

vengine

0.1.0

Generated by Doxygen 1.11.0

1 vengine	1
1.1 VEngine - Vulkan Graphics Engine	1
1.1.1 Features	1
1.1.1.1 Planned Features:	1
1.1.2 Prerequisites	2
1.1.3 Usage	2
1.1.3.1 Build	2
1.1.3.2 Run	2
1.1.4 Key Bindings	2
1.1.5 Documentation	2
1.1.6 Commit Norms	2
1.1.7 License	3
1.1.8 Acknowledgements	3
2 Namespace Index	5
2.1 Namespace List	5
3 Hierarchical Index	7
3.1 Class Hierarchy	7
4 Class Index	9
4.1 Class List	9
5 File Index	11
5.1 File List	11
6 Namespace Documentation	13
6.1 ven Namespace Reference	13
6.1.1 Enumeration Type Documentation	15
6.1.1.1 ENGINE_STATE	15
6.1.1.2 GUI_STATE	15
6.1.2 Function Documentation	15
6.1.2.1 hashCombine()	15
6.1.3 Variable Documentation	16
6.1.3.1 COLOR_MAX	16
6.1.3.2 DEFAULT_AMBIENT_LIGHT_COLOR	16
6.1.3.3 DEFAULT_AMBIENT_LIGHT_INTENSITY	16
6.1.3.4 DEFAULT_CLEAR_COLOR	16
6.1.3.5 DEFAULT_CLEAR_DEPTH	16
6.1.3.6 DEFAULT_FAR	17
6.1.3.7 DEFAULT_FOV	17
6.1.3.8 DEFAULT_HEIGHT	17
6.1.3.9 DEFAULT_LIGHT_COLOR	17
6.1.3.10 DEFAULT_LIGHT_INTENSITY	17

6.1.3.11	DEFAULT_LIGHT_RADIUS	17
6.1.3.12	DEFAULT_LOOK_SPEED	17
6.1.3.13	DEFAULT_MAX_SETS	18
6.1.3.14	DEFAULT_MOVE_SPEED	18
6.1.3.15	DEFAULT_NEAR	18
6.1.3.16	DEFAULT_POSITION	18
6.1.3.17	DEFAULT_ROTATION	18
6.1.3.18	DEFAULT_TITLE	18
6.1.3.19	DEFAULT_WIDTH	18
6.1.3.20	MAX_FRAMES_IN_FLIGHT	19
6.1.3.21	MAX_LIGHTS	19
6.1.3.22	SHADERS_BIN_PATH	19
7	Class Documentation	21
7.1	ven::ARenderSystemBase Class Reference	21
7.1.1	Detailed Description	23
7.1.2	Constructor & Destructor Documentation	23
7.1.2.1	ARenderSystemBase()	23
7.1.2.2	~ARenderSystemBase()	23
7.1.3	Member Function Documentation	24
7.1.3.1	createPipeline()	24
7.1.3.2	createPipelineLayout()	24
7.1.3.3	getDevice()	25
7.1.3.4	getPipelineLayout()	25
7.1.3.5	getShaders()	26
7.1.3.6	render()	26
7.1.4	Member Data Documentation	26
7.1.4.1	m_device	26
7.1.4.2	m_pipelineLayout	26
7.1.4.3	m_shaders	27
7.2	ven::Buffer Class Reference	27
7.2.1	Detailed Description	30
7.2.2	Constructor & Destructor Documentation	30
7.2.2.1	Buffer() [1/2]	30
7.2.2.2	~Buffer()	30
7.2.2.3	Buffer() [2/2]	30
7.2.3	Member Function Documentation	30
7.2.3.1	descriptorInfo()	30
7.2.3.2	descriptorInfoForIndex()	31
7.2.3.3	flush()	32
7.2.3.4	flushIndex()	32
7.2.3.5	getAlignment()	33

7.2.3.6 getAlignmentSize()	33
7.2.3.7 getBuffer()	34
7.2.3.8 getBufferSize()	34
7.2.3.9 getInstanceCount()	34
7.2.3.10 getInstanceSize()	34
7.2.3.11 getMappedMemory()	34
7.2.3.12 getMemoryPropertyFlags()	34
7.2.3.13 getUsageFlags()	35
7.2.3.14 invalidate()	35
7.2.3.15 invalidateIndex()	35
7.2.3.16 map()	36
7.2.3.17 operator=()	37
7.2.3.18 unmap()	37
7.2.3.19 writeToBuffer()	37
7.2.3.20 writeToIndex()	37
7.2.4 Member Data Documentation	38
7.2.4.1 m_alignmentSize	38
7.2.4.2 m_buffer	38
7.2.4.3 m_bufferSize	38
7.2.4.4 m_device	39
7.2.4.5 m_instanceCount	39
7.2.4.6 m_instanceSize	39
7.2.4.7 m_mapped	39
7.2.4.8 m_memory	39
7.2.4.9 m_memoryPropertyFlags	39
7.2.4.10 m_usageFlags	40
7.3 ven::DescriptorPool::Builder Class Reference	40
7.3.1 Detailed Description	42
7.3.2 Constructor & Destructor Documentation	42
7.3.2.1 Builder()	42
7.3.3 Member Function Documentation	42
7.3.3.1 addPoolSize()	42
7.3.3.2 build()	43
7.3.3.3 setMaxSets()	43
7.3.3.4 setPoolFlags()	43
7.3.4 Member Data Documentation	44
7.3.4.1 m_device	44
7.3.4.2 m_maxSets	44
7.3.4.3 m_poolFlags	44
7.3.4.4 m_poolSizes	44
7.4 ven::DescriptorSetLayout::Builder Class Reference	45
7.4.1 Detailed Description	46

7.4.2 Constructor & Destructor Documentation	46
7.4.2.1 Builder()	46
7.4.3 Member Function Documentation	46
7.4.3.1 addBinding()	46
7.4.3.2 build()	47
7.4.4 Member Data Documentation	47
7.4.4.1 m_bindings	47
7.4.4.2 m_device	47
7.5 ven::Model::Builder Struct Reference	48
7.5.1 Detailed Description	49
7.5.2 Member Function Documentation	49
7.5.2.1 loadModel()	49
7.5.2.2 processMesh()	49
7.5.2.3 processNode()	49
7.5.3 Member Data Documentation	49
7.5.3.1 indices	49
7.5.3.2 vertices	50
7.6 ven::Camera Class Reference	50
7.6.1 Detailed Description	52
7.6.2 Constructor & Destructor Documentation	52
7.6.2.1 Camera() [1/2]	52
7.6.2.2 ~Camera()	52
7.6.2.3 Camera() [2/2]	53
7.6.3 Member Function Documentation	53
7.6.3.1 getFar()	53
7.6.3.2 getFov()	53
7.6.3.3 getInverseView()	54
7.6.3.4 getLookSpeed()	54
7.6.3.5 getMoveSpeed()	54
7.6.3.6 getNear()	55
7.6.3.7 getProjection()	55
7.6.3.8 getView()	55
7.6.3.9 operator=()	55
7.6.3.10 setFar()	56
7.6.3.11 setFov()	56
7.6.3.12 setLookSpeed()	57
7.6.3.13 setMoveSpeed()	57
7.6.3.14 setNear()	58
7.6.3.15 setOrthographicProjection()	58
7.6.3.16 setPerspectiveProjection()	58
7.6.3.17 setViewDirection()	58
7.6.3.18 setViewTarget()	59

7.6.3.19 setViewXYZ()	59
7.6.4 Member Data Documentation	59
7.6.4.1 m_far	59
7.6.4.2 m_fov	59
7.6.4.3 m_inverseViewMatrix	59
7.6.4.4 m_lookSpeed	59
7.6.4.5 m_moveSpeed	60
7.6.4.6 m_near	60
7.6.4.7 m_projectionMatrix	60
7.6.4.8 m_viewMatrix	60
7.6.4.9 transform3D	60
7.7 ven::Colors Class Reference	61
7.7.1 Detailed Description	63
7.7.2 Member Data Documentation	63
7.7.2.1 AQUA_3	63
7.7.2.2 AQUA_4	63
7.7.2.3 AQUA_V	63
7.7.2.4 BLACK_3	63
7.7.2.5 BLACK_4	63
7.7.2.6 BLACK_V	63
7.7.2.7 BLUE_3	64
7.7.2.8 BLUE_4	64
7.7.2.9 BLUE_V	64
7.7.2.10 COLOR_PRESETS_3	64
7.7.2.11 COLOR_PRESETS_4	65
7.7.2.12 COLOR_PRESETS_VK	65
7.7.2.13 CYAN_3	65
7.7.2.14 CYAN_4	66
7.7.2.15 CYAN_V	66
7.7.2.16 FUCHSIA_3	66
7.7.2.17 FUCHSIA_4	66
7.7.2.18 FUCHSIA_V	66
7.7.2.19 GRAY_3	66
7.7.2.20 GRAY_4	66
7.7.2.21 GRAY_V	67
7.7.2.22 GREEN_3	67
7.7.2.23 GREEN_4	67
7.7.2.24 GREEN_V	67
7.7.2.25 LIME_3	67
7.7.2.26 LIME_4	67
7.7.2.27 LIME_V	67
7.7.2.28 MAGENTA_3	68

7.7.2.29 MAGENTA_4	68
7.7.2.30 MAGENTA_V	68
7.7.2.31 MAROON_3	68
7.7.2.32 MAROON_4	68
7.7.2.33 MAROON_V	68
7.7.2.34 NAVY_3	68
7.7.2.35 NAVY_4	69
7.7.2.36 NAVY_V	69
7.7.2.37 NIGHT_BLUE_3	69
7.7.2.38 NIGHT_BLUE_4	69
7.7.2.39 NIGHT_BLUE_V	69
7.7.2.40 OLIVE_3	69
7.7.2.41 OLIVE_4	69
7.7.2.42 OLIVE_V	70
7.7.2.43 RED_3	70
7.7.2.44 RED_4	70
7.7.2.45 RED_V	70
7.7.2.46 SILVER_3	70
7.7.2.47 SILVER_4	70
7.7.2.48 SILVER_V	70
7.7.2.49 SKY_BLUE_3	71
7.7.2.50 SKY_BLUE_4	71
7.7.2.51 SKY_BLUE_V	71
7.7.2.52 SUNSET_3	71
7.7.2.53 SUNSET_4	71
7.7.2.54 SUNSET_V	71
7.7.2.55 TEAL_3	71
7.7.2.56 TEAL_4	72
7.7.2.57 TEAL_V	72
7.7.2.58 WHITE_3	72
7.7.2.59 WHITE_4	72
7.7.2.60 WHITE_V	72
7.7.2.61 YELLOW_3	72
7.7.2.62 YELLOW_4	72
7.7.2.63 YELLOW_V	73
7.8 ven::DescriptorPool Class Reference	73
7.8.1 Detailed Description	75
7.8.2 Constructor & Destructor Documentation	75
7.8.2.1 DescriptorPool() [1/2]	75
7.8.2.2 ~DescriptorPool()	76
7.8.2.3 DescriptorPool() [2/2]	76
7.8.3 Member Function Documentation	76

7.8.3.1 allocateDescriptor()	76
7.8.3.2 freeDescriptors()	76
7.8.3.3 getDescriptorPool()	77
7.8.3.4 operator=()	77
7.8.3.5 resetPool()	77
7.8.4 Friends And Related Symbol Documentation	77
7.8.4.1 DescriptorWriter	77
7.8.5 Member Data Documentation	77
7.8.5.1 m_descriptorPool	77
7.8.5.2 m_device	78
7.9 ven::DescriptorSetLayout Class Reference	78
7.9.1 Detailed Description	80
7.9.2 Constructor & Destructor Documentation	80
7.9.2.1 DescriptorSetLayout() [1/2]	80
7.9.2.2 ~DescriptorSetLayout()	81
7.9.2.3 DescriptorSetLayout() [2/2]	81
7.9.3 Member Function Documentation	81
7.9.3.1 getDescriptorSetLayout()	81
7.9.3.2 operator=()	81
7.9.4 Friends And Related Symbol Documentation	81
7.9.4.1 DescriptorWriter	81
7.9.5 Member Data Documentation	82
7.9.5.1 m_bindings	82
7.9.5.2 m_descriptorSetLayout	82
7.9.5.3 m_device	82
7.10 ven::DescriptorWriter Class Reference	82
7.10.1 Detailed Description	84
7.10.2 Constructor & Destructor Documentation	84
7.10.2.1 DescriptorWriter() [1/2]	84
7.10.2.2 ~DescriptorWriter()	84
7.10.2.3 DescriptorWriter() [2/2]	84
7.10.3 Member Function Documentation	84
7.10.3.1 build()	84
7.10.3.2 operator=()	85
7.10.3.3 overwrite()	85
7.10.3.4 writeBuffer()	85
7.10.3.5 writelImage()	85
7.10.4 Member Data Documentation	85
7.10.4.1 m_pool	85
7.10.4.2 m_setLayout	86
7.10.4.3 m_writes	86
7.11 ven::Device Class Reference	86

7.11.1 Detailed Description	89
7.11.2 Constructor & Destructor Documentation	89
7.11.2.1 Device() [1/2]	89
7.11.2.2 ~Device()	90
7.11.2.3 Device() [2/2]	90
7.11.3 Member Function Documentation	90
7.11.3.1 beginSingleTimeCommands()	90
7.11.3.2 checkDeviceExtensionSupport()	90
7.11.3.3 checkValidationLayerSupport()	90
7.11.3.4 copyBuffer()	91
7.11.3.5 copyBufferToImage()	91
7.11.3.6 createBuffer()	91
7.11.3.7 createCommandPool()	92
7.11.3.8 createImageWithInfo()	92
7.11.3.9 createInstance()	92
7.11.3.10 createLogicalDevice()	93
7.11.3.11 createSurface()	93
7.11.3.12 device()	94
7.11.3.13 endSingleTimeCommands()	94
7.11.3.14 findMemoryType()	95
7.11.3.15 findPhysicalQueueFamilies()	95
7.11.3.16 findQueueFamilies()	95
7.11.3.17 findSupportedFormat()	96
7.11.3.18 getCommandPool()	96
7.11.3.19 getGraphicsQueue()	96
7.11.3.20 getPhysicalDevice()	96
7.11.3.21 getRequiredExtensions()	97
7.11.3.22 getSwapChainSupport()	97
7.11.3.23 graphicsQueue()	97
7.11.3.24 hasGlfwRequiredInstanceExtensions()	97
7.11.3.25 isDeviceSuitable()	98
7.11.3.26 operator=()	98
7.11.3.27 pickPhysicalDevice()	98
7.11.3.28 populateDebugMessengerCreateInfo()	99
7.11.3.29 presentQueue()	99
7.11.3.30 querySwapChainSupport()	99
7.11.3.31 setupDebugMessenger()	100
7.11.3.32 surface()	100
7.11.4 Member Data Documentation	100
7.11.4.1 enableValidationLayers	100
7.11.4.2 m_commandPool	100
7.11.4.3 m_debugMessenger	101

7.11.4.4 m_device	101
7.11.4.5 m_deviceExtensions	101
7.11.4.6 m_graphicsQueue	101
7.11.4.7 m_instance	101
7.11.4.8 m_physicalDevice	101
7.11.4.9 m_presentQueue	102
7.11.4.10 m_properties	102
7.11.4.11 m_surface	102
7.11.4.12 m_validationLayers	102
7.11.4.13 m_window	102
7.12 ven::Engine Class Reference	103
7.12.1 Detailed Description	104
7.12.2 Constructor & Destructor Documentation	104
7.12.2.1 Engine() [1/2]	104
7.12.2.2 ~Engine()	105
7.12.2.3 Engine() [2/2]	105
7.12.3 Member Function Documentation	105
7.12.3.1 createInstance()	105
7.12.3.2 createSurface()	106
7.12.3.3 loadObjects()	106
7.12.3.4 mainLoop()	107
7.12.3.5 operator=()	108
7.12.4 Member Data Documentation	108
7.12.4.1 m_device	108
7.12.4.2 m_globalPool	109
7.12.4.3 m_gui	109
7.12.4.4 m_instance	109
7.12.4.5 m_lights	109
7.12.4.6 m_objects	109
7.12.4.7 m_renderer	109
7.12.4.8 m_state	109
7.12.4.9 m_surface	110
7.12.4.10 m_window	110
7.13 ven::EventManager Class Reference	110
7.13.1 Detailed Description	112
7.13.2 Constructor & Destructor Documentation	112
7.13.2.1 EventManager() [1/2]	112
7.13.2.2 ~EventManager()	112
7.13.2.3 EventManager() [2/2]	112
7.13.3 Member Function Documentation	112
7.13.3.1 handleEvents()	112
7.13.3.2 isKeyJustPressed()	113

7.13.3.3 moveCamera()	113
7.13.3.4 operator=()	114
7.13.3.5 updateEngineState()	114
7.13.4 Member Data Documentation	114
7.13.4.1 m_keys	114
7.13.4.2 m_keyState	115
7.14 ven::FrameInfo Struct Reference	115
7.14.1 Detailed Description	117
7.14.2 Member Data Documentation	117
7.14.2.1 camera	117
7.14.2.2 commandBuffer	117
7.14.2.3 frameIndex	117
7.14.2.4 frameTime	117
7.14.2.5 globalDescriptorSet	117
7.14.2.6 lights	118
7.14.2.7 objects	118
7.15 ven::Gui::funcs Struct Reference	118
7.15.1 Detailed Description	118
7.15.2 Member Function Documentation	119
7.15.2.1 IsLegacyNativeDupe()	119
7.16 ven::GlobalUbo Struct Reference	120
7.16.1 Detailed Description	120
7.16.2 Member Data Documentation	121
7.16.2.1 ambientLightColor	121
7.16.2.2 inverseView	121
7.16.2.3 numLights	121
7.16.2.4 pointLights	121
7.16.2.5 projection	121
7.16.2.6 view	121
7.17 ven::Gui Class Reference	122
7.17.1 Detailed Description	123
7.17.2 Constructor & Destructor Documentation	123
7.17.2.1 Gui() [1/2]	123
7.17.2.2 ~Gui()	123
7.17.2.3 Gui() [2/2]	124
7.17.3 Member Function Documentation	124
7.17.3.1 cameraSection()	124
7.17.3.2 cleanup()	125
7.17.3.3 devicePropertiesSection()	125
7.17.3.4 getState()	125
7.17.3.5 init()	126
7.17.3.6 initStyle()	127

7.17.3.7 inputsSection()	127
7.17.3.8 lightsSection()	127
7.17.3.9 objectsSection()	127
7.17.3.10 operator=()	127
7.17.3.11 render()	128
7.17.3.12 rendererSection()	128
7.17.3.13 renderFrameWindow()	129
7.17.3.14 setState()	129
7.17.4 Member Data Documentation	130
7.17.4.1 m_io	130
7.17.4.2 m_state	130
7.18 std::hash< ven::Model::Vertex > Struct Reference	130
7.18.1 Detailed Description	130
7.18.2 Member Function Documentation	131
7.18.2.1 operator()()	131
7.19 ven::KeyAction Struct Reference	131
7.19.1 Detailed Description	132
7.19.2 Member Data Documentation	132
7.19.2.1 dir	132
7.19.2.2 key	132
7.19.2.3 value	132
7.20 ven::KeyMappings Struct Reference	133
7.20.1 Detailed Description	133
7.20.2 Member Data Documentation	134
7.20.2.1 lookDown	134
7.20.2.2 lookLeft	134
7.20.2.3 lookRight	134
7.20.2.4 lookUp	134
7.20.2.5 moveBackward	134
7.20.2.6 moveDown	134
7.20.2.7 moveForward	134
7.20.2.8 moveLeft	135
7.20.2.9 moveRight	135
7.20.2.10 moveUp	135
7.20.2.11 toggleGui	135
7.21 ven::Light Class Reference	135
7.21.1 Detailed Description	137
7.21.2 Member Typedef Documentation	137
7.21.2.1 Map	137
7.21.3 Constructor & Destructor Documentation	137
7.21.3.1 ~Light()	137
7.21.3.2 Light() [1/3]	138

7.21.3.3 Light() [2/3]	138
7.21.3.4 Light() [3/3]	138
7.21.4 Member Function Documentation	138
7.21.4.1 createLight()	138
7.21.4.2 getId()	138
7.21.4.3 getName()	139
7.21.4.4 operator=() [1/2]	139
7.21.4.5 operator=() [2/2]	139
7.21.4.6 setName()	139
7.21.5 Member Data Documentation	139
7.21.5.1 color	139
7.21.5.2 m_lightId	139
7.21.5.3 m_name	140
7.21.5.4 transform3D	140
7.22 ven::LightPushConstantData Struct Reference	140
7.22.1 Detailed Description	141
7.22.2 Member Data Documentation	141
7.22.2.1 color	141
7.22.2.2 position	141
7.22.2.3 radius	141
7.23 ven::Model Class Reference	141
7.23.1 Detailed Description	143
7.23.2 Constructor & Destructor Documentation	143
7.23.2.1 Model() [1/2]	143
7.23.2.2 ~Model()	144
7.23.2.3 Model() [2/2]	144
7.23.3 Member Function Documentation	144
7.23.3.1 bind()	144
7.23.3.2 createIndexBuffer()	144
7.23.3.3 createModelFromFile()	145
7.23.3.4 createVertexBuffer()	145
7.23.3.5 draw()	146
7.23.3.6 operator=()	146
7.23.4 Member Data Documentation	146
7.23.4.1 m_device	146
7.23.4.2 m_hasIndexBuffer	146
7.23.4.3 m_indexBuffer	146
7.23.4.4 m_indexCount	147
7.23.4.5 m_vertexBuffer	147
7.23.4.6 m_vertexCount	147
7.24 ven::Object Class Reference	147
7.24.1 Detailed Description	149

7.24.2 Member Typedef Documentation	149
7.24.2.1 Map	149
7.24.3 Constructor & Destructor Documentation	149
7.24.3.1 ~Object()	149
7.24.3.2 Object() [1/3]	150
7.24.3.3 Object() [2/3]	150
7.24.3.4 Object() [3/3]	150
7.24.4 Member Function Documentation	150
7.24.4.1 createObject()	150
7.24.4.2 getId()	151
7.24.4.3 getModel()	151
7.24.4.4 getName()	151
7.24.4.5 operator=() [1/2]	151
7.24.4.6 operator=() [2/2]	151
7.24.4.7 setModel()	152
7.24.4.8 setName()	152
7.24.5 Member Data Documentation	152
7.24.5.1 m_model	152
7.24.5.2 m_name	153
7.24.5.3 m_objId	153
7.24.5.4 transform3D	153
7.25 ven::ObjectPushConstantData Struct Reference	153
7.25.1 Detailed Description	154
7.25.2 Member Data Documentation	154
7.25.2.1 modelMatrix	154
7.25.2.2 normalMatrix	154
7.26 ven::ObjectRenderSystem Class Reference	154
7.26.1 Detailed Description	157
7.26.2 Constructor & Destructor Documentation	157
7.26.2.1 ObjectRenderSystem() [1/2]	157
7.26.2.2 ObjectRenderSystem() [2/2]	157
7.26.3 Member Function Documentation	158
7.26.3.1 operator=()	158
7.26.3.2 render()	158
7.27 ven::PipelineConfigInfo Struct Reference	159
7.27.1 Detailed Description	159
7.27.2 Constructor & Destructor Documentation	160
7.27.2.1 PipelineConfigInfo() [1/2]	160
7.27.2.2 PipelineConfigInfo() [2/2]	160
7.27.3 Member Function Documentation	160
7.27.3.1 operator=()	160
7.27.4 Member Data Documentation	160

7.27.4.1 attributeDescriptions	160
7.27.4.2 bindingDescriptions	160
7.27.4.3 colorBlendAttachment	160
7.27.4.4 colorBlendInfo	161
7.27.4.5 depthStencilInfo	161
7.27.4.6 dynamicStateEnables	161
7.27.4.7 dynamicStateInfo	161
7.27.4.8 inputAssemblyInfo	161
7.27.4.9 multisampleInfo	161
7.27.4.10 pipelineLayout	162
7.27.4.11 rasterizationInfo	162
7.27.4.12 renderPass	162
7.27.4.13 subpass	162
7.28 ven::PointLightData Struct Reference	162
7.28.1 Detailed Description	163
7.28.2 Member Data Documentation	163
7.28.2.1 color	163
7.28.2.2 position	163
7.29 ven::PointLightRenderSystem Class Reference	163
7.29.1 Detailed Description	166
7.29.2 Constructor & Destructor Documentation	166
7.29.2.1 PointLightRenderSystem() [1/2]	166
7.29.2.2 PointLightRenderSystem() [2/2]	167
7.29.3 Member Function Documentation	167
7.29.3.1 operator=()	167
7.29.3.2 render()	167
7.29.3.3 update()	168
7.30 ven::QueueFamilyIndices Struct Reference	168
7.30.1 Detailed Description	169
7.30.2 Member Function Documentation	169
7.30.2.1 isComplete()	169
7.30.3 Member Data Documentation	169
7.30.3.1 graphicsFamily	169
7.30.3.2 graphicsFamilyHasValue	169
7.30.3.3 presentFamily	170
7.30.3.4 presentFamilyHasValue	170
7.31 ven::Renderer Class Reference	170
7.31.1 Detailed Description	172
7.31.2 Constructor & Destructor Documentation	172
7.31.2.1 Renderer() [1/2]	172
7.31.2.2 ~Renderer()	173
7.31.2.3 Renderer() [2/2]	173

7.31.3 Member Function Documentation	173
7.31.3.1 beginFrame()	173
7.31.3.2 beginSwapChainRenderPass()	173
7.31.3.3 createCommandBuffers()	174
7.31.3.4 endFrame()	174
7.31.3.5 endSwapChainRenderPass()	174
7.31.3.6 freeCommandBuffers()	175
7.31.3.7 getAspectRatio()	175
7.31.3.8 getClearColor()	175
7.31.3.9 getCurrentCommandBuffer()	176
7.31.3.10 getFrameIndex()	176
7.31.3.11 getSwapChainRenderPass()	177
7.31.3.12 getWindow()	177
7.31.3.13 isFrameInProgress()	177
7.31.3.14 operator=()	178
7.31.3.15 recreateSwapChain()	178
7.31.3.16 setClearColor()	178
7.31.4 Member Data Documentation	179
7.31.4.1 m_clearValues	179
7.31.4.2 m_commandBuffers	179
7.31.4.3 m_currentFrameIndex	179
7.31.4.4 m_currentImageIndex	179
7.31.4.5 m_device	179
7.31.4.6 m_isFrameStarted	179
7.31.4.7 m_swapChain	180
7.31.4.8 m_window	180
7.32 ven::Shaders Class Reference	180
7.32.1 Detailed Description	182
7.32.2 Constructor & Destructor Documentation	182
7.32.2.1 Shaders() [1/2]	182
7.32.2.2 ~Shaders()	183
7.32.2.3 Shaders() [2/2]	183
7.32.3 Member Function Documentation	183
7.32.3.1 bind()	183
7.32.3.2 createGraphicsPipeline()	183
7.32.3.3 createShaderModule()	184
7.32.3.4 defaultPipelineConfigInfo()	184
7.32.3.5 operator=()	185
7.32.3.6 readFile()	185
7.32.4 Member Data Documentation	185
7.32.4.1 m_device	185
7.32.4.2 m_fragShaderModule	185

7.32.4.3 m_graphicsPipeline	185
7.32.4.4 m_vertShaderModule	186
7.33 ven::SwapChain Class Reference	186
7.33.1 Detailed Description	189
7.33.2 Constructor & Destructor Documentation	189
7.33.2.1 SwapChain() [1/3]	189
7.33.2.2 SwapChain() [2/3]	189
7.33.2.3 ~SwapChain()	190
7.33.2.4 SwapChain() [3/3]	190
7.33.3 Member Function Documentation	190
7.33.3.1 acquireNextImage()	190
7.33.3.2 chooseSwapExtent()	190
7.33.3.3 chooseSwapPresentMode()	190
7.33.3.4 chooseSwapSurfaceFormat()	191
7.33.3.5 compareSwapFormats()	191
7.33.3.6 createDepthResources()	191
7.33.3.7 createFrameBuffers()	191
7.33.3.8 createImageViews()	191
7.33.3.9 createRenderPass()	191
7.33.3.10 createSwapChain()	191
7.33.3.11 createSyncObjects()	192
7.33.3.12 extentAspectRatio()	192
7.33.3.13 findDepthFormat()	192
7.33.3.14 getFrameBuffer()	192
7.33.3.15 getImageView()	192
7.33.3.16 getRenderPass()	192
7.33.3.17 getSwapChainExtent()	193
7.33.3.18 getSwapChainImageFormat()	193
7.33.3.19 height()	193
7.33.3.20 imageCount()	193
7.33.3.21 init()	193
7.33.3.22 operator=()	194
7.33.3.23 submitCommandBuffers()	194
7.33.3.24 width()	194
7.33.4 Member Data Documentation	194
7.33.4.1 m_currentFrame	194
7.33.4.2 m_depthImageMemory	194
7.33.4.3 m_depthImages	194
7.33.4.4 m_depthImageViews	195
7.33.4.5 m_device	195
7.33.4.6 m_imageAvailableSemaphores	195
7.33.4.7 m_imagesInFlight	195

7.33.4.8 m_inFlightFences	195
7.33.4.9 m_oldSwapChain	195
7.33.4.10 m_renderFinishedSemaphores	196
7.33.4.11 m_renderPass	196
7.33.4.12 m_swapChain	196
7.33.4.13 m_swapChainDepthFormat	196
7.33.4.14 m_swapChainExtent	196
7.33.4.15 m_swapChainFrameBuffers	196
7.33.4.16 m_swapChainImageFormat	197
7.33.4.17 m_swapChainImages	197
7.33.4.18 m_swapChainImageViews	197
7.33.4.19 m_windowExtent	197
7.34 ven::SwapChainSupportDetails Struct Reference	197
7.34.1 Detailed Description	198
7.34.2 Member Data Documentation	198
7.34.2.1 capabilities	198
7.34.2.2 formats	198
7.34.2.3 presentModes	198
7.35 ven::Transform3DComponent Class Reference	199
7.35.1 Detailed Description	199
7.35.2 Member Function Documentation	199
7.35.2.1 mat4()	199
7.35.2.2 normalMatrix()	200
7.35.3 Member Data Documentation	200
7.35.3.1 rotation	200
7.35.3.2 scale	200
7.35.3.3 translation	200
7.36 ven::Model::Vertex Struct Reference	200
7.36.1 Detailed Description	201
7.36.2 Member Function Documentation	201
7.36.2.1 getAttributeDescriptions()	201
7.36.2.2 getBindingDescriptions()	202
7.36.2.3 operator==()	202
7.36.3 Member Data Documentation	202
7.36.3.1 color	202
7.36.3.2 normal	202
7.36.3.3 position	203
7.36.3.4 uv	203
7.37 ven::Window Class Reference	203
7.37.1 Detailed Description	204
7.37.2 Constructor & Destructor Documentation	204
7.37.2.1 Window() [1/2]	204

