.27.3 Member Function Documentation

6.27.3.1 get_ambient()

```
glm::vec3 ObjMaterial::get_ambient ( ) const [inline]
```

Definition at line 19 of file [ObjMaterial.h](#).

```
00019 { return ambient; };
```

References [ambient](#).

6.27.3.2 get_diffuse()

```
glm::vec3 ObjMaterial::get_diffuse () const [inline]
```

Definition at line 20 of file [ObjMaterial.h](#).

```
00020 { return diffuse; };
```

References [diffuse](#).

6.27.3.3 get_dissolve()

```
float ObjMaterial::get_dissolve () const [inline]
```

Definition at line 27 of file [ObjMaterial.h](#).

```
00027 { return dissolve; };
```

References [dissolve](#).

6.27.3.4 get_emission()

```
glm::vec3 ObjMaterial::get_emission () const [inline]
```

Definition at line 23 of file [ObjMaterial.h](#).

```
00023 { return emission; };
```

References [emission](#).

6.27.3.5 get_illum()

```
int ObjMaterial::get_illum () const [inline]
```

Definition at line 29 of file [ObjMaterial.h](#).

```
00029 { return illum; };
```

References [illum](#).

6.27.3.6 get_ior()

```
float ObjMaterial::get_ior () const [inline]
```

Definition at line 26 of file [ObjMaterial.h](#).

```
00026 { return ior; };
```

References [ior](#).

6.27.3.7 get_shininess()

```
float ObjMaterial::get_shininess () const [inline]
```

Definition at line 25 of file [ObjMaterial.h](#).
00025 { **return** shininess; };

References [shininess](#).

6.27.3.8 get_specular()

```
glm::vec3 ObjMaterial::get_specular () const [inline]
```

Definition at line 21 of file [ObjMaterial.h](#).
00021 { **return** specular; };

References [specular](#).

6.27.3.9 get_textureID()

```
int ObjMaterial::get_textureID () const [inline]
```

Definition at line 30 of file [ObjMaterial.h](#).
00030 { **return** textureID; };

References [textureID](#).

6.27.3.10 get_transmittance()

```
glm::vec3 ObjMaterial::get_transmittance () const [inline]
```

Definition at line 22 of file [ObjMaterial.h](#).
00022 { **return** transmittance; };

References [transmittance](#).

6.27.4 Field Documentation

6.27.4.1 ambient

```
glm::vec3 ObjMaterial::ambient = glm::vec3(0.1f, 0.1f, 0.1f)
```

Definition at line 32 of file [ObjMaterial.h](#).

Referenced by [get_ambient\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.2 diffuse

```
glm::vec3 ObjMaterial::diffuse = glm::vec3(0.7f, 0.7f, 0.7f)
```

Definition at line 33 of file [ObjMaterial.h](#).

Referenced by [get_diffuse\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.3 dissolve

```
float ObjMaterial::dissolve = 1.f
```

Definition at line 39 of file [ObjMaterial.h](#).

Referenced by [get_dissolve\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.4 emission

```
glm::vec3 ObjMaterial::emission = glm::vec3(0.0f, 0.0f, 0.10)
```

Definition at line 36 of file [ObjMaterial.h](#).

Referenced by [get_emission\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.5 illum

```
int ObjMaterial::illum = 0
```

Definition at line 41 of file [ObjMaterial.h](#).

Referenced by [get_illum\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.6 ior

```
float ObjMaterial::ior = 1.0f
```

Definition at line 38 of file [ObjMaterial.h](#).

Referenced by [get_ior\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.7 shininess

```
float ObjMaterial::shininess = 0.f
```

Definition at line 37 of file [ObjMaterial.h](#).

Referenced by [get_shininess\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.8 specular

```
glm::vec3 ObjMaterial::specular = glm::vec3(1.0f, 1.0f, 1.0f)
```

Definition at line 34 of file [ObjMaterial.h](#).

Referenced by [get_specular\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.9 textureID

```
int ObjMaterial::textureID = -1
```

Definition at line 42 of file [ObjMaterial.h](#).

Referenced by [get_textureID\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

6.27.4.10 transmittance

```
glm::vec3 ObjMaterial::transmittance = glm::vec3(0.0f, 0.0f, 0.0f)
```

Definition at line 35 of file [ObjMaterial.h](#).

Referenced by [get_transmittance\(\)](#), [ObjLoader::load\(\)](#), and [ObjMaterial\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ObjMaterial.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ObjMaterial.cpp](#)

6.28 OmniDirShadowMap Class Reference

```
#include <OmniDirShadowMap.h>
```

Inheritance diagram for OmniDirShadowMap:

Collaboration diagram for OmniDirShadowMap:

Public Member Functions

- [OmniDirShadowMap \(\)](#)
- [bool init \(GLuint width, GLuint height\)](#)
- [void write \(\)](#)
- [void read \(GLenum texture_unit\)](#)
- [~OmniDirShadowMap \(\)](#)

Additional Inherited Members

6.28.1 Detailed Description

Definition at line [4](#) of file [OmniDirShadowMap.h](#).

6.28.2 Constructor & Destructor Documentation

6.28.2.1 OmniDirShadowMap()

```
OmniDirShadowMap::OmniDirShadowMap ( )
```

Definition at line [4](#) of file [OmniDirShadowMap.cpp](#).
00004 : [ShadowMap\(\) { }](#)

6.28.2.2 ~OmniDirShadowMap()

```
OmniDirShadowMap::~OmniDirShadowMap ( )
```

Definition at line [57](#) of file [OmniDirShadowMap.cpp](#).
00057 : { }

6.28.3 Member Function Documentation

6.28.3.1 init()

```
bool OmniDirShadowMap::init (
    GLuint width,
    GLuint height ) [virtual]
```

Reimplemented from [ShadowMap](#).

Definition at line 6 of file [OmniDirShadowMap.cpp](#).

```
00007 {
00008
00009     shadow_width = width;
00010     shadow_height = height;
00011
00012     glGenFramebuffers(1, &FBO);
00013
00014     glGenTextures(1, &shadow_map);
00015     glBindTexture(GL_TEXTURE_CUBE_MAP, shadow_map);
00016
00017     for (size_t i = 0; i < 6; i++) {
00018         // keep in mind that all following f.e. negative_x, positive_y,...etc. are reachable
00019         // by simply increment positive_x stepwise
00020         glTexImage2D((GLenum)(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i), 0, GL_DEPTH_COMPONENT, shadow_width,
00021                     shadow_height, 0, GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00022     }
00023
00024     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00025     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00026
00027     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00028     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00029     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00030
00031     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00032     glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, shadow_map, 0);
00033
00034     glDrawBuffer(GL_NONE);
00035     glReadBuffer(GL_NONE);
00036
00037     GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00038
00039     if (status != GL_FRAMEBUFFER_COMPLETE) {
00040         printf("Framebuffer error: %i\n", status);
00041         return false;
00042     }
00043
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045
00046     return true;
00047 }
```

References [ShadowMap::FBO](#), [ShadowMap::shadow_height](#), [ShadowMap::shadow_map](#), and [ShadowMap::shadow_width](#).

6.28.3.2 read()

```
void OmniDirShadowMap::read (
    GLenum texture_unit ) [virtual]
```

Reimplemented from [ShadowMap](#).

Definition at line 51 of file [OmniDirShadowMap.cpp](#).

```
00052 {
00053     glActiveTexture(GL_TEXTURE0 + texture_unit);
00054     glBindTexture(GL_TEXTURE_CUBE_MAP, shadow_map);
00055 }
```

References [ShadowMap::shadow_map](#).

6.28.3.3 write()

```
void OmniDirShadowMap::write ( ) [virtual]
```

Reimplemented from [ShadowMap](#).

Definition at line 49 of file [OmniDirShadowMap.cpp](#).

```
00049 { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }
```

References [ShadowMap::FBO](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniDirShadowMap.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniDirShadowMap.cpp](#)

6.29 OmniDirShadowShaderProgram Class Reference

```
#include <OmniDirShadowShaderProgram.h>
```

Inheritance diagram for OmniDirShadowShaderProgram:

Collaboration diagram for OmniDirShadowShaderProgram:

Public Member Functions

- [OmniDirShadowShaderProgram \(\)](#)
- void [reload \(\)](#)
- [~OmniDirShadowShaderProgram \(\)](#)

Additional Inherited Members

6.29.1 Detailed Description

Definition at line 4 of file [OmniDirShadowShaderProgram.h](#).

6.29.2 Constructor & Destructor Documentation

6.29.2.1 OmniDirShadowShaderProgram()

```
OmniDirShadowShaderProgram::OmniDirShadowShaderProgram ( )
```

Definition at line 3 of file [OmniDirShadowShaderProgram.cpp](#).

```
00003 { }
```

6.29.2.2 ~OmniDirShadowShaderProgram()

```
OmniDirShadowShaderProgram::~OmniDirShadowShaderProgram ( )
```

Definition at line 7 of file [OmniDirShadowShaderProgram.cpp](#).
00007 { }

6.29.3 Member Function Documentation

6.29.3.1 reload()

```
void OmniDirShadowShaderProgram::reload ( )
```

Definition at line 5 of file [OmniDirShadowShaderProgram.cpp](#).
00005 { [create_from_files](#)(this->vertex_location, this->geometry_location, this->fragment_location); }

References [ShaderProgram::create_from_files\(\)](#), [ShaderProgram::fragment_location](#), [ShaderProgram::geometry_location](#), and [ShaderProgram::vertex_location](#).

Here is the call graph for this function:

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniDirShadowShaderProgram.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniDirShadowShaderProgram.cpp](#)

6.30 OmniShadowMapPass Class Reference

```
#include <OmniShadowMapPass.h>
```

Inheritance diagram for OmniShadowMapPass:

Collaboration diagram for OmniShadowMapPass:

Public Member Functions

- [OmniShadowMapPass \(\)](#)
- void [execute](#) (std::shared_ptr<[PointLight](#)> p_light, std::shared_ptr<[Scene](#)> scene)
- void [set_game_object_uniforms](#) (glm::mat4 model, glm::mat4 normal_model)
- void [create_shader_program \(\)](#)
- [~OmniShadowMapPass \(\)](#)

Private Attributes

- std::shared_ptr<[OmniDirShadowShaderProgram](#)> [shader_program](#)

6.30.1 Detailed Description

Definition at line 10 of file [OmniShadowMapPass.h](#).

6.30.2 Constructor & Destructor Documentation

6.30.2.1 OmniShadowMapPass()

```
OmniShadowMapPass::OmniShadowMapPass ( )
```

Definition at line 4 of file [OmniShadowMapPass.cpp](#).
00004 { [create_shader_program\(\)](#); }

References [create_shader_program\(\)](#).

Here is the call graph for this function:

6.30.2.2 ~OmniShadowMapPass()

```
OmniShadowMapPass::~OmniShadowMapPass ( )
```

Definition at line 53 of file [OmniShadowMapPass.cpp](#).
00053 { }

6.30.3 Member Function Documentation

6.30.3.1 create_shader_program()

```
void OmniShadowMapPass::create_shader_program ( ) [virtual]
```

Implements [RenderPassSceneDependend](#).

Definition at line 46 of file [OmniShadowMapPass.cpp](#).

```
00047 {  
00048     shader\_program = std::make_shared<OmniDirShadowShaderProgram> (OmniDirShadowShaderProgram{});  
00049     shader\_program->create_from_files(  
00050         "rasterizer/shadows/omni_shadow_map.vert", "rasterizer/shadows/omni_shadow_map.geom",  
         "rasterizer/shadows/omni_shadow_map.frag");  
00051 }
```

References [shader_program](#).

Referenced by [OmniShadowMapPass\(\)](#).

Here is the caller graph for this function:

6.30.3.2 execute()

```
void OmniShadowMapPass::execute (
    std::shared_ptr< PointLight > p_light,
    std::shared_ptr< Scene > scene )
```

Definition at line 6 of file [OmniShadowMapPass.cpp](#).

```
00007 {
00008
00009     shader_program->use_shader_program();
00010
00011     glViewport(0, 0, p_light->get_omni_shadow_map()->get_shadow_width(),
00012     p_light->get_omni_shadow_map()->get_shadow_height());
00013
00014     p_light->get_omni_shadow_map()->write();
00015     glClear(GL_DEPTH_BUFFER_BIT);
00016
00017     shader_program->setUniformVec3(p_light->get_position(), "light_pos");
00018     shader_program->setUniformFloat(p_light->get_far_plane(), "far_plane");
00019
00020     std::vector<glm::mat4> light_matrices = p_light->calculate_light_transform();
00021
00022     std::stringstream ss;
00023     for (uint32_t i = 0; i < 6; i++) {
00024
00025         ss << "light_matrices[" << i << "]";
00026         shader_program->setUniformMatrix4fv(light_matrices[i], ss.str());
00027     }
00028
00029     shader_program->validate_program();
00030
00031     std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00032
00033     for (std::shared_ptr<GameObject> object : game_objects) {
00034
00035         /* if (object_is_visible(object)) {*/
00036         set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00037
00038         object->render();
00039     }
00040
00041     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00042 }
```

References [set_game_object_uniforms\(\)](#), and [shader_program](#).

Here is the call graph for this function:

6.30.3.3 set_game_object_uniforms()

```
void OmniShadowMapPass::set_game_object_uniforms (
    glm::mat4 model,
    glm::mat4 normal_model ) [virtual]
```

Implements [RenderPassSceneDependend](#).

Definition at line 44 of file [OmniShadowMapPass.cpp](#).

```
00044 { shader_program->setUniformMatrix4fv(model, "model"); }
```

References [shader_program](#).

Referenced by [execute\(\)](#).

Here is the caller graph for this function:

6.30.4 Field Documentation

6.30.4.1 shader_program

```
std::shared_ptr<OmniDirShadowShaderProgram> OmniShadowMapPass::shader_program [private]
```

Definition at line 23 of file [OmniShadowMapPass.h](#).

Referenced by [create_shader_program\(\)](#), [execute\(\)](#), and [set_game_object_uniforms\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniShadowMapPass.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[OmniShadowMapPass.cpp](#)

6.31 PointLight Class Reference

```
#include <PointLight.h>
```

Inheritance diagram for PointLight:

Collaboration diagram for PointLight:

Public Member Functions

- [PointLight \(\)](#)
- [PointLight \(GLuint shadow_width, GLuint shadow_height, GLfloat near, GLfloat far, GLfloat red, GLfloat green, GLfloat blue, GLfloat radiance, GLfloat x_pos, GLfloat y_pos, GLfloat z_pos, GLfloat con, GLfloat lin, GLfloat exp\)](#)
- std::vector< glm::mat4 > [calculate_light_transform \(\)](#)
- void [set_position \(glm::vec3 position\)](#)
- std::shared_ptr< OmniDirShadowMap > [get_omni_shadow_map \(\)](#)
- GLfloat [get_far_plane \(\)](#)
- glm::vec3 [get_position \(\)](#)
- GLfloat [get_constant_factor \(\)](#)
- GLfloat [get_linear_factor \(\)](#)
- GLfloat [get_exponent_factor \(\)](#)
- [~PointLight \(\)](#)

Protected Attributes

- std::shared_ptr< OmniDirShadowMap > [omni_dir_shadow_map](#)
- glm::vec3 [position](#)
- GLfloat [constant](#)
- GLfloat [linear](#)
- GLfloat [exponent](#)
- GLfloat [far_plane](#)

6.31.1 Detailed Description

Definition at line 9 of file [PointLight.h](#).

6.31.2 Constructor & Destructor Documentation

6.31.2.1 PointLight() [1/2]

```
PointLight::PointLight ( )
```

Definition at line 3 of file [PointLight.cpp](#).

```
00003   :
00004
00005     position(glm::vec3(0.0f)), constant(1.0f), linear(0.0f), exponent(0.0f), far_plane(0.f)
00006
00007 {
00008 }
```

6.31.2.2 PointLight() [2/2]

```
PointLight::PointLight (
    GLuint shadow_width,
    GLuint shadow_height,
    GLfloat near,
    GLfloat far,
    GLfloat red,
    GLfloat green,
    GLfloat blue,
    GLfloat radiance,
    GLfloat x_pos,
    GLfloat y_pos,
    GLfloat z_pos,
    GLfloat con,
    GLfloat lin,
    GLfloat exp )
```

Definition at line 10 of file [PointLight.cpp](#).

```
00011   :
00012
00013     Light(red, green, blue, radiance),
00014     omni_dir_shadow_map(std::make_shared<OmniDirShadowMap>()),
00015
00016     position(glm::vec3(x_pos, y_pos, z_pos)), constant(con), linear(lin), exponent(exp),
00017     far_plane(far)
00018
00019     float aspect = (float)shadow_width / (float)shadow_height;
00020     light_proj = glm::perspective(glm::radians(90.0f), aspect, near, far);
00021     omni_dir_shadow_map->init(shadow_width, shadow_height);
00022 }
```

References [Light::light_proj](#), and [omni_dir_shadow_map](#).

6.31.2.3 ~PointLight()

```
PointLight::~PointLight ( )
```

Definition at line 50 of file [PointLight.cpp](#).

```
00050 { }
```

6.31.3 Member Function Documentation

6.31.3.1 calculate_light_transform()

```
std::vector< glm::mat4 > PointLight::calculate_light_transform ( )
```

Definition at line 24 of file [PointLight.cpp](#).

```
00025 {
00026     std::vector<glm::mat4> light_matrices;
00027     //make sure all light matrices align with the order we were defining in OmniShadowMap
00028     //GL_TEXTURE_CUBE_MAP_POSITIVE_X+i; therefore start off with glm::vec3(1.0, 0.0,0.0)
00029     //+x,-x
00030     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(1.0, 0.0, 0.0),
00031     glm::vec3(0.0, -1.0, 0.0)));
00032     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(-1.0, 0.0, 0.0),
00033     glm::vec3(0.0, -1.0, 0.0)));
00034     //+y,-y
00035     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, 1.0, 0.0),
00036     glm::vec3(0.0, 0.0, 1.0)));
00037     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, 0.0, -1.0),
00038     glm::vec3(0.0, 0.0, -1.0)));
00039     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, -1.0, 0.0),
00040     glm::vec3(0.0, 0.0, -1.0)));
00041     return light_matrices;
00042 }
```

References [Light::light_proj](#), and [position](#).

6.31.3.2 get_constant_factor()

```
GLfloat PointLight::get_constant_factor ( ) [inline]
```

Definition at line 24 of file [PointLight.h](#).

```
00024 { return constant; };
```

References [constant](#).

6.31.3.3 get_exponent_factor()

```
GLfloat PointLight::get_exponent_factor ( ) [inline]
```

Definition at line 27 of file [PointLight.h](#).

```
00027 { return exponent; };
```

References [exponent](#).

6.31.3.4 get_far_plane()

```
GLfloat PointLight::get_far_plane ( )
```

Definition at line 46 of file [PointLight.cpp](#).
00046 { **return** far_plane; }

References [far_plane](#).

6.31.3.5 get_linear_factor()

```
GLfloat PointLight::get_linear_factor ( ) [inline]
```

Definition at line 25 of file [PointLight.h](#).
00025 { **return** linear; };

References [linear](#).

6.31.3.6 get_omni_shadow_map()

```
std::shared_ptr< OmniDirShadowMap > PointLight::get_omni_shadow_map ( ) [inline]
```

Definition at line 21 of file [PointLight.h](#).
00021 { **return** omni_dir_shadow_map; };

References [omni_dir_shadow_map](#).

6.31.3.7 get_position()

```
glm::vec3 PointLight::get_position ( )
```

Definition at line 48 of file [PointLight.cpp](#).
00048 { **return** position; }

References [position](#).

6.31.3.8 set_position()

```
void PointLight::set_position (  
    glm::vec3 position )
```

Definition at line 44 of file [PointLight.cpp](#).
00044 { **this->position** = position; }

References [position](#).

6.31.4 Field Documentation

6.31.4.1 constant

```
GLfloat PointLight::constant [protected]
```

Definition at line 37 of file [PointLight.h](#).

Referenced by [get_constant_factor\(\)](#).

6.31.4.2 exponent

```
GLfloat PointLight::exponent [protected]
```

Definition at line 37 of file [PointLight.h](#).

Referenced by [get_exponent_factor\(\)](#).

6.31.4.3 far_plane

```
GLfloat PointLight::far_plane [protected]
```

Definition at line 39 of file [PointLight.h](#).

Referenced by [get_far_plane\(\)](#).

6.31.4.4 linear

```
GLfloat PointLight::linear [protected]
```

Definition at line 37 of file [PointLight.h](#).

Referenced by [get_linear_factor\(\)](#).

6.31.4.5 omni_dir_shadow_map

```
std::shared_ptr<OmniDirShadowMap> PointLight::omni_dir_shadow_map [protected]
```

Definition at line 33 of file [PointLight.h](#).

Referenced by [get_omni_shadow_map\(\)](#), and [PointLight\(\)](#).

6.31.4.6 position

```
glm::vec3 PointLight::position [protected]
```

Definition at line 35 of file [PointLight.h](#).

Referenced by [calculate_light_transform\(\)](#), [get_position\(\)](#), and [set_position\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[PointLight.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/[PointLight.cpp](#)

6.32 Quad Class Reference

```
#include <Quad.h>
```

Collaboration diagram for Quad:

Public Member Functions

- [Quad \(\)](#)
- void [render \(\)](#)
- [~Quad \(\)](#)

Private Attributes

- GLuint [q_vao](#)
- GLuint [q_vbo](#)
- float [vertices](#) [20]

6.32.1 Detailed Description

Definition at line 8 of file [Quad.h](#).

6.32.2 Constructor & Destructor Documentation

6.32.2.1 Quad()

```
Quad::Quad ( )
```

Definition at line 3 of file [Quad.cpp](#).

```
00004 {
00005     glGenVertexArrays(1, &q_vao);
00006     glGenBuffers(1, &q_vbo);
00007
00008     glBindVertexArray(q_vao);
00009     glBindBuffer(GL_ARRAY_BUFFER, q_vbo);
00010     glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), &vertices, GL_STATIC_DRAW);
00011     glEnableVertexAttribArray(0);
00012     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
00013     glEnableVertexAttribArray(1);
00014     glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(3 * sizeof(float)));
00015 }
```

References [q_vao](#), [q_vbo](#), and [vertices](#).

6.32.2.2 ~Quad()

```
Quad::~Quad ( )
```

Definition at line 24 of file [Quad.cpp](#).

```
00025 {
00026     glDeleteVertexArrays(1, &q_vao);
00027     glDeleteBuffers(1, &q_vbo);
00028 }
```

References [q_vao](#), and [q_vbo](#).

6.32.3 Member Function Documentation

6.32.3.1 render()

```
void Quad::render ( )
```

Definition at line 17 of file [Quad.cpp](#).

```
00018 {
00019     glBindVertexArray(q_vao);
00020     glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
00021     glBindVertexArray(0);
00022 }
```

References [q_vao](#).

Referenced by [LightingPass::execute\(\)](#), and [LoadingScreen::render\(\)](#).

Here is the caller graph for this function:

6.32.4 Field Documentation

6.32.4.1 q_vao

```
GLuint Quad::q_vao [private]
```

Definition at line 17 of file [Quad.h](#).

Referenced by [Quad\(\)](#), [render\(\)](#), and [~Quad\(\)](#).

6.32.4.2 q_vbo

```
GLuint Quad::q_vbo [private]
```

Definition at line 17 of file [Quad.h](#).

Referenced by [Quad\(\)](#), and [~Quad\(\)](#).

6.32.4.3 vertices

```
float Quad::vertices[20] [private]
```

Initial value:

```
= {  
    -1.0f,  
    1.0f,  
    0.0f,  
    0.0f,  
    1.0f,  
    1.0f,  
    -1.0f,  
    -1.0f,  
    0.0f,  
    0.0f,  
    0.0f,  
    0.0f,  
    1.0f,  
    1.0f,  
    0.0f,  
    0.0f,  
    1.0f,  
    1.0f,  
    1.0f,  
    -1.0f,  
    0.0f,  
    0.0f,  
    0.0f  
}
```

Definition at line 19 of file [Quad.h](#).

Referenced by [Quad\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Quad.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Quad.cpp](#)

6.33 RandomNumbers Class Reference

```
#include <RandomNumbers.h>
```

Collaboration diagram for RandomNumbers:

Public Member Functions

- [RandomNumbers \(\)](#)
- [void read \(\)](#)
- [~RandomNumbers \(\)](#)

Private Member Functions

- [void generate_random_numbers \(\)](#)

Private Attributes

- [GLuint random_number_id](#)
- [std::shared_ptr< GLfloat\[\]> random_number_data](#)

6.33.1 Detailed Description

Definition at line [9](#) of file [RandomNumbers.h](#).

6.33.2 Constructor & Destructor Documentation

6.33.2.1 RandomNumbers()

```
RandomNumbers::RandomNumbers ( )
```

Definition at line [5](#) of file [RandomNumbers.cpp](#).

```
00006 {
00007     generate_random_numbers ();
00008
00009     glGenTextures(1, &random_number_id);
00010     glBindTexture(GL_TEXTURE_2D, random_number_id);
00011     // i think we won't need nearest option; so stick to linear
00012     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00013     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00014     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00015     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00016     //assuming full HD will be maximum resolution
00017     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, MAX_RESOLUTION_X, MAX_RESOLUTION_Y, 0, GL_RGBA, GL_FLOAT,
00018     random_number_data.get ());
00019 }
```

References [generate_random_numbers\(\)](#), [MAX_RESOLUTION_X](#), [MAX_RESOLUTION_Y](#), [random_number_data](#), and [random_number_id](#).

Here is the call graph for this function:

6.33.2.2 ~RandomNumbers()

```
RandomNumbers::~RandomNumbers ( )
```

Definition at line 50 of file [RandomNumbers.cpp](#).
 00050 { glDeleteTextures(1, &[random_number_id](#)); }

References [random_number_id](#).

6.33.3 Member Function Documentation

6.33.3.1 generate_random_numbers()

```
void RandomNumbers::generate_random_numbers ( ) [private]
```

Definition at line 27 of file [RandomNumbers.cpp](#).

```
00028 {
00029
00030     random\_number\_data = std::shared_ptr<GLfloat[]>(new GLfloat[MAX_RESOLUTION_X * MAX_RESOLUTION_Y *
 4]);
00031     std::mt19937_64 gen64(25121995);
00032     std::uniform_real_distribution<float> dis(0, 1);
00033
00034     for (int i = 0; i < MAX_RESOLUTION_X; i++) {
00035
00036         for (int k = 0; k < MAX_RESOLUTION_Y; k++) {
00037
00038             const GLfloat random_offset[4] = { dis(gen64), dis(gen64), dis(gen64), dis(gen64) };
00039
00040             GLuint index = (MAX_RESOLUTION_Y * i + k) * 4;
00041
00042             *(random\_number\_data.get() + index) = random_offset[0];
00043             *(random\_number\_data.get() + index + 1) = random_offset[1];
00044             *(random\_number\_data.get() + index + 2) = random_offset[2];
00045             *(random\_number\_data.get() + index + 3) = random_offset[3];
00046     }
00047 }
00048 }
```

References [MAX_RESOLUTION_X](#), [MAX_RESOLUTION_Y](#), and [random_number_data](#).

Referenced by [RandomNumbers\(\)](#).

Here is the caller graph for this function:

6.33.3.2 read()

```
void RandomNumbers::read ( )
```

Definition at line 21 of file [RandomNumbers.cpp](#).

```
00022 {
00023     glBindTexture(GL_TEXTURE0 + (GLenum)RANDOM\_NUMBERS\_SLOT;
00024     glBindTexture(GL_TEXTURE_2D, random\_number\_id);
00025 }
```

References [random_number_id](#), and [RANDOM_NUMBERS_SLOT](#).

6.33.4 Field Documentation

6.33.4.1 random_number_data

```
std::shared_ptr<GLfloat[]> RandomNumbers::random_number_data [private]
```

Definition at line 20 of file [RandomNumbers.h](#).

Referenced by [generate_random_numbers\(\)](#), and [RandomNumbers\(\)](#).

6.33.4.2 random_number_id

```
GLuint RandomNumbers::random_number_id [private]
```

Definition at line 19 of file [RandomNumbers.h](#).

Referenced by [RandomNumbers\(\)](#), [read\(\)](#), and [~RandomNumbers\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/util/[RandomNumbers.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/util/[RandomNumbers.cpp](#)

6.34 Renderer Class Reference

```
#include <Renderer.h>
```

Collaboration diagram for Renderer:

Public Member Functions

- [Renderer \(GLuint window_width, GLuint window_height\)](#)
- void [drawFrame \(std::shared_ptr< Camera > main_camera, std::shared_ptr< Scene > scene, glm::mat4 projection_matrix, GLfloat delta_time\)](#)
- void [update_window_params \(GLuint window_width, GLuint window_height\)](#)
- void [reload_shader_programs \(\)](#)
- [~Renderer \(\)](#)

Private Attributes

- GLuint `window_width`
- GLuint `window_height`
- std::shared_ptr< `GBuffer` > `gbuffer`
- `ShaderIncludes` `shader_includes`
- std::vector< std::shared_ptr< `RenderPass` > > `render_passes`
- std::shared_ptr< `OmniShadowMapPass` > `omni_shadow_map_pass`
- std::shared_ptr< `DirectionalShadowMapPass` > `directional_shadow_map_pass`
- std::shared_ptr< `GeometryPass` > `geometry_pass`
- std::shared_ptr< `LightingPass` > `lighting_pass`

6.34.1 Detailed Description

Definition at line 12 of file [Renderer.h](#).

6.34.2 Constructor & Destructor Documentation

6.34.2.1 `Renderer()`

```
Renderer::Renderer (
    GLuint window_width,
    GLuint window_height )
```

Definition at line 4 of file [Renderer.cpp](#).

```
00004 :
00005
00006     window_width(window_width), window_height(window_height),
00007     gbuffer(std::make_shared<GBuffer>(window_width, window_height)), shader_includes(),
00008     omni_shadow_map_pass(std::make_shared<OmniShadowMapPass>()),
00009     directional_shadow_map_pass(std::make_shared<DirectionalShadowMapPass>()),
00010     geometry_pass(std::make_shared<GeometryPass>()), lighting_pass(std::make_shared<LightingPass>())
00011 {
00012     render_passes.push_back(omni_shadow_map_pass);
00013     render_passes.push_back(directional_shadow_map_pass);
00014     render_passes.push_back(geometry_pass);
00015     render_passes.push_back(lightning_pass);
00016
00017     gbuffer->create();
00018 }
```

References `directional_shadow_map_pass`, `gbuffer`, `geometry_pass`, `lighting_pass`, `omni_shadow_map_pass`, and `render_passes`.

6.34.2.2 `~Renderer()`

```
Renderer::~Renderer ( )
```

Definition at line 56 of file [Renderer.cpp](#).

```
00056 { }
```

6.34.3 Member Function Documentation

6.34.3.1 drawFrame()

```
void Renderer::drawFrame (
    std::shared_ptr< Camera > main_camera,
    std::shared_ptr< Scene > scene,
    glm::mat4 projection_matrix,
    GLfloat delta_time )
```

Definition at line 20 of file [Renderer.cpp](#).

```
00021 {
00022
00023     directional_shadow_map_pass->execute(projection_matrix, main_camera, window_width, window_height,
00024     scene);
00025     // omni shadow map passes for our point lights
00026     std::vector<std::shared_ptr<PointLight>> p_lights = scene->get_point_lights();
00027     for (size_t p_light_count = 0; p_light_count < scene->get_point_light_count(); p_light_count++) {
00028         omni_shadow_map_pass->execute(p_lights[p_light_count], scene);
00029     }
00030
00031     //we will now start the geometry pass
00032     geometry_pass->execute(projection_matrix, main_camera, window_width, window_height,
00033     gbuffer->get_id(), delta_time, scene);
00034     // after geometry pass we can now do the lighting
00035     lighting_pass->execute(projection_matrix, main_camera, scene, gbuffer, delta_time);
00036 }
```

References [directional_shadow_map_pass](#), [gbuffer](#), [geometry_pass](#), [lighting_pass](#), [omni_shadow_map_pass](#), [window_height](#), and [window_width](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:

6.34.3.2 reload_shader_programs()

```
void Renderer::reload_shader_programs ( )
```

Definition at line 46 of file [Renderer.cpp](#).

```
00047 {
00048     // also reload all shader include files
00049     shader_includes = ShaderIncludes();
00050
00051     for (std::shared_ptr<RenderPass> render_pass : render_passes) {
00052         render_pass->create_shader_program();
00053     }
00054 }
```

References [render_passes](#), and [shader_includes](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:

6.34.3.3 update_window_params()

```
void Renderer::update_window_params (
    GLuint window_width,
    GLuint window_height )
```

Definition at line 38 of file [Renderer.cpp](#).

```
00039 {
00040     this->window_width = window_width;
00041     this->window_height = window_height;
00042     gbuffer->update_window_params(window_width, window_height);
00043     gbuffer->create();
00044 }
```

References [gbuffer](#), [window_height](#), and [window_width](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:

6.34.4 Field Documentation

6.34.4.1 directional_shadow_map_pass

```
std::shared_ptr<DirectionalShadowMapPass> Renderer::directional_shadow_map_pass [private]
```

Definition at line 34 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [Renderer\(\)](#).

6.34.4.2 gbuffer

```
std::shared_ptr<GBuffer> Renderer::gbuffer [private]
```

Definition at line 26 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), [Renderer\(\)](#), and [update_window_params\(\)](#).

6.34.4.3 geometry_pass

```
std::shared_ptr<GeometryPass> Renderer::geometry_pass [private]
```

Definition at line 35 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [Renderer\(\)](#).

6.34.4.4 `lighting_pass`

```
std::shared_ptr<LightingPass> Renderer::lighting_pass [private]
```

Definition at line 36 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [Renderer\(\)](#).

6.34.4.5 `omni_shadow_map_pass`

```
std::shared_ptr<OmniShadowMapPass> Renderer::omni_shadow_map_pass [private]
```

Definition at line 33 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [Renderer\(\)](#).

6.34.4.6 `render_passes`

```
std::vector<std::shared_ptr<RenderPass>> Renderer::render_passes [private]
```

Definition at line 31 of file [Renderer.h](#).

Referenced by [reload_shader_programs\(\)](#), and [Renderer\(\)](#).

6.34.4.7 `shader_includes`

```
ShaderIncludes Renderer::shader_includes [private]
```

Definition at line 28 of file [Renderer.h](#).

Referenced by [reload_shader_programs\(\)](#).

6.34.4.8 `window_height`

```
GLuint Renderer::window_height [private]
```

Definition at line 24 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [update_window_params\(\)](#).

6.34.4.9 window_width

```
GLuint Renderer::window_width [private]
```

Definition at line 24 of file [Renderer.h](#).

Referenced by [drawFrame\(\)](#), and [update_window_params\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[Renderer.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[Renderer.cpp](#)

6.35 RenderPass Class Reference

```
#include <RenderPass.h>
```

Inheritance diagram for RenderPass:

Collaboration diagram for RenderPass:

Public Member Functions

- virtual void [create_shader_program \(\)=0](#)

6.35.1 Detailed Description

Definition at line 2 of file [RenderPass.h](#).

6.35.2 Member Function Documentation

6.35.2.1 create_shader_program()

```
virtual void RenderPass::create_shader_program ( ) [pure virtual]
```

Implemented in [GeometryPass](#), [LightingPass](#), [DirectionalShadowMapPass](#), [OmniShadowMapPass](#), and [RenderPassSceneDependend](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/[RenderPass.h](#)

6.36 RenderPassSceneDependend Class Reference

```
#include <RenderPassSceneDependend.h>
```

Inheritance diagram for RenderPassSceneDependend:

Collaboration diagram for RenderPassSceneDependend:

Public Member Functions

- [RenderPassSceneDependend \(\)](#)
- virtual void [set_game_object_uniforms \(glm::mat4 model, glm::mat4 normal_model\)=0](#)
- virtual void [create_shader_program \(\)=0](#)
- [~RenderPassSceneDependend \(\)](#)

6.36.1 Detailed Description

Definition at line [7](#) of file [RenderPassSceneDependend.h](#).

6.36.2 Constructor & Destructor Documentation

6.36.2.1 RenderPassSceneDependend()

```
RenderPassSceneDependend::RenderPassSceneDependend ( )
```

Definition at line [3](#) of file [RenderPassSceneDependend.cpp](#).
00003 : [RenderPass\(\) { }](#)

6.36.2.2 ~RenderPassSceneDependend()

```
RenderPassSceneDependend::~RenderPassSceneDependend ( )
```

Definition at line [5](#) of file [RenderPassSceneDependend.cpp](#).
00005 { }

6.36.3 Member Function Documentation

6.36.3.1 create_shader_program()

```
virtual void RenderPassSceneDependend::create_shader_program ( ) [pure virtual]
```

Implements [RenderPass](#).

Implemented in [GeometryPass](#), [DirectionalShadowMapPass](#), and [OmniShadowMapPass](#).

6.36.3.2 set_game_object_uniforms()

```
virtual void RenderPassSceneDependend::set_game_object_uniforms (
    glm::mat4 model,
    glm::mat4 normal_model ) [pure virtual]
```

Implemented in [GeometryPass](#), [DirectionalShadowMapPass](#), and [OmniShadowMapPass](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[RenderPassSceneDependend.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[RenderPassSceneDependend.cpp](#)

6.37 RepeatMode Class Reference

```
#include <RepeatMode.h>
```

Inheritance diagram for RepeatMode:

Collaboration diagram for RepeatMode:

Public Member Functions

- [RepeatMode \(\)](#)
- void [activate \(\)](#) override
- [~RepeatMode \(\)](#)

6.37.1 Detailed Description

Definition at line 3 of file [RepeatMode.h](#).

6.37.2 Constructor & Destructor Documentation

6.37.2.1 RepeatMode()

```
RepeatMode::RepeatMode ( )
```

Definition at line 3 of file [RepeatMode.cpp](#).
00003 { }

6.37.2.2 ~RepeatMode()

```
RepeatMode::~RepeatMode ( )
```

Definition at line 11 of file [RepeatMode.cpp](#).
00011 { }

6.37.3 Member Function Documentation

6.37.3.1 activate()

```
void RepeatMode::activate ( ) [override], [virtual]
```

Implements [TextureWrappingMode](#).

Definition at line 5 of file [RepeatMode.cpp](#).

```
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00009 }
```

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[RepeatMode.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[RepeatMode.cpp](#)

6.38 Rotation Struct Reference

```
#include <Rotation.h>
```

Collaboration diagram for Rotation:

Data Fields

- GLfloat [degrees](#)
- glm::vec3 [axis](#)

6.38.1 Detailed Description

Definition at line 5 of file [Rotation.h](#).

6.38.2 Field Documentation

6.38.2.1 axis

```
glm::vec3 Rotation::axis
```

Definition at line 8 of file [Rotation.h](#).

Referenced by [GameObject::get_world_trafo\(\)](#), and [Scene::load_models\(\)](#).

6.38.2.2 degrees

```
GLfloat Rotation::degrees
```

Definition at line 7 of file [Rotation.h](#).

Referenced by [GameObject::get_world_trafo\(\)](#), and [Scene::load_models\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Rotation.h](#)

6.39 Scene Class Reference

```
#include <Scene.h>
```

Collaboration diagram for Scene:

Public Member Functions

- [Scene \(\)](#)
- [Scene \(std::shared_ptr< Camera > main_camera, std::shared_ptr< Window > main_window\)](#)
- [std::thread spwan \(\)](#)
- [GLuint get_point_light_count \(\) const](#)
- [std::shared_ptr< DirectionalLight > get_sun \(\)](#)
- [std::vector< std::shared_ptr< PointLight > > get_point_lights \(\) const](#)
- [std::vector< ObjMaterial > get_materials \(\)](#)
- [GLfloat get_progress \(\)](#)
- [int get_texture_count \(int index\)](#)
- [bool get_context_setup \(\) const](#)
- [std::shared_ptr< Clouds > get_clouds \(\)](#)
- [std::vector< std::shared_ptr< GameObject > > get_game_objects \(\) const](#)
- [void add_game_object \(const std::string &model_path, glm::vec3 translation, GLfloat scale, \[Rotation rot\\)\]\(#\)](#)
- [void load_models \(\)](#)
- [bool is_loaded \(\)](#)
- [void setup_game_object_context \(\)](#)
- [void bind_textures_and_buffer \(\)](#)
- [void unbind_textures_and_buffer \(\)](#)
- [void set_context_setup \(bool context_setup\)](#)
- [~Scene \(\)](#)

Private Member Functions

- bool `object_is_visible` (std::shared_ptr< `GameObject` > `game_object`)

Private Attributes

- std::shared_ptr< `Camera` > `main_camera`
- std::shared_ptr< `DirectionalLight` > `sun`
- std::vector< std::shared_ptr< `PointLight` > > `point_lights`
- std::shared_ptr< `Clouds` > `clouds`
- std::shared_ptr< `Window` > `main_window`
- std::shared_ptr< `ViewFrustumCulling` > `view_frustum_culling`
- std::vector< std::shared_ptr< `GameObject` > > `game_objects`
- GLfloat `progress`
- bool `loaded_scene`
- std::mutex `mx_progress`
- std::mutex `mx_isLoaded`
- std::mutex `mx_space_ship`
- bool `context_setup`

6.39.1 Detailed Description

Definition at line 16 of file `Scene.h`.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 `Scene()` [1/2]

```
Scene::Scene( )
```

Definition at line 5 of file `Scene.cpp`.

```
00005      :
00006
00007     main_camera(), sun(std::make_shared<DirectionalLight>(static_cast<uint32_t>(4096),
00008     static_cast<uint32_t>(4096), 1.f, 1.f, 1.f, 1.f, -0.1f, -0.8f, -0.1f,
00009     main_camera->get_near_plane(), main_camera->get_far_plane(), NUM_CASCADES)),
00010     clouds(std::make_shared<Clouds>()), main_window(),
00011     view_frustum_culling(std::make_shared<ViewFrustumCulling>(ViewFrustumCulling{})), progress(0.f),
00012     loaded_scene(false), context_setup(false)
00011 {
00012 }
```

6.39.2.2 Scene() [2/2]

```
Scene::Scene (
    std::shared_ptr< Camera > main_camera,
    std::shared_ptr< Window > main_window )
```

Definition at line 14 of file [Scene.cpp](#).

```
00014 :
00015 :
00016     main_camera(main_camera), sun(std::make_shared<DirectionalLight>(static_cast<uint32_t>(4096),
00017                                         static_cast<uint32_t>(4096), 1.f, 1.f, 1.f, 1.f, -0.1f, -0.8f,
00018                                         -0.1f, main_camera->get_near_plane(), main_camera->get_far_plane(),
00019                                         NUM_CASCADES)),
00020     clouds(std::make_shared<Clouds>()), main_window(main_window),
00021     view_frustum_culling(std::make_shared<ViewFrustumCulling>(ViewFrustumCulling{})),
00022     progress(0.f), loaded_scene(false), context_setup(false)
00023 {
00024     point_lights.reserve(MAX_POINT_LIGHTS);
00025     point_lights.push_back(std::make_shared<PointLight>(
00026         static_cast<uint32_t>(1024), static_cast<uint32_t>(1024), 0.01f, 100.f, 0.0f, 1.0f, 0.0f, 1.0f,
00027         0.0f, 0.0f, 0.0f, 0.1f, 0.1f));
00028     point_lights[0]->set_position(glm::vec3(0.0, -24.f, -24.0));
00029 }
```

References [MAX_POINT_LIGHTS](#), and [point_lights](#).

6.39.2.3 ~Scene()

```
Scene::~Scene ( )
```

Definition at line 114 of file [Scene.cpp](#).

```
00114 { }
```

6.39.3 Member Function Documentation

6.39.3.1 add_game_object()

```
void Scene::add_game_object (
    const std::string & model_path,
    glm::vec3 translation,
    GLfloat scale,
    Rotation rot )
```

Definition at line 96 of file [Scene.cpp](#).

```
00097 {
00098     game_objects.push_back(std::make_shared<GameObject>(GameObject()));
00099     game_objects.back()->init(model_path, translation, scale, rot);
00100 }
```

References [game_objects](#).

6.39.3.2 bind_textures_and_buffer()

```
void Scene::bind_textures_and_buffer ( )
```

Definition at line 86 of file [Scene.cpp](#).

```
00086 { game_objects[0]->get_model()->bind_ressources(); }
```

References [game_objects](#).

6.39.3.3 get_clouds()

```
std::shared_ptr< Clouds > Scene::get_clouds ( )
```

Definition at line 102 of file [Scene.cpp](#).

```
00102 { return clouds; }
```

References [clouds](#).

6.39.3.4 get_context_setup()

```
bool Scene::get_context_setup ( ) const
```

Definition at line 94 of file [Scene.cpp](#).

```
00094 { return context_setup; }
```

References [context_setup](#).

6.39.3.5 get_game_objects()

```
std::vector< std::shared_ptr< GameObject > > Scene::get_game_objects ( ) const
```

Definition at line 104 of file [Scene.cpp](#).

```
00104 { return game_objects; }
```

References [game_objects](#).

6.39.3.6 get_materials()

```
std::vector< ObjMaterial > Scene::get_materials ( )
```

Definition at line 66 of file [Scene.cpp](#).

```
00066 { return game_objects[0]->get_model()->get_materials(); }
```

References [game_objects](#).

6.39.3.7 get_point_light_count()

```
GLuint Scene::get_point_light_count ( ) const
```

Definition at line 31 of file [Scene.cpp](#).

```
00031 { return static_cast<uint32_t>(point_lights.size()); }
```

References [point_lights](#).

6.39.3.8 get_point_lights()

```
std::vector< std::shared_ptr< PointLight > > Scene::get_point_lights ( ) const
```

Definition at line 35 of file [Scene.cpp](#).

```
00035 { return point_lights; }
```

References [point_lights](#).

6.39.3.9 get_progress()

```
GLfloat Scene::get_progress ( )
```

Definition at line 74 of file [Scene.cpp](#).

```
00075 {  
00076     std::lock_guard<std::mutex> guard{ mx_progress };  
00077     return progress;  
00078 }
```

References [mx_progress](#), and [progress](#).

6.39.3.10 get_sun()

```
std::shared_ptr< DirectionalLight > Scene::get_sun ( )
```

Definition at line 33 of file [Scene.cpp](#).

```
00033 { return sun; }
```

References [sun](#).

6.39.3.11 get_texture_count()

```
int Scene::get_texture_count (   
    int index )
```

Definition at line 90 of file [Scene.cpp](#).

```
00090 { return game_objects[0]->get_model()->get_texture_count(); }
```

References [game_objects](#).

6.39.3.12 is_loaded()

```
bool Scene::is_loaded ( )
```

Definition at line 68 of file [Scene.cpp](#).

```
00069 {
00070     std::lock_guard<std::mutex> guard{ mx_isLoaded };
00071     return loaded_scene;
00072 }
```

References [loaded_scene](#), and [mx_isLoaded](#).

6.39.3.13 load_models()

```
void Scene::load_models ( )
```

Definition at line 37 of file [Scene.cpp](#).

```
00038 {
00039
00040     glm::vec3 sponza_offset = glm::vec3(0.f, 0.0f, 0.0f);
00041     GLfloat sponza_scale = 10.f;
00042     Rotation sponza_rot;
00043     sponza_rot.degrees = 0.0f;
00044     sponza_rot.axis = glm::vec3(0.0f, 1.0f, 0.0f);
00045
00046     glm::vec3 clouds_offset = glm::vec3(-3.f, 20.0f, -3.0f);
00047     glm::vec3 clouds_scale = glm::vec3(1.f, 1.f, 1.f);
00048     clouds->set_scale(clouds_scale);
00049     clouds->set_translation(clouds_offset);
00050
00051     std::stringstream modelFile;
00052     modelFile << CMAKELISTS_DIR << "/Resources/Models/dinosaurs.obj";
00053     /*.../Resources/Models/Pillum/PillumPainting_Export.obj",*/
00054     /*.../Resources/Models/crytek-sponza/sponza_triang.obj",*/
00055
00056     std::shared_ptr<GameObject> sponza = std::make_shared<GameObject>(modelFile.str(), sponza_offset,
00057     sponza_scale, sponza_rot);
00058     progress += 1.f;
00059     game_objects.push_back(sponza);
00060
00061     mx_isLoaded.lock();
00062     loaded_scene = true;
00063     mx_isLoaded.unlock();
00064 }
```

References [Rotation::axis](#), [clouds](#), [Rotation::degrees](#), [game_objects](#), [loaded_scene](#), [mx_isLoaded](#), and [progress](#).

Referenced by [spwan\(\)](#).

Here is the caller graph for this function:

6.39.3.14 object_is_visible()

```
bool Scene::object_is_visible (
    std::shared_ptr< GameObject > game_object ) [private]
```

Definition at line 106 of file [Scene.cpp](#).

```
00107 {
00108     return view_frustum_culling->is_inside(static_cast<GLfloat>(main_window->get_buffer_width()) /
00109         static_cast<GLfloat>(main_window->get_buffer_height()),
00110         main_camera,
00111         game_object->get_aabb(),
00112         game_object->get_world_trafo());
```

References [main_camera](#), [main_window](#), and [view_frustum_culling](#).

6.39.3.15 set_context_setup()

```
void Scene::set_context_setup (
    bool context_setup )
```

Definition at line 92 of file [Scene.cpp](#).

```
00092 { this->context_setup = context_setup; }
```

References [context_setup](#).

6.39.3.16 setup_game_object_context()

```
void Scene::setup_game_object_context ( )
```

Definition at line 80 of file [Scene.cpp](#).

```
00081 {
00082     game_objects[0]->get_model()->create_render_context();
00083     context_setup = true;
00084 }
```

References [context_setup](#), and [game_objects](#).

6.39.3.17 spwan()

```
std::thread Scene::spwan ( ) [inline]
```

Definition at line 22 of file [Scene.h](#).

```
00023 {
00024     return std::thread([=] { load_models(); });
00025 }
```

References [load_models\(\)](#).

Here is the call graph for this function:

6.39.3.18 unbind_textures_and_buffer()

```
void Scene::unbind_textures_and_buffer ( )
```

Definition at line 88 of file [Scene.cpp](#).

```
00088 { game_objects[0]->get_model()->unbind_resources(); }
```

References [game_objects](#).

6.39.4 Field Documentation

6.39.4.1 clouds

```
std::shared_ptr<Clouds> Scene::clouds [private]
```

Definition at line 60 of file [Scene.h](#).

Referenced by [get_clouds\(\)](#), and [load_models\(\)](#).

6.39.4.2 context_setup

```
bool Scene::context_setup [private]
```

Definition at line 74 of file [Scene.h](#).

Referenced by [get_context_setup\(\)](#), [set_context_setup\(\)](#), and [setup_game_object_context\(\)](#).

6.39.4.3 game_objects

```
std::vector<std::shared_ptr<GameObject>> Scene::game_objects [private]
```

Definition at line 65 of file [Scene.h](#).

Referenced by [add_game_object\(\)](#), [bind_textures_and_buffer\(\)](#), [get_game_objects\(\)](#), [get_materials\(\)](#), [get_texture_count\(\)](#), [load_models\(\)](#), [setup_game_object_context\(\)](#), and [unbind_textures_and_buffer\(\)](#).

6.39.4.4 loaded_scene

```
bool Scene::loaded_scene [private]
```

Definition at line 68 of file [Scene.h](#).

Referenced by [is_loaded\(\)](#), and [load_models\(\)](#).

6.39.4.5 main_camera

```
std::shared_ptr<Camera> Scene::main_camera [private]
```

Definition at line 55 of file [Scene.h](#).

Referenced by [object_is_visible\(\)](#).

6.39.4.6 main_window

```
std::shared_ptr<Window> Scene::main_window [private]
```

Definition at line 62 of file [Scene.h](#).

Referenced by [object_is_visible\(\)](#).

6.39.4.7 mx_isLoaded

```
std::mutex Scene::mx_isLoaded [private]
```

Definition at line 71 of file [Scene.h](#).

Referenced by [is_loaded\(\)](#), and [load_models\(\)](#).

6.39.4.8 mx_progress

```
std::mutex Scene::mx_progress [private]
```

Definition at line 70 of file [Scene.h](#).

Referenced by [get_progress\(\)](#).

6.39.4.9 mx_space_ship

```
std::mutex Scene::mx_space_ship [private]
```

Definition at line 72 of file [Scene.h](#).

6.39.4.10 point_lights

```
std::vector<std::shared_ptr<PointLight>> Scene::point_lights [private]
```

Definition at line 59 of file [Scene.h](#).

Referenced by [get_point_light_count\(\)](#), [get_point_lights\(\)](#), and [Scene\(\)](#).

6.39.4.11 progress

GLfloat Scene::progress [private]

Definition at line 67 of file [Scene.h](#).

Referenced by [get_progress\(\)](#), and [load_models\(\)](#).

6.39.4.12 sun

std::shared_ptr<[DirectionalLight](#)> Scene::sun [private]

Definition at line 58 of file [Scene.h](#).

Referenced by [get_sun\(\)](#).

6.39.4.13 view_frustum_culling

std::shared_ptr<[ViewFrustumCulling](#)> Scene::view_frustum_culling [private]

Definition at line 63 of file [Scene.h](#).

Referenced by [object_is_visible\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Scene.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Scene.cpp](#)

6.40 ShaderIncludes Class Reference

```
#include <ShaderIncludes.h>
```

Collaboration diagram for [ShaderIncludes](#):

Public Member Functions

- [ShaderIncludes \(\)](#)
- [~ShaderIncludes \(\)](#)

Private Attributes

- std::vector< const char * > [includeNames](#)
- std::vector< const char * > [file_locations_relative](#)

6.40.1 Detailed Description

Definition at line 6 of file [ShaderIncludes.h](#).

6.40.2 Constructor & Destructor Documentation

6.40.2.1 ShaderIncludes()

ShaderIncludes::ShaderIncludes ()

Definition at line 14 of file [ShaderIncludes.cpp](#).

```
00015 {  
00016  
00017     assert(includeNames.size() == file_locations_relative.size());  
00018  
00019     std::vector<std::string> file_locations_abs;  
00020     for (uint32_t i = 0; i < static_cast<uint32_t>(includeNames.size()); i++) {  
00021  
00022         std::stringstream aux;  
00023         aux << CMAKELISTS_DIR;  
00024         aux << file_locations_relative[i];  
00025         file_locations_abs.push_back(aux.str());  
00026     }  
00027  
00028     for (uint32_t i = 0; i < static_cast<uint32_t>(includeNames.size()); i++) {  
00029  
00030         File file(file_locations_abs[i]);  
00031         std::string file_content = file.read();  
00032         char tmpstr[200];  
00033         sprintf(tmpstr, 200, "%s", includeNames[i]);  
00034         glNamedStringARB(GL_SHADER_INCLUDE_ARB, static_cast<GLint>(strlen(tmpstr)), tmpstr,  
00035             static_cast<GLint>(strlen(file_content.c_str())), file_content.c_str());  
00036 }
```

References [file_locations_relative](#), [includeNames](#), and [File::read\(\)](#).

Here is the call graph for this function:

6.40.2.2 ~ShaderIncludes()

ShaderIncludes::~ShaderIncludes ()

Definition at line 38 of file [ShaderIncludes.cpp](#).

```
00038 { }
```

6.40.3 Field Documentation

6.40.3.1 file_locations_relative

```
std::vector<const char*> ShaderIncludes::file_locations_relative [private]
```

Initial value:

```
= { "/Src/host_device_shared.h",
  "/Resources/Shaders/common/Matlib.glsl",
  "/Resources/Shaders/pbr/microfacet.glsl",
  "/Resources/Shaders/common/ShadingLibrary.glsl",
  "/Resources/Shaders/pbr/brdf/disney.glsl",
  "/Resources/Shaders/pbr/brdf/frostbite.glsl",
  "/Resources/Shaders/pbr/brdf/pbrBook.glsl",
  "/Resources/Shaders/pbr/brdf/phong.glsl",
  "/Resources/Shaders/pbr/brdf/unreal4.glsl",
  "/Resources/Shaders/clouds/clouds.glsl",
  "/Resources/Shaders/common/grad_noise.glsl",
  "/Resources/Shaders/common/worley_noise.glsl",
  "/Src/bindings.h",
  "/Src/GlobalValues.h",
  "/Resources/Shaders/common/directional_light.glsl",
  "/Resources/Shaders/common/light.glsl",
  "/Resources/Shaders/common/material.glsl",
  "/Resources/Shaders/common/point_light.glsl" }
```

Definition at line 33 of file [ShaderIncludes.h](#).

Referenced by [ShaderIncludes\(\)](#).

6.40.3.2 includeNames

```
std::vector<const char*> ShaderIncludes::includeNames [private]
```

Initial value:

```
= { "host_device_shared.h",
  "Matlib.glsl",
  "microfacet.glsl",
  "ShadingLibrary.glsl",
  "disney.glsl",
  "frostbite.glsl",
  "pbrBook.glsl",
  "phong.glsl",
  "unreal4.glsl",
  "clouds.glsl",
  "grad_noise.glsl",
  "worley_noise.glsl",
  "bindings.h",
  "GlobalValues.h",
  "directional_light.glsl",
  "light.glsl",
  "material.glsl",
  "point_light.glsl" }
```

Definition at line 14 of file [ShaderIncludes.h](#).

Referenced by [ShaderIncludes\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[ShaderIncludes.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[ShaderIncludes.cpp](#)

6.41 ShaderProgram Class Reference

```
#include <ShaderProgram.h>
```

Inheritance diagram for ShaderProgram:

Collaboration diagram for ShaderProgram:

Public Member Functions

- [ShaderProgram \(\)](#)
- [ShaderProgram \(const ShaderProgram &\) = default](#)
- [void create_from_files \(const char *vertex_location, const char *fragment_location\)](#)
- [void create_from_files \(const char *vertex_location, const char *geometry_location, const char *fragment_location\)](#)
- [void create_computer_shader_program_from_file \(const char *compute_location\)](#)
- [bool setUniformVec3 \(glm::vec3 uniform, const std::string &shaderUniformName\)](#)
- [bool setUniformFloat \(GLfloat uniform, const std::string &shaderUniformName\)](#)
- [bool setUniformInt \(GLint uniform, const std::string &shaderUniformName\)](#)
- [bool setUniformMatrix4fv \(glm::mat4 matrix, const std::string &shaderUniformName\)](#)
- [bool setUniformBlockBinding \(GLuint block_binding, const std::string &shaderUniformName\)](#)
- [GLuint get_id \(\) const](#)
- [void use_shader_program \(\)](#)
- [void validate_program \(\)](#)
- [~ShaderProgram \(\)](#)

Protected Member Functions

- [void add_shader \(GLuint program, const char *shader_code, GLenum shader_type\)](#)
- [void compile_shader_program \(const char *vertex_code, const char *fragment_code\)](#)
- [void compile_shader_program \(const char *vertex_code, const char *geometry_code, const char *fragment_code\)](#)
- [void compile_compute_shader_program \(const char *compute_code\)](#)
- [void compile_program \(\)](#)
- [bool validateUniformLocation \(GLint uniformLocation\)](#)
- [GLuint getUniformLocation \(const std::string &shaderUniformName, bool &validity\)](#)
- [void clear_shader_program \(\)](#)

Protected Attributes

- [std::string shader_base_dir](#)
- [GLuint program_id](#)
- [const char * vertex_location](#)
- [const char * fragment_location](#)
- [const char * geometry_location](#)
- [const char * compute_location](#)

6.41.1 Detailed Description

Definition at line 13 of file [ShaderProgram.h](#).

6.41.2 Constructor & Destructor Documentation

6.41.2.1 ShaderProgram() [1/2]

```
ShaderProgram::ShaderProgram ( )
```

Definition at line 6 of file [ShaderProgram.cpp](#).

```
00006      :
00007
00008     program_id(0), vertex_location(""), fragment_location(""), geometry_location(""),
00009     compute_location("fragment_location")
00010 {
00011
00012     std::stringstream aux;
00013     aux << CMAKELISTS_DIR;
00014     aux << "/Resources/Shaders/";
00015
00016     shader_base_dir = aux.str();
00017 }
```

References [shader_base_dir](#).

6.41.2.2 ShaderProgram() [2/2]

```
ShaderProgram::ShaderProgram (
    const ShaderProgram & ) [default]
```

6.41.2.3 ~ShaderProgram()

```
ShaderProgram::~ShaderProgram ( )
```

Definition at line 318 of file [ShaderProgram.cpp](#).

```
00318 { clear_shader_program(); }
```

References [clear_shader_program\(\)](#).

Here is the call graph for this function:

6.41.3 Member Function Documentation

6.41.3.1 add_shader()

```
void ShaderProgram::add_shader (
    GLuint program,
    const char * shader_code,
    GLenum shader_type ) [protected]
```

Definition at line 108 of file [ShaderProgram.cpp](#).

```
00109 {
00110     GLuint shader = glCreateShader(shader_type);
00111
00112     // the opengl function wants c -style char array of code and the length in an array ... so we do it
00113     const GLchar* code[1];
00114     code[0] = shader_code;
00115
00116     GLint code_length[1];
00117     code_length[0] = (GLint)strlen(shader_code);
00118
00119     glShaderSource(shader, 1, code, code_length);
00120     glCompileShader(shader);
00121     //glCompileShaderIncludeARB(shader);
00122
00123     GLint result = 0;
00124     GLchar eLog[1024] = { 0 };
00125
00126     //retrieve status of the shader and print if any error occured
00127     glGetShaderiv(shader, GL_COMPILE_STATUS, &result);
00128
00129     if (!result) {
00130         glGetShaderInfoLog(shader, sizeof(eLog), NULL, eLog);
00131         printf("Error compiling the %d shader: '%s'\n", shader_type, eLog);
00132         printf("%s", shader_code);
00133         return;
00134     }
00135
00136     // we are happy, everything went well so attach shader to program
00137     glBindShader(program, shader);
00138 }
```

Referenced by [compile_compute_shader_program\(\)](#), and [compile_shader_program\(\)](#).

Here is the caller graph for this function:

6.41.3.2 clear_shader_program()

```
void ShaderProgram::clear_shader_program ( ) [protected]
```

Definition at line 308 of file [ShaderProgram.cpp](#).

```
00309 {
00310     //don't trash the id's!!
00311     //delete it from memory!!
00312     if (program_id != 0) {
00313         glDeleteProgram(program_id);
00314         program_id = 0;
00315     }
00316 }
```

References [program_id](#).

Referenced by [~ShaderProgram\(\)](#).

Here is the caller graph for this function:

6.41.3.3 compile_compute_shader_program()

```
void ShaderProgram::compile_compute_shader_program (
    const char * compute_code ) [protected]
```

Definition at line 175 of file [ShaderProgram.cpp](#).

```
00176 {
00177     program_id = glCreateProgram();
00178
00179     if (!program_id) {
00180         printf("Error creating shader program!\n");
00181         return;
00182     }
00183
00184     add_shader(program_id, compute_code, GL_COMPUTE_SHADER);
00185
00186     compile_program();
00187 }
```

References [add_shader\(\)](#), [compile_program\(\)](#), and [program_id](#).

Referenced by [create_computer_shader_program_from_file\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.4 compile_program()

```
void ShaderProgram::compile_program ( ) [protected]
```

Definition at line 189 of file [ShaderProgram.cpp](#).

```
00190 {
00191     //as simple as that; opengl will link it for us :
00192     gllinkProgram(program_id);
00193     GLint result = 0;
00194     GLchar eLog[1024] = { 0 };
00195
00196     glGetProgramiv(program_id, GL_LINK_STATUS, &result);
00197
00198     if (!result) {
00199         glGetProgramInfoLog(program_id, sizeof(eLog), NULL, eLog);
00200         printf("Error linking program: '%s'\n", eLog);
00201         return;
00202     }
00203
00204     validate_program();
00205 }
```

References [program_id](#), and [validate_program\(\)](#).

Referenced by [compile_compute_shader_program\(\)](#), and [compile_shader_program\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.5 compile_shader_program() [1/2]

```
void ShaderProgram::compile_shader_program (
    const char * vertex_code,
    const char * fragment_code ) [protected]
```

Definition at line 140 of file [ShaderProgram.cpp](#).

```
00141 {
00142     // retrieve the id; we need to reference it later on
00143     program_id = glCreateProgram();
00144
00145     if (!program_id) {
00146         printf("Error creating shader program !\n");
```

```

00147     return;
00148 }
00149 // we will always need one vertex ShaderProgram
00150 add_shader(program_id, vertex_code, GL_VERTEX_SHADER);
00151 // and one fragment ShaderProgram
00152 add_shader(program_id, fragment_code, GL_FRAGMENT_SHADER);
00153
00154 //we attached all shaders
00155 //so compile program
00156 compile_program();
00157 }
```

References [add_shader\(\)](#), [compile_program\(\)](#), and [program_id](#).

Referenced by [create_from_files\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.6 compile_shader_program() [2/2]

```

void ShaderProgram::compile_shader_program (
    const char * vertex_code,
    const char * geometry_code,
    const char * fragment_code ) [protected]
```

Definition at line 159 of file [ShaderProgram.cpp](#).

```

00160 {
00161     program_id = glCreateProgram();
00162
00163     if (!program_id) {
00164         printf("Error creating shader program!\n");
00165         return;
00166     }
00167
00168     add_shader(program_id, vertex_code, GL_VERTEX_SHADER);
00169     add_shader(program_id, geometry_code, GL_GEOMETRY_SHADER);
00170     add_shader(program_id, fragment_code, GL_FRAGMENT_SHADER);
00171
00172     compile_program();
00173 }
```

References [add_shader\(\)](#), [compile_program\(\)](#), and [program_id](#).

Here is the call graph for this function:

6.41.3.7 create_computer_shader_program_from_file()

```

void ShaderProgram::create_computer_shader_program_from_file (
    const char * compute_location )
```

Definition at line 72 of file [ShaderProgram.cpp](#).

```

00073 {
00074
00075     std::stringstream comp_shader;
00076     comp_shader << shader_base_dir << compute_location;
00077     File compute_shader_file(comp_shader.str());
00078     std::string file = compute_shader_file.read();
00079
00080     const char* compute_code = file.c_str();
00081
00082     this->compute_location = compute_location;
00083
00084     compile_compute_shader_program(compute_code);
00085 }
```

References [compile_compute_shader_program\(\)](#), [compute_location](#), [File::read\(\)](#), and [shader_base_dir](#).

Referenced by [ComputeShaderProgram::reload\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.8 `create_from_files()` [1/2]

```
void ShaderProgram::create_from_files (
    const char * vertex_location,
    const char * fragment_location )
```

Definition at line 19 of file [ShaderProgram.cpp](#).

```
00020 {
00021
00022     std::stringstream vertex_shader;
00023     std::stringstream fragment_shader;
00024     vertex_shader << shader_base_dir << vertex_location;
00025     fragment_shader << shader_base_dir << fragment_location;
00026
00027     File vertex_shader_file(vertex_shader.str());
00028     File fragment_shader_file(fragment_shader.str());
00029
00030     std::string vertex_string = vertex_shader_file.read();
00031     std::string fragment_string = fragment_shader_file.read();
00032
00033     //we need c-like strings ....
00034     const char* vertex_code = vertex_string.c_str();
00035     const char* fragment_code = fragment_string.c_str();
00036
00037     this->vertex_location = (vertex_location);
00038     this->fragment_location = (fragment_location);
00039
00040     compile_shader_program(vertex_code, fragment_code);
00041 }
```

References [compile_shader_program\(\)](#), [fragment_location](#), [File::read\(\)](#), [shader_base_dir](#), and [vertex_location](#).

Referenced by [OmniDirShadowShaderProgram::reload\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.9 `create_from_files()` [2/2]

```
void ShaderProgram::create_from_files (
    const char * vertex_location,
    const char * geometry_location,
    const char * fragment_location )
```

Definition at line 43 of file [ShaderProgram.cpp](#).

```
00044 {
00045
00046     std::stringstream vertex_shader;
00047     std::stringstream geometry_shader;
00048     std::stringstream fragment_shader;
00049     vertex_shader << shader_base_dir << vertex_location;
00050     geometry_shader << shader_base_dir << geometry_location;
00051     fragment_shader << shader_base_dir << fragment_location;
00052
00053     File vertex_shader_file(vertex_shader.str());
00054     File geometry_shader_file(geometry_shader.str());
00055     File fragment_shader_file(fragment_shader.str());
00056
00057     std::string vertex_string = vertex_shader_file.read();
00058     std::string geometry_string = geometry_shader_file.read();
00059     std::string fragment_string = fragment_shader_file.read();
00060
00061     const char* vertex_code = vertex_string.c_str();
00062     const char* geometry_code = geometry_string.c_str();
00063     const char* fragment_code = fragment_string.c_str();
00064
00065     this->vertex_location = vertex_location;
00066     this->fragment_location = fragment_location;
00067     this->geometry_location = geometry_location;
00068
00069     compile_shader_program(vertex_code, geometry_code, fragment_code);
00070 }
```

References [compile_shader_program\(\)](#), [fragment_location](#), [geometry_location](#), [File::read\(\)](#), [shader_base_dir](#), and [vertex_location](#).

Here is the call graph for this function:

6.41.3.10 get_id()

```
GLuint ShaderProgram::get_id ( ) const
```

Definition at line 87 of file [ShaderProgram.cpp](#).
 00087 { **return** program_id; }

References [program_id](#).

6.41.3.11 getUniformLocation()

```
GLuint ShaderProgram::getUniformLocation (
    const std::string & shaderUniformName,
    bool & validity ) [protected]
```

Definition at line 285 of file [ShaderProgram.cpp](#).

```
00286 {
00287     GLuint uniform_location = glGetUniformLocation(program_id, shaderUniformName.c_str());
00288     validity = validateUniformLocation(uniform_location);
00289
00290 #ifdef NDEBUG
00291     // nondebug
00292
00293 #else
00294
00295     if (!validity) {
00296         /*std::stringstream ss;
00297             ss << "You have set a wrong uniform! "
00298             << "Name: " << shaderUniformName << "\n";
00299
00300         std::cout << ss.str();*/
00301     }
00302
00303 #endif
00304
00305     return uniform_location;
00306 }
```

References [program_id](#), and [validateUniformLocation\(\)](#).

Referenced by [setUniformFloat\(\)](#), [setUniformInt\(\)](#), [setUniformMatrix4fv\(\)](#), and [setUniformVec3\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.41.3.12 setUniformBlockBinding()

```
bool ShaderProgram::setUniformBlockBinding (
    GLuint block_binding,
    const std::string & shaderUniformName )
```

Definition at line 257 of file [ShaderProgram.cpp](#).

```
00258 {
00259
00260     bool validity = true;
00261     GLint uniform_location = glGetUniformLocation(program_id, shaderUniformName.c_str());
00262
00263     (uniform_location < 0) ? validity = false : validity = true;
00264
00265     if (validity) {
00266         glUniformBlockBinding(program_id, uniform_location, block_binding);
00267     } else {
00268 #ifdef NDEBUG
00269     // nondebug
00270
00271 #else
00272     //printf("Error at setting uniform block binding!");
00273 #endif
00274     }
00275
00276     return validity;
00277 }
```

References [program_id](#).

6.41.3.13 setUniformFloat()

```
bool ShaderProgram::setUniformFloat (
    GLfloat uniform,
    const std::string & shaderUniformName )
```

Definition at line 219 of file [ShaderProgram.cpp](#).

```
00220 {
00221
00222     bool validity = true;
00223     GLuint uniform_location = getUniformLocation(shaderUniformName, validity);
00224
00225     if (validity) {
00226         glUniform1f(uniform_location, uniform);
00227     }
00228
00229     return validity;
00230 }
```

References [getUniformLocation\(\)](#).

Here is the call graph for this function:

6.41.3.14 setUniformInt()

```
bool ShaderProgram::setUniformInt (
    GLint uniform,
    const std::string & shaderUniformName )
```

Definition at line 232 of file [ShaderProgram.cpp](#).

```
00233 {
00234     bool validity = true;
00235     GLuint uniform_location = getUniformLocation(shaderUniformName, validity);
00236
00237     if (validity) {
00238         glUniform1i(uniform_location, uniform);
00239     }
00240
00241     return validity;
00242 }
```

References [getUniformLocation\(\)](#).

Here is the call graph for this function:

6.41.3.15 setUniformMatrix4fv()

```
bool ShaderProgram::setUniformMatrix4fv (
    glm::mat4 matrix,
    const std::string & shaderUniformName )
```

Definition at line 244 of file [ShaderProgram.cpp](#).

```
00245 {
00246
00247     bool validity = true;
00248     GLuint uniform_location = getUniformLocation(shaderUniformName, validity);
00249
00250     if (validity) {
00251         glUniformMatrix4fv(uniform_location, 1, GL_FALSE, glm::value\_ptr(matrix));
00252     }
00253
00254     return validity;
00255 }
```

References [getUniformLocation\(\)](#).

Here is the call graph for this function:

6.41.3.16 setUniformVec3()

```
bool ShaderProgram::setUniformVec3 (
    glm::vec3 uniform,
    const std::string & shaderUniformName )
```

Definition at line 207 of file [ShaderProgram.cpp](#).

```
00208 {
00209     bool validity = true;
00210     GLuint uniform_location = getUniformLocation(shaderUniformName, validity);
00211
00212     if (validity) {
00213         glUniform3f(uniform_location, uniform.x, uniform.y, uniform.z);
00214     }
00215
00216     return validity;
00217 }
```

References [getUniformLocation\(\)](#).

Here is the call graph for this function:

6.41.3.17 use_shader_program()

```
void ShaderProgram::use_shader_program ( )
```

Definition at line 106 of file [ShaderProgram.cpp](#).

```
00106 { glUseProgram(program_id); }
```

References [program_id](#).

6.41.3.18 validate_program()

```
void ShaderProgram::validate_program ( )
```

Definition at line 89 of file [ShaderProgram.cpp](#).

```
00090 {
00091
00092     GLint result = 0;
00093     GLchar eLog[1024] = { 0 };
00094
00095     glValidateProgram(program_id);
00096
00097     glGetProgramiv(program_id, GL_VALIDATE_STATUS, &result);
00098
00099     if (!result) {
00100         glGetProgramInfoLog(program_id, sizeof(eLog), NULL, eLog);
00101         printf("Error validating program: '%s'\n", eLog);
00102         return;
00103     }
00104 }
```

References [program_id](#).

Referenced by [compile_program\(\)](#).

Here is the caller graph for this function:

6.41.3.19 validateUniformLocation()

```
bool ShaderProgram::validateUniformLocation (
    GLint uniformLocation ) [protected]
```

Definition at line 279 of file [ShaderProgram.cpp](#).

```
00280 {
00281     // if uniform location is invalid (f.e. var disappears because of optimizing of unused vars)
00282     return (uniformLocation == -1) ? false : true;
00283 }
```

Referenced by [getUniformLocation\(\)](#).

Here is the caller graph for this function:

6.41.4 Field Documentation

6.41.4.1 compute_location

```
const char* ShaderProgram::compute_location [protected]
```

Definition at line 46 of file [ShaderProgram.h](#).

Referenced by [create_computer_shader_program_from_file\(\)](#), and [ComputeShaderProgram::reload\(\)](#).

6.41.4.2 fragment_location

```
const char* ShaderProgram::fragment_location [protected]
```

Definition at line 44 of file [ShaderProgram.h](#).

Referenced by [create_from_files\(\)](#), and [OmniDirShadowShaderProgram::reload\(\)](#).

6.41.4.3 geometry_location

```
const char* ShaderProgram::geometry_location [protected]
```

Definition at line 45 of file [ShaderProgram.h](#).

Referenced by [create_from_files\(\)](#), and [OmniDirShadowShaderProgram::reload\(\)](#).

6.41.4.4 program_id

```
GLuint ShaderProgram::program_id [protected]
```

Definition at line 40 of file [ShaderProgram.h](#).

Referenced by [clear_shader_program\(\)](#), [compile_compute_shader_program\(\)](#), [compile_program\(\)](#), [compile_shader_program\(\)](#), [get_id\(\)](#), [GeometryPassShaderProgram::get_program_id\(\)](#), [getUniformLocation\(\)](#), [setUniformBlockBinding\(\)](#), [use_shader_program\(\)](#), and [validate_program\(\)](#).

6.41.4.5 shader_base_dir

```
std::string ShaderProgram::shader_base_dir [protected]
```

Definition at line 39 of file [ShaderProgram.h](#).

Referenced by [create_computer_shader_program_from_file\(\)](#), [create_from_files\(\)](#), and [ShaderProgram\(\)](#).

6.41.4.6 vertex_location

```
const char* ShaderProgram::vertex_location [protected]
```

Definition at line 43 of file [ShaderProgram.h](#).

Referenced by [create_from_files\(\)](#), and [OmniDirShadowShaderProgram::reload\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[ShaderProgram.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/[ShaderProgram.cpp](#)

6.42 ShadowMap Class Reference

```
#include <ShadowMap.h>
```

Inheritance diagram for ShadowMap:

Collaboration diagram for ShadowMap:

Public Member Functions

- [ShadowMap \(\)](#)
- virtual bool [init \(GLuint width, GLuint height\)](#)
- virtual void [write \(\)](#)
- virtual void [read \(GLenum texture_unit\)](#)
- GLuint [get_shadow_width \(\) const](#)
- GLuint [get_shadow_height \(\) const](#)
- GLuint [get_id \(\) const](#)
- virtual [~ShadowMap \(\)](#)

Protected Attributes

- GLuint FBO
- GLuint shadow_map
- GLuint shadow_width
- GLuint shadow_height

6.42.1 Detailed Description

Definition at line 6 of file [ShadowMap.h](#).

6.42.2 Constructor & Destructor Documentation

6.42.2.1 ShadowMap()

```
ShadowMap::ShadowMap ( )
```

Definition at line 4 of file [ShadowMap.cpp](#).

```
00004     :
00005
00006     FBO(0), shadow_map(0), shadow_width(0), shadow_height(0)
00007
00008 {
00009 }
```

6.42.2.2 ~ShadowMap()

```
ShadowMap::~ShadowMap ( ) [virtual]
```

Definition at line 59 of file [ShadowMap.cpp](#).

```
00060 {
00061     if (FBO) {
00062         glDeleteFramebuffers(1, &FBO);
00063     }
00064
00065     if (shadow_map) {
00066         glDeleteTextures(1, &shadow_map);
00067     }
00068 }
```

References [FBO](#), and [shadow_map](#).

6.42.3 Member Function Documentation

6.42.3.1 get_id()

```
GLuint ShadowMap::get_id ( ) const [inline]
```

Definition at line 16 of file [ShadowMap.h](#).
 00016 { **return shadow_map;** };

References [shadow_map](#).

6.42.3.2 get_shadow_height()

```
GLuint ShadowMap::get_shadow_height ( ) const [inline]
```

Definition at line 15 of file [ShadowMap.h](#).
 00015 { **return shadow_height;** };

References [shadow_height](#).

6.42.3.3 get_shadow_width()

```
GLuint ShadowMap::get_shadow_width ( ) const [inline]
```

Definition at line 14 of file [ShadowMap.h](#).
 00014 { **return shadow_width;** };

References [shadow_width](#).

6.42.3.4 init()

```
bool ShadowMap::init (
    GLuint width,
    GLuint height ) [virtual]
```

Reimplemented in [OmniDirShadowMap](#).

Definition at line 11 of file [ShadowMap.cpp](#).

```
00012 {
00013     shadow_width = width;
00014     shadow_height = height;
00015
00016     glGenFramebuffers(1, &FBO);
00017
00018     glGenTextures(1, &shadow_map);
00019     glBindTexture(GL_TEXTURE_2D, shadow_map);
00020     glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadow_width, shadow_height, 0,
00021                 GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00022     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00023     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00024
00025     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
00026     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
00027
00028     float border_color[] = { 1.0f, 1.0f, 1.0f, 1.0f };
```

```

00029     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
00030
00031     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00032     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, shadow_map, 0);
00033
00034     glDrawBuffer(GL_NONE);
00035     glReadBuffer(GL_NONE);
00036
00037     GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00038
00039     if (status != GL_FRAMEBUFFER_COMPLETE) {
00040         printf("Framebuffer error: %i\n", status);
00041         return false;
00042     }
00044
00045     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00046
00047     return true;
00048 }
```

References [FBO](#), [shadow_height](#), [shadow_map](#), and [shadow_width](#).

6.42.3.5 `read()`

```
void ShadowMap::read (
    GLenum texture_unit ) [virtual]
```

Reimplemented in [OmniDirShadowMap](#).

Definition at line 52 of file [ShadowMap.cpp](#).

```

00053 {
00054
00055     glBindTexture(GL_TEXTURE_2D, shadow_map);
00056
00057 }
```

References [shadow_map](#).

6.42.3.6 `write()`

```
void ShadowMap::write ( ) [virtual]
```

Reimplemented in [OmniDirShadowMap](#).

Definition at line 50 of file [ShadowMap.cpp](#).

```
00050 { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }
```

References [FBO](#).

6.42.4 Field Documentation

6.42.4.1 FBO

GLuint ShadowMap::FBO [protected]

Definition at line 21 of file [ShadowMap.h](#).

Referenced by [OmniDirShadowMap::init\(\)](#), [init\(\)](#), [OmniDirShadowMap::write\(\)](#), [write\(\)](#), and [~ShadowMap\(\)](#).

6.42.4.2 shadow_height

GLuint ShadowMap::shadow_height [protected]

Definition at line 22 of file [ShadowMap.h](#).

Referenced by [get_shadow_height\(\)](#), [OmniDirShadowMap::init\(\)](#), and [init\(\)](#).

6.42.4.3 shadow_map

GLuint ShadowMap::shadow_map [protected]

Definition at line 21 of file [ShadowMap.h](#).

Referenced by [get_id\(\)](#), [OmniDirShadowMap::init\(\)](#), [init\(\)](#), [OmniDirShadowMap::read\(\)](#), [read\(\)](#), and [~ShadowMap\(\)](#).

6.42.4.4 shadow_width

GLuint ShadowMap::shadow_width [protected]

Definition at line 22 of file [ShadowMap.h](#).

Referenced by [get_shadow_width\(\)](#), [OmniDirShadowMap::init\(\)](#), and [init\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/[ShadowMap.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/[ShadowMap.cpp](#)

6.43 SkyBox Class Reference

```
#include <SkyBox.h>
```

Collaboration diagram for SkyBox:

Public Member Functions

- `SkyBox ()`
- `void draw_sky_box (glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width, GLuint window_height, GLfloat delta_time)`
- `void reload ()`
- `~SkyBox ()`

Private Attributes

- `GLfloat movement_speed = 0.1f`
- `GLfloat shader_playback_time`
- `std::shared_ptr< Mesh > sky_mesh`
- `std::shared_ptr< ShaderProgram > shader_program`
- `GLuint texture_id`
- `GLuint uniform_projection`
- `GLuint uniform_view`

6.43.1 Detailed Description

Definition at line 21 of file [SkyBox.h](#).

6.43.2 Constructor & Destructor Documentation

6.43.2.1 SkyBox()

```
SkyBox::SkyBox ( )
```

Definition at line 5 of file [SkyBox.cpp](#).

```
00006 {
00007     std::stringstream skybox_base_dir;
00008     skybox_base_dir << CMAKELISTS_DIR;
00009     skybox_base_dir << "/Resources/Textures/Skybox/DOOM2016/";
00010
00011     std::stringstream texture_loading;
00012     std::array<std::string, 6> skybox_textures = { "DOOM16RT.png", "DOOM16LF.png", "DOOM16UP.png",
00013     "DOOM16DN.png", "DOOM16FT.png", "DOOM16BK.png" };
00014     std::vector<std::string> skybox_faces;
00015
00016     for (uint32_t i = 0; i < static_cast<uint32_t>(skybox_textures.size()); i++) {
00017
00018         texture_loading << skybox_base_dir.str() << skybox_textures[i];
00019         skybox_faces.push_back(texture_loading.str());
00020         texture_loading.str(std::string());
00021     }
00022
00023     //time_t timer;
00024     //srand(time(&timer));
00025     srand(0);
00026     shader_playback_time = 1;
00027
00028     shader_program = std::make_shared<ShaderProgram>();
00029     shader_program->create_from_files("skybox/SkyBox.vert", "skybox/SkyBox.frag");
00030
00031     //texture setup
00032     glGenTextures(1, &texture_id);
00033     glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id);
00034 }
```

```
00035     int width, height, bit_depth;
00036
00037     for (size_t i = 0; i < 6; i++) {
00038
00039         unsigned char* texture_data = stbi_load(skybox_faces[i].c_str(), &width, &height, &bit_depth, 0);
00040         if (!texture_data) {
00041             printf("Failed to find: %s\n", skybox_faces[i].c_str());
00042             return;
00043         }
00044
00045         glTexImage2D(static_cast<GLenum>(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i), 0, GL_RGBA, width, height,
00046         0, GL_RGBA, GL_UNSIGNED_BYTE, texture_data);
00047
00048         stbi_image_free(texture_data);
00049     }
00050
00051     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00052     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00053     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00054     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00055     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00056
00057 // Mesh Setup
00058 std::vector<unsigned int> sky_box_indices = { //front
00059     0,
00060     1,
00061     2,
00062     2,
00063     1,
00064     3,
00065     2,
00066     3,
00067     5,
00068     5,
00069     3,
00070     7,
00071     //back
00072     5,
00073     7,
00074     4,
00075     4,
00076     7,
00077     6,
00078     //left
00079     4,
00080     6,
00081     0,
00082     0,
00083     6,
00084     1,
00085     //top
00086     4,
00087     0,
00088     5,
00089     5,
00090     0,
00091     2,
00092     //bottom
00093     1,
00094     6,
00095     3,
00096     3,
00097     6,
00098     7
00099 };
00100
00101 std::vector<Vertex> sky_box_vertices = {
00102
00103     Vertex(glm::vec3(-1.0f, 1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00104     glm::vec2(0.0f, 0.0f)),
00105     Vertex(glm::vec3(-1.0f, -1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00106     glm::vec2(0.0f, 0.0f)),
00107     Vertex(glm::vec3(1.0f, 1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00108     glm::vec2(0.0f, 0.0f)),
00109     Vertex(glm::vec3(1.0f, -1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00110     glm::vec2(0.0f, 0.0f)),
00111     Vertex(glm::vec3(-1.0f, 1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00112     glm::vec2(0.0f, 0.0f)),
00113     Vertex(glm::vec3(1.0f, 1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00114     glm::vec2(0.0f, 0.0f)),
```

```

00115     Vertex(glm::vec3(-1.0f, -1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00116         glm::vec2(0.0f, 0.0f)),
00117     Vertex(glm::vec3(1.0f, -1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00118         glm::vec2(0.0f, 0.0f)),
00119     );
00120 };
00121 sky_mesh = std::make_shared<Mesh>(sky_box_vertices, sky_box_indices);
00123 }

```

References [shader_playback_time](#), [shader_program](#), [sky_mesh](#), and [texture_id](#).

6.43.2.2 ~SkyBox()

```
SkyBox::~SkyBox ( )
```

Definition at line 163 of file [SkyBox.cpp](#).

```
00163 { glDeleteTextures(1, &texture_id); }
```

References [texture_id](#).

6.43.3 Member Function Documentation

6.43.3.1 draw_sky_box()

```

void SkyBox::draw_sky_box (
    glm::mat4 projection_matrix,
    glm::mat4 view_matrix,
    GLuint window_width,
    GLuint window_height,
    GLfloat delta_time )

```

Definition at line 125 of file [SkyBox.cpp](#).

```

00126 {
00127
00128 // https://learnopengl.com/Advanced-OpenGL/Cubemaps
00129 GLfloat velocity = movement_speed * delta_time;
00130 shader_playback_time = static_cast<GLfloat>(fmod(shader_playback_time + velocity, 10000));
00131
00132 glm::mat4 new_view_matrix = glm::mat4(glm::mat3(view_matrix));
00133
00134 glDepthMask(GL_FALSE);
00135 glDepthFunc(GL_LESS);
00136 //std::time_t now = std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
00137
00138 shader_program->use_shader_program();
00139
00140 shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00141 shader_program->setUniformMatrix4fv(new_view_matrix, "view");
00142 shader_program->setUniformInt(SKYBOX_TEXTURES_SLOT, "skybox");
00143
00144 glBindTexture(GL_TEXTURE0 + SKYBOX_TEXTURES_SLOT);
00145 glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id);
00146
00147 shader_program->validate_program();
00148
00149 sky_mesh->render();
00150
00151 glDepthMask(GL_TRUE);
00152 glDepthFunc(GL_LESS);

```

```
00153     glBindTexture(GL_TEXTURE_CUBE_MAP, 0);  
00154 }  
00155 }
```

References [movement_speed](#), [shader_playback_time](#), [shader_program](#), [sky_mesh](#), [SKYBOX_TEXTURES_SLOT](#), and [texture_id](#).

Referenced by [GeometryPass::execute\(\)](#).

Here is the caller graph for this function:

6.43.3.2 reload()

```
void SkyBox::reload ( )
```

Definition at line 157 of file [SkyBox.cpp](#).

```
00158 {  
00159     shader_program = std::make_shared<ShaderProgram>();  
00160     shader_program->create_from_files("Shaders/SkyBox.vert", "Shaders/SkyBox.frag");  
00161 }
```

References [shader_program](#).

6.43.4 Field Documentation

6.43.4.1 movement_speed

```
GLfloat SkyBox::movement_speed = 0.1f [private]
```

Definition at line 32 of file [SkyBox.h](#).

Referenced by [draw_sky_box\(\)](#).

6.43.4.2 shader_playback_time

```
GLfloat SkyBox::shader_playback_time [private]
```

Definition at line 34 of file [SkyBox.h](#).

Referenced by [draw_sky_box\(\)](#), and [SkyBox\(\)](#).

6.43.4.3 shader_program

```
std::shared_ptr<ShaderProgram> SkyBox::shader_program [private]
```

Definition at line 37 of file [SkyBox.h](#).

Referenced by [draw_sky_box\(\)](#), [reload\(\)](#), and [SkyBox\(\)](#).

6.43.4.4 sky_mesh

```
std::shared_ptr<Mesh> SkyBox::sky_mesh [private]
```

Definition at line 36 of file [SkyBox.h](#).

Referenced by [draw_sky_box\(\)](#), and [SkyBox\(\)](#).

6.43.4.5 texture_id

```
GLuint SkyBox::texture_id [private]
```

Definition at line 39 of file [SkyBox.h](#).

Referenced by [draw_sky_box\(\)](#), [SkyBox\(\)](#), and [~SkyBox\(\)](#).

6.43.4.6 uniform_projection

```
GLuint SkyBox::uniform_projection [private]
```

Definition at line 40 of file [SkyBox.h](#).

6.43.4.7 uniform_view

```
GLuint SkyBox::uniform_view [private]
```

Definition at line 40 of file [SkyBox.h](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/[SkyBox.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/[SkyBox.cpp](#)

6.44 Texture Class Reference

```
#include <Texture.h>
```

Collaboration diagram for Texture:

Public Member Functions

- `Texture ()`
- `Texture (const char *file_loc, std::shared_ptr< TextureWrappingMode > wrapping_mode)`
- `bool load_texture_without_alpha_channel ()`
- `bool load_texture_with_alpha_channel ()`
- `bool load_SRGB_texture_without_alpha_channel ()`
- `bool load_SRGB_texture_with_alpha_channel ()`
- `std::string get_filename () const`
- `GLuint get_id () const`
- `void use_texture (unsigned int index)`
- `void unbind_texture (unsigned int index)`
- `void clear_texture_context ()`
- `~Texture ()`

Private Attributes

- `GLuint textureID`
- `int width`
- `int height`
- `int bit_depth`
- `std::shared_ptr< TextureWrappingMode > wrapping_mode`
- `std::string file_location`

6.44.1 Detailed Description

Definition at line 15 of file `Texture.h`.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 `Texture()` [1/2]

```
Texture::Texture ( )
```

Definition at line 6 of file `Texture.cpp`.

```
00006   :
00007
00008     textureID(0), width(0), height(0), bit_depth(0),
00009     //go with repeat as standard ...
00010     wrapping_mode(std::make_shared<RepeatMode>()), file_location(std::string(""))
00011
00012 {
00013 }
```

6.44.2.2 Texture() [2/2]

```
Texture::Texture (
    const char * file_loc,
    std::shared_ptr< TextureWrappingMode > wrapping_mode )
```

Definition at line 15 of file [Texture.cpp](#).

```
00015
00016
00017     textureID(0), width(0), height(0), bit_depth(0),
00018     //go with repeat as standard ...
00019     wrapping_mode(wrapping_mode), file_location(std::string(file_loc))
00020
00021 {
00022 }
```

6.44.2.3 ~Texture()

```
Texture::~Texture ( )
```

Definition at line 179 of file [Texture.cpp](#).

```
00179 { clear_texture_context(); }
```

References [clear_texture_context\(\)](#).

Here is the call graph for this function:

6.44.3 Member Function Documentation

6.44.3.1 clear_texture_context()

```
void Texture::clear_texture_context ( )
```

Definition at line 167 of file [Texture.cpp](#).

```
00168 {
00169     glDeleteTextures(1, &textureID);
00170     textureID = 0;
00171     width = 0;
00172     height = 0;
00173     bit_depth = 0;
00174     file_location = std::string("");
00175 }
```

References [bit_depth](#), [file_location](#), [height](#), [textureID](#), and [width](#).

Referenced by [~Texture\(\)](#).

Here is the caller graph for this function:

6.44.3.2 get_filename()

```
std::string Texture::get_filename() const
```

Definition at line 157 of file [Texture.cpp](#).

```
00157 { return file_location; }
```

References [file_location](#).

6.44.3.3 get_id()

```
GLuint Texture::get_id() const
```

Definition at line 177 of file [Texture.cpp](#).

```
00177 { return textureID; }
```

References [textureID](#).

Referenced by [GUI::render\(\)](#).

Here is the caller graph for this function:

6.44.3.4 load_SRGB_texture_with_alpha_channel()

```
bool Texture::load_SRGB_texture_with_alpha_channel()
```

Definition at line 122 of file [Texture.cpp](#).

```
00123 {
00124
00125     stbi_set_flip_vertically_on_load(true);
00126     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00127     if (!texture_data) {
00128         printf("Failed to find: %s\n", file_location.c_str());
00129         return false;
00130     }
00131
00132     glGenTextures(1, &textureID);
00133     glBindTexture(GL_TEXTURE_2D, textureID);
00134
00135     wrapping_mode->activate();
00136
00137     glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB_ALPHA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
00138                 texture_data);
00139     glGenerateMipmap(GL_TEXTURE_2D);
00140
00141     // to not interpolate between transparent and not trans coloars
00142     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00143     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00144
00145     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00146     //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00147     // INVALID ENUM changed to GL_LINEAR
00148     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00149
00150     glBindTexture(GL_TEXTURE_2D, 0);
00151
00152     stbi_image_free(texture_data);
00153
00154     return true;
00155 }
```

References [bit_depth](#), [file_location](#), [height](#), [textureID](#), [width](#), and [wrapping_mode](#).

6.44.3.5 load_SRGB_texture_without_alpha_channel()

```
bool Texture::load_SRGB_texture_without_alpha_channel ( )
```

Definition at line 90 of file [Texture.cpp](#).

```
00091 {
00092
00093     stbi_set_flip_vertically_on_load(true);
00094     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00095     if (!texture_data) {
00096         printf("Failed to find: %s\n", file_location.c_str());
00097         return false;
00098     }
00099
00100    glGenTextures(1, &textureID);
00101    glBindTexture(GL_TEXTURE_2D, textureID);
00102
00103    wrapping_mode->activate();
00104
00105    // i think we won't need nearest option; so stick to linear
00106    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00107
00108    //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00109    // INVALID ENUM changed to GL_LINEAR
00110    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00111
00112    glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, texture_data);
00113    glGenerateMipmap(GL_TEXTURE_2D);
00114
00115    glBindTexture(GL_TEXTURE_2D, 0);
00116
00117    stbi_image_free(texture_data);
00118
00119    return true;
00120 }
```

References [bit_depth](#), [file_location](#), [height](#), [textureID](#), [width](#), and [wrapping_mode](#).

6.44.3.6 load_texture_with_alpha_channel()

```
bool Texture::load_texture_with_alpha_channel ( )
```

Definition at line 56 of file [Texture.cpp](#).

```
00057 {
00058
00059     stbi_set_flip_vertically_on_load(true);
00060     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00061     if (!texture_data) {
00062         printf("Failed to find: %s\n", file_location.c_str());
00063         return false;
00064     }
00065
00066     glGenTextures(1, &textureID);
00067     glBindTexture(GL_TEXTURE_2D, textureID);
00068
00069     wrapping_mode->activate();
00070
00071     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, texture_data);
00072     glGenerateMipmap(GL_TEXTURE_2D);
00073
00074     // to not interpolate between transparent and not trans coloars
00075     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00076     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00077
00078     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00079     //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00080     // INVALID ENUM changed to GL_LINEAR
00081     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00082
00083     glBindTexture(GL_TEXTURE_2D, 0);
00084
00085     stbi_image_free(texture_data);
00086
00087     return true;
00088 }
```

References [bit_depth](#), [file_location](#), [height](#), [textureID](#), [width](#), and [wrapping_mode](#).

Referenced by [GUI::GUI\(\)](#), and [LoadingScreen::init\(\)](#).

Here is the caller graph for this function:

6.44.3.7 load_texture_without_alpha_channel()

```
bool Texture::load_texture_without_alpha_channel ( )
```

Definition at line 24 of file [Texture.cpp](#).

```
00025 {  
00026  
00027     stbi_set_flip_vertically_on_load(true);  
00028     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);  
00029     if (!texture_data) {  
00030         printf("Failed to find: %s\n", file_location.c_str());  
00031         return false;  
00032     }  
00033  
00034     glGenTextures(1, &textureID);  
00035     glBindTexture(GL_TEXTURE_2D, textureID);  
00036  
00037     wrapping_mode->activate();  
00038  
00039     // i think we won't need nearest option; so stick to linear  
00040     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);  
00041  
00042     //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
00043     // INVALID ENUM changed to GL_LINEAR  
00044     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
00045  
00046     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, texture_data);  
00047     glGenerateMipmap(GL_TEXTURE_2D);  
00048  
00049     glBindTexture(GL_TEXTURE_2D, 0);  
00050  
00051     stbi_image_free(texture_data);  
00052  
00053     return true;  
00054 }
```

References [bit_depth](#), [file_location](#), [height](#), [textureID](#), [width](#), and [wrapping_mode](#).

Referenced by [LoadingScreen::init\(\)](#).

Here is the caller graph for this function:

6.44.3.8 unbind_texture()

```
void Texture::unbind_texture (  
    unsigned int index )
```

Definition at line 165 of file [Texture.cpp](#).

```
00165 { glBindTexture(GL_TEXTURE_2D, GL_TEXTURE0 + index); }
```

6.44.3.9 use_texture()

```
void Texture::use_texture (  
    unsigned int index )
```

Definition at line 159 of file [Texture.cpp](#).

```
00160 {  
00161     glActiveTexture(GL_TEXTURE0 + index);  
00162     glBindTexture(GL_TEXTURE_2D, textureID);  
00163 }
```

References [textureID](#).

Referenced by [LoadingScreen::render\(\)](#).

Here is the caller graph for this function:

6.44.4 Field Documentation

6.44.4.1 bit_depth

```
int Texture::bit_depth [private]
```

Definition at line 39 of file [Texture.h](#).

Referenced by [clear_texture_context\(\)](#), [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), and [load_texture_without_alpha_channel\(\)](#).

6.44.4.2 file_location

```
std::string Texture::file_location [private]
```

Definition at line 43 of file [Texture.h](#).

Referenced by [clear_texture_context\(\)](#), [get_filename\(\)](#), [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), and [load_texture_without_alpha_channel\(\)](#).

6.44.4.3 height

```
int Texture::height [private]
```

Definition at line 39 of file [Texture.h](#).

Referenced by [clear_texture_context\(\)](#), [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), and [load_texture_without_alpha_channel\(\)](#).

6.44.4.4 textureID

```
GLuint Texture::textureID [private]
```

Definition at line 38 of file [Texture.h](#).

Referenced by [clear_texture_context\(\)](#), [get_id\(\)](#), [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), [load_texture_without_alpha_channel\(\)](#), and [use_texture\(\)](#).

6.44.4.5 width

```
int Texture::width [private]
```

Definition at line 39 of file [Texture.h](#).

Referenced by [clear_texture_context\(\)](#), [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), and [load_texture_without_alpha_channel\(\)](#).

6.44.4.6 wrapping_mode

```
std::shared_ptr<TextureWrappingMode> Texture::wrapping_mode [private]
```

Definition at line 41 of file [Texture.h](#).

Referenced by [load_SRGB_texture_with_alpha_channel\(\)](#), [load_SRGB_texture_without_alpha_channel\(\)](#), [load_texture_with_alpha_channel\(\)](#), and [load_texture_without_alpha_channel\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[Texture.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[Texture.cpp](#)

6.45 TextureWrappingMode Class Reference

```
#include <TextureWrappingMode.h>
```

Inheritance diagram for TextureWrappingMode:

Collaboration diagram for TextureWrappingMode:

Public Member Functions

- virtual void [activate \(\)=0](#)

6.45.1 Detailed Description

Definition at line 5 of file [TextureWrappingMode.h](#).

6.45.2 Member Function Documentation

6.45.2.1 activate()

```
virtual void TextureWrappingMode::activate ( ) [pure virtual]
```

Implemented in [ClampToEdgeMode](#), [MirroredRepeatMode](#), and [RepeatMode](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/[TextureWrappingMode.h](#)

6.46 Vertex Struct Reference

```
#include <Vertex.h>
```

Collaboration diagram for Vertex:

Public Member Functions

- [Vertex \(\)](#)
- [Vertex \(glm::vec3 pos, glm::vec3 normal, glm::vec3 color, glm::vec2 texture_coords\)](#)
- [glm::vec3 get_position \(\) const](#)
- [glm::vec3 get_normal \(\) const](#)
- [glm::vec3 get_color \(\) const](#)
- [glm::vec2 get_tex_coors \(\) const](#)
- [bool operator== \(const Vertex &other\) const](#)

Data Fields

- [glm::vec3 position](#)
- [glm::vec3 normal](#)
- [glm::vec3 color](#)
- [glm::vec2 texture_coords](#)

6.46.1 Detailed Description

Definition at line 5 of file [Vertex.h](#).

6.46.2 Constructor & Destructor Documentation

6.46.2.1 Vertex() [1/2]

```
Vertex::Vertex ( ) [inline]
```

Definition at line 13 of file [Vertex.h](#).

```
00014 {
00015
00016     this->position = glm::vec3(0);
00017     this->normal = glm::vec3(0);
00018     this->color = glm::vec3(0);
00019     this->texture_coords = glm::vec2(0);
00020 }
```

6.46.2.2 Vertex() [2/2]

```
Vertex::Vertex (
    glm::vec3 pos,
    glm::vec3 normal,
    glm::vec3 color,
    glm::vec2 texture_coords ) [inline]
```

Definition at line 22 of file [Vertex.h](#).

```
00023 {
00024
00025     this->position = pos;
00026     this->normal = normal;
00027     this->color = color;
00028     this->texture_coords = texture_coords;
00029 };
```

References [color](#), [normal](#), and [texture_coords](#).

6.46.3 Member Function Documentation

6.46.3.1 get_color()

```
glm::vec3 Vertex::get_color ( ) const [inline]
```

Definition at line 35 of file [Vertex.h](#).

```
00035 { return color; }
```

References [color](#).

6.46.3.2 get_normal()

```
glm::vec3 Vertex::get_normal ( ) const [inline]
```

Definition at line 33 of file [Vertex.h](#).

```
00033 { return normal; }
```

References [normal](#).

6.46.3.3 get_position()

```
glm::vec3 Vertex::get_position() const [inline]
```

Definition at line 31 of file [Vertex.h](#).
00031 { **return position;** }

References [position](#).

6.46.3.4 get_tex_coors()

```
glm::vec2 Vertex::get_tex_coors() const [inline]
```

Definition at line 37 of file [Vertex.h](#).
00037 { **return texture_coords;** }

References [texture_coords](#).

6.46.3.5 operator==()

```
bool Vertex::operator==(const Vertex & other) const [inline]
```

Definition at line 39 of file [Vertex.h](#).
00040 {
00041
00042 **return position == other.position && normal == other.normal && texture_coors ==**
00043 **other.texture_coors && color == other.color;**
00044 }

References [color](#), [normal](#), [position](#), and [texture_coords](#).

6.46.4 Field Documentation

6.46.4.1 color

```
glm::vec3 Vertex::color
```

Definition at line 10 of file [Vertex.h](#).

Referenced by [get_color\(\)](#), [operator==\(\)](#), and [Vertex\(\)](#).

6.46.4.2 normal

```
glm::vec3 Vertex::normal
```

Definition at line 9 of file [Vertex.h](#).

Referenced by [get_normal\(\)](#), [ObjLoader::load\(\)](#), [std::hash<Vertex>::operator\(\)\(\)](#), [operator==\(\)](#), and [Vertex\(\)](#).

6.46.4.3 position

```
glm::vec3 Vertex::position
```

Definition at line 8 of file [Vertex.h](#).

Referenced by [get_position\(\)](#), [ObjLoader::load\(\)](#), [std::hash<Vertex>::operator\(\)\(\)](#), and [operator==\(\)](#).

6.46.4.4 texture_coords

```
glm::vec2 Vertex::texture_coords
```

Definition at line 11 of file [Vertex.h](#).

Referenced by [get_tex_coors\(\)](#), [std::hash<Vertex>::operator\(\)\(\)](#), [operator==\(\)](#), and [Vertex\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[Vertex.h](#)

6.47 ViewFrustumCulling Class Reference

```
#include <ViewFrustumCulling.h>
```

Collaboration diagram for ViewFrustumCulling:

Data Structures

- struct [frustum_plane](#)

Public Member Functions

- [ViewFrustumCulling \(\)](#)
- bool [is_inside \(GLfloat ratio, std::shared_ptr< Camera > main_camera, std::shared_ptr< AABB > bounding_box, glm::mat4 model\)](#)
- void [render_view_frustum \(\)](#)
- [~ViewFrustumCulling \(\)](#)

Private Member Functions

- void `init` (std::vector< glm::vec3 > frustum_corner)
- bool `corners_outside_plane` (std::vector< glm::vec3 > aabb_corners, `frustum_plane` plane, GLuint outcode_pattern)
- GLfloat `plane_point_distance` (`frustum_plane` plane, glm::vec3 corner)
- void `update_frustum_param` (GLfloat `near_plane`, GLfloat `far_plane`, GLfloat `fov`, GLfloat `ratio`, std::shared_ptr< Camera > `main_camera`)

Private Attributes

- unsigned int `VBO`
- unsigned int `VAO`
- unsigned int `EBO`
- unsigned int `m_drawCount`
- GLfloat `near_plane`
- GLfloat `far_plane`
- GLfloat `fov`
- GLfloat `ratio`
- GLfloat `tan`
- GLfloat `near_height`
- GLfloat `near_width`
- GLfloat `far_height`
- GLfloat `far_width`
- std::shared_ptr< Camera > `main_camera`
- glm::vec3 `dir`
- glm::vec3 `near_center`
- glm::vec3 `far_center`
- glm::vec3 `near_top_left`
- glm::vec3 `near_top_right`
- glm::vec3 `near_bottom_left`
- glm::vec3 `near_bottom_right`
- glm::vec3 `far_top_left`
- glm::vec3 `far_top_right`
- glm::vec3 `far_bottom_left`
- glm::vec3 `far_bottom_right`
- `frustum_plane frustum_planes [NUM_FRUSTUM_PLANES]`

6.47.1 Detailed Description

Definition at line 15 of file [ViewFrustumCulling.h](#).

6.47.2 Constructor & Destructor Documentation

6.47.2.1 ViewFrustumCulling()

ViewFrustumCulling::ViewFrustumCulling ()

Definition at line 3 of file [ViewFrustumCulling.cpp](#).

```
00003   :
00004
00005     VBO(-1), VAO(-1), EBO(-1), m_drawCount(0),
00006     //we get that as input
00007     near_plane(0.f), far_plane(0.f), fov(0.f), ratio(0.f),
00008
00009     //calculate as soon as we become params
00010     tan(0.f), near_height(0.f), near_width(0.f), far_height(0.f), far_width(0.f), main_camera(),
00011
00012     dir(glm::vec3(0.f)), near_center(glm::vec3(0.f)), far_center(glm::vec3(0.f)),
00013
00014     near_top_left(glm::vec3(0.f)), near_top_right(glm::vec3(0.f)), near_bottom_left(glm::vec3(0.f)),
00015     near_bottom_right(glm::vec3(0.f)),
00016     far_top_left(glm::vec3(0.f)), far_top_right(glm::vec3(0.f)), far_bottom_left(glm::vec3(0.f)),
00017     far_bottom_right(glm::vec3(0.f))
00018 {
00019 }
```

6.47.2.2 ~ViewFrustumCulling()

ViewFrustumCulling::~ViewFrustumCulling ()

Definition at line 315 of file [ViewFrustumCulling.cpp](#).

```
00316 {
00317     glDeleteVertexArrays(1, &VAO);
00318     glDeleteBuffers(1, &VBO);
00319     glDeleteBuffers(1, &EBO);
00320 }
```

References [EBO](#), [VAO](#), and [VBO](#).

6.47.3 Member Function Documentation

6.47.3.1 corners_outside_plane()

```
bool ViewFrustumCulling::corners_outside_plane (
    std::vector< glm::vec3 > aabb_corners,
    frustum_plane plane,
    GLuint outcode_pattern ) [private]
```

Definition at line 74 of file [ViewFrustumCulling.cpp](#).

```
00075 {
00076
00077     GLint outcode = outcode_pattern;
00078
00079     for (int i = 0; i < static_cast<int>(aabb_corners.size()); i++) {
00080
00081         if (plane_point_distance(plane, aabb_corners[i]) < 0.0f) {
00082
00083             if (i == 0) {
00084                 outcode = outcode_pattern;
00085             } else {
00086                 outcode = outcode & outcode_pattern;
00087             }
00088
00089     }
00090 }
```

```

00088     } else {
00089         if (i == 0) {
00090             outcode = 0;
00091         } else {
00092             outcode = outcode & 0;
00093         }
00094     }
00095 }
00096
00097 return (outcode != 0) ? true : false;
00098 }
```

References [plane_point_distance\(\)](#).

Referenced by [is_inside\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.47.3.2 init()

```
void ViewFrustumCulling::init (
    std::vector< glm::vec3 > frustum_corner ) [private]
```

Definition at line 207 of file [ViewFrustumCulling.cpp](#).

```

00208 {
00209
00210 //unsigned int num_corners = 8;
00211 m_drawCount = 36; //num_corners;
00212
00213 float vertices[] = {
00214
00215     frustum_corner[0].x,
00216     frustum_corner[0].y,
00217     frustum_corner[0].z, // left bottom front
00218     frustum_corner[1].x,
00219     frustum_corner[1].y,
00220     frustum_corner[1].z, // left bottom back
00221     frustum_corner[2].x,
00222     frustum_corner[2].y,
00223     frustum_corner[2].z, // left top front
00224     frustum_corner[3].x,
00225     frustum_corner[3].y,
00226     frustum_corner[3].z, // left top back
00227     frustum_corner[4].x,
00228     frustum_corner[4].y,
00229     frustum_corner[4].z, // right bottom front
00230     frustum_corner[5].x,
00231     frustum_corner[5].y,
00232     frustum_corner[5].z, //right bottom back
00233     frustum_corner[6].x,
00234     frustum_corner[6].y,
00235     frustum_corner[6].z, //right top front
00236     frustum_corner[7].x,
00237     frustum_corner[7].y,
00238     frustum_corner[7].z //right top back
00239 };
00240
00241
00242 unsigned int indices[] = {
00243     // note that we start from 0!
00244     //left
00245     0,
00246     1,
00247     3,
00248     0,
00249     2,
00250     3,
00251     //right
00252     4,
00253     5,
00254     7,
00255     4,
00256     6,
00257     7,
00258     //top
00259     3,
00260     2,
00261     7,
```

```

00262     3,
00263     2,
00264     8,
00265 //bottom
00266     0,
00267     1,
00268     4,
00269     0,
00270     1,
00271     5,
00272 //back
00273     1,
00274     3,
00275     5,
00276     1,
00277     3,
00278     7,
00279 //front
00280     0,
00281     2,
00282     4,
00283     0,
00284     2,
00285     6,
00286 };
00287
00288 glGenVertexArrays(1, &VAO);
00289 glGenBuffers(1, &VBO);
00290 glGenBuffers(1, &EBO);
00291 // bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex
attributes(s).
00292 glBindVertexArray(VAO);
00293
00294 glBindBuffer(GL_ARRAY_BUFFER, VBO);
00295 glBindBuffer(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
00296
00297 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
00298 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
00299
00300 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
00301 glEnableVertexAttribArray(0);
00302
00303 // note that this is allowed, the call to glVertexAttribPointer registered VBO as the vertex
attribute's bound vertex buffer object so afterwards we can safely unbind
00304 glBindBuffer(GL_ARRAY_BUFFER, 0);
00305
00306 // remember: do NOT unbind the EBO while a VAO is active as the bound element buffer object IS
stored in the VAO; keep the EBO bound.
00307 //glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00308
00309 // You can unbind the VAO afterwards so other VAO calls won't accidentally modify this VAO, but this
rarely happens. Modifying other
00310 // VAOs requires a call to glBindVertexArray anyways so we generally don't unbind VAOs (nor VBOs)
when it's not directly necessary.
00311 glBindVertexArray(0);
00312 }

```

References [EBO](#), [m_drawCount](#), [VAO](#), and [VBO](#).