7.37.2.2 ~Window()	204
7.37.2.3 Window() [2/2]	205
7.37.3 Member Function Documentation	205
7.37.3.1 createWindow()	205
7.37.3.2 createWindowSurface()	205
7.37.3.3 framebufferResizeCallback()	206
7.37.3.4 getExtent()	206
7.37.3.5 getGLFWWindow()	207
7.37.3.6 operator=()	207
7.37.3.7 resetWindowResizedFlag()	207
7.37.3.8 setFullscreen()	207
7.37.3.9 wasWindowResized()	208
7.37.4 Member Data Documentation	208
7.37.4.1 m_framebufferResized	208
7.37.4.2 m_height	208
7.37.4.3 m_width	208
7.37.4.4 m_window	208
8 File Documentation	209
8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag File Reference	209
8.2 fragment_point_light.frag	209
8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag File Reference	209
8.4 fragment_shader.frag	209
8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert File Reference	210
8.6 vertex_point_light.vert	210
8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference	211
8.8 vertex_shader.vert	211
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference	212
8.9.1 Detailed Description	213
8.10 Buffer.hpp	213
8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference	215
8.11.1 Detailed Description	216
8.12 Camera.hpp	216
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference	217
8.14 Colors.hpp	218
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference	221
8.15.1 Detailed Description	222
8.16 DescriptorPool.hpp	222
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference	223
8.17.1 Detailed Description	224
8.18 DescriptorSetLayout.hpp	224

8.19	/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference	225
8.19.1	Detailed Description	227
8.20	DescriptorWriter.hpp	227
8.21	/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference	227
8.21.1	Detailed Description	228
8.22	Device.hpp	229
8.23	/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference	230
8.23.1	Detailed Description	231
8.24	Engine.hpp	231
8.25	/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp File Reference	232
8.25.1	Detailed Description	233
8.26	EventManager.hpp	234
8.27	/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference	234
8.27.1	Detailed Description	235
8.28	FrameInfo.hpp	236
8.29	/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp File Reference	236
8.29.1	Detailed Description	237
8.30	Gui.hpp	238
8.31	/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp File Reference	238
8.31.1	Detailed Description	239
8.32	Light.hpp	240
8.33	/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference	240
8.33.1	Detailed Description	241
8.34	Model.hpp	242
8.35	/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference	243
8.35.1	Detailed Description	244
8.36	Object.hpp	244
8.37	/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference	244
8.37.1	Detailed Description	246
8.38	Renderer.hpp	246
8.39	/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp File Reference	247
8.39.1	Detailed Description	248
8.40	ARenderSystemBase.hpp	248
8.41	/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp File Reference	249
8.41.1	Detailed Description	250
8.42	ObjectRenderSystem.hpp	250
8.43	/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp File Reference	251
8.43.1	Detailed Description	252
8.44	PointLightRenderSystem.hpp	252

8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference	253
8.45.1 Detailed Description	254
8.46 Shaders.hpp	254
8.47 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference	255
8.47.1 Detailed Description	257
8.48 SwapChain.hpp	257
8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp File Reference	258
8.49.1 Detailed Description	259
8.50 Transform3DComponent.hpp	260
8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference	260
8.52 Utils.hpp	261
8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference	261
8.53.1 Detailed Description	263
8.53.2 Macro Definition Documentation	263
8.53.2.1 GLFW_INCLUDE_VULKAN	263
8.54 Window.hpp	263
8.55 /home/runner/work/VEngine/VEngine/README.md File Reference	264
8.56 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference	264
8.57 buffer.cpp	264
8.58 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference	265
8.59 camera.cpp	266
8.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference	268
8.61 descriptorPool.cpp	268
8.62 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference	269
8.63 descriptorSetLayout.cpp	269
8.64 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference	270
8.65 descriptorWriter.cpp	270
8.66 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference	271
8.66.1 Function Documentation	272
8.66.1.1 CreateDebugUtilsMessengerEXT()	272
8.66.1.2 debugCallback()	273
8.66.1.3 DestroyDebugUtilsMessengerEXT()	273
8.67 device.cpp	274
8.68 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference	280
8.69 engine.cpp	280
8.70 /home/runner/work/VEngine/VEngine/src/eventManager.cpp File Reference	282
8.71 eventManager.cpp	283
8.72 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference	284
8.72.1 Variable Documentation	284
8.72.1.1 DESCRIPTOR_COUNT	284
8.73 init.cpp	285
8.74 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference	286

8.75 render.cpp	286
8.76 /home/runner/work/VEngine/VEngine/src/light.cpp File Reference	290
8.77 light.cpp	291
8.78 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference	291
8.78.1 Function Documentation	291
8.78.1.1 main()	291
8.79 main.cpp	292
8.80 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference	292
8.80.1 Macro Definition Documentation	293
8.80.1.1 GLM_ENABLE_EXPERIMENTAL	293
8.81 model.cpp	293
8.82 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference	296
8.83 renderer.cpp	296
8.84 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference	298
8.85 shaders.cpp	298
8.86 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference	301
8.87 swapChain.cpp	301
8.88 /home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp File Reference	306
8.89 objectRenderSystem.cpp	306
8.90 /home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp File Reference	307
8.91 pointLightRenderSystem.cpp	307
8.92 /home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp File Reference	308
8.93 renderSystemBase.cpp	308
8.94 /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp File Reference	309
8.95 transform3DComponent.cpp	309
8.96 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference	310
8.97 window.cpp	311
Index	313

Chapter 1

vengine

1.1 VEngine - Vulkan Graphics Engine

WORK IN PROGRESS!

Welcome to **VEngine**, a Vulkan-based graphics engine.

This project is designed to provide a high-performance and flexible foundation for building 3D applications and games, taking full advantage of the Vulkan API.

1.1.1 Features

- **Vulkan Rendering Pipeline:** Leveraging Vulkan for high-performance graphics rendering
- **Basic Camera System:** Control camera movement in the 3D space
- **Input System:** Keyboard-based controls for movement and looking around
- **Model Loading:** Import 3D models using [assimp](#)
- **Real-time debugging:** Toggle debug windows using key bindings
- **Doxygen Documentation:** Automatically generated documentation hosted on [GitHub Pages](#)

1.1.1.1 Planned Features:

- **Cross-platform support** (Linux, macOS, Windows)
- Improve shadow
- Physics Integration
- Support for more input devices (e.g., mouse, game controller)
- Audio Integration

1.1.2 Prerequisites

Make sure you have the following dependencies installed on your system:

- [CMake 3.27](#)
- [C++20](#)
- [Vulkan](#)
- [GLM](#)

1.1.3 Usage

1.1.3.1 Build

```
$> ./tools/build.sh build  
[...]
```

This script also handle several other commands: `clean`, `format` and `doc`.

1.1.3.2 Run

```
$> ./vengine  
[...]
```

1.1.4 Key Bindings

The following keyboard controls are currently available for interacting with the engine:

Key	Description
z	Move forward
S	Move backward
q	Move left
D	Move right
SHIFT	Move down
SPACE	Move up
arrow up	Look up
arrow down	Look down
arrow left	Look left
arrow right	Look right
F1	Show debug windows

1.1.5 Documentation

The documentation is generated using [Doxygen](#). You can access the latest version on the [GitHub Pages](#).

1.1.6 Commit Norms

Commit Type	Description
build	Changes that affect the build system or external dependencies (npm, make, etc.)
ci	Changes related to integration files and scripts or configuration (Travis, Ansible, BrowserStack, etc.)
feat	Addition of a new feature
fix	Bug fix
perf	Performance improvements
refactor	Modification that neither adds a new feature nor improves performance
style	Change that does not affect functionality or semantics (indentation, formatting, adding space, renaming a variable, etc.)
docs	Writing or updating documentation
test	Addition or modification of tests

1.1.7 License

This project is licensed under the MIT License - see the [LICENSE](#) file for details.

1.1.8 Acknowledgements

Special thanks to [Brendan Galea](#) for inspiration and resources related to Vulkan development.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ven	13
---------------------	-------	--------------------

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ven::ARenderSystemBase	21
ven::ObjectRenderSystem	154
ven::PointLightRenderSystem	163
ven::Buffer	27
ven::DescriptorPool::Builder	40
ven::DescriptorSetLayout::Builder	45
ven::Model::Builder	48
ven::Camera	50
ven::Colors	61
ven::DescriptorPool	73
ven::DescriptorSetLayout	78
ven::DescriptorWriter	82
ven::Device	86
ven::Engine	103
ven::EventManager	110
ven::FrameInfo	115
ven::Gui::funcs	118
ven::GlobalUbo	120
ven::Gui	122
std::hash< ven::Model::Vertex >	130
ven::KeyAction	131
ven::KeyMappings	133
ven::Light	135
ven::LightPushConstantData	140
ven::Model	141
ven::Object	147
ven::ObjectPushConstantData	153
ven::PipelineConfigInfo	159
ven::PointLightData	162
ven::QueueFamilyIndices	168
ven::Renderer	170
ven::Shaders	180
ven::SwapChain	186
ven::SwapChainSupportDetails	197
ven::Transform3DComponent	199
ven::Model::Vertex	200
ven::Window	203

Chapter 4

Class Index

4.1 Class List

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

ven::ARenderSystemBase	Abstract class for render system base	21
ven::Buffer	Class for buffer	27
ven::DescriptorPool::Builder	40
ven::DescriptorSetLayout::Builder	45
ven::Model::Builder	48
ven::Camera	Class for camera	50
ven::Colors	Class for colors	61
ven::DescriptorPool	Class for descriptor pool	73
ven::DescriptorSetLayout	Class for descriptor set layout	78
ven::DescriptorWriter	Class for descriptor writer	82
ven::Device	Class for device	86
ven::Engine	Class for engine	103
ven::EventManager	Class for event manager	110
ven::FrameInfo	115
ven::Gui::funcs	118
ven::GlobalUbo	120
ven::Gui	Class for Gui	122
std::hash< ven::Model::Vertex >	130
ven::KeyAction	131
ven::KeyMappings	133
ven::Light	Class for light	135
ven::LightPushConstantData	140
ven::Model	Class for model	141

ven::Object	
Class for object	147
ven::ObjectPushConstantData	153
ven::ObjectRenderSystem	
Class for object render system	154
ven::PipelineConfigInfo	159
ven::PointLightData	162
ven::PointLightRenderSystem	
Class for point light system	163
ven::QueueFamilyIndices	168
ven::Renderer	
Class for renderer	170
ven::Shaders	
Class for shaders	180
ven::SwapChain	
Class for swap chain	186
ven::SwapChainSupportDetails	197
ven::Transform3DComponent	
Class for 3D transformation	199
ven::Model::Vertex	200
ven::Window	
Class for window	203

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag	209
/home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag	209
/home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert	210
/home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert	211
/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp	
This file contains the Buffer class	212
/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp	
This file contains the Camera class	215
/home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp	217
/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp	
This file contains the Device class	227
/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp	
This file contains the Engine class	230
/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp	
This file contains the EventManager class	232
/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp	
This file contains the FrameInfo class	234
/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp	
This file contains the ImGuiWindowManager class	236
/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp	
This file contains the Light class	238
/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp	
This file contains the Model class	240
/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp	
This file contains the Object class	243
/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp	
This file contains the Renderer class	244
/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp	
This file contains the Shader class	253
/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp	
This file contains the Shader class	255
/home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp	
This file contains the Transform3DComponent class	258
/home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp	260
/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp	
This file contains the Window class	261

/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/ DescriptorPool.hpp	
This file contains the DescriptorPool class	221
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/ DescriptorSetLayout.hpp	
This file contains the DescriptorSetLayout class	223
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/ DescriptorWriter.hpp	
This file contains the DescriptorsWriter class	225
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ ARenderSystemBase.hpp	
This file contains the ARenderSystemBase class	247
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ ObjectRenderSystem.hpp	
This file contains the ObjectRenderSystem class	249
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ PointLightRenderSystem.hpp	
This file contains the PointLightRenderSystem class	251
/home/runner/work/VEngine/VEngine/src/ buffer.cpp	264
/home/runner/work/VEngine/VEngine/src/ camera.cpp	265
/home/runner/work/VEngine/VEngine/src/ device.cpp	271
/home/runner/work/VEngine/VEngine/src/ engine.cpp	280
/home/runner/work/VEngine/VEngine/src/ eventManager.cpp	282
/home/runner/work/VEngine/VEngine/src/ light.cpp	290
/home/runner/work/VEngine/VEngine/src/ main.cpp	291
/home/runner/work/VEngine/VEngine/src/ model.cpp	292
/home/runner/work/VEngine/VEngine/src/ renderer.cpp	296
/home/runner/work/VEngine/VEngine/src/ shaders.cpp	298
/home/runner/work/VEngine/VEngine/src/ swapChain.cpp	301
/home/runner/work/VEngine/VEngine/src/ transform3DComponent.cpp	309
/home/runner/work/VEngine/VEngine/src/ window.cpp	310
/home/runner/work/VEngine/VEngine/src/descriptors/ descriptorPool.cpp	268
/home/runner/work/VEngine/VEngine/src/descriptors/ descriptorSetLayout.cpp	269
/home/runner/work/VEngine/VEngine/src/descriptors/ descriptorWriter.cpp	270
/home/runner/work/VEngine/VEngine/src/gui/ init.cpp	284
/home/runner/work/VEngine/VEngine/src/gui/ render.cpp	286
/home/runner/work/VEngine/VEngine/src/system/ objectRenderSystem.cpp	306
/home/runner/work/VEngine/VEngine/src/system/ pointLightRenderSystem.cpp	307
/home/runner/work/VEngine/VEngine/src/system/ renderSystemBase.cpp	308

Chapter 6

Namespace Documentation

6.1 ven Namespace Reference

Classes

- class [ARenderSystemBase](#)
Abstract class for render system base.
- class [Buffer](#)
Class for buffer.
- class [Camera](#)
Class for camera.
- class [Colors](#)
Class for colors.
- class [DescriptorPool](#)
Class for descriptor pool.
- class [DescriptorSetLayout](#)
Class for descriptor set layout.
- class [DescriptorWriter](#)
Class for descriptor writer.
- class [Device](#)
Class for device.
- class [Engine](#)
Class for engine.
- class [EventManager](#)
Class for event manager.
- struct [FrameInfo](#)
- struct [GlobalUbo](#)
- class [Gui](#)
Class for Gui.
- struct [KeyAction](#)
- struct [KeyMappings](#)
- class [Light](#)
Class for light.
- struct [LightPushConstantData](#)
- class [Model](#)
Class for model.

- class [Object](#)
Class for object.
- struct [ObjectPushConstantData](#)
- class [ObjectRenderSystem](#)
Class for object render system.
- struct [PipelineConfigInfo](#)
- struct [PointLightData](#)
- class [PointLightRenderSystem](#)
Class for point light system.
- struct [QueueFamilyIndices](#)
- class [Renderer](#)
Class for renderer.
- class [Shaders](#)
Class for shaders.
- class [SwapChain](#)
Class for swap chain.
- struct [SwapChainSupportDetails](#)
- class [Transform3DComponent](#)
Class for 3D transformation.
- class [Window](#)
Class for window.

Enumerations

- enum [ENGINE_STATE](#) : uint8_t { [EDITOR](#) = 0 , [GAME](#) = 1 , [PAUSED](#) = 2 , [EXIT](#) = 3 }
- enum [GUI_STATE](#) : uint8_t { [VISIBLE](#) = 0 , [HIDDEN](#) = 1 }

Functions

- template<typename T , typename... Rest>
void [hashCombine](#) (std::size_t &seed, const T &v, const Rest &... rest)

Variables

- static constexpr glm::vec3 [DEFAULT_POSITION](#) {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 [DEFAULT_ROTATION](#) {0.F, 0.F, 0.F}
- static constexpr float [DEFAULT_FOV](#) = glm::radians(50.0F)
- static constexpr float [DEFAULT_NEAR](#) = 0.1F
- static constexpr float [DEFAULT_FAR](#) = 100.F
- static constexpr float [DEFAULT_MOVE_SPEED](#) = 3.F
- static constexpr float [DEFAULT_LOOK_SPEED](#) = 1.5F
- static constexpr float [COLOR_MAX](#) = 255.0F
- static constexpr uint32_t [DEFAULT_MAX_SETS](#) = 1000
- static constexpr uint16_t [MAX_LIGHTS](#) = 10
- static constexpr float [DEFAULT_AMBIENT_LIGHT_INTENSITY](#) = .2F
- static constexpr glm::vec4 [DEFAULT_AMBIENT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_AMBIENT_LIGHT_INTENSITY](#)}
- static constexpr float [DEFAULT_LIGHT_INTENSITY](#) = .2F
- static constexpr float [DEFAULT_LIGHT_RADIUS](#) = 0.1F
- static constexpr glm::vec4 [DEFAULT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_LIGHT_INTENSITY](#)}
- static constexpr VkClearColorValue [DEFAULT_CLEAR_COLOR](#) = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue [DEFAULT_CLEAR_DEPTH](#) = {1.0F, 0}
- static constexpr std::string_view [SHADERS_BIN_PATH](#) = "build/shaders/"
- static constexpr int [MAX_FRAMES_IN_FLIGHT](#) = 2
- static constexpr uint32_t [DEFAULT_WIDTH](#) = 1920
- static constexpr uint32_t [DEFAULT_HEIGHT](#) = 1080
- static constexpr std::string_view [DEFAULT_TITLE](#) = "VEngine"

6.1.1 Enumeration Type Documentation

6.1.1.1 ENGINE_STATE

```
enum ven::ENGINE_STATE : uint8_t
```

Enumerator

EDITOR	
GAME	
PAUSED	
EXIT	

Definition at line 21 of file [Engine.hpp](#).

6.1.1.2 GUI_STATE

```
enum ven::GUI_STATE : uint8_t
```

Enumerator

VISIBLE	
HIDDEN	

Definition at line 18 of file [Gui.hpp](#).

6.1.2 Function Documentation

6.1.2.1 hashCombine()

```
template<typename T , typename... Rest>
void ven::hashCombine (
    std::size_t & seed,
    const T & v,
    const Rest &... rest)
```

Definition at line 14 of file [Utils.hpp](#).

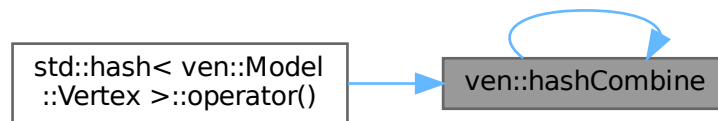
References [hashCombine\(\)](#).

Referenced by [hashCombine\(\)](#), and [std::hash< ven::Model::Vertex >::operator\(\)\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3 Variable Documentation

6.1.3.1 COLOR_MAX

```
float ven::COLOR_MAX = 255.0F [static], [constexpr]
```

Definition at line 15 of file [Colors.hpp](#).

6.1.3.2 DEFAULT_AMBIENT_LIGHT_COLOR

```
glm::vec4 ven::DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENSITY}
[static], [constexpr]
```

Definition at line 20 of file [FrameInfo.hpp](#).

6.1.3.3 DEFAULT_AMBIENT_LIGHT_INTENSITY

```
float ven::DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F [static], [constexpr]
```

Definition at line 19 of file [FrameInfo.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

6.1.3.4 DEFAULT_CLEAR_COLOR

```
VkClearColorValue ven::DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 20 of file [Renderer.hpp](#).

6.1.3.5 DEFAULT_CLEAR_DEPTH

```
VkClearDepthStencilValue ven::DEFAULT_CLEAR_DEPTH = {1.0F, 0} [static], [constexpr]
```

Definition at line 21 of file [Renderer.hpp](#).

6.1.3.6 DEFAULT_FAR

```
float ven::DEFAULT_FAR = 100.F [static], [constexpr]
```

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

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.7 DEFAULT_FOV

```
float ven::DEFAULT_FOV = glm::radians(50.0F) [static], [constexpr]
```

Definition at line 18 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.8 DEFAULT_HEIGHT

```
uint32_t ven::DEFAULT_HEIGHT = 1080 [static], [constexpr]
```

Definition at line 18 of file [Window.hpp](#).

6.1.3.9 DEFAULT_LIGHT_COLOR

```
glm::vec4 ven::DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY} [static],  
[constexpr]
```

Definition at line 20 of file [Light.hpp](#).

6.1.3.10 DEFAULT_LIGHT_INTENSITY

```
float ven::DEFAULT_LIGHT_INTENSITY = .2F [static], [constexpr]
```

Definition at line 18 of file [Light.hpp](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

6.1.3.11 DEFAULT_LIGHT_RADIUS

```
float ven::DEFAULT_LIGHT_RADIUS = 0.1F [static], [constexpr]
```

Definition at line 19 of file [Light.hpp](#).

6.1.3.12 DEFAULT_LOOK_SPEED

```
float ven::DEFAULT_LOOK_SPEED = 1.5F [static], [constexpr]
```

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

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.13 DEFAULT_MAX_SETS

```
uint32_t ven::DEFAULT_MAX_SETS = 1000 [static], [constexpr]
```

Definition at line 15 of file [DescriptorPool.hpp](#).

6.1.3.14 DEFAULT_MOVE_SPEED

```
float ven::DEFAULT_MOVE_SPEED = 3.F [static], [constexpr]
```

Definition at line 22 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.15 DEFAULT_NEAR

```
float ven::DEFAULT_NEAR = 0.1F [static], [constexpr]
```

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

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.16 DEFAULT_POSITION

```
glm::vec3 ven::DEFAULT_POSITION {0.F, 0.F, -2.5F} [static], [constexpr]
```

Definition at line 15 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.17 DEFAULT_ROTATION

```
glm::vec3 ven::DEFAULT_ROTATION {0.F, 0.F, 0.F} [static], [constexpr]
```

Definition at line 16 of file [Camera.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

6.1.3.18 DEFAULT_TITLE

```
std::string_view ven::DEFAULT_TITLE = "VEngine" [static], [constexpr]
```

Definition at line 19 of file [Window.hpp](#).

6.1.3.19 DEFAULT_WIDTH

```
uint32_t ven::DEFAULT_WIDTH = 1920 [static], [constexpr]
```

Definition at line 17 of file [Window.hpp](#).

6.1.3.20 MAX_FRAMES_IN_FLIGHT

```
int ven::MAX_FRAMES_IN_FLIGHT = 2 [static], [constexpr]
```

Definition at line 16 of file [SwapChain.hpp](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::SwapChain::createSyncObjects\(\)](#), [ven::Renderer::endFrame\(\)](#), [ven::Engine::Engine\(\)](#), [ven::Engine::mainLoop\(\)](#), [ven::SwapChain::submitCommandBuffers\(\)](#), and [ven::SwapChain::~~SwapChain\(\)](#).

6.1.3.21 MAX_LIGHTS

```
uint16_t ven::MAX_LIGHTS = 10 [static], [constexpr]
```

Definition at line 17 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::update\(\)](#).

6.1.3.22 SHADERS_BIN_PATH

```
std::string_view ven::SHADERS_BIN_PATH = "build/shaders/" [static], [constexpr]
```

Definition at line 17 of file [Shaders.hpp](#).

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

Chapter 7

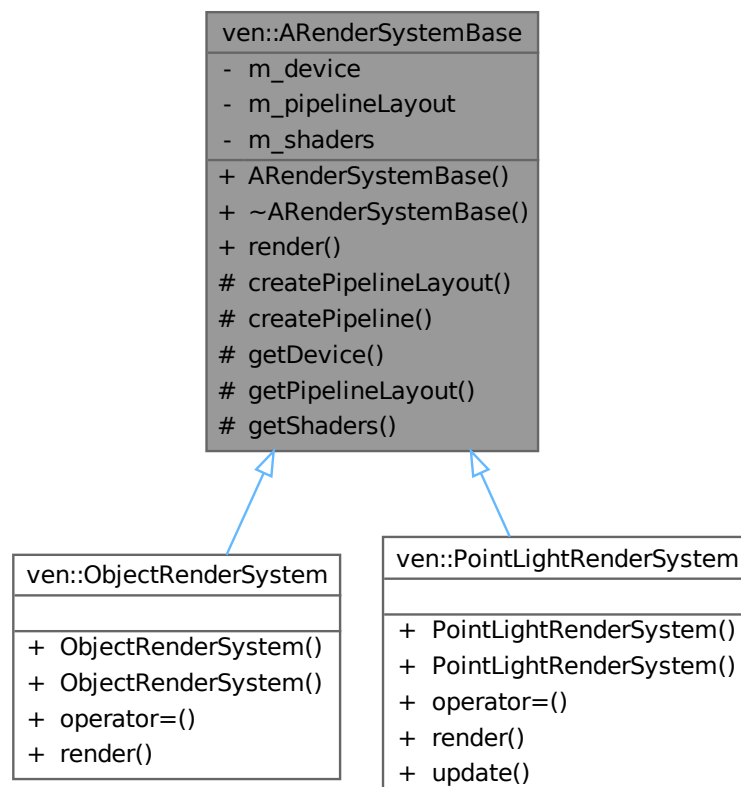
Class Documentation

7.1 ven::ARenderSystemBase Class Reference

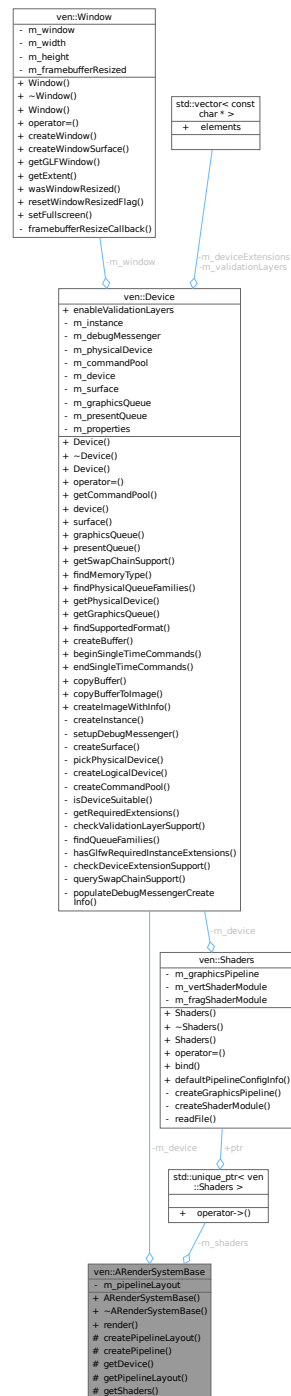
Abstract class for render system base.

```
#include <ARenderSystemBase.hpp>
```

Inheritance diagram for ven::ARenderSystemBase:



Collaboration diagram for ven::ARenderSystemBase:



Public Member Functions

- [ARenderSystemBase \(Device &device\)](#)
- virtual `~ARenderSystemBase ()`
- virtual void `render (const FrameInfo &frameInfo) const =0`

Protected Member Functions

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

Private Attributes

- [Device](#) & [m_device](#)
- VkPipelineLayout [m_pipelineLayout](#) {nullptr}
- std::unique_ptr< [Shaders](#) > [m_shaders](#)

7.1.1 Detailed Description

Abstract class for render system base.

Definition at line 22 of file [ARenderSystemBase.hpp](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ARenderSystemBase()

```
ven::ARenderSystemBase::ARenderSystemBase (
    Device & device)    [inline], [explicit]
```

Definition at line 26 of file [ARenderSystemBase.hpp](#).

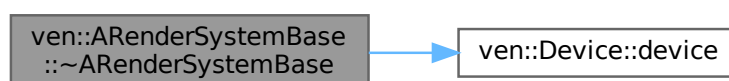
7.1.2.2 ~ARenderSystemBase()

```
virtual ven::ARenderSystemBase::~~ARenderSystemBase ()    [inline], [virtual]
```

Definition at line 27 of file [ARenderSystemBase.hpp](#).

References [ven::Device::device\(\)](#), [m_device](#), and [m_pipelineLayout](#).

Here is the call graph for this function:



7.1.3 Member Function Documentation

7.1.3.1 createPipeline()

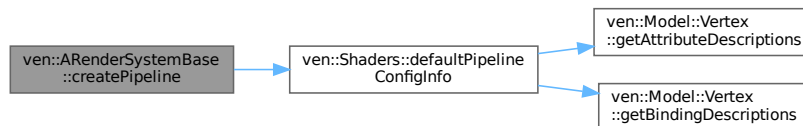
```
void ven::ARenderSystemBase::createPipeline (
    VkRenderPass renderPass,
    const std::string & shadersVertPath,
    const std::string & shadersFragPath,
    bool isLight) [protected]
```

Definition at line 26 of file [renderSystemBase.cpp](#).

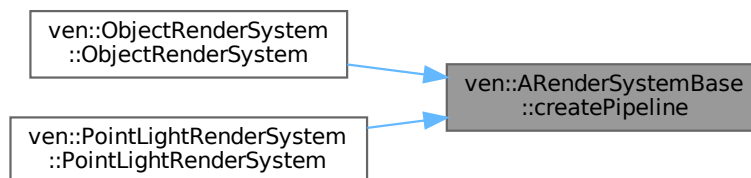
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.2 createPipelineLayout()

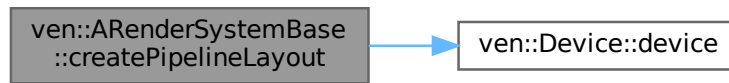
```
void ven::ARenderSystemBase::createPipelineLayout (
    VkDescriptorSetLayout globalSetLayout,
    uint32_t pushConstantSize) [protected]
```

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

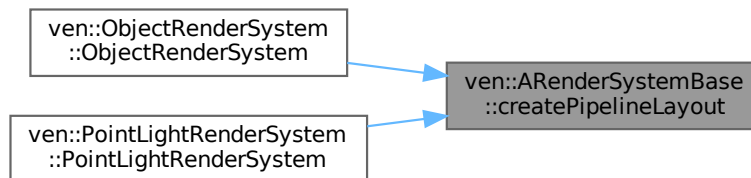
References [ven::Device::device\(\)](#), `m_device`, and `m_pipelineLayout`.

Referenced by [ven::ObjectRenderSystem::ObjectRenderSystem\(\)](#), and [ven::PointLightRenderSystem::PointLightRenderSystem\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.3 getDevice()

```
Device & ven::ARenderSystemBase::getDevice () const [inline], [nodiscard], [protected]
```

Definition at line 36 of file [ARenderSystemBase.hpp](#).

References [m_device](#).

7.1.3.4 getPipelineLayout()

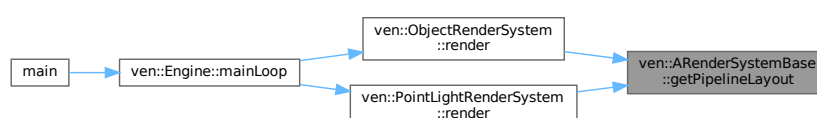
```
VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]
```

Definition at line 37 of file [ARenderSystemBase.hpp](#).

References [m_pipelineLayout](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

Here is the caller graph for this function:



7.1.3.5 getShaders()

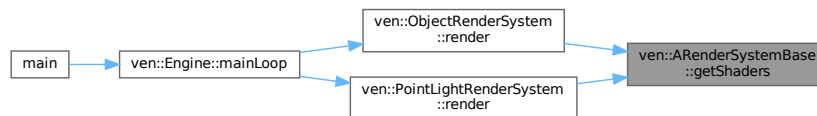
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],
[nodiscard], [protected]
```

Definition at line 38 of file [ARenderSystemBase.hpp](#).

References [m_shaders](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

Here is the caller graph for this function:



7.1.3.6 render()

```
virtual void ven::ARenderSystemBase::render (
    const FrameInfo & frameInfo) const [pure virtual]
```

Implemented in [ven::ObjectRenderSystem](#), and [ven::PointLightRenderSystem](#).

7.1.4 Member Data Documentation

7.1.4.1 m_device

```
Device& ven::ARenderSystemBase::m_device [private]
```

Definition at line 42 of file [ARenderSystemBase.hpp](#).

Referenced by [createPipelineLayout\(\)](#), [getDevice\(\)](#), and [~ARenderSystemBase\(\)](#).

7.1.4.2 m_pipelineLayout

```
VkPipelineLayout ven::ARenderSystemBase::m_pipelineLayout {nullptr} [private]
```

Definition at line 43 of file [ARenderSystemBase.hpp](#).

Referenced by [createPipelineLayout\(\)](#), [getPipelineLayout\(\)](#), and [~ARenderSystemBase\(\)](#).

7.1.4.3 m_shaders

```
std::unique_ptr<Shaders> ven::ARenderSystemBase::m_shaders [private]
```

Definition at line 44 of file [ARenderSystemBase.hpp](#).

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

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

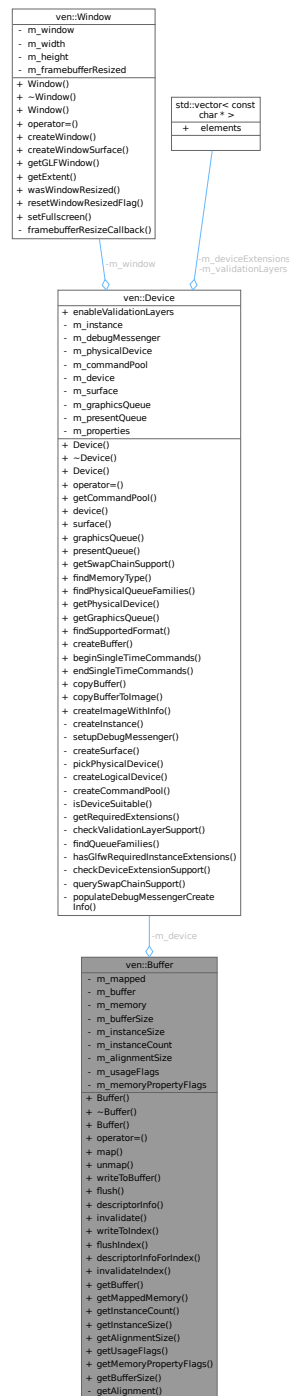
- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/[ARenderSystemBase.hpp](#)
- /home/runner/work/VEngine/VEngine/src/system/[renderSystemBase.cpp](#)

7.2 ven::Buffer Class Reference

Class for buffer.

```
#include <Buffer.hpp>
```

Collaboration diagram for ven::Buffer:



Public Member Functions

- [Buffer](#) ([Device](#) &device, VkDeviceSize instanceSize, uint32_t instanceCount, VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize minOffsetAlignment=1)
- [~Buffer](#) ()
- [Buffer](#) (const [Buffer](#) &)=delete
- [Buffer](#) & [operator=](#) (const [Buffer](#) &)=delete

- `VkResult` `map` (`VkDeviceSize` size=`VK_WHOLE_SIZE`, `VkDeviceSize` offset=0)
Map a memory range of this buffer.
- `void` `unmap` ()
Unmap a mapped memory range.
- `void` `writeToBuffer` (`const void *`data, `VkDeviceSize` size=`VK_WHOLE_SIZE`, `VkDeviceSize` offset=0) `const`
Copies the specified data to the mapped buffer.
- `VkResult` `flush` (`VkDeviceSize` size=`VK_WHOLE_SIZE`, `VkDeviceSize` offset=0) `const`
Flush a memory range of the buffer to make it visible to the device.
- `VkDescriptorBufferInfo` `descriptorInfo` (`const VkDeviceSize` size=`VK_WHOLE_SIZE`, `const VkDeviceSize` offset=0) `const`
Create a buffer info descriptor.
- `VkResult` `invalidate` (`VkDeviceSize` size=`VK_WHOLE_SIZE`, `VkDeviceSize` offset=0) `const`
Invalidate a memory range of the buffer to make it visible to the host.
- `void` `writeToIndex` (`const void *`data, `const VkDeviceSize` index) `const`
*Copies "instanceSize" bytes of data to the mapped buffer at an offset of index * alignmentSize.*
- `VkResult` `flushIndex` (`const VkDeviceSize` index) `const`
*Flush the memory range at index * alignmentSize of the buffer to make it visible to the device.*
- `VkDescriptorBufferInfo` `descriptorInfoForIndex` (`const VkDeviceSize` index) `const`
Create a buffer info descriptor.
- `VkResult` `invalidateIndex` (`const VkDeviceSize` index) `const`
Invalidate a memory range of the buffer to make it visible to the host.
- `VkBuffer` `getBuffer` () `const`
- `void *` `getMappedMemory` () `const`
- `uint32_t` `getInstanceCount` () `const`
- `VkDeviceSize` `getInstanceSize` () `const`
- `VkDeviceSize` `getAlignmentSize` () `const`
- `VkBufferUsageFlags` `getUsageFlags` () `const`
- `VkMemoryPropertyFlags` `getMemoryPropertyFlags` () `const`
- `VkDeviceSize` `getBufferSize` () `const`

Static Private Member Functions

- `static VkDeviceSize` `getAlignment` (`VkDeviceSize` instanceSize, `VkDeviceSize` minOffsetAlignment)
Returns the minimum instance size required to be compatible with devices minOffsetAlignment.

Private Attributes

- `Device` & `m_device`
- `void *` `m_mapped` = `nullptr`
- `VkBuffer` `m_buffer` = `VK_NULL_HANDLE`
- `VkDeviceMemory` `m_memory` = `VK_NULL_HANDLE`
- `VkDeviceSize` `m_bufferSize`
- `VkDeviceSize` `m_instanceSize`
- `uint32_t` `m_instanceCount`
- `VkDeviceSize` `m_alignmentSize`
- `VkBufferUsageFlags` `m_usageFlags`
- `VkMemoryPropertyFlags` `m_memoryPropertyFlags`

7.2.1 Detailed Description

Class for buffer.

Definition at line 18 of file [Buffer.hpp](#).

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Buffer() [1/2]

```
ven::Buffer::Buffer (
    Device & device,
    VkDeviceSize instanceSize,
    uint32_t instanceCount,
    VkBufferUsageFlags usageFlags,
    VkMemoryPropertyFlags memoryPropertyFlags,
    VkDeviceSize minOffsetAlignment = 1)
```

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

References [ven::Device::createBuffer\(\)](#), [m_alignmentSize](#), [m_buffer](#), [m_bufferSize](#), [m_instanceCount](#), [m_memory](#), [m_memoryPropertyFlags](#), and [m_usageFlags](#).

Here is the call graph for this function:



7.2.2.2 ~Buffer()

```
ven::Buffer::~~Buffer ()
```

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

7.2.2.3 Buffer() [2/2]

```
ven::Buffer::Buffer (
    const Buffer & ) [delete]
```

7.2.3 Member Function Documentation

7.2.3.1 descriptorInfo()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfo (
    const VkDeviceSize size = VK_WHOLE_SIZE,
    const VkDeviceSize offset = 0) const [inline], [nodiscard]
```

Create a buffer info descriptor.

Parameters

<i>size</i>	(Optional) Size of the memory range of the descriptor
<i>offset</i>	(Optional) Byte offset from beginning

Returns

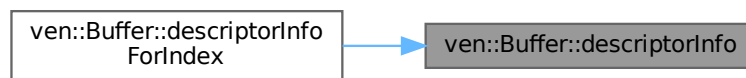
VkDescriptorBufferInfo of specified offset and range

Definition at line 74 of file [Buffer.hpp](#).

References [m_buffer](#).

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

Here is the caller graph for this function:



7.2.3.2 descriptorInfoForIndex()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfoForIndex (
    const VkDeviceSize index) const [inline], [nodiscard]
```

Create a buffer info descriptor.

Parameters

<i>index</i>	Specifies the region given by <code>index * alignmentSize</code>
--------------	--

Returns

VkDescriptorBufferInfo for instance at index

Definition at line 113 of file [Buffer.hpp](#).

References [descriptorInfo\(\)](#), and [m_alignmentSize](#).

Here is the call graph for this function:



7.2.3.3 flush()

```
VkResult ven::Buffer::flush (
    VkDeviceSize size = VK_WHOLE_SIZE,
    VkDeviceSize offset = 0) const [nodiscard]
```

Flush a memory range of the buffer to make it visible to the device.

Note

Only required for non-coherent memory

Parameters

<i>size</i>	(Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush the complete buffer range.
<i>offset</i>	(Optional) Byte offset from beginning

Returns

VkResult of the flush call

Definition at line 53 of file [buffer.cpp](#).

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

Here is the caller graph for this function:



7.2.3.4 flushIndex()

```
VkResult ven::Buffer::flushIndex (
    const VkDeviceSize index) const [inline], [nodiscard]
```

Flush the memory range at index * alignmentSize of the buffer to make it visible to the device.

Parameters

<i>index</i>	Used in offset calculation
--------------	----------------------------

Definition at line 103 of file [Buffer.hpp](#).

References [flush\(\)](#), and [m_alignmentSize](#).

Here is the call graph for this function:



7.2.3.5 getAlignment()

```

VkDeviceSize ven::Buffer::getAlignment (
    VkDeviceSize instanceSize,
    VkDeviceSize minOffsetAlignment) [static], [private]
  
```

Returns the minimum instance size required to be compatible with devices minOffsetAlignment.

Parameters

<i>instanceSize</i>	The size of an instance
<i>minOffsetAlignment</i>	The minimum required alignment, in bytes, for the offset member (eg minUniformBufferOffsetAlignment)

Returns

VkResult of the buffer mapping call

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

7.2.3.6 getAlignmentSize()

```

VkDeviceSize ven::Buffer::getAlignmentSize () const [inline], [nodiscard]
  
```

Definition at line 130 of file [Buffer.hpp](#).

References [m_alignmentSize](#).

7.2.3.7 getBuffer()

```
VkBuffer ven::Buffer::getBuffer () const [inline], [nodiscard]
```

Definition at line 126 of file [Buffer.hpp](#).

References [m_buffer](#).

7.2.3.8 getBufferSize()

```
VkDeviceSize ven::Buffer::getBufferSize () const [inline], [nodiscard]
```

Definition at line 133 of file [Buffer.hpp](#).

References [m_bufferSize](#).

7.2.3.9 getInstanceCount()

```
uint32_t ven::Buffer::getInstanceCount () const [inline], [nodiscard]
```

Definition at line 128 of file [Buffer.hpp](#).

References [m_instanceCount](#).

7.2.3.10 getInstanceSize()

```
VkDeviceSize ven::Buffer::getInstanceSize () const [inline], [nodiscard]
```

Definition at line 129 of file [Buffer.hpp](#).

References [m_instanceSize](#).

7.2.3.11 getMappedMemory()

```
void * ven::Buffer::getMappedMemory () const [inline], [nodiscard]
```

Definition at line 127 of file [Buffer.hpp](#).

References [m_mapped](#).

7.2.3.12 getMemoryPropertyFlags()

```
VkMemoryPropertyFlags ven::Buffer::getMemoryPropertyFlags () const [inline], [nodiscard]
```

Definition at line 132 of file [Buffer.hpp](#).

References [m_memoryPropertyFlags](#).

7.2.3.13 getUsageFlags()

```
VkBufferUsageFlags ven::Buffer::getUsageFlags () const [inline], [nodiscard]
```

Definition at line 131 of file [Buffer.hpp](#).

References [m_usageFlags](#).

7.2.3.14 invalidate()

```
VkResult ven::Buffer::invalidate (
    VkDeviceSize size = VK_WHOLE_SIZE,
    VkDeviceSize offset = 0) const [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host.

Note

Only required for non-coherent memory

Parameters

<i>size</i>	(Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to invalidate the complete buffer range.
<i>offset</i>	(Optional) Byte offset from beginning

Returns

VkResult of the invalidate call

Definition at line 63 of file [buffer.cpp](#).

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

Here is the caller graph for this function:



7.2.3.15 invalidateIndex()

```
VkResult ven::Buffer::invalidateIndex (
    const VkDeviceSize index) const [inline], [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host.

Note

Only required for non-coherent memory

Parameters

<i>index</i>	Specifies the region to invalidate: $\text{index} * \text{alignmentSize}$
--------------	---

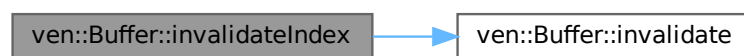
Returns

VkResult of the invalidate call

Definition at line 124 of file [Buffer.hpp](#).

References [invalidate\(\)](#), and [m_alignmentSize](#).

Here is the call graph for this function:



7.2.3.16 map()

```
VkResult ven::Buffer::map (
    VkDeviceSize size = VK_WHOLE_SIZE,
    VkDeviceSize offset = 0)
```

Map a memory range of this buffer.

If successful, mapped points to the specified buffer range.

Parameters

<i>size</i>	(Optional) Size of the memory range to map. Pass <code>VK_WHOLE_SIZE</code> to map the complete buffer range.
<i>offset</i>	(Optional) Byte offset from beginning

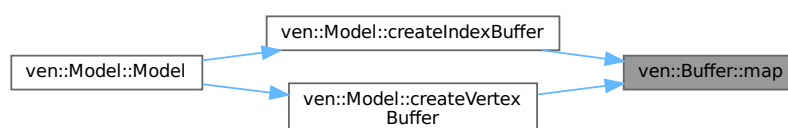
Returns

VkResult of the buffer mapping call

Definition at line 26 of file [buffer.cpp](#).

Referenced by [ven::Model::createIndexBuffer\(\)](#), and [ven::Model::createVertexBuffer\(\)](#).

Here is the caller graph for this function:



7.2.3.17 operator=()

```
Buffer & ven::Buffer::operator= (
    const Buffer & ) [delete]
```

7.2.3.18 unmap()

```
void ven::Buffer::unmap ()
```

Unmap a mapped memory range.

Note

Does not return a result as vkUnmapMemory can't fail

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

7.2.3.19 writeToBuffer()

```
void ven::Buffer::writeToBuffer (
    const void * data,
    VkDeviceSize size = VK_WHOLE_SIZE,
    VkDeviceSize offset = 0) const
```

Copies the specified data to the mapped buffer.

Default value writes whole buffer range

Parameters

<i>data</i>	Pointer to the data to copy
<i>size</i>	(Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the complete buffer range.
<i>offset</i>	(Optional) Byte offset from beginning of mapped region

Definition at line 40 of file [buffer.cpp](#).

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

Here is the caller graph for this function:



7.2.3.20 writeToIndex()

```
void ven::Buffer::writeToIndex (
    const void * data,
    const VkDeviceSize index) const [inline]
```

Copies "instanceSize" bytes of data to the mapped buffer at an offset of index * alignmentSize.

Parameters

<i>data</i>	Pointer to the data to copy
<i>index</i>	Used in offset calculation

Definition at line 96 of file [Buffer.hpp](#).

References [m_alignmentSize](#), [m_instanceSize](#), and [writeToBuffer\(\)](#).

Here is the call graph for this function:



7.2.4 Member Data Documentation

7.2.4.1 m_alignmentSize

```
VkDeviceSize ven::Buffer::m_alignmentSize [private]
```

Definition at line 155 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), [descriptorInfoForIndex\(\)](#), [flushIndex\(\)](#), [getAlignmentSize\(\)](#), [invalidateIndex\(\)](#), and [writeToIndex\(\)](#).

7.2.4.2 m_buffer

```
VkBuffer ven::Buffer::m_buffer = VK_NULL_HANDLE [private]
```

Definition at line 149 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), [descriptorInfo\(\)](#), and [getBuffer\(\)](#).

7.2.4.3 m_bufferSize

```
VkDeviceSize ven::Buffer::m_bufferSize [private]
```

Definition at line 152 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getBufferSize\(\)](#).

7.2.4.4 m_device

```
Device& ven::Buffer::m_device [private]
```

Definition at line 147 of file [Buffer.hpp](#).

7.2.4.5 m_instanceCount

```
uint32_t ven::Buffer::m_instanceCount [private]
```

Definition at line 154 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getInstanceCount\(\)](#).

7.2.4.6 m_instanceSize

```
VkDeviceSize ven::Buffer::m_instanceSize [private]
```

Definition at line 153 of file [Buffer.hpp](#).

Referenced by [getInstanceSize\(\)](#), and [writeToIndex\(\)](#).

7.2.4.7 m_mapped

```
void* ven::Buffer::m_mapped = nullptr [private]
```

Definition at line 148 of file [Buffer.hpp](#).

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

7.2.4.8 m_memory

```
VkDeviceMemory ven::Buffer::m_memory = VK_NULL_HANDLE [private]
```

Definition at line 150 of file [Buffer.hpp](#).

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

7.2.4.9 m_memoryPropertyFlags

```
VkMemoryPropertyFlags ven::Buffer::m_memoryPropertyFlags [private]
```

Definition at line 157 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getMemoryPropertyFlags\(\)](#).

7.2.4.10 m_usageFlags

```
VkBufferUsageFlags ven::Buffer::m_usageFlags [private]
```

Definition at line 156 of file [Buffer.hpp](#).

Referenced by [Buffer\(\)](#), and [getUsageFlags\(\)](#).

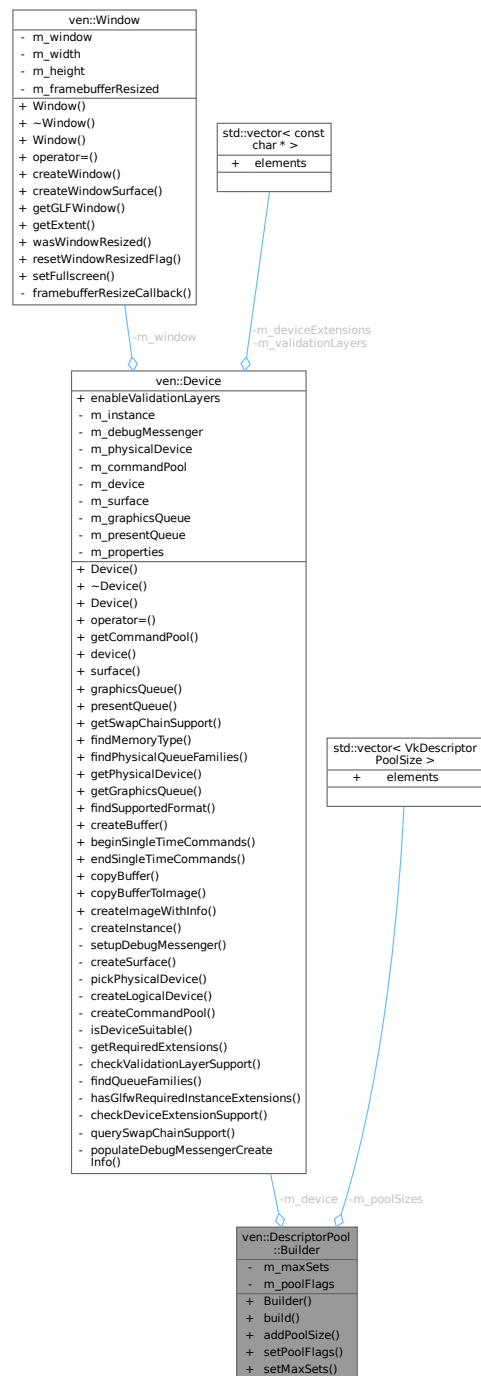
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/buffer.cpp](#)

7.3 ven::DescriptorPool::Builder Class Reference

```
#include <DescriptorPool.hpp>
```


Collaboration diagram for ven::DescriptorPool::Builder:



Public Member Functions

- [Builder](#) ([Device](#) &device)
- `std::unique_ptr< DescriptorPool > build ()` const
- [Builder](#) & [addPoolSize](#) (const `VkDescriptorType` descriptorType, const `uint32_t` count)
- [Builder](#) & [setPoolFlags](#) (const `VkDescriptorPoolCreateFlags` flags)
- [Builder](#) & [setMaxSets](#) (const `uint32_t` count)

Private Attributes

- [Device](#) & [m_device](#)
- `std::vector< VkDescriptorPoolSize >` [m_poolSizes](#)
- `uint32_t` [m_maxSets](#) {`DEFAULT_MAX_SETS`}
- `VkDescriptorPoolCreateFlags` [m_poolFlags](#) {0}

7.3.1 Detailed Description

Definition at line 26 of file [DescriptorPool.hpp](#).

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Builder()

```
ven::DescriptorPool::Builder::Builder (
    Device & device) [inline], [explicit]
```

Definition at line 30 of file [DescriptorPool.hpp](#).

7.3.3 Member Function Documentation

7.3.3.1 addPoolSize()

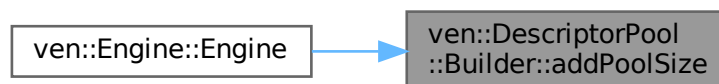
```
Builder & ven::DescriptorPool::Builder::addPoolSize (
    const VkDescriptorType descriptorType,
    const uint32_t count) [inline]
```

Definition at line 34 of file [DescriptorPool.hpp](#).

References [m_poolSizes](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.3.3.2 build()

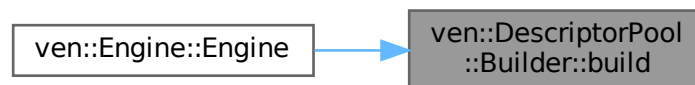
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],
[nodiscard]
```

Definition at line 32 of file [DescriptorPool.hpp](#).

References [m_device](#), [m_maxSets](#), [m_poolFlags](#), and [m_poolSizes](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.3.3.3 setMaxSets()

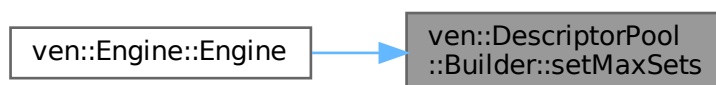
```
Builder & ven::DescriptorPool::Builder::setMaxSets (
    const uint32_t count) [inline]
```

Definition at line 36 of file [DescriptorPool.hpp](#).

References [m_maxSets](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.3.3.4 setPoolFlags()

```
Builder & ven::DescriptorPool::Builder::setPoolFlags (
    const VkDescriptorPoolCreateFlags flags) [inline]
```

Definition at line 35 of file [DescriptorPool.hpp](#).

References [m_poolFlags](#).

7.3.4 Member Data Documentation

7.3.4.1 m_device

`Device& ven::DescriptorPool::Builder::m_device` [private]

Definition at line 40 of file [DescriptorPool.hpp](#).

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

7.3.4.2 m_maxSets

`uint32_t ven::DescriptorPool::Builder::m_maxSets` {[DEFAULT_MAX_SETS](#)} [private]

Definition at line 42 of file [DescriptorPool.hpp](#).

Referenced by [build\(\)](#), and [setMaxSets\(\)](#).

7.3.4.3 m_poolFlags

`VkDescriptorPoolCreateFlags ven::DescriptorPool::Builder::m_poolFlags` {0} [private]

Definition at line 43 of file [DescriptorPool.hpp](#).

Referenced by [build\(\)](#), and [setPoolFlags\(\)](#).

7.3.4.4 m_poolSizes

`std::vector<VkDescriptorPoolSize> ven::DescriptorPool::Builder::m_poolSizes` [private]

Definition at line 41 of file [DescriptorPool.hpp](#).

Referenced by [addPoolSize\(\)](#), and [build\(\)](#).

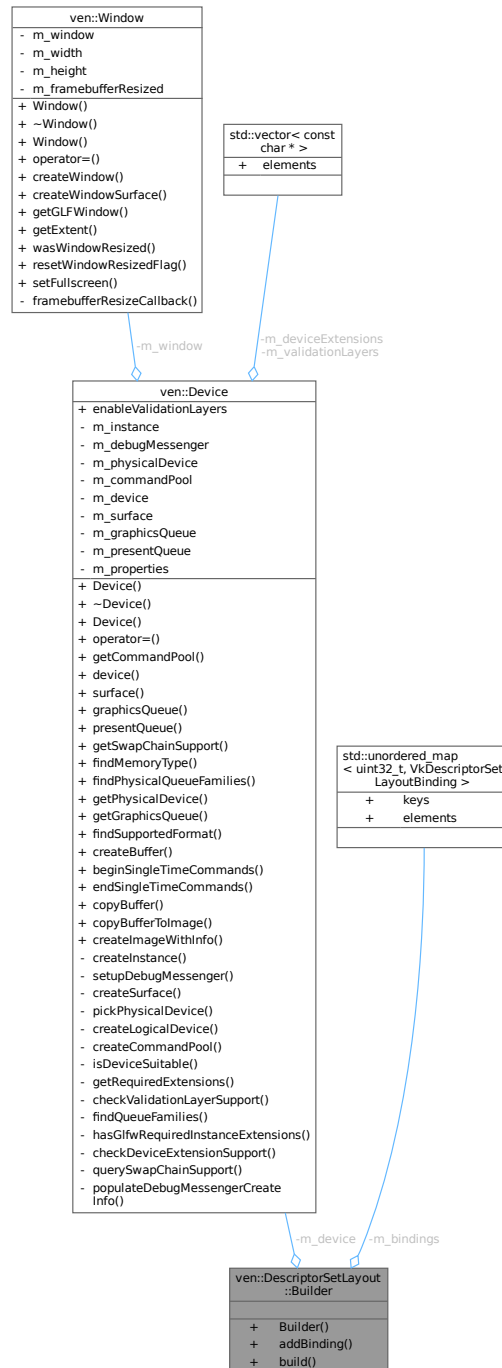
The documentation for this class was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#)

7.4 ven::DescriptorSetLayout::Builder Class Reference

```
#include <DescriptorSetLayout.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout::Builder:



Public Member Functions

- [Builder](#) ([Device](#) &device)

- [Builder](#) & [addBinding](#) (uint32_t binding, VkDescriptorType descriptorType, VkShaderStageFlags stageFlags, uint32_t count=1)
- std::unique_ptr< [DescriptorSetLayout](#) > [build](#) () const

Private Attributes

- [Device](#) & [m_device](#)
- std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > [m_bindings](#)

7.4.1 Detailed Description

Definition at line 25 of file [DescriptorSetLayout.hpp](#).

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Builder()

```
ven::DescriptorSetLayout::Builder::Builder (
    Device & device) [inline], [explicit]
```

Definition at line 29 of file [DescriptorSetLayout.hpp](#).

7.4.3 Member Function Documentation

7.4.3.1 addBinding()

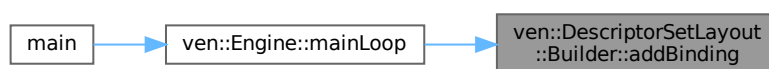
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

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

References [m_bindings](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.4.3.2 build()

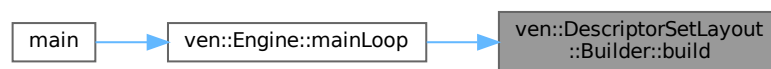
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

Definition at line 32 of file [DescriptorSetLayout.hpp](#).

References [m_bindings](#), and [m_device](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.4.4 Member Data Documentation

7.4.4.1 m_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorSetLayout::Builder↔
::m_bindings [private]
```

Definition at line 37 of file [DescriptorSetLayout.hpp](#).

Referenced by [addBinding\(\)](#), and [build\(\)](#).

7.4.4.2 m_device

```
Device& ven::DescriptorSetLayout::Builder::m_device [private]
```

Definition at line 36 of file [DescriptorSetLayout.hpp](#).

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

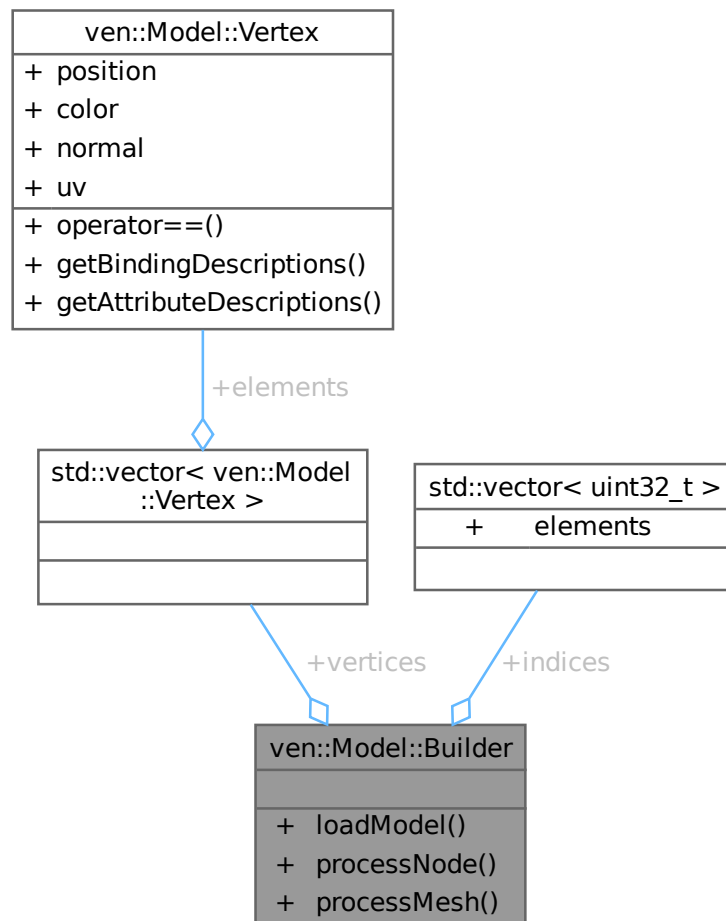
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#)

7.5 ven::Model::Builder Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model::Builder:



Public Member Functions

- void [loadModel](#) (const std::string &filename)
- void [processNode](#) (const aiNode *node, const aiScene *scene)
- void [processMesh](#) (const aiMesh *mesh, const aiScene *scene)

Public Attributes

- std::vector< [Vertex](#) > [vertices](#)
- std::vector< uint32_t > [indices](#)

7.5.1 Detailed Description

Definition at line 43 of file [Model.hpp](#).

7.5.2 Member Function Documentation

7.5.2.1 loadModel()

```
void ven::Model::Builder::loadModel (
    const std::string & filename)
```

Definition at line 117 of file [model.cpp](#).

Referenced by [ven::Model::createModelFromFile\(\)](#).

Here is the caller graph for this function:



7.5.2.2 processMesh()

```
void ven::Model::Builder::processMesh (
    const aiMesh * mesh,
    const aiScene * scene)
```

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

References [ven::Colors::BLACK_3](#), [ven::Model::Vertex::position](#), and [ven::Colors::WHITE_3](#).

7.5.2.3 processNode()

```
void ven::Model::Builder::processNode (
    const aiNode * node,
    const aiScene * scene)
```

Definition at line 132 of file [model.cpp](#).

7.5.3 Member Data Documentation

7.5.3.1 indices

```
std::vector<uint32_t> ven::Model::Builder::indices
```

Definition at line 45 of file [Model.hpp](#).

Referenced by [ven::Model::Model\(\)](#).

7.5.3.2 vertices