6.47.3.3 is_inside()

```

bool ViewFrustumCulling::is_inside (
    GLfloat ratio,
    std::shared_ptr< Camera > main_camera,
    std::shared_ptr< AABB > bounding_box,
    glm::mat4 model )

```

Definition at line 22 of file [ViewFrustumCulling.cpp](#).

```

00023 {
00024     GLfloat near_plane = main_camera->get_near_plane();
00025     GLfloat far_plane = main_camera->get_far_plane();
00026     GLfloat fov = main_camera->get_fov();
00027
00028     update_frustum_param(near_plane, far_plane, fov, ratio, main_camera);
00029
00030     std::vector<glm::vec3> aabb_corners = bounding_box->get_corners(model);

```

```

00031 //layout: [0]: near plane, [1] far plane, [2] up , [3] bottom , [4]: left ,
00032 //right [5]: [5]: right
00033 //outcodes (binary) : 100000 , 010000 , 000100 , 001000 , 000010, 000001
00034 //outcodes (dezi) : 32 , 16 , 4 , 8
00035 , 2 , 1
00036 bool result = true;
00037
00038 GLint outcode_near_plane = 32;
00039 GLint outcode_far_plane = 16;
00040 GLint outcode_up = 4;
00041 GLint outcode_bottom = 8;
00042 GLint outcode_left = 2;
00043 GLint outcode_right = 1;
00044 //GLint outcode;
00045
00046 GLint outcodes_pattern[NUM_FRUSTUM_PLANES] = { outcode_near_plane, outcode_far_plane, outcode_up,
00047 outcode_bottom, outcode_left, outcode_right };
00048 for (int i = 0; i < NUM_FRUSTUM_PLANES; i++) {
00049
00050 frustum_plane plane = frustum_planes[i];
00051
00052 if (corners_outside_plane(aabb_corners, plane, outcodes_pattern[i])) {
00053
00054 result = false;
00055 break;
00056 }
00057 }
00058
00059 return result;
00060 }

```

References [corners_outside_plane\(\)](#), [far_plane](#), [fov](#), [frustum_planes](#), [main_camera](#), [near_plane](#), [NUM_FRUSTUM_PLANES](#), [ratio](#), and [update_frustum_param\(\)](#).

Here is the call graph for this function:

6.47.3.4 `plane_point_distance()`

```

GLfloat ViewFrustumCulling::plane_point_distance (
    frustum_plane plane,
    glm::vec3 corner ) [private]

```

Definition at line 100 of file [ViewFrustumCulling.cpp](#).

```

00101 {
00102     GLfloat result = 0.0f;
00103
00104     glm::vec3 plane_normal = plane.normal;
00105     glm::vec3 plane_position = plane.position;
00106
00107     GLfloat d = glm::dot(plane_normal, plane_position);
00108
00109     result = (glm::dot(plane_normal, corner) - d) / glm::length(plane_normal);
00110
00111     return result;
00112 }

```

References [ViewFrustumCulling::frustum_plane::normal](#), and [ViewFrustumCulling::frustum_plane::position](#).

Referenced by [corners_outside_plane\(\)](#).

Here is the caller graph for this function:

6.47.3.5 render_view_frustum()

```
void ViewFrustumCulling::render_view_frustum ( )
```

Definition at line 62 of file [ViewFrustumCulling.cpp](#).

```
00063 {
00064     // seeing as we only have a single VAO there's no need to bind it every time,
00065     // but we'll do so to keep things a bit more organized
00066     glBindVertexArray(VAO);
00067     //glDrawArrays(GL_TRIANGLES, 0, 6);
00068     glDrawElements(GL_TRIANGLES, m_drawCount, GL_UNSIGNED_INT, 0);
00069
00070     //unbind all again
00071     glBindVertexArray(0);
00072 }
```

References [m_drawCount](#), and [VAO](#).

6.47.3.6 update_frustum_param()

```
void ViewFrustumCulling::update_frustum_param (
    GLfloat near_plane,
    GLfloat far_plane,
    GLfloat fov,
    GLfloat ratio,
    std::shared_ptr< Camera > main_camera ) [private]
```

Definition at line 114 of file [ViewFrustumCulling.cpp](#).

```
00115 {
00116     this->near_plane = near_plane;
00117     this->far_plane = far_plane;
00118     this->fov = fov;
00119     this->ratio = ratio;
00120
00121     tan = glm::tan(glm::radians(fov) * 0.5f);
00122     near_height = near_plane * tan;
00123     near_width = near_height * ratio;
00124     far_height = far_plane * tan;
00125     far_width = far_height * ratio;
00126
00127     this->main_camera = main_camera;
00128
00129     near_center = main_camera->get_camera_position() + main_camera->get_camera_direction() * near_plane;
00130     far_center = main_camera->get_camera_position() + main_camera->get_camera_direction() * far_plane;
00131
00132     glm::vec3 aux_position, aux, aux_normal;
00133
00134     //layout: [0]: near plane
00135     frustum_planes[0].normal = main_camera->get_camera_direction();
00136     frustum_planes[0].position = near_center;
00137
00138     // [1] far plane
00139     frustum_planes[1].normal = -main_camera->get_camera_direction();
00140     frustum_planes[1].position = far_center;
00141
00142     aux_position = near_center + main_camera->get_up_axis() * near_height;
00143     aux = aux_position - main_camera->get_camera_position();
00144     aux = glm::normalize(aux);
00145     aux_normal = glm::cross(aux, main_camera->get_right_axis());
00146
00147     // [2] top
00148     frustum_planes[2].normal = normalize(aux_normal);
00149     frustum_planes[2].position = aux_position;
00150
00151     aux_position = near_center - main_camera->get_up_axis() * near_height;
00152     aux = aux_position - main_camera->get_camera_position();
00153     aux = glm::normalize(aux);
00154     aux_normal = glm::cross(main_camera->get_right_axis(), aux);
00155
00156     // [3] bottom
00157     frustum_planes[3].normal = normalize(aux_normal);
00158     frustum_planes[3].position = aux_position;
```

```

00159     aux_position = near_center - main_camera->get_right_axis() * near_width;
00160     aux = aux_position - main_camera->get_camera_position();
00161     aux = glm::normalize(aux);
00162     aux_normal = glm::cross(aux, main_camera->get_up_axis());
00163
00164     // [4]: left
00165     frustum_planes[4].normal = normalize(aux_normal);
00166     frustum_planes[4].position = aux_position;
00167
00168     aux_position = near_center + main_camera->get_right_axis() * near_width;
00169     aux = aux_position - main_camera->get_camera_position();
00170     aux = glm::normalize(aux);
00171     aux_normal = glm::cross(main_camera->get_up_axis(), aux);
00172
00173     // [5]: right
00174     frustum_planes[5].normal = normalize(aux_normal);
00175     frustum_planes[5].position = aux_position;
00176
00177     std::vector<glm::vec3> frustum_corners;
00178
00179     //frustum_corners.push_back(near_center - main_camera->get_right_axis() * near_width -
00180     // main_camera->get_up_axis() * near_height); // left bottom front
00181     //
00182     //frustum_corners.push_back(far_center - main_camera->get_right_axis() * far_width -
00183     // main_camera->get_up_axis() * far_height); // left bottom back
00184     //
00185     //frustum_corners.push_back(near_center - main_camera->get_right_axis() * near_width +
00186     // main_camera->get_up_axis() * near_height); // left top front
00187     //
00188     //frustum_corners.push_back(far_center - main_camera->get_right_axis() * far_width +
00189     // main_camera->get_up_axis() * far_height); // left top back
00190     //
00191     //frustum_corners.push_back(near_center + main_camera->get_right_axis() * near_width -
00192     // main_camera->get_up_axis() * near_height); // right bottom front
00193     //
00194     //frustum_corners.push_back(far_center + main_camera->get_right_axis() * far_width -
00195     // main_camera->get_up_axis() * far_height); // right bottom back
00196     //
00197     //frustum_corners.push_back(near_center + main_camera->get_right_axis() * near_width +
00198     // main_camera->get_up_axis() * near_height); // right top front
00199     //
00200     //frustum_corners.push_back(far_center + main_camera->get_right_axis() * far_width +
00201     // main_camera->get_up_axis() * far_height); // right top back
00202
00203
00204     //init(frustum_corners);
00205 }

```

References `far_center`, `far_height`, `far_plane`, `far_width`, `fov`, `frustum_planes`, `main_camera`, `near_center`, `near_height`, `near_plane`, `near_width`, `ViewFrustumCulling::frustum_plane::normal`, `ViewFrustumCulling::frustum_plane::position`, `ratio`, and `tan`.

Referenced by `is_inside()`.

Here is the caller graph for this function:

6.47.4 Field Documentation

6.47.4.1 dir

```
glm::vec3 ViewFrustumCulling::dir [private]
```

Definition at line 39 of file `ViewFrustumCulling.h`.

6.47.4.2 EBO

```
unsigned int ViewFrustumCulling::EBO [private]
```

Definition at line 27 of file [ViewFrustumCulling.h](#).

Referenced by [init\(\)](#), and [~ViewFrustumCulling\(\)](#).

6.47.4.3 far_bottom_left

```
glm::vec3 ViewFrustumCulling::far_bottom_left [private]
```

Definition at line 45 of file [ViewFrustumCulling.h](#).

6.47.4.4 far_bottom_right

```
glm::vec3 ViewFrustumCulling::far_bottom_right [private]
```

Definition at line 45 of file [ViewFrustumCulling.h](#).

6.47.4.5 far_center

```
glm::vec3 ViewFrustumCulling::far_center [private]
```

Definition at line 39 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.6 far_height

```
GLfloat ViewFrustumCulling::far_height [private]
```

Definition at line 35 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.7 far_plane

```
GLfloat ViewFrustumCulling::far_plane [private]
```

Definition at line 32 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.8 far_top_left

```
glm::vec3 ViewFrustumCulling::far_top_left [private]
```

Definition at line 45 of file [ViewFrustumCulling.h](#).

6.47.4.9 far_top_right

```
glm::vec3 ViewFrustumCulling::far_top_right [private]
```

Definition at line 45 of file [ViewFrustumCulling.h](#).

6.47.4.10 far_width

```
GLfloat ViewFrustumCulling::far_width [private]
```

Definition at line 35 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.11 fov

```
GLfloat ViewFrustumCulling::fov [private]
```

Definition at line 32 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.12 frustum_planes

```
frustum_plane ViewFrustumCulling::frustum_planes[NUM_FRUSTUM_PLANES] [private]
```

Definition at line 55 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.13 m_drawCount

```
unsigned int ViewFrustumCulling::m_drawCount [private]
```

Definition at line 29 of file [ViewFrustumCulling.h](#).

Referenced by [init\(\)](#), and [render_view_frustum\(\)](#).

6.47.4.14 main_camera

```
std::shared_ptr<Camera> ViewFrustumCulling::main_camera [private]
```

Definition at line 37 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.15 near_bottom_left

```
glm::vec3 ViewFrustumCulling::near_bottom_left [private]
```

Definition at line 43 of file [ViewFrustumCulling.h](#).

6.47.4.16 near_bottom_right

```
glm::vec3 ViewFrustumCulling::near_bottom_right [private]
```

Definition at line 43 of file [ViewFrustumCulling.h](#).

6.47.4.17 `near_center`

```
glm::vec3 ViewFrustumCulling::near_center [private]
```

Definition at line 39 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.18 `near_height`

```
GLfloat ViewFrustumCulling::near_height [private]
```

Definition at line 35 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.19 `near_plane`

```
GLfloat ViewFrustumCulling::near_plane [private]
```

Definition at line 32 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.20 `near_top_left`

```
glm::vec3 ViewFrustumCulling::near_top_left [private]
```

Definition at line 43 of file [ViewFrustumCulling.h](#).

6.47.4.21 `near_top_right`

```
glm::vec3 ViewFrustumCulling::near_top_right [private]
```

Definition at line 43 of file [ViewFrustumCulling.h](#).

6.47.4.22 near_width

```
GLfloat ViewFrustumCulling::near_width [private]
```

Definition at line 35 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.23 ratio

```
GLfloat ViewFrustumCulling::ratio [private]
```

Definition at line 32 of file [ViewFrustumCulling.h](#).

Referenced by [is_inside\(\)](#), and [update_frustum_param\(\)](#).

6.47.4.24 tan

```
GLfloat ViewFrustumCulling::tan [private]
```

Definition at line 35 of file [ViewFrustumCulling.h](#).

Referenced by [update_frustum_param\(\)](#).

6.47.4.25 VAO

```
unsigned int ViewFrustumCulling::VAO [private]
```

Definition at line 27 of file [ViewFrustumCulling.h](#).

Referenced by [init\(\)](#), [render_view_frustum\(\)](#), and [~ViewFrustumCulling\(\)](#).

6.47.4.26 VBO

```
unsigned int ViewFrustumCulling::VBO [private]
```

Definition at line 27 of file [ViewFrustumCulling.h](#).

Referenced by [init\(\)](#), and [~ViewFrustumCulling\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ViewFrustumCulling.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/scene/[ViewFrustumCulling.cpp](#)

6.48 Window Class Reference

```
#include <Window.h>
```

Collaboration diagram for Window:

Public Member Functions

- `Window ()`
- `Window (GLint window_width, GLint window_height)`
- `bool get_should_close ()`
- `void swap_buffers ()`
- `int initialize ()`
- `void update_viewport ()`
- `GLuint get_buffer_width () const`
- `GLuint get_buffer_height () const`
- `GLfloat get_x_change ()`
- `GLfloat get_y_change ()`
- `GLFWwindow * get_window ()`
- `bool * get_keys ()`
- `~Window ()`

Private Member Functions

- `void init_callbacks ()`

Static Private Member Functions

- `static void framebuffer_size_callback (GLFWwindow *window, int width, int height)`
- `static void key_callback (GLFWwindow *window, int key, int code, int action, int mode)`
- `static void mouse_callback (GLFWwindow *window, double x_pos, double y_pos)`
- `static void mouse_button_callback (GLFWwindow *window, int button, int action, int mods)`

Private Attributes

- `GLFWwindow * main_window`
- `GLint window_width`
- `GLint window_height`
- `bool keys [1024]`
- `GLfloat last_x`
- `GLfloat last_y`
- `GLfloat x_change`
- `GLfloat y_change`
- `bool mouse_first_moved`
- `GLint window_buffer_width`
- `GLint window_buffer_height`

6.48.1 Detailed Description

Definition at line 7 of file [Window.h](#).

6.48.2 Constructor & Destructor Documentation

6.48.2.1 Window() [1/2]

```
Window::Window ( )
```

Definition at line 4 of file [Window.cpp](#).

```
00004           : window_width(800), window_height(600), x_change(0.0f), y_change(0.0f)
00005 {
00006
00007     // all keys non-pressed in the beginning
00008     for (size_t i = 0; i < 1024; i++) {
00009         keys[i] = 0;
00010     }
00011
00012     initialize();
00013 }
```

References [initialize\(\)](#), and [keys](#).

Here is the call graph for this function:

6.48.2.2 Window() [2/2]

```
Window::Window (
    GLint window_width,
    GLint window_height )
```

Definition at line 16 of file [Window.cpp](#).

```
00016           :
00017
00018     window_width(window_width), window_height(window_height), x_change(0.0f), y_change(0.0f)
00019 {
00020
00021     // all keys non-pressed in the beginning
00022     for (size_t i = 0; i < 1024; i++) {
00023         keys[i] = 0;
00024     }
00025
00026     initialize();
00027 }
```

References [initialize\(\)](#), and [keys](#).

Here is the call graph for this function:

6.48.2.3 ~Window()

```
Window::~Window ( )
```

Definition at line 121 of file [Window.cpp](#).

```
00122 {
00123     glfwDestroyWindow(main_window);
00124     glfwTerminate();
00125 }
```

References [main_window](#).

6.48.3 Member Function Documentation

6.48.3.1 framebuffer_size_callback()

```
void Window::framebuffer_size_callback (
    GLFWwindow * window,
    int width,
    int height ) [static], [private]
```

Definition at line 136 of file [Window.cpp](#).

```
00136 { }
```

Referenced by [init_callbacks\(\)](#).

Here is the caller graph for this function:

6.48.3.2 get_buffer_height()

```
GLuint Window::get_buffer_height () const [inline]
```

Definition at line 21 of file [Window.h](#).

```
00021 { return window_buffer_height; }
```

References [window_buffer_height](#).

6.48.3.3 get_buffer_width()

```
GLuint Window::get_buffer_width () const [inline]
```

Definition at line 20 of file [Window.h](#).

```
00020 { return window_buffer_width; }
```

References [window_buffer_width](#).

6.48.3.4 get_keys()

```
bool * Window::get_keys () [inline]
```

Definition at line 28 of file [Window.h](#).

```
00028 { return keys; }
```

References [keys](#).

6.48.3.5 get_should_close()

```
bool Window::get_should_close ( ) [inline]
```

Definition at line 12 of file [Window.h](#).

```
00012 { return glfwWindowShouldClose(main_window); }
```

References [main_window](#).

6.48.3.6 get_window()

```
GLFWwindow * Window::get_window ( ) [inline]
```

Definition at line 26 of file [Window.h](#).

```
00026 { return main_window; }
```

References [main_window](#).

6.48.3.7 get_x_change()

```
GLfloat Window::get_x_change ( )
```

Definition at line 107 of file [Window.cpp](#).

```
00108 {  
00109     GLfloat the_change = x_change;  
00110     x_change = 0.0f;  
00111     return the_change;  
00112 }
```

References [x_change](#).

6.48.3.8 get_y_change()

```
GLfloat Window::get_y_change ( )
```

Definition at line 114 of file [Window.cpp](#).

```
00115 {  
00116     GLfloat the_change = y_change;  
00117     y_change = 0.0f;  
00118     return the_change;  
00119 }
```

References [y_change](#).

6.48.3.9 init_callbacks()

```
void Window::init_callbacks ( ) [private]
```

Definition at line 127 of file [Window.cpp](#).

```
00128 {
00129     //TODO: remember this section for our later game logic
00130     //for the space ship to fly around
00131     glfwSetKeyCallback(main_window, key_callback);
00132     glfwSetMouseButtonCallback(main_window, mouse_button_callback);
00133     glfwSetFramebufferSizeCallback(main_window, framebuffer_size_callback);
00134 }
```

References [framebuffer_size_callback\(\)](#), [key_callback\(\)](#), [main_window](#), and [mouse_button_callback\(\)](#).

Referenced by [initialize\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.48.3.10 initialize()

```
int Window::initialize ( )
```

Definition at line 29 of file [Window.cpp](#).

```
00030 {
00031
00032     if (!glfwInit ()) {
00033
00034         printf ("GLFW Init failed!");
00035         glfwTerminate ();
00036         return 1;
00037     }
00038
00039     // setup glfw window properties
00040
00041     //lets work with nothing older than version 3
00042     glfwWindowHint (GLFW_CONTEXT_VERSION_MAJOR, 4);
00043     glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR, 6);
00044
00045     // core profile = no backward compatibility
00046     glfwWindowHint (GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00047
00048     // allow forward compatibility
00049     glfwWindowHint (GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00050
00051     //allow it to resize
00052     glfwWindowHint (GLFW_RESIZABLE, GLFW_TRUE);
00053
00054
00055 #ifdef NDEBUG
00056     glfwWindowHint (GLFW_OPENGL_DEBUG_CONTEXT, false);
00057 #else
00058     glfwWindowHint (GLFW_OPENGL_DEBUG_CONTEXT, true);
00059 #endif
00060
00061     //retrieve new window
00062     main_window = glfwCreateWindow (window_width, window_height, "\\_\_ Epic graphics from hell \\_\_",
00063                                     NULL, NULL);
00064
00065     if (!main_window) {
00066
00067         printf ("GLFW Window creation failed!");
00068         glfwTerminate ();
00069         return 1;
00070     }
00071
00072     // get buffer size information
00073     glfwGetFramebufferSize (main_window, &window_buffer_width, &window_buffer_height);
00074
00075     // set context for GLEW to use
00076     glfwMakeContextCurrent (main_window);
00077
00078     if (!gladLoadGLLoader ((GLADloadproc) glfwGetProcAddress)) {
00079         std::cout << "Failed to initialize OpenGL context" << std::endl;
00080         return -1;
00081     }
```

```

00081     //disabling frame limits
00082     glfwSwapInterval(0);
00084
00085     //Handle key + mouse Input
00086     init_callbacks();
00087     glfwSetInputMode(main_window, GLFW_CURSOR, GLFW_CURSOR_NORMAL);
00088
00089     glEnable(GL_DEPTH_TEST);
00090
00091     // setup viewport size
00092     glViewport(0, 0, window_buffer_width, window_buffer_height);
00093
00094     glfwSetWindowUserPointer(main_window, this);
00095
00096     return 0;
00097 }
```

References [init_callbacks\(\)](#), [main_window](#), [window_buffer_height](#), [window_buffer_width](#), [window_height](#), and [window_width](#).

Referenced by [Window\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.48.3.11 key_callback()

```

void Window::key_callback (
    GLFWwindow * window,
    int key,
    int code,
    int action,
    int mode ) [static], [private]
```

Definition at line 138 of file [Window.cpp](#).

```

00139 {
00140     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00141
00142     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS) {
00143         glfwSetWindowShouldClose(window, GL_TRUE);
00144     }
00145
00146     if (key >= 0 && key < 1024) {
00147         if (action == GLFW_PRESS) {
00148             the_window->keys[key] = true;
00149
00150         } else if (action == GLFW_RELEASE) {
00151             the_window->keys[key] = false;
00152         }
00153     }
00154 }
```

References [keys](#).

Referenced by [init_callbacks\(\)](#).

Here is the caller graph for this function:

6.48.3.12 mouse_button_callback()

```
void Window::mouse_button_callback (
    GLFWwindow * window,
    int button,
    int action,
    int mods ) [static], [private]
```

Definition at line 176 of file [Window.cpp](#).

```
00177 {
00178     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00179
00180     if ((action == GLFW_PRESS) && (button == GLFW_MOUSE_BUTTON_RIGHT)) {
00181         glfwSetCursorPosCallback(window, mouse_callback);
00182     } else {
00183         the_window->mouse_first_moved = true;
00184         glfwSetCursorPosCallback(window, NULL);
00185     }
00186 }
```

References [mouse_callback\(\)](#), and [mouse_first_moved](#).

Referenced by [init_callbacks\(\)](#).

Here is the call graph for this function: Here is the caller graph for this function:

6.48.3.13 mouse_callback()

```
void Window::mouse_callback (
    GLFWwindow * window,
    double x_pos,
    double y_pos ) [static], [private]
```

Definition at line 156 of file [Window.cpp](#).

```
00157 {
00158     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00159
00160     // need to handle first occurrence of a mouse moving event
00161     if (the_window->mouse_first_moved) {
00162         the_window->last_x = (float)x_pos;
00163         the_window->last_y = (float)y_pos;
00164         the_window->mouse_first_moved = false;
00165     }
00166
00167     the_window->x_change = (float)(x_pos - the_window->last_x);
00168     // take care of correct subtraction :
00169     the_window->y_change = (float)(the_window->last_y - y_pos);
00170
00171     //update params
00172     the_window->last_x = (float)x_pos;
00173     the_window->last_y = (float)y_pos;
00174 }
```

References [last_x](#), [last_y](#), [mouse_first_moved](#), [x_change](#), and [y_change](#).

Referenced by [mouse_button_callback\(\)](#).

Here is the caller graph for this function:

6.48.3.14 swap_buffers()

```
void Window::swap_buffers ( ) [inline]
```

Definition at line 13 of file [Window.h](#).

```
00013 { glfwSwapBuffers(main_window); }
```

References [main_window](#).

6.48.3.15 update_viewport()

```
void Window::update_viewport ( )
```

Definition at line 99 of file [Window.cpp](#).

```
00100 {  
00101  
00102     glfwGetFramebufferSize(main\_window, &window\_buffer\_width, &window\_buffer\_height);  
00103     // setup viewport size  
00104     glViewport(0, 0, window\_buffer\_width, window\_buffer\_height);  
00105 }
```

References [main_window](#), [window_buffer_height](#), and [window_buffer_width](#).

6.48.4 Field Documentation

6.48.4.1 keys

```
bool Window::keys[1024] [private]
```

Definition at line 37 of file [Window.h](#).

Referenced by [get_keys\(\)](#), [key_callback\(\)](#), and [Window\(\)](#).

6.48.4.2 last_x

```
GLfloat Window::last_x [private]
```

Definition at line 38 of file [Window.h](#).

Referenced by [mouse_callback\(\)](#).

6.48.4.3 last_y

```
GLfloat Window::last_y [private]
```

Definition at line 39 of file [Window.h](#).

Referenced by [mouse_callback\(\)](#).

6.48.4.4 main_window

```
GLFWwindow* Window::main_window [private]
```

Definition at line 33 of file [Window.h](#).

Referenced by [get_should_close\(\)](#), [get_window\(\)](#), [init_callbacks\(\)](#), [initialize\(\)](#), [swap_buffers\(\)](#), [update_viewport\(\)](#), and [~Window\(\)](#).

6.48.4.5 mouse_first_moved

```
bool Window::mouse_first_moved [private]
```

Definition at line 42 of file [Window.h](#).

Referenced by [mouse_button_callback\(\)](#), and [mouse_callback\(\)](#).

6.48.4.6 window_buffer_height

```
GLint Window::window_buffer_height [private]
```

Definition at line 45 of file [Window.h](#).

Referenced by [get_buffer_height\(\)](#), [initialize\(\)](#), and [update_viewport\(\)](#).

6.48.4.7 window_buffer_width

```
GLint Window::window_buffer_width [private]
```

Definition at line 45 of file [Window.h](#).

Referenced by [get_buffer_width\(\)](#), [initialize\(\)](#), and [update_viewport\(\)](#).

6.48.4.8 window_height

```
GLint Window::window_height [private]
```

Definition at line 35 of file [Window.h](#).

Referenced by [initialize\(\)](#).

6.48.4.9 window_width

```
GLint Window::window_width [private]
```

Definition at line 35 of file [Window.h](#).

Referenced by [initialize\(\)](#).

6.48.4.10 x_change

```
GLfloat Window::x_change [private]
```

Definition at line 40 of file [Window.h](#).

Referenced by [get_x_change\(\)](#), and [mouse_callback\(\)](#).

6.48.4.11 y_change

```
GLfloat Window::y_change [private]
```

Definition at line 41 of file [Window.h](#).

Referenced by [get_y_change\(\)](#), and [mouse_callback\(\)](#).

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

- C:/Users/jonas/Desktop/GraphicEngine/Src/window/[Window.h](#)
- C:/Users/jonas/Desktop/GraphicEngine/Src/window/[Window.cpp](#)

Chapter 7

File Documentation

7.1 C:/Users/jonas/Desktop/GraphicEngine/Src/app/App.cpp File Reference

```
#include <thread>
#include <mutex>
#include <memory>
#include <vector>
#include <string>
#include <iostream>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include "File.h"
#include "GUI.h"
#include "LoadingScreen.h"
#include "Renderer.h"
#include "Scene.h"
#include "Window.h"
#include "Camera.h"
#include "GlobalValues.h"
#include "host_device_shared.h"
#include "DebugApp.h"
```

Include dependency graph for App.cpp:

Functions

- int `main ()`

7.1.1 Function Documentation

7.1.1.1 main()

```
int main ( )
```

Definition at line 26 of file App.cpp.

```
00027 {
00028
00029     bool loading_screen_finished = false;
00030
00031     GLint window_width = 1200;
00032     GLint window_height = 800;
00033
00034     // make sure to initialize window first
00035     // this will create opengl context!
00036     std::shared_ptr<Window> main_window = std::make_shared<Window>(window_width, window_height);
00037
00038     DebugApp debugCallbacks;
00039
00040     Renderer renderer(window_width, window_height);
00041
00042     GUI gui;
00043     gui.init(main_window);
00044
00045     LoadingScreen loading_screen;
00046     loading_screen.init();
00047
00048     std::shared_ptr<Camera> main_camera = std::make_shared<Camera>();
00049
00050     std::shared_ptr<Scene> scene = std::make_shared<Scene>(main_camera, main_window);
00051
00052     // load scene in another thread than the rendering thread; would block otherwise
00053     std::thread t1 = scene->spawn();
00054     t1.detach();
00055
00056     GLfloat delta_time = 0.0f;
00057     GLfloat last_time = 0.0f;
00058
00059     //enable depth testing
00060     glEnable(GL_DEPTH_TEST);
00061
00062
00063     while (!main_window->get_should_close()) {
00064
00065         glViewport(0, 0, window_width, window_height);
00066
00067         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
00068         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00069
00070         //we need the projection matrix, just use glm::perspective function
00071         glm::mat4 projection_matrix = glm::perspectiveFov(
00072             glm::radians(main_camera->get_fov()), (GLfloat)window_width, (GLfloat)window_height,
00073             main_camera->get_near_plane(), main_camera->get_far_plane());
00074
00075         //we should make the application independent of processor speed :)
00076         // take time into account is crucial
00077         // concept of delta time: https://bell0bytes.eu/keeping-track-of-time/
00078         GLfloat now = (float)glfwGetTime();
00079         delta_time = now - last_time;
00080         last_time = now;
00081
00082         //poll all events incoming from user
00083         glfwPollEvents();
00084
00085         // handle events for the camera
00086         main_camera->key_control(main_window->get_keys(), delta_time);
00087         main_camera->mouse_control(main_window->get_x_change(), main_window->get_y_change());
00088
00089         if (scene->is_loaded()) {
00090
00091             if (!loading_screen_finished) loading_screen_finished = true;
00092
00093             if (!scene->get_context_setup()) scene->setup_game_object_context();
00094
00095             renderer.drawFrame(main_camera, scene, projection_matrix, delta_time);
00096
00097         } else {
00098             // play the audio
00099             //SoundEngine->play2D("Audio/Red_Dead_Redemption_2 _Loading_Screen.mp3", true); //
00100             loading_screen.render();
00101         }
00102
00103         bool shader_hot_reload_triggered = false;
00104         gui.render(!scene->is_loaded(), scene->get_progress(), shader_hot_reload_triggered);
00105
00106         if (shader_hot_reload_triggered) renderer.reload_shader_programs();
```

```

00106     gui.update_user_input(scene);
00107
00108     main_window->update_viewport();
00109     GLuint new_window_width = main_window->get_buffer_width();
00110     GLuint new_window_height = main_window->get_buffer_height();
00111
00112     if ((static_cast<GLint>(new_window_width) == window_width && static_cast<GLint>(new_window_height)
00113 == window_height) == false) {
00114
00115         window_height = new_window_height;
00116         window_width = new_window_width;
00117         renderer.update_window_params(window_width, window_height);
00118     }
00119
00120     main_window->swap_buffers();
00121 }
00122 }
```

References [Renderer::drawFrame\(\)](#), [LoadingScreen::init\(\)](#), [GUI::init\(\)](#), [Renderer::reload_shader_programs\(\)](#), [LoadingScreen::render\(\)](#), [GUI::render\(\)](#), [GUI::update_user_input\(\)](#), and [Renderer::update_window_params\(\)](#).

Here is the call graph for this function:

7.2 App.cpp

[Go to the documentation of this file.](#)

```

00001 // include ability to execute threads
00002 #include <thread>
00003 #include <mutex>
00004 #include <memory>
00005 #include <vector>
00006 #include <string>
00007 #include <iostream>
00008
00009 #include <glad/glad.h>
00010 #include <GLFW/glfw3.h>
00011
00012 #include "File.h"
00013 #include "GUI.h"
00014 #include "LoadingScreen.h"
00015 #include "Renderer.h"
00016
00017 //all scene/game logic/ game object related stuff
00018 #include "Scene.h"
00019 #include "Window.h"
00020 #include "Camera.h"
00021
00022 #include "GlobalValues.h"
00023 #include "host_device_shared.h"
00024 #include "DebugApp.h"
00025
00026 int main()
00027 {
00028
00029     bool loading_screen_finished = false;
00030
00031     GLint window_width = 1200;
00032     GLint window_height = 800;
00033
00034     // make sure to initialize window first
00035     // this will create opengl context!
00036     std::shared_ptr<Window> main_window = std::make_shared<Window>(window_width, window_height);
00037
00038     DebugApp debugCallbacks;
00039
00040     Renderer renderer(window_width, window_height);
00041
00042     GUI gui;
00043     gui.init(main_window);
00044
00045     LoadingScreen loading_screen;
00046     loading_screen.init();
00047
00048     std::shared_ptr<Camera> main_camera = std::make_shared<Camera>();
00049
00050     std::shared_ptr<Scene> scene = std::make_shared<Scene>(main_camera, main_window);
00051
00052     // load scene in an other thread than the rendering thread; would block otherwise
```

```

00053     std::thread t1 = scene->spawn();
00054     t1.detach();
00055
00056     GLfloat delta_time = 0.0f;
00057     GLfloat last_time = 0.0f;
00058
00059     //enable depth testing
00060     glEnable(GL_DEPTH_TEST);
00061
00062
00063     while (!main_window->get_should_close()) {
00064
00065         glViewport(0, 0, window_width, window_height);
00066
00067         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
00068         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00069
00070         //we need the projection matrix, just use glm::perspective function
00071         glm::mat4 projection_matrix = glm::perspectiveFov(
00072             glm::radians(main_camera->get_fov()), (GLfloat)window_width, (GLfloat)window_height,
00073             main_camera->get_near_plane(), main_camera->get_far_plane());
00074
00075         //we should make the application independent of processor speed :)
00076         // take time into account is crucial
00077         // concept of delta time: https://bell0bytes.eu/keeping-track-of-time/
00078         GLfloat now = (float)glfwGetTime();
00079         delta_time = now - last_time;
00080         last_time = now;
00081
00082         //poll all events incoming from user
00083         glfwPollEvents();
00084
00085         // handle events for the camera
00086         main_camera->key_control(main_window->get_keys(), delta_time);
00087         main_camera->mouse_control(main_window->get_x_change(), main_window->get_y_change());
00088
00089         if (scene->is_loaded()) {
00090
00091             if (!loading_screen_finished) loading_screen_finished = true;
00092
00093             if (!scene->get_context_setup()) scene->setup_game_object_context();
00094
00095             renderer.drawFrame(main_camera, scene, projection_matrix, delta_time);
00096
00097         } else {
00098             // play the audio
00099             //SoundEngine->play2D("Audio/Red_Dead_Redemption_2 _Loading_Screen.mp3", true); //
00100             loading_screen.render();
00101
00102             bool shader_hot_reload_triggered = false;
00103             gui.render(!scene->is_loaded(), scene->get_progress(), shader_hot_reload_triggered);
00104
00105             if (shader_hot_reload_triggered) renderer.reload_shader_programs();
00106
00107             gui.update_user_input(scene);
00108
00109             main_window->update_viewport();
00110             GLuint new_window_width = main_window->get_buffer_width();
00111             GLuint new_window_height = main_window->get_buffer_height();
00112
00113             if ((static_cast<GLint>(new_window_width) == window_width && static_cast<GLint>(new_window_height)
00114             == window_height) == false) {
00115
00116                 window_height = new_window_height;
00117                 window_width = new_window_width;
00118                 renderer.update_window_params(window_width, window_height);
00119             }
00120             main_window->swap_buffers();
00121         }
00122     }

```

7.3 C:/Users/jonas/Desktop/GraphicEngine/Src/bindings.h File Reference

```
#include "host_device_shared.h"
#include "GlobalValues.h"
```

Include dependency graph for bindings.h: This graph shows which files directly or indirectly include this file:

Macros

- `#define MODEL_TEXTURES_SLOT 0`
- `#define SKYBOX_TEXTURES_SLOT MODEL_TEXTURES_SLOT + MAX_TEXTURE_COUNT`
- `#define GBUFFER_TEXTURES_SLOT SKYBOX_TEXTURES_SLOT + 1`
- `#define D_LIGHT_SHADOW_TEXTURES_SLOT GBUFFER_TEXTURES_SLOT + G_BUFFER_SIZE`
- `#define P_LIGHT_SHADOW_TEXTURES_SLOT D_LIGHT_SHADOW_TEXTURES_SLOT + 1`
- `#define NOISE_128D_TEXTURES_SLOT P_LIGHT_SHADOW_TEXTURES_SLOT + MAX_POINT_LIGHTS`
- `#define NOISE_32D_TEXTURES_SLOT NOISE_128D_TEXTURES_SLOT + 1`
- `#define NOISE_CELL_POSITIONS_SLOT NOISE_32D_TEXTURES_SLOT + 1`
- `#define RANDOM_NUMBERS_SLOT NOISE_CELL_POSITIONS_SLOT + NUM_CELL_POSITIONS`
- `#define NOISE_128D_IMAGE_SLOT 0`
- `#define NOISE_32D_IMAGE_SLOT 1`
- `#define STORAGE_BUFFER_MATERIAL_ID_BINDING 0`
- `#define UNIFORM_LIGHT_MATRICES_BINDING 1`

7.3.1 Macro Definition Documentation

7.3.1.1 D_LIGHT_SHADOW_TEXTURES_SLOT

```
#define D_LIGHT_SHADOW_TEXTURES_SLOT GBUFFER_TEXTURES_SLOT + G_BUFFER_SIZE
```

Definition at line 9 of file [bindings.h](#).

7.3.1.2 GBUFFER_TEXTURES_SLOT

```
#define GBUFFER_TEXTURES_SLOT SKYBOX_TEXTURES_SLOT + 1
```

Definition at line 8 of file [bindings.h](#).

7.3.1.3 MODEL_TEXTURES_SLOT

```
#define MODEL_TEXTURES_SLOT 0
```

Definition at line 6 of file [bindings.h](#).

7.3.1.4 NOISE_128D_IMAGE_SLOT

```
#define NOISE_128D_IMAGE_SLOT 0
```

Definition at line 18 of file [bindings.h](#).

7.3.1.5 NOISE_128D_TEXTURES_SLOT

```
#define NOISE_128D_TEXTURES_SLOT P_LIGHT_SHADOW_TEXTURES_SLOT + MAX_POINT_LIGHTS
```

Definition at line 12 of file [bindings.h](#).

7.3.1.6 NOISE_32D_IMAGE_SLOT

```
#define NOISE_32D_IMAGE_SLOT 1
```

Definition at line 19 of file [bindings.h](#).

7.3.1.7 NOISE_32D_TEXTURES_SLOT

```
#define NOISE_32D_TEXTURES_SLOT NOISE_128D_TEXTURES_SLOT + 1
```

Definition at line 13 of file [bindings.h](#).

7.3.1.8 NOISE_CELL_POSITIONS_SLOT

```
#define NOISE_CELL_POSITIONS_SLOT NOISE_32D_TEXTURES_SLOT + 1
```

Definition at line 14 of file [bindings.h](#).

7.3.1.9 P_LIGHT_SHADOW_TEXTURES_SLOT

```
#define P_LIGHT_SHADOW_TEXTURES_SLOT D_LIGHT_SHADOW_TEXTURES_SLOT + 1
```

Definition at line 11 of file [bindings.h](#).

7.3.1.10 RANDOM_NUMBERS_SLOT

```
#define RANDOM_NUMBERS_SLOT NOISE_CELL_POSITIONS_SLOT + NUM_CELL_POSITIONS
```

Definition at line 15 of file [bindings.h](#).

7.3.1.11 SKYBOX_TEXTURES_SLOT

```
#define SKYBOX_TEXTURES_SLOT MODEL_TEXTURES_SLOT + MAX_TEXTURE_COUNT
```

Definition at line 7 of file [bindings.h](#).

7.3.1.12 STORAGE_BUFFER_MATERIAL_ID_BINDING

```
#define STORAGE_BUFFER_MATERIAL_ID_BINDING 0
```

Definition at line 22 of file [bindings.h](#).

7.3.1.13 UNIFORM_LIGHT_MATRICES_BINDING

```
#define UNIFORM_LIGHT_MATRICES_BINDING 1
```

Definition at line 23 of file [bindings.h](#).

7.4 bindings.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "host_device_shared.h"
00003 #include "GlobalValues.h"
00004
00005 // define all texture unit offsets to all textures
00006 #define MODEL_TEXTURES_SLOT 0
00007 #define SKYBOX_TEXTURES_SLOT MODEL_TEXTURES_SLOT + MAX_TEXTURE_COUNT
00008 #define GBUFFER_TEXTURES_SLOT SKYBOX_TEXTURES_SLOT + 1 // we have 1 skybox 3D texture
00009 #define D_LIGHT_SHADOW_TEXTURES_SLOT GBUFFER_TEXTURES_SLOT + G_BUFFER_SIZE
00010 // for the CSM now using 1 sampler array layer --> only one texture unit needed
00011 #define P_LIGHT_SHADOW_TEXTURES_SLOT D_LIGHT_SHADOW_TEXTURES_SLOT + 1
00012 #define NOISE_128D_TEXTURES_SLOT P_LIGHT_SHADOW_TEXTURES_SLOT + MAX_POINT_LIGHTS
00013 #define NOISE_32D_TEXTURES_SLOT NOISE_128D_TEXTURES_SLOT + 1
00014 #define NOISE_CELL_POSITIONS_SLOT NOISE_32D_TEXTURES_SLOT + 1
00015 #define RANDOM_NUMBERS_SLOT NOISE_CELL_POSITIONS_SLOT + NUM_CELL_POSITIONS
00016
00017 // all image slots
00018 #define NOISE_128D_IMAGE_SLOT 0
00019 #define NOISE_32D_IMAGE_SLOT 1
00020
00021 // storage buffer bindings
00022 #define STORAGE_BUFFER_MATERIAL_ID_BINDING 0
00023 #define UNIFORM_LIGHT_MATRICES_BINDING 1
```

7.5 C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.cpp File Reference

```
#include "Camera.h"
Include dependency graph for Camera.cpp:
```

7.6 Camera.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Camera.h"
00002
00003 Camera::Camera() :
00004
00005     position(glm::vec3(0.0f, 50.0f, 0.0f)),
00006     //here we want the normal coord. axis z is showing to us !!
00007     front(glm::vec3(0.0f, 0.0f, -1.0f)), world_up(glm::vec3(0.0f, 1.0f, 0.0f)),
00008     right(glm::normalize(glm::cross(front, world_up))),
00009     up(glm::normalize(glm::cross(right, front))), yaw(-60.0f), pitch(0.0f), movement_speed(35.0f),
00010     turn_speed(0.25f), near_plane(0.1f), far_plane(1000.f),
00011     fov(45.f)
00012 }
00013
00014 Camera::Camera(glm::vec3 start_position, glm::vec3 start_up, GLfloat start_yaw, GLfloat start_pitch,
00015     GLfloat start_move_speed, GLfloat start_turn_speed,
00016     GLfloat near_plane, GLfloat far_plane, GLfloat fov) :
00017
00018     position(start_position),
00019     //here we want the normal coord. axis z is showing to us !!
00020     front(glm::vec3(0.0f, 0.0f, -1.0f)), world_up(start_up), right(glm::normalize(glm::cross(front,
00021         world_up))), up(glm::normalize(glm::cross(right, front))),
00022     yaw(start_yaw), pitch(start_pitch), movement_speed(start_move_speed),
00023     turn_speed(start_turn_speed), near_plane(near_plane), far_plane(far_plane), fov(fov)
00024
00025 void Camera::key_control(bool* keys, GLfloat delta_time)
00026 {
00027     GLfloat velocity = movement_speed * delta_time;
00028
00029     if (keys[GLFW_KEY_W]) {
00030
00031         position += front * velocity;
00032     }
00033
00034     if (keys[GLFW_KEY_D]) {
00035
00036         position += right * velocity;
00037     }
00038
00039     if (keys[GLFW_KEY_A]) {
00040
00041         position += -right * velocity;
00042     }
00043
00044     if (keys[GLFW_KEY_S]) {
00045
00046         position += -front * velocity;
00047     }
00048
00049     if (keys[GLFW_KEY_Q]) {
00050
00051         yaw += -velocity;
00052     }
00053
00054     if (keys[GLFW_KEY_E]) {
00055
00056         yaw += velocity;
00057     }
00058 }
00059
00060 void Camera::mouse_control(GLfloat x_change, GLfloat y_change)
00061 {
00062
00063     //here we only want to support views 90 degrees to each side
00064     //again choose turn speed well in respect to its ordinal scale
00065     x_change *= turn_speed;
00066     y_change *= turn_speed;
00067
00068     yaw += x_change;
00069     pitch += y_change;
00070
00071     if (pitch > 89.0f) {
00072         pitch = 89.0f;
00073     }
00074
00075     if (pitch < -89.0f) {
00076         pitch = -89.0f;
00077     }

```

```

00078
00079     // by changing the rotations you need to update all parameters
00080     // for we retrieve them later for further calculations!
00081     update();
00082 }
00083
00084 void Camera::set_near_plane(GLfloat near_plane) { this->near_plane = near_plane; }
00085
00086 void Camera::set_far_plane(GLfloat far_plane) { this->far_plane = far_plane; }
00087
00088 void Camera::set_fov(GLfloat fov) { this->fov = fov; }
00089
00090 void Camera::set_camera_position(glm::vec3 new_camera_position) { this->position =
00091     new_camera_position; }
00092
00093     glm::mat4 Camera::get_viewmatrix() const
00094 {
00095     //very necessary for further calc
00096     return glm::lookAt(position, position + front, up);
00097 }
00098
00099 Camera::~Camera() { }
00100
00101 void Camera::update()
00102 {
00103     //https://learnopengl.com/Getting-started/Camera?fbclid=IwAR1WEr4jt6IyWC52s_WKYHtaFoeug37pG5YqbDPifgn5F1UXPbUjWbJWiq
00104     // thats a bit tricky; have a look to link above if there a questions :)
00105     // but simple geometrical analysis
00106     // consider yaw you are turnig to the side; pitch as you move the head forward and back; roll
00107     // rotations around z-axis will make you dizzy :))
00108     // notice that to roll will not change my front vector
00109     front.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
00110     front.y = sin(glm::radians(pitch));
00111     front.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
00112     front = glm::normalize(front);
00113
00114     //retrieve the right vector with some world_up
00115     right = glm::normalize(glm::cross(front, world_up));
00116
00117     // but this means the up vector must again be calculated with right vector calculated!!!
00118     up = glm::normalize(glm::cross(right, front));
00119 }

```

7.7 C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.h File Reference

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
```

Include dependency graph for Camera.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class Camera

7.8 Camera.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glm/glm.hpp>
00003 #include <glm/gtc/matrix_transform.hpp>
00004
00005 #include <glad/glad.h>
00006 #include <GLFW/glfw3.h>
00007
00008 class Camera {
```

```

00009
00010     public:
00011     Camera();
00012
00013     Camera(glm::vec3 start_position, glm::vec3 start_up, GLfloat start_yaw, GLfloat start_pitch, GLfloat
00014         start_move_speed, GLfloat start_turn_speed,
00015         GLfloat near_plane, GLfloat far_plane, GLfloat fov);
00016
00017     void key_control(bool* keys, GLfloat delta_time);
00018     void mouse_control(GLfloat x_change, GLfloat y_change);
00019
00020     glm::vec3 get_camera_position() const { return position; };
00021     glm::vec3 get_camera_direction() const { return glm::normalize(front); };
00022     glm::vec3 get_up_axis() const { return up; };
00023     glm::vec3 get_right_axis() const { return right; };
00024     GLfloat get_near_plane() const { return near_plane; };
00025     GLfloat get_far_plane() const { return far_plane; };
00026     GLfloat get_fov() const { return fov; };
00027     glm::mat4 get_viewmatrix() const;
00028
00029     void set_near_plane(GLfloat near_plane);
00030     void set_far_plane(GLfloat far_plane);
00031     void set_fov(GLfloat fov);
00032     void set_camera_position(glm::vec3 new_camera_position);
00033
00034
00035     ~Camera();
00036
00037     private:
00038     glm::vec3 position;
00039     glm::vec3 front;
00040     glm::vec3 world_up;
00041     glm::vec3 right;
00042     glm::vec3 up;
00043
00044     GLfloat yaw;
00045     GLfloat pitch;
00046
00047     GLfloat movement_speed;
00048     GLfloat turn_speed;
00049
00050     GLfloat near_plane, far_plane, fov;
00051
00052     void update();
00053 };

```

7.9 C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProgram.cpp File Reference

```
#include "ComputeShaderProgram.h"
Include dependency graph for ComputeShaderProgram.cpp:
```

7.10 ComputeShaderProgram.cpp

[Go to the documentation of this file.](#)

```

00001 #include "ComputeShaderProgram.h"
00002
00003 ComputeShaderProgram::ComputeShaderProgram() { }
00004
00005 void ComputeShaderProgram::reload() { create_computer_shader_program_from_file(compute_location); }
00006
00007 ComputeShaderProgram::~ComputeShaderProgram() { }

```

7.11 C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProgram.h File Reference

```
#include "ShaderProgram.h"
#include "host_device_shared.h"
```

```
#include <glad/glad.h>
```

Include dependency graph for ComputeShaderProgram.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ComputeShaderProgram](#)

7.12 ComputeShaderProgram.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "ShaderProgram.h"
00003 #include "host_device_shared.h"
00004
00005 #include <glad/glad.h>
00006
00007 class ComputeShaderProgram : public ShaderProgram {
00008     public:
00009     ComputeShaderProgram();
0010
0011     void reload();
0012
0013     ~ComputeShaderProgram();
0014
0015     private:
0016 };
```

7.13 C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.cpp File Reference

```
#include "DebugApp.h"
```

Include dependency graph for DebugApp.cpp:

Functions

- void APIENTRY [glDebugOutput](#) (GLenum source, GLenum type, unsigned int id, GLenum severity, GLsizei length, const char *message, const void *userParam)

7.13.1 Function Documentation

7.13.1.1 glDebugOutput()

```
void APIENTRY glDebugOutput (
    GLenum source,
    GLenum type,
    unsigned int id,
    GLenum severity,
    GLsizei length,
    const char * message,
    const void * userParam )
```

Definition at line 4 of file DebugApp.cpp.

```
00005 {
00006     // ignore non-significant error/warning codes
00007     if (id == 131169 || id == 131185 || id == 131218 || id == 131204) return;
00008
00009     std::cout << "-----" << std::endl;
00010     std::cout << "Debug message (" << id << "): " << message << std::endl;
00011
00012     switch (source) {
00013         case GL_DEBUG_SOURCE_API:
00014             std::cout << "Source: API";
00015             break;
00016         case GL_DEBUG_SOURCE_WINDOW_SYSTEM:
00017             std::cout << "Source: Window System";
00018             break;
00019         case GL_DEBUG_SOURCE_SHADER_COMPILER:
00020             std::cout << "Source: Shader Compiler";
00021             break;
00022         case GL_DEBUG_SOURCE_THIRD_PARTY:
00023             std::cout << "Source: Third Party";
00024             break;
00025         case GL_DEBUG_SOURCE_APPLICATION:
00026             std::cout << "Source: Application";
00027             break;
00028         case GL_DEBUG_SOURCE_OTHER:
00029             std::cout << "Source: Other";
00030             break;
00031     }
00032     std::cout << std::endl;
00033
00034     switch (type) {
00035         case GL_DEBUG_TYPE_ERROR:
00036             std::cout << "Type: Error";
00037             break;
00038         case GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR:
00039             std::cout << "Type: Deprecated Behaviour";
00040             break;
00041         case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR:
00042             std::cout << "Type: Undefined Behaviour";
00043             break;
00044         case GL_DEBUG_TYPE_PORTABILITY:
00045             std::cout << "Type: Portability";
00046             break;
00047         case GL_DEBUG_TYPE_PERFORMANCE:
00048             std::cout << "Type: Performance";
00049             break;
00050         case GL_DEBUG_TYPE_MARKER:
00051             std::cout << "Type: Marker";
00052             break;
00053         case GL_DEBUG_TYPE_PUSH_GROUP:
00054             std::cout << "Type: Push Group";
00055             break;
00056         case GL_DEBUG_TYPE_POP_GROUP:
00057             std::cout << "Type: Pop Group";
00058             break;
00059         case GL_DEBUG_TYPE_OTHER:
00060             std::cout << "Type: Other";
00061             break;
00062     }
00063     std::cout << std::endl;
00064
00065     switch (severity) {
00066         case GL_DEBUG_SEVERITY_HIGH:
00067             std::cout << "Severity: high";
00068             break;
00069         case GL_DEBUG_SEVERITY_MEDIUM:
00070             std::cout << "Severity: medium";
00071             break;
00072         case GL_DEBUG_SEVERITY_LOW:
00073             std::cout << "Severity: low";
```

```
00074     break;
00075     case GL_DEBUG_SEVERITY_NOTIFICATION:
00076         std::cout << "Severity: notification";
00077         break;
00078     }
00079     std::cout << std::endl;
00080     std::cout << std::endl;
00081 }
```

Referenced by [DebugApp::DebugApp\(\)](#).

Here is the caller graph for this function:

7.14 DebugApp.cpp

[Go to the documentation of this file.](#)

```
00001 #include "DebugApp.h"
00002
00003 // stolen from: https://learnopengl.com/In-Practice/Debugging
00004 void APIENTRY glDebugOutput(GLenum source, GLenum type, unsigned int id, GLenum severity, GLsizei
length, const char* message, const void* userParam)
00005 {
00006     // ignore non-significant error/warning codes
00007     if (id == 131169 || id == 131185 || id == 131218 || id == 131204) return;
00008
00009     std::cout << "-----" << std::endl;
00010     std::cout << "Debug message (" << id << "): " << message << std::endl;
00011
00012     switch (source) {
00013     case GL_DEBUG_SOURCE_API:
00014         std::cout << "Source: API";
00015         break;
00016     case GL_DEBUG_SOURCE_WINDOW_SYSTEM:
00017         std::cout << "Source: Window System";
00018         break;
00019     case GL_DEBUG_SOURCE_SHADER_COMPILER:
00020         std::cout << "Source: Shader Compiler";
00021         break;
00022     case GL_DEBUG_SOURCE_THIRD_PARTY:
00023         std::cout << "Source: Third Party";
00024         break;
00025     case GL_DEBUG_SOURCE_APPLICATION:
00026         std::cout << "Source: Application";
00027         break;
00028     case GL_DEBUG_SOURCE_OTHER:
00029         std::cout << "Source: Other";
00030         break;
00031     }
00032     std::cout << std::endl;
00033
00034     switch (type) {
00035     case GL_DEBUG_TYPE_ERROR:
00036         std::cout << "Type: Error";
00037         break;
00038     case GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR:
00039         std::cout << "Type: Deprecated Behaviour";
00040         break;
00041     case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR:
00042         std::cout << "Type: Undefined Behaviour";
00043         break;
00044     case GL_DEBUG_TYPE_PORTABILITY:
00045         std::cout << "Type: Portability";
00046         break;
00047     case GL_DEBUG_TYPE_PERFORMANCE:
00048         std::cout << "Type: Performance";
00049         break;
00050     case GL_DEBUG_TYPE_MARKER:
00051         std::cout << "Type: Marker";
00052         break;
00053     case GL_DEBUG_TYPE_PUSH_GROUP:
00054         std::cout << "Type: Push Group";
00055         break;
00056     case GL_DEBUG_TYPE_POP_GROUP:
00057         std::cout << "Type: Pop Group";
00058         break;
00059     case GL_DEBUG_TYPE_OTHER:
00060         std::cout << "Type: Other";
00061         break;
00062     }
```

```

00063     std::cout << std::endl;
00064
00065     switch (severity) {
00066         case GL_DEBUG_SEVERITY_HIGH:
00067             std::cout << "Severity: high";
00068             break;
00069         case GL_DEBUG_SEVERITY_MEDIUM:
00070             std::cout << "Severity: medium";
00071             break;
00072         case GL_DEBUG_SEVERITY_LOW:
00073             std::cout << "Severity: low";
00074             break;
00075         case GL_DEBUG_SEVERITY_NOTIFICATION:
00076             std::cout << "Severity: notification";
00077             break;
00078     }
00079     std::cout << std::endl;
00080     std::cout << std::endl;
00081 }
00082 DebugApp::DebugApp()
00083 {
00084 #ifdef NDEBUG
00085     // nondebug
00086 #else
00087     int flags;
00088     glGetIntegerv(GL_CONTEXT_FLAGS, &flags);
00089     if (flags & GL_CONTEXT_FLAG_DEBUG_BIT) {
00090         // initialize debug output
00091         glEnable(GL_DEBUG_OUTPUT);
00092         glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS);
00093         glDebugMessageCallback(glDebugOutput, nullptr);
00094         glDebugMessageControl(GL_DONT_CARE, GL_DONT_CARE, GL_DONT_CARE, 0, nullptr, GL_TRUE);
00095     }
00096 #endif
00097 }
00098
00099 bool DebugApp::areErrorPrintAll(const std::string& AdditionalArrayMessage, const char* file, int line)
00100 {
00101 #ifdef NDEBUG
00102     // nondebug
00103     return false;
00104 #else
00105     GLenum err;
00106     bool isError = false;
00107     while ((err = glGetError()) != GL_NO_ERROR) {
00108         std::string errorCode;
00109         switch (err) {
00110             case GL_INVALID_ENUM:
00111                 errorCode = "INVALID_ENUM";
00112                 break;
00113             case GL_INVALID_VALUE:
00114                 errorCode = "INVALID_VALUE";
00115                 break;
00116             case GL_INVALID_OPERATION:
00117                 errorCode = "INVALID_OPERATION";
00118                 break;
00119             case GL_STACK_OVERFLOW:
00120                 errorCode = "STACK_OVERFLOW";
00121                 break;
00122             case GL_STACK_UNDERFLOW:
00123                 errorCode = "STACK_UNDERFLOW";
00124                 break;
00125             case GL_OUT_OF_MEMORY:
00126                 errorCode = "OUT_OF_MEMORY";
00127                 break;
00128             case GL_INVALID_FRAMEBUFFER_OPERATION:
00129                 errorCode = "INVALID_FRAMEBUFFER_OPERATION";
00130                 break;
00131         }
00132         std::cout << "Error, " << errorCode << " | "
00133             << "\nAdditional Error Message: " << AdditionalArrayMessage << " "
00134             << "In File: " << file << ", Line: " << line << std::endl;
00135     isError = true;
00136 }
00137
00138     return isError;
00139     // debug code
00140 #endif
00141 }
00142
00143 bool DebugApp::arePreError(const std::string& AdditionalArrayMessage, const char* file, int line)
00144 {
00145 #ifdef NDEBUG
00146     // nondebug
00147     return false;
00148 #else
00149     // debug code

```

```

00150     GLenum err;
00151     bool isError = false;
00152     while ((err = glGetError()) != GL_NO_ERROR) {
00153         std::string errorCode;
00154         switch (err) {
00155             case GL_INVALID_ENUM:
00156                 errorCode = "INVALID_ENUM";
00157                 break;
00158             case GL_INVALID_VALUE:
00159                 errorCode = "INVALID_VALUE";
00160                 break;
00161             case GL_INVALID_OPERATION:
00162                 errorCode = "INVALID_OPERATION";
00163                 break;
00164             case GL_STACK_OVERFLOW:
00165                 errorCode = "STACK_OVERFLOW";
00166                 break;
00167             case GL_STACK_UNDERFLOW:
00168                 errorCode = "STACK_UNDERFLOW";
00169                 break;
00170             case GL_OUT_OF_MEMORY:
00171                 errorCode = "OUT_OF_MEMORY";
00172                 break;
00173             case GL_INVALID_FRAMEBUFFER_OPERATION:
00174                 errorCode = "INVALID_FRAMEBUFFER_OPERATION";
00175                 break;
00176         }
00177         std::cout << errorCode << " Error appears before executing the function, "
00178             << " | "
00179             << "\nAdditional Error Message: " << AdditionalErrorMessage << " "
00180             << "In File: " << file << ", Line: " << line << std::endl;
00181         isError = true;
00182     }
00183     return isError;
00184 }
00185
00186 #endif
00187 }
00188 DebugApp::~DebugApp() { }

```

7.15 C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.h File Reference

```
#include <string>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <iostream>
```

Include dependency graph for DebugApp.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [DebugApp](#)

7.16 DebugApp.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <string>
00003 #include <glad/glad.h>
00004 #include <GLFW/glfw3.h>
00005
00006 #include <iostream>
00007
00008 class DebugApp {
00009     public:
00010     DebugApp();
00011

```

```
00012     bool areErrorPrintAll(const std::string& AdditionalArrayMessage = "Empty", const char* file =
00013         __FILE__, int line = __LINE__);
00014     bool arePreError(const std::string& AdditionalArrayMessage = "Empty", const char* file = __FILE__,
00015         int line = __LINE__);
00016     ~DebugApp();
00017
00018     private:
00019 };
```

7.17 C:/Users/jonas/Desktop/GraphicEngine/Src/GlobalValues.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define NUM_MIN_CASCADES 3
- #define MAX_RESOLUTION_X 1920
- #define MAX_RESOLUTION_Y 1080

Variables

- const int G_BUFFER_SIZE = 4
- const int NUM_FRUSTUM_PLANES = 6
- const int NUM_CLOUDS = 1

7.17.1 Macro Definition Documentation

7.17.1.1 MAX_RESOLUTION_X

```
#define MAX_RESOLUTION_X 1920
```

Definition at line 5 of file [GlobalValues.h](#).