```
std::vector<Vertex> ven::Model::Builder::vertices
```

Definition at line 44 of file [Model.hpp](#).

Referenced by [ven::Model::Model\(\)](#).

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

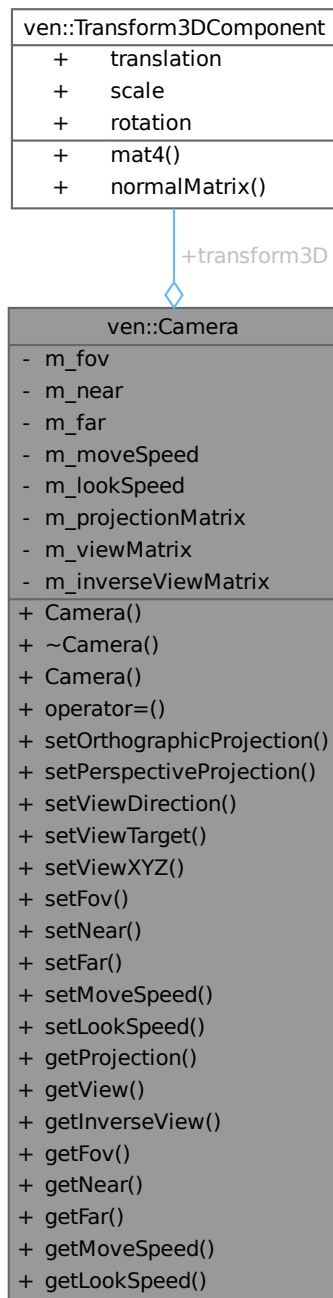
- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.6 ven::Camera Class Reference

Class for camera.

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

Collaboration diagram for ven::Camera:



Public Member Functions

- [Camera](#) ()=default
- [~Camera](#) ()=default
- [Camera](#) (const [Camera](#) &)=delete
- [Camera](#) & [operator=](#) (const [Camera](#) &)=delete
- void [setOrthographicProjection](#) (float left, float right, float top, float bottom, float near, float far)

- void [setPerspectiveProjection](#) (float aspect)
- void [setViewDirection](#) (glm::vec3 position, glm::vec3 direction, glm::vec3 up={0.F, -1.F, 0.F})
- void [setViewTarget](#) (const glm::vec3 position, const glm::vec3 target, const glm::vec3 up={0.F, -1.F, 0.F})
- void [setViewXYZ](#) (glm::vec3 position, glm::vec3 rotation)
- void [setFov](#) (const float fov)
- void [setNear](#) (const float near)
- void [setFar](#) (const float far)
- void [setMoveSpeed](#) (const float moveSpeed)
- void [setLookSpeed](#) (const float lookSpeed)
- const glm::mat4 & [getProjection](#) () const
- const glm::mat4 & [getView](#) () const
- const glm::mat4 & [getInverseView](#) () const
- float [getFov](#) () const
- float [getNear](#) () const
- float [getFar](#) () const
- float [getMoveSpeed](#) () const
- float [getLookSpeed](#) () const

Public Attributes

- [Transform3DComponent](#) transform3D {[DEFAULT_POSITION](#), {1.F, 1.F, 1.F}, [DEFAULT_ROTATION](#)}

Private Attributes

- float [m_fov](#) {[DEFAULT_FOV](#)}
- float [m_near](#) {[DEFAULT_NEAR](#)}
- float [m_far](#) {[DEFAULT_FAR](#)}
- float [m_moveSpeed](#) {[DEFAULT_MOVE_SPEED](#)}
- float [m_lookSpeed](#) {[DEFAULT_LOOK_SPEED](#)}
- glm::mat4 [m_projectionMatrix](#) {1.F}
- glm::mat4 [m_viewMatrix](#) {1.F}
- glm::mat4 [m_inverseViewMatrix](#) {1.F}

7.6.1 Detailed Description

Class for camera.

Definition at line 30 of file [Camera.hpp](#).

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Camera() [1/2]

```
ven::Camera::Camera () [default]
```

7.6.2.2 ~Camera()

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

7.6.2.3 Camera() [2/2]

```
ven::Camera::Camera (
    const Camera & ) [delete]
```

7.6.3 Member Function Documentation

7.6.3.1 getFar()

```
float ven::Camera::getFar () const [inline], [nodiscard]
```

Definition at line 56 of file [Camera.hpp](#).

References [m_far](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.2 getFov()

```
float ven::Camera::getFov () const [inline], [nodiscard]
```

Definition at line 54 of file [Camera.hpp](#).

References [m_fov](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.3 getInverseView()

```
const glm::mat4 & ven::Camera::getInverseView () const [inline], [nodiscard]
```

Definition at line 53 of file [Camera.hpp](#).

References [m_inverseViewMatrix](#).

7.6.3.4 getLookSpeed()

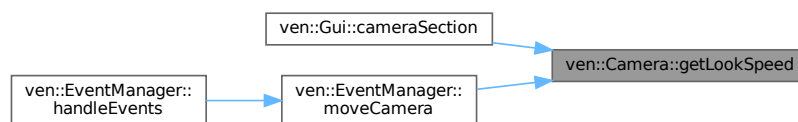
```
float ven::Camera::getLookSpeed () const [inline], [nodiscard]
```

Definition at line 58 of file [Camera.hpp](#).

References [m_lookSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



7.6.3.5 getMoveSpeed()

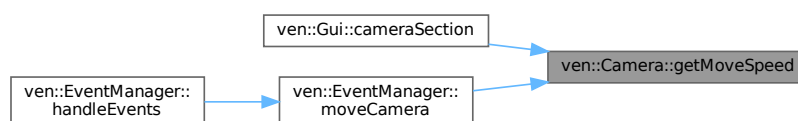
```
float ven::Camera::getMoveSpeed () const [inline], [nodiscard]
```

Definition at line 57 of file [Camera.hpp](#).

References [m_moveSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



7.6.3.6 getNear()

```
float ven::Camera::getNear () const [inline], [nodiscard]
```

Definition at line 55 of file [Camera.hpp](#).

References [m_near](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.7 getProjection()

```
const glm::mat4 & ven::Camera::getProjection () const [inline], [nodiscard]
```

Definition at line 51 of file [Camera.hpp](#).

References [m_projectionMatrix](#).

7.6.3.8 getView()

```
const glm::mat4 & ven::Camera::getView () const [inline], [nodiscard]
```

Definition at line 52 of file [Camera.hpp](#).

References [m_viewMatrix](#).

7.6.3.9 operator=()

```
Camera & ven::Camera::operator= (  
    const Camera & ) [delete]
```

7.6.3.10 setFar()

```
void ven::Camera::setFar (
    const float far) [inline]
```

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

References [m_far](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.11 setFov()

```
void ven::Camera::setFov (
    const float fov) [inline]
```

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

References [m_fov](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.12 setLookSpeed()

```
void ven::Camera::setLookSpeed (
    const float lookSpeed) [inline]
```

Definition at line 49 of file [Camera.hpp](#).

References [m_lookSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.13 setMoveSpeed()

```
void ven::Camera::setMoveSpeed (
    const float moveSpeed) [inline]
```

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

References [m_moveSpeed](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.14 setNear()

```
void ven::Camera::setNear (
    const float near) [inline]
```

Definition at line 46 of file [Camera.hpp](#).

References [m_near](#).

Referenced by [ven::Gui::cameraSection\(\)](#).

Here is the caller graph for this function:



7.6.3.15 setOrthographicProjection()

```
void ven::Camera::setOrthographicProjection (
    float left,
    float right,
    float top,
    float bottom,
    float near,
    float far)
```

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

References [m_projectionMatrix](#).

7.6.3.16 setPerspectiveProjection()

```
void ven::Camera::setPerspectiveProjection (
    float aspect)
```

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

7.6.3.17 setViewDirection()

```
void ven::Camera::setViewDirection (
    glm::vec3 position,
    glm::vec3 direction,
    glm::vec3 up = {0.F, -1.F, 0.F})
```

Definition at line 29 of file [camera.cpp](#).

7.6.3.18 setViewTarget()

```
void ven::Camera::setViewTarget (
    const glm::vec3 position,
    const glm::vec3 target,
    const glm::vec3 up = {0.F, -1.F, 0.F}) [inline]
```

Definition at line 43 of file [Camera.hpp](#).

7.6.3.19 setViewXYZ()

```
void ven::Camera::setViewXYZ (
    glm::vec3 position,
    glm::vec3 rotation)
```

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

7.6.4 Member Data Documentation

7.6.4.1 m_far

```
float ven::Camera::m_far {DEFAULT_FAR} [private]
```

Definition at line 66 of file [Camera.hpp](#).

Referenced by [getFar\(\)](#), and [setFar\(\)](#).

7.6.4.2 m_fov

```
float ven::Camera::m_fov {DEFAULT_FOV} [private]
```

Definition at line 64 of file [Camera.hpp](#).

Referenced by [getFov\(\)](#), and [setFov\(\)](#).

7.6.4.3 m_inverseViewMatrix

```
glm::mat4 ven::Camera::m_inverseViewMatrix {1.F} [private]
```

Definition at line 71 of file [Camera.hpp](#).

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

7.6.4.4 m_lookSpeed

```
float ven::Camera::m_lookSpeed {DEFAULT_LOOK_SPEED} [private]
```

Definition at line 68 of file [Camera.hpp](#).

Referenced by [getLookSpeed\(\)](#), and [setLookSpeed\(\)](#).

7.6.4.5 m_moveSpeed

```
float ven::Camera::m_moveSpeed {DEFAULT_MOVE_SPEED} [private]
```

Definition at line 67 of file [Camera.hpp](#).

Referenced by [getMoveSpeed\(\)](#), and [setMoveSpeed\(\)](#).

7.6.4.6 m_near

```
float ven::Camera::m_near {DEFAULT_NEAR} [private]
```

Definition at line 65 of file [Camera.hpp](#).

Referenced by [getNear\(\)](#), and [setNear\(\)](#).

7.6.4.7 m_projectionMatrix

```
glm::mat4 ven::Camera::m_projectionMatrix {1.F} [private]
```

Definition at line 69 of file [Camera.hpp](#).

Referenced by [getProjection\(\)](#), and [setOrthographicProjection\(\)](#).

7.6.4.8 m_viewMatrix

```
glm::mat4 ven::Camera::m_viewMatrix {1.F} [private]
```

Definition at line 70 of file [Camera.hpp](#).

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

7.6.4.9 transform3D

```
Transform3DComponent ven::Camera::transform3D {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}
```

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

Referenced by [ven::Gui::cameraSection\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

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

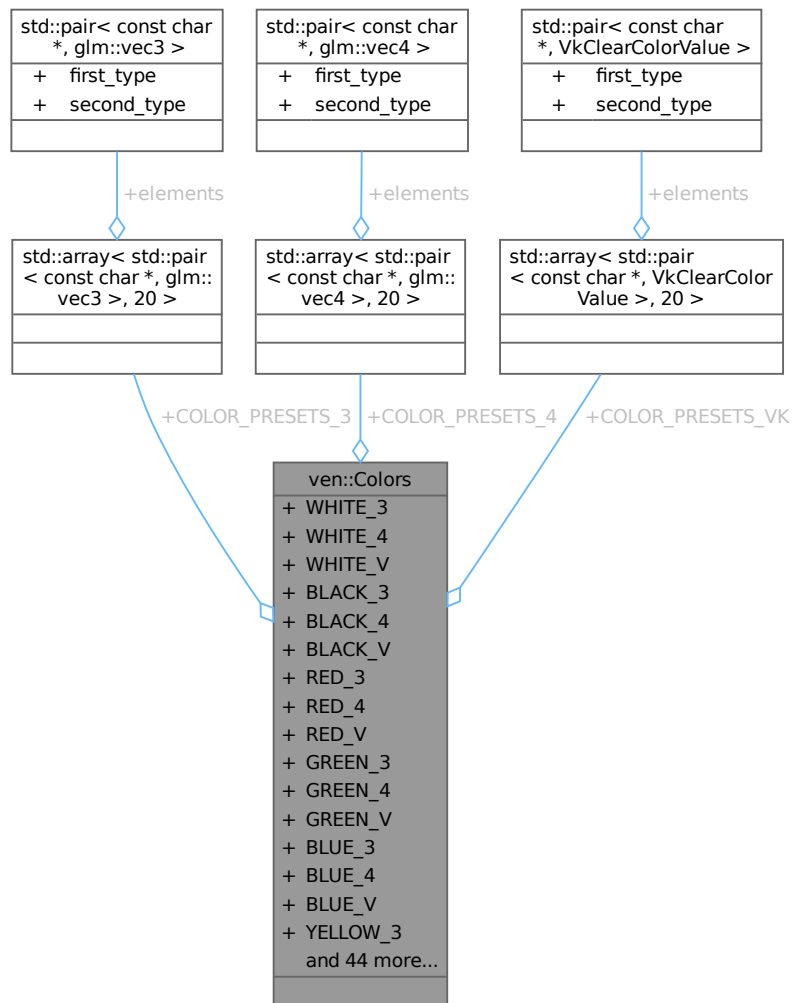
- [/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/camera.cpp](#)

7.7 ven::Colors Class Reference

Class for colors.

```
#include <Colors.hpp>
```

Collaboration diagram for ven::Colors:



Static Public Attributes

- static constexpr glm::vec3 [WHITE_3](#) = glm::vec3([COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [WHITE_4](#) = { 1.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [WHITE_V](#) = { { 1.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [BLACK_3](#) = glm::vec3(0.0F)
- static constexpr glm::vec4 [BLACK_4](#) = { 0.0F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [BLACK_V](#) = { { 0.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [RED_3](#) = glm::vec3([COLOR_MAX](#), 0.0F, 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [RED_4](#) = { 1.0F, 0.0F, 0.0F, 1.0F }

- static constexpr VkClearColorValue [RED_V](#) = { { 1.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [GREEN_3](#) = glm::vec3(0.0F, [COLOR_MAX](#), 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [GREEN_4](#) = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [GREEN_V](#) = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [BLUE_3](#) = glm::vec3(0.0F, 0.0F, [COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [BLUE_4](#) = { 0.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [BLUE_V](#) = { { 0.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [YELLOW_3](#) = glm::vec3([COLOR_MAX](#), [COLOR_MAX](#), 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [YELLOW_4](#) = { 1.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [YELLOW_V](#) = { { 1.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [CYAN_3](#) = glm::vec3(0.0F, [COLOR_MAX](#), [COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [CYAN_4](#) = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [CYAN_V](#) = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [MAGENTA_3](#) = glm::vec3([COLOR_MAX](#), 0.0F, [COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [MAGENTA_4](#) = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [MAGENTA_V](#) = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [SILVER_3](#) = glm::vec3(192.0F, 192.0F, 192.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [SILVER_4](#) = { 0.75F, 0.75F, 0.75F, 1.0F }
- static constexpr VkClearColorValue [SILVER_V](#) = { { 0.75F, 0.75F, 0.75F, 1.0F } }
- static constexpr glm::vec3 [GRAY_3](#) = glm::vec3(128.0F, 128.0F, 128.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [GRAY_4](#) = { 0.5F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue [GRAY_V](#) = { { 0.5F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 [MAROON_3](#) = glm::vec3(128.0F, 0.0F, 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [MAROON_4](#) = { 0.5F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [MAROON_V](#) = { { 0.5F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [OLIVE_3](#) = glm::vec3(128.0F, 128.0F, 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [OLIVE_4](#) = { 0.5F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [OLIVE_V](#) = { { 0.5F, 0.5F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [LIME_3](#) = glm::vec3(0.0F, [COLOR_MAX](#), 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [LIME_4](#) = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [LIME_V](#) = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [AQUA_3](#) = glm::vec3(0.0F, [COLOR_MAX](#), [COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [AQUA_4](#) = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [AQUA_V](#) = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [TEAL_3](#) = glm::vec3(0.0F, 128.0F, 128.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [TEAL_4](#) = { 0.0F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue [TEAL_V](#) = { { 0.0F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 [NAVY_3](#) = glm::vec3(0.0F, 0.0F, 128.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [NAVY_4](#) = { 0.0F, 0.0F, 0.5F, 1.0F }
- static constexpr VkClearColorValue [NAVY_V](#) = { { 0.0F, 0.0F, 0.5F, 1.0F } }
- static constexpr glm::vec3 [FUCHSIA_3](#) = glm::vec3([COLOR_MAX](#), 0.0F, [COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [FUCHSIA_4](#) = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [FUCHSIA_V](#) = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [NIGHT_BLUE_3](#) = glm::vec3(25.0F, 25.0F, 112.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [NIGHT_BLUE_4](#) = { 0.098F, 0.098F, 0.439F, 1.0F }
- static constexpr VkClearColorValue [NIGHT_BLUE_V](#) = { { 0.098F, 0.098F, 0.439F, 1.0F } }
- static constexpr glm::vec3 [SKY_BLUE_3](#) = glm::vec3(102.0F, 178.0F, 255.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [SKY_BLUE_4](#) = { 0.4F, 0.698F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [SKY_BLUE_V](#) = { { 0.4F, 0.698F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [SUNSET_3](#) = glm::vec3(255.0F, 128.0F, 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [SUNSET_4](#) = { 1.0F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [SUNSET_V](#) = { { 1.0F, 0.5F, 0.0F, 1.0F } }
- static constexpr std::array< std::pair< const char *, glm::vec3 >, 20 > [COLOR_PRESETS_3](#)
- static constexpr std::array< std::pair< const char *, glm::vec4 >, 20 > [COLOR_PRESETS_4](#)
- static constexpr std::array< std::pair< const char *, VkClearColorValue >, 20 > [COLOR_PRESETS_VK](#)

7.7.1 Detailed Description

Class for colors.

Definition at line 22 of file [Colors.hpp](#).

7.7.2 Member Data Documentation

7.7.2.1 AQUA_3

```
glm::vec3 ven::Colors::AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 78 of file [Colors.hpp](#).

7.7.2.2 AQUA_4

```
glm::vec4 ven::Colors::AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 79 of file [Colors.hpp](#).

7.7.2.3 AQUA_V

```
VkClearColorValue ven::Colors::AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 80 of file [Colors.hpp](#).

7.7.2.4 BLACK_3

```
glm::vec3 ven::Colors::BLACK_3 = glm::vec3(0.0F) [static], [constexpr]
```

Definition at line 30 of file [Colors.hpp](#).

Referenced by [ven::Model::Builder::processMesh\(\)](#).

7.7.2.5 BLACK_4

```
glm::vec4 ven::Colors::BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 31 of file [Colors.hpp](#).

7.7.2.6 BLACK_V

```
VkClearColorValue ven::Colors::BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 32 of file [Colors.hpp](#).

7.7.2.7 BLUE_3

```
glm::vec3 ven::Colors::BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 42 of file [Colors.hpp](#).

7.7.2.8 BLUE_4

```
glm::vec4 ven::Colors::BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 43 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.9 BLUE_V

```
VkClearColorValue ven::Colors::BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 44 of file [Colors.hpp](#).

7.7.2.10 COLOR_PRESETS_3

```
std::array<std::pair<const char *, glm::vec3>, 20> ven::Colors::COLOR_PRESETS_3 [static], [constexpr]
```

Initial value:

```
= {{
    {"White", WHITE_3},
    {"Black", BLACK_3},
    {"Red", RED_3},
    {"Green", GREEN_3},
    {"Blue", BLUE_3},
    {"Yellow", YELLOW_3},
    {"Cyan", CYAN_3},
    {"Magenta", MAGENTA_3},
    {"Silver", SILVER_3},
    {"Gray", GRAY_3},
    {"Maroon", MAROON_3},
    {"Olive", OLIVE_3},
    {"Lime", LIME_3},
    {"Aqua", AQUA_3},
    {"Teal", TEAL_3},
    {"Navy", NAVY_3},
    {"Fuchsia", FUCHSIA_3},
    {"Night Blue", NIGHT_BLUE_3},
    {"Sky Blue", SKY_BLUE_3},
    {"Sunset", SUNSET_3}
}}
```

Definition at line 107 of file [Colors.hpp](#).

Referenced by [ven::Gui::lightsSection\(\)](#).

7.7.2.11 COLOR_PRESETS_4

```
std::array<std::pair<const char *, glm::vec4>, 20> ven::Colors::COLOR_PRESETS_4 [static],
[constexpr]
```

Initial value:

```
= {{
    {"White", WHITE_4},
    {"Black", BLACK_4},
    {"Red", RED_4},
    {"Green", GREEN_4},
    {"Blue", BLUE_4},
    {"Yellow", YELLOW_4},
    {"Cyan", CYAN_4},
    {"Magenta", MAGENTA_4},
    {"Silver", SILVER_4},
    {"Gray", GRAY_4},
    {"Maroon", MAROON_4},
    {"Olive", OLIVE_4},
    {"Lime", LIME_4},
    {"Aqua", AQUA_4},
    {"Teal", TEAL_4},
    {"Navy", NAVY_4},
    {"Fuchsia", FUCHSIA_4},
    {"Night Blue", NIGHT_BLUE_4},
    {"Sky Blue", SKY_BLUE_4},
    {"Sunset", SUNSET_4}
}}
```

Definition at line 130 of file [Colors.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

7.7.2.12 COLOR_PRESETS_VK

```
std::array<std::pair<const char *, VkClearColorValue>, 20> ven::Colors::COLOR_PRESETS_VK
[static], [constexpr]
```

Initial value:

```
= {{
    {"White", WHITE_V},
    {"Black", BLACK_V},
    {"Red", RED_V},
    {"Green", GREEN_V},
    {"Blue", BLUE_V},
    {"Yellow", YELLOW_V},
    {"Cyan", CYAN_V},
    {"Magenta", MAGENTA_V},
    {"Silver", SILVER_V},
    {"Gray", GRAY_V},
    {"Maroon", MAROON_V},
    {"Olive", OLIVE_V},
    {"Lime", LIME_V},
    {"Aqua", AQUA_V},
    {"Teal", TEAL_V},
    {"Navy", NAVY_V},
    {"Fuchsia", FUCHSIA_V},
    {"Night Blue", NIGHT_BLUE_V},
    {"Sky Blue", SKY_BLUE_V},
    {"Sunset", SUNSET_V}
}}
```

Definition at line 153 of file [Colors.hpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

7.7.2.13 CYAN_3

```
glm::vec3 ven::Colors::CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 50 of file [Colors.hpp](#).

7.7.2.14 CYAN_4

```
glm::vec4 ven::Colors::CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 51 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.15 CYAN_V

```
VkClearColorValue ven::Colors::CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 52 of file [Colors.hpp](#).

7.7.2.16 FUCHSIA_3

```
glm::vec3 ven::Colors::FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 90 of file [Colors.hpp](#).

7.7.2.17 FUCHSIA_4

```
glm::vec4 ven::Colors::FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 91 of file [Colors.hpp](#).

7.7.2.18 FUCHSIA_V

```
VkClearColorValue ven::Colors::FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 92 of file [Colors.hpp](#).

7.7.2.19 GRAY_3

```
glm::vec3 ven::Colors::GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 62 of file [Colors.hpp](#).

7.7.2.20 GRAY_4

```
glm::vec4 ven::Colors::GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 63 of file [Colors.hpp](#).

Referenced by [ven::Gui::objectsSection\(\)](#).

7.7.2.21 GRAY_V

```
VkClearColorValue ven::Colors::GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 64 of file [Colors.hpp](#).

7.7.2.22 GREEN_3

```
glm::vec3 ven::Colors::GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 38 of file [Colors.hpp](#).

7.7.2.23 GREEN_4

```
glm::vec4 ven::Colors::GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 39 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.24 GREEN_V

```
VkClearColorValue ven::Colors::GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 40 of file [Colors.hpp](#).

7.7.2.25 LIME_3

```
glm::vec3 ven::Colors::LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 74 of file [Colors.hpp](#).

7.7.2.26 LIME_4

```
glm::vec4 ven::Colors::LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 75 of file [Colors.hpp](#).

7.7.2.27 LIME_V

```
VkClearColorValue ven::Colors::LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 76 of file [Colors.hpp](#).

7.7.2.28 MAGENTA_3

```
glm::vec3 ven::Colors::MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 54 of file [Colors.hpp](#).

7.7.2.29 MAGENTA_4

```
glm::vec4 ven::Colors::MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 55 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.30 MAGENTA_V

```
VkClearColorValue ven::Colors::MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 56 of file [Colors.hpp](#).

7.7.2.31 MAROON_3

```
glm::vec3 ven::Colors::MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 66 of file [Colors.hpp](#).

7.7.2.32 MAROON_4

```
glm::vec4 ven::Colors::MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 67 of file [Colors.hpp](#).

7.7.2.33 MAROON_V

```
VkClearColorValue ven::Colors::MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 68 of file [Colors.hpp](#).

7.7.2.34 NAVY_3

```
glm::vec3 ven::Colors::NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 86 of file [Colors.hpp](#).

7.7.2.35 NAVY_4

```
glm::vec4 ven::Colors::NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 87 of file [Colors.hpp](#).

7.7.2.36 NAVY_V

```
VkClearColorValue ven::Colors::NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 88 of file [Colors.hpp](#).

7.7.2.37 NIGHT_BLUE_3

```
glm::vec3 ven::Colors::NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 94 of file [Colors.hpp](#).

7.7.2.38 NIGHT_BLUE_4

```
glm::vec4 ven::Colors::NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F } [static], [constexpr]
```

Definition at line 95 of file [Colors.hpp](#).

7.7.2.39 NIGHT_BLUE_V

```
VkClearColorValue ven::Colors::NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } } [static],  
[constexpr]
```

Definition at line 96 of file [Colors.hpp](#).

7.7.2.40 OLIVE_3

```
glm::vec3 ven::Colors::OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 70 of file [Colors.hpp](#).

7.7.2.41 OLIVE_4

```
glm::vec4 ven::Colors::OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 71 of file [Colors.hpp](#).

7.7.2.42 OLIVE_V

```
VkClearColorValue ven::Colors::OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 72 of file [Colors.hpp](#).

7.7.2.43 RED_3

```
glm::vec3 ven::Colors::RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 34 of file [Colors.hpp](#).

7.7.2.44 RED_4

```
glm::vec4 ven::Colors::RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 35 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.45 RED_V

```
VkClearColorValue ven::Colors::RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 36 of file [Colors.hpp](#).

7.7.2.46 SILVER_3

```
glm::vec3 ven::Colors::SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 58 of file [Colors.hpp](#).

7.7.2.47 SILVER_4

```
glm::vec4 ven::Colors::SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F } [static], [constexpr]
```

Definition at line 59 of file [Colors.hpp](#).

7.7.2.48 SILVER_V

```
VkClearColorValue ven::Colors::SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } } [static], [constexpr]
```

Definition at line 60 of file [Colors.hpp](#).

7.7.2.49 SKY_BLUE_3

```
glm::vec3 ven::Colors::SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 98 of file [Colors.hpp](#).

7.7.2.50 SKY_BLUE_4

```
glm::vec4 ven::Colors::SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 99 of file [Colors.hpp](#).

7.7.2.51 SKY_BLUE_V

```
VkClearColorValue ven::Colors::SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 100 of file [Colors.hpp](#).

7.7.2.52 SUNSET_3

```
glm::vec3 ven::Colors::SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 102 of file [Colors.hpp](#).

7.7.2.53 SUNSET_4

```
glm::vec4 ven::Colors::SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 103 of file [Colors.hpp](#).

7.7.2.54 SUNSET_V

```
VkClearColorValue ven::Colors::SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 104 of file [Colors.hpp](#).

7.7.2.55 TEAL_3

```
glm::vec3 ven::Colors::TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 82 of file [Colors.hpp](#).

7.7.2.56 TEAL_4

```
glm::vec4 ven::Colors::TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 83 of file [Colors.hpp](#).

7.7.2.57 TEAL_V

```
VkClearColorValue ven::Colors::TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } } [static], [constexpr]
```

Definition at line 84 of file [Colors.hpp](#).

7.7.2.58 WHITE_3

```
glm::vec3 ven::Colors::WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 26 of file [Colors.hpp](#).

Referenced by [ven::Model::Builder::processMesh\(\)](#).

7.7.2.59 WHITE_4

```
glm::vec4 ven::Colors::WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 27 of file [Colors.hpp](#).

7.7.2.60 WHITE_V

```
VkClearColorValue ven::Colors::WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 28 of file [Colors.hpp](#).

7.7.2.61 YELLOW_3

```
glm::vec3 ven::Colors::YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX [static],  
[constexpr]
```

Definition at line 46 of file [Colors.hpp](#).

7.7.2.62 YELLOW_4

```
glm::vec4 ven::Colors::YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 47 of file [Colors.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

7.7.2.63 YELLOW_V

```
VkClearColorValue ven::Colors::YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } } [static], [constexpr]
```

Definition at line 48 of file [Colors.hpp](#).

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

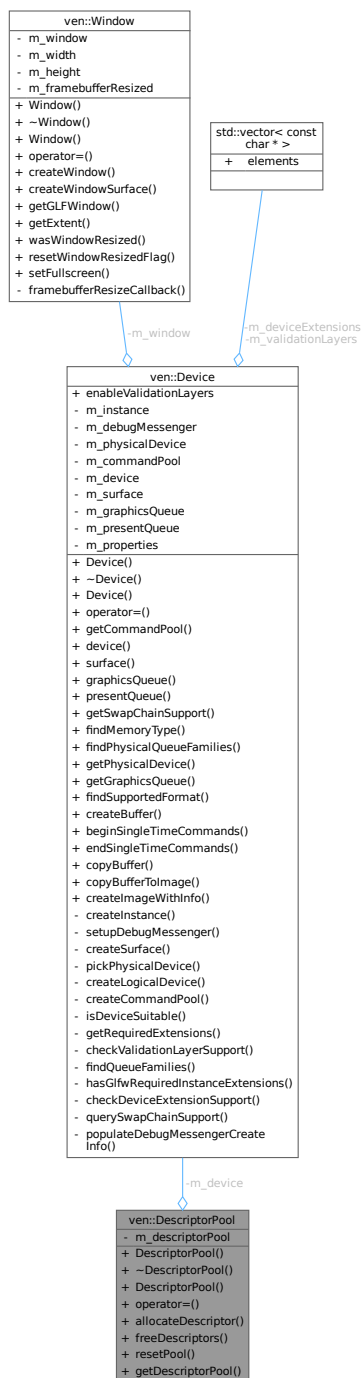
- [/home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp](#)

7.8 ven::DescriptorPool Class Reference

Class for descriptor pool.

```
#include <DescriptorPool.hpp>
```

Collaboration diagram for `ven::DescriptorPool`:



Classes

- class [Builder](#)

Public Member Functions

- [DescriptorPool](#) ([Device](#) &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags, const std::vector< VkDescriptorPoolSize > &poolSizes)

- [~DescriptorPool](#) ()
- [DescriptorPool](#) (const [DescriptorPool](#) &)=delete
- [DescriptorPool](#) & [operator=](#) (const [DescriptorPool](#) &)=delete
- bool [allocateDescriptor](#) (VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet &descriptor) const
- void [freeDescriptors](#) (const std::vector< VkDescriptorSet > &descriptors) const
- void [resetPool](#) () const
- VkDescriptorPool [getDescriptorPool](#) () const

Private Attributes

- [Device](#) & [m_device](#)
- VkDescriptorPool [m_descriptorPool](#)

Friends

- class [DescriptorWriter](#)

7.8.1 Detailed Description

Class for descriptor pool.

Definition at line 22 of file [DescriptorPool.hpp](#).

7.8.2 Constructor & Destructor Documentation

7.8.2.1 DescriptorPool() [1/2]

```
ven::DescriptorPool::DescriptorPool (
    Device & device,
    uint32_t maxSets,
    VkDescriptorPoolCreateFlags poolFlags,
    const std::vector< VkDescriptorPoolSize > & poolSizes)
```

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

References [ven::Device::device\(\)](#), [m_descriptorPool](#), and [m_device](#).

Here is the call graph for this function:



7.8.2.2 ~DescriptorPool()

```
ven::DescriptorPool::~~DescriptorPool () [inline]
```

Definition at line 48 of file [DescriptorPool.hpp](#).

References [ven::Device::device\(\)](#), [m_descriptorPool](#), and [m_device](#).

Here is the call graph for this function:



7.8.2.3 DescriptorPool() [2/2]

```
ven::DescriptorPool::DescriptorPool (
    const DescriptorPool & ) [delete]
```

7.8.3 Member Function Documentation

7.8.3.1 allocateDescriptor()

```
bool ven::DescriptorPool::allocateDescriptor (
    VkDescriptorSetLayout descriptorSetLayout,
    VkDescriptorSet & descriptor) const
```

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

7.8.3.2 freeDescriptors()

```
void ven::DescriptorPool::freeDescriptors (
    const std::vector< VkDescriptorSet > & descriptors) const [inline]
```

Definition at line 54 of file [DescriptorPool.hpp](#).

References [ven::Device::device\(\)](#), [m_descriptorPool](#), and [m_device](#).

Here is the call graph for this function:



7.8.3.3 getDescriptorPool()

```
VkDescriptorPool ven::DescriptorPool::getDescriptorPool () const [inline], [nodiscard]
```

Definition at line 57 of file [DescriptorPool.hpp](#).

References [m_descriptorPool](#).

7.8.3.4 operator=()

```
DescriptorPool & ven::DescriptorPool::operator= (
    const DescriptorPool & ) [delete]
```

7.8.3.5 resetPool()

```
void ven::DescriptorPool::resetPool () const [inline]
```

Definition at line 55 of file [DescriptorPool.hpp](#).

References [ven::Device::device\(\)](#), [m_descriptorPool](#), and [m_device](#).

Here is the call graph for this function:



7.8.4 Friends And Related Symbol Documentation

7.8.4.1 DescriptorWriter

```
friend class DescriptorWriter [friend]
```

Definition at line 63 of file [DescriptorPool.hpp](#).

7.8.5 Member Data Documentation

7.8.5.1 m_descriptorPool

```
VkDescriptorPool ven::DescriptorPool::m_descriptorPool [private]
```

Definition at line 62 of file [DescriptorPool.hpp](#).

Referenced by [DescriptorPool\(\)](#), [freeDescriptors\(\)](#), [getDescriptorPool\(\)](#), [resetPool\(\)](#), and [~DescriptorPool\(\)](#).

7.8.5.2 m_device

```
Device& ven::DescriptorPool::m_device [private]
```

Definition at line 61 of file [DescriptorPool.hpp](#).

Referenced by [DescriptorPool\(\)](#), [freeDescriptors\(\)](#), [resetPool\(\)](#), and [~DescriptorPool\(\)](#).

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

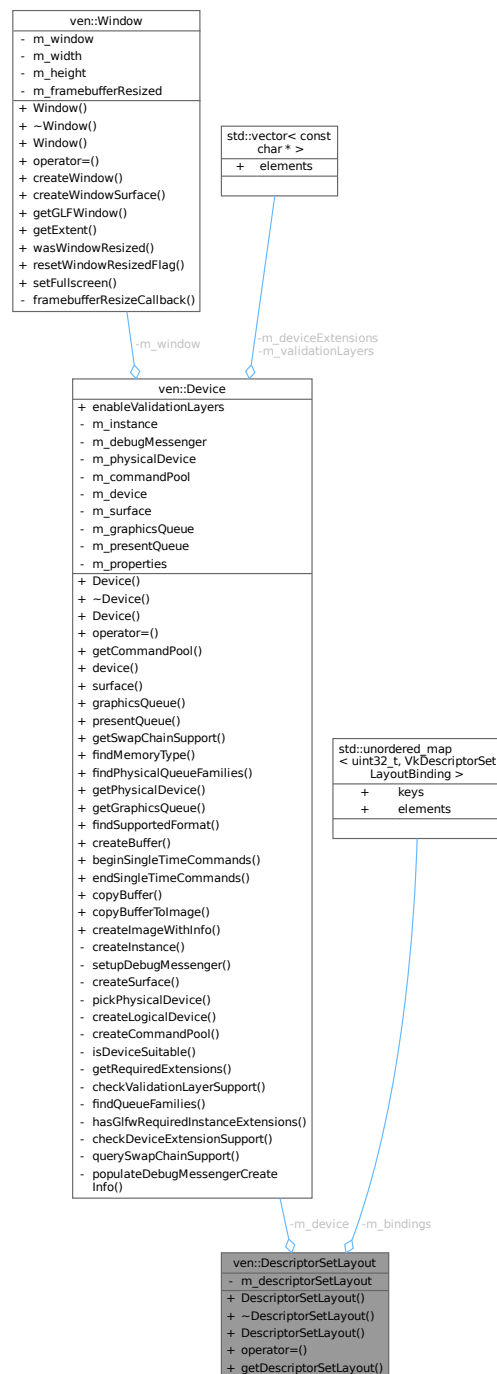
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp](#)

7.9 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

```
#include <DescriptorSetLayout.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout:



Classes

- class [Builder](#)

Public Member Functions

- [DescriptorSetLayout](#) ([Device](#) &device, const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > &bindings)

- [~DescriptorSetLayout\(\)](#)
- [DescriptorSetLayout\(const DescriptorSetLayout &\)=delete](#)
- [DescriptorSetLayout & operator=\(const DescriptorSetLayout &\)=delete](#)
- [VkDescriptorSetLayout getDescriptorSetLayout\(\)](#) const

Private Attributes

- [Device](#) & [m_device](#)
- [VkDescriptorSetLayout m_descriptorSetLayout](#)
- [std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > m_bindings](#)

Friends

- class [DescriptorWriter](#)

7.9.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file [DescriptorSetLayout.hpp](#).

7.9.2 Constructor & Destructor Documentation

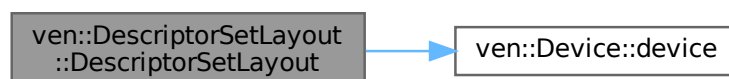
7.9.2.1 DescriptorSetLayout() [1/2]

```
ven::DescriptorSetLayout::DescriptorSetLayout (
    Device & device,
    const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > & bindings)
```

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

References [ven::Device::device\(\)](#), [m_descriptorSetLayout](#), and [m_device](#).

Here is the call graph for this function:



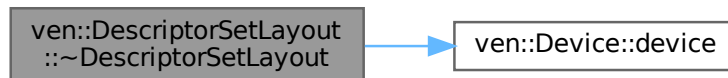
7.9.2.2 ~DescriptorSetLayout()

```
ven::DescriptorSetLayout::~~DescriptorSetLayout () [inline]
```

Definition at line 42 of file [DescriptorSetLayout.hpp](#).

References [ven::Device::device\(\)](#), [m_descriptorSetLayout](#), and [m_device](#).

Here is the call graph for this function:



7.9.2.3 DescriptorSetLayout() [2/2]

```
ven::DescriptorSetLayout::DescriptorSetLayout (
    const DescriptorSetLayout & ) [delete]
```

7.9.3 Member Function Documentation

7.9.3.1 getDescriptorSetLayout()

```
VkDescriptorSetLayout ven::DescriptorSetLayout::getDescriptorSetLayout () const [inline]
```

Definition at line 47 of file [DescriptorSetLayout.hpp](#).

References [m_descriptorSetLayout](#).

7.9.3.2 operator=()

```
DescriptorSetLayout & ven::DescriptorSetLayout::operator= (
    const DescriptorSetLayout & ) [delete]
```

7.9.4 Friends And Related Symbol Documentation

7.9.4.1 DescriptorWriter

```
friend class DescriptorWriter [friend]
```

Definition at line 55 of file [DescriptorSetLayout.hpp](#).

7.9.5 Member Data Documentation

7.9.5.1 m_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorSetLayout::m_bindings [private]
```

Definition at line 53 of file [DescriptorSetLayout.hpp](#).

Referenced by [ven::DescriptorWriter::writeBuffer\(\)](#).

7.9.5.2 m_descriptorSetLayout

```
VkDescriptorSetLayout ven::DescriptorSetLayout::m_descriptorSetLayout [private]
```

Definition at line 52 of file [DescriptorSetLayout.hpp](#).

Referenced by [DescriptorSetLayout\(\)](#), [getDescriptorSetLayout\(\)](#), and [~DescriptorSetLayout\(\)](#).

7.9.5.3 m_device

```
Device& ven::DescriptorSetLayout::m_device [private]
```

Definition at line 51 of file [DescriptorSetLayout.hpp](#).

Referenced by [DescriptorSetLayout\(\)](#), and [~DescriptorSetLayout\(\)](#).

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

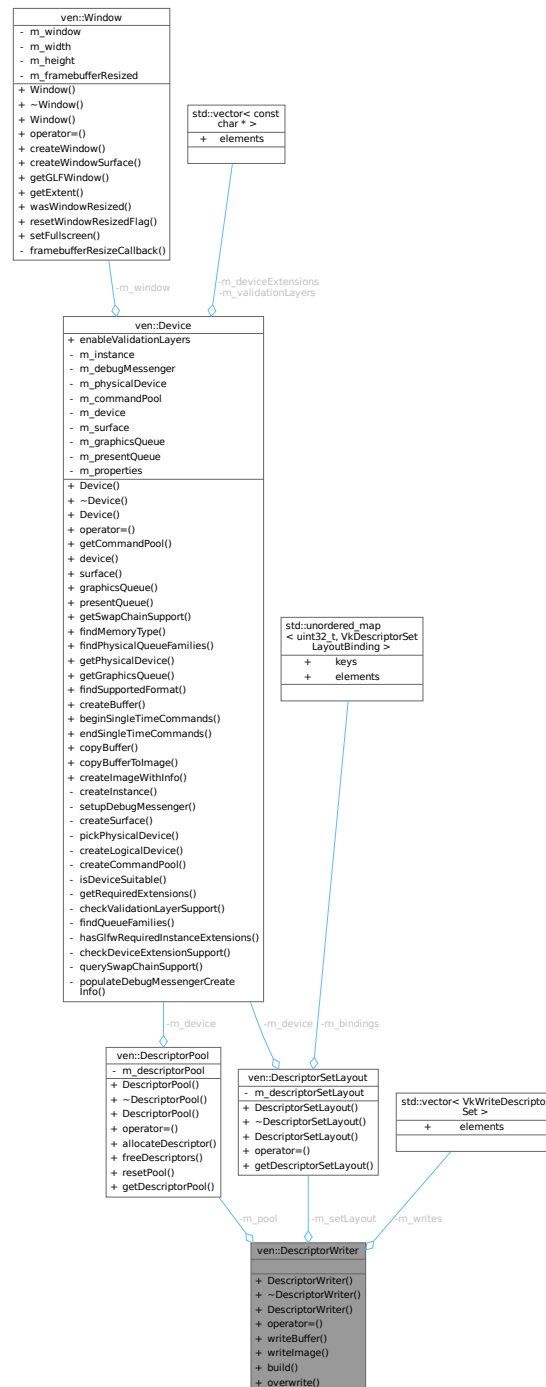
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#)

7.10 ven::DescriptorWriter Class Reference

Class for descriptor writer.

```
#include <DescriptorWriter.hpp>
```

Collaboration diagram for ven::DescriptorWriter:



Public Member Functions

- `DescriptorWriter` (`DescriptorSetLayout` &setLayout, `DescriptorPool` &pool)
- `~DescriptorWriter` ()=default
- `DescriptorWriter` (const `DescriptorWriter` &)=delete
- `DescriptorWriter` & operator= (const `DescriptorWriter` &)=delete
- `DescriptorWriter` & writeBuffer (uint32_t binding, const `VkDescriptorBufferInfo` *bufferInfo)

- [DescriptorWriter](#) & [writeImage](#) (uint32_t binding, const VkDescriptorImageInfo *imageInfo)
- bool [build](#) (VkDescriptorSet &set)
- void [overwrite](#) (const VkDescriptorSet &set)

Private Attributes

- [DescriptorSetLayout](#) & [m_setLayout](#)
- [DescriptorPool](#) & [m_pool](#)
- std::vector< [VkWriteDescriptorSet](#) > [m_writes](#)

7.10.1 Detailed Description

Class for descriptor writer.

Definition at line 21 of file [DescriptorWriter.hpp](#).

7.10.2 Constructor & Destructor Documentation

7.10.2.1 DescriptorWriter() [1/2]

```
ven::DescriptorWriter::DescriptorWriter (
    DescriptorSetLayout & setLayout,
    DescriptorPool & pool) [inline]
```

Definition at line 25 of file [DescriptorWriter.hpp](#).

7.10.2.2 ~DescriptorWriter()

```
ven::DescriptorWriter::~DescriptorWriter () [default]
```

7.10.2.3 DescriptorWriter() [2/2]

```
ven::DescriptorWriter::DescriptorWriter (
    const DescriptorWriter & ) [delete]
```

7.10.3 Member Function Documentation

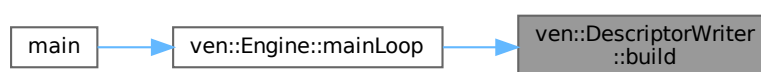
7.10.3.1 build()

```
bool ven::DescriptorWriter::build (
    VkDescriptorSet & set)
```

Definition at line 43 of file [descriptorWriter.cpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.10.3.2 operator=()

```
DescriptorWriter & ven::DescriptorWriter::operator= (
    const DescriptorWriter & ) [delete]
```

7.10.3.3 overwrite()

```
void ven::DescriptorWriter::overwrite (
    const VkDescriptorSet & set)
```

Definition at line 52 of file [descriptorWriter.cpp](#).

7.10.3.4 writeBuffer()

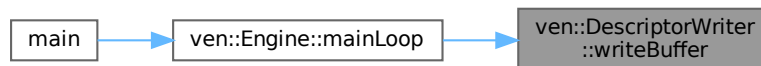
```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
    uint32_t binding,
    const VkDescriptorBufferInfo * bufferInfo)
```

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

References [ven::DescriptorSetLayout::m_bindings](#), [m_setLayout](#), and [m_writes](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.10.3.5 writeImage()

```
ven::DescriptorWriter & ven::DescriptorWriter::writeImage (
    uint32_t binding,
    const VkDescriptorImageInfo * imageInfo)
```

Definition at line 24 of file [descriptorWriter.cpp](#).

7.10.4 Member Data Documentation

7.10.4.1 m_pool

```
DescriptorPool& ven::DescriptorWriter::m_pool [private]
```

Definition at line 40 of file [DescriptorWriter.hpp](#).

7.10.4.2 m_setLayout

`DescriptorSetLayout& ven::DescriptorWriter::m_setLayout [private]`

Definition at line 39 of file [DescriptorWriter.hpp](#).

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

7.10.4.3 m_writes

`std::vector<VkWriteDescriptorSet> ven::DescriptorWriter::m_writes [private]`

Definition at line 41 of file [DescriptorWriter.hpp](#).

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

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

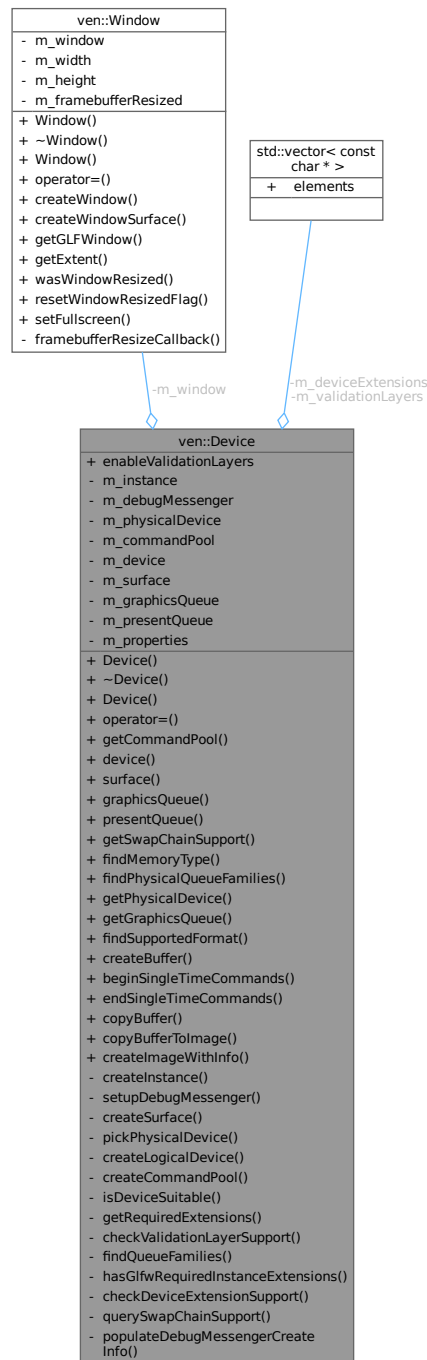
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp](#)

7.11 ven::Device Class Reference

Class for device.

```
#include <Device.hpp>
```

Collaboration diagram for ven::Device:



Public Member Functions

- [Device](#) ([Window](#) &window)
- [~Device](#) ()
- [Device](#) (const [Device](#) &)=delete
- [Device](#) & [operator=](#) (const [Device](#) &)=delete
- [VkCommandPool](#) [getCommandPool](#) () const

- VkDevice [device](#) () const
- VkSurfaceKHR [surface](#) () const
- VkQueue [graphicsQueue](#) () const
- VkQueue [presentQueue](#) () const
- [SwapChainSupportDetails](#) [getSwapChainSupport](#) () const
- uint32_t [findMemoryType](#) (uint32_t typeFilter, VkMemoryPropertyFlags properties) const
- [QueueFamilyIndices](#) [findPhysicalQueueFamilies](#) () const
- VkPhysicalDevice [getPhysicalDevice](#) () const
- VkQueue [getGraphicsQueue](#) () const
- VkFormat [findSupportedFormat](#) (const std::vector< VkFormat > &candidates, VkImageTiling tiling, VkFormatFeatureFlags features) const
- void [createBuffer](#) (VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
- VkCommandBuffer [beginSingleTimeCommands](#) () const
- void [endSingleTimeCommands](#) (VkCommandBuffer commandBuffer) const
- void [copyBuffer](#) (VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const
- void [copyBufferToImage](#) (VkBuffer buffer, VkImage image, uint32_t width, uint32_t height, uint32_t layerCount) const
- void [createImageWithInfo](#) (const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags properties, VkImage &image, VkDeviceMemory &imageMemory) const

Public Attributes

- const bool [enableValidationLayers](#) = true

Private Member Functions

- void [createInstance](#) ()
- void [setupDebugMessenger](#) ()
- void [createSurface](#) ()
- void [pickPhysicalDevice](#) ()
- void [createLogicalDevice](#) ()
- void [createCommandPool](#) ()
- bool [isDeviceSuitable](#) (VkPhysicalDevice [device](#)) const
- std::vector< const char * > [getRequiredExtensions](#) () const
- bool [checkValidationLayerSupport](#) () const
- [QueueFamilyIndices](#) [findQueueFamilies](#) (VkPhysicalDevice [device](#)) const
- void [hasGlfwRequiredInstanceExtensions](#) () const
- bool [checkDeviceExtensionSupport](#) (VkPhysicalDevice [device](#)) const
- [SwapChainSupportDetails](#) [querySwapChainSupport](#) (VkPhysicalDevice [device](#)) const

Static Private Member Functions

- static void [populateDebugMessengerCreateInfo](#) (VkDebugUtilsMessengerCreateInfoEXT &createInfo)

Private Attributes

- VkInstance [m_instance](#)
- VkDebugUtilsMessengerEXT [m_debugMessenger](#)
- VkPhysicalDevice [m_physicalDevice](#) = VK_NULL_HANDLE
- [Window](#) & [m_window](#)
- VkCommandPool [m_commandPool](#)
- VkDevice [m_device](#)
- VkSurfaceKHR [m_surface](#)
- VkQueue [m_graphicsQueue](#)
- VkQueue [m_presentQueue](#)
- VkPhysicalDeviceProperties [m_properties](#)
- const std::vector< const char * > [m_validationLayers](#) = {"VK_LAYER_KHRONOS_validation"}
- const std::vector< const char * > [m_deviceExtensions](#) = {VK_KHR_SWAPCHAIN_EXTENSION_NAME}

7.11.1 Detailed Description

Class for device.

Definition at line 34 of file [Device.hpp](#).

7.11.2 Constructor & Destructor Documentation

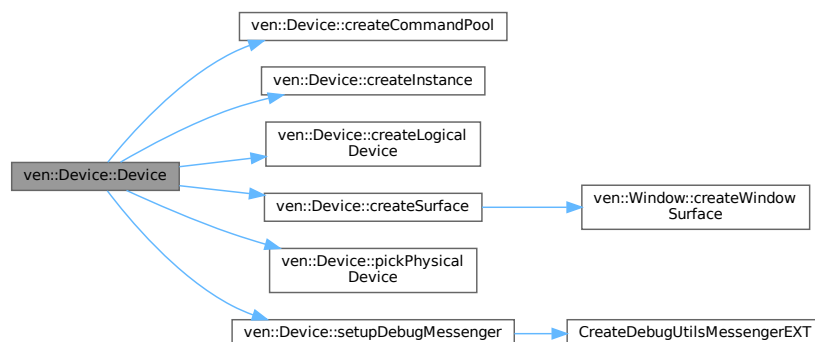
7.11.2.1 Device() [1/2]

```
ven::Device::Device (
    Window & window) [explicit]
```

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

References [createCommandPool\(\)](#), [createInstance\(\)](#), [createLogicalDevice\(\)](#), [createSurface\(\)](#), [pickPhysicalDevice\(\)](#), and [setupDebugMessenger\(\)](#).

Here is the call graph for this function:



7.11.2.2 ~Device()

```
ven::Device::~~Device ()
```

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

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

Here is the call graph for this function:



7.11.2.3 Device() [2/2]

```
ven::Device::Device (
    const Device & ) [delete]
```

7.11.3 Member Function Documentation

7.11.3.1 beginSingleTimeCommands()

```
VkCommandBuffer ven::Device::beginSingleTimeCommands () const [nodiscard]
```

Definition at line 408 of file [device.cpp](#).

7.11.3.2 checkDeviceExtensionSupport()

```
bool ven::Device::checkDeviceExtensionSupport (
    VkPhysicalDevice device) const [private]
```

Definition at line 286 of file [device.cpp](#).

7.11.3.3 checkValidationLayerSupport()

```
bool ven::Device::checkValidationLayerSupport () const [nodiscard], [private]
```

Definition at line 223 of file [device.cpp](#).

7.11.3.4 copyBuffer()

```
void ven::Device::copyBuffer (
    VkBuffer srcBuffer,
    VkBuffer dstBuffer,
    VkDeviceSize size) const
```

Definition at line 442 of file [device.cpp](#).

7.11.3.5 copyBufferToImage()

```
void ven::Device::copyBufferToImage (
    VkBuffer buffer,
    VkImage image,
    uint32_t width,
    uint32_t height,
    uint32_t layerCount) const
```

Definition at line 455 of file [device.cpp](#).

7.11.3.6 createBuffer()

```
void ven::Device::createBuffer (
    VkDeviceSize size,
    VkBufferUsageFlags usage,
    VkMemoryPropertyFlags properties,
    VkBuffer & buffer,
    VkDeviceMemory & bufferMemory) const
```

Definition at line 381 of file [device.cpp](#).

Referenced by [ven::Buffer::Buffer\(\)](#).

Here is the caller graph for this function:



7.11.3.7 createCommandPool()

```
void ven::Device::createCommandPool () [private]
```

Definition at line 169 of file [device.cpp](#).

References [ven::QueueFamilyIndices::graphicsFamily](#).

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

Here is the caller graph for this function:



7.11.3.8 createImageWithInfo()

```
void ven::Device::createImageWithInfo (
    const VkImageCreateInfo & imageInfo,
    VkMemoryPropertyFlags properties,
    VkImage & image,
    VkDeviceMemory & imageMemory) const
```

Definition at line 476 of file [device.cpp](#).

7.11.3.9 createInstance()

```
void ven::Device::createInstance () [private]
```

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

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

Here is the caller graph for this function:



7.11.3.10 createLogicalDevice()

```
void ven::Device::createLogicalDevice () [private]
```

Definition at line 122 of file [device.cpp](#).

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

Here is the caller graph for this function:



7.11.3.11 createSurface()

```
void ven::Device::createSurface () [inline], [private]
```

Definition at line 77 of file [Device.hpp](#).

References [ven::Window::createWindowSurface\(\)](#), [m_instance](#), [m_surface](#), and [m_window](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.12 device()

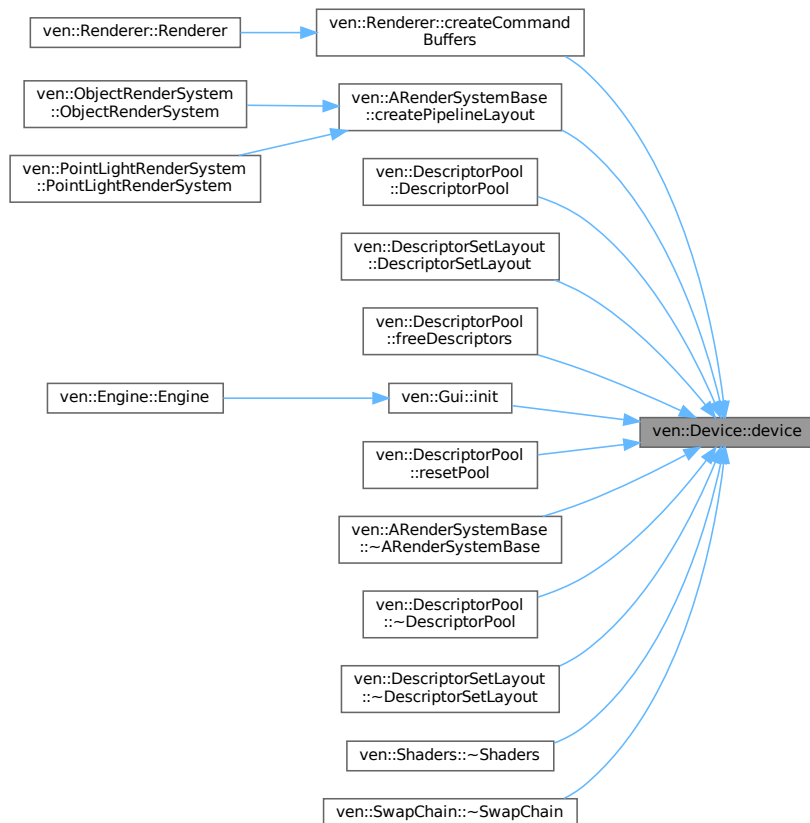
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

Definition at line 51 of file [Device.hpp](#).

References [m_device](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), [ven::DescriptorPool::DescriptorPool\(\)](#), [ven::DescriptorSetLayout::DescriptorSetLayout\(\)](#), [ven::DescriptorPool::freeDescriptors\(\)](#), [ven::Gui::init\(\)](#), [ven::DescriptorPool::resetPool\(\)](#), [ven::ARenderSystemBase::~ARenderSystemBase\(\)](#), [ven::DescriptorPool::~DescriptorPool\(\)](#), [ven::DescriptorSetLayout::~DescriptorSetLayout\(\)](#), [ven::Shaders::~Shaders\(\)](#), and [ven::SwapChain::~SwapChain\(\)](#).

Here is the caller graph for this function:



7.11.3.13 endSingleTimeCommands()

```
void ven::Device::endSingleTimeCommands (
    VkCommandBuffer commandBuffer) const
```

Definition at line 427 of file [device.cpp](#).

7.11.3.14 findMemoryType()

```
uint32_t ven::Device::findMemoryType (
    uint32_t typeFilter,
    VkMemoryPropertyFlags properties) const [nodiscard]
```

Definition at line 366 of file [device.cpp](#).

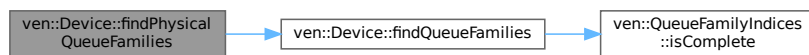
7.11.3.15 findPhysicalQueueFamilies()

```
QueueFamilyIndices ven::Device::findPhysicalQueueFamilies () const [inline], [nodiscard]
```

Definition at line 58 of file [Device.hpp](#).

References [findQueueFamilies\(\)](#), and [m_physicalDevice](#).

Here is the call graph for this function:



7.11.3.16 findQueueFamilies()

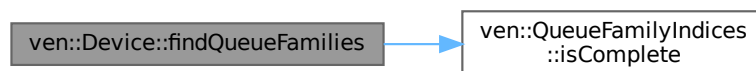
```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
    VkPhysicalDevice device) const [private]
```

Definition at line 302 of file [device.cpp](#).

References [ven::QueueFamilyIndices::graphicsFamily](#), [ven::QueueFamilyIndices::graphicsFamilyHasValue](#), [ven::QueueFamilyIndices::isComplete\(\)](#), [ven::QueueFamilyIndices::presentFamily](#), and [ven::QueueFamilyIndices::presentFamilyHasValue](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.17 findSupportedFormat()

```
VkFormat ven::Device::findSupportedFormat (
    const std::vector< VkFormat > & candidates,
    VkImageTiling tiling,
    VkFormatFeatureFlags features) const [nodiscard]
```

Definition at line 352 of file [device.cpp](#).

7.11.3.18 getCommandPool()

```
VkCommandPool ven::Device::getCommandPool () const [inline], [nodiscard]
```

Definition at line 50 of file [Device.hpp](#).

References [m_commandPool](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#).

Here is the caller graph for this function:



7.11.3.19 getGraphicsQueue()

```
VkQueue ven::Device::getGraphicsQueue () const [inline], [nodiscard]
```

Definition at line 60 of file [Device.hpp](#).

References [m_graphicsQueue](#).

7.11.3.20 getPhysicalDevice()

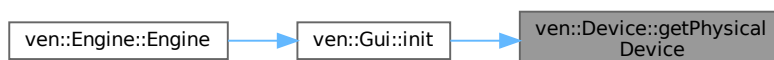
```
VkPhysicalDevice ven::Device::getPhysicalDevice () const [inline], [nodiscard]
```

Definition at line 59 of file [Device.hpp](#).

References [m_physicalDevice](#).

Referenced by [ven::Gui::init\(\)](#).

Here is the caller graph for this function:



7.11.3.21 getRequiredExtensions()

```
std::vector< const char * > ven::Device::getRequiredExtensions () const [nodiscard], [private]
```

Definition at line 248 of file [device.cpp](#).

7.11.3.22 getSwapChainSupport()

```
SwapChainSupportDetails ven::Device::getSwapChainSupport () const [inline], [nodiscard]
```

Definition at line 56 of file [Device.hpp](#).

References [m_physicalDevice](#), and [querySwapChainSupport\(\)](#).

Here is the call graph for this function:

**7.11.3.23 graphicsQueue()**

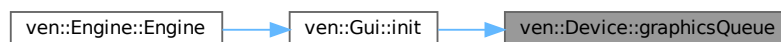
```
VkQueue ven::Device::graphicsQueue () const [inline], [nodiscard]
```

Definition at line 53 of file [Device.hpp](#).

References [m_graphicsQueue](#).

Referenced by [ven::Gui::init\(\)](#).

Here is the caller graph for this function:

**7.11.3.24 hasGlfwRequiredInstanceExtensions()**

```
void ven::Device::hasGlfwRequiredInstanceExtensions () const [private]
```

Definition at line 263 of file [device.cpp](#).

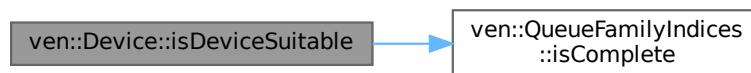
7.11.3.25 isDeviceSuitable()

```
bool ven::Device::isDeviceSuitable (
    VkPhysicalDevice device) const [private]
```

Definition at line 183 of file [device.cpp](#).

References [ven::QueueFamilyIndices::isComplete\(\)](#).

Here is the call graph for this function:



7.11.3.26 operator=()

```
Device & ven::Device::operator= (
    const Device & ) [delete]
```

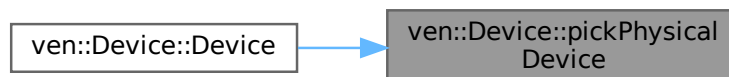
7.11.3.27 pickPhysicalDevice()

```
void ven::Device::pickPhysicalDevice () [private]
```

Definition at line 96 of file [device.cpp](#).

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

Here is the caller graph for this function:



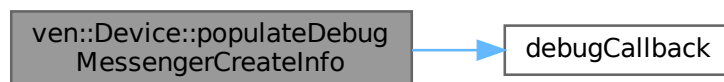
7.11.3.28 populateDebugMessengerCreateInfo()