7.17.1.2 MAX_RESOLUTION_Y

```
#define MAX_RESOLUTION_Y 1080
```

Definition at line 6 of file [GlobalValues.h](#).

7.17.1.3 NUM_MIN_CASCADES

```
#define NUM_MIN_CASCADES 3
```

Definition at line 4 of file [GlobalValues.h](#).

7.17.2 Variable Documentation

7.17.2.1 G_BUFFER_SIZE

```
const int G_BUFFER_SIZE = 4
```

Definition at line 8 of file [GlobalValues.h](#).

Referenced by [GBuffer::create\(\)](#).

7.17.2.2 NUM_CLOUDS

```
const int NUM_CLOUDS = 1
```

Definition at line 10 of file [GlobalValues.h](#).

7.17.2.3 NUM_FRUSTUM_PLANES

```
const int NUM_FRUSTUM_PLANES = 6
```

Definition at line 9 of file [GlobalValues.h](#).

Referenced by [ViewFrustumCulling::is_inside\(\)](#).

7.18 GlobalValues.h

[Go to the documentation of this file.](#)

```
00001 #ifndef COMMONVALS
00002 #define COMMONVALS
00003
00004 #define NUM_MIN_CASCADES 3
00005 #define MAX_RESOLUTION_X 1920
00006 #define MAX_RESOLUTION_Y 1080
00007
00008 const int G_BUFFER_SIZE = 4;
00009 const int NUM_FRUSTUM_PLANES = 6;
00010 const int NUM_CLOUDS = 1;
00011
00012 #endif
```

7.19 C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.cpp File Reference

```
#include "GUI.h"
#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>
#include <sstream>
Include dependency graph for GUI.cpp:
```

7.20 GUI.cpp

[Go to the documentation of this file.](#)

```
00001 #include "GUI.h"
00002
00003 #include <imgui.h>
00004 #include <imgui_impl_glfw.h>
00005 #include <imgui_impl_opengl3.h>
00006 #include <sstream>
00007
00008 GUI::GUI()
00009 {
00010
00011     // give some arbitrary values; we will update these values after 1 frame :
00012     this->direccional_light_radiance = 4.0f;
00013
00014     this->directional_light_color[0] = 1;
00015     this->directional_light_color[1] = 1;
00016     this->directional_light_color[2] = 1;
00017
00018     this->directional_light_direction[0] = -0.1f;
00019     this->directional_light_direction[1] = -1.f;
00020     this->directional_light_direction[2] = -0.1f;
00021
00022     this->cloud_speed = 6;
00023     this->cloud_scale = 0.63f;
00024     this->cloud_density = 0.493f;
00025     this->cloud_pillowness = 0.966f;
00026     this->cloud_cirrus_effect = 0.034f;
00027
00028     this->cloud_mesh_scale[0] = 1000.f;
00029     this->cloud_mesh_scale[1] = 5.f;
00030     this->cloud_mesh_scale[2] = 1000.f;
00031
00032     this->cloud_mesh_offset[0] = -.364f;
00033     this->cloud_mesh_offset[1] = 367.f;
00034     this->cloud_mesh_offset[2] = -18.351f;
00035
00036     this->cloud_powder_effect = true;
00037
00038     this->cloud_movement_direction[0] = 1.f;
00039     this->cloud_movement_direction[1] = 1.f;
00040     this->cloud_movement_direction[2] = 1.f;
00041
00042     this->cloud_num_march_steps = 8;
00043     this->cloud_num_march_steps_to_light = 3;
00044
00045     this->shadow_map_res_index = 3;
00046     this->shadow_resolution_changed = false;
00047     this->num_shadow_cascades = NUM_CASCADES;
00048     this->pcf_radius = 2;
00049     this->cascaded_shadow_intensity = 0.65f;
00050
00051     this->available_shadow_map_resolutions[0] = "512";
00052     this->available_shadow_map_resolutions[1] = "1024";
00053     this->available_shadow_map_resolutions[2] = "2048";
00054     this->available_shadow_map_resolutions[3] = "4096";
00055
00056
00057     std::stringstream texture_base_dir;
00058     texture_base_dir << CMAKELISTS_DIR;
00059     texture_base_dir << "/Resources/Textures/";
00060
00061     std::stringstream texture_logo;
00062     texture_logo << texture_base_dir.str() << "Loading_Screen/Engine_logo.png";
```

```
00063     logo_tex = Texture(texture_logo.str().c_str(), std::make_shared<RepeatMode>());
00064     logo_tex.load_texture_with_alpha_channel();
00065 }
00066
00067 void GUI::init(std::shared_ptr<Window> main_window)
00068 {
00069     // Setup Dear ImGui context
00070     IMGUI_CHECKVERSION();
00071     ImGui::CreateContext();
00072     //ImGuiIO& io = ImGui::GetIO();
00073     const ImGuiStyle& style = ImGui::GetStyle();
00074     // Setup Platform/Renderer bindings
00075     ImGui_ImplGlfw_InitForOpenGL(main_window->get_window(), true);
00076     const char* glsl_version = "#version 460";
00077     ImGui_ImplOpenGL3_Init(glsl_version);
00078     // Setup Dear ImGui style
00079     ImGui::StyleColorsDark();
00080     ImGui::PushStyleVar(ImGuiStyleVar_WindowRounding, 10);
00081     ImGui::PushStyleVar(ImGuiStyleVar_FrameRounding, 10);
00082     ImGui::PushStyleVar(ImGuiStyleVar_FrameBorderSize, 1);
00083 }
00084
00085 void GUI::render(bool loading_in_progress, float progress, bool& shader_hot_reload_triggered)
00086 {
00087
00088     // feed inputs to dear imgui, start new frame
00089     //UI.start_new_frame();
00090     ImGui_ImplOpenGL3_NewFrame();
00091     ImGui_ImplGlfw_NewFrame();
00092     ImGui::NewFrame();
00093     // render your GUI
00094     ImGui::Begin("GUI v1.3.3");
00095
00096     if (loading_in_progress) {
00097         ImGui::ProgressBar(progress, ImVec2(0.0f, 0.0f));
00098         ImGui::SameLine(0.0f, ImGui::GetStyle().ItemInnerSpacing.x);
00099         ImGui::Text("Loading scene");
00100         ImGui::Separator();
00101     }
00102
00103
00104     if (ImGui::CollapsingHeader("Hot shader reload")) {
00105
00106         if (ImGui::Button("Hot reload ALL shaders!")) {
00107             shader_hot_reload_triggered = true;
00108         }
00109     }
00110
00111     ImGui::Separator();
00112
00113     if (ImGui::CollapsingHeader("Graphic Settings")) {
00114
00115         if (ImGui::TreeNode("Directional Light")) {
00116             ImGui::Separator();
00117             ImGui::SliderFloat("Radiance", &direccional_light_radiance, 0.0f, 50.0f);
00118             ImGui::Separator();
00119             // Edit a color (stored as ~4 floats)
00120             ImGui::ColorEdit3("Directional Light Color", directional_light_color);
00121             ImGui::Separator();
00122             ImGui::SliderFloat3("Light Direction", directional_light_direction, -1.f, 1.0f);
00123
00124         if (ImGui::TreeNode("Shadows")) {
00125
00126             int shadow_map_res_index_before = shadow_map_res_index;
00127             ImGui::Combo("Shadow Map Resolution", &shadow_map_res_index, available_shadow_map_resolutions,
00128                         IM_ARRAYSIZE(available_shadow_map_resolutions));
00129             if (shadow_map_res_index_before != shadow_map_res_index) shadow_resolution_changed = true;
00130
00131             int num_cascades_before = num_shadow_cascades;
00132             ImGui::SliderInt("# cascades", &num_shadow_cascades, NUM_MIN_CASCADES, NUM_CASCADES);
00133             if (num_cascades_before != num_shadow_cascades) shadow_resolution_changed = true;
00134
00135             ImGui::SliderInt("PCF radius", &pcf_radius, 1, 20);
00136             ImGui::SliderFloat("Shadow intensity", &cascaded_shadow_intensity, 0.0f, 1.0f);
00137
00138         ImGui::TreePop();
00139     }
00140
00141     ImGui::TreePop();
00142
00143
00144     if (ImGui::TreeNode("Cloud Settings")) {
00145
00146         ImGui::SliderInt("Speed", &cloud_speed, 0, 30);
00147         ImGui::SliderInt("# march steps", &cloud_num_march_steps, 1, 128);
00148         ImGui::SliderInt("# march steps to light", &cloud_num_march_steps_to_light, 1, 128);
00149
00150 }
```

```

00149     ImGui::SliderFloat3("Movement Direction", cloud_movement_direction, -10.f, 10.0f);
00150     ImGui::SliderFloat("Illumination intensity", &cloud_scale, 0.f, 1.0f);
00151     ImGui::SliderFloat("Density", &cloud_density, 0.f, 1.0f);
00152     ImGui::SliderFloat("Pillowness", &cloud_pillowness, 0.f, 1.0f);
00153     ImGui::SliderFloat("Cirrus effect", &cloud_cirrus_effect, 0.f, 1.0f);
00154     ImGui::Checkbox("Powder effect", &cloud_powder_effect);
00155     ImGui::SliderFloat3("Scale", cloud_mesh_scale, 0.f, 1000.0f);
00156     ImGui::SliderFloat3("Translation", cloud_mesh_offset, -200.f, 400.0f);
00157
00158     ImGui::TreePop();
00159 }
00160 }
00161 ImGui::Separator();
00162 /*if (ImGui::CollapsingHeader("Audio Settings")) {
00163     ImGui::SliderFloat("Volume", &sound_volume, 0.0f, 1.0f);
00164 }*/
00165
00166     ImGui::Separator();
00167
00168 if (ImGui::CollapsingHeader("GUI Settings")) {
00169     ImGuiStyle& style = ImGui::GetStyle();
00170     if (ImGui::SliderFloat("Frame Rounding", &style.FrameRounding, 0.0f, 12.0f, "% .0f")) {
00171         style.GrabRounding = style.FrameRounding; // Make GrabRounding always the same value as
00172         FrameRounding
00173     }
00174     {
00175         bool border = (style.FrameBorderSize > 0.0f);
00176         if (ImGui::Checkbox("FrameBorder", &border)) {
00177             style.FrameBorderSize = border ? 1.0f : 0.0f;
00178         }
00179         ImGui::SliderFloat("WindowRounding", &style.WindowRounding, 0.0f, 12.0f, "% .0f");
00180     }
00181     ImGui::Separator();
00182
00183 if (ImGui::CollapsingHeader("KEY Bindings")) {
00184     ImGui::Text("WASD for moving Forward, backward and to the side\nQE for rotating ");
00185 }
00186
00187     ImGui::Separator();
00188
00189     ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate,
00190     ImGui::GetIO().Framerate);
00191 //ImGui::ShowDemoWindow();
00192
00193     ImGui::Image((void*)logo_tex.get_id(), ImVec2(200, 200), ImVec2(0, 1), ImVec2(1, 0));
00194
00195     ImGui::End();
00196
00197 // feed inputs to dear imgui, start new frame
00198 ImGui::Render();
00199 ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
00200 }
00201
00202
00203 void GUI::update_user_input(std::shared_ptr<Scene> scene)
00204 {
00205     std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00206     main_light->set_radiance(directional_light_radiance);
00207     main_light->get_shadow_map()->set_intensity(cascaded_shadow_intensity);
00208     main_light->get_shadow_map()->set_pcf_radius(pcf_radius);
00209
00210     glm::vec3 new_main_light_color(directional_light_color[0], directional_light_color[1],
00211     directional_light_color[2]);
00212
00213     main_light->set_color(new_main_light_color);
00214
00215     glm::vec3 new_main_light_pos(directional_light_direction[0], directional_light_direction[1],
00216     directional_light_direction[2]);
00217
00218     main_light->set_direction(new_main_light_pos);
00219
00220     glm::vec3 cloud_move(cloud_movement_direction[0], cloud_movement_direction[1],
00221     cloud_movement_direction[2]);
00222
00223     std::shared_ptr<Clouds> clouds = scene->get_clouds();
00224
00225     clouds->set_movement_direction(cloud_move);
00226     clouds->set_movement_speed(static_cast<float>(cloud_speed));
00227     clouds->set_density(cloud_density);
00228     clouds->set_scale(cloud_scale);
00229     clouds->set_pillowness(cloud_pillowness);
00230 }
```

```

00231     clouds->set_cirrus_effect(cloud_cirrus_effect);
00232     clouds->set_powder_effect(cloud_powder_effect);
00233
00234     clouds->set_scale(glm::vec3(cloud_mesh_scale[0], cloud_mesh_scale[1], cloud_mesh_scale[2]));
00235
00236     clouds->set_translation(glm::vec3(cloud_mesh_offset[0], cloud_mesh_offset[1],
00237                               cloud_mesh_offset[2]));
00238     GLfloat shadow_map_resolution = 4096.f;
00239
00240     if (shadow_resolution_changed) {
00241
00242         switch (shadow_map_res_index) {
00243             case 0:
00244                 shadow_map_resolution = 512.f;
00245                 break;
00246             case 1:
00247                 shadow_map_resolution = 1024.f;
00248                 break;
00249             case 2:
00250                 shadow_map_resolution = 2048.f;
00251                 break;
00252             case 3:
00253                 shadow_map_resolution = 4096.f;
00254         }
00255
00256         main_light->update_shadow_map(shadow_map_resolution, shadow_map_resolution, NUM_CASCADES);
00257
00258         shadow_resolution_changed = false;
00259     }
00260 }
00261
00262 GUI::~GUI ()
00263 {
00264     ImGui_ImplOpenGL3_Shutdown();
00265     ImGui_ImplGlfw_Shutdown();
00266     ImGui::DestroyContext();
00267 }
```

7.21 C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.h File Reference

```

#include <memory>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include "Window.h"
#include "Texture.h"
#include "Clouds.h"
#include "host_device_shared.h"
#include "Scene.h"
```

Include dependency graph for GUI.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class **GUI**

7.22 GUI.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <memory>
00003
00004 #include <glad/glad.h>
00005 #include <GLFW/glfw3.h>
00006
00007 #include "Window.h"
```

```

00008 #include "Texture.h"
00009 #include "Clouds.h"
00010 #include "host_device_shared.h"
00011
00012 #include "Scene.h"
00013
00014 class GUI {
00015     public:
00016     GUI();
00017
00018     void init(std::shared_ptr<Window> main_window);
00019
00020     void render(bool loading_in_progress, float progress, bool& shader_hot_reload_triggered);
00021
00022     void update_user_input(std::shared_ptr<Scene> scene);
00023
00024     ~GUI();
00025
00026     private:
00027     Texture logo_tex;
00028
00029     float direcional_light_radiance;
00030     float directional_light_color[3];
00031     float directional_light_direction[3];
00032
00033     int cloud_speed;
00034     float cloud_scale;
00035     float cloud_density;
00036     float cloud_pillowness;
00037     float cloud_cirrus_effect;
00038     float cloud_mesh_scale[3];
00039     float cloud_mesh_offset[3];
00040     bool cloud_powder_effect;
00041     float cloud_movement_direction[3];
00042     int cloud_num_march_steps;
00043     int cloud_num_march_steps_to_light;
00044
00045     int shadow_map_res_index;
00046     bool shadow_resolution_changed;
00047     int num_shadow_cascades;
00048     int pcf_radius;
00049     float cascaded_shadow_intensity;
00050     const char* available_shadow_map_resolutions[4];
00051 };

```

7.23 C:/Users/jonas/Desktop/GraphicEngine/Src/host_device_shared.h File Reference

This graph shows which files directly or indirectly include this file:

Variables

- const int **NUM_CASCADES** = 3
- const int **MAX_POINT_LIGHTS** = 1
- const int **MAX_MATERIALS** = 35
- const int **SKYBOX_MATERIAL_ID** = **MAX_MATERIALS**
- const int **CLOUDS_MATERIAL_ID** = **MAX_MATERIALS** + 1
- const int **MAX_TEXTURE_COUNT** = 30
- const float **PI** = 3.14159265359f
- const int **NUM_CELL_POSITIONS** = 5

7.23.1 Variable Documentation

7.23.1.1 CLOUDS_MATERIAL_ID

```
const int CLOUDS_MATERIAL_ID = MAX_MATERIALS + 1
```

Definition at line 11 of file [host_device_shared.h](#).

7.23.1.2 MAX_MATERIALS

```
const int MAX_MATERIALS = 35
```

Definition at line 9 of file [host_device_shared.h](#).

Referenced by [ObjLoader::load\(\)](#).

7.23.1.3 MAX_POINT_LIGHTS

```
const int MAX_POINT_LIGHTS = 1
```

Definition at line 6 of file [host_device_shared.h](#).

Referenced by [Scene::Scene\(\)](#).

7.23.1.4 MAX_TEXTURE_COUNT

```
const int MAX_TEXTURE_COUNT = 30
```

Definition at line 14 of file [host_device_shared.h](#).

7.23.1.5 NUM_CASCADES

```
const int NUM_CASCADES = 3
```

Definition at line 5 of file [host_device_shared.h](#).

Referenced by [DirectionalLight::calc_cascaded_slots\(\)](#), [DirectionalShadowMapPass::execute\(\)](#), [DirectionalLight::get_cascaded_slots\(\)](#), [GUI::GUI\(\)](#), [CascadedShadowMap::init\(\)](#), [GUI::render\(\)](#), [LightingPass::set_uniforms\(\)](#), and [GUI::update_user_input\(\)](#).

7.23.1.6 NUM_CELL_POSITIONS

```
const int NUM_CELL_POSITIONS = 5
```

Definition at line 20 of file [host_device_shared.h](#).

Referenced by [Noise::create_res128_noise\(\)](#), [Noise::create_res32_noise\(\)](#), [Noise::delete_textures\(\)](#), [Noise::generate_num_cells_text\(\)](#), [Noise::Noise\(\)](#), and [Noise::update\(\)](#).

7.23.1.7 PI

```
const float PI = 3.14159265359f
```

Definition at line 17 of file [host_device_shared.h](#).

7.23.1.8 SKYBOX_MATERIAL_ID

```
const int SKYBOX_MATERIAL_ID = MAX_MATERIALS
```

Definition at line 10 of file [host_device_shared.h](#).

7.24 host_device_shared.h

[Go to the documentation of this file.](#)

```
00001 #ifndef GLOBALS_HOST_DEVICE
00002 #define GLOBALS_HOST_DEVICE
00003
00004 // lights
00005 const int NUM_CASCADES = 3;
00006 const int MAX_POINT_LIGHTS = 1;
00007
00008 // materials
00009 const int MAX_MATERIALS = 35;
00010 const int SKYBOX_MATERIAL_ID = MAX_MATERIALS;
00011 const int CLOUDS_MATERIAL_ID = MAX_MATERIALS + 1;
00012
00013 //texture
00014 const int MAX_TEXTURE_COUNT = 30;
00015
00016 //math
00017 const float PI = 3.14159265359f;
00018
00019 // noise
00020 const int NUM_CELL_POSITIONS = 5;
00021
00022 #endif // !GLOBALS_HOST_DEVICE
```

7.25 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← GBuffer.cpp File Reference

```
#include "GBuffer.h"
#include <stdio.h>
```

Include dependency graph for GBuffer.cpp:

7.26 GBuffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include "GBuffer.h"
00002 #include <stdio.h>
00003
00004 GBuffer::GBuffer()
00005 {
00006     this->window_width = 1024;
00007     this->window_height = 768;
00008 }
00009
00010 GBuffer::GBuffer(GLint window_width, GLint window_height)
00011 {
00012
00013     this->window_width = window_width;
00014     this->window_height = window_height;
00015 }
00016
00017 void GBuffer::create()
00018 {
00019
00020     glGenFramebuffers(1, &g_buffer);
00021     glBindFramebuffer(GL_FRAMEBUFFER, g_buffer);
00022
00023     glGenTextures(1, &g_position);
00024     glBindTexture(GL_TEXTURE_2D, g_position);
00025     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_FLOAT, NULL);
00026     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00027     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00028     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[0], GL_TEXTURE_2D, g_position, 0);
00029
00030     glGenTextures(1, &g_normal);
00031     glBindTexture(GL_TEXTURE_2D, g_normal);
00032     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_FLOAT, NULL);
00033     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00034     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00035     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[1], GL_TEXTURE_2D, g_normal, 0);
00036
00037     glGenTextures(1, &g_albedo);
00038     glBindTexture(GL_TEXTURE_2D, g_albedo);
00039     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA,
00040         GL_UNSIGNED_BYTE, NULL);
00041     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00042     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00043     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[2], GL_TEXTURE_2D, g_albedo, 0);
00044
00045     glGenTextures(1, &g_material_id);
00046     glBindTexture(GL_TEXTURE_2D, g_material_id);
00047     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, window_width, window_height, 0, GL_RGBA, GL_UNSIGNED_INT,
00048         NULL);
00049     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00050     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00051     glFramebufferTexture2D(GL_FRAMEBUFFER, g_buffer_attachment[3], GL_TEXTURE_2D, g_material_id, 0);
00052
00053     // create and attach depth buffer (renderbuffer)
00054     // renderbuffers are a bit more performant
00055     glGenRenderbuffers(1, &g_depth);
00056     glBindRenderbuffer(GL_RENDERBUFFER, g_depth);
00057     glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT32F, window_width, window_height);
00058     glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, g_depth);
00059
00060     if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE) {
00061         printf("Framebuffer not complete!");
00062     }
00063
00064     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00065 }
00066
00067 void GBuffer::read(std::shared_ptr<ShaderProgram> shader_program)
00068 {
00069     // GBUFFER
00070     GLuint texture_index = GBUFFER_TEXTURES_SLOT;
00071     shader_program->setUniformInt(texture_index, "g_position");
00072     glEnableTexture(GL_TEXTURE0 + texture_index);
00073     glBindTexture(GL_TEXTURE_2D, g_position);
00074
00075     texture_index++;
00076     shader_program->setUniformInt(texture_index, "g_normal");
00077     glEnableTexture(GL_TEXTURE0 + texture_index);
00078     glBindTexture(GL_TEXTURE_2D, g_normal);
00079
00080     texture_index++;

```

```

00081     shader_program->setUniformInt(texture_index, "g_albedo");
00082     glBindTexture(GL_TEXTURE0 + texture_index, g_albedo);
00083     glBindTexture(GL_TEXTURE_2D, g_albedo);
00084
00085     texture_index++;
00086     shader_program->setUniformInt(texture_index, "g_material_id");
00087     glBindTexture(GL_TEXTURE0 + texture_index, g_material_id);
00088     glBindTexture(GL_TEXTURE_2D, g_material_id);
00089 }
00090
00091 void GBuffer::update_window_params(GLuint window_width, GLuint window_height)
00092 {
00093     this->window_width = window_width;
00094     this->window_height = window_height;
00095 }
00096
00097 GBuffer::~GBuffer()
00098 {
00099
00100     glDeleteFramebuffers(1, &g_buffer);
00101     glDeleteTextures(1, &g_position);
00102     glDeleteTextures(1, &g_normal);
00103     glDeleteTextures(1, &g_albedo);
00104     glDeleteTextures(1, &g_material_id);
00105 }

```

7.27 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← GBuffer.h File Reference

```

#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <memory>
#include "ShaderProgram.h"
#include "GlobalValues.h"
#include "bindings.h"

```

Include dependency graph for GBuffer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class GBuffer

7.28 GBuffer.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <glad/glad.h>
00003 #include <glm/glm.hpp>
00004 #include <glm/gtc/type_ptr.hpp>
00005 #include <memory>
00006
00007 #include "ShaderProgram.h"
00008 #include "GlobalValues.h"
00009 #include "bindings.h"
00010
00011 class GBuffer {
00012 public:
00013     GBuffer();
00014     GBuffer(GLint window_width, GLint window_height);
00015
00016     void create();
00017     void read(std::shared_ptr<ShaderProgram> shader_program);
00018
00019     void update_window_params(GLuint window_width, GLuint window_height);
00020
00021     GLuint get_id() const { return g_buffer; }

```

```

00022
00023     ~GBuffer();
00024
00025     private:
00026     GLuint g_buffer;
00027
00028     GLuint g_position, g_normal, g_albedo, g_material_id, g_depth;
00029
00030     GLuint g_buffer_attachment[G_BUFFER_SIZE] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1,
00031                                         GL_COLOR_ATTACHMENT2, GL_COLOR_ATTACHMENT3 };
00032
00033 };

```

7.29 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ GeometryPass.cpp File Reference

```
#include "GeometryPass.h"
Include dependency graph for GeometryPass.cpp:
```

7.30 GeometryPass.cpp

[Go to the documentation of this file.](#)

```

00001 #include "GeometryPass.h"
00002
00003 GeometryPass::GeometryPass() : skybox() { create_shader_program(); }
00004
00005 void GeometryPass::execute(glm::mat4 projection_matrix, std::shared_ptr<Camera> main_camera, GLuint
00006     window_width, GLuint window_height, GLuint gbuffer_id,
00007     GLfloat delta_time, std::shared_ptr<Scene> scene)
00008 {
00009     glBindFramebuffer(GL_FRAMEBUFFER, gbuffer_id);
00010
00011     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00012     glViewport(0, 0, window_width, window_height);
00013
00014     glEnable(GL_CULL_FACE);
00015     glCullFace(GL_BACK);
00016     glFrontFace(GL_CCW);
00017
00018     shader_program->use_shader_program();
00019
00020     glm::mat4 view_matrix = main_camera->get_viewmatrix();
00021     std::vector<ObjMaterial> materials = scene->get_materials();
00022
00023     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00024     shader_program->setUniformMatrix4fv(view_matrix, "view");
00025
00026     std::stringstream ss;
00027     for (uint32_t i = 0; i < static_cast<uint32_t>(scene->get_texture_count(0)); i++) {
00028
00029         ss << "model_textures[" << i << "]";
00030         shader_program->setUniformInt(MODEL_TEXTURES_SLOT + i, ss.str());
00031         ss.clear();
00032         ss.str(std::string());
00033     }
00034
00035     for (uint32_t i = 0; i < static_cast<uint32_t>(materials.size()); i++) {
00036
00037         ss << "materials[" << i << "].ambient";
00038         shader_program->setUniformVec3(materials[i].get_ambient(), ss.str());
00039         ss.clear();
00040         ss.str(std::string());
00041
00042         ss << "materials[" << i << "].diffuse";
00043         shader_program->setUniformVec3(materials[i].get_diffuse(), ss.str());
00044         ss.clear();
00045         ss.str(std::string());
00046
00047         ss << "materials[" << i << "].specular";
00048         shader_program->setUniformVec3(materials[i].get_specular(), ss.str());
00049         ss.clear();

```

```

00050     ss.str(std::string());
00051
00052     ss << "materials[" << i << "].transmittance";
00053     shader_program->setUniformVec3(materials[i].get_transmittance(), ss.str());
00054     ss.clear();
00055     ss.str(std::string());
00056
00057     ss << "materials[" << i << "].emission";
00058     shader_program->setUniformVec3(materials[i].get_emission(), ss.str());
00059     ss.clear();
00060     ss.str(std::string());
00061
00062     ss << "materials[" << i << "].shininess";
00063     shader_program->setUniformFloat(materials[i].get_shininess(), ss.str());
00064     ss.clear();
00065     ss.str(std::string());
00066
00067     ss << "materials[" << i << "].ior";
00068     shader_program->setUniformFloat(materials[i].get_ior(), ss.str());
00069     ss.clear();
00070     ss.str(std::string());
00071
00072     ss << "materials[" << i << "].dissolve";
00073     shader_program->setUniformFloat(materials[i].get_dissolve(), ss.str());
00074     ss.clear();
00075     ss.str(std::string());
00076
00077     ss << "materials[" << i << "].illum";
00078     shader_program->setUniformInt(materials[i].get_illum(), ss.str());
00079     ss.clear();
00080     ss.str(std::string());
00081
00082     ss << "materials[" << i << "].textureID";
00083     shader_program->setUniformInt(materials[i].get_textureID(), ss.str());
00084     ss.clear();
00085     ss.str(std::string());
00086 }
00087
00088 shader_program->validate_program();
00089
00090 scene->bind_textures_and_buffer();
00091
00092 //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
00093 //aab->render();
00094 //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
00095 //
00096
00097 std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00098
00099 for (std::shared_ptr<GameObject> object : game_objects) {
00100
00101 /* if (object_is_visible(object)) {*/
00102
00103     set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00104
00105     object->render();
00106     //}
00107 }
00108
00109 skybox.draw_sky_box(projection_matrix, view_matrix, window_width, window_height, delta_time);
00110
00111 /*glCullFace(GL_FRONT);
00112 glFrontFace(GL_CCW);*/
00113 // render the AABB for the clouds
00114 glBindFramebuffer(GL_FRAMEBUFFER, 0);
00115
00116 std::shared_ptr<Clouds> clouds = scene->get_clouds();
00117 clouds->render(projection_matrix, view_matrix, window_width, window_height);
00118
00119 }
00120
00121 void GeometryPass::create_shader_program()
00122 {
00123     this->shader_program = std::make_shared<GeometryPassShaderProgram>(GeometryPassShaderProgram{});
00124     this->shader_program->create_from_files("rasterizer/g_buffer_geometry_pass.vert",
00125                                             "rasterizer/g_buffer_geometry_pass.frag");
00126 }
00127
00128 void GeometryPass::set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model)
00129 {
00130     shader_program->setUniformMatrix4fv(model, "model");
00131     shader_program->setUniformMatrix4fv(normal_model, "normal_model");
00132 }
00133
00134 GeometryPass::~GeometryPass() { }

```

7.31 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← GeometryPass.h File Reference

```
#include "RenderPassSceneDependend.h"
#include "DirectionalLight.h"
#include "GeometryPassShaderProgram.h"
#include "SkyBox.h"
#include "ViewFrustumCulling.h"
#include "Camera.h"
#include "Clouds.h"
#include "Scene.h"
#include <array>
```

Include dependency graph for GeometryPass.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [GeometryPass](#)

7.32 GeometryPass.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "RenderPassSceneDependend.h"
00003 #include "DirectionalLight.h"
00004 #include "GeometryPassShaderProgram.h"
00005 #include "SkyBox.h"
00006 #include "ViewFrustumCulling.h"
00007 #include "Camera.h"
00008 #include "Clouds.h"
00009 #include "Scene.h"
00010 #include "SkyBox.h"
00011
00012 #include <array>
00013
00014 class GeometryPass : public RenderPassSceneDependend {
00015     public:
00016     GeometryPass();
00017
00018     void execute(glm::mat4 projection_matrix, std::shared_ptr<Camera> main_camera, GLuint window_width,
00019                 GLuint window_height, GLuint gbuffer_id,
00020                 GLfloat delta_time, std::shared_ptr<Scene>);
00021
00022     void create_shader_program();
00023
00024     void set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model);
00025
00026     ~GeometryPass();
00027
00028     private:
00029     std::shared_ptr<GeometryPassShaderProgram> shader_program;
00030     SkyBox skybox;
00031 };
```

7.33 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← GeometryPassShaderProgram.cpp File Reference

```
#include "GeometryPassShaderProgram.h"
Include dependency graph for GeometryPassShaderProgram.cpp:
```

7.34 GeometryPassShaderProgram.cpp

[Go to the documentation of this file.](#)

```
00001 #include "GeometryPassShaderProgram.h"
00002
00003 GeometryPassShaderProgram::GeometryPassShaderProgram() { }
00004
00005 GeometryPassShaderProgram::~GeometryPassShaderProgram() { }
```

7.35 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← GeometryPassShaderProgram.h File Reference

```
#include "ShaderProgram.h"
#include "host_device_shared.h"
```

Include dependency graph for GeometryPassShaderProgram.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [GeometryPassShaderProgram](#)

7.36 GeometryPassShaderProgram.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "ShaderProgram.h"
00003 #include "host_device_shared.h"
00004
00005 class GeometryPassShaderProgram : public ShaderProgram {
00006
00007     public:
00008     GeometryPassShaderProgram();
00009
00010     GLuint get_program_id() { return program_id; }
00011
00012     ~GeometryPassShaderProgram();
00013
00014     protected:
00015 };
```

7.37 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← LightingPass.cpp File Reference

```
#include "LightingPass.h"
Include dependency graph for LightingPass.cpp:
```

7.38 LightingPass.cpp

[Go to the documentation of this file.](#)

```

00001 #include "LightingPass.h"
00002
00003 LightingPass::LightingPass() :
00004     current_offset(glm::vec3(0.0f)), quad()
00005 {
00006
00007     create_shader_program();
00008 }
00009
00010
00011
00012 void LightingPass::execute(
00013     glm::mat4 projection_matrix, std::shared_ptr<Camera> main_camera, std::shared_ptr<Scene> scene,
00014     std::shared_ptr<GBuffer> gbuffer, float delta_time)
00015 {
00016     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00017
00018     std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00019     std::shared_ptr<Clouds> cloud = scene->get_clouds();
00020     std::vector<std::shared_ptr<PointLight>> point_lights = scene->get_point_lights();
00021
00022     shader_program->use_shader_program();
00023
00024     set_uniforms(projection_matrix, main_camera, scene, gbuffer, delta_time);
00025
00026     //bind textures to their units
00027     main_light->get_shadow_map()->read(D_LIGHT_SHADOW_TEXTURES_SLOT);
00028     cloud->read();
00029
00030     for (uint32_t i = 0; i < static_cast<GLuint>(point_lights.size()); i++) {
00031         point_lights[i]->get_omni_shadow_map()->read(P_LIGHT_SHADOW_TEXTURES_SLOT + i);
00032     }
00033
00034     // render screen filling quad
00035     quad.render();
00036
00037     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00038 }
00039
00040 void LightingPass::create_shader_program()
00041 {
00042     shader_program = std::make_shared<LightingPassShaderProgram>(<LightingPassShaderProgram>());
00043     shader_program->create_from_files("rasterizer/g_buffer_lighting_pass.vert",
00044                                         "rasterizer/g_buffer_lighting_pass.frag");
00045 }
00046
00047 void LightingPass::set_uniforms(
00048     glm::mat4 projection_matrix, std::shared_ptr<Camera> main_camera, std::shared_ptr<Scene> scene,
00049     std::shared_ptr<GBuffer> gbuffer, float delta_time)
00050 {
00051     // VP
00052     glm::mat4 view_matrix = main_camera->get_viewmatrix();
00053     shader_program->setUniformMatrix4fv(view_matrix, "view");
00054     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00055
00056     // SUN UNIFORMS
00057     std::shared_ptr<DirectionalLight> main_light = scene->get_sun();
00058     shader_program->setUniformFloat(main_light->get_radiance(), "directional_light.base.radiance");
00059     shader_program->setUniformVec3(main_light->get_color(), "directional_light.base.color");
00060     shader_program->setUniformVec3(main_light->get_direction(), "directional_light.direction");
00061
00062     // EVERYTHING REGARDING THE SHADOW CASCADE
00063     shader_program->setUniformInt(D_LIGHT_SHADOW_TEXTURES_SLOT, "directional_shadow_maps");
00064
00065     std::vector<GLfloat> cascade_slots = main_light->get_cascaded_slots();
00066
00067     std::stringstream ss;
00068     for (uint32_t i = 0; i < NUM_CASCADES; i++) {
00069
00070         glm::vec4 clip_end_slot = projection_matrix * glm::vec4(0.0f, 0.0f, -cascade_slots[i + 1], 1.0f);
00071         ss << "cascade_endpoints[" << i << "]";
00072         shader_program->setUniformFloat(clip_end_slot.z, ss.str());
00073         ss.clear();
00074         ss.str(std::string());
00075
00076     shader_program->setUniformInt(main_light->get_shadow_map()->get_pcf_radius(), "pcf_radius");
00077
00078     // READ GBUFFER
00079     gbuffer->read(shader_program);

```

```

00080
00081 // POINT LIGHTS
00082 std::vector<std::shared_ptr<PointLight>> point_lights = scene->get_point_lights();
00083
00084 shader_program->setUniformInt(static_cast<uint32_t>(point_lights.size()), "point_light_count");
00085
00086 for (uint32_t i = 0; i < static_cast<uint32_t>(point_lights.size()); i++) {
00087
00088     ss << "point_lights[" << i << "].base.color";
00089     shader_program->setUniformVec3(point_lights[i]->get_color(), ss.str());
00090     ss.clear();
00091     ss.str(std::string());
00092
00093     ss << "point_lights[" << i << "].base.radiance";
00094     shader_program->setUniformFloat(point_lights[i]->get_radiance(), ss.str());
00095     ss.clear();
00096     ss.str(std::string());
00097
00098     ss << "point_lights[" << i << "].position";
00099     shader_program->setUniformVec3(point_lights[i]->get_position(), ss.str());
00100     ss.clear();
00101     ss.str(std::string());
00102
00103     ss << "point_lights[" << i << "].base.constant";
00104     shader_program->setUniformFloat(point_lights[i]->get_constant_factor(), ss.str());
00105     ss.clear();
00106     ss.str(std::string());
00107
00108     ss << "point_lights[" << i << "].linear";
00109     shader_program->setUniformFloat(point_lights[i]->get_linear_factor(), ss.str());
00110     ss.clear();
00111     ss.str(std::string());
00112
00113     ss << "point_lights[" << i << "].exponent";
00114     shader_program->setUniformFloat(point_lights[i]->get_exponent_factor(), ss.str());
00115     ss.clear();
00116     ss.str(std::string());
00117
00118     ss << "omni_shadow_maps[" << i << "].shadow_map";
00119     shader_program->setUniformInt(GLint(P_LIGHT_SHADOW_TEXTURES_SLOT + i), ss.str());
00120     ss.clear();
00121     ss.str(std::string());
00122
00123     ss << "omni_shadow_maps[" << i << "]far_plane";
00124     shader_program->setUniformFloat(point_lights[i]->get_far_plane(), ss.str());
00125     ss.clear();
00126     ss.str(std::string());
00127 }
00128
00129 // CAMERA
00130 glm::vec3 camera_position = main_camera->get_camera_position();
00131 shader_program->setUniformVec3(camera_position, "eye_position");
00132
00133 // MATERIALS
00134 std::vector<ObjMaterial> materials = scene->get_materials();
00135 for (uint32_t i = 0; i < static_cast<uint32_t>(materials.size()); i++) {
00136
00137     ss << "materials[" << i << "].ambient";
00138     shader_program->setUniformVec3(materials[i].get_ambient(), ss.str());
00139     ss.clear();
00140     ss.str(std::string());
00141
00142     ss << "materials[" << i << "].diffuse";
00143     shader_program->setUniformVec3(materials[i].get_diffuse(), ss.str());
00144     ss.clear();
00145     ss.str(std::string());
00146
00147     ss << "materials[" << i << "].specular";
00148     shader_program->setUniformVec3(materials[i].get_specular(), ss.str());
00149     ss.clear();
00150     ss.str(std::string());
00151
00152     ss << "materials[" << i << "].transmittance";
00153     shader_program->setUniformVec3(materials[i].get_transmittance(), ss.str());
00154     ss.clear();
00155     ss.str(std::string());
00156
00157     ss << "materials[" << i << "].emission";
00158     shader_program->setUniformVec3(materials[i].get_emission(), ss.str());
00159     ss.clear();
00160     ss.str(std::string());
00161
00162     ss << "materials[" << i << "].shininess";
00163     shader_program->setUniformFloat(materials[i].get_shininess(), ss.str());
00164     ss.clear();
00165     ss.str(std::string());
00166

```

```

00167     ss << "materials[" << i << "].ior";
00168     shader_program->setUniformFloat(materials[i].get_ior(), ss.str());
00169     ss.clear();
00170     ss.str(std::string());
00171
00172     ss << "materials[" << i << "].dissolve";
00173     shader_program->setUniformFloat(materials[i].get_dissolve(), ss.str());
00174     ss.clear();
00175     ss.str(std::string());
00176
00177     ss << "materials[" << i << "].illum";
00178     shader_program->setUniformInt(materials[i].get_illum(), ss.str());
00179     ss.clear();
00180     ss.str(std::string());
00181
00182     ss << "materials[" << i << "].textureID";
00183     shader_program->setUniformInt(materials[i].get_textureID(), ss.str());
00184     ss.clear();
00185     ss.str(std::string());
00186 }
00187
00188 // CLOUDS
00189
00190 std::shared_ptr<Clouds> cloud = scene->get_clouds();
00191
00192 shader_program->setUniformVec3(cloud->get_radius(), "cloud.radius");
00193 GLfloat velocity = cloud->get_movement_speed() * delta_time;
00194 current_offset = current_offset + cloud->get_movement_direction() * velocity;
00195 shader_program->setUniformVec3(current_offset, "cloud.offset");
00196 shader_program->setUniformMatrix4fv(cloud->get_model(), "cloud.model_to_world");
00197 shader_program->setUniformFloat(cloud->get_scale(), "cloud.scale");
00198 shader_program->setUniformFloat(1.f - cloud->get_density(), "cloud.threshold");
00199 shader_program->setUniformFloat(cloud->get_pillowness(), "cloud.pillowness");
00200 shader_program->setUniformFloat(cloud->get_cirrus_effect(), "cloud.cirrus_effect");
00201 shader_program->setUniformInt(cloud->get_num_march_steps(), "cloud.num_march_steps");
00202 shader_program->setUniformInt(cloud->get_num_march_steps_to_light(),
00203 "cloud.num_march_steps_to_light");
00204 if (cloud->get_powder_effect()) {
00205     shader_program->setUniformInt(true, "cloud.powder_effect");
00206 } else {
00207     shader_program->setUniformInt(false, "cloud.powder_effect");
00208 }
00209
00210 shader_program->setUniformBlockBinding(UNIFORM_LIGHT_MATRICES_BINDING, "LightSpaceMatrices");
00211
00212 shader_program->setUniformInt(RANDOM_NUMBERS_SLOT, "random_number");
00213
00214 shader_program->setUniformInt(NOISE_128D_TEXTURES_SLOT, "noise_texture_1");
00215 shader_program->setUniformInt(NOISE_32D_TEXTURES_SLOT, "noise_texture_2");
00216
00217 shader_program->validate_program();
00218 }
00219
00220 LightingPass::~LightingPass() { }

```

7.39 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/← LightingPass.h File Reference

```

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <ctime>
#include <chrono>
#include <cassert>
#include <time.h>
#include <memory>
#include "LightingPassShaderProgram.h"
#include "Quad.h"
#include "GBuffer.h"
#include "ObjMaterial.h"
#include "Noise.h"
#include "Clouds.h"

```

```
#include "Camera.h"
#include "Scene.h"
#include "RandomNumbers.h"
```

Include dependency graph for LightingPass.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [LightingPass](#)

7.40 LightingPass.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glm/glm.hpp>
00003 #include <glm/gtc/matrix_transform.hpp>
00004 #include <glm/gtc/type_ptr.hpp>
00005
00006 #include <ctime>
00007 #include <chrono>
00008 #include <cassert>
00009 #include <time.h>
00010 #include <memory>
00011
00012 #include "LightingPassShaderProgram.h"
00013 #include "Quad.h"
00014 #include "GBuffer.h"
00015 #include "ObjMaterial.h"
00016 #include "Noise.h"
00017 #include "Clouds.h"
00018 #include "Quad.h"
00019 #include "Camera.h"
00020 #include "Scene.h"
00021 #include "RandomNumbers.h"
00022
00023 class LightingPass : public RenderPass {
00024     public:
00025     LightingPass();
00026
00027     void execute(glm::mat4 projection_matrix, std::shared_ptr<Camera>, std::shared_ptr<Scene> scene,
00028                 std::shared_ptr<GBuffer> gbuffer, float delta_time);
00029
00030     void create_shader_program();
00031
00032     ~LightingPass();
00033
00034     private:
00035     glm::vec3 current_offset;
00036
00037     void set_uniforms(
00038         glm::mat4 projection_matrix, std::shared_ptr<Camera> main_camera, std::shared_ptr<Scene> scene,
00039         std::shared_ptr<GBuffer> gbuffer, float delta_time);
00040
00041     Quad quad;
00042 };
```

7.41 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ ↵ LightingPassShaderProgram.cpp File Reference

```
#include "LightingPassShaderProgram.h"
#include "bindings.h"
Include dependency graph for LightingPassShaderProgram.cpp:
```

7.42 LightingPassShaderProgram.cpp

[Go to the documentation of this file.](#)

```
00001 #include "LightingPassShaderProgram.h"
00002 #include "bindings.h"
00003
00004 LightingPassShaderProgram::LightingPassShaderProgram() { }
00005
00006 LightingPassShaderProgram::~LightingPassShaderProgram() { }
```

7.43 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ ↵ LightingPassShaderProgram.h File Reference

```
#include "ShaderProgram.h"
#include "PointLight.h"
#include "host_device_shared.h"
```

Include dependency graph for LightingPassShaderProgram.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [LightingPassShaderProgram](#)

7.44 LightingPassShaderProgram.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "ShaderProgram.h"
00003 #include "PointLight.h"
00004 #include "host_device_shared.h"
00005
00006 class LightingPassShaderProgram : public ShaderProgram {
00007     public:
00008     LightingPassShaderProgram();
00009
0010     ~LightingPassShaderProgram();
0011
0012     private:
0013 };
```

7.45 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/ ↵ RenderPass.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- class [RenderPass](#)

7.46 RenderPass.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 class RenderPass {
00003     public:
00004     virtual void create_shader_program() = 0;
00005     private:
00006 };
00007 }
```

7.47 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_← screen>LoadingScreen.cpp File Reference

```
#include "LoadingScreen.h"
#include <sstream>
Include dependency graph for LoadingScreen.cpp:
```

7.48 LoadingScreen.cpp

[Go to the documentation of this file.](#)

```
00001 #include "LoadingScreen.h"
00002
00003 #include <sstream>
00004
00005 LoadingScreen::LoadingScreen() { create_shader_program(); }
00006
00007 void LoadingScreen::init()
00008 {
00009
00010     std::stringstream texture_base_dir;
00011     texture_base_dir << CMAKELISTS_DIR;
00012     texture_base_dir << "/Resources/Textures/";
00013
00014     std::stringstream texture_loading_screen;
00015     texture_loading_screen << texture_base_dir.str() << "Loading_Screen/ukraine.jpg";
00016
00017     loading_screen_tex = Texture(texture_loading_screen.str().c_str(), std::make_shared<RepeatMode>());
00018     loading_screen_tex.load_texture_without_alpha_channel();
00019
00020     std::stringstream texture_logo;
00021     texture_logo << texture_base_dir.str() << "Loading_Screen/Engine_logo.png";
00022     logo_tex = Texture(texture_logo.str().c_str(), std::make_shared<RepeatMode>());
00023     logo_tex.load_texture_with_alpha_channel();
00024 }
00025
00026 void LoadingScreen::render()
00027 {
00028
00029     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00030
00031     loading_screen_tex.use_texture(0);
00032
00033     loading_screen_shader_program->use_shader_program();
00034
00035     loading_screen_quad.render();
00036
00037     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00038 }
00039
00040 void LoadingScreen::create_shader_program()
00041 {
00042
00043     loading_screen_shader_program = std::make_shared<ShaderProgram>(ShaderProgram{});
00044
00045     loading_screen_shader_program->create_from_files("loading_screen/loading_screen.vert",
00046         "loading_screen/loading_screen.frag");
00047
00048 LoadingScreen::~LoadingScreen() { }
```

7.49 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen>LoadingScreen.h File Reference

```
#include "Quad.h"
#include "ShaderProgram.h"
#include "Texture.h"
```

Include dependency graph for LoadingScreen.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [LoadingScreen](#)

7.50 LoadingScreen.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "Quad.h"
00003 #include "ShaderProgram.h"
00004 #include "Texture.h"
00005
00006 class LoadingScreen {
00007 public:
00008     LoadingScreen();
00009
0010     void init();
0011     void render();
0012
0013     ~LoadingScreen();
0014
0015 private:
0016     //everything necessary for the loading screen
0017     Quad loading_screen_quad;
0018     Texture loading_screen_tex;
0019     Texture logo_tex;
0020
0021     std::shared_ptr<ShaderProgram> loading_screen_shader_program;
0022
0023     void create_shader_program();
0024 };
```

7.51 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.cpp File Reference

```
#include "Renderer.h"
#include <gsl/gsl>
```

Include dependency graph for Renderer.cpp:

7.52 Renderer.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Renderer.h"
00002 #include <gsl/gsl>
00003
00004 Renderer::Renderer(GLuint window_width, GLuint window_height) :
00005     window_width(window_width), window_height(window_height),
00006     gbuffer(std::make_shared<GBuffer>(window_width, window_height)), shader_includes(),
```

```

00007     omni_shadow_map_pass(std::make_shared<OmniShadowMapPass>()),
00008     directional_shadow_map_pass(std::make_shared<DirectionalShadowMapPass>()),
00009     geometry_pass(std::make_shared<GeometryPass>()), lighting_pass(std::make_shared<LightingPass>())
00009
00010 {
00011
00012     render_passes.push_back(omni_shadow_map_pass);
00013     render_passes.push_back(directional_shadow_map_pass);
00014     render_passes.push_back(geometry_pass);
00015     render_passes.push_back(lighting_pass);
00016
00017     gbuffer->create();
00018 }
00019
00020 void Renderer::drawFrame(std::shared_ptr<Camera> main_camera, std::shared_ptr<Scene> scene, glm::mat4
    projection_matrix, GLfloat delta_time)
00021 {
00022
00023     directional_shadow_map_pass->execute(projection_matrix, main_camera, window_width, window_height,
    scene);
00024
00025     // omni shadow map passes for our point lights
00026     std::vector<std::shared_ptr<PointLight>> p_lights = scene->get_point_lights();
00027     for (size_t p_light_count = 0; p_light_count < scene->get_point_light_count(); p_light_count++) {
00028         omni_shadow_map_pass->execute(p_lights[p_light_count], scene);
00029     }
00030
00031     //we will now start the geometry pass
00032     geometry_pass->execute(projection_matrix, main_camera, window_width, window_height,
    gbuffer->get_id(), delta_time, scene);
00033
00034     // after geometry pass we can now do the lighting
00035     lighting_pass->execute(projection_matrix, main_camera, scene, gbuffer, delta_time);
00036 }
00037
00038 void Renderer::update_window_params(GLuint window_width, GLuint window_height)
00039 {
00040     this->window_width = window_width;
00041     this->window_height = window_height;
00042     gbuffer->update_window_params(window_width, window_height);
00043     gbuffer->create();
00044 }
00045
00046 void Renderer::reload_shader_programs()
00047 {
00048     // also reload all shader include files
00049     shader_includes = ShaderIncludes();
00050
00051     for (std::shared_ptr<RenderPass> render_pass : render_passes) {
00052         render_pass->create_shader_program();
00053     }
00054 }
00055
00056 Renderer::~Renderer() { }

```

7.53 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.h File Reference

```
#include <glad/glad.h>
#include "DirectionalShadowMapPass.h"
#include "OmniShadowMapPass.h"
#include "LightingPass.h"
#include "GeometryPass.h"
#include "GBuffer.h"
#include "ShaderIncludes.h"
```

Include dependency graph for Renderer.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Renderer](#)

7.54 Renderer.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <glad/glad.h>
00004
00005 #include "DirectionalShadowMapPass.h"
00006 #include "OmniShadowMapPass.h"
00007 #include "LightingPass.h"
00008 #include "GeometryPass.h"
00009 #include "GBuffer.h"
00010 #include "ShaderIncludes.h"
00011
00012 class Renderer {
00013     public:
00014     Renderer(GLuint window_width, GLuint window_height);
00015
00016     void drawFrame(std::shared_ptr<Camera> main_camera, std::shared_ptr<Scene> scene, glm::mat4
projection_matrix, GLfloat delta_time);
00017
00018     void update_window_params(GLuint window_width, GLuint window_height);
00019     void reload_shader_programs();
00020
00021     ~Renderer();
00022
00023     private:
00024     GLuint window_width, window_height;
00025
00026     std::shared_ptr<GBuffer> gbuffer;
00027
00028     ShaderIncludes shader_includes;
00029
00030     // all render passes
00031     std::vector<std::shared_ptr<RenderPass>> render_passes;
00032
00033     std::shared_ptr<OmniShadowMapPass> omni_shadow_map_pass;
00034     std::shared_ptr<DirectionalShadowMapPass> directional_shadow_map_pass;
00035     std::shared_ptr<GeometryPass> geometry_pass;
00036     std::shared_ptr<LightingPass> lighting_pass;
00037 };
```

7.55 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneDependend.cpp File Reference

```
#include "RenderPassSceneDependend.h"
```

Include dependency graph for RenderPassSceneDependend.cpp:

7.56 RenderPassSceneDependend.cpp

[Go to the documentation of this file.](#)

```
00001 #include "RenderPassSceneDependend.h"
00002
00003 RenderPassSceneDependend::RenderPassSceneDependend() : RenderPass() { }
00004
00005 RenderPassSceneDependend::~RenderPassSceneDependend() { }
```

7.57 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneDependend.h File Reference

```
#include <vector>
#include "Model.h"
#include "RenderPass.h"
```

Include dependency graph for RenderPassSceneDependend.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [RenderPassSceneDependend](#)

7.58 RenderPassSceneDependend.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <vector>
00003
00004 #include "Model.h"
00005 #include "RenderPass.h"
00006
00007 class RenderPassSceneDependend : public RenderPass {
00008     public:
00009     RenderPassSceneDependend();
0010
0011     virtual void set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model) = 0;
0012     virtual void create_shader_program() = 0;
0013
0014     ~RenderPassSceneDependend();
0015
0016     private:
0017 };
```

7.59 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.cpp File Reference

```
#include "ShaderIncludes.h"
#include "File.h"
#include <stdio.h>
#include <string.h>
#include <array>
#include <sstream>
#include <cassert>
Include dependency graph for ShaderIncludes.cpp:
```