```
void ven::Device::populateDebugMessengerCreateInfo (
    VkDebugUtilsMessengerCreateInfoEXT & createInfo) [static], [private]
```

Definition at line 200 of file [device.cpp](#).

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

Here is the call graph for this function:



7.11.3.29 presentQueue()

```
VkQueue ven::Device::presentQueue () const [inline], [nodiscard]
```

Definition at line 54 of file [Device.hpp](#).

References [m_presentQueue](#).

7.11.3.30 querySwapChainSupport()

```
ven::SwapChainSupportDetails ven::Device::querySwapChainSupport (
    VkPhysicalDevice device) const [private]
```

Definition at line 331 of file [device.cpp](#).

References [ven::SwapChainSupportDetails::capabilities](#), [ven::SwapChainSupportDetails::formats](#), and [ven::SwapChainSupportDetails::capabilities](#).

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

Here is the caller graph for this function:



7.11.3.31 `setupDebugMessenger()`

```
void ven::Device::setupDebugMessenger () [private]
```

Definition at line 213 of file [device.cpp](#).

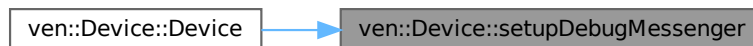
References [CreateDebugUtilsMessengerEXT\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.32 `surface()`

```
VkSurfaceKHR ven::Device::surface () const [inline], [nodiscard]
```

Definition at line 52 of file [Device.hpp](#).

References [m_surface](#).

7.11.4 Member Data Documentation

7.11.4.1 `enableValidationLayers`

```
const bool ven::Device::enableValidationLayers = true
```

Definition at line 41 of file [Device.hpp](#).

7.11.4.2 `m_commandPool`

```
VkCommandPool ven::Device::m_commandPool [private]
```

Definition at line 96 of file [Device.hpp](#).

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

7.11.4.3 m_debugMessenger

```
VkDebugUtilsMessengerEXT ven::Device::m_debugMessenger [private]
```

Definition at line 93 of file [Device.hpp](#).

7.11.4.4 m_device

```
VkDevice ven::Device::m_device [private]
```

Definition at line 98 of file [Device.hpp](#).

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

7.11.4.5 m_deviceExtensions

```
const std::vector<const char *> ven::Device::m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION↵  
_NAME} [private]
```

Definition at line 105 of file [Device.hpp](#).

7.11.4.6 m_graphicsQueue

```
VkQueue ven::Device::m_graphicsQueue [private]
```

Definition at line 100 of file [Device.hpp](#).

Referenced by [getGraphicsQueue\(\)](#), and [graphicsQueue\(\)](#).

7.11.4.7 m_instance

```
VkInstance ven::Device::m_instance [private]
```

Definition at line 92 of file [Device.hpp](#).

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

7.11.4.8 m_physicalDevice

```
VkPhysicalDevice ven::Device::m_physicalDevice = VK_NULL_HANDLE [private]
```

Definition at line 94 of file [Device.hpp](#).

Referenced by [findPhysicalQueueFamilies\(\)](#), [getPhysicalDevice\(\)](#), and [getSwapChainSupport\(\)](#).

7.11.4.9 m_presentQueue

```
VkQueue ven::Device::m_presentQueue [private]
```

Definition at line 101 of file [Device.hpp](#).

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

7.11.4.10 m_properties

```
VkPhysicalDeviceProperties ven::Device::m_properties [private]
```

Definition at line 102 of file [Device.hpp](#).

7.11.4.11 m_surface

```
VkSurfaceKHR ven::Device::m_surface [private]
```

Definition at line 99 of file [Device.hpp](#).

Referenced by [createSurface\(\)](#), and [surface\(\)](#).

7.11.4.12 m_validationLayers

```
const std::vector<const char *> ven::Device::m_validationLayers = {"VK_LAYER_KHRONOS_validation"}  
[private]
```

Definition at line 104 of file [Device.hpp](#).

7.11.4.13 m_window

```
Window& ven::Device::m_window [private]
```

Definition at line 95 of file [Device.hpp](#).

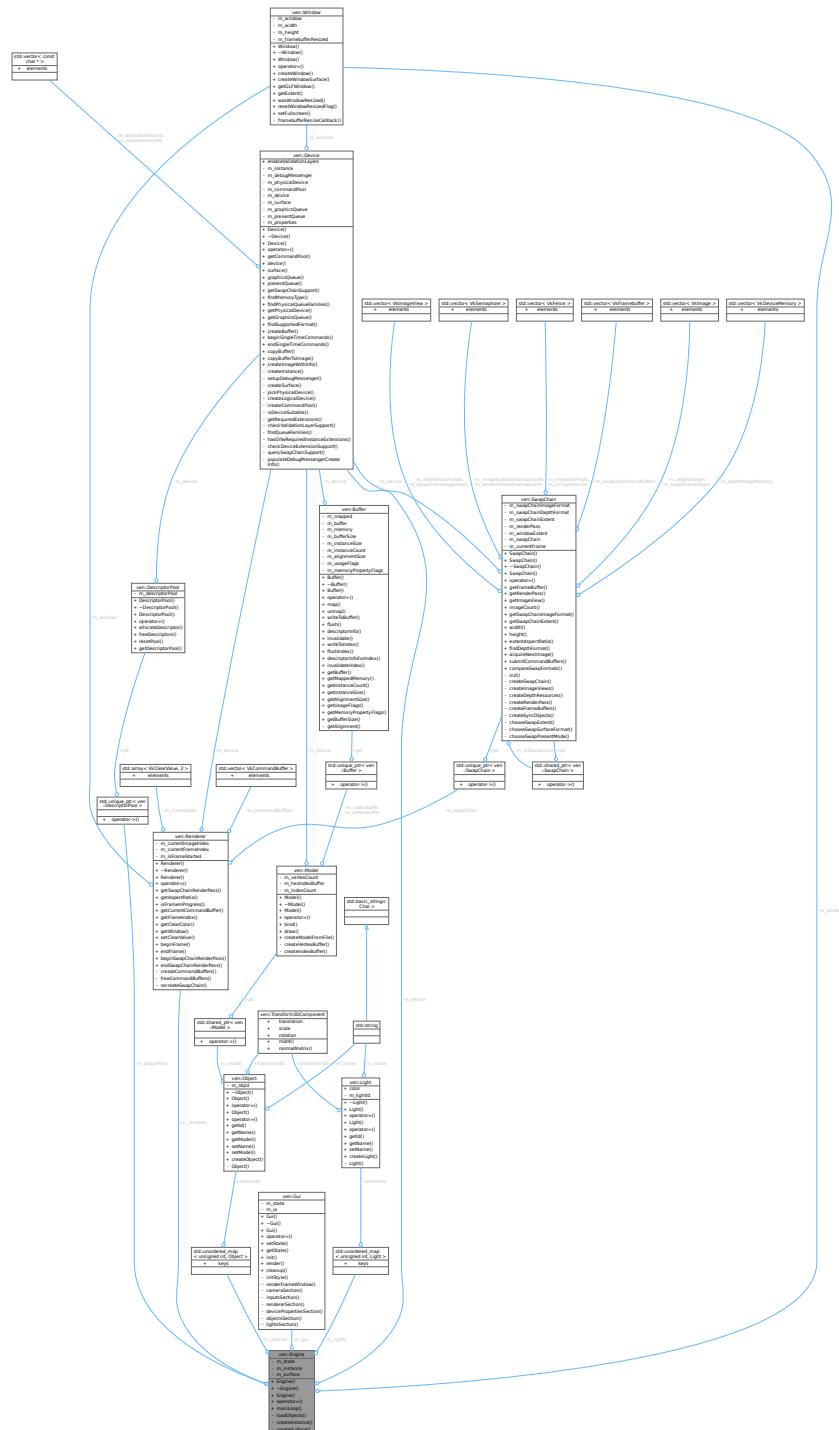
Referenced by [createSurface\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/device.cpp](#)

Class for engine.

Collaboration diagram for ven::Engine:



Public Member Functions

- [Engine](#) (uint32_t=DEFAULT_WIDTH, uint32_t=DEFAULT_HEIGHT, const std::string &title=DEFAULT_TITLE.data())
- [~Engine](#) ()=default
- [Engine](#) (const [Engine](#) &)=delete
- [Engine operator=](#) (const [Engine](#) &)=delete
- void [mainLoop](#) ()

Private Member Functions

- void [loadObjects](#) ()
- void [createInstance](#) ()
- void [createSurface](#) ()

Private Attributes

- [ENGINE_STATE](#) m_state {EXIT}
- [Window](#) m_window
- [Device](#) m_device {m_window}
- [Renderer](#) m_renderer {m_window, m_device}
- [Gui](#) m_gui
- std::unique_ptr< [DescriptorPool](#) > m_globalPool
- [Object::Map](#) m_objects
- [Light::Map](#) m_lights
- [VkInstance](#) m_instance {nullptr}
- [VkSurfaceKHR](#) m_surface {nullptr}

7.12.1 Detailed Description

Class for engine.

Definition at line 33 of file [Engine.hpp](#).

7.12.2 Constructor & Destructor Documentation

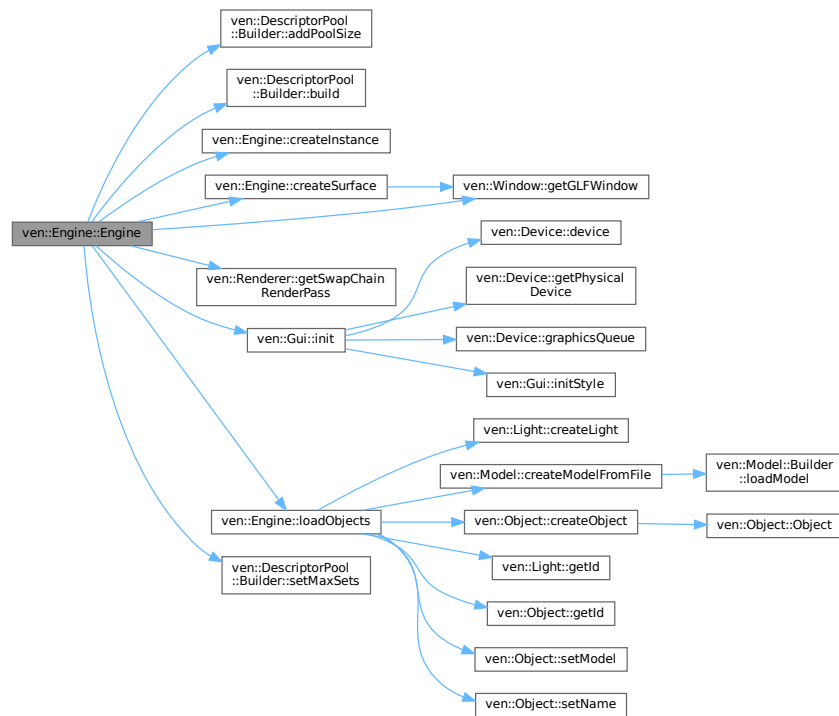
7.12.2.1 Engine() [1/2]

```
ven::Engine::Engine (
    uint32_t width = DEFAULT_WIDTH,
    uint32_t height = DEFAULT_HEIGHT,
    const std::string & title = DEFAULT_TITLE.data()) [explicit]
```

Definition at line 15 of file [engine.cpp](#).

References [ven::DescriptorPool::Builder::addPoolSize\(\)](#), [ven::DescriptorPool::Builder::build\(\)](#), [createInstance\(\)](#), [createSurface\(\)](#), [ven::EDITOR](#), [ven::Window::getGLFWWindow\(\)](#), [ven::Renderer::getSwapChainRenderPass\(\)](#), [ven::Gui::init\(\)](#), [loadObjects\(\)](#), [m_device](#), [m_globalPool](#), [m_instance](#), [m_renderer](#), [m_window](#), [ven::MAX_FRAMES_IN_FLIGHT](#), and [ven::DescriptorPool::Builder::setMaxSets\(\)](#).

Here is the call graph for this function:



7.12.2.2 ~Engine()

```
ven::Engine::~~Engine () [default]
```

7.12.2.3 Engine() [2/2]

```
ven::Engine::Engine (
    const Engine & ) [delete]
```

7.12.3 Member Function Documentation

7.12.3.1 createInstance()

```
void ven::Engine::createInstance () [private]
```

Definition at line 23 of file [engine.cpp](#).

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

Here is the caller graph for this function:



7.12.3.2 createSurface()

```
void ven::Engine::createSurface () [inline], [private]
```

Definition at line 63 of file [Engine.hpp](#).

References [ven::Window::getGLFWWindow\(\)](#), [m_instance](#), [m_surface](#), and [m_window](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.3 loadObjects()

```
void ven::Engine::loadObjects () [private]
```

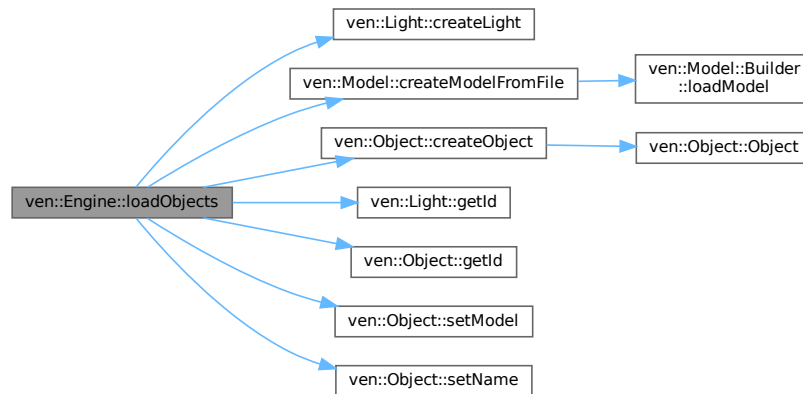
Definition at line 42 of file [engine.cpp](#).

References [ven::Colors::BLUE_4](#), [ven::Light::color](#), [ven::Light::createLight\(\)](#), [ven::Model::createModelFromFile\(\)](#), [ven::Object::createObject\(\)](#), [ven::Colors::CYAN_4](#), [ven::Light::getId\(\)](#), [ven::Object::getId\(\)](#), [ven::Colors::GREEN_4](#),

[ven::Colors::MAGENTA_4](#), [ven::Colors::RED_4](#), [ven::Transform3DComponent::scale](#), [ven::Object::setModel\(\)](#), [ven::Object::setName\(\)](#), [ven::Light::transform3D](#), [ven::Object::transform3D](#), [ven::Transform3DComponent::translation](#), and [ven::Colors::YELLOW_4](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.4 mainLoop()

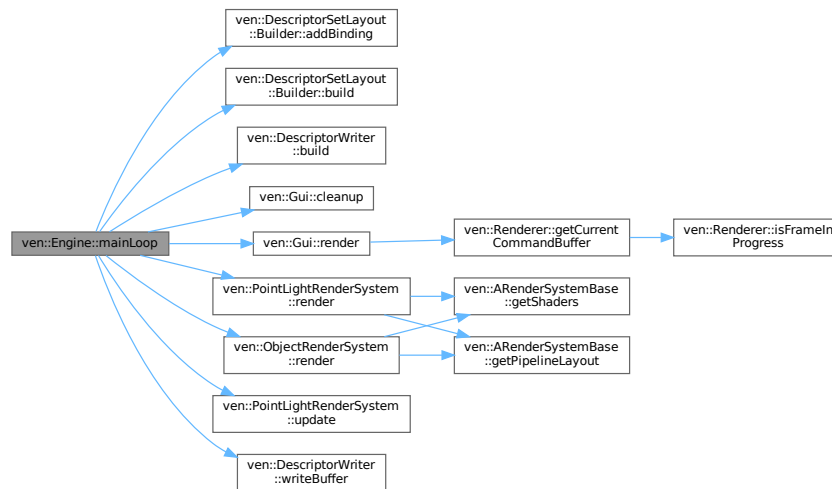
```
void ven::Engine::mainLoop ()
```

Definition at line 87 of file [engine.cpp](#).

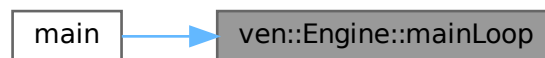
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::DescriptorWriter::build\(\)](#), [ven::Gui::cleanup\(\)](#), [ven::EXIT](#), [ven::FrameInfo::frameIndex](#), [ven::MAX_FRAMES_IN_FLIGHT](#), [ven::Gui::render\(\)](#), [ven::ObjectRenderSystem::render\(\)](#), [ven::PointLightRenderSystem::render\(\)](#), [ven::PointLightRenderSystem::update\(\)](#), [ven::VISIBLE](#), and [ven::DescriptorWriter::writeBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.5 operator=()

```

Engine ven::Engine::operator= (
    const Engine & ) [delete]
  
```

7.12.4 Member Data Documentation

7.12.4.1 m_device

```

Device ven::Engine::m_device {m_window} [private]
  
```

Definition at line 52 of file [Engine.hpp](#).

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

7.12.4.2 m_globalPool

```
std::unique_ptr<DescriptorPool> ven::Engine::m_globalPool [private]
```

Definition at line 55 of file [Engine.hpp](#).

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

7.12.4.3 m_gui

```
Gui ven::Engine::m_gui [private]
```

Definition at line 54 of file [Engine.hpp](#).

7.12.4.4 m_instance

```
VkInstance ven::Engine::m_instance {nullptr} [private]
```

Definition at line 59 of file [Engine.hpp](#).

Referenced by [createSurface\(\)](#), and [Engine\(\)](#).

7.12.4.5 m_lights

```
Light::Map ven::Engine::m_lights [private]
```

Definition at line 57 of file [Engine.hpp](#).

7.12.4.6 m_objects

```
Object::Map ven::Engine::m_objects [private]
```

Definition at line 56 of file [Engine.hpp](#).

7.12.4.7 m_renderer

```
Renderer ven::Engine::m_renderer {m_window, m_device} [private]
```

Definition at line 53 of file [Engine.hpp](#).

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

7.12.4.8 m_state

```
ENGINE_STATE ven::Engine::m_state {EXIT} [private]
```

Definition at line 49 of file [Engine.hpp](#).

7.12.4.9 m_surface

```
VkSurfaceKHR ven::Engine::m_surface {nullptr} [private]
```

Definition at line 60 of file [Engine.hpp](#).

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

7.12.4.10 m_window

```
Window ven::Engine::m_window [private]
```

Definition at line 51 of file [Engine.hpp](#).

Referenced by [createSurface\(\)](#), and [Engine\(\)](#).

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

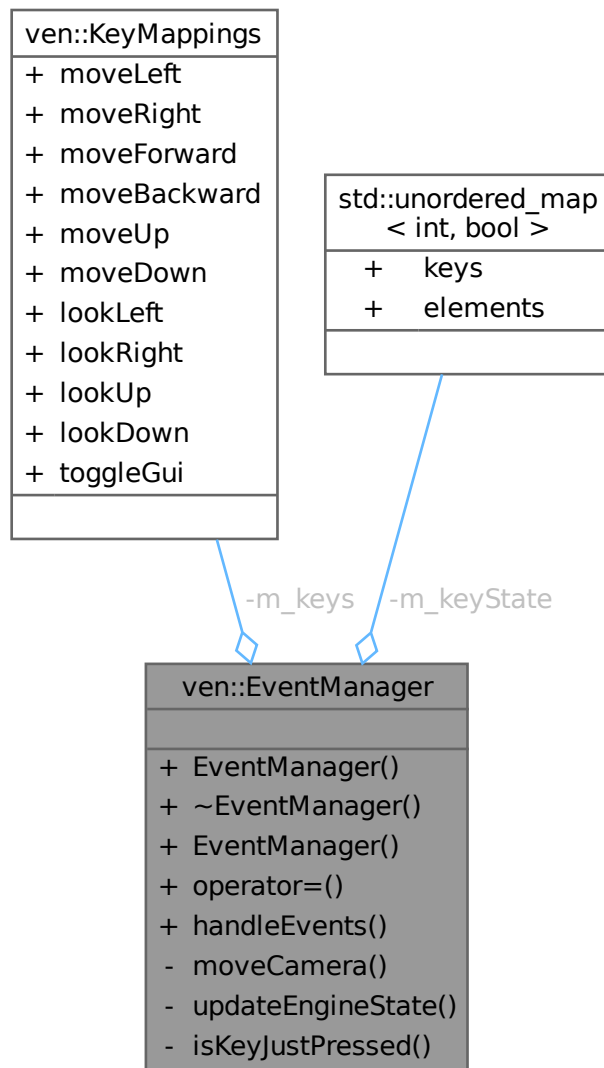
- [/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/engine.cpp](#)

7.13 ven::EventManager Class Reference

Class for event manager.

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::EventManager:



Public Member Functions

- `EventManager()`=default
- `~EventManager()`=default
- `EventManager (const EventManager &)=delete`
- `EventManager & operator= (const EventManager &)=delete`
- `void handleEvents (GLFWwindow *window, ENGINE_STATE *engineState, Camera &camera, Gui &gui, float dt) const`

Private Member Functions

- `void moveCamera (GLFWwindow *window, Camera &camera, Gui &gui, float dt) const`

Static Private Member Functions

- static void [updateEngineState](#) ([ENGINE_STATE](#) *engineState, const [ENGINE_STATE](#) newState)
- static bool [isKeyJustPressed](#) (GLFWwindow *window, int key, std::unordered_map< int, bool > &keyStates)

Private Attributes

- [KeyMappings](#) [m_keys](#) {}
- std::unordered_map< int, bool > [m_keyState](#)

7.13.1 Detailed Description

Class for event manager.

Definition at line 38 of file [EventManager.hpp](#).

7.13.2 Constructor & Destructor Documentation

7.13.2.1 EventManager() [1/2]

```
ven::EventManager::EventManager () [default]
```

7.13.2.2 ~EventManager()

```
ven::EventManager::~EventManager () [default]
```

7.13.2.3 EventManager() [2/2]

```
ven::EventManager::EventManager (
    const EventManager & ) [delete]
```

7.13.3 Member Function Documentation

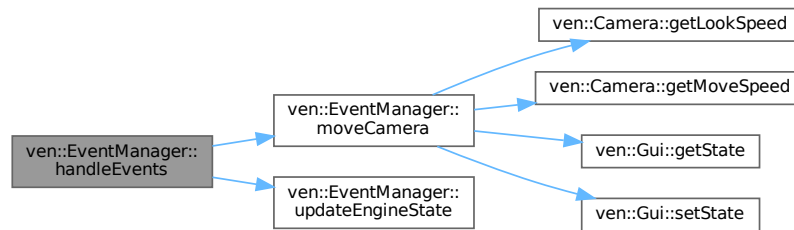
7.13.3.1 handleEvents()

```
void ven::EventManager::handleEvents (
    GLFWwindow * window,
    ENGINE\_STATE * engineState,
    Camera & camera,
    Gui & gui,
    float dt) const
```

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

References [ven::EXIT](#), [moveCamera\(\)](#), and [updateEngineState\(\)](#).

Here is the call graph for this function:



7.13.3.2 isKeyJustPressed()

```

bool ven::EventManager::isKeyJustPressed (
    GLFWwindow * window,
    int key,
    std::unordered_map< int, bool > & keyStates) [static], [private]
  
```

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

7.13.3.3 moveCamera()

```

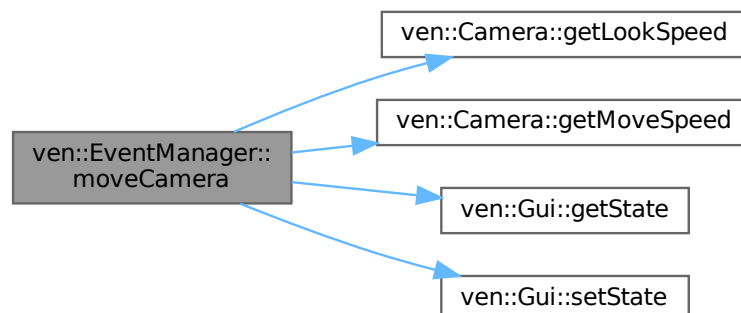
void ven::EventManager::moveCamera (
    GLFWwindow * window,
    Camera & camera,
    Gui & gui,
    float dt) const [private]
  
```

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

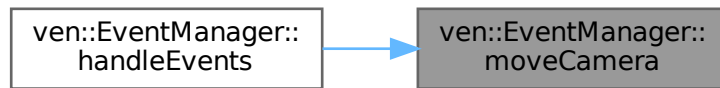
References [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::Gui::getState\(\)](#), [ven::HIDDEN](#), [ven::Transform3DComponent::rotation](#), [ven::Gui::setState\(\)](#), [ven::Camera::transform3D](#), [ven::Transform3DComponent::translation](#), and [ven::VISIBLE](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.4 operator=()

```
EventManager & ven::EventManager::operator= (
    const EventManager & ) [delete]
```

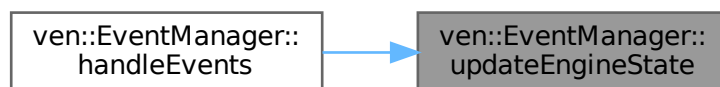
7.13.3.5 updateEngineState()

```
static void ven::EventManager::updateEngineState (
    ENGINE_STATE * engineState,
    const ENGINE_STATE newState) [inline], [static], [private]
```

Definition at line 53 of file [EventManager.hpp](#).

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

Here is the caller graph for this function:



7.13.4 Member Data Documentation

7.13.4.1 m_keys

```
KeyMappings ven::EventManager::m_keys {} [private]
```

Definition at line 56 of file [EventManager.hpp](#).

7.13.4.2 m_keyState

```
std::unordered_map<int, bool> ven::EventManager::m_keyState [mutable], [private]
```

Definition at line 57 of file [EventManager.hpp](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/eventManager.cpp](#)

7.14 ven::FrameInfo Struct Reference

```
#include <FrameInfo.hpp>
```


7.14.1 Detailed Description

Definition at line 38 of file [FrameInfo.hpp](#).

7.14.2 Member Data Documentation

7.14.2.1 camera

```
Camera& ven::FrameInfo::camera
```

Definition at line 43 of file [FrameInfo.hpp](#).

7.14.2.2 commandBuffer

```
VkCommandBuffer ven::FrameInfo::commandBuffer
```

Definition at line 42 of file [FrameInfo.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

7.14.2.3 frameIndex

```
unsigned long ven::FrameInfo::frameIndex
```

Definition at line 40 of file [FrameInfo.hpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

7.14.2.4 frameTime

```
float ven::FrameInfo::frameTime
```

Definition at line 41 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::update\(\)](#).

7.14.2.5 globalDescriptorSet

```
VkDescriptorSet ven::FrameInfo::globalDescriptorSet
```

Definition at line 44 of file [FrameInfo.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::render\(\)](#).

7.14.2.6 lights

`Light::Map& ven::FrameInfo::lights`

Definition at line 46 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::render\(\)](#), and [ven::PointLightRenderSystem::update\(\)](#).

7.14.2.7 objects

`Object::Map& ven::FrameInfo::objects`

Definition at line 45 of file [FrameInfo.hpp](#).

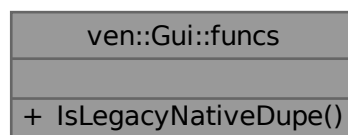
Referenced by [ven::ObjectRenderSystem::render\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp](#)

7.15 ven::Gui::funcs Struct Reference

Collaboration diagram for `ven::Gui::funcs`:



Static Public Member Functions

- static bool [IsLegacyNativeDupe](#) (const ImGuiKey key)

7.15.1 Detailed Description

Definition at line 58 of file [Gui.hpp](#).

7.15.2 Member Function Documentation

7.15.2.1 IsLegacyNativeDupe()

```
static bool ven::Gui::funcs::IsLegacyNativeDupe (  
    const ImGuiKey key) [inline], [static]
```

Definition at line 58 of file [Gui.hpp](#).

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



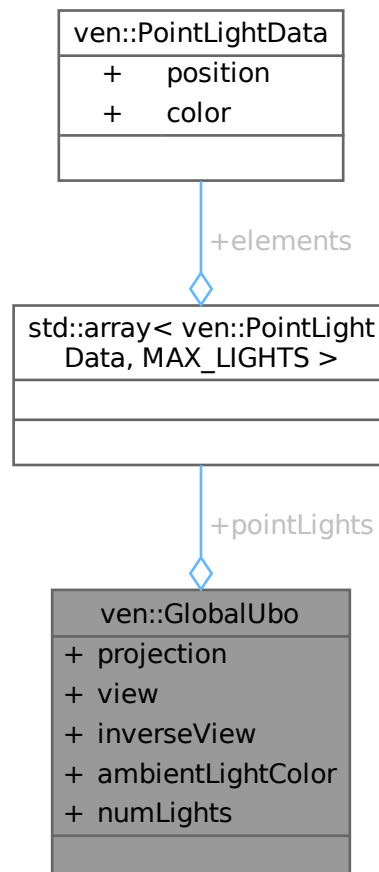
The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp](#)

7.16 ven::GlobalUbo Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::GlobalUbo:



Public Attributes

- glm::mat4 [projection](#) {1.F}
- glm::mat4 [view](#) {1.F}
- glm::mat4 [inverseView](#) {1.F}
- glm::vec4 [ambientLightColor](#) {DEFAULT_AMBIENT_LIGHT_COLOR}
- std::array< [PointLightData](#), MAX_LIGHTS > [pointLights](#)
- uint16_t [numLights](#)

7.16.1 Detailed Description

Definition at line 28 of file [FrameInfo.hpp](#).

7.16.2 Member Data Documentation

7.16.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {DEFAULT_AMBIENT_LIGHT_COLOR}
```

Definition at line 33 of file [FrameInfo.hpp](#).

Referenced by [ven::Gui::renderSection\(\)](#).

7.16.2.2 inverseView

```
glm::mat4 ven::GlobalUbo::inverseView {1.F}
```

Definition at line 32 of file [FrameInfo.hpp](#).

7.16.2.3 numLights

```
uint16_t ven::GlobalUbo::numLights
```

Definition at line 35 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::update\(\)](#).

7.16.2.4 pointLights

```
std::array<PointLightData, MAX_LIGHTS> ven::GlobalUbo::pointLights
```

Definition at line 34 of file [FrameInfo.hpp](#).

Referenced by [ven::PointLightRenderSystem::update\(\)](#).

7.16.2.5 projection

```
glm::mat4 ven::GlobalUbo::projection {1.F}
```

Definition at line 30 of file [FrameInfo.hpp](#).

7.16.2.6 view

```
glm::mat4 ven::GlobalUbo::view {1.F}
```

Definition at line 31 of file [FrameInfo.hpp](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp](#)

7.17 ven::Gui Class Reference

Class for [Gui](#).

```
#include <Gui.hpp>
```

Collaboration diagram for ven::Gui:

ven::Gui
<ul style="list-style-type: none"> - m_state - m_io
<ul style="list-style-type: none"> + Gui() + ~Gui() + Gui() + operator=() + setState() + getState() + init() + render() + cleanup() - initStyle() - renderFrameWindow() - cameraSection() - inputsSection() - rendererSection() - devicePropertiesSection() - objectsSection() - lightsSection()

Classes

- struct [funcs](#)

Public Member Functions

- [Gui](#) ()=default
- [~Gui](#) ()=default
- [Gui](#) (const [Gui](#) &)=delete
- [Gui](#) & [operator=](#) (const [Gui](#) &)=delete
- void [setState](#) (const [GUI_STATE](#) state)
- [GUI_STATE](#) [getState](#) () const

Static Public Member Functions

- static void `init` (GLFWwindow *window, VkInstance instance, const `Device` *device, VkRenderPass renderPass)
- static void `render` (`Renderer` *renderer, std::unordered_map< unsigned int, `Object` > &objects, std::unordered_map< unsigned int, `Light` > &lights, `Camera` &camera, VkPhysicalDevice physicalDevice, `GlobalUbo` &ubo)
- static void `cleanup` ()

Static Private Member Functions

- static void `initStyle` ()
- static void `renderFrameWindow` ()
- static void `cameraSection` (`Camera` &camera)
- static void `inputsSection` (const ImGuiIO *io)
- static void `rendererSection` (`Renderer` *renderer, `GlobalUbo` &ubo)
- static void `devicePropertiesSection` (VkPhysicalDeviceProperties deviceProperties)
- static void `objectsSection` (std::unordered_map< unsigned int, `Object` > &objects)
- static void `lightsSection` (std::unordered_map< unsigned int, `Light` > &lights)

Private Attributes

- `GUI_STATE m_state` {VISIBLE}

Static Private Attributes

- static ImGuiIO * `m_io` = nullptr

7.17.1 Detailed Description

Class for `Gui`.

Definition at line 28 of file `Gui.hpp`.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 Gui() [1/2]

```
ven::Gui::Gui () [default]
```

7.17.2.2 ~Gui()

```
ven::Gui::~Gui () [default]
```

7.17.2.3 Gui() [2/2]

```
ven::Gui::Gui (
    const Gui & ) [delete]
```

7.17.3 Member Function Documentation

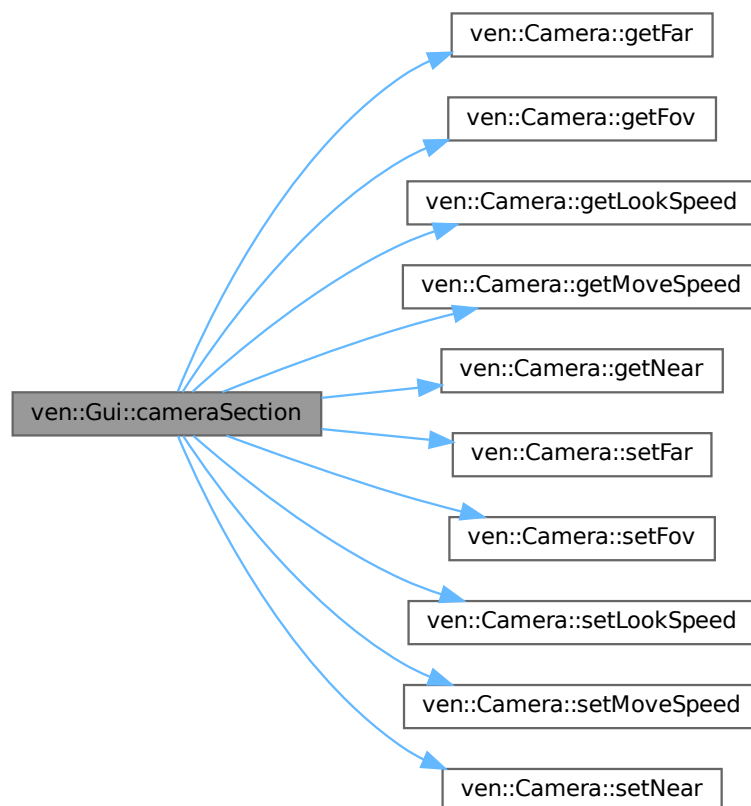
7.17.3.1 cameraSection()

```
void ven::Gui::cameraSection (
    Camera & camera) [static], [private]
```

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

References [ven::DEFAULT_FAR](#), [ven::DEFAULT_FOV](#), [ven::DEFAULT_LOOK_SPEED](#), [ven::DEFAULT_MOVE_SPEED](#), [ven::DEFAULT_NEAR](#), [ven::DEFAULT_POSITION](#), [ven::DEFAULT_ROTATION](#), [ven::Camera::getFar\(\)](#), [ven::Camera::getFov\(\)](#), [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::Camera::getNear\(\)](#), [ven::Transform3DComponent::rotation](#), [ven::Camera::setFar\(\)](#), [ven::Camera::setFov\(\)](#), [ven::Camera::setLookSpeed\(\)](#), [ven::Camera::setMoveSpeed\(\)](#), [ven::Camera::setNear\(\)](#), [ven::Camera::transform3D](#), and [ven::Transform3DComponent::translation](#).

Here is the call graph for this function:



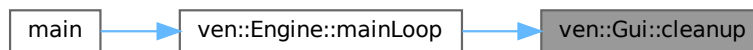
7.17.3.2 cleanup()

```
void ven::Gui::cleanup () [static]
```

Definition at line 9 of file [render.cpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.17.3.3 devicePropertiesSection()

```
void ven::Gui::devicePropertiesSection (
    VkPhysicalDeviceProperties deviceProperties) [static], [private]
```

Definition at line 244 of file [render.cpp](#).

7.17.3.4 getState()

```
GUI_STATE ven::Gui::getState () const [inline], [nodiscard]
```

Definition at line 44 of file [Gui.hpp](#).

References [m_state](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



7.17.3.5 init()

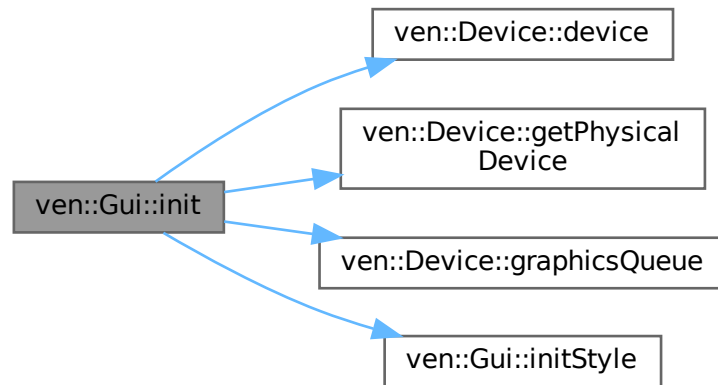
```
void ven::Gui::init (
    GLFWwindow * window,
    VkInstance instance,
    const Device * device,
    VkRenderPass renderPass) [static]
```

Definition at line 11 of file [init.cpp](#).

References [DESCRIPTOR_COUNT](#), [ven::Device::device\(\)](#), [ven::Device::getPhysicalDevice\(\)](#), [ven::Device::graphicsQueue\(\)](#), [initStyle\(\)](#), and [m_io](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



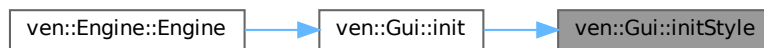
7.17.3.6 initStyle()

```
void ven::Gui::initStyle () [static], [private]
```

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

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

Here is the caller graph for this function:



7.17.3.7 inputsSection()

```
void ven::Gui::inputsSection (
    const ImGuiIO * io) [static], [private]
```

Definition at line 222 of file [render.cpp](#).

7.17.3.8 lightsSection()

```
void ven::Gui::lightsSection (
    std::unordered_map< unsigned int, Light > & lights) [static], [private]
```

Definition at line 178 of file [render.cpp](#).

References [ven::Colors::COLOR_PRESETS_3](#), and [ven::DEFAULT_LIGHT_INTENSITY](#).

7.17.3.9 objectsSection()

```
void ven::Gui::objectsSection (
    std::unordered_map< unsigned int, Object > & objects) [static], [private]
```

Definition at line 159 of file [render.cpp](#).

References [ven::Colors::GRAY_4](#).

7.17.3.10 operator=()

```
Gui & ven::Gui::operator= (
    const Gui & ) [delete]
```

7.17.3.11 render()

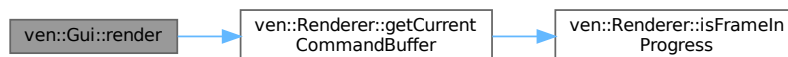
```
void ven::Gui::render (
    Renderer * renderer,
    std::unordered_map< unsigned int, Object > & objects,
    std::unordered_map< unsigned int, Light > & lights,
    Camera & camera,
    VkPhysicalDevice physicalDevice,
    GlobalUbo & ubo) [static]
```

Definition at line 16 of file [render.cpp](#).

References [ven::Renderer::getCurrentCommandBuffer\(\)](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



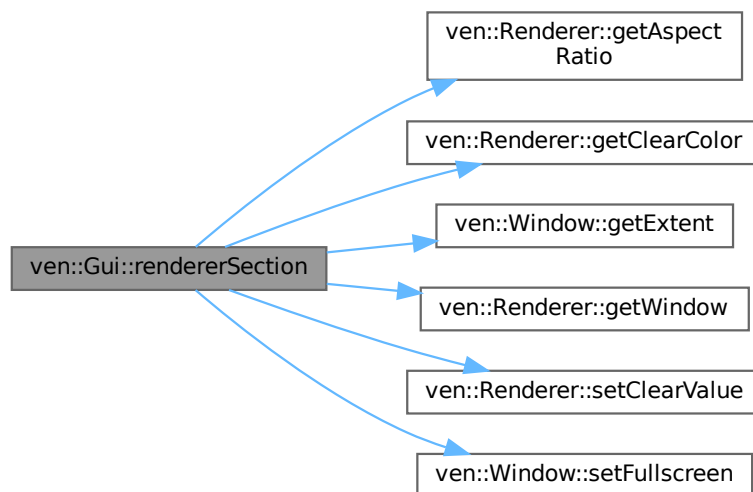
7.17.3.12 rendererSection()

```
void ven::Gui::rendererSection (
    Renderer * renderer,
    GlobalUbo & ubo) [static], [private]
```

Definition at line 51 of file [render.cpp](#).

References [ven::GlobalUbo::ambientLightColor](#), [ven::Colors::COLOR_PRESETS_4](#), [ven::Colors::COLOR_PRESETS_VK](#), [ven::DEFAULT_AMBIENT_LIGHT_INTENSITY](#), [ven::Renderer::getAspectRatio\(\)](#), [ven::Renderer::getClearColor\(\)](#), [ven::Window::getExtent\(\)](#), [ven::Renderer::getWindow\(\)](#), [ven::Renderer::setClearColor\(\)](#), and [ven::Window::setFullscreen\(\)](#).

Here is the call graph for this function:



7.17.3.13 renderFrameWindow()

```
void ven::Gui::renderFrameWindow () [static], [private]
```

Definition at line 40 of file [render.cpp](#).

7.17.3.14 setState()

```
void ven::Gui::setState (
    const GUI_STATE state) [inline]
```

Definition at line 43 of file [Gui.hpp](#).

References [m_state](#).

Referenced by [ven::EventManager::moveCamera\(\)](#).

Here is the caller graph for this function:



7.17.4 Member Data Documentation

7.17.4.1 m_io

```
ImGuiIO * ven::Gui::m_io = nullptr [static], [private]
```

Definition at line 60 of file [Gui.hpp](#).

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

7.17.4.2 m_state

```
GUI_STATE ven::Gui::m_state {VISIBLE} [private]
```

Definition at line 61 of file [Gui.hpp](#).

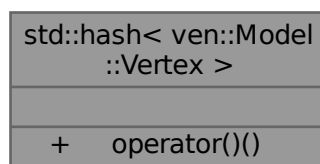
Referenced by [getState\(\)](#), and [setState\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/init.cpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/render.cpp](#)

7.18 std::hash< ven::Model::Vertex > Struct Reference

Collaboration diagram for `std::hash< ven::Model::Vertex >`:



Public Member Functions

- `size_t operator() (ven::Model::Vertex const &vertex) const` noexcept

7.18.1 Detailed Description

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

7.18.2 Member Function Documentation

7.18.2.1 operator()

```
size_t std::hash< ven::Model::Vertex >::operator() (
    ven::Model::Vertex const & vertex) const    [inline], [noexcept]
```

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

References [ven::hashCombine\(\)](#).

Here is the call graph for this function:



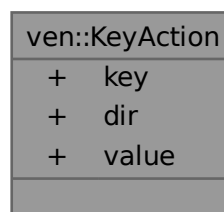
The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.19 ven::KeyAction Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyAction:



Public Attributes

- uint16_t [key](#)
- glm::vec3 * [dir](#)
- glm::vec3 [value](#)

7.19.1 Detailed Description

Definition at line 13 of file [EventManager.hpp](#).

7.19.2 Member Data Documentation

7.19.2.1 dir

```
glm::vec3* ven::KeyAction::dir
```

Definition at line 15 of file [EventManager.hpp](#).

7.19.2.2 key

```
uint16_t ven::KeyAction::key
```

Definition at line 14 of file [EventManager.hpp](#).

7.19.2.3 value

```
glm::vec3 ven::KeyAction::value
```

Definition at line 16 of file [EventManager.hpp](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp](#)

7.20 ven::KeyMappings Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyMappings:

ven::KeyMappings
+ moveLeft
+ moveRight
+ moveForward
+ moveBackward
+ moveUp
+ moveDown
+ lookLeft
+ lookRight
+ lookUp
+ lookDown
+ toggleGui

Public Attributes

- uint16_t [moveLeft](#) = GLFW_KEY_A
- uint16_t [moveRight](#) = GLFW_KEY_D
- uint16_t [moveForward](#) = GLFW_KEY_W
- uint16_t [moveBackward](#) = GLFW_KEY_S
- uint16_t [moveUp](#) = GLFW_KEY_SPACE
- uint16_t [moveDown](#) = GLFW_KEY_LEFT_SHIFT
- uint16_t [lookLeft](#) = GLFW_KEY_LEFT
- uint16_t [lookRight](#) = GLFW_KEY_RIGHT
- uint16_t [lookUp](#) = GLFW_KEY_UP
- uint16_t [lookDown](#) = GLFW_KEY_DOWN
- uint16_t [toggleGui](#) = GLFW_KEY_F1

7.20.1 Detailed Description

Definition at line 19 of file [EventManager.hpp](#).

7.20.2 Member Data Documentation

7.20.2.1 lookDown

```
uint16_t ven::KeyMappings::lookDown = GLFW_KEY_DOWN
```

Definition at line 29 of file [EventManager.hpp](#).

7.20.2.2 lookLeft

```
uint16_t ven::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

Definition at line 26 of file [EventManager.hpp](#).

7.20.2.3 lookRight

```
uint16_t ven::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

Definition at line 27 of file [EventManager.hpp](#).

7.20.2.4 lookUp

```
uint16_t ven::KeyMappings::lookUp = GLFW_KEY_UP
```

Definition at line 28 of file [EventManager.hpp](#).

7.20.2.5 moveBackward

```
uint16_t ven::KeyMappings::moveBackward = GLFW_KEY_S
```

Definition at line 23 of file [EventManager.hpp](#).

7.20.2.6 moveDown

```
uint16_t ven::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

Definition at line 25 of file [EventManager.hpp](#).

7.20.2.7 moveForward

```
uint16_t ven::KeyMappings::moveForward = GLFW_KEY_W
```

Definition at line 22 of file [EventManager.hpp](#).

7.20.2.8 moveLeft

```
uint16_t ven::KeyMappings::moveLeft = GLFW_KEY_A
```

Definition at line 20 of file [EventManager.hpp](#).

7.20.2.9 moveRight

```
uint16_t ven::KeyMappings::moveRight = GLFW_KEY_D
```

Definition at line 21 of file [EventManager.hpp](#).

7.20.2.10 moveUp

```
uint16_t ven::KeyMappings::moveUp = GLFW_KEY_SPACE
```

Definition at line 24 of file [EventManager.hpp](#).

7.20.2.11 toggleGui

```
uint16_t ven::KeyMappings::toggleGui = GLFW_KEY_F1
```

Definition at line 30 of file [EventManager.hpp](#).

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

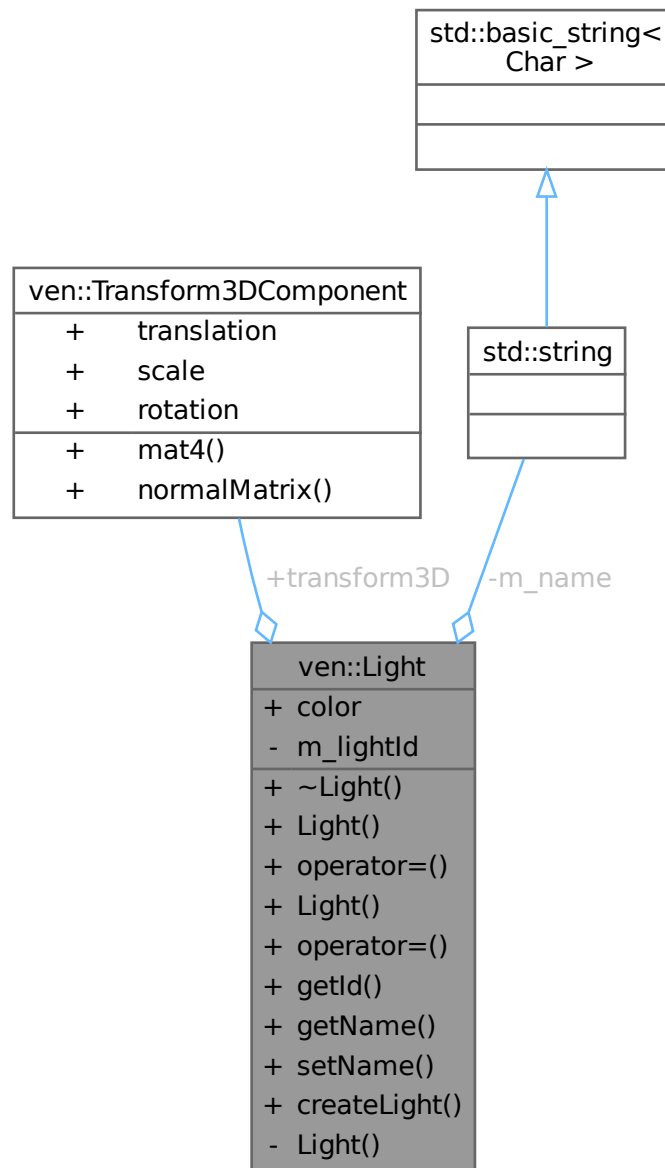
- [/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp](#)

7.21 ven::Light Class Reference

Class for light.

```
#include <Light.hpp>
```

Collaboration diagram for ven::Light:



Public Types

- using `Map` = `std::unordered_map<unsigned int, Light>`

Public Member Functions

- `~Light()`=default
- `Light` (const `Light` &)=delete
- `Light` & `operator=` (const `Light` &)=delete

- [Light](#) ([Light](#) &&)=default
- [Light](#) & [operator=](#) ([Light](#) &&)=default
- unsigned int [getId](#) () const
- std::string [getName](#) () const
- void [setName](#) (const std::string &name)

Static Public Member Functions

- static [Light](#) [createLight](#) (float radius=[DEFAULT_LIGHT_RADIUS](#), glm::vec4 [color](#)=[DEFAULT_LIGHT_COLOR](#))

Public Attributes

- glm::vec4 [color](#) {[DEFAULT_LIGHT_COLOR](#)}
- [Transform3DComponent](#) [transform3D](#) {}

Private Member Functions

- [Light](#) (const unsigned int [lightId](#))

Private Attributes

- unsigned int [m_lightId](#)
- std::string [m_name](#) {"point light"}

7.21.1 Detailed Description

Class for light.

Definition at line 27 of file [Light.hpp](#).

7.21.2 Member Typedef Documentation

7.21.2.1 Map

```
using ven::Light::Map = std::unordered_map<unsigned int, Light>
```

Definition at line 31 of file [Light.hpp](#).

7.21.3 Constructor & Destructor Documentation

7.21.3.1 ~Light()

```
ven::Light::~Light () [default]
```

7.21.3.2 Light() [1/3]

```
ven::Light::Light (
    const Light & ) [delete]
```

7.21.3.3 Light() [2/3]

```
ven::Light::Light (
    Light && ) [default]
```

7.21.3.4 Light() [3/3]

```
ven::Light::Light (
    const unsigned int lightId) [inline], [explicit], [private]
```

Definition at line 52 of file [Light.hpp](#).

7.21.4 Member Function Documentation**7.21.4.1 createLight()**

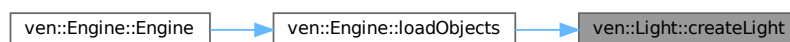
```
ven::Light ven::Light::createLight (
    float radius = DEFAULT\_LIGHT\_RADIUS,
    glm::vec4 color = DEFAULT\_LIGHT\_COLOR) [static]
```

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

References [color](#), [ven::Transform3DComponent::scale](#), and [transform3D](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:

**7.21.4.2 getId()**

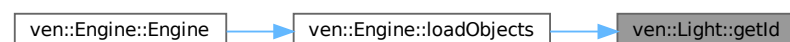
```
unsigned int ven::Light::getId () const [inline], [nodiscard]
```

Definition at line 45 of file [Light.hpp](#).

References [m_lightId](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.21.4.3 getName()

```
std::string ven::Light::getName () const [inline], [nodiscard]
```

Definition at line 46 of file [Light.hpp](#).

References [m_name](#).

7.21.4.4 operator=() [1/2]

```
Light & ven::Light::operator= (  
    const Light & ) [delete]
```

7.21.4.5 operator=() [2/2]

```
Light & ven::Light::operator= (  
    Light && ) [default]
```

7.21.4.6 setName()

```
void ven::Light::setName (  
    const std::string & name) [inline]
```

Definition at line 48 of file [Light.hpp](#).

References [m_name](#).

7.21.5 Member Data Documentation

7.21.5.1 color

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

Definition at line 42 of file [Light.hpp](#).

Referenced by [createLight\(\)](#), and [ven::Engine::loadObjects\(\)](#).

7.21.5.2 m_lightId

```
unsigned int ven::Light::m_lightId [private]
```

Definition at line 54 of file [Light.hpp](#).

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

7.21.5.3 m_name

```
std::string ven::Light::m_name {"point light"} [private]
```

Definition at line 55 of file [Light.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.21.5.4 transform3D

```
Transform3DComponent ven::Light::transform3D {}
```

Definition at line 43 of file [Light.hpp](#).

Referenced by [createLight\(\)](#), and [ven::Engine::loadObjects\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/light.cpp](#)

7.22 ven::LightPushConstantData Struct Reference

```
#include <PointLightRenderSystem.hpp>
```

Collaboration diagram for [ven::LightPushConstantData](#):

ven::LightPushConstantData	
+	position
+	color
+	radius

Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [radius](#)

7.22.1 Detailed Description

Definition at line 14 of file [PointLightRenderSystem.hpp](#).

7.22.2 Member Data Documentation

7.22.2.1 color

```
glm::vec4 ven::LightPushConstantData::color {}
```

Definition at line 16 of file [PointLightRenderSystem.hpp](#).

7.22.2.2 position

```
glm::vec4 ven::LightPushConstantData::position {}
```

Definition at line 15 of file [PointLightRenderSystem.hpp](#).

Referenced by [ven::PointLightRenderSystem::render\(\)](#).

7.22.2.3 radius

```
float ven::LightPushConstantData::radius
```

Definition at line 17 of file [PointLightRenderSystem.hpp](#).

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

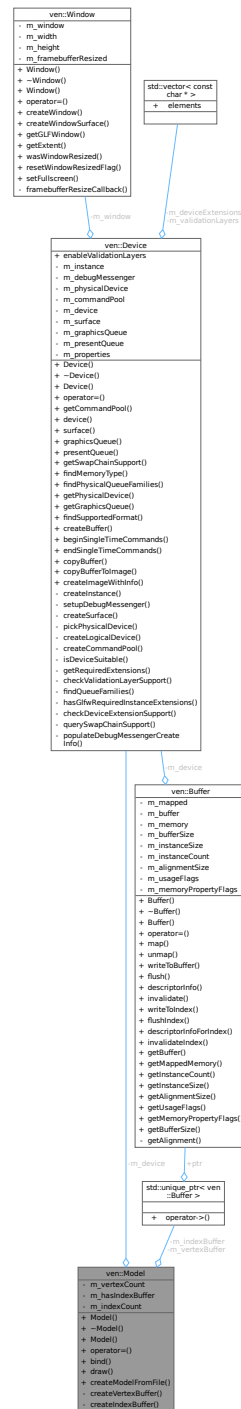
- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp](#)

7.23 ven::Model Class Reference

Class for model.

```
#include <Model.hpp>
```

Collaboration diagram for `ven::Model`:



Classes

- struct [Builder](#)
- struct [Vertex](#)

Public Member Functions

- [Model](#) ([Device](#) &device, const [Builder](#) &builder)

- `~Model()`=default
- `Model(const Model &)=delete`
- `void operator=(const Model &)=delete`
- `void bind(VkCommandBuffer commandBuffer) const`
- `void draw(VkCommandBuffer commandBuffer) const`

Static Public Member Functions

- `static std::unique_ptr< Model > createModelFromFile(Device &device, const std::string &filename)`

Private Member Functions

- `void createVertexBuffer(const std::vector< Vertex > &vertices)`
- `void createIndexBuffer(const std::vector< uint32_t > &indices)`

Private Attributes

- `Device & m_device`
- `std::unique_ptr< Buffer > m_vertexBuffer`
- `uint32_t m_vertexCount`
- `bool m_hasIndexBuffer {false}`
- `std::unique_ptr< Buffer > m_indexBuffer`
- `uint32_t m_indexCount`

7.23.1 Detailed Description

Class for model.

Definition at line 25 of file [Model.hpp](#).

7.23.2 Constructor & Destructor Documentation

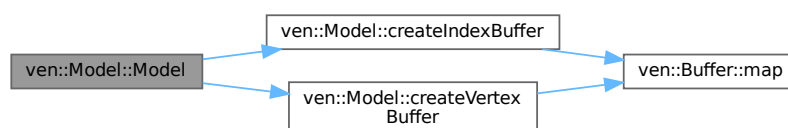
7.23.2.1 Model() [1/2]

```
ven::Model::Model (
    Device & device,
    const Builder & builder)
```

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

References [createIndexBuffer\(\)](#), [createVertexBuffer\(\)](#), [ven::Model::Builder::indices](#), and [ven::Model::Builder::vertices](#).

Here is the call graph for this function:



7.23.2.2 ~Model()

```
ven::Model::~~Model () [default]
```

7.23.2.3 Model() [2/2]

```
ven::Model::Model (
    const Model & ) [delete]
```

7.23.3 Member Function Documentation

7.23.3.1 bind()

```
void ven::Model::bind (
    VkCommandBuffer commandBuffer) const
```

Definition at line 78 of file [model.cpp](#).

7.23.3.2 createIndexBuffer()

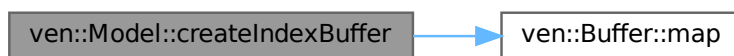
```
void ven::Model::createIndexBuffer (
    const std::vector< uint32_t > & indices) [private]
```

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

References [ven::Buffer::map\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.3 createModelFromFile()

```
std::unique_ptr< ven::Model > ven::Model::createModelFromFile (
    Device & device,
    const std::string & filename) [static]
```

Definition at line 89 of file [model.cpp](#).

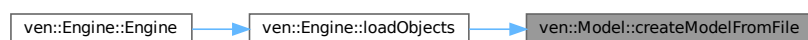
References [ven::Model::Builder::loadModel\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.4 createVertexBuffer()

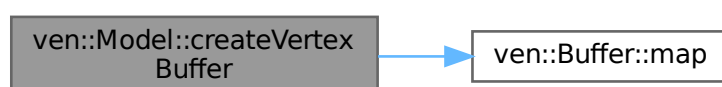
```
void ven::Model::createVertexBuffer (
    const std::vector< Vertex > & vertices) [private]
```

Definition at line 31 of file [model.cpp](#).

References [ven::Buffer::map\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.5 draw()

```
void ven::Model::draw (
    VkCommandBuffer commandBuffer) const
```

Definition at line 69 of file [model.cpp](#).

7.23.3.6 operator=()

```
void ven::Model::operator= (
    const Model & ) [delete]
```

7.23.4 Member Data Documentation

7.23.4.1 m_device

```
Device& ven::Model::m_device [private]
```

Definition at line 68 of file [Model.hpp](#).

7.23.4.2 m_hasIndexBuffer

```
bool ven::Model::m_hasIndexBuffer {false} [private]
```

Definition at line 72 of file [Model.hpp](#).

7.23.4.3 m_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

Definition at line 73 of file [Model.hpp](#).

7.23.4.4 m_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

Definition at line 74 of file [Model.hpp](#).

7.23.4.5 m_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

Definition at line 69 of file [Model.hpp](#).

7.23.4.6 m_vertexCount

```
uint32_t ven::Model::m_vertexCount [private]
```

Definition at line 70 of file [Model.hpp](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.24 ven::Object Class Reference

Class for object.

```
#include <Object.hpp>
```


- `Object` (const `Object` &)=delete
- `Object & operator=` (const `Object` &)=delete
- `Object` (`Object` &&)=default
- `Object & operator=` (`Object` &&)=default
- unsigned int `getId` () const
- std::string `getName` () const
- std::shared_ptr< `Model` > `getModel` () const
- void `setName` (const std::string &name)
- void `setModel` (const std::shared_ptr< `Model` > &model)

Static Public Member Functions

- static `Object` `createObject` ()

Public Attributes

- `Transform3DComponent` `transform3D` {}

Private Member Functions

- `Object` (const unsigned int objId)

Private Attributes

- unsigned int `m_objId`
- std::string `m_name`
- std::shared_ptr< `Model` > `m_model`

7.24.1 Detailed Description

Class for object.

Definition at line 24 of file `Object.hpp`.

7.24.2 Member Typedef Documentation

7.24.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

Definition at line 28 of file `Object.hpp`.

7.24.3 Constructor & Destructor Documentation

7.24.3.1 ~Object()

```
ven::Object::~Object () [default]
```

7.24.3.2 `Object()` [1/3]

```
ven::Object::Object (
    const Object & ) [delete]
```

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

Here is the caller graph for this function:



7.24.3.3 `Object()` [2/3]

```
ven::Object::Object (
    Object && ) [default]
```

7.24.3.4 `Object()` [3/3]

```
ven::Object::Object (
    const unsigned int objId) [inline], [explicit], [private]
```

Definition at line 50 of file [Object.hpp](#).

7.24.4 Member Function Documentation

7.24.4.1 `createObject()`

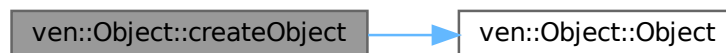
```
static Object ven::Object::createObject () [inline], [static]
```

Definition at line 37 of file [Object.hpp](#).

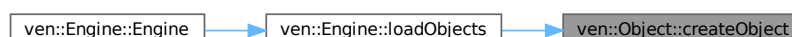
References [Object\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.4.2 getId()

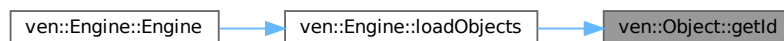
```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

Definition at line 39 of file [Object.hpp](#).

References [m_objId](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.4.3 getModel()

```
std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]
```

Definition at line 41 of file [Object.hpp](#).

References [m_model](#).

7.24.4.4 getName()

```
std::string ven::Object::getName () const [inline], [nodiscard]
```

Definition at line 40 of file [Object.hpp](#).

References [m_name](#).

7.24.4.5 operator=() [1/2]

```
Object & ven::Object::operator= (  
    const Object & ) [delete]
```

7.24.4.6 operator=() [2/2]

```
Object & ven::Object::operator= (  
    Object && ) [default]
```

7.24.4.7 setModel()

```
void ven::Object::setModel (
    const std::shared_ptr< Model > & model) [inline]
```

Definition at line 44 of file [Object.hpp](#).

References [m_model](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.4.8 setName()

```
void ven::Object::setName (
    const std::string & name) [inline]
```

Definition at line 43 of file [Object.hpp](#).