7.60 ShaderIncludes.cpp

[Go to the documentation of this file.](#)

```
00001 #include "ShaderIncludes.h"
00002 #include "File.h"
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <array>
00007 #include <sstream>
00008 #include <cassert>
00009
0010 // this method is setting all files we want to use in a shader per #include
0011 // you have to specify the name(how file appears in shader)
0012 // and its actual file location relatively
0013 // https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_shading_language_include.txt
0014 ShaderIncludes::ShaderIncludes()
00015 {
00016
00017     assert(includeNames.size() == file_locations_relative.size());
00018
00019     std::vector<std::string> file_locations_abs;
00020     for (uint32_t i = 0; i < static_cast<uint32_t>(includeNames.size()); i++) {
00021
00022         std::stringstream aux;
00023         aux << CMAKELISTS_DIR;
```

```

00024     aux << file_locations_relative[i];
00025     file_locations_abs.push_back(aux.str());
00026 }
00027
00028 for (uint32_t i = 0; i < static_cast<uint32_t>(includeNames.size()); i++) {
00029
00030     File file(file_locations_abs[i]);
00031     std::string file_content = file.read();
00032     char tmpstr[200];
00033     snprintf(tmpstr, 200, "%s", includeNames[i]);
00034     glNamedStringARB(GL_SHADER_INCLUDE_ARB, static_cast<GLint>(strlen(tmpstr)), tmpstr,
00035     static_cast<GLint>(strlen(file_content.c_str())), file_content.c_str());
00036 }
00037
00038 ShaderIncludes::~ShaderIncludes() { }

```

7.61 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.h File Reference

```
#include <vector>
#include <glad/glad.h>
```

Include dependency graph for ShaderIncludes.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ShaderIncludes](#)

7.62 ShaderIncludes.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <vector>
00003 #include <glad/glad.h>
00004
00005 // https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_shading_language_include.txt
00006 class ShaderIncludes {
00007
00008 public:
00009     ShaderIncludes();
00010
00011     ~ShaderIncludes();
00012
00013 private:
00014     std::vector<const char*> includeNames = { "host_device_shared.h",
00015         "Matlib.glsl",
00016         "microfacet.glsl",
00017         "ShadingLibrary.glsl",
00018         "disney.glsl",
00019         "frostbite.glsl",
00020         "pbrBook.glsl",
00021         "phong.glsl",
00022         "unreal4.glsl",
00023         "clouds.glsl",
00024         "grad_noise.glsl",
00025         "worley_noise.glsl",
00026         "bindings.h",
00027         "GlobalValues.h",
00028         "directional_light.glsl",
00029         "light.glsl",
00030         "material.glsl",
00031         "point_light.glsl" };
00032
00033     std::vector<const char*> file_locations_relative = { "/Src/host_device_shared.h",
00034         "/Resources/Shaders/common/Matlib.glsl",
00035         "/Resources/Shaders/pbr/microfacet.glsl",
00036         "/Resources/Shaders/common/ShadingLibrary.glsl",
00037         "/Resources/Shaders/pbr/brdf/disney.glsl",
00038         "/Resources/Shaders/pbr/brdf/frostbite.glsl",

```

```

00039     "/Resources/Shaders/pbr/brdf/pbrBook.glsl",
00040     "/Resources/Shaders/pbr/brdf/phong.glsl",
00041     "/Resources/Shaders/pbr/brdf/unreal4.glsl",
00042     "/Resources/Shaders/clouds/clouds.glsl",
00043     "/Resources/Shaders/common/grad_noise.glsl",
00044     "/Resources/Shaders/common/worley_noise.glsl",
00045     "/Src/bindings.h",
00046     "/Src/GlobalValues.h",
00047     "/Resources/Shaders/common/directional_light.glsl",
00048     "/Resources/Shaders/common/light.glsl",
00049     "/Resources/Shaders/common/material.glsl",
00050     "/Resources/Shaders/common/point_light.glsl" );
00051 };

```

7.63 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Shader Program.cpp File Reference

```

#include "ShaderProgram.h"
#include "File.h"
#include <sstream>
Include dependency graph for ShaderProgram.cpp:

```

7.64 ShaderProgram.cpp

[Go to the documentation of this file.](#)

```

00001 #include "ShaderProgram.h"
00002 #include "File.h"
00003
00004 #include <sstream>
00005
00006 ShaderProgram::ShaderProgram() :
00007
00008     program_id(0), vertex_location(""), fragment_location(""), geometry_location(""),
00009     compute_location("fragment_location")
00010 {
00011
00012     std::stringstream aux;
00013     aux << CMAKELISTS_DIR;
00014     aux << "/Resources/Shaders/";
00015
00016     shader_base_dir = aux.str();
00017 }
00018
00019 void ShaderProgram::create_from_files(const char* vertex_location, const char* fragment_location)
00020 {
00021
00022     std::stringstream vertex_shader;
00023     std::stringstream fragment_shader;
00024     vertex_shader << shader_base_dir << vertex_location;
00025     fragment_shader << shader_base_dir << fragment_location;
00026
00027     File vertex_shader_file(vertex_shader.str());
00028     File fragment_shader_file(fragment_shader.str());
00029
00030     std::string vertex_string = vertex_shader_file.read();
00031     std::string fragment_string = fragment_shader_file.read();
00032
00033 //we need c-like strings ....
00034     const char* vertex_code = vertex_string.c_str();
00035     const char* fragment_code = fragment_string.c_str();
00036
00037     this->vertex_location = (vertex_location);
00038     this->fragment_location = (fragment_location);
00039
00040     compile_shader_program(vertex_code, fragment_code);
00041 }
00042
00043 void ShaderProgram::create_from_files(const char* vertex_location, const char* geometry_location,
00044                                         const char* fragment_location)
00045 {

```

```
00046     std::stringstream vertex_shader;
00047     std::stringstream geometry_shader;
00048     std::stringstream fragment_shader;
00049     vertex_shader << shader_base_dir << vertex_location;
00050     geometry_shader << shader_base_dir << geometry_location;
00051     fragment_shader << shader_base_dir << fragment_location;
00052
00053     File vertex_shader_file(vertex_shader.str());
00054     File geometry_shader_file(geometry_shader.str());
00055     File fragment_shader_file(fragment_shader.str());
00056
00057     std::string vertex_string = vertex_shader_file.read();
00058     std::string geometry_string = geometry_shader_file.read();
00059     std::string fragment_string = fragment_shader_file.read();
00060
00061     const char* vertex_code = vertex_string.c_str();
00062     const char* geometry_code = geometry_string.c_str();
00063     const char* fragment_code = fragment_string.c_str();
00064
00065     this->vertex_location = vertex_location;
00066     this->fragment_location = fragment_location;
00067     this->geometry_location = geometry_location;
00068
00069     compile_shader_program(vertex_code, geometry_code, fragment_code);
00070 }
00071
00072 void ShaderProgram::create_computer_shader_program_from_file(const char* compute_location)
00073 {
00074
00075     std::stringstream comp_shader;
00076     comp_shader << shader_base_dir << compute_location;
00077     File compute_shader_file(comp_shader.str());
00078     std::string file = compute_shader_file.read();
00079
00080     const char* compute_code = file.c_str();
00081
00082     this->compute_location = compute_location;
00083
00084     compile_compute_shader_program(compute_code);
00085 }
00086
00087 GLuint ShaderProgram::get_id() const { return program_id; }
00088
00089 void ShaderProgram::validate_program()
00090 {
00091
00092     GLint result = 0;
00093     GLchar eLog[1024] = { 0 };
00094
00095     glValidateProgram(program_id);
00096
00097     glGetProgramiv(program_id, GL_VALIDATE_STATUS, &result);
00098
00099     if (!result) {
00100         glGetProgramInfoLog(program_id, sizeof(eLog), NULL, eLog);
00101         printf("Error validating program: '%s'\n", eLog);
00102         return;
00103     }
00104 }
00105
00106 void ShaderProgram::use_shader_program() { glUseProgram(program_id); }
00107
00108 void ShaderProgram::add_shader(GLuint program, const char* shader_code, GLenum shader_type)
00109 {
00110     GLuint shader = glCreateShader(shader_type);
00111
00112     // the opengl function wants c -style char array of code and the length in an array ... so we do it
00113     const GLchar* code[1];
00114     code[0] = shader_code;
00115
00116     GLint code_length[1];
00117     code_length[0] = (GLint)strlen(shader_code);
00118
00119     glShaderSource(shader, 1, code, code_length);
00120     glCompileShader(shader);
00121     //glCompileShaderIncludeARB(shader);
00122
00123     GLint result = 0;
00124     GLchar eLog[1024] = { 0 };
00125
00126     //retrieve status of the shader and print if any error occurred
00127     glGetShaderiv(shader, GL_COMPILE_STATUS, &result);
00128
00129     if (!result) {
00130         glGetShaderInfoLog(shader, sizeof(eLog), NULL, eLog);
00131         printf("Error compiling the %d shader: '%s'\n", shader_type, eLog);
00132         printf("%s", shader_code);
```

```

00133     return;
00134 }
00135
00136 // we are happy, everything went well so attach shader to program
00137 glAttachShader(program, shader);
00138 }
00139
00140 void ShaderProgram::compile_shader_program(const char* vertex_code, const char* fragment_code)
00141 {
00142 // retrieve the id; we need to reference it later on
00143 program_id = glCreateProgram();
00144
00145 if (!program_id) {
00146     printf("Error creating shader program !\n");
00147     return;
00148 }
00149 // we will always need one vertex ShaderProgram
00150 add_shader(program_id, vertex_code, GL_VERTEX_SHADER);
00151 // and one fragment ShaderProgram
00152 add_shader(program_id, fragment_code, GL_FRAGMENT_SHADER);
00153
00154 //we attached all shaders
00155 //so compile program
00156 compile_program();
00157 }
00158
00159 void ShaderProgram::compile_shader_program(const char* vertex_code, const char* geometry_code, const
00160 char* fragment_code)
00161 {
00162     program_id = glCreateProgram();
00163
00164 if (!program_id) {
00165     printf("Error creating shader program!%\n");
00166     return;
00167 }
00168
00169 add_shader(program_id, vertex_code, GL_VERTEX_SHADER);
00170 add_shader(program_id, geometry_code, GL_GEOMETRY_SHADER);
00171 add_shader(program_id, fragment_code, GL_FRAGMENT_SHADER);
00172
00173 compile_program();
00174
00175 void ShaderProgram::compile_compute_shader_program(const char* compute_code)
00176 {
00177     program_id = glCreateProgram();
00178
00179 if (!program_id) {
00180     printf("Error creating shader program!%\n");
00181     return;
00182 }
00183
00184 add_shader(program_id, compute_code, GL_COMPUTE_SHADER);
00185
00186 compile_program();
00187 }
00188
00189 void ShaderProgram::compile_program()
00190 {
00191 //as simple as that; opengl will link it for us :)
00192 glLinkProgram(program_id);
00193 GLint result = 0;
00194 GLchar eLog[1024] = { 0 };
00195
00196 glGetProgramiv(program_id, GL_LINK_STATUS, &result);
00197
00198 if (!result) {
00199     glGetProgramInfoLog(program_id, sizeof(eLog), NULL, eLog);
00200     printf("Error linking program: '%s'\n", eLog);
00201     return;
00202 }
00203
00204 validate_program();
00205 }
00206
00207 bool ShaderProgram::setUniformVec3(glm::vec3 uniform, const std::string& shaderUniformName)
00208 {
00209     bool validity = true;
00210     GLuint uniform_location = glGetUniformLocation(shaderUniformName, validity);
00211
00212     if (validity) {
00213         glUniform3f(uniform_location, uniform.x, uniform.y, uniform.z);
00214     }
00215
00216     return validity;
00217 }
00218

```

```
00219 bool ShaderProgram::setUniformFloat(GLfloat uniform, const std::string& shaderUniformName)
00220 {
00221     bool validity = true;
00222     GLuint uniform_location = glGetUniformLocation(shaderUniformName, validity);
00224
00225     if (validity) {
00226         glUniform1f(uniform_location, uniform);
00227     }
00228
00229     return validity;
00230 }
00231
00232 bool ShaderProgram::setUniformInt(GLint uniform, const std::string& shaderUniformName)
00233 {
00234     bool validity = true;
00235     GLuint uniform_location = glGetUniformLocation(shaderUniformName, validity);
00236
00237     if (validity) {
00238         glUniform1i(uniform_location, uniform);
00239     }
00240
00241     return validity;
00242 }
00243
00244 bool ShaderProgram::setUniformMatrix4fv(glm::mat4 matrix, const std::string& shaderUniformName)
00245 {
00246
00247     bool validity = true;
00248     GLuint uniform_location = glGetUniformLocation(shaderUniformName, validity);
00249
00250     if (validity) {
00251         glUniformMatrix4fv(uniform_location, 1, GL_FALSE, glm::value_ptr(matrix));
00252     }
00253
00254     return validity;
00255 }
00256
00257 bool ShaderProgram::setUniformBlockBinding(GLuint block_binding, const std::string& shaderUniformName)
00258 {
00259
00260     bool validity = true;
00261     GLint uniform_location = glGetUniformBlockIndex(program_id, shaderUniformName.c_str());
00262
00263     (uniform_location < 0) ? validity = false : validity = true;
00264
00265     if (validity) {
00266         glUniformBlockBinding(program_id, uniform_location, block_binding);
00267     } else {
00268 #ifdef NDEBUG
00269         // nondebug
00270
00271     #else
00272         //printf("Error at setting uniform block binding!");
00273     #endif
00274     }
00275
00276     return validity;
00277 }
00278
00279 bool ShaderProgram::validateUniformLocation(GLint uniformLocation)
00280 {
00281     // if uniform location is invalid (f.e. var disappears because of optimizing of unused vars)
00282     return (uniformLocation == -1) ? false : true;
00283 }
00284
00285 GLuint ShaderProgram::getUniformLocation(const std::string& shaderUniformName, bool& validity)
00286 {
00287     GLuint uniform_location = glGetUniformLocation(program_id, shaderUniformName.c_str());
00288     validity = validateUniformLocation(uniform_location);
00289
00290 #ifdef NDEBUG
00291     // nondebug
00292
00293     #else
00294
00295     if (!validity) {
00296         /*std::stringstream ss;
00297         ss << "You have set a wrong uniform!
00298             << "Name: " << shaderUniformName << "\n";
00299
00300             std::cout << ss.str();*/
00301     }
00302
00303 #endif
00304
00305     return uniform_location;
```

```

00306 }
00307
00308 void ShaderProgram::clear_shader_program()
00309 {
00310     //don't trash the id's!!
00311     //delete it from memory!!
00312     if (program_id != 0) {
00313         glDeleteProgram(program_id);
00314         program_id = 0;
00315     }
00316 }
00317
00318 ShaderProgram::~ShaderProgram() { clear_shader_program(); }

```

7.65 C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Shader Program.h File Reference

```

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <cassert>

```

Include dependency graph for ShaderProgram.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ShaderProgram](#)

7.66 ShaderProgram.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <iostream>
00003 #include <fstream>
00004 #include <sstream>
00005 #include <string>
00006 #include <vector>
00007
00008 #include <glad/glad.h>
00009 #include <glm/glm.hpp>
00010 #include <glm/gtc/type_ptr.hpp>
00011 #include <cassert>
00012
00013 class ShaderProgram {
00014     public:
00015     ShaderProgram();
00016
00017     ShaderProgram(const ShaderProgram&) = default;
00018
00019     void create_from_files(const char* vertex_location, const char* fragment_location);
00020     void create_from_files(const char* vertex_location, const char* geometry_location, const char*
fragment_location);
00021
00022     void create_computer_shader_program_from_file(const char* compute_location);
00023
00024     bool setUniformVec3(glm::vec3 uniform, const std::string& shaderUniformName);
00025     bool setUniformFloat(GLfloat uniform, const std::string& shaderUniformName);
00026     bool setUniformInt(GLint uniform, const std::string& shaderUniformName);
00027     bool setUniformMatrix4fv(glm::mat4 matrix, const std::string& shaderUniformName);
00028     bool setUniformBlockBinding(GLuint block_binding, const std::string& shaderUniformName);
00029

```

```

00030     GLuint get_id() const;
00031
00032     void use_shader_program();
00033
00034     void validate_program();
00035
00036     ~ShaderProgram();
00037
00038 protected:
00039     std::string shader_base_dir;
00040     GLuint program_id;
00041
00042     // the file locations from our shaders
00043     const char* vertex_location;
00044     const char* fragment_location;
00045     const char* geometry_location;
00046     const char* compute_location;
00047
00048     void add_shader(GLuint program, const char* shader_code, GLenum shader_type);
00049
00050     void compile_shader_program(const char* vertex_code, const char* fragment_code);
00051     void compile_shader_program(const char* vertex_code, const char* geometry_code, const char*
fragment_code);
00052
00053     void compile_compute_shader_program(const char* compute_code);
00054     void compile_program();
00055
00056     bool validateUniformLocation(GLint uniformLocation);
00057     GLuint getUniformLocation(const std::string& shaderUniformName, bool& validity);
00058
00059     void clear_shader_program();
00060 };

```

7.67 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.cpp File Reference

```
#include "AABB.h"
#include <algorithm>
Include dependency graph for AABB.cpp:
```

7.68 AABB.cpp

[Go to the documentation of this file.](#)

```

00001 #include "AABB.h"
00002 #include <algorithm>
00003
00004 AABB::AABB() { }
00005
00006 std::vector<glm::vec3> AABB::get_corners(glm::mat4 model)
00007 {
00008     std::vector<glm::vec3> corners_world_space;
00009
00010     for (glm::vec3 corner : corners) {
00011         corners_world_space.push_back(glm::vec3(model * glm::vec4(corner, 1.0f)));
00012     }
00013 }
00014
00015     return corners_world_space;
00016 }
00017
00018 void AABB::init(GLfloat minX, GLfloat maxX, GLfloat minY, GLfloat maxY, GLfloat minZ, GLfloat maxZ)
00019 {
00020
00021     this->minX = minX;
00022     this->maxX = maxX;
00023     this->minY = minY;
00024     this->maxY = maxY;
00025     this->minZ = minZ;
00026     this->maxZ = maxZ;
00027
00028     corners.push_back(glm::vec3(minX, minY, minZ));
00029     corners.push_back(glm::vec3(minX, minY, maxZ));
00030

```

```

00031 corners.push_back(glm::vec3(minX, maxY, minZ));
00032 corners.push_back(glm::vec3(minX, maxY, maxZ));
00033 corners.push_back(glm::vec3(maxX, minY, minZ));
00034 corners.push_back(glm::vec3(maxX, minY, maxZ));
00035 corners.push_back(glm::vec3(maxX, maxY, minZ));
00036 corners.push_back(glm::vec3(maxX, maxY, maxZ));
00037
00038 // 0: left bottom front
00039 vertices.push_back(Vertex(glm::vec3(minX, minY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00040 glm::vec2(0.f)));
00041 // 1: right bottom front
00042 vertices.push_back(Vertex(glm::vec3(maxX, minY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00043 glm::vec2(0.f)));
00044 // 2: left top front
00045 vertices.push_back(Vertex(glm::vec3(minX, maxY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00046 glm::vec2(0.f)));
00047 // 3: right top front
00048 vertices.push_back(Vertex(glm::vec3(maxX, maxY, maxZ), glm::vec3(0.f), glm::vec3(0.f),
00049 glm::vec2(0.f)));
00050 // 4: left bottom far
00051 vertices.push_back(Vertex(glm::vec3(minX, minY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00052 glm::vec2(0.f)));
00053 // 5: right bottom far
00054 vertices.push_back(Vertex(glm::vec3(maxX, minY, minZ), glm::vec3(0.f), glm::vec3(0.f),
00055 glm::vec2(0.f)));
00056 indices = { // note that we start from 0!
00057     //left
00058     4,
00059     2,
00060     6,
00061     4,
00062     0,
00063     2,
00064
00065     //right
00066     3,
00067     5,
00068     7,
00069     5,
00070     3,
00071     1,
00072
00073     //top
00074     2,
00075     3,
00076     6,
00077     6,
00078     3,
00079     7,
00080
00081     //bottom
00082     4,
00083     1,
00084     0,
00085     5,
00086     1,
00087     4,
00088
00089     //back
00090     7,
00091     4,
00092     6,
00093     5,
00094     4,
00095     7,
00096
00097     //front
00098     0,
00099     3,
00100     2,
00101     0,
00102     1,
00103     3
00104
00105 };
00106
00107 mesh = std::make_shared<Mesh>(vertices, indices);
00108 }
00109

```

```

00110 glm::vec3 AABB::get_radius() { return glm::vec3(std::abs(maxX - minX), std::abs(maxY - minY),
00111     std::abs(maxZ - minZ)); }
00112 void AABB::render() { mesh->render(); }
00113
00114 AABB::~AABB() { }

```

7.69 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.h File Reference

```

#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <vector>
#include <memory>
#include "Mesh.h"

```

Include dependency graph for AABB.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [AABB](#)

7.70 AABB.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <glad/glad.h>
00004 #include <GLFW/glfw3.h>
00005 #include <glm/glm.hpp>
00006 #include <glm/gtc/matrix_transform.hpp>
00007 #include <glm/gtc/type_ptr.hpp>
00008 #include <vector>
00009 #include <memory>
00010
00011 #include "Mesh.h"
00012
00013 class AABB {
00014     public:
00015     AABB();
00016
00017     std::vector<glm::vec3> get_corners(glm::mat4 model);
00018
00019     void init(GLfloat minX, GLfloat maxX, GLfloat minY, GLfloat maxY, GLfloat minZ, GLfloat maxZ);
00020
00021     glm::vec3 get_radius();
00022
00023     void render();
00024
00025     ~AABB();
00026
00027     private:
00028     std::vector<Vertex> vertices;
00029     std::vector<unsigned int> indices;
00030
00031     std::shared_ptr<Mesh> mesh;
00032
00033     std::vector<glm::vec3> corners;
00034
00035     GLfloat minX, maxX, minY, maxY, minZ, maxZ;
00036 };

```

7.71 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Clouds.cpp File Reference

```
#include "Clouds.h"
Include dependency graph for Clouds.cpp:
```

7.72 Clouds.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Clouds.h"
00002
00003 Clouds::Clouds() :
00004     shader_program(std::make_shared<ShaderProgram>(ShaderProgram())),
00005     noise(std::make_shared<Noise>()), random_numbers(std::make_shared<RandomNumbers>()),
00006     model(glm::mat4(1.f)), aabb(AABB()),
00007     minX(-1.f), maxX(1.f), minY(-1.f), maxY(1.f), minZ(-1.f), maxZ(1.f),
00008     movement_direction(glm::vec3(0.0f, 0.0f, 1.0f)), scale_factor(glm::vec3(1.f)),
00009     translation(glm::vec3(0.0f)),
00010     movement_speed(0.65f), density(0.7f), scale(0.63f), pillowness(1.0f), cirrus_effect(0.0f),
00011     num_march_steps(8), num_march_steps_to_light(3),
00012     powder_effect(true)
00013 {
00014     shader_program->create_from_files("clouds/CloudsRectangle.vert", "clouds/CloudsRectangle.frag");
00015     create_noise_textures();
00016     aabb.init(minX, maxX, minY, maxY, minZ, maxZ);
00017 }
00018
00019     void Clouds::render(glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width, GLuint
00020     window_height)
00021 {
00022     shader_program->use_shader_program();
00023     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00024     shader_program->setUniformMatrix4fv(view_matrix, "view");
00025     shader_program->setUniformMatrix4fv(get_model(), "model");
00026
00027     shader_program->validate_program();
00028
00029     //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
00030     aabb.render();
00031     //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
00032 }
00033
00034 void Clouds::read()
00035 {
00036     random_numbers->read();
00037     noise->read_res128_noise();
00038     noise->read_res32_noise();
00039 }
00040
00041
00042 void Clouds::create_noise_textures()
00043 {
00044     noise->create_res128_noise();
00045     noise->create_res32_noise();
00046 }
00047
00048
00049
00050 void Clouds::set_powder_effect(bool cloud_powder_effect) { this->powder_effect = cloud_powder_effect;
00051 }
00052
00053 void Clouds::set_cirrus_effect(GLfloat cirrus_effect) { this->cirrus_effect = cirrus_effect; }
00054
00055 void Clouds::set_pillowness(GLfloat cloud_pillowness) { this->pillowness = cloud_pillowness; }
00056
00057 void Clouds::set_scale(GLfloat scale) { this->scale = scale; }
00058
00059 void Clouds::set_density(GLfloat density) { this->density = density; }
```

```

00065
00066 void Clouds::set_movement_speed(GLfloat speed) { movement_speed = speed; }
00067
00068 void Clouds::set_scale(glm::vec3 scale) { scale_factor = scale; }
00069
00070 void Clouds::set_translation(glm::vec3 translation) { this->translation = translation; }
00071
00072 glm::mat4 Clouds::get_model() const
00073 {
00074     glm::mat4 model = glm::mat4(1.f);
00075     model = glm::translate(model, translation);
00076     model = glm::scale(model, scale_factor);
00077     return model;
00078 }
00079
00080 void Clouds::set_movement_direction(glm::vec3 movement_dir) { this->movement_direction = movement_dir;
}
00081
00082 void Clouds::set_num_march_steps(GLuint num_march_steps) { this->num_march_steps = num_march_steps; }
00083
00084 void Clouds::set_num_march_steps_to_light(GLuint num_march_steps_to_light) {
    this->num_march_steps_to_light = num_march_steps_to_light;
}
00085
00086 Clouds::~Clouds() { }
```

7.73 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Clouds.h File Reference

```
#include <memory>
#include "AABB.h"
#include "ShaderProgram.h"
#include "Noise.h"
#include "RandomNumbers.h"
```

Include dependency graph for Clouds.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Clouds](#)

7.74 Clouds.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <memory>
00003
00004 #include "AABB.h"
00005 #include "ShaderProgram.h"
00006 #include "Noise.h"
00007 #include "RandomNumbers.h"
00008
00009 class Clouds {
00010 public:
00011     Clouds();
00012
00013     void render(glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width, GLuint
window_height);
00014
00015     void read();
00016
00017     void create_noise_textures();
00018
00019     glm::mat4 get_model() const;
00020     glm::vec3 get_movement_direction() const { return movement_direction; };
00021     glm::vec3 get_radius() const { return scale_factor / 2.f; };
00022     glm::mat4 get_normal_model() const { return glm::transpose(glm::inverse(model)); };
00023     glm::vec3 get_mesh_scale() const { return scale_factor; };
00024     GLfloat get_movement_speed() const { return movement_speed; };
```

```

00025     GLfloat get_density() const { return density; };
00026     GLfloat get_scale() const { return scale; };
00027     GLfloat get_pillowness() const { return pillowness; };
00028     GLfloat get_cirrus_effect() const { return cirrus_effect; };
00029     GLuint get_num_march_steps() const { return num_march_steps; };
00030     GLuint get_num_march_steps_to_light() const { return num_march_steps_to_light; };
00031     bool get_powder_effect() const { return powder_effect; };
00032
00033     std::shared_ptr<ShaderProgram> get_shader_program() const { return shader_program; };
00034
00035     void set_powder_effect(bool cloud_powder_effect);
00036     void set_cirrus_effect(GLfloat cirrus_effect);
00037     void set_pillowness(GLfloat cloud_pillowness);
00038     void set_scale(GLfloat scale);
00039     void set_density(GLfloat density);
00040     void set_movement_speed(GLfloat speed);
00041     void set_scale(glm::vec3 scale);
00042     void set_translation(glm::vec3 translation);
00043     void set_movement_direction(glm::vec3 movement_dir);
00044     void set_num_march_steps(GLuint num_march_steps);
00045     void set_num_march_steps_to_light(GLuint num_march_steps_to_light);
00046
00047 ~Clouds();
00048
00049 private:
00050     std::shared_ptr<ShaderProgram> shader_program;
00051     std::shared_ptr<Noise> noise;
00052     std::shared_ptr<RandomNumbers> random_numbers;
00053
00054     glm::mat4 model;
00055     AABB aabb;
00056
00057     GLfloat minX, maxX, minY, maxY, minZ, maxZ;
00058
00059     glm::vec3 movement_direction;
00060     glm::vec3 scale_factor, translation;
00061
00062     GLfloat movement_speed, density, scale, pillowness, cirrus_effect;
00063
00064     GLuint num_march_steps, num_march_steps_to_light;
00065
00066     bool powder_effect;
00067 };

```

7.75 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Noise.cpp File Reference

```
#include "Noise.h"
#include <sstream>
Include dependency graph for Noise.cpp:
```

7.76 Noise.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Noise.h"
00002
00003 #include <sstream>
00004
00005 Noise::Noise() :
00006     texture_dim_1(128), texture_dim_2(32)
00008
00009 {
00010
00011     create_shader_programs();
00012
00013     // we need 3d-voxel grids with different sizes for
00014     // creating different worley frequencies
00015     for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00016
00017         num_cells_per_axis[i] = static_cast<GLuint>(pow(2, i + 1));
00018         generate_cells(num_cells_per_axis[i], i);
00019     }

```

```
00020
00021     generate_textures();
00022 }
00023
00024 void Noise::create_shader_programs()
00025 {
00026     texture_1_shader_program = std::make_shared<ComputeShaderProgram>();
00027     texture_2_shader_program = std::make_shared<ComputeShaderProgram>();
00028
00029     texture_1_shader_program->create_computer_shader_program_from_file("clouds/noise_texture_128_res.comp");
00030     texture_2_shader_program->create_computer_shader_program_from_file("clouds/noise_texture_32_res.comp");
00031 }
00032
00033 void Noise::generate_textures()
00034 {
00035
00036     generate_num_cells_textures();
00037     generate_res128_noise_texture();
00038     generate_res32_noise_texture();
00039 }
00040
00041 void Noise::generate_num_cells_textures()
00042 {
00043     glGenTextures(NUM_CELL_POSITIONS, cell_ids);
00044
00045     for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00046
00047         glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00048
00049         glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, num_cells_per_axis[i], num_cells_per_axis[i],
00050             num_cells_per_axis[i], 0, GL_RGBA, GL_FLOAT, cell_data[i].data());
00051
00052         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00053         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00054         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00055         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00056         glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00057
00058     }
00059 }
00060
00061 void Noise::generate_res128_noise_texture()
00062 {
00063     glGenTextures(1, &texture_1_id);
00064
00065     glActiveTexture(GL_TEXTURE0 + NOISE_128D_TEXTURES_SLOT);
00066     glBindTexture(GL_TEXTURE_3D, texture_1_id);
00067
00068     glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, texture_dim_1, texture_dim_1, texture_dim_1, 0, GL_RGBA,
00069                 GL_FLOAT, NULL);
00070
00071     glBindImageTexture(NOISE_128D_IMAGE_SLOT, texture_1_id, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA32F);
00072
00073     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00074     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00075     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
00076     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00077     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00078
00079 }
00080
00081 void Noise::generate_res32_noise_texture()
00082 {
00083
00084     glGenTextures(1, &texture_2_id);
00085
00086     glActiveTexture(GL_TEXTURE0 + NOISE_32D_TEXTURES_SLOT);
00087     glBindTexture(GL_TEXTURE_3D, texture_2_id);
00088
00089     glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA32F, texture_dim_2, texture_dim_2, texture_dim_2, 0, GL_RGBA,
00090                 GL_FLOAT, NULL);
00091
00092     glBindImageTexture(NOISE_32D_IMAGE_SLOT, texture_2_id, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA32F);
00093
00094     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00095     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00096     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
00097     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00098     glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00099
00100 }
```

```

00102 void Noise::print_comp_shader_capabilities()
00103 {
00104     int work_grp_cnt[3];
00105
00106     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 0, &work_grp_cnt[0]);
00107     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 1, &work_grp_cnt[1]);
00108     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_COUNT, 2, &work_grp_cnt[2]);
00109
00110     printf("max global (total) work group counts x:%i y:%i z:%i\n", work_grp_cnt[0], work_grp_cnt[1],
00111         work_grp_cnt[2]);
00112
00113     int work_grp_size[3];
00114
00115     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 0, &work_grp_size[0]);
00116     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 1, &work_grp_size[1]);
00117     glGetIntegeri_v(GL_MAX_COMPUTE_WORK_GROUP_SIZE, 2, &work_grp_size[2]);
00118
00119     printf("max local (in one shader) work group sizes x:%i y:%i z:%i\n", work_grp_size[0],
00120         work_grp_size[1], work_grp_size[2]);
00121
00122 void Noise::update()
00123 {
00124     delete_textures();
00125
00126     texture_1_shader_program->reload();
00127     texture_2_shader_program->reload();
00128
00129     for (int i = 0; i < NUM_CELL_POSITIONS; i++) {
00130
00131         generate_cells(num_cells_per_axis[i], i);
00132     }
00133
00134     generate_textures();
00135
00136     create_res128_noise();
00137     create_res32_noise();
00138
00139 }
00140
00141 void Noise::set_num_cells(GLuint num_cells_per_axis, GLuint index) { this->num_cells_per_axis[index] =
00142     num_cells_per_axis; }
00143 /**
00144     @ num_cells_per_axis:    current voxel grid dimension
00145     @ cell_index:           index into global array for all voxel grids
00146
00147     needed for generating different worley frequencies later on
00148 */
00149 */
00150 void Noise::generate_cells(GLuint num_cells_per_axis, GLuint cell_index)
00151 {
00152
00153     cell_data[cell_index].reserve(num_cells_per_axis * num_cells_per_axis * num_cells_per_axis * 4);
00154
00155     // guess which birthday this is ;
00156     std::mt19937_64 gen64(25121995);
00157     std::uniform_real_distribution<float> dis(0, 1);
00158
00159     //depth
00160     for (int i = 0; i < static_cast<int>(num_cells_per_axis); i++) {
00161         //height
00162         for (int k = 0; k < static_cast<int>(num_cells_per_axis); k++) {
00163             //width
00164             for (int m = 0; m < static_cast<int>(num_cells_per_axis); m++) {
00165
00166                 // from: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glTexImage3D.xhtml
00167                 // "The first element corresponds to the lower left corner of the texture image.
00168                 // Subsequent elements progress left-to-right through the remaining texels in the lowest row
00169                 // of the texture image, and then in successively higher rows of the texture image.
00170                 // The final element corresponds to the upper right corner of the texture image."
00171
00172                 const GLfloat random_offset[3] = { dis(gen64), dis(gen64), dis(gen64) };
00173
00174                 GLfloat position[3] = { (m + random_offset[0]), (k + random_offset[1]), (i + random_offset[2]) };
00175
00176                 cell_data[cell_index].push_back(position[0]);
00177                 cell_data[cell_index].push_back(position[1]);
00178                 cell_data[cell_index].push_back(position[2]);
00179                 cell_data[cell_index].push_back(1.0f);
00180
00181                 // i leave this more c-style approach for my further me
00182                 // to clarify things :
00183                 //GLuint index = (i + num_cells_per_axis * (k + m * num_cells_per_axis)) * 4;
00184                 /*GLfloat position[3] = { (i + random_offset[0]),

```

```

00185     random_offset[1]),
00186     random_offset[2]));*/
00187
00188     /*cell_data[cell_index][index] = position[0];
00189      cell_data[cell_index][index + 1] = position[1];
00190      cell_data[cell_index][index + 2] = position[2];
00191      cell_data[cell_index][index + 3] = 1.0f;*/
00192 }
00193 }
00194 }
00195 }
00196
00197 void Noise::create_res128_noise()
00198 {
00199     texture_1_shader_program->use_shader_program();
00200
00201     texture_1_shader_program->setUniformInt(NOISE_128D_IMAGE_SLOT, "noise");
00202
00203     std::stringstream ss;
00204
00205     for (uint32_t i = 0; i < NUM_CELL_POSITIONS; i++) {
00206
00207         ss << "cell_positions[" << i << "]";
00208         texture_1_shader_program->setUniformInt(NOISE_CELL_POSITIONS_SLOT + i, ss.str());
00209         ss.clear();
00210         ss.str(std::string());
00211
00212         ss << "num_cells[" << i << "]";
00213         texture_1_shader_program->setUniformInt(num_cells_per_axis[i], ss.str());
00214         ss.clear();
00215         ss.str(std::string());
00216
00217         glActiveTexture(GL_TEXTURE0 + NOISE_CELL_POSITIONS_SLOT + i);
00218         glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00219     }
00220 }
00221
00222     glDispatchCompute((GLuint)texture_dim_1, (GLuint)texture_dim_1, (GLuint)texture_dim_1);
00223
00224 // make sure writing to image has finished before read
00225 glMemoryBarrier(GL_SHADER_IMAGE_ACCESS_BARRIER_BIT);
00226 //glMemoryBarrier(GL_ALL_BARRIER_BITS);
00227
00228 glBindTexture(GL_TEXTURE_3D, 0);
00229 }
00230
00231 void Noise::create_res32_noise()
00232 {
00233
00234     texture_2_shader_program->use_shader_program();
00235
00236     texture_2_shader_program->setUniformInt(NOISE_32D_IMAGE_SLOT, "noise");
00237
00238     std::stringstream ss;
00239
00240     for (uint32_t i = 0; i < NUM_CELL_POSITIONS; i++) {
00241
00242         ss << "cell_positions[" << i << "]";
00243         texture_2_shader_program->setUniformInt(NOISE_CELL_POSITIONS_SLOT + i, ss.str());
00244         ss.clear();
00245         ss.str(std::string());
00246
00247         ss << "num_cells[" << i << "]";
00248         texture_2_shader_program->setUniformInt(num_cells_per_axis[i], ss.str());
00249         ss.clear();
00250         ss.str(std::string());
00251
00252         glActiveTexture(GL_TEXTURE0 + NOISE_CELL_POSITIONS_SLOT + i);
00253         glBindTexture(GL_TEXTURE_3D, cell_ids[i]);
00254     }
00255
00256     glDispatchCompute((GLuint)texture_dim_2, (GLuint)texture_dim_2, (GLuint)texture_dim_2);
00257
00258 // make sure writing to image has finished before read
00259 glMemoryBarrier(GL_SHADER_IMAGE_ACCESS_BARRIER_BIT);
00260 //glMemoryBarrier(GL_ALL_BARRIER_BITS);
00261 glBindTexture(GL_TEXTURE_3D, 0);
00262 }
00263
00264 void Noise::read_res128_noise()
00265 {
00266     GLuint texture_index = GL_TEXTURE0 + NOISE_128D_TEXTURES_SLOT;
00267     glActiveTexture((GLenum)texture_index);
00268     glBindTexture(GL_TEXTURE_3D, texture_1_id);
00269 }
```

```

00270
00271 void Noise::read_res32_noise()
00272 {
00273     GLuint texture_index = GL_TEXTURE0 + NOISE_32D_TEXTURES_SLOT;
00274     glActiveTexture((GLenum)texture_index);
00275     glBindTexture(GL_TEXTURE_3D, texture_2_id);
00276 }
00277
00278 void Noise::delete_textures()
00279 {
00280     glDeleteTextures(1, &texture_1_id);
00281     glDeleteTextures(1, &texture_2_id);
00282
00283     glDeleteTextures(NUM_CELL_POSITIONS, cell_ids);
00284
00285 }
00286
00287 Noise::~Noise() { delete_textures(); }

```

7.77 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Noise.h File Reference

```

#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <random>
#include <memory>
#include <array>
#include <vector>
#include "ComputeShaderProgram.h"
#include "bindings.h"

```

Include dependency graph for Noise.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Noise](#)

7.78 Noise.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <glad/glad.h>
00004 #include <GLFW/glfw3.h>
00005 #include <glm/glm.hpp>
00006 #include <random>
00007 #include <memory>
00008 #include <array>
00009 #include <vector>
00010
00011 #include "ComputeShaderProgram.h"
00012 #include "bindings.h"
00013
00014 // inspired by:
// http://advances.realtimerendering.com/s2015/The%20Real-time%20Volumetric%20Cloudscapes%20of%20Horizon%20-%20Zero%20Dawn
00015 class Noise {
00016 public:
00017     Noise();
00018
00019     void create_res128_noise();
00020     void create_res32_noise();
00021
00022     void read_res128_noise();
00023     void read_res32_noise();
00024

```

```

00025     void update();
00026
00027     void set_num_cells(GLuint num_cells_per_axis, GLuint index);
00028
00029     ~Noise();
00030
00031     private:
00032     void create_shader_programs();
00033     void generate_cells(GLuint num_cells_per_axis, GLuint cell_index);
00034
00035     void generate_textures();
00036     void generate_num_cells_textures();
00037     void generate_res128_noise_texture();
00038     void generate_res32_noise_texture();
00039
00040     void delete_textures();
00041
00042     void print_comp_shader_capabilities();
00043
00044     // first texture dim = 128^3
00045     GLuint texture_1_id;
00046     GLuint texture_dim_1;
00047     std::shared_ptr<ComputeShaderProgram> texture_1_shader_program;
00048
00049     // 2nd texture dim = 32^3
00050     GLuint texture_2_id;
00051     GLuint texture_dim_2;
00052     std::shared_ptr<ComputeShaderProgram> texture_2_shader_program;
00053
00054     GLuint cell_ids[NUM_CELL_POSITIONS];
00055     GLuint num_cells_per_axis[NUM_CELL_POSITIONS];
00056     std::array<std::vector<GLfloat>, NUM_CELL_POSITIONS

```

7.79 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Perlin.cpp File Reference

7.80 Perlin.cpp

[Go to the documentation of this file.](#)

```

00001 // #include <vector>
00002 // #include "perlin.h"
00003
00004 //
00005 //
00006     /// fade(t)= t^3 (10 + t * (t* 6 - 15))
00007     //double PERLIN_H::fade(double t) {
00008     //
00009     //     return t * t * t * (t * (t * 6 - 15) + 10);
00010    //};
00011 //
00012     /// linear interpolation between a and b: For polishing
00013     /// t = [0,1] is the persentage of the distance bewteen a and b
00014     //double PERLIN_H::lerp(double t, double a, double b) {
00015     //     return a + (b - a) * t;
00016    //}
00017 //
00018 //
00019 ///
00020     //double PERLIN_H::grad(int hash, double x, double y, double z) {
00021     //     int h = hash & 15;                                // CONVERT LO 4 BITS OF HASH CODE
00022     //     double u = h < 8 ? x : y,                         // INTO 12 GRADIENT DIRECTIONS.
00023     //     v = h < 4 ? y : h == 12 || h == 14 ? x : z;
00024     //     return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v);
00025    //}
00026 //
00027 //
00028     /// generate Perlin noise
00029     /// Parameter:
00030     /// p is (psyodo) "random" value
00031     //double PERLIN_H::perlin_noise(float x, float y, std::vector<int>& p) {
00032     //     int z = 0.5;
00033     //
00034     //     // x,y,z abrunden und mit 1111 1111 verunden == modulo 255
00035     //     int X = (int)floor(x) & 255,                      // FIND UNIT CUBE THAT
00036     //             Y = (int)floor(y) & 255,                      // CONTAINS POINT.
00037     //             Z = (int)floor(z) & 255;

```

```

00038 //
00039 //    // Kommastellen
00040 //    x -= floor(x);
00041 //    y -= floor(y);
00042 //    z -= floor(z);
00043 //
00044 //    double u = fade(x),                                // COMPUTE FADE CURVES
00045 //        v = fade(y),                                // FOR EACH OF X,Y,Z.
00046 //        w = fade(z);
00047 //
00048 //    int A = p[X] + Y, AA = p[A] + Z, AB = p[A + 1] + Z,      // HASH COORDINATES OF
00049 //        B = p[X + 1] + Y, BA = p[B] + Z, BB = p[B + 1] + Z;      // THE 8 CUBE CORNERS,
00050 //
00051 //    // return perlin noise of x and y
00052 //    return lerp(w,
00053 //        lerp(v, lerp(u, grad(p[AA], x, y, z), // AND ADD
00054 //            grad(p[BA], x - 1, y, z)), // BLENDED
00055 //            lerp(u, grad(p[AB], x, y - 1, z), // RESULTS
00056 //                grad(p[BB], x - 1, y - 1, z))), // FROM 8
00057 //        lerp(v, lerp(u, grad(p[AA + 1], x, y, z - 1), // CORNERS
00058 //            grad(p[BA + 1], x - 1, y, z - 1)), // OF CUBE
00059 //            lerp(u, grad(p[AB + 1], x, y - 1, z - 1),
00060 //                grad(p[BB + 1], x - 1, y - 1, z - 1))));}
00061 //}
00062 //
00063 //void PERLIN_H::get_Permutation_Vector(std::vector<int>* permVectorPointer) {
00064 //    permVectorPointer->clear();
00065 //
00066 //    std::vector<int> permutation = { 151,160,137,91,90,15,
00067 //        131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
00068 //        190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
00069 //        88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
00070 //        77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
00071 //        102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
00072 //        135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
00073 //        5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
00074 //        223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
00075 //        129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
00076 //        251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
00077 //        49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
00078 //        138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
00079 //    };
00080 //
00081 //    for (int j = 0; j < 2; j++)
00082 //        for (int i = 0; i < 256; i++) {
00083 //            permVectorPointer -> push_back(permutation[i]);
00084 //        }
00085 //}
00086 //
00087 //
00088 //std::vector<int> PERLIN_H::my
00089 ///// initialise 2x permutation of a hard coded 256 int vector
00090 //std::vector<int> PERLIN_H::kansei() {
00091 /////
00092 ///*
00093 //std::vector<int> PERLIN_H::myFunction() {
00094 //    std::vector<int> p;
00095 //
00096 //    std::vector<int> permutation = { 151,160,137,91,90,15,
00097 //        131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,
00098 //        190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,
00099 //        88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,
00100 //        77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,
00101 //        102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,
00102 //        135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,
00103 //        5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,
00104 //        223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,
00105 //        129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,
00106 //        251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,
00107 //        49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,
00108 //        138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180
00109 //    };
00110 //
00111 //    for (int j = 0; j < 2; j++)
00112 //        for (int i = 0; i < 256; i++) {
00113 //            p.push_back(permutation[i]);
00114 //        }
00115 //
00116 //    return p;
00117 //*/
00118 //*/

```

7.81 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmospheric_effects/clouds/Perlin.h File Reference

```
#include <vector>
Include dependency graph for Perlin.h:
```

Functions

- double [fade](#) (double t)
- double [lerp](#) (double t, double a, double b)
- double [grad](#) (int hash, double x, double y, double z)
- double [perlin_noise](#) (float x, float y, std::vector< int > &p)
- std::vector< int > [getPermutationVector](#) ()

7.81.1 Function Documentation

7.81.1.1 [fade\(\)](#)

```
double fade (
    double t )
```

Definition at line 27 of file [Perlin.h](#).

```
00027 { return t * t * t * (t * (t * 6 - 15) + 10); };
```

Referenced by [perlin_noise\(\)](#).

Here is the caller graph for this function:

7.81.1.2 [getPermutationVector\(\)](#)

```
std::vector< int > getPermutationVector ( )
```

Definition at line 115 of file [Perlin.h](#).

```
00116 {
00117     std::vector<int> p;
00118
00119     std::vector<int> permutation = { 151,
00120         160,
00121         137,
00122         91,
00123         90,
00124         15,
00125         131,
00126         13,
00127         201,
00128         95,
00129         96,
00130         53,
00131         194,
00132         233,
00133         7,
00134         225,
00135         140,
00136         36,
00137         103,
00138         30,
```

00139 69,
00140 142,
00141 8,
00142 99,
00143 37,
00144 240,
00145 21,
00146 10,
00147 23,
00148 190,
00149 6,
00150 148,
00151 247,
00152 120,
00153 234,
00154 75,
00155 0,
00156 26,
00157 197,
00158 62,
00159 94,
00160 252,
00161 219,
00162 203,
00163 117,
00164 35,
00165 11,
00166 32,
00167 57,
00168 177,
00169 33,
00170 88,
00171 237,
00172 149,
00173 56,
00174 87,
00175 174,
00176 20,
00177 125,
00178 136,
00179 171,
00180 168,
00181 68,
00182 175,
00183 74,
00184 165,
00185 71,
00186 134,
00187 139,
00188 48,
00189 27,
00190 166,
00191 77,
00192 146,
00193 158,
00194 231,
00195 83,
00196 111,
00197 229,
00198 122,
00199 60,
00200 211,
00201 133,
00202 230,
00203 220,
00204 105,
00205 92,
00206 41,
00207 55,
00208 46,
00209 245,
00210 40,
00211 244,
00212 102,
00213 143,
00214 54,
00215 65,
00216 25,
00217 63,
00218 161,
00219 1,
00220 216,
00221 80,
00222 73,
00223 209,
00224 76,
00225 132,

```
00226    187,  
00227    208,  
00228    89,  
00229    18,  
00230    169,  
00231    200,  
00232    196,  
00233    135,  
00234    130,  
00235    116,  
00236    188,  
00237    159,  
00238    86,  
00239    164,  
00240    100,  
00241    109,  
00242    198,  
00243    173,  
00244    186,  
00245    3,  
00246    64,  
00247    52,  
00248    217,  
00249    226,  
00250    250,  
00251    124,  
00252    123,  
00253    5,  
00254    202,  
00255    38,  
00256    147,  
00257    118,  
00258    126,  
00259    255,  
00260    82,  
00261    85,  
00262    212,  
00263    207,  
00264    206,  
00265    59,  
00266    227,  
00267    47,  
00268    16,  
00269    58,  
00270    17,  
00271    182,  
00272    189,  
00273    28,  
00274    42,  
00275    223,  
00276    183,  
00277    170,  
00278    213,  
00279    119,  
00280    248,  
00281    152,  
00282    2,  
00283    44,  
00284    154,  
00285    163,  
00286    70,  
00287    221,  
00288    153,  
00289    101,  
00290    155,  
00291    167,  
00292    43,  
00293    172,  
00294    9,  
00295    129,  
00296    22,  
00297    39,  
00298    253,  
00299    19,  
00300    98,  
00301    108,  
00302    110,  
00303    79,  
00304    113,  
00305    224,  
00306    232,  
00307    178,  
00308    185,  
00309    112,  
00310    104,  
00311    218,  
00312    246,
```

```
00313     97,
00314     228,
00315     251,
00316     34,
00317     242,
00318     193,
00319     238,
00320     210,
00321     144,
00322     12,
00323     191,
00324     179,
00325     162,
00326     241,
00327     81,
00328     51,
00329     145,
00330     235,
00331     249,
00332     14,
00333     239,
00334     107,
00335     49,
00336     192,
00337     214,
00338     31,
00339     181,
00340     199,
00341     106,
00342     157,
00343     184,
00344     84,
00345     204,
00346     176,
00347     115,
00348     121,
00349     50,
00350     45,
00351     127,
00352     4,
00353     150,
00354     254,
00355     138,
00356     236,
00357     205,
00358     93,
00359     222,
00360     114,
00361     67,
00362     29,
00363     24,
00364     72,
00365     243,
00366     141,
00367     128,
00368     195,
00369     78,
00370     66,
00371     215,
00372     61,
00373     156,
00374     180 };
00375
00376     for (int j = 0; j < 2; j++)
00377         for (int i = 0; i < 256; i++) {
00378             p.push_back(permuation[i]);
00379         }
00380
00381     return p;
00382 }
```

7.81.1.3 grad()

```
double grad (
    int hash,
    double x,
    double y,
    double z )
```

Definition at line 35 of file [Perlin.h](#).

```
00036 {
00037     int h = hash & 15; // CONVERT LO 4 BITS OF HASH CODE
00038     double u = h < 8 ? x : y, // INTO 12 GRADIENT DIRECTIONS.
00039     v = h < 4 ? y
00040     : h == 12 || h == 14 ? x
00041     : z;
00042     return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v);
00043 }
```

Referenced by [perlin_noise\(\)](#).

Here is the caller graph for this function:

7.81.1.4 lerp()

```
double lerp (
    double t,
    double a,
    double b )
```

Definition at line 31 of file [Perlin.h](#).

```
00031 { return a + (b - a) * t; }
```

Referenced by [perlin_noise\(\)](#).

Here is the caller graph for this function:

7.81.1.5 perlin_noise()

```
double perlin_noise (
    float x,
    float y,
    std::vector< int > & p )
```

Definition at line 49 of file [Perlin.h](#).

```
00050 {
00051     int z = 0.5;
00052
00053     // x,y,z abrunden und mit 1111 1111 verunden == modulo 255
00054     int X = (int)floor(x) & 255, // FIND UNIT CUBE THAT
00055     Y = (int)floor(y) & 255, // CONTAINS POINT.
00056     Z = (int)floor(z) & 255;
00057
00058     // Kommastellen
00059     x -= floor(x); // FIND RELATIVE X,Y,Z
00060     y -= floor(y); // OF POINT IN CUBE.
00061     z -= floor(z);
00062
00063     double u = fade(x), // COMPUTE FADE CURVES
00064     v = fade(y), // FOR EACH OF X,Y,Z.
00065     w = fade(z);
00066
00067     int A = p[X] + Y, AA = p[A] + Z, AB = p[A + 1] + Z, // HASH COORDINATES OF
00068     B = p[X + 1] + Y, BA = p[B] + Z, BB = p[B + 1] + Z; // THE 8 CUBE CORNERS,
00069
00070     // return perlin noise of x and y
00071     return lerp(w,
00072         lerp(v,
00073             lerp(u,
00074                 grad(p[AA], x, y, z), // AND ADD
00075                 grad(p[BA], x - 1, y, z)), // BLENDED
00076                 lerp(u,
00077                     grad(p[AB], x, y - 1, z), // RESULTS
00078                     grad(p[BB], x - 1, y - 1, z))), // FROM 8
00079                 lerp(v,
00080                     lerp(u,
00081                         grad(p[AA + 1], x, y, z - 1), // CORNERS
00082                         grad(p[BA + 1], x - 1, y, z - 1)), // OF CUBE
00083                         lerp(u, grad(p[AB + 1], x, y - 1, z - 1), grad(p[BB + 1], x - 1, y - 1, z - 1))));
```

References [fade\(\)](#), [grad\(\)](#), and [lerp\(\)](#).

Here is the call graph for this function:

7.82 Perlin.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <vector>
00003 /**
00004 ///////////////////////////////////////////////////
00005 ///////////////////////////////////////////////////
00006 //
00007 //double fade(double t);
00008 //double lerp(double t, double a, double b);
00009 //double grad(int hash, double x, double y, double z);
00010 //double perlin_noise(float x, float y, std::vector<int>& p);
00011 //
00012 ///// tests
00013 ////// something is wrong with this vector return value
00014 //std::vector<int> myFunction(int value, double bullshit) {
00015 //    std::vector<int> r{ 2,3,4,4 };
00016 //    return r;
00017 //}
00018 //
00019 //void get_Permutation_Vector(std::vector<int>* pVectorPointer);
00020 //
00021 //
00022 //
00023 ///////////////////////////////////////////////////
00024 //endif // end PERLIN_H if
00025
00026 // fade(t)= t^3 (10 + t* (t* 6 - 15))
00027 double fade(double t) { return t * t * t * (t * (t * 6 - 15) + 10); }
00028
00029 // linear interpolation between a and b: For polishing
00030 // t = [0,1] is the percentage of the distance bewteen a and b
00031 double lerp(double t, double a, double b) { return a + (b - a) * t; }
00032
00033
00034 //
00035 double grad(int hash, double x, double y, double z)
00036 {
00037     int h = hash & 15; // CONVERT LO 4 BITS OF HASH CODE
00038     double u = h < 8 ? x : y, // INTO 12 GRADIENT DIRECTIONS.
00039             v = h < 4 ? y
00040             : h == 12 || h == 14 ? x
00041             : z;
00042     return ((h & 1) == 0 ? u : -u) + ((h & 2) == 0 ? v : -v);
00043 }
00044
00045
00046 // generate Perlin noise
00047 // Parameter:
00048 // p is (psyodo) "random" value
00049 double perlin_noise(float x, float y, std::vector<int>& p)
00050 {
00051     int z = 0.5;
00052
00053     // x,y,z abrunden und mit 1111 1111 verunden == modulo 255
00054     int X = (int)floor(x) & 255, // FIND UNIT CUBE THAT
00055         Y = (int)floor(y) & 255, // CONTAINS POINT.
00056         Z = (int)floor(z) & 255;
00057
00058     // Kommastellen
00059     x -= floor(x); // FIND RELATIVE X,Y,Z
00060     y -= floor(y); // OF POINT IN CUBE.
00061     z -= floor(z);
00062
00063     double u = fade(x), // COMPUTE FADE CURVES
00064         v = fade(y), // FOR EACH OF X,Y,Z.
00065         w = fade(z);
00066
00067     int A = p[X] + Y, AA = p[A] + Z, AB = p[A + 1] + Z, // HASH COORDINATES OF
00068         B = p[X + 1] + Y, BA = p[B] + Z, BB = p[B + 1] + Z; // THE 8 CUBE CORNERS,
00069
00070     // return perlin noise of x and y
00071     return lerp(w,
00072         lerp(v,
00073             lerp(u,
00074                 grad(p[AA], x, y, z), // AND ADD
00075                 grad(p[BA], x - 1, y, z)), // BLENDED
00076                 lerp(u,
00077                     grad(p[AB], x, y - 1, z), // RESULTS
00078                     grad(p[BB], x - 1, y - 1, z))), // FROM 8
00079         lerp(v,
00080             lerp(u,
00081                 grad(p[AA + 1], x, y, z - 1), // CORNERS
00082                 grad(p[BA + 1], x - 1, y, z - 1))), // OF CUBE

```

```
00083     lerp(u, grad(p[AB + 1], x, y - 1, z - 1), grad(p[BB + 1], x - 1, y - 1, z - 1))));  
00084 }  
00085 //  
00086 //void get_Permutation_Vector(std::vector<int>* permVectorPointer) {  
00087 //    permVectorPointer->clear();  
00088 //  
00089 //    std::vector<int> permutation = { 151,160,137,91,90,15,  
00090 //        131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10,23,  
00091 //        190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177,33,  
00092 //        88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,166,  
00093 //        77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,244,  
00094 //        102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,196,  
00095 //        135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,123,  
00096 //        5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28,42,  
00097 //        223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,172,9,  
00098 //        129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,228,  
00099 //        251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,107,  
00100 //        49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,254,  
00101 //        138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180  
00102 //    };  
00103 //  
00104 //    for (int j = 0; j < 2; j++)  
00105 //        for (int i = 0; i < 256; i++) {  
00106 //            permVectorPointer->push_back(permutation[i]);  
00107 //        }  
00108 //}  
00109  
00110  
00111 //initialise 2x permutation of a hard coded 256 int vector  
00112 //std::vector<int> PERLIN_H::kansei() {  
00113  
00114  
00115 std::vector<int> getPermutationVector()  
00116 {  
00117     std::vector<int> p;  
00118  
00119     std::vector<int> permutation = { 151,  
00120         160,  
00121         137,  
00122         91,  
00123         90,  
00124         15,  
00125         131,  
00126         13,  
00127         201,  
00128         95,  
00129         96,  
00130         53,  
00131         194,  
00132         233,  
00133         7,  
00134         225,  
00135         140,  
00136         36,  
00137         103,  
00138         30,  
00139         69,  
00140         142,  
00141         8,  
00142         99,  
00143         37,  
00144         240,  
00145         21,  
00146         10,  
00147         23,  
00148         190,  
00149         6,  
00150         148,  
00151         247,  
00152         120,  
00153         234,  
00154         75,  
00155         0,  
00156         26,  
00157         197,  
00158         62,  
00159         94,  
00160         252,  
00161         219,  
00162         203,  
00163         117,  
00164         35,  
00165         11,  
00166         32,  
00167         57,  
00168         177,  
00169         33,
```

00170 88,
00171 237,
00172 149,
00173 56,
00174 87,
00175 174,
00176 20,
00177 125,
00178 136,
00179 171,
00180 168,
00181 68,
00182 175,
00183 74,
00184 165,
00185 71,
00186 134,
00187 139,
00188 48,
00189 27,
00190 166,
00191 77,
00192 146,
00193 158,
00194 231,
00195 83,
00196 111,
00197 229,
00198 122,
00199 60,
00200 211,
00201 133,
00202 230,
00203 220,
00204 105,
00205 92,
00206 41,
00207 55,
00208 46,
00209 245,
00210 40,
00211 244,
00212 102,
00213 143,
00214 54,
00215 65,
00216 25,
00217 63,
00218 161,
00219 1,
00220 216,
00221 80,
00222 73,
00223 209,
00224 76,
00225 132,
00226 187,
00227 208,
00228 89,
00229 18,
00230 169,
00231 200,
00232 196,
00233 135,
00234 130,
00235 116,
00236 188,
00237 159,
00238 86,
00239 164,
00240 100,
00241 109,
00242 198,
00243 173,
00244 186,
00245 3,
00246 64,
00247 52,
00248 217,
00249 226,
00250 250,
00251 124,
00252 123,
00253 5,
00254 202,
00255 38,
00256 147,

00257 118,
00258 126,
00259 255,
00260 82,
00261 85,
00262 212,
00263 207,
00264 206,
00265 59,
00266 227,
00267 47,
00268 16,
00269 58,
00270 17,
00271 182,
00272 189,
00273 28,
00274 42,
00275 223,
00276 183,
00277 170,
00278 213,
00279 119,
00280 248,
00281 152,
00282 2,
00283 44,
00284 154,
00285 163,
00286 70,
00287 221,
00288 153,
00289 101,
00290 155,
00291 167,
00292 43,
00293 172,
00294 9,
00295 129,
00296 22,
00297 39,
00298 253,
00299 19,
00300 98,
00301 108,
00302 110,
00303 79,
00304 113,
00305 224,
00306 232,
00307 178,
00308 185,
00309 112,
00310 104,
00311 218,
00312 246,
00313 97,
00314 228,
00315 251,
00316 34,
00317 242,
00318 193,
00319 238,
00320 210,
00321 144,
00322 12,
00323 191,
00324 179,
00325 162,
00326 241,
00327 81,
00328 51,
00329 145,
00330 235,
00331 249,
00332 14,
00333 239,
00334 107,
00335 49,
00336 192,
00337 214,
00338 31,
00339 181,
00340 199,
00341 106,
00342 157,
00343 184,

```

00344     84,
00345     204,
00346     176,
00347     115,
00348     121,
00349     50,
00350     45,
00351     127,
00352     4,
00353     150,
00354     254,
00355     138,
00356     236,
00357     205,
00358     93,
00359     222,
00360     114,
00361     67,
00362     29,
00363     24,
00364     72,
00365     243,
00366     141,
00367     128,
00368     195,
00369     78,
00370     66,
00371     215,
00372     61,
00373     156,
00374     180 };
00375
00376     for (int j = 0; j < 2; j++) {
00377         for (int i = 0; i < 256; i++) {
00378             p.push_back(permuation[i]);
00379         }
00380
00381     return p;
00382 }
```

7.83 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.cpp File Reference

#include "GameObject.h"
Include dependency graph for GameObject.cpp:

7.84 GameObject.cpp

[Go to the documentation of this file.](#)

```

00001 #include "GameObject.h"
00002
00003 GameObject::GameObject() : model(std::make_shared<Model>(Model())) { }
00004
00005 GameObject::GameObject(const std::string& model_path, glm::vec3 translation, GLfloat scale, Rotation
00006     rot) : model(std::make_shared<Model>())
00007 {
00008     model->load_model_in_ram(model_path);
00009     this->translation = translation;
00010     this->scale_factor = scale;
00011     this->rot = rot;
00012 }
00013 void GameObject::init(const std::string& model_path, glm::vec3 translation, GLfloat scale, Rotation
00014     rot)
00015 {
00016     model = std::make_shared<Model>(Model());
00017     model->load_model_in_ram(model_path);
00018     this->translation = translation;
00019     this->scale_factor = scale;
00020     this->rot = rot;
00021 }
00022 glm::mat4 GameObject::get_world_trafo()
```

```

00023 {
00024     glm::mat4 model_to_world = glm::mat4(1.0);
00025     model_to_world = glm::translate(model_to_world, translation);
00026     model_to_world = glm::scale(model_to_world, glm::vec3(scale_factor));
00027     model_to_world = glm::rotate(model_to_world, glm::radians(rot.degrees), rot.axis);
00028
00029     return model_to_world;
00030 }
00031
00032 glm::mat4 GameObject::get_normal_world_trafo()
00033 {
00034     glm::mat4 world_trafo = get_world_trafo();
00035     return glm::transpose(glm::inverse(world_trafo));
00036 }
00037
00038 void GameObject::render() { model->render(); }
00039
00040 std::shared_ptr<AABB> GameObject::get_aabb() { return model->get_aabb(); }
00041
00042 std::shared_ptr<Model> GameObject::get_model() { return model; }
00043
00044 void GameObject::translate(glm::vec3 translate) { this->translation = translate; }
00045
00046 void GameObject::rotate(Rotation rot) { this->rot = rot; }
00047
00048 void GameObject::scale(GLfloat scale_factor) { this->scale_factor = scale_factor; }
00049
00050 GameObject::~GameObject() { }

```

7.85 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.h File Reference

```
#include "Model.h"
#include "Rotation.h"
```

Include dependency graph for GameObject.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [GameObject](#)

7.86 GameObject.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include "Model.h"
00004 #include "Rotation.h"
00005
00006 class GameObject {
00007 public:
00008     GameObject();
00009
00010     GameObject(const std::string& model_path, glm::vec3 translation, GLfloat scale, Rotation rot);
00011
00012     void init(const std::string& model_path, glm::vec3 translation, GLfloat scale, Rotation rot);
00013
00014     glm::mat4 get_world_trafo();
00015     glm::mat4 get_normal_world_trafo();
00016
00017     std::shared_ptr<AABB> get_aabb();
00018     std::shared_ptr<Model> get_model();
00019
00020     void translate(glm::vec3 translate);
00021     void scale(GLfloat scale_factor);
00022     void rotate(Rotation rot);
00023
00024     void render();
00025
00026     ~GameObject();

```

```

00027
00028
00029     private:
00030     std::shared_ptr<Model> model;
00031
00032     GLfloat scale_factor;
00033     Rotation rot;
00034     glm::vec3 translation;
00035 };

```

7.87 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional← _light/CascadedShadowMap.cpp File Reference

```

#include "CascadedShadowMap.h"
#include "bindings.h"
#include <iostream>
Include dependency graph for CascadedShadowMap.cpp:

```

7.88 CascadedShadowMap.cpp

[Go to the documentation of this file.](#)

```

00001 #include "CascadedShadowMap.h"
00002 #include "bindings.h"
00003 #include <iostream>
00004
00005 CascadedShadowMap::CascadedShadowMap() :
00006
00007     FBO(0), shadow_maps(0), shadow_width(0), shadow_height(0), matrices_UBO(0),
00008     num_active_cascades(0), pcf_radius(1), intensity(1)
00009 {
00010 }
00011
00012 bool CascadedShadowMap::init(GLuint width, GLuint height, GLuint num_cascades)
00013 {
00014
00015     shadow_width = width;
00016     shadow_height = height;
00017
00018     num_active_cascades = num_cascades;
00019
00020     glGenFramebuffers(1, &FBO);
00021     glGenTextures(1, &shadow_maps);
00022     glBindTexture(GL_TEXTURE_2D_ARRAY, shadow_maps);
00023     glTexImage3D(GL_TEXTURE_2D_ARRAY, 0, GL_DEPTH_COMPONENT32F, shadow_width, shadow_height,
00024                 NUM_CASCADES, 0, GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00025
00026     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00027     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00028     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
00029     glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
00030
00031     constexpr float bordercolor[] = { 1.0f, 1.0f, 1.0f, 1.0f };
00032     glTexParameterfv(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_BORDER_COLOR, bordercolor);
00033
00034     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00035     glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, shadow_maps, 0);
00036     glDrawBuffer(GL_NONE);
00037     glReadBuffer(GL_NONE);
00038
00039     int status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00040     if (status != GL_FRAMEBUFFER_COMPLETE) {
00041         std::cout << "ERROR::FRAMEBUFFER:: Framebuffer is not complete!";
00042         throw 0;
00043     }
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045
00046 // setting up our buffer for the light matrices
00047 // for every cascade we will have 1 matrix in the geometry shader
00048     glGenBuffers(1, &matrices_UBO);
00049     glBindBuffer(GL_UNIFORM_BUFFER, matrices_UBO);

```

```

00050     glBindBuffer(GL_UNIFORM_BUFFER, sizeof(glm::mat4) * NUM_CASCADES, nullptr, GL_DYNAMIC_DRAW);
00051     glBindBufferBase(GL_UNIFORM_BUFFER, UNIFORM_LIGHT_MATRICES_BINDING, matrices_UBO);
00052     glBindBuffer(GL_UNIFORM_BUFFER, 0);
00053
00054     return true;
00055 }
00056
00057 void CascadedShadowMap::write_light_matrices(std::vector<glm::mat4x4>& lightMatrices)
00058 {
00059
00060     glBindBuffer(GL_UNIFORM_BUFFER, matrices_UBO);
00061     for (size_t i = 0; i < lightMatrices.size(); ++i) {
00062         glBindBufferSubData(GL_UNIFORM_BUFFER, i * sizeof(glm::mat4x4), sizeof(glm::mat4x4),
00063                             &lightMatrices[i]);
00064     }
00065     glBindBuffer(GL_UNIFORM_BUFFER, 0);
00066
00067 void CascadedShadowMap::write() { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }
00068
00069 void CascadedShadowMap::read(GLenum texture_unit)
00070 {
00071     glActiveTexture(GL_TEXTURE0 + texture_unit);
00072     glBindTexture(GL_TEXTURE_2D_ARRAY, shadow_maps);
00073 }
00074
00075 void CascadedShadowMap::set_pcf_radius(GLuint radius) { pcf_radius = radius; }
00076
00077 void CascadedShadowMap::set_intensity(GLfloat intensity) { this->intensity = intensity; }
00078
00079 CascadedShadowMap::~CascadedShadowMap()
00080 {
00081     if (FBO) {
00082         glDeleteFramebuffers(1, &FBO);
00083     }
00084
00085     if (shadow_maps) {
00086         glDeleteTextures(1, &shadow_maps);
00087     }
00088 }

```

7.89 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/CascadedShadowMap.h File Reference

```

#include <stdio.h>
#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <vector>
#include "GlobalValues.h"
#include "host_device_shared.h"

```

Include dependency graph for CascadedShadowMap.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [CascadedShadowMap](#)

7.90 CascadedShadowMap.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <stdio.h>
00003 #include <glad/glad.h>
00004 #include <glm/glm.hpp>
00005 #include <glm/gtc/type_ptr.hpp>

```

```

00006 #include <vector>
00007
00008 #include "GlobalValues.h"
00009 #include "host_device_shared.h"
00010
00011 class CascadedShadowMap {
00012 public:
00013     CascadedShadowMap();
00014
00015     bool init(GLuint width, GLuint height, GLuint num_cascades);
00016     void write();
00017     void read(GLenum texture_unit);
00018
00019     void write_light_matrices(std::vector<glm::mat4x4>& lightMatrices);
00020     void set_pcf_radius(GLuint radius);
00021     void set_intensity(GLfloat intensity);
00022
00023     GLfloat get_intensity() const { return intensity; }
00024     GLuint get_shadow_width() const { return shadow_width; }
00025     GLuint get_shadow_height() const { return shadow_height; }
00026     GLuint get_id() const { return shadow_maps; }
00027     GLuint get_num_active_cascades() const { return num_active_cascades; }
00028     GLuint get_pcf_radius() const { return pcf_radius; }
00029
00030     ~CascadedShadowMap();
00031
00032 protected:
00033     GLuint FBO, shadow_maps;
00034
00035     GLuint shadow_width, shadow_height;
00036
00037     GLuint matrices_UBO;
00038
00039     GLuint num_active_cascades;
00040
00041     GLuint pcf_radius;
00042
00043     GLfloat intensity;
00044 };

```

7.91 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalLight.cpp File Reference

#include "DirectionalLight.h"
Include dependency graph for DirectionalLight.cpp:

7.92 DirectionalLight.cpp

[Go to the documentation of this file.](#)

```

00001 #include "DirectionalLight.h"
00002
00003 DirectionalLight::DirectionalLight() :
00004
00005     Light(), shadow_map(std::make_shared<CascadedShadowMap>()),
00006
00007     direction(glm::vec3{ 0, 0, 0 }),
00008
00009     shadow_near_plane(0.f), shadow_far_plane(0.f),
00010
00011     cascade_light_matrices(NUM_CASCADES, glm::mat4(0.f))
00012
00013 {
00014
00015     light_proj = glm::ortho(-20.0f, 20.0f, -20.0f, 20.0f, 0.1f, 100.f);
00016 }
00017
00018
00019 DirectionalLight::DirectionalLight(GLuint shadow_width, GLuint shadow_height, GLfloat red, GLfloat
green, GLfloat blue, GLfloat radiance, GLfloat x_dir,
00020     GLfloat y_dir, GLfloat z_dir, GLfloat near_plane, GLfloat far_plane, int num_cascades) :
00021
00022     Light(red, green, blue, radiance),
00023     shadow_map(std::make_shared<CascadedShadowMap>()), direction(glm::vec3{ x_dir, y_dir, z_dir })

```

```
00024
00025     shadow_near_plane(near_plane), shadow_far_plane(far_plane),
00026
00027     cascade_light_matrices(NUM_CASCADES, glm::mat4(0.f))
00028
00029 {
00030
00031     light_proj = glm::ortho(-20.0f, 20.0f, -20.0f, 20.0f, 0.1f, 100.f);
00032
00033     shadow_map->init(shadow_width, shadow_height, num_cascades);
00034 }
00035
00036 glm::mat4 DirectionalLight::get_light_view_matrix() const { return glm::lookAt(direction,
00037     glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f)); }
00038
00039 glm::vec3 DirectionalLight::get_direction() const { return direction; }
00040
00041 glm::vec3 DirectionalLight::get_color() const { return color; }
00042
00043 float DirectionalLight::get_radiance() const { return radiance; }
00044
00045 std::vector<GLfloat> DirectionalLight::get_cascaded_slots() const
00046 {
00047     std::vector<GLfloat> result;
00048
00049     for (int i = 0; i < NUM_CASCADES + 1; i++) {
00050
00051         result.push_back(cascade_slots[i]);
00052     }
00053
00054     return result;
00055 }
00056
00057 std::vector<glm::mat4>& DirectionalLight::get_cascaded_light_matrices() { return
00058     cascade_light_matrices; }
00059
00060 void DirectionalLight::update_shadow_map(GLfloat shadow_width, GLfloat shadow_height, GLuint
00061     num_cascades)
00062 {
00063     shadow_map.reset(new CascadedShadowMap);
00064     shadow_map->init((GLuint)shadow_width, (GLuint)shadow_height, num_cascades);
00065 }
00066
00067 std::vector<glm::vec4> DirectionalLight::get_frustum_corners_world_space(const glm::mat4& proj, const
00068     glm::mat4& view)
00069 {
00070     const auto inv = glm::inverse(proj * view);
00071
00072     std::vector<glm::vec4> frustumCorners;
00073     for (unsigned int x = 0; x < 2; ++x) {
00074         for (unsigned int y = 0; y < 2; ++y) {
00075             for (unsigned int z = 0; z < 2; ++z) {
00076                 const glm::vec4 pt = inv * glm::vec4(2.0f * x - 1.0f, 2.0f * y - 1.0f, 2.0f * z - 1.0f, 1.0f);
00077                 frustumCorners.push_back(pt / pt.w);
00078             }
00079         }
00080     }
00081
00082     return frustumCorners;
00083 }
00084
00085 void DirectionalLight::calc_cascaded_slots()
00086 {
00087     GLuint number_of_elements = shadow_map->get_num_active_cascades();
00088
00089     for (int i = 0; i < NUM_CASCADES + 1; i++) {
00090         cascade_slots[i] = 100000.f;
00091     }
00092
00093     for (int i = 0; i < static_cast<int>(number_of_elements + 1); i++) {
00094         if (i == 0) {
00095             (cascade_slots)[i] = shadow_near_plane;
00096         } else {
00097             (cascade_slots)[i] = (shadow_far_plane) * ((GLfloat)i / (GLfloat)(number_of_elements));
00098         }
00099     }
00100
00101 //cascade_slots = { shadow_near_plane, shadow_far_plane / 50.f, shadow_far_plane / 25.f,
00102 shadow_far_plane };
00103
00104     return;
00105 }
```

```

00106 void DirectionalLight::calc_orthogonal_projections(
00107     glm::mat4 camera_view_matrix, GLfloat fov, GLuint window_width, GLuint window_height, GLuint
00108     current_num_cascades)
00109 {
00110     //calc the start and end point for our cascaded shadow maps
00111     calc_cascaded_slots();
00112
00113     for (int i = 0; i < static_cast<int>(current_num_cascades); i++) {
00114
00115         glm::mat4 curr_cascade_proj = glm::perspective(glm::radians(fov), (float)window_width /
00116         (float)window_height, cascade_slots[i], cascade_slots[i + 1]);
00117
00118         std::vector<glm::vec4> frustumCornerWorldSpace =
00119         get_frustum_corners_world_space(curr_cascade_proj, camera_view_matrix);
00120
00121         glm::vec3 center = glm::vec3(0, 0, 0);
00122         for (const auto& v : frustumCornerWorldSpace) {
00123             center += glm::vec3(v);
00124
00125         center /= frustumCornerWorldSpace.size();
00126
00127         glm::mat4 light_view_matrix = glm::lookAt(center - get_direction(), center, glm::vec3(0.0f, 1.0f,
0.0f));
00128
00129         // the # of frustum corners = 8
00130         GLfloat minX = std::numeric_limits<float>::max();
00131         GLfloat maxX = std::numeric_limits<float>::min();
00132         GLfloat minY = std::numeric_limits<float>::max();
00133         GLfloat maxY = std::numeric_limits<float>::min();
00134         GLfloat minZ = std::numeric_limits<float>::max();
00135         GLfloat maxZ = std::numeric_limits<float>::min();
00136
00137         for (unsigned int m = 0; m < frustumCornerWorldSpace.size(); m++) {
00138             //transform each corner from view to world space
00139             glm::vec4 v_light_view = light_view_matrix * frustumCornerWorldSpace[m];
00140             minX = std::min(minX, v_light_view.x);
00141             maxX = std::max(maxX, v_light_view.x);
00142             // we always have negative y values
00143             minY = std::min(minY, v_light_view.y);
00144             maxY = std::max(maxY, v_light_view.y);
00145
00146             minZ = std::min(minZ, v_light_view.z);
00147             maxZ = std::max(maxZ, v_light_view.z);
00148         }
00149
00150         // Tune this parameter according to the scene
00151         // for having objects casting shadows that are actually not in the frustum :)
00152         constexpr float zMult = 10.0f;
00153         if (minZ < 0) {
00154             minZ *= zMult;
00155         } else {
00156             minZ /= zMult;
00157         }
00158         if (maxZ < 0) {
00159             maxZ /= zMult;
00160         } else {
00161             maxZ *= zMult;
00162         }
00163
00164         glm::mat4 light_projection = glm::ortho(minX, maxX, minY, maxY, minZ, maxZ);
00165
00166         cascade_light_matrices[i] = light_projection * light_view_matrix;
00167     }
00168 }
00169
00170 glm::mat4 DirectionalLight::calculate_light_transform() { return light_proj * get_light_view_matrix(); }
00171
00172 void DirectionalLight::set_direction(glm::vec3 direction) { this->direction = direction; }
00173
00174 void DirectionalLight::set_radiance(float radiance) { this->radiance = radiance; }
00175
00176 void DirectionalLight::set_color(glm::vec3 color) { this->color = color; }
00177
00178 DirectionalLight::~DirectionalLight() { }

```

7.93 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalLight.h File Reference

```
#include <memory>
#include <limits>
#include <vector>
#include <array>
#include "Light.h"
#include "host_device_shared.h"
```

Include dependency graph for DirectionalLight.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [DirectionalLight](#)

7.94 DirectionalLight.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <memory>
00003 #include <limits>
00004 #include <vector>
00005 #include <array>
00006
00007 #include "Light.h"
00008 #include "host_device_shared.h"
00009
00010 class DirectionalLight : public Light {
00011     public:
00012     DirectionalLight();
00013
00014     DirectionalLight(GLuint shadow_width, GLuint shadow_height, GLfloat red, GLfloat green, GLfloat
blue, GLfloat radiance, GLfloat x_dir, GLfloat y_dir,
00015                 GLfloat z_dir, GLfloat near_plane, GLfloat far_plane, int num_cascades);
00016
00017     glm::mat4 calculate_light_transform();
00018
00019     std::shared_ptr<CascadedShadowMap> get_shadow_map() const { return shadow_map; }
00020
00021     glm::vec3 get_direction() const;
00022     glm::vec3 get_color() const;
00023     float get_radiance() const;
00024     glm::mat4 get_light_view_matrix() const;
00025     std::vector<GLfloat> get_cascaded_slots() const;
00026     std::vector<glm::mat4>& get_cascaded_light_matrices();
00027
00028     void set_direction(glm::vec3 direction);
00029     void set_radiance(float radiance);
00030     void set_color(glm::vec3 color);
00031
00032     void update_shadow_map(GLfloat shadow_width, GLfloat shadow_height, GLuint num_cascades);
00033
00034     void calc_orthogonal_projections(glm::mat4 camera_view_matrix, GLfloat fov, GLuint window_width,
GLuint window_height, GLuint current_num_cascades);
00035
00036
00037 ~DirectionalLight();
00038
00039 private:
00040     std::vector<glm::vec4> get_frustum_corners_world_space(const glm::mat4& proj, const glm::mat4&
view);
00041     void calc_cascaded_slots();
00042
00043     std::shared_ptr<CascadedShadowMap> shadow_map;
00044
00045     glm::vec3 direction;
00046     GLfloat shadow_near_plane, shadow_far_plane;
00047
00048     std::array<GLfloat, NUM_CASCADES + 1> cascade_slots;
00049     std::vector<glm::mat4> cascade_light_matrices;
00050 };
```

7.95 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalShadowMapPass.cpp File Reference

```
#include "DirectionalShadowMapPass.h"
#include <memory>
Include dependency graph for DirectionalShadowMapPass.cpp:
```

7.96 DirectionalShadowMapPass.cpp

[Go to the documentation of this file.](#)

```
00001 #include "DirectionalShadowMapPass.h"
00002 #include <memory>
00003
00004 DirectionalShadowMapPass::DirectionalShadowMapPass() { create_shader_program(); }
00005
00006 void DirectionalShadowMapPass::execute(
00007     glm::mat4 projection, std::shared_ptr<Camera> main_camera, GLuint window_width, GLuint
00008     window_height, std::shared_ptr<Scene> scene)
00009 {
00010     std::shared_ptr<DirectionalLight> sun = scene->get_sun();
00011     //retrieve shadow map before our geometry pass
00012     sun->calc_orthogonal_projections(main_camera->get_viewmatrix(), main_camera->get_fov(),
00013     window_width, window_height, NUM_CASCADES);
00014     shader_program->use_shader_program();
00015
00016     sun->get_shadow_map()->write();
00017
00018     glViewport(0, 0, sun->get_shadow_map()->get_shadow_width(),
00019     sun->get_shadow_map()->get_shadow_height());
00020     glClear(GL_DEPTH_BUFFER_BIT);
00021
00022     glEnable(GL_CULL_FACE);
00023     glCullFace(GL_BACK);
00024     glFrontFace(GL_CCW);
00025
00026     //glCullFace(GL_FRONT); // avoid peter panning
00027     sun->get_shadow_map()->write_light_matrices(sun->get_cascaded_light_matrices());
00028
00029     shader_program->setUniformBlockBinding(UNIFORM_LIGHT_MATRICES_BINDING, "LightSpaceMatrices");
00030
00031     std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00032
00033     for (std::shared_ptr<GameObject> object : game_objects) {
00034         /* if (object_is_visible(object)) {*/
00035             set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00036
00037             object->render();
00038             //}
00039         }
00040
00041     }
00042
00043     //glCullFace(GL_BACK);
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045     shader_program->validate_program();
00046 }
00047
00048 void DirectionalShadowMapPass::create_shader_program()
00049 {
00050     shader_program = std::make_shared<ShaderProgram>(ShaderProgram{});
00051     shader_program->create_from_files(
00052         "rasterizer/shadows/directional_shadow_map.vert",
00053         "rasterizer/shadows/directional_shadow_map.geom", "rasterizer/shadows/directional_shadow_map.frag");
00054
00055     void DirectionalShadowMapPass::set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model)
00056 {
00057     // DO NOT set neither normal model nor material_id hence we didn't need it
00058     shader_program->setUniformMatrix4fv(model, "model");
00059 }
00060
00061 DirectionalShadowMapPass::~DirectionalShadowMapPass() { }
```

7.97

C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalShadowMapPass.h File

Reference

305

7.97 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/directional_light/DirectionalShadowMapPass.h File Reference

```
#include "RenderPassSceneDependend.h"
#include "DirectionalLight.h"
#include "ShaderProgram.h"
#include "ViewFrustumCulling.h"
#include "Scene.h"
```

Include dependency graph for DirectionalShadowMapPass.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [DirectionalShadowMapPass](#)

7.98 DirectionalShadowMapPass.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include "RenderPassSceneDependend.h"
00004 #include "DirectionalLight.h"
00005 #include "ShaderProgram.h"
00006 #include "ViewFrustumCulling.h"
00007 #include "Scene.h"
00008
00009 class DirectionalShadowMapPass : public RenderPassSceneDependend {
00010     public:
00011     DirectionalShadowMapPass();
00012
00013     void execute(glm::mat4 projection, std::shared_ptr<Camera> main_camera, GLuint window_width, GLuint
window_height, std::shared_ptr<Scene> scene);
00014
00015     void create_shader_program();
00016
00017     void set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model);
00018
00019     ~DirectionalShadowMapPass();
00020
00021     private:
00022     std::shared_ptr<ShaderProgram> shader_program;
00023 };
```

7.99 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.cpp File Reference

```
#include "Light.h"
```

Include dependency graph for Light.cpp:

7.100 Light.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Light.h"
00002
00003 Light::Light() :
00004     color(glm::vec3(1.0f)), radiance(1.0f)
```

```

00006
00007 {
00008 }
00009
0010 Light::Light(GLfloat red, GLfloat green, GLfloat blue, GLfloat radiance) :
0011     color(glm::vec3(red, green, blue)), radiance(radiance)
0012
0013 {
0014 }
0015 }
0016
0017 Light::~Light() { }

```

7.101 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.h File Reference

```

#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include "CascadedShadowMap.h"

```

Include dependency graph for Light.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Light](#)

7.102 Light.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <glad/glad.h>
00003 #include <glm/glm.hpp>
00004 #include <glm/gtc/matrix_transform.hpp>
00005
00006 #include "CascadedShadowMap.h"
00007
00008 class Light {
00009     public:
0010     Light();
0011
0012     Light(GLfloat red, GLfloat green, GLfloat blue, GLfloat radiance);
0013
0014     glm::vec3 get_color() const { return color; };
0015     float get_radiance() const { return radiance; };
0016
0017     ~Light();
0018
0019     protected:
0020     glm::vec3 color;
0021     float radiance;
0022
0023     glm::mat4 light_proj;
0024 };

```

7.103 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/OmniDirShadowMap.cpp File Reference

```

#include "OmniDirShadowMap.h"

```

Include dependency graph for OmniDirShadowMap.cpp:

7.104 OmniDirShadowMap.cpp

[Go to the documentation of this file.](#)

```

00001 #include "OmniDirShadowMap.h"
00002
00003
00004 OmniDirShadowMap::OmniDirShadowMap() : ShadowMap() { }
00005
00006 bool OmniDirShadowMap::init(GLuint width, GLuint height)
00007 {
00008     shadow_width = width;
00009     shadow_height = height;
00010
00011     glGenFramebuffers(1, &FBO);
00012
00013     glGenTextures(1, &shadow_map);
00014     glBindTexture(GL_TEXTURE_CUBE_MAP, shadow_map);
00015
00016     for (size_t i = 0; i < 6; i++) {
00017         // keep in mind that all following f.e. negative_x, positive_y,...etc. are reachable
00018         // by simply increment positive_x stepwise
00019         glTexImage2D(GLenum(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i), 0, GL_DEPTH_COMPONENT, shadow_width,
00020                     shadow_height, 0, GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00021     }
00022
00023     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00024     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00025
00026     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00027     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00028     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00029
00030     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00031     glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, shadow_map, 0);
00032
00033     glDrawBuffer(GL_NONE);
00034     glReadBuffer(GL_NONE);
00035
00036     GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00037
00038     if (status != GL_FRAMEBUFFER_COMPLETE) {
00039
00040         printf("Framebuffer error: %i\n", status);
00041         return false;
00042     }
00043
00044     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00045
00046     return true;
00047 }
00048
00049 void OmniDirShadowMap::write() { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }
00050
00051 void OmniDirShadowMap::read(GLenum texture_unit)
00052 {
00053     glActiveTexture(GL_TEXTURE0 + texture_unit);
00054     glBindTexture(GL_TEXTURE_CUBE_MAP, shadow_map);
00055 }
00056
00057 OmniDirShadowMap::~OmniDirShadowMap() { }
```

7.105 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/OmniDirShadowMap.h File Reference

```
#include "ShadowMap.h"
```

Include dependency graph for OmniDirShadowMap.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [OmniDirShadowMap](#)

7.106 OmniDirShadowMap.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "ShadowMap.h"
00003
00004 class OmniDirShadowMap : public ShadowMap {
00005     public:
00006     OmniDirShadowMap();
00007
00008     bool init(GLuint width, GLuint height);
00009
00010     void write();
00011
00012     void read(GLenum texture_unit);
00013
00014     ~OmniDirShadowMap();
00015 };
```

7.107 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/OmniDirShadowShaderProgram.cpp File Reference

```
#include "OmniDirShadowShaderProgram.h"
```

Include dependency graph for OmniDirShadowShaderProgram.cpp:

7.108 OmniDirShadowShaderProgram.cpp

[Go to the documentation of this file.](#)

```
00001 #include "OmniDirShadowShaderProgram.h"
00002
00003 OmniDirShadowShaderProgram::OmniDirShadowShaderProgram() { }
00004
00005 void OmniDirShadowShaderProgram::reload() { create_from_files(this->vertex_location,
    this->geometry_location, this->fragment_location); }
00006
00007 OmniDirShadowShaderProgram::~OmniDirShadowShaderProgram() { }
```

7.109 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/OmniDirShadowShaderProgram.h File Reference

```
#include "ShaderProgram.h"
```

Include dependency graph for OmniDirShadowShaderProgram.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [OmniDirShadowShaderProgram](#)

7.110 OmniDirShadowShaderProgram.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "ShaderProgram.h"
00003
00004 class OmniDirShadowShaderProgram : public ShaderProgram {
00005
00006 public:
00007     OmniDirShadowShaderProgram();
00008
00009     void reload();
00010
00011     ~OmniDirShadowShaderProgram();
00012
00013 private:
00014 };
```

7.111 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/OmniShadowMapPass.cpp File Reference

```
#include "OmniShadowMapPass.h"
#include <sstream>
Include dependency graph for OmniShadowMapPass.cpp:
```

7.112 OmniShadowMapPass.cpp

[Go to the documentation of this file.](#)

```
00001 #include "OmniShadowMapPass.h"
00002 #include <sstream>
00003
00004 OmniShadowMapPass::OmniShadowMapPass() { create_shader_program(); }
00005
00006 void OmniShadowMapPass::execute(std::shared_ptr<PointLight> p_light, std::shared_ptr<Scene> scene)
00007 {
00008     shader_program->use_shader_program();
00009
00010     glViewport(0, 0, p_light->get_omni_shadow_map()->get_shadow_width(),
00011     p_light->get_omni_shadow_map()->get_shadow_height());
00012
00013     p_light->get_omni_shadow_map()->write();
00014     glClear(GL_DEPTH_BUFFER_BIT);
00015
00016     shader_program->setUniformVec3(p_light->get_position(), "light_pos");
00017     shader_program->setUniformFloat(p_light->get_far_plane(), "far_plane");
00018
00019     std::vector<glm::mat4> light_matrices = p_light->calculate_light_transform();
00020
00021     std::stringstream ss;
00022     for (uint32_t i = 0; i < 6; i++) {
00023
00024         ss << "light_matrices[" << i << "]";
00025         shader_program->setUniformMatrix4fv(light_matrices[i], ss.str());
00026     }
00027
00028     shader_program->validate_program();
00029
00030     std::vector<std::shared_ptr<GameObject>> game_objects = scene->get_game_objects();
00031
00032     for (std::shared_ptr<GameObject> object : game_objects) {
00033
00034         /* if (object_is_visible(object)) {*/
00035         set_game_object_uniforms(object->get_world_trafo(), object->get_normal_world_trafo());
00036
00037         object->render();
00038         //}
00039     }
00040
00041     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00042 }
```

```

00043
00044 void OmniShadowMapPass::set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model) {
    shader_program->setUniformMatrix4fv(model, "model");
}
00045
00046 void OmniShadowMapPass::create_shader_program()
00047 {
00048     shader_program = std::make_shared<OmniDirShadowShaderProgram>(OmniDirShadowShaderProgram{});
00049     shader_program->create_from_files(
00050         "rasterizer/shadows/omni_shadow_map.vert", "rasterizer/shadows/omni_shadow_map.geom",
00051         "rasterizer/shadows/omni_shadow_map.frag");
}
00052
00053 OmniShadowMapPass::~OmniShadowMapPass() { }

```

7.113 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/OmniShadowMapPass.h File Reference

```

#include "RenderPassSceneDependend.h"
#include "PointLight.h"
#include "OmniDirShadowShaderProgram.h"
#include "ViewFrustumCulling.h"
#include "Clouds.h"
#include "Scene.h"

```

Include dependency graph for OmniShadowMapPass.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [OmniShadowMapPass](#)

7.114 OmniShadowMapPass.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include "RenderPassSceneDependend.h"
00004 #include "PointLight.h"
00005 #include "OmniDirShadowShaderProgram.h"
00006 #include "ViewFrustumCulling.h"
00007 #include "Clouds.h"
00008 #include "Scene.h"
00009
00010 class OmniShadowMapPass : public RenderPassSceneDependend {
00011     public:
00012     OmniShadowMapPass();
00013
00014     void execute(std::shared_ptr<PointLight> p_light, std::shared_ptr<Scene> scene);
00015
00016     void set_game_object_uniforms(glm::mat4 model, glm::mat4 normal_model);
00017
00018     void create_shader_program();
00019
00020     ~OmniShadowMapPass();
00021
00022     private:
00023     std::shared_ptr<OmniDirShadowShaderProgram> shader_program;
00024 };

```

7.115 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_← light/PointLight.cpp File Reference

```

#include "PointLight.h"

```

Include dependency graph for PointLight.cpp:

7.116 PointLight.cpp

[Go to the documentation of this file.](#)

```

00001 #include "PointLight.h"
00002
00003 PointLight::PointLight() :
00004     position(glm::vec3(0.0f)), constant(1.0f), linear(0.0f), exponent(0.0f), far_plane(0.f)
00005     position(glm::vec3(0.0f)), constant(1.0f), linear(0.0f), exponent(0.0f), far_plane(0.f)
00006 {
00007 }
00008 }
00009
00010 PointLight::PointLight(GLuint shadow_width, GLuint shadow_height, GLfloat near, GLfloat far, GLfloat
00011     red, GLfloat green, GLfloat blue, GLfloat radiance,
00012     GLfloat x_pos, GLfloat y_pos, GLfloat z_pos, GLfloat con, GLfloat lin, GLfloat exp) :
00013     Light(red, green, blue, radiance),
00014     omni_dir_shadow_map(std::make_shared<OmniDirShadowMap>()),
00015     position(glm::vec3(x_pos, y_pos, z_pos)), constant(con), linear(lin), exponent(exp),
00016     far_plane(far)
00017 {
00018
00019     float aspect = (float)shadow_width / (float)shadow_height;
00020     light_proj = glm::perspective(glm::radians(90.0f), aspect, near, far);
00021     omni_dir_shadow_map->init(shadow_width, shadow_height);
00022 }
00023
00024 std::vector<glm::mat4> PointLight::calculate_light_transform()
00025 {
00026     std::vector<glm::mat4> light_matrices;
00027     //make sure all light matrices align with the order we were defining in OmniShadowMap
00028     //GL_TEXTURE_CUBE_MAP_POSITIVE_X+i; therefoe start off with glm::vec3(1.0, 0.0, 0.0)
00029     //+x,-x
00030     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(1.0, 0.0, 0.0),
00031         glm::vec3(0.0, -1.0, 0.0)));
00032     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(-1.0, 0.0, 0.0),
00033         glm::vec3(0.0, -1.0, 0.0)));
00034     //+y,-y
00035     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, 1.0, 0.0),
00036         glm::vec3(0.0, 0.0, 1.0)));
00037     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, -1.0, 0.0),
00038         glm::vec3(0.0, 0.0, -1.0)));
00039     //+z,-z
00040     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, 0.0, 1.0),
00041         glm::vec3(0.0, -1.0, 0.0)));
00042     light_matrices.push_back(light_proj * glm::lookAt(position, position + glm::vec3(0.0, 0.0, -1.0),
00043         glm::vec3(0.0, 1.0, 0.0)));
00044     return light_matrices;
00045 }
00046 GLfloat PointLight::get_far_plane() { return far_plane; }
00047
00048 glm::vec3 PointLight::get_position() { return position; }
00049
00050 PointLight::~PointLight() { }

```

7.117 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/point_light/PointLight.h File Reference

```

#include <vector>
#include <memory>
#include "Light.h"
#include "OmniDirShadowMap.h"
#include "ShaderProgram.h"

```

Include dependency graph for PointLight.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [PointLight](#)

7.118 PointLight.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <vector>
00003 #include <memory>
00004
00005 #include "Light.h"
00006 #include "OmniDirShadowMap.h"
00007 #include "ShaderProgram.h"
00008
00009 class PointLight : public Light {
00010
00011     public:
00012     PointLight();
00013
00014     PointLight(GLuint shadow_width, GLuint shadow_height, GLfloat near, GLfloat far, GLfloat red,
00015                 GLfloat green, GLfloat blue, GLfloat radiance, GLfloat x_pos,
00016                 GLfloat y_pos, GLfloat z_pos, GLfloat con, GLfloat lin, GLfloat exp);
00017
00018     std::vector<glm::mat4> calculate_light_transform();
00019
00020     void set_position(glm::vec3 position);
00021
00022     std::shared_ptr<OmniDirShadowMap> get_omni_shadow_map() { return omni_dir_shadow_map; };
00023     GLfloat get_far_plane();
00024     glm::vec3 get_position();
00025     GLfloat get_constant_factor() { return constant; };
00026     GLfloat get_linear_factor() { return linear; };
00027     GLfloat get_exponent_factor() { return exponent; };
00028
00029     ~PointLight();
00030
00031     protected:
00032     std::shared_ptr<OmniDirShadowMap> omni_dir_shadow_map;
00033
00034     glm::vec3 position;
00035
00036     GLfloat constant, linear, exponent;
00037     GLfloat far_plane;
00038
00039 };
00040 };
```

7.119 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.cpp File Reference

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "Mesh.h"
#include "Texture.h"
Include dependency graph for Mesh.cpp:
```

7.120 Mesh.cpp

[Go to the documentation of this file.](#)

```
00001 #include <glm/glm.hpp>
00002 #include <glm/gtc/matrix_transform.hpp>
00003 #include <glm/gtc/type_ptr.hpp>
00004
00005 #include "Mesh.h"
00006 #include "Texture.h"
00007
00008 Mesh::Mesh() : m_vao(-1), m_ibo(-1), m_drawCount(0), vertices(std::vector<Vertex>()),
00009                 indices(std::vector<uint32_t>()) {}
00010 Mesh::Mesh(std::vector<Vertex>& vertices, std::vector<unsigned int>& indices) :
```

```

00011     vertices(vertices), indices(indices)
00012 {
00013
00014
00015     uint32_t numVertices = static_cast<uint32_t>(vertices.size());
00016     uint32_t num_indices = static_cast<uint32_t>(indices.size());
00017
00018     m_drawCount = num_indices;
00019     glGenVertexArrays(1, &m_vao);
00020     glBindVertexArray(m_vao);
00021
00022     glGenBuffers(1, &m_ibo);
00023     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_ibo);
00024     // Dynamic Draw = lower Performance.
00025     glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(this->indices[0]) * num_indices, &(this->indices[0]), GL_DYNAMIC_DRAW);
00026
00027     glGenBuffers(NUM_BUFFERS, m_vab);
00028     glBindBuffer(GL_ARRAY_BUFFER, m_vab[POSITION_VB]);
00029     glBufferData(GL_ARRAY_BUFFER, numVertices * sizeof(this->vertices[0]), &(this->vertices[0]), GL_DYNAMIC_DRAW);
00030
00031     //enable Vertex Attribs for Pos, Norm, Textcood
00032     // Vertex Position
00033     //Stride bytes: just the size of Vertex, offset = use offsetof funktion
00034     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, position));
00035     glEnableVertexAttribArray(0);
00036     //Vertex Normal
00037     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, normal));
00038     glEnableVertexAttribArray(1);
00039     //Vertex Normal
00040     glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, color));
00041     glEnableVertexAttribArray(2);
00042     // Vertex Texture Cood
00043     glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)(offsetof(Vertex, texture_coords)));
00044     glEnableVertexAttribArray(3);
00045
00046     //unbind everything after setting the attrs
00047     glBindBuffer(GL_ARRAY_BUFFER, 0);
00048     glBindVertexArray(0);
00049     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00050 }
00051
00052
00053 void Mesh::render()
00054 {
00055
00056     glBindVertexArray(m_vao);
00057     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_ibo);
00058     //Draw Triangles
00059     glDrawElements(GL_TRIANGLES, m_drawCount, GL_UNSIGNED_INT, 0);
00060
00061     //unbind all again
00062     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00063     glBindVertexArray(0);
00064 }
00065
00066 Mesh::~Mesh()
00067 {
00068
00069     glDeleteVertexArrays(1, &m_vao);
00070     glDeleteBuffers(1, &m_ibo);
00071     glDeleteBuffers(NUM_BUFFERS, m_vab);
00072 }

```

7.121 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.h File Reference

```
#include <glm/glm.hpp>
#include <vector>
#include <glad/glad.h>
#include "Texture.h"
#include "Vertex.h"
#include "GlobalValues.h"
```

Include dependency graph for Mesh.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class Mesh

7.122 Mesh.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <glm/glm.hpp>
00003 #include <vector>
00004 #include <glad/glad.h>
00005
00006 #include "Texture.h"
00007 #include "Vertex.h"
00008 #include "GlobalValues.h"
00009
00010 // this a simple Mesh without mesh generation
00011 class Mesh {
00012     public:
00013     Mesh(std::vector<Vertex>& vertices, std::vector<unsigned int>& indices);
00014
00015     Mesh();
00016
00017     void render();
00018
00019     std::vector<Vertex> getVertices() const { return this->vertices; }
00020     std::vector<unsigned int> getIndices() const { return this->indices; }
00021
00022     ~Mesh();
00023
00024     private:
00025     // render data
00026     // unsigned int VAO, VBO, EBO;
00027     enum {
00028         POSITION = 0,
00029         NORMAL = 1,
00030         COLOR = 2,
00031         TEXTURECOORD = 3
00032     };
00033
00034     enum { POSITION_VB, NUM_BUFFERS };
00035
00036     // Vertex Array Object
00037     GLuint m_vao, m_ibo;
00038     // Vertex array buffer
00039     GLuint m_vab[NUM_BUFFERS];
00040
00041     uint32_t m_drawCount;
00042     std::vector<Vertex> vertices;
00043     std::vector<uint32_t> indices;
00044
00045 };

```

7.123 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.cpp File Reference

```
#include "Model.h"
Include dependency graph for Model.cpp:
```

7.124 Model.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Model.h"
00002
00003 Model::Model() : aabb(std::make_shared<AABB>()) { }
00004

```

```

00005 std::shared_ptr<AABB> Model::get_aabb() { return aabb; }
00006
00007 std::vector<ObjMaterial> Model::get_materials() const { return materials; }
00008
00009 int Model::get_texture_count() const { return static_cast<uint32_t>(texture_list.size()); }
00010
00011 void Model::load_model_in_ram(const std::string& model_path)
00012 {
00013
00014     loader = ObjLoader();
00015     loader.load(model_path, vertices, indices, textures, materials, materialIndex);
00016 }
00017
00018 // all OpenGL calls need to be on the same thread!
00019 // hence we have to decouple the loading task from all OpenGL agnostic code
00020 void Model::create_render_context()
00021 {
00022     texture_list.resize(textures.size());
00023
00024     for (uint32_t i = 0; i < static_cast<uint32_t>(textures.size()); i++) {
00025
00026         texture_list[i] = std::make_shared<Texture>(textures[i].c_str(), std::make_shared<RepeatMode>());
00027
00028         if (!texture_list[i]->load_SRGB_texture_without_alpha_channel()) {
00029             printf("Failed to load texture at: %s\n", textures[i].c_str());
00030             texture_list[i].reset();
00031         }
00032     }
00033
00034     mesh = std::make_shared<Mesh>(vertices, indices);
00035
00036 // https://www.khronos.org/opengl/wiki/Shader_Storage_Buffer_Object
00037 glGenBuffers(1, &ssbo);
00038 glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);
00039 glBufferData(GL_SHADER_STORAGE_BUFFER,
00040     materialIndex.size() * sizeof(glm::vec4),
00041     materialIndex.data(),
00042     GL_STREAM_READ); //sizeof(data) only works for statically sized C/C++ arrays.
00043 }
00044
00045 void Model::bind_ressources()
00046 {
00047     glBindBufferBase(GL_SHADER_STORAGE_BUFFER, STORAGE_BUFFER_MATERIAL_ID_BINDING, ssbo);
00048     for (int i = 0; i < static_cast<int>(texture_list.size()); i++) {
00049         texture_list[i]->use_texture(i + MODEL_TEXTURES_SLOT);
00050     }
00051 }
00052
00053 void Model::unbind_resources()
00054 {
00055     glBindBuffer(GL_SHADER_STORAGE_BUFFER, 0);
00056     for (int i = 0; i < static_cast<int>(texture_list.size()); i++) {
00057         texture_list[i]->unbind_texture(i + MODEL_TEXTURES_SLOT);
00058     }
00059 }
00060
00061 void Model::render() { mesh->render(); }
00062
00063 Model::~Model()
00064 {
00065     // unbind material index buffer
00066 }

```

7.125 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.h File Reference

```

#include <iostream>
#include <vector>
#include <unordered_map>
#include <memory>
#include "Mesh.h"
#include "Vertex.h"
#include "AABB.h"
#include "GlobalValues.h"
#include "ObjMaterial.h"
#include "ObjLoader.h"

```

```
#include "bindings.h"
```

Include dependency graph for Model.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Model](#)

7.126 Model.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <iostream>
00003 #include <vector>
00004 #include <unordered_map>
00005 #include <memory>
00006
00007 #include "Mesh.h"
00008 #include "Vertex.h"
00009 #include "AABB.h"
00010 #include "GlobalValues.h"
00011 #include "ObjMaterial.h"
00012 #include "ObjLoader.h"
00013
00014 #include "bindings.h"
00015
00016 class Model {
00017     public:
00018     Model();
00019
00020     void load_model_in_ram(const std::string& model_path);
00021
00022     void create_render_context();
00023
00024     void bind_ressources();
00025
00026     void unbind_resources();
00027
00028     std::shared_ptr<AABB> get_aabb();
00029     std::vector<ObjMaterial> get_materials() const;
00030     int get_texture_count() const;
00031
00032     void render();
00033
00034     ~Model();
00035
00036     private:
00037     // buffer for material id's
00038     GLuint ssbo;
00039
00040     ObjLoader loader;
00041
00042     std::shared_ptr<AABB> aabb;
00043
00044     std::shared_ptr<Mesh> mesh;
00045     std::vector<Vertex> vertices;
00046     std::vector<unsigned int> indices;
00047     std::vector<std::shared_ptr<Texture>> texture_list;
00048     std::vector<ObjMaterial> materials;
00049     std::vector<glm::vec4> materialIndex;
00050     std::vector<std::string> textures;
00051 };
```

7.127 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.cpp File Reference

```
#include "ObjLoader.h"
```

```
#include <tiny_obj_loader.h>
```

Include dependency graph for ObjLoader.cpp:

Macros

- `#define TINYOBJLOADER_IMPLEMENTATION`

7.127.1 Macro Definition Documentation

7.127.1.1 TINYOBJLOADER_IMPLEMENTATION

```
#define TINYOBJLOADER_IMPLEMENTATION
```

Definition at line 3 of file ObjLoader.cpp.

7.128 ObjLoader.cpp

[Go to the documentation of this file.](#)

```
00001 #include "ObjLoader.h"
00002
00003 #define TINYOBJLOADER_IMPLEMENTATION
00004 #include <tiny_obj_loader.h>
00005
00006 ObjLoader::ObjLoader() { }
00007
00008 void ObjLoader::load(std::string modelFile, std::vector<Vertex>& vertices, std::vector<unsigned int>&
00009   indices, std::vector<std::string>& texture_list,
00010   std::vector<ObjMaterial>& materials, std::vector<glm::vec4>& materialIndex)
00011 {
00012   tinyobj::ObjReaderConfig reader_config;
00013   tinyobj::ObjReader reader;
00014
00015   if (!reader.ParseFromFile(modelFile, reader_config)) {
00016     if (!reader.Error().empty()) {
00017       std::cerr << "TinyObjReader: " << reader.Error();
00018     }
00019     exit(EXIT_FAILURE);
00020   }
00021
00022   if (!reader.Warning().empty()) {
00023     std::cout << "TinyObjReader: " << reader.Warning();
00024   }
00025
00026   auto& tol_materials = reader.GetMaterials();
00027   //texture_list.reserve(tol_materials.size());
00028
00029   if (static_cast<GLuint>(tol_materials.size() > MAX_MATERIALS)) std::runtime_error("ObjLoader: We try
00030   to load more materials than MAX_MATERIALS is defined!");
00031
00032   // texture at position 0 is plain texture to handle non existing materials
00033   int texture_id = 1;
00034
00035   std::stringstream texture_base_dir;
00036   texture_base_dir << CMAKELISTS_DIR << "/Resources/Textures/plain.png";
00037   texture_list.push_back(texture_base_dir.str());
00038
00039   // we now iterate over all materials to get diffuse textures
00040   for (size_t i = 0; i < tol_materials.size(); i++) {
00041
00042     const tinyobj::material_t* mp = &tol_materials[i];
00043     ObjMaterial material;
00044     material.ambient = glm::vec3(mp->ambient[0], mp->ambient[1], mp->ambient[2]);
00045     material.diffuse = glm::vec3(mp->diffuse[0], mp->diffuse[1], mp->diffuse[2]);
00046     material.specular = glm::vec3(mp->specular[0], mp->specular[1], mp->specular[2]);
00047     material.emission = glm::vec3(mp->emission[0], mp->emission[1], mp->emission[2]);
00048     material.transmittance = glm::vec3(mp->transmittance[0], mp->transmittance[1],
00049                                         mp->transmittance[2]);
00050     material.dissolve = mp->dissolve;
00051     material.ior = mp->ior;
00052     material.shininess = mp->shininess;
```

```

00051     material.illum = mp->illum;
00052
00053     if (mp->diffuse_texname.length() > 0) {
00054
00055         std::string relative_texture_filename = mp->diffuse_texname;
00056         std::string texture_filename = get_base_dir(modelFile) + "/" + relative_texture_filename;
00057
00058         texture_list.push_back(texture_filename);
00059
00060         material.textureID = texture_id;
00061         texture_id++;
00062
00063     } else {
00064
00065         // this means no texture was assigned; we catch it here and assign our plain texture
00066         // at position 0
00067         material.textureID = 0;
00068     }
00069
00070     materials.push_back(material);
00071 }
00072
00073 // for the case no .mtl file is given place some random standard material ...
00074 if (tol_materials.empty()) {
00075     materials.emplace_back(ObjMaterial());
00076 }
00077
00078 auto& attrib = reader.GetAttrib();
00079 auto& shapes = reader.GetShapes();
00080
00081 std::unordered_map<Vertex, uint32_t> vertices_map{};
00082
00083 // Loop over shapes
00084 for (size_t s = 0; s < shapes.size(); s++) {
00085     // prepare for enlargement
00086     vertices.reserve(shapes[s].mesh.indices.size() + vertices.size());
00087     indices.reserve(shapes[s].mesh.indices.size() + indices.size());
00088
00089     // Loop over faces(polygon)
00090     size_t index_offset = 0;
00091     for (size_t f = 0; f < shapes[s].mesh.num_face_vertices.size(); f++) {
00092
00093         size_t fv = size_t(shapes[s].mesh.num_face_vertices[f]);
00094
00095         // Loop over vertices in the face.
00096         for (size_t v = 0; v < fv; v++) {
00097
00098             // access to vertex
00099             tinyobj::index_t idx = shapes[s].mesh.indices[index_offset + v];
00100             tinyobj::real_t vx = attrib.vertices[3 * size_t(idx.vertex_index) + 0];
00101             tinyobj::real_t vy = attrib.vertices[3 * size_t(idx.vertex_index) + 1];
00102             tinyobj::real_t vz = attrib.vertices[3 * size_t(idx.vertex_index) + 2];
00103             glm::vec3 pos = { vx, vy, vz };
00104
00105             minX = std::min(minX, pos.x);
00106             maxX = std::max(maxX, pos.x);
00107             minY = std::min(minY, pos.y);
00108             maxY = std::max(maxY, pos.y);
00109             minZ = std::min(minZ, pos.z);
00110             maxZ = std::max(maxZ, pos.z);
00111
00112             glm::vec3 normals(0.0f);
00113             // Check if 'normal_index' is zero or positive. negative = no normal data
00114             if (idx.normal_index >= 0 && !attrib.normals.empty()) {
00115                 tinyobj::real_t nx = attrib.normals[3 * size_t(idx.normal_index) + 0];
00116                 tinyobj::real_t ny = attrib.normals[3 * size_t(idx.normal_index) + 1];
00117                 tinyobj::real_t nz = attrib.normals[3 * size_t(idx.normal_index) + 2];
00118                 normals = glm::vec3(nx, ny, nz);
00119             }
00120
00121             glm::vec3 color(-1.f);
00122             if (!attrib.colors.empty()) {
00123                 tinyobj::real_t red = attrib.colors[3 * size_t(idx.vertex_index) + 0];
00124                 tinyobj::real_t green = attrib.colors[3 * size_t(idx.vertex_index) + 1];
00125                 tinyobj::real_t blue = attrib.colors[3 * size_t(idx.vertex_index) + 2];
00126                 color = glm::vec3(red, green, blue);
00127             }
00128
00129             glm::vec2 tex_coords(0.0f);
00130             // Check if 'texcoord_index' is zero or positive. negative = no texcoord data
00131             if (idx.texcoord_index >= 0 && !attrib.texcoords.empty()) {
00132                 tinyobj::real_t tx = attrib.texcoords[2 * size_t(idx.texcoord_index) + 0];
00133                 // flip y coordinate !
00134                 tinyobj::real_t ty = 1.f - attrib.texcoords[2 * size_t(idx.texcoord_index) + 1];
00135                 tex_coords = glm::vec2(tx, ty);
00136             }
00137 }
```

```

00138     Vertex vert{ pos, normals, color, tex_coords };
00139
00140     if (vertices_map.count(vert) == 0) {
00141
00142         vertices_map[vert] = static_cast<uint32_t>(vertices.size());
00143         vertices.push_back(vert);
00144     }
00145
00146     indices.push_back(vertices_map[vert]);
00147 }
00148
00149     index_offset += fv;
00150
00151     // per-face material; face usually is triangle
00152     // matToTex[shapes[s].mesh.material_ids[f]]
00153     materialIndex.push_back(glm::vec4(shapes[s].mesh.material_ids[f], 0.0f, 0.0f, 0.0f));
00154 }
00155 }
00156
00157 // precompute normals if no provided
00158 if (attrib.normals.empty()) {
00159
00160     for (size_t i = 0; i < indices.size(); i += 3) {
00161         Vertex& v0 = vertices[indices[i + 0]];
00162         Vertex& v1 = vertices[indices[i + 1]];
00163         Vertex& v2 = vertices[indices[i + 2]];
00164
00165         glm::vec3 n = glm::normalize(glm::cross((v1.position - v0.position), (v2.position -
00166             v0.position)));
00167         v0.normal = n;
00168         v1.normal = n;
00169         v2.normal = n;
00170     }
00171 }
00172
00173 ObjLoader::~ObjLoader() { }

```

7.129 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.h File Reference

```
#include <memory>
#include <stdexcept>
#include "Mesh.h"
#include "Texture.h"
#include "Vertex.h"
#include "ObjMaterial.h"
#include "host_device_shared.h"
```

Include dependency graph for ObjLoader.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ObjLoader](#)

7.130 ObjLoader.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <memory>
00003 #include <stdexcept>
00004
00005 #include "Mesh.h"
00006 #include "Texture.h"
00007 #include "Vertex.h"
00008 #include "ObjMaterial.h"
00009 #include "host_device_shared.h"
```

```

00010
00011 class ObjLoader {
00012 public:
00013     ObjLoader();
00014
00015     void load(std::string modelFile, std::vector<Vertex>& vertices, std::vector<unsigned int>& indices,
00016             std::vector<std::string>& texture_list,
00017             std::vector<ObjMaterial>& materials, std::vector<glm::vec4>& materialIndex);
00018
00019     ~ObjLoader();
00020
00021     private:
00022     GLfloat minX = std::numeric_limits<float>::max();
00023     GLfloat maxX = std::numeric_limits<float>::min();
00024     GLfloat minY = std::numeric_limits<float>::max();
00025     GLfloat maxY = std::numeric_limits<float>::min();
00026     GLfloat minZ = std::numeric_limits<float>::max();
00027     GLfloat maxZ = std::numeric_limits<float>::min();
00028
00029     static std::string get_base_dir(const std::string& filepath)
00030     {
00031         if (filepath.find_last_of("//\\") != std::string::npos) return filepath.substr(0,
00032             filepath.find_last_of("//\\"));
00033         return "";
00034     };

```

7.131 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.cpp File Reference

```
#include "ObjMaterial.h"
Include dependency graph for ObjMaterial.cpp:
```

7.132 ObjMaterial.cpp

[Go to the documentation of this file.](#)

```

00001 #include "ObjMaterial.h"
00002
00003 ObjMaterial::ObjMaterial()
00004 {
00005
00006     this->ambient = glm::vec3(0.1f, 0.1f, 0.1f);
00007     this->diffuse = glm::vec3(0.7f, 0.7f, 0.7f);
00008     this->specular = glm::vec3(1.0f, 1.0f, 1.0f);
00009     this->transmittance = glm::vec3(0.0f, 0.0f, 0.0f);
00010     this->emission = glm::vec3(0.0f, 0.0f, 0.10);
00011     this->shininess = 0.f;
00012     this->ior = 1.0f;
00013     this->dissolve = 1.f;
00014     this->illum = 0;
00015     this->textureID = -1;
00016 }
00017
00018 ObjMaterial::ObjMaterial(glm::vec3 ambient, glm::vec3 diffuse, glm::vec3 specular, glm::vec3
00019     transmittance, glm::vec3 emission, float shininess, float ior,
00020     float dissolve, int illum, int textureID)
00021 {
00022     this->ambient = ambient;
00023     this->diffuse = diffuse;
00024     this->specular = specular;
00025     this->transmittance = transmittance;
00026     this->emission = emission;
00027     this->shininess = shininess;
00028     this->ior = ior;
00029     this->dissolve = dissolve;
00030     this->illum = illum;
00031     this->textureID = textureID;
00032 }
00033 ObjMaterial::~ObjMaterial() { }
```

7.133 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.h File Reference

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "GlobalValues.h"
#include "host_device_shared.h"
#include "ShaderProgram.h"
```

Include dependency graph for ObjMaterial.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ObjMaterial](#)

7.134 ObjMaterial.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glad/glad.h>
00003 #include <GLFW/glfw3.h>
00004 #include <glm/glm.hpp>
00005 #include <glm/gtc/matrix_transform.hpp>
00006 #include <glm/gtc/type_ptr.hpp>
00007
00008 #include "GlobalValues.h"
00009 #include "host_device_shared.h"
00010 #include "ShaderProgram.h"
00011
00012 class ObjMaterial {
00013     public:
00014     ObjMaterial();
00015
00016     ObjMaterial(glm::vec3 ambient, glm::vec3 diffuse, glm::vec3 specular, glm::vec3 transmittance,
00017                 glm::vec3 emission, float shininess, float ior, float dissolve,
00018                 int illum, int textureID);
00019
00020     glm::vec3 get_ambient() const { return ambient; }
00021     glm::vec3 get_diffuse() const { return diffuse; }
00022     glm::vec3 get_specular() const { return specular; }
00023     glm::vec3 get_transmittance() const { return transmittance; }
00024     glm::vec3 get_emission() const { return emission; }
00025
00026     float get_shininess() const { return shininess; }
00027     float get_iор() const { return ior; }
00028     float get_dissolve() const { return dissolve; }
00029
00030     int get_illum() const { return illum; }
00031     int get_textureID() const { return textureID; }
00032
00033     glm::vec3 ambient = glm::vec3(0.1f, 0.1f, 0.1f);
00034     glm::vec3 diffuse = glm::vec3(0.7f, 0.7f, 0.7f);
00035     glm::vec3 specular = glm::vec3(1.0f, 1.0f, 1.0f);
00036     glm::vec3 transmittance = glm::vec3(0.0f, 0.0f, 0.0f);
00037     glm::vec3 emission = glm::vec3(0.0f, 0.0f, 0.1f);
00038     float shininess = 0.f;
00039     float ior = 1.0f; // index of refraction
00040     float dissolve = 1.f; // 1 == opaque; 0 == fully transparent
00041
00042     int illum = 0;
00043     int textureID = -1;
00044
00045     ~ObjMaterial();
00046
00047     private:
```

7.135 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.cpp File Reference

```
#include "Quad.h"
Include dependency graph for Quad.cpp:
```

7.136 Quad.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Quad.h"
00002
00003 Quad::Quad()
00004 {
00005     glGenVertexArrays(1, &q_vao);
00006     glGenBuffers(1, &q_vbo);
00007
00008     glBindVertexArray(q_vao);
00009     glBindBuffer(GL_ARRAY_BUFFER, q_vbo);
00010     glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), &vertices, GL_STATIC_DRAW);
00011     glEnableVertexAttribArray(0);
00012     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
00013     glEnableVertexAttribArray(1);
00014     glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(3 * sizeof(float)));
00015 }
00016
00017 void Quad::render()
00018 {
00019     glBindVertexArray(q_vao);
00020     glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
00021     glBindVertexArray(0);
00022 }
00023
00024 Quad::~Quad()
00025 {
00026     glDeleteVertexArrays(1, &q_vao);
00027     glDeleteBuffers(1, &q_vbo);
00028 }
```

7.137 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.h File Reference

```
#include <glm/glm.hpp>
#include <vector>
#include <glad/glad.h>
#include "GlobalValues.h"
```

Include dependency graph for Quad.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Quad](#)

7.138 Quad.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glm/glm.hpp>
00003 #include <vector>
00004 #include <glad/glad.h>
00005
00006 #include "GlobalValues.h"
00007
00008 class Quad {
00009     public:
00010     Quad();
00011
00012     void render();
00013
00014     ~Quad();
00015
00016     private:
00017     GLuint q_vao, q_vbo;
00018
00019     float vertices[20] = {
00020         //positions           //tex coords
00022         -1.0f,
00023         1.0f,
00024         0.0f,
00025         0.0f,
00026         1.0f,
00027         -1.0f,
00028         -1.0f,
00029         0.0f,
00030         0.0f,
00031         0.0f,
00032         1.0f,
00033         1.0f,
00034         0.0f,
00035         1.0f,
00036         1.0f,
00037         1.0f,
00038         -1.0f,
00039         0.0f,
00040         1.0f,
00041         0.0f
00042     };
00043 };
00044 };
```

7.139 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Rotatio.n.h File Reference

```
#include <glad/glad.h>
#include <glm/glm.hpp>
```

Include dependency graph for Rotation.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [Rotation](#)

7.140 Rotation.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glad/glad.h>
00003 #include <glm/glm.hpp>
00004
00005 struct Rotation {
00006
00007     GLfloat degrees;
00008     glm::vec3 axis;
00009 };
```

7.141 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.cpp File Reference

```
#include "Scene.h"
#include <sstream>
Include dependency graph for Scene.cpp:
```

7.142 Scene.cpp

[Go to the documentation of this file.](#)

```
00001 #include "Scene.h"
00002
00003 #include <sstream>
00004
00005 Scene::Scene() :
00006     main_camera(), sun(std::make_shared<DirectionalLight>(static_cast<uint32_t>(4096),
00007                         static_cast<uint32_t>(4096), 1.f, 1.f, 1.f, 1.f, -0.1f, -0.8f, -0.1f,
00008                         main_camera->get_near_plane(), main_camera->get_far_plane(), NUM_CASCADES)),
00009     clouds(std::make_shared<Clouds>()), main_window(),
00010     view_frustum_culling(std::make_shared<ViewFrustumCulling>(ViewFrustumCulling{})), progress(0.f),
00011     loaded_scene(false), context_setup(false)
00012 {
00013 }
00014 Scene::Scene(std::shared_ptr<Camera> main_camera, std::shared_ptr<Window> main_window) :
00015     main_camera(main_camera), sun(std::make_shared<DirectionalLight>(static_cast<uint32_t>(4096),
00016                         static_cast<uint32_t>(4096), 1.f, 1.f, 1.f, 1.f, -0.1f, -0.8f,
00017                         -0.1f, main_camera->get_near_plane(), main_camera->get_far_plane(),
00018                         NUM_CASCADES)),
00019     clouds(std::make_shared<Clouds>()), main_window(main_window),
00020     view_frustum_culling(std::make_shared<ViewFrustumCulling>(ViewFrustumCulling{})),
00021     progress(0.f), loaded_scene(false), context_setup(false)
00022 {
00023
00024     point_lights.reserve(MAX_POINT_LIGHTS);
00025     point_lights.push_back(std::make_shared<PointLight>(
00026         static_cast<uint32_t>(1024), static_cast<uint32_t>(1024), 0.01f, 100.f, 0.0f, 1.0f, 0.0f, 1.0f,
00027         0.0f, 0.0f, 0.1f, 0.1f, 0.1f));
00028     point_lights[0]->set_position(glm::vec3(0.0, -24.f, -24.0));
00029 }
00030
00031 GLuint Scene::get_point_light_count() const { return static_cast<uint32_t>(point_lights.size()); }
00032
00033 std::shared_ptr<DirectionalLight> Scene::get_sun() { return sun; }
00034
00035 std::vector<std::shared_ptr<PointLight>> Scene::get_point_lights() const { return point_lights; }
00036
00037 void Scene::load_models()
00038 {
00039
00040     glm::vec3 sponza_offset = glm::vec3(0.f, 0.0f, 0.0f);
00041     GLfloat sponza_scale = 10.f;
00042     Rotation sponza_rot;
00043     sponza_rot.degrees = 0.0f;
00044     sponza_rot.axis = glm::vec3(0.0f, 1.0f, 0.0f);
00045
00046     glm::vec3 clouds_offset = glm::vec3(-3.f, 20.0f, -3.0f);
00047     glm::vec3 clouds_scale = glm::vec3(1.f, 1.f, 1.f);
00048     clouds->set_scale(clouds_scale);
00049     clouds->set_translation(clouds_offset);
00050
00051     std::stringstream modelFile;
00052     modelFile << CMAKELISTS_DIR << "/Resources/Models/dinosaurs.obj";
00053     /*..../Resources/Models/Pillum/PillumPainting_Export.obj",*/
00054     /*..../Resources/Models/crytek-sponza/sponza_trigl.obj",*/
00055
00056     std::shared_ptr<GameObject> sponza = std::make_shared<GameObject>(modelFile.str(), sponza_offset,
00057     sponza_scale, sponza_rot);
00058     progress += 1.f;
00059     game_objects.push_back(sponza);
00060 }
```

```

00061     mx_isLoaded.lock();
00062     loaded_scene = true;
00063     mx_isLoaded.unlock();
00064 }
00065
00066 std::vector<ObjMaterial> Scene::get_materials() { return
00067     game_objects[0]->get_model()->get_materials(); }
00068
00069 bool Scene::is_loaded()
00070 {
00071     std::lock_guard<std::mutex> guard{ mx_isLoaded };
00072     return loaded_scene;
00073 }
00074 GLfloat Scene::get_progress()
00075 {
00076     std::lock_guard<std::mutex> guard{ mx_progress };
00077     return progress;
00078 }
00079
00080 void Scene::setup_game_object_context()
00081 {
00082     game_objects[0]->get_model()->create_render_context();
00083     context_setup = true;
00084 }
00085
00086 void Scene::bind_textures_and_buffer() { game_objects[0]->get_model()->bind_ressources(); }
00087
00088 void Scene::unbind_textures_and_buffer() { game_objects[0]->get_model()->unbind_resources(); }
00089
00090 int Scene::get_texture_count(int index) { return game_objects[0]->get_model()->get_texture_count(); }
00091
00092 void Scene::set_context_setup(bool context_setup) { this->context_setup = context_setup; }
00093
00094 bool Scene::get_context_setup() const { return context_setup; }
00095
00096 void Scene::add_game_object(const std::string& model_path, glm::vec3 translation, GLfloat scale,
00097     Rotation rot)
00098 {
00099     game_objects.push_back(std::make_shared<GameObject>(GameObject()));
00100     game_objects.back()->init(model_path, translation, scale, rot);
00101 }
00102
00103 std::shared_ptr<Clouds> Scene::get_clouds() { return clouds; }
00104
00105 std::vector<std::shared_ptr<GameObject>> Scene::get_game_objects() const { return game_objects; }
00106
00107 bool Scene::object_is_visible(std::shared_ptr<GameObject> game_object)
00108 {
00109     return view_frustum_culling->is_inside(static_cast<GLfloat>(main_window->get_buffer_width()) /
00110         static_cast<GLfloat>(main_window->get_buffer_height()),
00111         main_camera,
00112         game_object->get_aabb(),
00113         game_object->get_world_trafo());
00114 }
00115
00116 Scene::~Scene() { }

```

7.143 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.h File Reference

```

#include <vector>
#include <mutex>
#include <thread>
#include "GameObject.h"
#include "ViewFrustumCulling.h"
#include "RenderPassSceneDependend.h"
#include "Window.h"
#include "Clouds.h"
#include "ObjMaterial.h"
#include "PointLight.h"
#include "Rotation.h"
#include "DirectionalLight.h"

```

Include dependency graph for Scene.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class Scene

7.144 Scene.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <vector>
00003 #include <mutex>
00004 #include <thread>
00005
00006 #include "GameObject.h"
00007 #include "ViewFrustumCulling.h"
00008 #include "RenderPassSceneDependend.h"
00009 #include "Window.h"
00010 #include "Clouds.h"
00011 #include "ObjMaterial.h"
00012 #include "PointLight.h"
00013 #include "Rotation.h"
00014 #include "DirectionalLight.h"
00015
00016 class Scene {
00017
00018     public:
00019     Scene();
00020     Scene(std::shared_ptr<Camera> main_camera, std::shared_ptr<Window> main_window);
00021
00022     std::thread spawn()
00023     {
00024         return std::thread([=] { load_models(); });
00025     }
00026
00027     GLuint get_point_light_count() const;
00028     std::shared_ptr<DirectionalLight> get_sun();
00029     std::vector<std::shared_ptr<PointLight>> get_point_lights() const;
00030     std::vector<ObjMaterial> get_materials();
00031     GLfloat get_progress();
00032     int get_texture_count(int index);
00033     bool get_context_setup() const;
00034     std::shared_ptr<Clouds> get_clouds();
00035     std::vector<std::shared_ptr<GameObject>> get_game_objects() const;
00036
00037
00038     void add_game_object(const std::string& model_path, glm::vec3 translation, GLfloat scale, Rotation
rot);
00039     void load_models();
00040
00041     bool is_loaded();
00042     void setup_game_object_context();
00043
00044     void bind_textures_and_buffer();
00045     void unbind_textures_and_buffer();
00046
00047     void set_context_setup(bool context_setup);
00048
00049
00050     ~Scene();
00051
00052     private:
00053     bool object_is_visible(std::shared_ptr<GameObject> game_object);
00054
00055     std::shared_ptr<Camera> main_camera;
00056
00057     //lights
00058     std::shared_ptr<DirectionalLight> sun;
00059     std::vector<std::shared_ptr<PointLight>> point_lights;
00060     std::shared_ptr<Clouds> clouds;
00061
00062     std::shared_ptr<Window> main_window;
00063     std::shared_ptr<ViewFrustumCulling> view_frustum_culling;
00064
00065     std::vector<std::shared_ptr<GameObject>> game_objects;
00066
00067     GLfloat progress;
00068     bool loaded_scene;
00069
00070     std::mutex mx_progress;
00071     std::mutex mx_isLoaded;
00072     std::mutex mx_space_ship;
00073
00074     bool context_setup;
00075 };

```

7.145 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.cpp File Reference

```
#include "ShadowMap.h"
Include dependency graph for ShadowMap.cpp:
```

7.146 ShadowMap.cpp

[Go to the documentation of this file.](#)

```
00001 #include "ShadowMap.h"
00002
00003
00004 ShadowMap::ShadowMap() :
00005     FBO(0), shadow_map(0), shadow_width(0), shadow_height(0)
00006
00007
00008 {
00009 }
00010
00011 bool ShadowMap::init(GLuint width, GLuint height)
00012 {
00013     shadow_width = width;
00014     shadow_height = height;
00015
00016     glGenFramebuffers(1, &FBO);
00017
00018     glGenTextures(1, &shadow_map);
00019     glBindTexture(GL_TEXTURE_2D, shadow_map);
00020     glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, shadow_width, shadow_height, 0,
00021                 GL_DEPTH_COMPONENT, GL_FLOAT, nullptr);
00022
00023     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00024     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00025
00026     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
00027     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
00028
00029     float border_color[] = { 1.0f, 1.0f, 1.0f, 1.0f };
00030     glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
00031
00032     glBindFramebuffer(GL_FRAMEBUFFER, FBO);
00033     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, shadow_map, 0);
00034
00035     glDrawBuffer(GL_NONE);
00036     glReadBuffer(GL_NONE);
00037
00038     GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
00039
00040     if (status != GL_FRAMEBUFFER_COMPLETE) {
00041         printf("Framebuffer error: %i\n", status);
00042         return false;
00043     }
00044
00045     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00046
00047     return true;
00048 }
00049
00050 void ShadowMap::write() { glBindFramebuffer(GL_FRAMEBUFFER, FBO); }
00051
00052 void ShadowMap::read(GLenum texture_unit)
00053 {
00054
00055     glActiveTexture(texture_unit);
00056     glBindTexture(GL_TEXTURE_2D, shadow_map);
00057 }
00058
00059 ShadowMap::~ShadowMap()
00060 {
00061     if (FBO) {
00062         glDeleteFramebuffers(1, &FBO);
00063     }
00064
00065     if (shadow_map) {
00066         glDeleteTextures(1, &shadow_map);
00067     }
00068 }
```

7.147 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.h File Reference

```
#include <stdio.h>
#include <glad/glad.h>
```

Include dependency graph for ShadowMap.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ShadowMap](#)

7.148 ShadowMap.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <stdio.h>
00004 #include <glad/glad.h>
00005
00006 class ShadowMap {
00007     public:
00008     ShadowMap();
00009
00010     virtual bool init(GLuint width, GLuint height);
00011     virtual void write();
00012     virtual void read(GLenum texture_unit);
00013
00014     GLuint get_shadow_width() const { return shadow_width; };
00015     GLuint get_shadow_height() const { return shadow_height; };
00016     GLuint get_id() const { return shadow_map; };
00017
00018     virtual ~ShadowMap();
00019
00020     protected:
00021     GLuint FBO, shadow_map;
00022     GLuint shadow_width, shadow_height;
00023 };
```

7.149 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.cpp File Reference

```
#include "SkyBox.h"
#include <array>
#include <sstream>
```

Include dependency graph for SkyBox.cpp:

7.150 SkyBox.cpp

[Go to the documentation of this file.](#)

```
00001 #include "SkyBox.h"
00002 #include <array>
00003 #include <sstream>
00004
00005 SkyBox::SkyBox()
00006 {
00007     std::stringstream skybox_base_dir;
00008     skybox_base_dir << CMAKELISTS_DIR;
00009     skybox_base_dir << "/Resources/Textures/Skybox/DOOM2016/";
```

```
00010
00011     std::stringstream texture_loading;
00012     std::array<std::string, 6> skybox_textures = { "DOOM16RT.png", "DOOM16LF.png", "DOOM16UP.png",
00013     "DOOM16DN.png", "DOOM16FT.png", "DOOM16BK.png" };
00014     std::vector<std::string> skybox_faces;
00015
00016     for (uint32_t i = 0; i < static_cast<uint32_t>(skybox_textures.size()); i++) {
00017
00018         texture_loading << skybox_base_dir.str() << skybox_textures[i];
00019         skybox_faces.push_back(texture_loading.str());
00020         texture_loading.str(std::string());
00021     }
00022
00023 //time_t timer;
00024 //srand(time(&timer));
00025 srand(0);
00026 shader_playback_time = 1;
00027
00028 shader_program = std::make_shared<ShaderProgram>();
00029 shader_program->create_from_files("skybox/SkyBox.vert", "skybox/SkyBox.frag");
00030
00031 //texture setup
00032 glGenTextures(1, &texture_id);
00033 glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id);
00034
00035     int width, height, bit_depth;
00036
00037     for (size_t i = 0; i < 6; i++) {
00038
00039         unsigned char* texture_data = stbi_load(skybox_faces[i].c_str(), &width, &height, &bit_depth, 0);
00040         if (!texture_data) {
00041             printf("Failed to find: %s\n", skybox_faces[i].c_str());
00042             return;
00043         }
00044
00045         glTexImage2D(static_cast<GLenum>(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i), 0, GL_RGBA, width, height,
00046         0, GL_RGBA, GL_UNSIGNED_BYTE, texture_data);
00047
00048         stbi_image_free(texture_data);
00049     }
00050
00051     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00052     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00053     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
00054     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00055     glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00056
00057 // Mesh Setup
00058     std::vector<unsigned int> sky_box_indices = { //front
00059         0,
00060         1,
00061         2,
00062         2,
00063         1,
00064         //right
00065         2,
00066         3,
00067         5,
00068         5,
00069         3,
00070         7,
00071         //back
00072         5,
00073         7,
00074         4,
00075         4,
00076         7,
00077         6,
00078         //left
00079         4,
00080         6,
00081         0,
00082         0,
00083         6,
00084         1,
00085         //top
00086         4,
00087         0,
00088         5,
00089         5,
00090         0,
00091         2,
00092         //bottom
00093         1,
00094         6,
```

```

00095     3,
00096     3,
00097     6,
00098     7
00099 };
00100 std::vector<Vertex> sky_box_vertices = {
00102     Vertex(glm::vec3(-1.0f, 1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00103             glm::vec2(0.0f, 0.0f)),
00104     Vertex(glm::vec3(-1.0f, -1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00105             glm::vec2(0.0f, 0.0f)),
00106     Vertex(glm::vec3(1.0f, 1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00107             glm::vec2(0.0f, 0.0f)),
00108     Vertex(glm::vec3(1.0f, -1.0f, -1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00109             glm::vec2(0.0f, 0.0f)),
00110
00111     Vertex(glm::vec3(-1.0f, 1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00112             glm::vec2(0.0f, 0.0f)),
00113     Vertex(glm::vec3(1.0f, 1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00114             glm::vec2(0.0f, 0.0f)),
00115     Vertex(glm::vec3(-1.0f, -1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00116             glm::vec2(0.0f, 0.0f)),
00117     Vertex(glm::vec3(1.0f, -1.0f, 1.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 0.0f),
00118             glm::vec2(0.0f, 0.0f)),
00119 };
00120
00121 sky_mesh = std::make_shared<Mesh>(sky_box_vertices, sky_box_indices);
00122 }
00124
00125 void SkyBox::draw_sky_box(glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width,
00126                             GLuint window_height, GLfloat delta_time)
00127 {
00128     // https://learnopengl.com/Advanced-OpenGL/Cubemaps
00129     GLfloat velocity = movement_speed * delta_time;
00130     shader_playback_time = static_cast<GLfloat>(fmod(shader_playback_time + velocity, 10000));
00131
00132     glm::mat4 new_view_matrix = glm::mat4(glm::mat3(view_matrix));
00133
00134     glDepthMask(GL_FALSE);
00135     glDepthFunc(GL_LEQUAL);
00136     //std::time_t now = std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
00137
00138     shader_program->use_shader_program();
00139
00140     shader_program->setUniformMatrix4fv(projection_matrix, "projection");
00141     shader_program->setUniformMatrix4fv(new_view_matrix, "view");
00142     shader_program->setUniformInt(SKYBOX_TEXTURES_SLOT, "skybox");
00143
00144     glBindTexture(GL_TEXTURE0 + SKYBOX_TEXTURES_SLOT);
00145     glBindTexture(GL_TEXTURE_CUBE_MAP, texture_id);
00146
00147     shader_program->validate_program();
00148
00149     sky_mesh->render();
00150
00151     glDepthMask(GL_TRUE);
00152     glDepthFunc(GL_LESS);
00153
00154     glBindTexture(GL_TEXTURE_CUBE_MAP, 0);
00155 }
00156
00157 void SkyBox::reload()
00158 {
00159     shader_program = std::make_shared<ShaderProgram>();
00160     shader_program->create_from_files("Shaders/SkyBox.vert", "Shaders/SkyBox.frag");
00161 }
00162
00163 SkyBox::~SkyBox() { glDeleteTextures(1, &texture_id); }

```

7.151 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.h File Reference

```
#include <vector>
#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <sstream>
#include <ctime>
#include <chrono>
#include <random>
#include <cassert>
#include <time.h>
#include "GlobalValues.h"
#include "Mesh.h"
#include "Vertex.h"
#include "ShaderProgram.h"
#include "bindings.h"
```

Include dependency graph for SkyBox.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [SkyBox](#)

7.152 SkyBox.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <vector>
00003
00004 #include <glad/glad.h>
00005 #include <glm/glm.hpp>
00006 #include <glm/gtc/matrix_transform.hpp>
00007 #include <glm/gtc/type_ptr.hpp>
00008 #include <sstream>
00009 #include <ctime>
00010 #include <chrono>
00011 #include <random>
00012 #include <cassert>
00013 #include <time.h>
00014
00015 #include "GlobalValues.h"
00016 #include "Mesh.h"
00017 #include "Vertex.h"
00018 #include "ShaderProgram.h"
00019 #include "bindings.h"
00020
00021 class SkyBox {
00022     public:
00023     SkyBox();
00024
00025     void draw_sky_box(glm::mat4 projection_matrix, glm::mat4 view_matrix, GLuint window_width, GLuint
window_height, GLfloat delta_time);
00026
00027     void reload();
00028
00029     ~SkyBox();
00030
00031     private:
00032     GLfloat movement_speed = 0.1f;
00033
00034     GLfloat shader_playback_time;
00035
00036     std::shared_ptr<Mesh> sky_mesh;
00037     std::shared_ptr<ShaderProgram> shader_program;
00038
00039     GLuint texture_id;
00040     GLuint uniform_projection, uniform_view;
00041 };
```

7.153 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.cpp File Reference

```
#include "ClampToEdgeMode.h"
Include dependency graph for ClampToEdgeMode.cpp:
```

7.154 ClampToEdgeMode.cpp

[Go to the documentation of this file.](#)

```
00001 #include "ClampToEdgeMode.h"
00002
00003 ClampToEdgeMode::ClampToEdgeMode() { }
00004
00005 void ClampToEdgeMode::activate()
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00009 }
00010
00011 ClampToEdgeMode::~ClampToEdgeMode() { }
```

7.155 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.h File Reference

```
#include "TextureWrappingMode.h"
```

Include dependency graph for ClampToEdgeMode.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ClampToEdgeMode](#)

7.156 ClampToEdgeMode.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "TextureWrappingMode.h"
00003
00004 class ClampToEdgeMode : public TextureWrappingMode {
00005     public:
00006     ClampToEdgeMode();
00007
00008     void activate() override;
00009
00010     ~ClampToEdgeMode();
00011
00012     private:
00013 };
```

7.157 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/MirroredRepeatMode.cpp File Reference

```
#include "MirroredRepeatMode.h"
Include dependency graph for MirroredRepeatMode.cpp:
```

7.158 MirroredRepeatMode.cpp

[Go to the documentation of this file.](#)

```
00001 #include "MirroredRepeatMode.h"
00002
00003 MirroredRepeatMode::MirroredRepeatMode() { }
00004
00005 void MirroredRepeatMode::activate()
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
00009 }
00010
00011 MirroredRepeatMode::~MirroredRepeatMode() { }
```

7.159 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/← MirroredRepeatMode.h File Reference

```
#include "TextureWrappingMode.h"
```

Include dependency graph for MirroredRepeatMode.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [MirroredRepeatMode](#)

7.160 MirroredRepeatMode.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "TextureWrappingMode.h"
00003
00004 class MirroredRepeatMode : public TextureWrappingMode {
00005     public:
00006     MirroredRepeatMode();
00007
00008     void activate() override;
00009
00010     ~MirroredRepeatMode();
00011
00012     private:
00013 },
```

7.161 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Repeat← Mode.cpp File Reference

```
#include "RepeatMode.h"
```

Include dependency graph for RepeatMode.cpp:

7.162 RepeatMode.cpp

[Go to the documentation of this file.](#)

```
00001 #include "RepeatMode.h"
00002
00003 RepeatMode::RepeatMode() { }
00004
00005 void RepeatMode::activate()
00006 {
00007     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
00008     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
00009 }
00010
00011 RepeatMode::~RepeatMode() { }
```

7.163 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/RepeatMode.h File Reference

#include "TextureWrappingMode.h"

Include dependency graph for RepeatMode.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [RepeatMode](#)

7.164 RepeatMode.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include "TextureWrappingMode.h"
00003 class RepeatMode : public TextureWrappingMode {
00004     public:
00005     RepeatMode();
00006
00007     void activate() override;
00008
00009     ~RepeatMode();
00010
00011     private:
00012 };
```

7.165 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Texture.cpp File Reference

#include <iostream>

#include "Texture.h"

Include dependency graph for Texture.cpp:

Macros

- #define [STB_IMAGE_IMPLEMENTATION](#)

7.165.1 Macro Definition Documentation

7.165.1.1 STB_IMAGE_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

Definition at line 1 of file [Texture.cpp](#).

7.166 Texture.cpp

[Go to the documentation of this file.](#)

```
00001 #define STB_IMAGE_IMPLEMENTATION
00002
00003 #include <iostream>
00004 #include "Texture.h"
00005
00006 Texture::Texture() :
00007
00008     textureID(0), width(0), height(0), bit_depth(0),
00009     //go with repeat as standard ...
00010     wrapping_mode(std::make_shared<RepeatMode>()), file_location(std::string(""))
00011
00012 {
00013 }
00014
00015 Texture::Texture(const char* file_loc, std::shared_ptr<TextureWrappingMode> wrapping_mode) :
00016
00017     textureID(0), width(0), height(0), bit_depth(0),
00018     //go with repeat as standard ...
00019     wrapping_mode(wrapping_mode), file_location(std::string(file_loc))
00020
00021 {
00022 }
00023
00024 bool Texture::load_texture_without_alpha_channel()
00025 {
00026
00027     stbi_set_flip_vertically_on_load(true);
00028     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00029     if (!texture_data) {
00030         printf("Failed to find: %s\n", file_location.c_str());
00031         return false;
00032     }
00033
00034     glGenTextures(1, &textureID);
00035     glBindTexture(GL_TEXTURE_2D, textureID);
00036
00037     wrapping_mode->activate();
00038
00039     // i think we won't need nearest option; so stick to linear
00040     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00041
00042     //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00043     // INVALID ENUM changed to GL_LINEAR
00044     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00045
00046     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, texture_data);
00047     glGenerateMipmap(GL_TEXTURE_2D);
00048
00049     glBindTexture(GL_TEXTURE_2D, 0);
00050
00051     stbi_image_free(texture_data);
00052
00053     return true;
00054 }
00055
00056 bool Texture::load_texture_with_alpha_channel()
00057 {
00058
00059     stbi_set_flip_vertically_on_load(true);
00060     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00061     if (!texture_data) {
```

```

00062     printf("Failed to find: %s\n", file_location.c_str());
00063     return false;
00064 }
00065
00066 glGenTextures(1, &textureID);
00067 glBindTexture(GL_TEXTURE_2D, textureID);
00068
00069 wrapping_mode->activate();
00070
00071 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, texture_data);
00072 glGenerateMipmap(GL_TEXTURE_2D);
00073
00074 // to not interpolate between transparent and not trans coloars
00075 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00076 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00077
00078 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00079 //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00080 // INVALID ENUM changed to GL_LINEAR
00081 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00082
00083 glBindTexture(GL_TEXTURE_2D, 0);
00084
00085 stbi_image_free(texture_data);
00086
00087 return true;
00088 }
00089
00090 bool Texture::load_SRGB_texture_without_alpha_channel()
00091 {
00092
00093     stbi_set_flip_vertically_on_load(true);
00094     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00095     if (!texture_data) {
00096         printf("Failed to find: %s\n", file_location.c_str());
00097         return false;
00098     }
00099
00100    glGenTextures(1, &textureID);
00101    glBindTexture(GL_TEXTURE_2D, textureID);
00102
00103    wrapping_mode->activate();
00104
00105 // i think we won't need nearest option; so stick to linear
00106 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00107
00108 //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00109 // INVALID ENUM changed to GL_LINEAR
00110 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00111
00112 glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, texture_data);
00113 glGenerateMipmap(GL_TEXTURE_2D);
00114
00115 glBindTexture(GL_TEXTURE_2D, 0);
00116
00117 stbi_image_free(texture_data);
00118
00119 return true;
00120 }
00121
00122 bool Texture::load_SRGB_texture_with_alpha_channel()
00123 {
00124
00125     stbi_set_flip_vertically_on_load(true);
00126     unsigned char* texture_data = stbi_load(file_location.c_str(), &width, &height, &bit_depth, 0);
00127     if (!texture_data) {
00128         printf("Failed to find: %s\n", file_location.c_str());
00129         return false;
00130     }
00131
00132    glGenTextures(1, &textureID);
00133    glBindTexture(GL_TEXTURE_2D, textureID);
00134
00135    wrapping_mode->activate();
00136
00137    glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB_ALPHA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
00138    texture_data);
00139    glGenerateMipmap(GL_TEXTURE_2D);
00140
00141 // to not interpolate between transparent and not trans coloars
00142 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00143 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00144
00145 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00146 //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
00147 // INVALID ENUM changed to GL_LINEAR

```

```

00148     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00149
00150     glBindTexture(GL_TEXTURE_2D, 0);
00151
00152     stbi_image_free(texture_data);
00153
00154     return true;
00155 }
00156
00157 std::string Texture::get_filename() const { return file_location; }
00158
00159 void Texture::use_texture(unsigned int index)
00160 {
00161     glActiveTexture(GL_TEXTURE0 + index);
00162     glBindTexture(GL_TEXTURE_2D, textureID);
00163 }
00164
00165 void Texture::unbind_texture(unsigned int index) { glBindTexture(GL_TEXTURE_2D, GL_TEXTURE0 + index); }
00166
00167 void Texture::clear_texture_context()
00168 {
00169     glDeleteTextures(1, &textureID);
00170     textureID = 0;
00171     width = 0;
00172     height = 0;
00173     bit_depth = 0;
00174     file_location = std::string("");
00175 }
00176
00177 GLuint Texture::get_id() const { return textureID; }
00178
00179 Texture::~Texture() { clear_texture_context(); }

```

7.167 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/← Texture.h File Reference

```

#include <glad/glad.h>
#include <stb_image.h>
#include <string.h>
#include <memory>
#include <string>
#include "TextureWrappingMode.h"
#include "RepeatMode.h"
#include "MirroredRepeatMode.h"
#include "ClampToEdgeMode.h"
#include "GlobalValues.h"

```

Include dependency graph for Texture.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Texture](#)

7.168 Texture.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002
00003 #include <glad/glad.h>
00004 #include <stb_image.h>
00005 #include <string.h>
00006 #include <memory>
00007 #include <string>
00008
00009 #include "TextureWrappingMode.h"

```

```

00010 #include "RepeatMode.h"
00011 #include "MirroredRepeatMode.h"
00012 #include "ClampToEdgeMode.h"
00013 #include "GlobalValues.h"
00014
00015 class Texture {
00016     public:
00017     Texture();
00018     Texture(const char* file_loc, std::shared_ptr<TextureWrappingMode> wrapping_mode);
00019
00020     bool load_texture_without_alpha_channel();
00021     bool load_texture_with_alpha_channel();
00022
00023     bool load_SRGB_texture_without_alpha_channel();
00024     bool load_SRGB_texture_with_alpha_channel();
00025
00026     std::string get_filename() const;
00027     GLuint get_id() const;
00028
00029     void use_texture(unsigned int index);
00030     void unbind_texture(unsigned int index);
00031
00032     void clear_texture_context();
00033
00034
00035     ~Texture();
00036
00037     private:
00038     GLuint textureID;
00039     int width, height, bit_depth;
00040
00041     std::shared_ptr<TextureWrappingMode> wrapping_mode;
00042
00043     std::string file_location;
00044 };

```

7.169 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/← TextureWrappingMode.h File Reference

#include <glad/glad.h>

Include dependency graph for TextureWrappingMode.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [TextureWrappingMode](#)

7.170 TextureWrappingMode.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 //lets mimic and c++-style interface :
00003 #include <glad/glad.h>
00004
00005 class TextureWrappingMode {
00006     public:
00007     virtual void activate() = 0;
00008
00009     private:
00010 };

```

7.171 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Vertex.h File Reference

#include <glm/glm.hpp>

#include <glm/gtx/hash.hpp>

Include dependency graph for Vertex.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct `Vertex`
- struct `std::hash< Vertex >`

Namespaces

- namespace `std`

7.172 Vertex.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <glm/glm.hpp>
00003 #include <glm/gtx/hash.hpp>
00004
00005 struct Vertex {
00006     public:
00007     // this is public because to access the size of pos, norm, tex Cood.
00008     glm::vec3 position;
00009     glm::vec3 normal;
00010     glm::vec3 color;
00011     glm::vec2 texture_coords;
00012
00013     Vertex()
00014     {
00015
00016         this->position = glm::vec3(0);
00017         this->normal = glm::vec3(0);
00018         this->color = glm::vec3(0);
00019         this->texture_coords = glm::vec2(0);
00020     }
00021
00022     Vertex(glm::vec3 pos, glm::vec3 normal, glm::vec3 color, glm::vec2 texture_coords)
00023     {
00024
00025         this->position = pos;
00026         this->normal = normal;
00027         this->color = color;
00028         this->texture_coords = texture_coords;
00029     };
00030
00031     glm::vec3 get_position() const { return position; }
00032
00033     glm::vec3 get_normal() const { return normal; }
00034
00035     glm::vec3 get_color() const { return color; }
00036
00037     glm::vec2 get_tex_coors() const { return texture_coords; }
00038
00039     bool operator==(const Vertex& other) const
00040     {
00041
00042         return position == other.position && normal == other.normal && texture_coords ==
00043             other.texture_coords && color == other.color;
00044     };
00045
00046     namespace std {
00047
00048         template <> struct hash<Vertex> {
00049
00050             size_t operator()(Vertex const& vert) const
00051             {
00052
00053                 size_t h1 = hash<glm::vec3>()(vert.position);
00054                 size_t h2 = hash<glm::vec3>()(vert.normal);
00055                 size_t h3 = hash<glm::vec2>()(vert.texture_coords);
00056
00057                 // combine hashed wonderfully :)))
00058                 return ((h1 ^ (h2 << 1)) >> 1) ^ h3;
00059             }
00060         };
00061     }
```

7.173 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.cpp File Reference

```
#include "ViewFrustumCulling.h"
Include dependency graph for ViewFrustumCulling.cpp:
```

7.174 ViewFrustumCulling.cpp

[Go to the documentation of this file.](#)

```
00001 #include "ViewFrustumCulling.h"
00002
00003 ViewFrustumCulling::ViewFrustumCulling() :
00004     VBO(-1), VAO(-1), EBO(-1), m_drawCount(0),
00005     //we get that as input
00006     near_plane(0.f), far_plane(0.f), fov(0.f), ratio(0.f),
00007     //calculate as soon as we become params
00008     tan(0.f), near_height(0.f), near_width(0.f), far_height(0.f), far_width(0.f), main_camera(),
00009     dir(glm::vec3(0.f)), near_center(glm::vec3(0.f)), far_center(glm::vec3(0.f)),
00010     near_top_left(glm::vec3(0.f)), near_top_right(glm::vec3(0.f)), near_bottom_left(glm::vec3(0.f)),
00011     near_bottom_right(glm::vec3(0.f)),
00012     far_top_left(glm::vec3(0.f)), far_top_right(glm::vec3(0.f)), far_bottom_left(glm::vec3(0.f)),
00013     far_bottom_right(glm::vec3(0.f))
00014
00015
00016
00017
00018 {
00019 }
00020
00021
00022 bool ViewFrustumCulling::is_inside(GLfloat ratio, std::shared_ptr<Camera> main_camera,
00023     std::shared_ptr<AABB> bounding_box, glm::mat4 model)
00024 {
00025     GLfloat near_plane = main_camera->get_near_plane();
00026     GLfloat far_plane = main_camera->get_far_plane();
00027     GLfloat fov = main_camera->get_fov();
00028     update_frustum_param(near_plane, far_plane, fov, ratio, main_camera);
00029
00030     std::vector<glm::vec3> aabb_corners = bounding_box->get_corners(model);
00031
00032     //layout: [0]: near plane, [1] far plane, [2] up , [3] bottom , [4]: left ,
00033     // [5]: right
00034     //outcodes (binary) : 100000      , 010000      , 000100 , 001000      , 000010, 000001
00035     //outcodes (dezi)   : 32          , 16          , 4           , 8
00036     , 2          , 1
00037     bool result = true;
00038
00039     GLint outcode_near_plane = 32;
00040     GLint outcode_far_plane = 16;
00041     GLint outcode_up = 4;
00042     GLint outcode_bottom = 8;
00043     GLint outcode_left = 2;
00044     GLint outcode_right = 1;
00045     //GLint outcode;
00046
00047     GLint outcodes_pattern[NUM_FRUSTUM_PLANES] = { outcode_near_plane, outcode_far_plane, outcode_up,
00048         outcode_bottom, outcode_left, outcode_right };
00049
00050     for (int i = 0; i < NUM_FRUSTUM_PLANES; i++) {
00051
00052         frustum_plane plane = frustum_planes[i];
00053
00054         if (corners_outside_plane(aabb_corners, plane, outcodes_pattern[i])) {
00055
00056             result = false;
00057             break;
00058         }
00059     }
00060
00061
00062 void ViewFrustumCulling::render_view_frustum()
```

```

00063 {
00064     // seeing as we only have a single VAO there's no need to bind it every time,
00065     // but we'll do so to keep things a bit more organized
00066     glBindVertexArray(VAO);
00067     //glDrawArrays(GL_TRIANGLES, 0, 6);
00068     glDrawElements(GL_TRIANGLES, m_drawCount, GL_UNSIGNED_INT, 0);
00069
00070     //unbind all again
00071     glBindVertexArray(0);
00072 }
00073
00074 bool ViewFrustumCulling::corners_outside_plane(std::vector<glm::vec3> aabb_corners, frustum_plane
00075     plane, GLuint outcode_pattern)
00076 {
00077     GLint outcode = outcode_pattern;
00078
00079     for (int i = 0; i < static_cast<int>(aabb_corners.size()); i++) {
00080
00081         if (plane_point_distance(plane, aabb_corners[i]) < 0.0f) {
00082
00083             if (i == 0) {
00084                 outcode = outcode_pattern;
00085             } else {
00086                 outcode = outcode & outcode_pattern;
00087             }
00088         } else {
00089             if (i == 0) {
00090                 outcode = 0;
00091             } else {
00092                 outcode = outcode & 0;
00093             }
00094         }
00095     }
00096
00097     return (outcode != 0) ? true : false;
00098 }
00099
00100 GLfloat ViewFrustumCulling::plane_point_distance(frustum_plane plane, glm::vec3 corner)
00101 {
00102     GLfloat result = 0.0f;
00103
00104     glm::vec3 plane_normal = plane.normal;
00105     glm::vec3 plane_position = plane.position;
00106
00107     GLfloat d = glm::dot(plane_normal, plane_position);
00108
00109     result = (glm::dot(plane_normal, corner) - d) / glm::length(plane_normal);
00110
00111     return result;
00112 }
00113
00114 void ViewFrustumCulling::update_frustum_param(GLfloat near_plane, GLfloat far_plane, GLfloat fov,
00115     GLfloat ratio, std::shared_ptr<Camera> main_camera)
00116 {
00117     this->near_plane = near_plane;
00118     this->far_plane = far_plane;
00119     this->fov = fov;
00120     this->ratio = ratio;
00121
00122     tan = glm::tan(glm::radians(fov) * 0.5f);
00123     near_height = near_plane * tan;
00124     near_width = near_height * ratio;
00125     far_height = far_plane * tan;
00126     far_width = far_height * ratio;
00127
00128     this->main_camera = main_camera;
00129
00130     near_center = main_camera->get_camera_position() + main_camera->get_camera_direction() * near_plane;
00131     far_center = main_camera->get_camera_position() + main_camera->get_camera_direction() * far_plane;
00132
00133     glm::vec3 aux_position, aux, aux_normal;
00134
00135     //layout: [0]: near plane
00136     frustum_planes[0].normal = main_camera->get_camera_direction();
00137     frustum_planes[0].position = near_center;
00138
00139     // [1] far plane
00140     frustum_planes[1].normal = -main_camera->get_camera_direction();
00141     frustum_planes[1].position = far_center;
00142
00143     aux_position = near_center + main_camera->get_up_axis() * near_height;
00144     aux = aux_position - main_camera->get_camera_position();
00145     aux = glm::normalize(aux);
00146     aux_normal = glm::cross(aux, main_camera->get_right_axis());
00147
00148     // [2] top

```

```

00148 frustum_planes[2].normal = normalize(aux_normal);
00149 frustum_planes[2].position = aux_position;
00150
00151 aux_position = near_center - main_camera->get_up_axis() * near_height;
00152 aux = aux_position - main_camera->get_camera_position();
00153 aux = glm::normalize(aux);
00154 aux_normal = glm::cross(main_camera->get_right_axis(), aux);
00155
00156 // [3] bottom
00157 frustum_planes[3].normal = normalize(aux_normal);
00158 frustum_planes[3].position = aux_position;
00159
00160 aux_position = near_center - main_camera->get_right_axis() * near_width;
00161 aux = aux_position - main_camera->get_camera_position();
00162 aux = glm::normalize(aux);
00163 aux_normal = glm::cross(aux, main_camera->get_up_axis());
00164
00165 // [4]: left
00166 frustum_planes[4].normal = normalize(aux_normal);
00167 frustum_planes[4].position = aux_position;
00168
00169 aux_position = near_center + main_camera->get_right_axis() * near_width;
00170 aux = aux_position - main_camera->get_camera_position();
00171 aux = glm::normalize(aux);
00172 aux_normal = glm::cross(main_camera->get_up_axis(), aux);
00173
00174 // [5]: right
00175 frustum_planes[5].normal = normalize(aux_normal);
00176 frustum_planes[5].position = aux_position;
00177
00178 std::vector<glm::vec3> frustum_corners;
00179
00180 //frustum_corners.push_back(near_center - main_camera->get_right_axis() * near_width -
00181 // main_camera->get_up_axis() * near_height); // left bottom front
00182 //
00183 //frustum_corners.push_back(far_center - main_camera->get_right_axis() * far_width -
00184 // main_camera->get_up_axis() * far_height); // left bottom back
00185 //
00186 //frustum_corners.push_back(near_center - main_camera->get_right_axis() * near_width +
00187 // main_camera->get_up_axis() * near_height); // left top front
00188 //
00189 //frustum_corners.push_back(far_center - main_camera->get_right_axis() * far_width +
00190 // main_camera->get_up_axis() * far_height); // left top back
00191 //
00192 //frustum_corners.push_back(near_center + main_camera->get_right_axis() * near_width -
00193 // main_camera->get_up_axis() * near_height); // right bottom front
00194 //
00195 //frustum_corners.push_back(far_center + main_camera->get_right_axis() * far_width -
00196 // main_camera->get_up_axis() * far_height); // right bottom back
00197 //
00198 //frustum_corners.push_back(near_center + main_camera->get_right_axis() * near_width +
00199 // main_camera->get_up_axis() * near_height); // right top front
00200 //
00201 //frustum_corners.push_back(far_center + main_camera->get_right_axis() * far_width +
00202 // main_camera->get_up_axis() * far_height); // right top back
00203
00204 //init(frustum_corners);
00205 }
00206
00207 void ViewFrustumCulling::init(std::vector<glm::vec3> frustum_corner)
00208 {
00209
00210 //unsigned int num_corners = 8;
00211 m_drawCount = 36; //num_corners;
00212
00213 float vertices[] = {
00214
00215     frustum_corner[0].x,
00216     frustum_corner[0].y,
00217     frustum_corner[0].z, // left bottom front
00218     frustum_corner[1].x,
00219     frustum_corner[1].y,
00220     frustum_corner[1].z, // left bottom back
00221     frustum_corner[2].x,
00222     frustum_corner[2].y,
00223     frustum_corner[2].z, // left top front
00224     frustum_corner[3].x,
00225     frustum_corner[3].y,
00226     frustum_corner[3].z, // left top back
00227     frustum_corner[4].x,
00228     frustum_corner[4].y,
00229     frustum_corner[4].z, // right bottom front
00230     frustum_corner[5].x,
00231     frustum_corner[5].y,
00232     frustum_corner[5].z, // right bottom back
00233     frustum_corner[6].x,
00234     frustum_corner[6].y,

```

```
00235     frustum_corner[6].z, //right top front
00236     frustum_corner[7].x,
00237     frustum_corner[7].y,
00238     frustum_corner[7].z //right top back
00239 };
00240
00241 unsigned int indices[] = {
00242     // note that we start from 0!
00243     //left
00244     0,
00245     1,
00246     3,
00247     0,
00248     2,
00249     3,
00250     3,
00251     //right
00252     4,
00253     5,
00254     7,
00255     4,
00256     6,
00257     7,
00258     //top
00259     3,
00260     2,
00261     7,
00262     3,
00263     2,
00264     8,
00265     //bottom
00266     0,
00267     1,
00268     4,
00269     0,
00270     1,
00271     5,
00272     //back
00273     1,
00274     3,
00275     5,
00276     1,
00277     3,
00278     7,
00279     //front
00280     0,
00281     2,
00282     4,
00283     0,
00284     2,
00285     6,
00286 };
00287
00288 glGenVertexArrays(1, &VAO);
00289 glGenBuffers(1, &VBO);
00290 glGenBuffers(1, &EBO);
00291 // bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex
00292 // attributes(s).
00293 glBindVertexArray(VAO);
00294 glBindBuffer(GL_ARRAY_BUFFER, VBO);
00295 glBindBuffer(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
00296
00297 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
00298 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
00299
00300 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
00301 glEnableVertexAttribArray(0);
00302
00303 // note that this is allowed, the call to glVertexAttribPointer registered VBO as the vertex
00304 // attribute's bound vertex buffer object so afterwards we can safely unbind
00305 glBindBuffer(GL_ARRAY_BUFFER, 0);
00306
00307 // remember: do NOT unbind the EBO while a VAO is active as the bound element buffer object IS
00308 // stored in the VAO; keep the EBO bound.
00309 //glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00310
00311 // You can unbind the VAO afterwards so other VAO calls won't accidentally modify this VAO, but this
00312 // rarely happens. Modifying other
00313 // VAOs requires a call to glBindVertexArray anyways so we generally don't unbind VAOs (nor VBOs)
00314 // when it's not directly necessary.
00315 glBindVertexArray(0);
00316 }
```

```
00317     glDeleteVertexArrays(1, &VAO);
00318     glDeleteBuffers(1, &VBO);
00319     glDeleteBuffers(1, &EBO);
00320 }
```

7.175 C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.h File Reference

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <vector>
#include <memory>
#include "Camera.h"
#include "AABB.h"
#include "GlobalValues.h"
```

Include dependency graph for ViewFrustumCulling.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [ViewFrustumCulling](#)
- struct [ViewFrustumCulling::frustum_plane](#)

7.176 ViewFrustumCulling.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002
00003 #include <glad/glad.h>
00004 #include <GLFW/glfw3.h>
00005 #include <glm/glm.hpp>
00006 #include <glm/gtc/matrix_transform.hpp>
00007 #include <glm/gtc/type_ptr.hpp>
00008 #include <vector>
00009 #include <memory>
00010
00011 #include "Camera.h"
00012 #include "AABB.h"
00013 #include "GlobalValues.h"
00014
00015 class ViewFrustumCulling {
00016 public:
00017     ViewFrustumCulling();
00018
00019     bool is_inside(GLfloat ratio, std::shared_ptr<Camera> main_camera, std::shared_ptr<AABB>
00020                 bounding_box, glm::mat4 model);
00021
00022     void render_view_frustum();
00023
00024     ~ViewFrustumCulling();
00025
00026     private:
00027     //render view frustum
00028     unsigned int VBO, VAO, EBO;
00029     unsigned int m_drawCount;
00030
00031     //we get that as input
00032     GLfloat near_plane, far_plane, fov, ratio;
00033
00034     //calculate as soon as we become params
```

```

00035     GLfloat tan, near_height, near_width, far_height, far_width;
00036
00037     std::shared_ptr<Camera> main_camera;
00038
00039     glm::vec3 dir, near_center, far_center;
00040
00041     //all corners of the frustum
00042     //near plane
00043     glm::vec3 near_top_left, near_top_right, near_bottom_left, near_bottom_right;
00044     //far plane
00045     glm::vec3 far_top_left, far_top_right, far_bottom_left, far_bottom_right;
00046
00047     //planes in Hesse normal form
00048     //layout: [0]: near plane, [1] far plane, [2] front, [3] bottom, [4]: left, [5]: right
00049     struct frustum_plane {
00050
00051         glm::vec3 normal;
00052         glm::vec3 position;
00053     };
00054
00055     frustum_plane frustum_planes[NUM_FRUSTUM_PLANES];
00056
00057     void init(std::vector<glm::vec3> frustum_corner);
00058
00059
00060     bool corners_outside_plane(std::vector<glm::vec3> aabb_corners, frustum_plane plane, GLuint
00061     outcode_pattern);
00062
00063     GLfloat plane_point_distance(frustum_plane plane, glm::vec3 corner);
00064
00065     void update_frustum_param(GLfloat near_plane, GLfloat far_plane, GLfloat fov, GLfloat ratio,
00066     std::shared_ptr<Camera> main_camera);
00067 };

```

7.177 C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.cpp File Reference

```

#include "File.h"
#include <iostream>
#include <fstream>
Include dependency graph for File.cpp:

```

7.178 File.cpp

[Go to the documentation of this file.](#)

```

00001 #include "File.h"
00002 #include <iostream>
00003 #include <fstream>
00004
00005 File::File(const std::string& file_location) { this->file_location = file_location; }
00006
00007 std::string File::read()
00008 {
00009     std::string content;
00010     std::ifstream file_stream(file_location, std::ios::in);
00011
00012     if (!file_stream.is_open()) {
00013         printf("Failed to read %. File does not exist.", file_location.c_str());
00014         return "";
00015     }
00016
00017     std::string line = "";
00018     while (!file_stream.eof()) {
00019         std::getline(file_stream, line);
00020         content.append(line + "\n");
00021     }
00022
00023     file_stream.close();
00024     return content;
00025 }
00026
00027 File::~File() { }

```

7.179 C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.h File Reference

```
#include <string>
```

Include dependency graph for File.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [File](#)

7.180 File.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <string>
00003
00004 class File {
00005
00006     public:
00007     explicit File(const std::string& file_location);
00008     std::string read();
00009
00010     ~File();
00011
00012     private:
00013     std::string file_location;
00014 };
00015 };
```

7.181 C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumbers.cpp File Reference

```
#include "RandomNumbers.h"
#include "host_device_shared.h"
#include "bindings.h"
```

Include dependency graph for RandomNumbers.cpp:

7.182 RandomNumbers.cpp

[Go to the documentation of this file.](#)

```
00001 #include "RandomNumbers.h"
00002 #include "host_device_shared.h"
00003 #include "bindings.h"
00004
00005 RandomNumbers::RandomNumbers()
00006 {
00007     generate_random_numbers();
00008
00009     glGenTextures(1, &random_number_id);
00010     glBindTexture(GL_TEXTURE_2D, random_number_id);
00011     // i think we won't need nearest option; so stick to linear
00012     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00013     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00014     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00015     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00016     //assuming full HD will be maximum resolution
00017     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, MAX_RESOLUTION_X, MAX_RESOLUTION_Y, 0, GL_RGBA, GL_FLOAT,
random_number_data.get());
```

```

00018     glBindTexture(GL_TEXTURE_2D, 0);
00019 }
00020
00021 void RandomNumbers::read()
00022 {
00023     glActiveTexture(GL_TEXTURE0 + (GLenum)RANDOM_NUMBERS_SLOT);
00024     glBindTexture(GL_TEXTURE_2D, random_number_id);
00025 }
00026
00027 void RandomNumbers::generate_random_numbers()
00028 {
00029
00030     random_number_data = std::shared_ptr<GLfloat[]>(new GLfloat[MAX_RESOLUTION_X * MAX_RESOLUTION_Y *
4]);
00031     std::mt19937_64 gen64(25121995);
00032     std::uniform_real_distribution<float> dis(0, 1);
00033
00034     for (int i = 0; i < MAX_RESOLUTION_X; i++) {
00035
00036         for (int k = 0; k < MAX_RESOLUTION_Y; k++) {
00037
00038             const GLfloat random_offset[4] = { dis(gen64), dis(gen64), dis(gen64), dis(gen64) };
00039
00040             GLuint index = (MAX_RESOLUTION_Y * i + k) * 4;
00041
00042             *(random_number_data.get() + index) = random_offset[0];
00043             *(random_number_data.get() + index + 1) = random_offset[1];
00044             *(random_number_data.get() + index + 2) = random_offset[2];
00045             *(random_number_data.get() + index + 3) = random_offset[3];
00046         }
00047     }
00048 }
00049
00050 RandomNumbers::~RandomNumbers() { glDeleteTextures(1, &random_number_id); }

```

7.183 C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumbers.h File Reference

```
#include <memory>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <random>
#include "GlobalValues.h"
```

Include dependency graph for RandomNumbers.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class RandomNumbers

7.184 RandomNumbers.h

[Go to the documentation of this file.](#)

```

00001 #pragma once
00002 #include <memory>
00003 #include <glad/glad.h>
00004 #include <GLFW/glfw3.h>
00005 #include <random>
00006
00007 #include "GlobalValues.h"
00008
00009 class RandomNumbers {
00010
00011     public:
00012     RandomNumbers();
00013
00014     void read();

```

```

00015 ~RandomNumbers();
00016
00017 private:
00018 GLuint random_number_id;
00019 std::shared_ptr<GLfloat[]> random_number_data;
00020
00021 void generate_random_numbers();
00022
00023 };

```

7.185 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.cpp

File Reference

```
#include "Window.h"
#include <iostream>
Include dependency graph for Window.cpp:
```

7.186 Window.cpp

[Go to the documentation of this file.](#)

```

00001 #include "Window.h"
00002 #include <iostream>
00003
00004 Window::Window() : window_width(800), window_height(600), x_change(0.0f), y_change(0.0f)
00005 {
00006
00007     // all keys non-pressed in the beginning
00008     for (size_t i = 0; i < 1024; i++) {
00009         keys[i] = 0;
00010     }
00011
00012     initialize();
00013 }
00014
00015 //please use this constructor; never the standard
00016 Window::Window(GLint window_width, GLint window_height) :
00017
00018     window_width(window_width), window_height(window_height), x_change(0.0f), y_change(0.0f)
00019 {
00020
00021     // all keys non-pressed in the beginning
00022     for (size_t i = 0; i < 1024; i++) {
00023         keys[i] = 0;
00024     }
00025
00026     initialize();
00027 }
00028
00029 int Window::initialize()
00030 {
00031
00032     if (!glfwInit()) {
00033
00034         printf("GLFW Init failed!");
00035         glfwTerminate();
00036         return 1;
00037     }
00038
00039     // setup glfw window properties
00040
00041     //lets work with nothing older than version 3
00042     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
00043     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 6);
00044
00045     // core profile = no backward compatibility
00046     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00047
00048     // allow forward compatibility
00049     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00050
00051     //allow it to resize
00052     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00053

```

```
00054
00055 #ifdef NDEBUG
00056     glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, false);
00057 #else
00058     glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, true);
00059 #endif
00060
00061 //retrieve new window
00062 main_window = glfwCreateWindow(window_width, window_height, "\\__/_ Epic graphics from hell \\__/",
00063 NULL, NULL);
00064 if (!main_window) {
00065     printf("GLFW Window creation failed!");
00066     glfwTerminate();
00067     return 1;
00068 }
00069
00070 // get buffer size information
00072 glfwGetFramebufferSize(main_window, &window_buffer_width, &window_buffer_height);
00073
00074 // set context for GLEW to use
00075 glfwMakeContextCurrent(main_window);
00076
00077 if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)) {
00078     std::cout << "Failed to initialize OpenGL context" << std::endl;
00079     return -1;
00080 }
00081
00082 //disabling frame limits
00083 glfwSwapInterval(0);
00084
00085 //Handle key + mouse Input
00086 init_callbacks();
00087 glfwSetInputMode(main_window, GLFW_CURSOR, GLFW_CURSOR_NORMAL);
00088
00089 glEnable(GL_DEPTH_TEST);
00090
00091 // setup viewport size
00092 glViewport(0, 0, window_buffer_width, window_buffer_height);
00093
00094 glfwSetWindowUserPointer(main_window, this);
00095
00096 return 0;
00097 }
00098
00099 void Window::update_viewport()
00100 {
00101
00102     glfwGetFramebufferSize(main_window, &window_buffer_width, &window_buffer_height);
00103     // setup viewport size
00104     glViewport(0, 0, window_buffer_width, window_buffer_height);
00105 }
00106
00107 GLfloat Window::get_x_change()
00108 {
00109     GLfloat the_change = x_change;
00110     x_change = 0.0f;
00111     return the_change;
00112 }
00113
00114 GLfloat Window::get_y_change()
00115 {
00116     GLfloat the_change = y_change;
00117     y_change = 0.0f;
00118     return the_change;
00119 }
00120
00121 Window::~Window()
00122 {
00123     glfwDestroyWindow(main_window);
00124     glfwTerminate();
00125 }
00126
00127 void Window::init_callbacks()
00128 {
00129     //TODO: remember this section for our later game logic
00130     //for the space ship to fly around
00131     glfwSetKeyCallback(main_window, key_callback);
00132     glfwSetMouseButtonCallback(main_window, mouse_button_callback);
00133     glfwSetFramebufferSizeCallback(main_window, framebuffer_size_callback);
00134 }
00135
00136 void Window::framebuffer_size_callback(GLFWwindow* window, int width, int height) { }
00137
00138 void Window::key_callback(GLFWwindow* window, int key, int code, int action, int mode)
00139 {
```

```

00140     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00141
00142     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS) {
00143         glfwSetWindowShouldClose(window, GL_TRUE);
00144     }
00145
00146     if (key >= 0 && key < 1024) {
00147         if (action == GLFW_PRESS) {
00148             the_window->keys[key] = true;
00149
00150         } else if (action == GLFW_RELEASE) {
00151             the_window->keys[key] = false;
00152         }
00153     }
00154 }
00155
00156 void Window::mouse_callback(GLFWwindow* window, double x_pos, double y_pos)
00157 {
00158     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00159
00160     // need to handle first occurrence of a mouse moving event
00161     if (the_window->mouse_first_moved) {
00162         the_window->last_x = (float)x_pos;
00163         the_window->last_y = (float)y_pos;
00164         the_window->mouse_first_moved = false;
00165     }
00166
00167     the_window->x_change = (float)(x_pos - the_window->last_x);
00168     // take care of correct subtraction :
00169     the_window->y_change = (float)(the_window->last_y - y_pos);
00170
00171     //update params
00172     the_window->last_x = (float)x_pos;
00173     the_window->last_y = (float)y_pos;
00174 }
00175
00176 void Window::mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
00177 {
00178     Window* the_window = static_cast<Window*>(glfwGetWindowUserPointer(window));
00179
00180     if ((action == GLFW_PRESS) && (button == GLFW_MOUSE_BUTTON_RIGHT)) {
00181         glfwSetCursorPosCallback(window, mouse_callback);
00182     } else {
00183         the_window->mouse_first_moved = true;
00184         glfwSetCursorPosCallback(window, NULL);
00185     }
00186 }

```

7.187 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.h File Reference

```
#include <stdio.h>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
```

Include dependency graph for Window.h: This graph shows which files directly or indirectly include this file:

Data Structures

- class [Window](#)

7.188 Window.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <stdio.h>
00003
00004 #include <glad/glad.h>
00005 #include <GLFW/glfw3.h>
```

```
00006
00007 class Window {
00008     public:
00009     Window();
0010     Window(GLint window_width, GLint window_height);
0011
0012     bool get_should_close() { return glfwWindowShouldClose(main_window); }
0013     void swap_buffers() { glfwSwapBuffers(main_window); }
0014
0015     // init glfw and its context ...
0016     int initialize();
0017
0018     void update_viewport();
0019
0020     GLuint get_buffer_width() const { return window_buffer_width; }
0021     GLuint get_buffer_height() const { return window_buffer_height; }
0022
0023     GLfloat get_x_change();
0024     GLfloat get_y_change();
0025
0026     GLFWwindow* get_window() { return main_window; }
0027
0028     bool* get_keys() { return keys; }
0029
0030     ~Window();
0031
0032     private:
0033     GLFWwindow* main_window;
0034
0035     GLint window_width, window_height;
0036     // what key(-s) was/were pressed
0037     bool keys[1024];
0038     GLfloat last_x;
0039     GLfloat last_y;
0040     GLfloat x_change;
0041     GLfloat y_change;
0042     bool mouse_first_moved;
0043
0044     //buffers to store our window data to
0045     GLint window_buffer_width, window_buffer_height;
0046
0047     //we need to start our window callbacks for interaction
0048     void init_callbacks();
0049     static void framebuffer_size_callback(GLFWwindow* window, int width, int height);
0050     //after window resizing, update framebuffers
0051
0052     //need to be static ...
0053     static void key_callback(GLFWwindow* window, int key, int code, int action, int mode);
0054     static void mouse_callback(GLFWwindow* window, double x_pos, double y_pos);
0055     static void mouse_button_callback(GLFWwindow* window, int button, int action, int mods);
0056 };
```


Index

- ~AABB
 - AABB, 14
- ~Camera
 - Camera, 20
- ~CascadedShadowMap
 - CascadedShadowMap, 28
- ~ClampToEdgeMode
 - ClampToEdgeMode, 34
- ~Clouds
 - Clouds, 37
- ~ComputeShaderProgram
 - ComputeShaderProgram, 49
- ~DebugApp
 - DebugApp, 51
- ~DirectionalLight
 - DirectionalLight, 54
- ~DirectionalShadowMapPass
 - DirectionalShadowMapPass, 62
- ~File
 - File, 65
- ~GBuffer
 - GBuffer, 73
- ~GUI
 - GUI, 84
- ~GameObject
 - GameObject, 68
- ~GeometryPass
 - GeometryPass, 78
- ~GeometryPassShaderProgram
 - GeometryPassShaderProgram, 82
- ~Light
 - Light, 94
- ~LightingPass
 - LightingPass, 96
- ~LightingPassShaderProgram
 - LightingPassShaderProgram, 101
- ~LoadingScreen
 - LoadingScreen, 102
- ~Mesh
 - Mesh, 107
- ~MirroredRepeatMode
 - MirroredRepeatMode, 110
- ~Model
 - Model, 112
- ~Noise
 - Noise, 118
- ~ObjLoader
 - ObjLoader, 127
- ~ObjMaterial
 - ObjMaterial, 133
- ~OmniDirShadowMap
 - OmniDirShadowMap, 138
- ~OmniDirShadowShaderProgram
 - OmniDirShadowShaderProgram, 140
- ~OmniShadowMapPass
 - OmniShadowMapPass, 142
- ~PointLight
 - PointLight, 145
- ~Quad
 - Quad, 150
- ~RandomNumbers
 - RandomNumbers, 152
- ~RenderPassSceneDependend
 - RenderPassSceneDependend, 160
- ~Renderer
 - Renderer, 155
- ~RepeatMode
 - RepeatMode, 162
- ~Scene
 - Scene, 165
- ~ShaderIncludes
 - ShaderIncludes, 173
- ~ShaderProgram
 - ShaderProgram, 176
- ~ShadowMap
 - ShadowMap, 186
- ~SkyBox
 - SkyBox, 192
- ~Texture
 - Texture, 196
- ~ViewFrustumCulling
 - ViewFrustumCulling, 207
- ~Window
 - Window, 219
- AABB, 13
 - ~AABB, 14
 - AABB, 14
 - corners, 16
 - get_corners, 14
 - get_radius, 14
 - indices, 16
 - init, 14
 - maxX, 17
 - maxY, 17
 - maxZ, 17
 - mesh, 17
 - minX, 17
 - minY, 18

minZ, 18
 render, 16
 vertices, 18
aabb
 Clouds, 44
 Model, 114
activate
 ClampToEdgeMode, 35
 MirroredRepeatMode, 110
 RepeatMode, 162
 TextureWrappingMode, 201
add_game_object
 Scene, 165
add_shader
 ShaderProgram, 176
ambient
 ObjMaterial, 135
App.cpp
 main, 229
areErrorPrintAll
 DebugApp, 51
arePreError
 DebugApp, 52
available_shadow_map_resolutions
 GUI, 88
axis
 Rotation, 163
bind_ressources
 Model, 112
bind_textures_and_buffer
 Scene, 165
bindings.h
 D_LIGHT_SHADOW_TEXTURES_SLOT, 233
 GBUFFER_TEXTURES_SLOT, 233
 MODEL_TEXTURES_SLOT, 233
 NOISE_128D_IMAGE_SLOT, 233
 NOISE_128D_TEXTURES_SLOT, 233
 NOISE_32D_IMAGE_SLOT, 234
 NOISE_32D_TEXTURES_SLOT, 234
 NOISE_CELL_POSITIONS_SLOT, 234
 P_LIGHT_SHADOW_TEXTURES_SLOT, 234
 RANDOM_NUMBERS_SLOT, 234
 SKYBOX_TEXTURES_SLOT, 234
 STORAGE_BUFFER_MATERIAL_ID_BINDING,
 235
 UNIFORM_LIGHT_MATRICES_BINDING, 235
bit_depth
 Texture, 200

C:/Users/jonas/Desktop/GraphicEngine/Src/app/App.cpp, 229, 231
C:/Users/jonas/Desktop/GraphicEngine/Src/bindings.h, 232, 235
C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.cpp, 235, 236
C:/Users/jonas/Desktop/GraphicEngine/Src/camera/Camera.h, 237

C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProg
 238
C:/Users/jonas/Desktop/GraphicEngine/Src/compute/ComputeShaderProg
 238, 239
C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.cpp,
 239, 241
C:/Users/jonas/Desktop/GraphicEngine/Src/debug/DebugApp.h,
 243
C:/Users/jonas/Desktop/GraphicEngine/Src/GlobalValues.h,
 244, 245
C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.cpp,
 246
C:/Users/jonas/Desktop/GraphicEngine/Src/gui/GUI.h,
 249
C:/Users/jonas/Desktop/GraphicEngine/Src/host_device_shared.h,
 250, 252
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GBuffer.cpp
 252, 253
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GBuffer.h,
 254
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryP
 255
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryP
 257
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryP
 257, 258
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/GeometryP
 258
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPas
 258, 259
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPas
 261, 262
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPas
 262, 263
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/LightingPas
 263
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/deferred/RenderPass
 263, 264
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/Load
 264
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/loading_screen/Load
 265
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.cpp,
 265
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/Renderer.h,
 266, 267
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneD
 267, 268
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/RenderPassSceneD
 267, 268
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.cpp,
 268
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderIncludes.h,
 269
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderProgram.cpp,
 270
C:/Users/jonas/Desktop/GraphicEngine/Src/renderer/ShaderProgram.h,
 274

C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.cpp, 275
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/AABB.h, 277
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.h, 278
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.cpp, 279
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.h, 280
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.cpp, 284
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.h, 285
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/atmosphere/CloudEffect.cpp, 287, 292
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.h, 296
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.cpp, 296
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/GameObject.h, 297
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.h, 298
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.cpp, 299
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.h, 300
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.cpp, 303
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.h, 304
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/direct/BiasedLighting.h, 305
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.h, 305
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/Light.cpp, 306, 307
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.h, 307, 308
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.cpp, 308
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.h, 309
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.cpp, 310
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.h, 310, 311
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/light/pointLight.cpp, 311, 312
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.cpp, 312
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Mesh.h, 313, 314
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.cpp, 314
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Model.h, 315, 316
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.cpp, 316, 317
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjLoader.h, 319
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.cpp, 320
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ObjMaterial.h, 321
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.cpp, 322
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Quad.h, 322, 323
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Rotation.h, 323
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.cpp, 324
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Scene.h, 325, 326
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.h, 327
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/shadows/ShadowMap.cpp, 328
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.cpp, 328
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/sky_box/SkyBox.h, 331
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.h, 332
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/ClampToEdgeMode.cpp, 332
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/MirroredRepeatMode.h, 333
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/MirroredRepeatMode.cpp, 333
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/RepeatMode.h, 333, 334
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/RepeatMode.cpp, 333, 334
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Texture.cpp, 334, 335
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/Texture.h, 337
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/texture/TextureWrapping.h, 338
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/Vertex.h, 338, 339
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.cpp, 340
C:/Users/jonas/Desktop/GraphicEngine/Src/scene/ViewFrustumCulling.h, 344
C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.cpp, 345
C:/Users/jonas/Desktop/GraphicEngine/Src/util/File.h, 346
C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumbers.cpp, 346

C:/Users/jonas/Desktop/GraphicEngine/Src/util/RandomNumber.h
 get_shadow_height, 30
 get_shadow_width, 30
 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.cpp
 init, 30
 intensity, 32
 C:/Users/jonas/Desktop/GraphicEngine/Src/window/Window.h
 matrices_UBO, 32
 num_active_cascades, 33
 pcf_radius, 33
 read, 31
 set_intensity, 31
 set_pcf_radius, 31
 shadow_height, 33
 shadow_maps, 33
 shadow_width, 33
 write, 31
 write_light_matrices, 32
 Camera, 18
 ~Camera, 20
 Camera, 19, 20
 far_plane, 25
 fov, 25
 front, 25
 get_camera_direction, 20
 get_camera_position, 21
 get_far_plane, 21
 get_fov, 21
 get_near_plane, 21
 get_right_axis, 21
 get_up_axis, 22
 get_viewmatrix, 22
 get_yaw, 22
 key_control, 22
 mouse_control, 23
 movement_speed, 26
 near_plane, 26
 pitch, 26
 position, 26
 right, 26
 set_camera_position, 23
 set_far_plane, 24
 set_fov, 24
 set_near_plane, 24
 turn_speed, 27
 up, 27
 update, 24
 world_up, 27
 yaw, 27
 cascade_light_matrices
 DirectionalLight, 59
 cascade_slots
 DirectionalLight, 60
 cascaded_shadow_intensity
 GUI, 88
 CascadedShadowMap, 28
 ~CascadedShadowMap, 28
 CascadedShadowMap, 28
 FBO, 32
 get_id, 29
 get_intensity, 29
 get_num_active_cascades, 29
 get_pcf_radius, 29
 cell_data
 Noise, 124
 cell_ids
 Noise, 125
 cirrus_effect
 Clouds, 44
 ClampToEdgeMode, 34
 ~ClampToEdgeMode, 34
 activate, 35
 ClampToEdgeMode, 34
 clear_shader_program
 ShaderProgram, 177
 clear_texture_context
 Texture, 196
 cloud_cirrus_effect
 GUI, 88
 cloud_density
 GUI, 88
 cloud_mesh_offset
 GUI, 89
 cloud_mesh_scale
 GUI, 89
 cloud_movement_direction
 GUI, 89
 cloud_num_march_steps
 GUI, 89
 cloud_num_march_steps_to_light
 GUI, 89
 cloud_pillowness
 GUI, 90
 cloud_powder_effect
 GUI, 90
 cloud_scale
 GUI, 90
 cloud_speed
 GUI, 90
 Clouds, 35
 ~Clouds, 37
 aabb, 44
 cirrus_effect, 44
 Clouds, 36
 create_noise_textures, 37
 density, 44

get_cirrus_effect, 37
get_density, 38
get_mesh_scale, 38
get_model, 38
get_movement_direction, 38
get_movement_speed, 39
get_normal_model, 39
get_num_march_steps, 39
get_num_march_steps_to_light, 39
get_pillowness, 39
get_powder_effect, 40
get_radius, 40
get_scale, 40
get_shader_program, 40
maxX, 45
maxY, 45
maxZ, 45
minX, 45
minY, 45
minZ, 46
model, 46
movement_direction, 46
movement_speed, 46
noise, 46
num_march_steps, 47
num_march_steps_to_light, 47
pillowness, 47
powder_effect, 47
random_numbers, 47
read, 40
render, 41
scale, 48
scale_factor, 48
set_cirrus_effect, 41
set_density, 41
set_movement_direction, 42
set_movement_speed, 42
set_num_march_steps, 42
set_num_march_steps_to_light, 42
set_pillowness, 43
set_powder_effect, 43
set_scale, 43
set_translation, 44
shader_program, 48
translation, 48

clouds
 Scene, 169

CLOUDS_MATERIAL_ID
 host_device_shared.h, 250

COLOR
 Mesh, 106

color
 Light, 95
 Vertex, 204

compile_compute_shader_program
 ShaderProgram, 177

compile_program
 ShaderProgram, 178

compile_shader_program
 ShaderProgram, 178, 179

compute_location
 ShaderProgram, 184

ComputeShaderProgram, 49
 ~ComputeShaderProgram, 49
 ComputeShaderProgram, 49
 reload, 49

constant
 PointLight, 148

context_setup
 Scene, 170

corners
 AABB, 16

corners_outside_plane
 ViewFrustumCulling, 207

create
 GBuffer, 73

create_computer_shader_program_from_file
 ShaderProgram, 179

create_from_files
 ShaderProgram, 179, 180

create_noise_textures
 Clouds, 37

create_render_context
 Model, 112

create_res128_noise
 Noise, 118

create_res32_noise
 Noise, 119

create_shader_program
 DirectionalShadowMapPass, 62
 GeometryPass, 78
 LightingPass, 97
 LoadingScreen, 103
 OmniShadowMapPass, 142
 RenderPass, 159
 RenderPassSceneDependend, 160

create_shader_programs
 Noise, 119

current_offset
 LightingPass, 100

D_LIGHT_SHADOW_TEXTURES_SLOT
 bindings.h, 233

DebugApp, 50
 ~DebugApp, 51
 areErrorPrintAll, 51
 arePreError, 52
 DebugApp, 50

DebugApp.cpp
 glDebugOutput, 239

degrees
 Rotation, 163

delete_textures
 Noise, 120

density
 Clouds, 44

diffuse

ObjMaterial, 136
 dir
 ViewFrustumCulling, 212
 direcional_light_radiance
 GUI, 90
 direction
 DirectionalLight, 60
 directional_light_color
 GUI, 91
 directional_light_direction
 GUI, 91
 directional_shadow_map_pass
 Renderer, 157
 DirectionalLight, 53
 ~DirectionalLight, 54
 calc_cascaded_slots, 55
 calc_orthogonal_projections, 55
 calculate_light_transform, 56
 cascade_light_matrices, 59
 cascade_slots, 60
 direction, 60
 DirectionalLight, 54
 get_cascaded_light_matrices, 56
 get_cascaded_slots, 57
 get_color, 57
 get_direction, 57
 get_frustum_corners_world_space, 57
 get_light_view_matrix, 58
 get_radiance, 58
 get_shadow_map, 58
 set_color, 58
 set_direction, 59
 set_radiance, 59
 shadow_far_plane, 60
 shadow_map, 60
 shadow_near_plane, 60
 update_shadow_map, 59
 DirectionalShadowMapPass, 61
 ~DirectionalShadowMapPass, 62
 create_shader_program, 62
 DirectionalShadowMapPass, 61
 execute, 62
 set_game_object_uniforms, 63
 shader_program, 64
 dissolve
 ObjMaterial, 136
 draw_sky_box
 SkyBox, 192
 drawFrame
 Renderer, 156
 EBO
 ViewFrustumCulling, 212
 emission
 ObjMaterial, 136
 execute
 DirectionalShadowMapPass, 62
 GeometryPass, 78
 LightingPass, 97
 OmniShadowMapPass, 142
 exponent
 PointLight, 148
 fade
 Perlin.h, 287
 far_bottom_left
 ViewFrustumCulling, 213
 far_bottom_right
 ViewFrustumCulling, 213
 far_center
 ViewFrustumCulling, 213
 far_height
 ViewFrustumCulling, 213
 far_plane
 Camera, 25
 PointLight, 148
 ViewFrustumCulling, 213
 far_top_left
 ViewFrustumCulling, 214
 far_top_right
 ViewFrustumCulling, 214
 far_width
 ViewFrustumCulling, 214
 FBO
 CascadedShadowMap, 32
 ShadowMap, 188
 File, 64
 ~File, 65
 File, 64
 file_location, 65
 read, 65
 file_location
 File, 65
 Texture, 200
 file_locations_relative
 ShaderIncludes, 173
 fov
 Camera, 25
 ViewFrustumCulling, 214
 fragment_location
 ShaderProgram, 184
 framebuffer_size_callback
 Window, 220
 front
 Camera, 25
 frustum_planes
 ViewFrustumCulling, 214
 g_albedo
 GBuffer, 75
 g_buffer
 GBuffer, 75
 g_buffer_attachment
 GBuffer, 75
 G_BUFFER_SIZE
 GlobalValues.h, 245
 g_depth
 GBuffer, 75

g_material_id
 GBuffer, 76
g_normal
 GBuffer, 76
g_position
 GBuffer, 76
game_objects
 Scene, 170
GameObject, 67
 ~GameObject, 68
 GameObject, 67, 68
 get_aabb, 68
 get_model, 68
 get_normal_world_trafo, 69
 get_world_trafo, 69
 init, 69
 model, 71
 render, 69
 rot, 71
 rotate, 70
 scale, 70
 scale_factor, 71
 translate, 70
 translation, 71
GBuffer, 72
 ~GBuffer, 73
 create, 73
 g_albedo, 75
 g_buffer, 75
 g_buffer_attachment, 75
 g_depth, 75
 g_material_id, 76
 g_normal, 76
 g_position, 76
 GBuffer, 72
 get_id, 74
 read, 74
 update_window_params, 74
 window_height, 76
 window_width, 76
gbuffer
 Renderer, 157
GBUFFER_TEXTURES_SLOT
 bindings.h, 233
generate_cells
 Noise, 120
generate_num_cells_textures
 Noise, 121
generate_random_numbers
 RandomNumbers, 153
generate_res128_noise_texture
 Noise, 121
generate_res32_noise_texture
 Noise, 122
generate_textures
 Noise, 122
geometry_location
 ShaderProgram, 184
geometry_pass
 Renderer, 157
GeometryPass, 77
 ~GeometryPass, 78
 create_shader_program, 78
 execute, 78
 GeometryPass, 77
 set_game_object_uniforms, 80
 shader_program, 81
 skybox, 81
GeometryPassShaderProgram, 81
 ~GeometryPassShaderProgram, 82
 GeometryPassShaderProgram, 82
 get_program_id, 82
get_aabb
 GameObject, 68
 Model, 113
get_ambient
 ObjMaterial, 133
get_base_dir
 ObjLoader, 127
get_buffer_height
 Window, 220
get_buffer_width
 Window, 220
get_camera_direction
 Camera, 20
get_camera_position
 Camera, 21
get_cascaded_light_matrices
 DirectionalLight, 56
get_cascaded_slots
 DirectionalLight, 57
get_cirrus_effect
 Clouds, 37
get_clouds
 Scene, 166
get_color
 DirectionalLight, 57
 Light, 94
 Vertex, 203
get_constant_factor
 PointLight, 146
get_context_setup
 Scene, 166
get_corners
 AABB, 14
get_density
 Clouds, 38
get_diffuse
 ObjMaterial, 133
get_direction
 DirectionalLight, 57
get_dissolve
 ObjMaterial, 134
get_emission
 ObjMaterial, 134
get_exponent_factor

PointLight, 146
 get_far_plane
 Camera, 21
 PointLight, 146
 get_filename
 Texture, 196
 get_fov
 Camera, 21
 get_frustum_corners_world_space
 DirectionalLight, 57
 get_game_objects
 Scene, 166
 get_id
 CascadedShadowMap, 29
 GBuffer, 74
 ShaderProgram, 180
 ShadowMap, 186
 Texture, 197
 get_illum
 ObjMaterial, 134
 get_intensity
 CascadedShadowMap, 29
 get_ior
 ObjMaterial, 134
 get_keys
 Window, 220
 get_light_view_matrix
 DirectionalLight, 58
 get_linear_factor
 PointLight, 147
 get_materials
 Model, 113
 Scene, 166
 get_mesh_scale
 Clouds, 38
 get_model
 Clouds, 38
 GameObject, 68
 get_movement_direction
 Clouds, 38
 get_movement_speed
 Clouds, 39
 get_near_plane
 Camera, 21
 get_normal
 Vertex, 203
 get_normal_model
 Clouds, 39
 get_normal_world_trafo
 GameObject, 69
 get_num_active_cascades
 CascadedShadowMap, 29
 get_num_march_steps
 Clouds, 39
 get_num_march_steps_to_light
 Clouds, 39
 get_omni_shadow_map
 PointLight, 147
 get_pcf_radius
 CascadedShadowMap, 29
 get_pillowness
 Clouds, 39
 get_point_light_count
 Scene, 166
 get_point_lights
 Scene, 167
 get_position
 PointLight, 147
 Vertex, 203
 get_powder_effect
 Clouds, 40
 get_program_id
 GeometryPassShaderProgram, 82
 get_progress
 Scene, 167
 get_radiance
 DirectionalLight, 58
 Light, 94
 get_radius
 AABB, 14
 Clouds, 40
 get_right_axis
 Camera, 21
 get_scale
 Clouds, 40
 get_shader_program
 Clouds, 40
 get_shadow_height
 CascadedShadowMap, 30
 ShadowMap, 187
 get_shadow_map
 DirectionalLight, 58
 get_shadow_width
 CascadedShadowMap, 30
 ShadowMap, 187
 get_shininess
 ObjMaterial, 134
 get_should_close
 Window, 220
 get_specular
 ObjMaterial, 135
 get_sun
 Scene, 167
 get_tex_coors
 Vertex, 204
 get_texture_count
 Model, 113
 Scene, 167
 get_textureID
 ObjMaterial, 135
 get_transmittance
 ObjMaterial, 135
 get_up_axis
 Camera, 22
 get_viewmatrix
 Camera, 22

get_window
 Window, 221
get_world_trafo
 GameObject, 69
get_x_change
 Window, 221
get_y_change
 Window, 221
get_yaw
 Camera, 22
getIndices
 Mesh, 107
getPermutationVector
 Perlin.h, 287
getUniformLocation
 ShaderProgram, 181
getVertices
 Mesh, 108
glDebugOutput
 DebugApp.cpp, 239
GlobalValues.h
 G_BUFFER_SIZE, 245
 MAX_RESOLUTION_X, 244
 MAX_RESOLUTION_Y, 244
 NUM_CLOUDS, 245
 NUM_FRUSTUM_PLANES, 245
 NUM_MIN_CASCADES, 244
grad
 Perlin.h, 290
GUI, 82
 ~GUI, 84
 available_shadow_map_resolutions, 88
 cascaded_shadow_intensity, 88
 cloud_cirrus_effect, 88
 cloud_density, 88
 cloud_mesh_offset, 89
 cloud_mesh_scale, 89
 cloud_movement_direction, 89
 cloud_num_march_steps, 89
 cloud_num_march_steps_to_light, 89
 cloud_pillowness, 90
 cloud_powder_effect, 90
 cloud_scale, 90
 cloud_speed, 90
 direccional_light_radiance, 90
 directional_light_color, 91
 directional_light_direction, 91
 GUI, 83
 init, 85
 logo_tex, 91
 num_shadow_cascades, 91
 pcf_radius, 91
 render, 85
 shadow_map_res_index, 92
 shadow_resolution_changed, 92
 update_user_input, 87
height
 Texture, 200
host_device_shared.h
 CLOUDS_MATERIAL_ID, 250
 MAX_MATERIALS, 251
 MAX_POINT_LIGHTS, 251
 MAX_TEXTURE_COUNT, 251
 NUM_CASCADES, 251
 NUM_CELL_POSITIONS, 251
 PI, 252
 SKYBOX_MATERIAL_ID, 252
illum
 ObjMaterial, 136
includeNames
 ShaderIncludes, 174
indices
 AABB, 16
 Mesh, 108
 Model, 114
init
 AABB, 14
 CascadedShadowMap, 30
 GameObject, 69
 GUI, 85
 LoadingScreen, 103
 OmniDirShadowMap, 138
 ShadowMap, 187
 ViewFrustumCulling, 208
init_callbacks
 Window, 221
initialize
 Window, 222
intensity
 CascadedShadowMap, 32
ior
 ObjMaterial, 136
is_inside
 ViewFrustumCulling, 209
is_loaded
 Scene, 167
key_callback
 Window, 223
key_control
 Camera, 22
keys
 Window, 225
last_x
 Window, 225
last_y
 Window, 225
lerp
 Perlin.h, 291
Light, 93
 ~Light, 94
 color, 95
 get_color, 94
 get_radiance, 94
 Light, 94

light_proj, 95
 radiance, 95
 light_proj
 Light, 95
 lighting_pass
 Renderer, 157
 LightingPass, 96
 ~LightingPass, 96
 create_shader_program, 97
 current_offset, 100
 execute, 97
 LightingPass, 96
 quad, 100
 set_uniforms, 97
 shader_program, 100
 LightingPassShaderProgram, 101
 ~LightingPassShaderProgram, 101
 LightingPassShaderProgram, 101
 linear
 PointLight, 148
 load
 ObjLoader, 128
 load_model_in_ram
 Model, 113
 load_models
 Scene, 168
 load_SRGB_texture_with_alpha_channel
 Texture, 197
 load_SRGB_texture_without_alpha_channel
 Texture, 197
 load_texture_with_alpha_channel
 Texture, 198
 load_texture_without_alpha_channel
 Texture, 198
 loaded_scene
 Scene, 170
 loader
 Model, 115
 loading_screen_quad
 LoadingScreen, 104
 loading_screen_shader_program
 LoadingScreen, 104
 LoadingScreen, 102
 ~LoadingScreen, 102
 create_shader_program, 103
 init, 103
 loading_screen_quad, 104
 loading_screen_shader_program, 104
 loading_screen_tex, 104
 LoadingScreen, 102
 logo_tex, 104
 render, 103
 logo_tex
 GUI, 91
 LoadingScreen, 104
 m_drawCount
 Mesh, 108
 ViewFrustumCulling, 215
 m_ibos
 Mesh, 109
 m_vabs
 Mesh, 109
 m_vaos
 Mesh, 109
 main
 App.cpp, 229
 main_camera
 Scene, 170
 ViewFrustumCulling, 215
 main_window
 Scene, 170
 Window, 225
 materialIndex
 Model, 115
 materials
 Model, 115
 matrices_UBO
 CascadedShadowMap, 32
 MAX_MATERIALS
 host_device_shared.h, 251
 MAX_POINT_LIGHTS
 host_device_shared.h, 251
 MAX_RESOLUTION_X
 GlobalValues.h, 244
 MAX_RESOLUTION_Y
 GlobalValues.h, 244
 MAX_TEXTURE_COUNT
 host_device_shared.h, 251
 maxX
 AABB, 17
 Clouds, 45
 ObjLoader, 130
 maxY
 AABB, 17
 Clouds, 45
 ObjLoader, 130
 maxZ
 AABB, 17
 Clouds, 45
 ObjLoader, 130
 Mesh, 105
 ~Mesh, 107
 COLOR, 106
 getIndices, 107
 getVertices, 108
 indices, 108
 m_drawCount, 108
 m_ibos, 109
 m_vabs, 109
 m_vaos, 109
 Mesh, 106, 107
 NORMAL, 106
 NUM_BUFFERS, 106
 POSITION, 106

POSITION_VB, 106
render, 108
TEXTURECOORD, 106
vertices, 109
mesh
 AABB, 17
 Model, 115
minX
 AABB, 17
 Clouds, 45
 ObjLoader, 131
minY
 AABB, 18
 Clouds, 45
 ObjLoader, 131
minZ
 AABB, 18
 Clouds, 46
 ObjLoader, 131
MirroredRepeatMode, 110
 ~MirroredRepeatMode, 110
 activate, 110
 MirroredRepeatMode, 110
Model, 111
 ~Model, 112
 aabb, 114
 bind_resources, 112
 create_render_context, 112
 get_aabb, 113
 get_materials, 113
 get_texture_count, 113
 indices, 114
 load_model_in_ram, 113
 loader, 115
 materialIndex, 115
 materials, 115
 mesh, 115
 Model, 112
 render, 114
 ssbo, 115
 texture_list, 116
 textures, 116
 unbind_resources, 114
 vertices, 116
model
 Clouds, 46
 GameObject, 71
MODEL_TEXTURES_SLOT
 bindings.h, 233
mouse_button_callback
 Window, 223
mouse_callback
 Window, 224
mouse_control
 Camera, 23
mouse_first_moved
 Window, 226
movement_direction

Clouds, 46
movement_speed
 Camera, 26
 Clouds, 46
 SkyBox, 193
mx_isLoaded
 Scene, 171
mx_progress
 Scene, 171
mx_space_ship
 Scene, 171
near_bottom_left
 ViewFrustumCulling, 215
near_bottom_right
 ViewFrustumCulling, 215
near_center
 ViewFrustumCulling, 215
near_height
 ViewFrustumCulling, 216
near_plane
 Camera, 26
 ViewFrustumCulling, 216
near_top_left
 ViewFrustumCulling, 216
near_top_right
 ViewFrustumCulling, 216
near_width
 ViewFrustumCulling, 216
Noise, 116
 ~Noise, 118
 cell_data, 124
 cell_ids, 125
 create_res128_noise, 118
 create_res32_noise, 119
 create_shader_programs, 119
 delete_textures, 120
 generate_cells, 120
 generate_num_cells_textures, 121
 generate_res128_noise_texture, 121
 generate_res32_noise_texture, 122
 generate_textures, 122
 Noise, 117
 num_cells_per_axis, 125
 print_comp_shader_capabilities, 123
 read_res128_noise, 123
 read_res32_noise, 123
 set_num_cells, 124
 texture_1_id, 125
 texture_1_shader_program, 125
 texture_2_id, 125
 texture_2_shader_program, 126
 texture_dim_1, 126
 texture_dim_2, 126
 update, 124
noise
 Clouds, 46
NOISE_128D_IMAGE_SLOT
 bindings.h, 233

NOISE_128D_TEXTURES_SLOT
 bindings.h, 233

NOISE_32D_IMAGE_SLOT
 bindings.h, 234

NOISE_32D_TEXTURES_SLOT
 bindings.h, 234

NOISE_CELL_POSITIONS_SLOT
 bindings.h, 234

NORMAL
 Mesh, 106

normal
 Vertex, 204
 ViewFrustumCulling::frustum_plane, 66

num_active_cascades
 CascadedShadowMap, 33

NUM_BUFFERS
 Mesh, 106

NUM_CASCADES
 host_device_shared.h, 251

NUM_CELL_POSITIONS
 host_device_shared.h, 251

num_cells_per_axis
 Noise, 125

NUM_CLOUDS
 GlobalValues.h, 245

NUM_FRUSTUM_PLANES
 GlobalValues.h, 245

num_march_steps
 Clouds, 47

num_march_steps_to_light
 Clouds, 47

NUM_MIN_CASCADES
 GlobalValues.h, 244

num_shadow_cascades
 GUI, 91

object_is_visible
 Scene, 168

ObjLoader, 126
 ~ObjLoader, 127
 get_base_dir, 127
 load, 128
 maxX, 130
 maxY, 130
 maxZ, 130
 minX, 131
 minY, 131
 minZ, 131
 ObjLoader, 127

ObjLoader.cpp
 TINYOBJLOADER_IMPLEMENTATION, 317

ObjMaterial, 131
 ~ObjMaterial, 133
 ambient, 135
 diffuse, 136
 dissolve, 136
 emission, 136
 get_ambient, 133
 get_diffuse, 133

get_dissolve, 134
 get_emission, 134
 get_illum, 134
 get_ior, 134
 get_shininess, 134
 get_specular, 135
 get_textureID, 135
 get_transmittance, 135
 illum, 136
 ior, 136
 ObjMaterial, 132
 shininess, 137
 specular, 137
 textureID, 137
 transmittance, 137

omni_dir_shadow_map
 PointLight, 148

omni_shadow_map_pass
 Renderer, 158

OmniDirShadowMap, 138
 ~OmniDirShadowMap, 138
 init, 138
 OmniDirShadowMap, 138
 read, 139
 write, 139

OmniDirShadowShaderProgram, 140
 ~OmniDirShadowShaderProgram, 140
 OmniDirShadowShaderProgram, 140
 reload, 141

OmniShadowMapPass, 141
 ~OmniShadowMapPass, 142
 create_shader_program, 142
 execute, 142
 OmniShadowMapPass, 142
 set_game_object_uniforms, 143
 shader_program, 143

operator()
 std::hash<Vertex>, 93

operator==
 Vertex, 204

P_LIGHT_SHADOW_TEXTURES_SLOT
 bindings.h, 234

pcf_radius
 CascadedShadowMap, 33
 GUI, 91

Perlin.h
 fade, 287
 getPermutationVector, 287
 grad, 290
 lerp, 291
 perlin_noise, 291

perlin_noise
 Perlin.h, 291

PI
 host_device_shared.h, 252

pillowness
 Clouds, 47

pitch

Camera, 26
plane_point_distance
 ViewFrustumCulling, 210
point_lights
 Scene, 171
PointLight, 144
 ~PointLight, 145
 calculate_light_transform, 146
 constant, 148
 exponent, 148
 far_plane, 148
 get_constant_factor, 146
 get_exponent_factor, 146
 get_far_plane, 146
 get_linear_factor, 147
 get_omni_shadow_map, 147
 get_position, 147
 linear, 148
 omni_dir_shadow_map, 148
 PointLight, 145
 position, 148
 set_position, 147
POSITION
 Mesh, 106
position
 Camera, 26
 PointLight, 148
 Vertex, 205
 ViewFrustumCulling::frustum_plane, 66
POSITION_VB
 Mesh, 106
powder_effect
 Clouds, 47
print_comp_shader_capabilities
 Noise, 123
program_id
 ShaderProgram, 184
progress
 Scene, 171
q_vao
 Quad, 150
q_vbo
 Quad, 151
Quad, 149
 ~Quad, 150
 q_vao, 150
 q_vbo, 151
 Quad, 149
 render, 150
 vertices, 151
quad
 LightingPass, 100
radiance
 Light, 95
random_number_data
 RandomNumbers, 154
random_number_id
 RandomNumbers, 154
random_numbers
 Clouds, 47
RANDOM_NUMBERS_SLOT
 bindings.h, 234
RandomNumbers, 152
 ~RandomNumbers, 152
 generate_random_numbers, 153
 random_number_data, 154
 random_number_id, 154
 RandomNumbers, 152
 read, 153
ratio
 ViewFrustumCulling, 217
read
 CascadedShadowMap, 31
 Clouds, 40
 File, 65
 GBuffer, 74
 OmniDirShadowMap, 139
 RandomNumbers, 153
 ShadowMap, 188
read_res128_noise
 Noise, 123
read_res32_noise
 Noise, 123
reload
 ComputeShaderProgram, 49
 OmniDirShadowShaderProgram, 141
 SkyBox, 193
reload_shader_programs
 Renderer, 156
render
 AABB, 16
 Clouds, 41
 GameObject, 69
 GUI, 85
 LoadingScreen, 103
 Mesh, 108
 Model, 114
 Quad, 150
render_passes
 Renderer, 158
render_view_frustum
 ViewFrustumCulling, 210
Renderer, 154
 ~Renderer, 155
 directional_shadow_map_pass, 157
 drawFrame, 156
 gbuffer, 157
 geometry_pass, 157
 lighting_pass, 157
 omni_shadow_map_pass, 158
 reload_shader_programs, 156
 render_passes, 158
 Renderer, 155
 shader_includes, 158
 update_window_params, 156

window_height, 158
 window_width, 158
 RenderPass, 159
 create_shader_program, 159
 RenderPassSceneDependend, 160
 ~RenderPassSceneDependend, 160
 create_shader_program, 160
 RenderPassSceneDependend, 160
 set_game_object_uniforms, 161
 RepeatMode, 161
 ~RepeatMode, 162
 activate, 162
 RepeatMode, 161
 right
 Camera, 26
 rot
 GameObject, 71
 rotate
 GameObject, 70
 Rotation, 162
 axis, 163
 degrees, 163
 scale
 Clouds, 48
 GameObject, 70
 scale_factor
 Clouds, 48
 GameObject, 71
 Scene, 163
 ~Scene, 165
 add_game_object, 165
 bind_textures_and_buffer, 165
 clouds, 169
 context_setup, 170
 game_objects, 170
 get_clouds, 166
 get_context_setup, 166
 get_game_objects, 166
 get_materials, 166
 get_point_light_count, 166
 get_point_lights, 167
 get_progress, 167
 get_sun, 167
 get_texture_count, 167
 is_loaded, 167
 load_models, 168
 loaded_scene, 170
 main_camera, 170
 main_window, 170
 mx_isLoaded, 171
 mx_progress, 171
 mx_space_ship, 171
 object_is_visible, 168
 point_lights, 171
 progress, 171
 Scene, 164
 set_context_setup, 168
 setup_game_object_context, 169
 spwan, 169
 sun, 172
 unbind_textures_and_buffer, 169
 view_frustum_culling, 172
 set_camera_position
 Camera, 23
 set_cirrus_effect
 Clouds, 41
 set_color
 DirectionalLight, 58
 set_context_setup
 Scene, 168
 set_density
 Clouds, 41
 set_direction
 DirectionalLight, 59
 set_far_plane
 Camera, 24
 set_fov
 Camera, 24
 set_game_object_uniforms
 DirectionalShadowMapPass, 63
 GeometryPass, 80
 OmniShadowMapPass, 143
 RenderPassSceneDependend, 161
 set_intensity
 CascadedShadowMap, 31
 set_movement_direction
 Clouds, 42
 set_movement_speed
 Clouds, 42
 set_near_plane
 Camera, 24
 set_num_cells
 Noise, 124
 set_num_march_steps
 Clouds, 42
 set_num_march_steps_to_light
 Clouds, 42
 set_pcf_radius
 CascadedShadowMap, 31
 set_pillowness
 Clouds, 43
 set_position
 PointLight, 147
 set_powder_effect
 Clouds, 43
 set_radiance
 DirectionalLight, 59
 set_scale
 Clouds, 43
 set_translation
 Clouds, 44
 set_uniforms
 LightingPass, 97
 setUniformBlockBinding
 ShaderProgram, 181
 setUniformFloat

ShaderProgram, 181
setUniformInt
 ShaderProgram, 182
setUniformMatrix4fv
 ShaderProgram, 182
setUniformVec3
 ShaderProgram, 182
setup_game_object_context
 Scene, 169
shader_base_dir
 ShaderProgram, 185
shader_includes
 Renderer, 158
shader_playback_time
 SkyBox, 193
shader_program
 Clouds, 48
 DirectionalShadowMapPass, 64
 GeometryPass, 81
 LightingPass, 100
 OmniShadowMapPass, 143
 SkyBox, 193
ShaderIncludes, 172
 ~ShaderIncludes, 173
 file_locations_relative, 173
 includeNames, 174
 ShaderIncludes, 173
ShaderProgram, 175
 ~ShaderProgram, 176
 add_shader, 176
 clear_shader_program, 177
 compile_compute_shader_program, 177
 compile_program, 178
 compile_shader_program, 178, 179
 compute_location, 184
 create_computer_shader_program_from_file, 179
 create_from_files, 179, 180
 fragment_location, 184
 geometry_location, 184
 get_id, 180
 getUniformLocation, 181
 program_id, 184
 setUniformBlockBinding, 181
 setUniformFloat, 181
 setUniformInt, 182
 setUniformMatrix4fv, 182
 setUniformVec3, 182
 shader_base_dir, 185
 ShaderProgram, 176
 use_shader_program, 183
 validate_program, 183
 validateUniformLocation, 183
 vertex_location, 185
shadow_far_plane
 DirectionalLight, 60
shadow_height
 CascadedShadowMap, 33
 ShadowMap, 189
shadow_map
 DirectionalLight, 60
 ShadowMap, 189
shadow_map_res_index
 GUI, 92
shadow_maps
 CascadedShadowMap, 33
shadow_near_plane
 DirectionalLight, 60
shadow_resolution_changed
 GUI, 92
shadow_width
 CascadedShadowMap, 33
 ShadowMap, 189
ShadowMap, 185
 ~ShadowMap, 186
 FBO, 188
 get_id, 186
 get_shadow_height, 187
 get_shadow_width, 187
 init, 187
 read, 188
 shadow_height, 189
 shadow_map, 189
 shadow_width, 189
 ShadowMap, 186
 write, 188
shininess
 ObjMaterial, 137
sky_mesh
 SkyBox, 193
SkyBox, 189
 ~SkyBox, 192
 draw_sky_box, 192
 movement_speed, 193
 reload, 193
 shader_playback_time, 193
 shader_program, 193
 sky_mesh, 193
 SkyBox, 190
 texture_id, 194
 uniform_projection, 194
 uniform_view, 194
skybox
 GeometryPass, 81
SKYBOX_MATERIAL_ID
 host_device_shared.h, 252
SKYBOX_TEXTURES_SLOT
 bindings.h, 234
specular
 ObjMaterial, 137
spwan
 Scene, 169
ssbo
 Model, 115
STB_IMAGE_IMPLEMENTATION
 Texture.cpp, 335
std, 11

std::hash< Vertex >, 92
 operator(), 93
 STORAGE_BUFFER_MATERIAL_ID_BINDING
 bindings.h, 235
 sun
 Scene, 172
 swap_buffers
 Window, 224

 tan
 ViewFrustumCulling, 217
 Texture, 194
 ~Texture, 196
 bit_depth, 200
 clear_texture_context, 196
 file_location, 200
 get_filename, 196
 get_id, 197
 height, 200
 load_SRGB_texture_with_alpha_channel, 197
 load_SRGB_texture_without_alpha_channel, 197
 load_texture_with_alpha_channel, 198
 load_texture_without_alpha_channel, 198
 Texture, 195
 textureID, 200
 unbind_texture, 199
 use_texture, 199
 width, 200
 wrapping_mode, 201
 Texture.cpp
 STB_IMAGE_IMPLEMENTATION, 335
 texture_1_id
 Noise, 125
 texture_1_shader_program
 Noise, 125
 texture_2_id
 Noise, 125
 texture_2_shader_program
 Noise, 126
 texture_coords
 Vertex, 205
 texture_dim_1
 Noise, 126
 texture_dim_2
 Noise, 126
 texture_id
 SkyBox, 194
 texture_list
 Model, 116
 TEXTURECOORD
 Mesh, 106
 textureID
 ObjMaterial, 137
 Texture, 200
 textures
 Model, 116
 TextureWrappingMode, 201
 activate, 201
 TINYOBJLOADER_IMPLEMENTATION
 ObjLoader.cpp, 317
 translate
 GameObject, 70
 translation
 Clouds, 48
 GameObject, 71
 transmittance
 ObjMaterial, 137
 turn_speed
 Camera, 27

 unbind_resources
 Model, 114
 unbind_texture
 Texture, 199
 unbind_textures_and_buffer
 Scene, 169
 UNIFORM_LIGHT_MATRICES_BINDING
 bindings.h, 235
 uniform_projection
 SkyBox, 194
 uniform_view
 SkyBox, 194
 up
 Camera, 27
 update
 Camera, 24
 Noise, 124
 update_frustum_param
 ViewFrustumCulling, 211
 update_shadow_map
 DirectionalLight, 59
 update_user_input
 GUI, 87
 update_viewport
 Window, 224
 update_window_params
 GBuffer, 74
 Renderer, 156
 use_shader_program
 ShaderProgram, 183
 use_texture
 Texture, 199

 validate_program
 ShaderProgram, 183
 validateUniformLocation
 ShaderProgram, 183
 VAO
 ViewFrustumCulling, 217
 VBO
 ViewFrustumCulling, 217
 Vertex, 202
 color, 204
 get_color, 203
 get_normal, 203
 get_position, 203
 get_tex_coors, 204
 normal, 204

operator==, 204
position, 205
texture_coords, 205
Vertex, 202, 203
vertex_location
 ShaderProgram, 185
vertices
 AABB, 18
 Mesh, 109
 Model, 116
 Quad, 151
view_frustum_culling
 Scene, 172
ViewFrustumCulling, 205
 ~ViewFrustumCulling, 207
 corners_outside_plane, 207
 dir, 212
 EBO, 212
 far_bottom_left, 213
 far_bottom_right, 213
 far_center, 213
 far_height, 213
 far_plane, 213
 far_top_left, 214
 far_top_right, 214
 far_width, 214
 fov, 214
 frustum_planes, 214
 init, 208
 is_inside, 209
 m_drawCount, 215
 main_camera, 215
 near_bottom_left, 215
 near_bottom_right, 215
 near_center, 215
 near_height, 216
 near_plane, 216
 near_top_left, 216
 near_top_right, 216
 near_width, 216
 plane_point_distance, 210
 ratio, 217
 render_view_frustum, 210
 tan, 217
 update_frustum_param, 211
VAO, 217
VBO, 217
ViewFrustumCulling, 206
ViewFrustumCulling::frustum_plane, 66
 normal, 66
 position, 66
width
 Texture, 200
Window, 218
 ~Window, 219
 framebuffer_size_callback, 220
 get_buffer_height, 220
 get_buffer_width, 220
 get_keys, 220
 get_should_close, 220
 get_window, 221
 get_x_change, 221
 get_y_change, 221
 init_callbacks, 221
 initialize, 222
 key_callback, 223
 keys, 225
 last_x, 225
 last_y, 225
 main_window, 225
 mouse_button_callback, 223
 mouse_callback, 224
 mouse_first_moved, 226
 swap_buffers, 224
 update_viewport, 224
 Window, 219
 window_buffer_height, 226
 window_buffer_width, 226
 window_height, 226
 window_width, 226
 x_change, 227
 y_change, 227
 window_buffer_height
 Window, 226
 window_buffer_width
 Window, 226
 window_height
 GBuffer, 76
 Renderer, 158
 Window, 226
 window_width
 GBuffer, 76
 Renderer, 158
 Window, 226
 world_up
 Camera, 27
wrapping_mode
 Texture, 201
write
 CascadedShadowMap, 31
 OmniDirShadowMap, 139
 ShadowMap, 188
write_light_matrices
 CascadedShadowMap, 32
x_change
 Window, 227
y_change
 Window, 227
yaw
 Camera, 27