References [m_name](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.5 Member Data Documentation

7.24.5.1 m_model

```
std::shared_ptr<Model> ven::Object::m_model [private]
```

Definition at line 54 of file [Object.hpp](#).

Referenced by [getModel\(\)](#), and [setModel\(\)](#).

7.24.5.2 m_name

```
std::string ven::Object::m_name [private]
```

Definition at line 53 of file [Object.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.24.5.3 m_objId

```
unsigned int ven::Object::m_objId [private]
```

Definition at line 52 of file [Object.hpp](#).

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

7.24.5.4 transform3D

```
Transform3DComponent ven::Object::transform3D {}
```

Definition at line 46 of file [Object.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#)

7.25 ven::ObjectPushConstantData Struct Reference

```
#include <ObjectRenderSystem.hpp>
```

Collaboration diagram for ven::ObjectPushConstantData:

ven::ObjectPushConstantData	
+	modelMatrix
+	normalMatrix

Public Attributes

- glm::mat4 [modelMatrix](#) {}
- glm::mat4 [normalMatrix](#) {}

7.25.1 Detailed Description

Definition at line 14 of file [ObjectRenderSystem.hpp](#).

7.25.2 Member Data Documentation

7.25.2.1 modelMatrix

```
glm::mat4 ven::ObjectPushConstantData::modelMatrix {}
```

Definition at line 15 of file [ObjectRenderSystem.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#).

7.25.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

Definition at line 16 of file [ObjectRenderSystem.hpp](#).

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

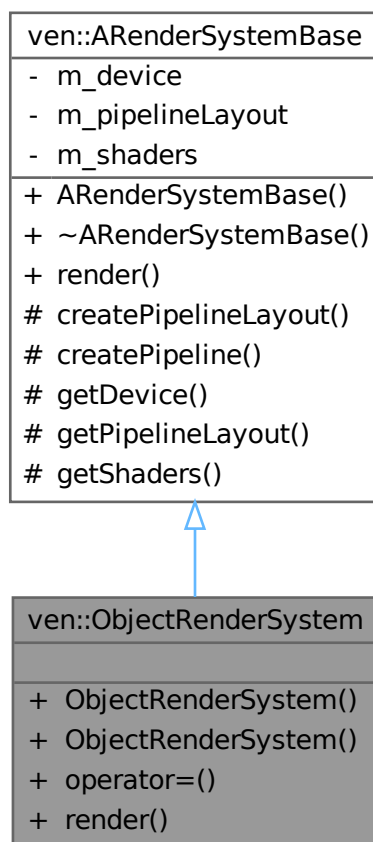
- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp](#)

7.26 ven::ObjectRenderSystem Class Reference

Class for object render system.

```
#include <ObjectRenderSystem.hpp>
```

Inheritance diagram for ven::ObjectRenderSystem:



Collaboration diagram for `ven::ObjectRenderSystem`:



Public Member Functions

- `ObjectRenderSystem` (`Device` &device, const `VkRenderPass` renderPass, const `VkDescriptorSetLayout` globalSetLayout)
- `ObjectRenderSystem` (const `ObjectRenderSystem` &)=delete
- `ObjectRenderSystem` & operator= (const `ObjectRenderSystem` &)=delete
- void `render` (const `FrameInfo` &frameInfo) const override

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- virtual [~ARenderSystemBase](#) ()

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

7.26.1 Detailed Description

Class for object render system.

Definition at line 24 of file [ObjectRenderSystem.hpp](#).

7.26.2 Constructor & Destructor Documentation

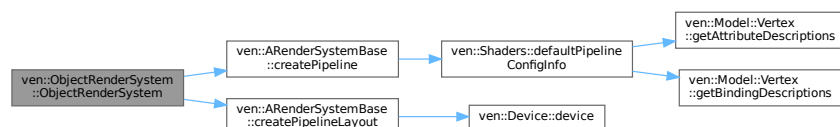
7.26.2.1 ObjectRenderSystem() [1/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 28 of file [ObjectRenderSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.26.2.2 ObjectRenderSystem() [2/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    const ObjectRenderSystem & ) [delete]
```

7.26.3 Member Function Documentation

7.26.3.1 operator=()

```
ObjectRenderSystem & ven::ObjectRenderSystem::operator= (
    const ObjectRenderSystem & ) [delete]
```

7.26.3.2 render()

```
void ven::ObjectRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

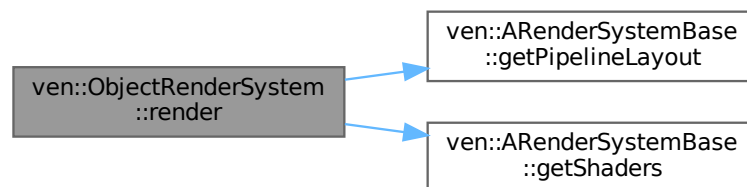
Implements [ven::ARenderSystemBase](#).

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

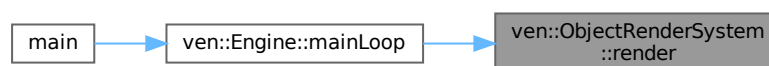
References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::ObjectPushConstantData::modelMatrix](#), and [ven::FrameInfo::objects](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



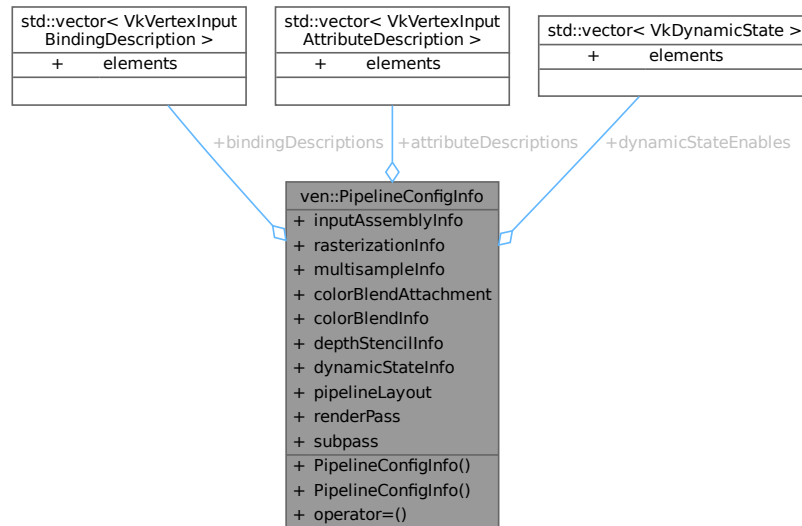
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp](#)

7.27 ven::PipelineConfigInfo Struct Reference

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::PipelineConfigInfo:



Public Member Functions

- [PipelineConfigInfo](#) ()=default
- [PipelineConfigInfo](#) (const [PipelineConfigInfo](#) &)=delete
- [PipelineConfigInfo](#) & [operator=](#) (const [PipelineConfigInfo](#) &)=delete

Public Attributes

- `std::vector< VkVertexInputBindingDescription >` [bindingDescriptions](#)
- `std::vector< VkVertexInputAttributeDescription >` [attributeDescriptions](#)
- `VkPipelineInputAssemblyStateCreateInfo` [inputAssemblyInfo](#) {}
- `VkPipelineRasterizationStateCreateInfo` [rasterizationInfo](#) {}
- `VkPipelineMultisampleStateCreateInfo` [multisampleInfo](#) {}
- `VkPipelineColorBlendAttachmentState` [colorBlendAttachment](#) {}
- `VkPipelineColorBlendStateCreateInfo` [colorBlendInfo](#) {}
- `VkPipelineDepthStencilStateCreateInfo` [depthStencilInfo](#) {}
- `std::vector< VkDynamicState >` [dynamicStateEnables](#)
- `VkPipelineDynamicStateCreateInfo` [dynamicStateInfo](#) {}
- `VkPipelineLayout` [pipelineLayout](#) = nullptr
- `VkRenderPass` [renderPass](#) = nullptr
- `uint32_t` [subpass](#) = 0

7.27.1 Detailed Description

Definition at line 19 of file [Shaders.hpp](#).

7.27.2 Constructor & Destructor Documentation

7.27.2.1 PipelineConfigInfo() [1/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo () [default]
```

7.27.2.2 PipelineConfigInfo() [2/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo (
    const PipelineConfigInfo & ) [delete]
```

7.27.3 Member Function Documentation

7.27.3.1 operator=()

```
PipelineConfigInfo & ven::PipelineConfigInfo::operator= (
    const PipelineConfigInfo & ) [delete]
```

7.27.4 Member Data Documentation

7.27.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

Definition at line 25 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

Definition at line 24 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

Definition at line 29 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

Definition at line 30 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

Definition at line 31 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

Definition at line 32 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

Definition at line 33 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

Definition at line 26 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

Definition at line 28 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

Definition at line 34 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.27.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

Definition at line 27 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.27.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

Definition at line 35 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.27.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 36 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

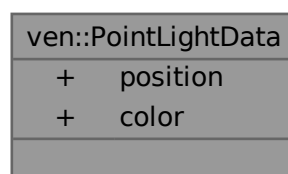
The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)

7.28 ven::PointLightData Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for `ven::PointLightData`:



Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}

7.28.1 Detailed Description

Definition at line 22 of file [FrameInfo.hpp](#).

7.28.2 Member Data Documentation

7.28.2.1 color

```
glm::vec4 ven::PointLightData::color {}
```

Definition at line 25 of file [FrameInfo.hpp](#).

7.28.2.2 position

```
glm::vec4 ven::PointLightData::position {}
```

Definition at line 24 of file [FrameInfo.hpp](#).

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

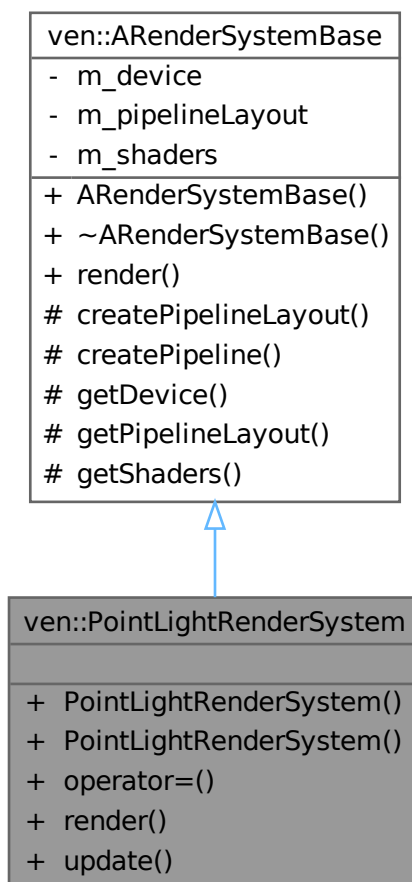
- /home/runner/work/VEngine/VEngine/include/VEngine/[FrameInfo.hpp](#)

7.29 ven::PointLightRenderSystem Class Reference

Class for point light system.

```
#include <PointLightRenderSystem.hpp>
```

Inheritance diagram for ven::PointLightRenderSystem:



Collaboration diagram for ven::PointLightRenderSystem:



Public Member Functions

- [PointLightRenderSystem](#) ([Device](#) &device, const [VkRenderPass](#) renderPass, const [VkDescriptorSetLayout](#) globalSetLayout)
- [PointLightRenderSystem](#) (const [PointLightRenderSystem](#) &)=delete
- [PointLightRenderSystem](#) & operator= (const [PointLightRenderSystem](#) &)=delete
- void [render](#) (const [FrameInfo](#) &frameInfo) const override

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- virtual [~ARenderSystemBase](#) ()

Static Public Member Functions

- static void [update](#) (const [FrameInfo](#) &frameInfo, [GlobalUbo](#) &ubo)

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

7.29.1 Detailed Description

Class for point light system.

Definition at line 25 of file [PointLightRenderSystem.hpp](#).

7.29.2 Constructor & Destructor Documentation

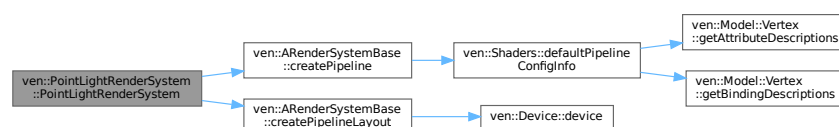
7.29.2.1 PointLightRenderSystem() [1/2]

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 29 of file [PointLightRenderSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.29.2.2 PointLightRenderSystem() [2/2]

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    const PointLightRenderSystem & ) [delete]
```

7.29.3 Member Function Documentation

7.29.3.1 operator=()

```
PointLightRenderSystem & ven::PointLightRenderSystem::operator= (
    const PointLightRenderSystem & ) [delete]
```

7.29.3.2 render()

```
void ven::PointLightRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

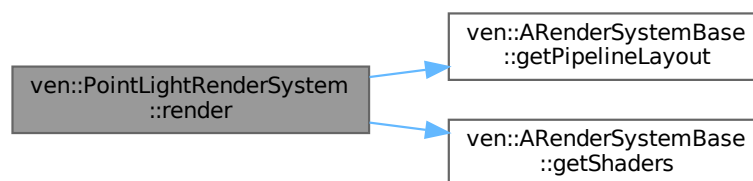
Implements [ven::ARenderSystemBase](#).

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

References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::FrameInfo::lights](#), and [ven::LightPushConstantData::position](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.29.3.3 update()

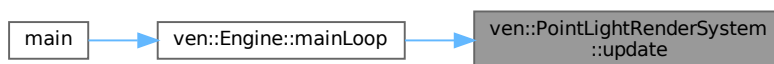
```
void ven::PointLightRenderSystem::update (
    const FrameInfo & frameInfo,
    GlobalUbo & ubo) [static]
```

Definition at line 21 of file [pointLightRenderSystem.cpp](#).

References [ven::FrameInfo::frameTime](#), [ven::FrameInfo::lights](#), [ven::MAX_LIGHTS](#), [ven::GlobalUbo::numLights](#), and [ven::GlobalUbo::pointLights](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



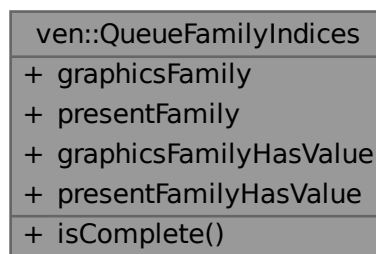
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp](#)

7.30 ven::QueueFamilyIndices Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for [ven::QueueFamilyIndices](#):



Public Member Functions

- bool [isComplete](#) () const

Public Attributes

- uint32_t [graphicsFamily](#) {}
- uint32_t [presentFamily](#) {}
- bool [graphicsFamilyHasValue](#) = false
- bool [presentFamilyHasValue](#) = false

7.30.1 Detailed Description

Definition at line 21 of file [Device.hpp](#).

7.30.2 Member Function Documentation

7.30.2.1 isComplete()

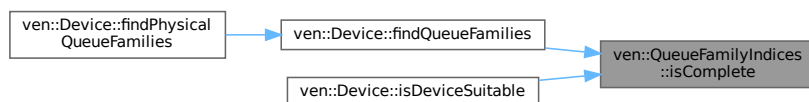
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

Definition at line 26 of file [Device.hpp](#).

References [graphicsFamilyHasValue](#), and [presentFamilyHasValue](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [ven::Device::isDeviceSuitable\(\)](#).

Here is the caller graph for this function:



7.30.3 Member Data Documentation

7.30.3.1 graphicsFamily

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

Definition at line 22 of file [Device.hpp](#).

Referenced by [ven::Device::createCommandPool\(\)](#), and [ven::Device::findQueueFamilies\(\)](#).

7.30.3.2 graphicsFamilyHasValue

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 24 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

7.30.3.3 presentFamily

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

Definition at line 23 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.30.3.4 presentFamilyHasValue

```
bool ven::QueueFamilyIndices::presentFamilyHasValue = false
```

Definition at line 25 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

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

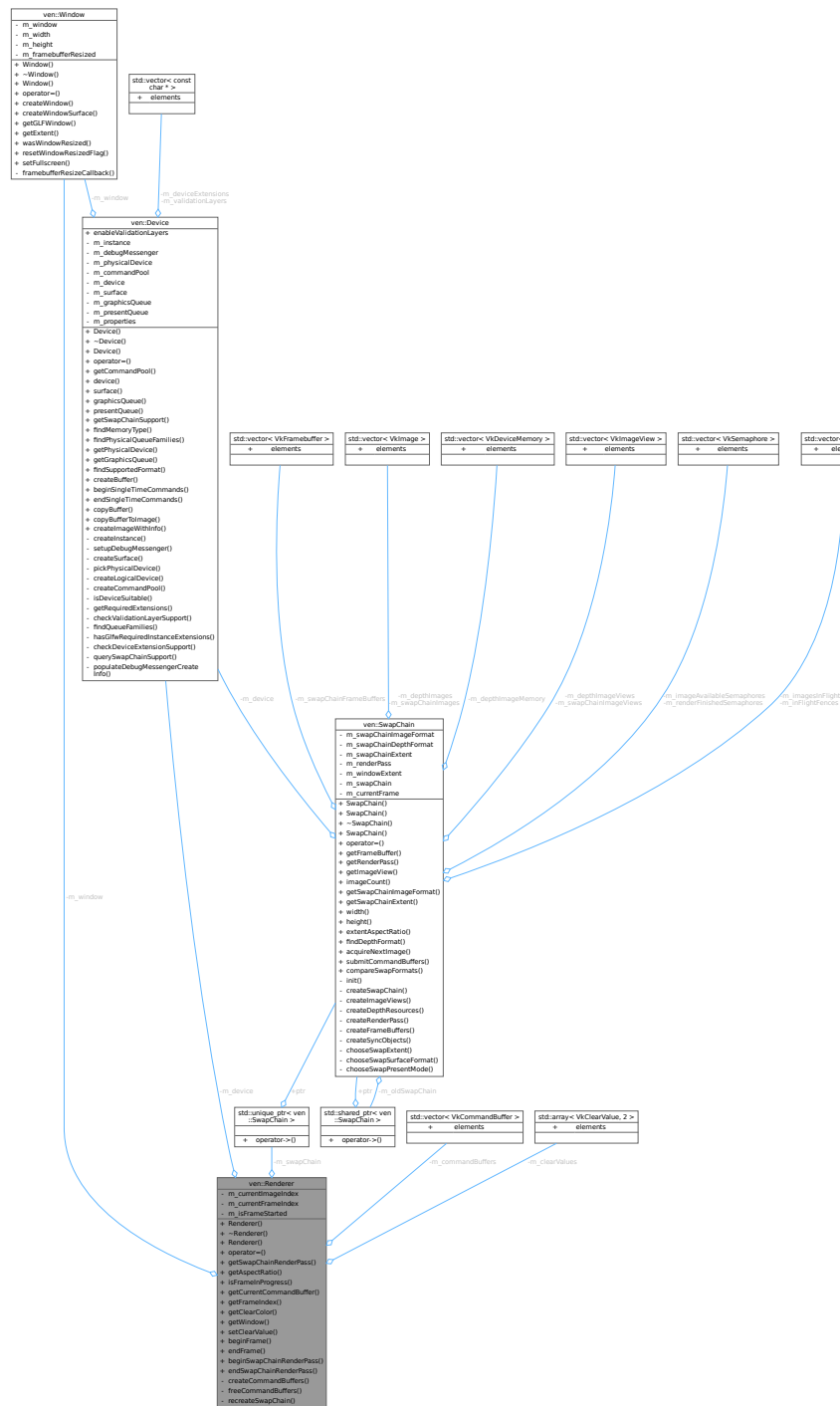
- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

7.31 ven::Renderer Class Reference

Class for renderer.

```
#include <Renderer.hpp>
```

Collaboration diagram for ven::Renderer:



Public Member Functions

- [Renderer](#) ([Window](#) &window, [Device](#) &device)
- [~Renderer](#) ()
- [Renderer](#) (const [Renderer](#) &)=delete
- [Renderer](#) & [operator=](#) (const [Renderer](#) &)=delete
- [VkRenderPass](#) [getSwapChainRenderPass](#) () const

- float [getAspectRatio](#) () const
- bool [isFrameInProgress](#) () const
- VkCommandBuffer [getCurrentCommandBuffer](#) () const
- unsigned long [getFrameIndex](#) () const
- std::array< float, 4 > [getClearColor](#) () const
- [Window](#) & [getWindow](#) () const
- void [setClearValue](#) (const VkClearColorValue clearColorValue=[DEFAULT_CLEAR_COLOR](#), const VkClearDepthStencilValue clearDepthValue=[DEFAULT_CLEAR_DEPTH](#))
- VkCommandBuffer [beginFrame](#) ()
- void [endFrame](#) ()
- void [beginSwapChainRenderPass](#) (VkCommandBuffer commandBuffer) const
- void [endSwapChainRenderPass](#) (VkCommandBuffer commandBuffer) const

Private Member Functions

- void [createCommandBuffers](#) ()
- void [freeCommandBuffers](#) ()
- void [recreateSwapChain](#) ()

Private Attributes

- [Window](#) & [m_window](#)
- [Device](#) & [m_device](#)
- std::unique_ptr< [SwapChain](#) > [m_swapChain](#)
- std::vector< VkCommandBuffer > [m_commandBuffers](#)
- std::array< VkClearColorValue, 2 > [m_clearValues](#) {[DEFAULT_CLEAR_COLOR](#), 1.0F, 0.F}
- uint32_t [m_currentImageIndex](#) {0}
- unsigned long [m_currentFrameIndex](#) {0}
- bool [m_isFrameStarted](#) {false}

7.31.1 Detailed Description

Class for renderer.

Definition at line 28 of file [Renderer.hpp](#).

7.31.2 Constructor & Destructor Documentation

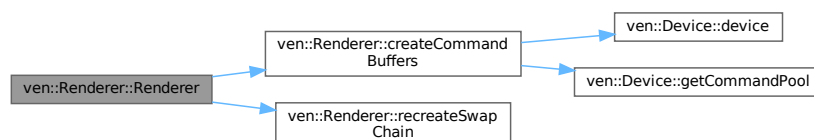
7.31.2.1 [Renderer\(\)](#) [1/2]

```
ven::Renderer::Renderer (
    Window & window,
    Device & device) [inline]
```

Definition at line 32 of file [Renderer.hpp](#).

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



7.31.2.2 ~Renderer()

```
ven::Renderer::~~Renderer () [inline]
```

Definition at line 33 of file [Renderer.hpp](#).

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

Here is the call graph for this function:



7.31.2.3 Renderer() [2/2]

```
ven::Renderer::Renderer (  
    const Renderer & ) [delete]
```

7.31.3 Member Function Documentation

7.31.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

Definition at line 43 of file [renderer.cpp](#).

7.31.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (  
    VkCommandBuffer commandBuffer) const
```

Definition at line 89 of file [renderer.cpp](#).

7.31.3.3 createCommandBuffers()

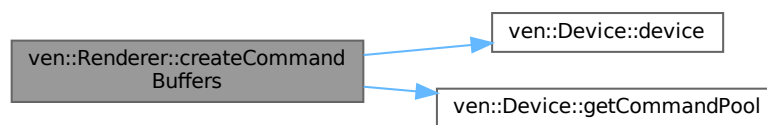
```
void ven::Renderer::createCommandBuffers () [private]
```

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

References [ven::Device::device\(\)](#), [ven::Device::getCommandPool\(\)](#), [m_commandBuffers](#), [m_device](#), and [ven::MAX_FRAMES_IN_FLIGHT](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.31.3.4 endFrame()

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.31.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

Definition at line 119 of file [renderer.cpp](#).

7.31.3.6 freeCommandBuffers()

```
void ven::Renderer::freeCommandBuffers () [private]
```

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

Referenced by [~Renderer\(\)](#).

Here is the caller graph for this function:



7.31.3.7 getAspectRatio()

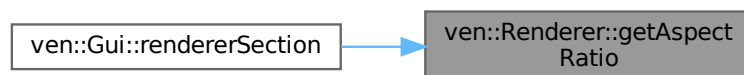
```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

Definition at line 39 of file [Renderer.hpp](#).

References [m_swapChain](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.3.8 getClearColor()

```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

Definition at line 44 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.3.9 `getCurrentCommandBuffer()`

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

Definition at line 41 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), [m_commandBuffers](#), and [m_currentFrameIndex](#).

Referenced by [ven::Gui::render\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



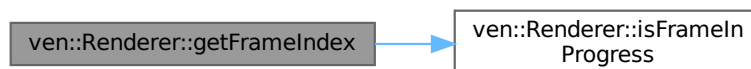
7.31.3.10 `getFrameIndex()`

```
unsigned long ven::Renderer::getFrameIndex () const [inline], [nodiscard]
```

Definition at line 43 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), and [m_currentFrameIndex](#).

Here is the call graph for this function:



7.31.3.11 getSwapChainRenderPass()

```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

Definition at line 38 of file [Renderer.hpp](#).

References [m_swapChain](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:

**7.31.3.12 getWindow()**

```
Window & ven::Renderer::getWindow () const [inline], [nodiscard]
```

Definition at line 51 of file [Renderer.hpp](#).

References [m_window](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:

**7.31.3.13 isFrameInProgress()**

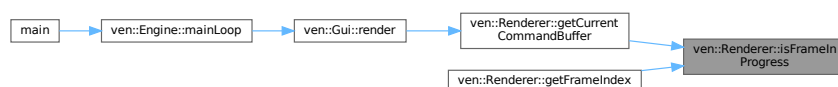
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

Definition at line 40 of file [Renderer.hpp](#).

References [m_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



7.31.3.14 operator=()

```
Renderer & ven::Renderer::operator= (
    const Renderer & ) [delete]
```

7.31.3.15 recreateSwapChain()

```
void ven::Renderer::recreateSwapChain () [private]
```

Definition at line 23 of file [renderer.cpp](#).

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

Here is the caller graph for this function:



7.31.3.16 setClearValue()

```
void ven::Renderer::setClearValue (
    const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

Definition at line 53 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.4 Member Data Documentation

7.31.4.1 m_clearValues

```
std::array<VkClearColor, 2> ven::Renderer::m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F}  
[private]
```

Definition at line 69 of file [Renderer.hpp](#).

Referenced by [getClearColor\(\)](#), and [setClearColor\(\)](#).

7.31.4.2 m_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

Definition at line 68 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

7.31.4.3 m_currentFrameIndex

```
unsigned long ven::Renderer::m_currentFrameIndex {0} [private]
```

Definition at line 72 of file [Renderer.hpp](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

7.31.4.4 m_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

Definition at line 71 of file [Renderer.hpp](#).

7.31.4.5 m_device

```
Device& ven::Renderer::m_device [private]
```

Definition at line 66 of file [Renderer.hpp](#).

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

7.31.4.6 m_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

Definition at line 73 of file [Renderer.hpp](#).

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

7.31.4.7 m_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]
```

Definition at line 67 of file [Renderer.hpp](#).

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

7.31.4.8 m_window

```
Window& ven::Renderer::m_window [private]
```

Definition at line 65 of file [Renderer.hpp](#).

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

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

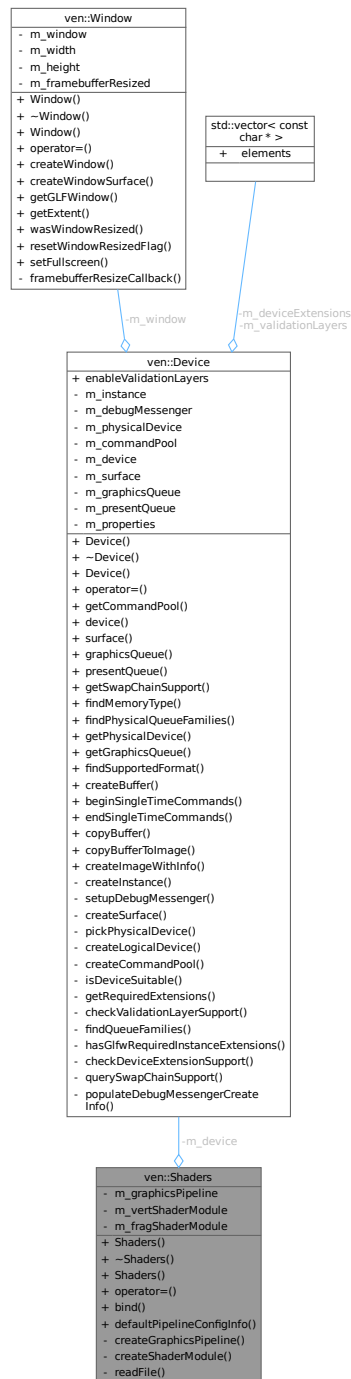
- [/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/renderer.cpp](#)

7.32 ven::Shaders Class Reference

Class for shaders.

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::Shaders:



Public Member Functions

- [Shaders](#) ([Device](#) &device, const std::string &vertFilepath, const std::string &fragFilepath, const [PipelineConfigInfo](#) &configInfo)
- [~Shaders](#) ()
- [Shaders](#) (const [Shaders](#) &)=delete
- [Shaders](#) &operator= (const [Shaders](#) &)=delete
- void [bind](#) (const VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void [defaultPipelineConfigInfo](#) ([PipelineConfigInfo](#) &configInfo)

Private Member Functions

- void [createGraphicsPipeline](#) (const std::string &vertFilepath, const std::string &fragFilepath, const [PipelineConfigInfo](#) &configInfo)
- void [createShaderModule](#) (const std::vector< char > &code, VkShaderModule *shaderModule) const

Static Private Member Functions

- static std::vector< char > [readFile](#) (const std::string &filename)

Private Attributes

- [Device](#) & [m_device](#)
- VkPipeline [m_graphicsPipeline](#) {nullptr}
- VkShaderModule [m_vertShaderModule](#) {nullptr}
- VkShaderModule [m_fragShaderModule](#) {nullptr}

7.32.1 Detailed Description

Class for shaders.

Definition at line 44 of file [Shaders.hpp](#).

7.32.2 Constructor & Destructor Documentation

7.32.2.1 Shaders() [1/2]

```
ven::Shaders::Shaders (
    Device & device,
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [inline]
```

Definition at line 48 of file [Shaders.hpp](#).

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

Here is the call graph for this function:



7.32.2.2 ~Shaders()

```
ven::Shaders::~~Shaders ()
```

Definition at line 8 of file [shaders.cpp](#).

References [ven::Device::device\(\)](#), [m_device](#), [m_fragShaderModule](#), [m_graphicsPipeline](#), and [m_vertShaderModule](#).

Here is the call graph for this function:



7.32.2.3 Shaders() [2/2]

```
ven::Shaders::Shaders (
    const Shaders & ) [delete]
```

7.32.3 Member Function Documentation

7.32.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

Definition at line 55 of file [Shaders.hpp](#).

References [m_graphicsPipeline](#).

7.32.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [private]
```

Definition at line 28 of file [shaders.cpp](#).

References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendState](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), [ven::PipelineConfigInfo::pipelineLayout](#), [ven::PipelineConfigInfo::rasterizationInfo](#), [ven::PipelineConfigInfo::renderPass](#), and [ven::PipelineConfigInfo::subpass](#).

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

Here is the caller graph for this function:



7.32.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

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

7.32.3.4 defaultPipelineConfigInfo()

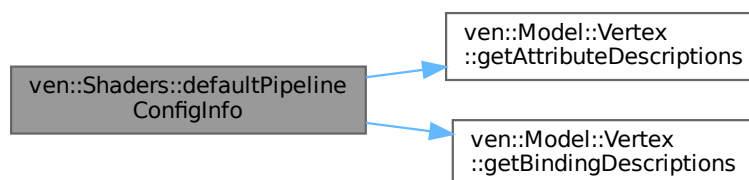
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 109 of file [shaders.cpp](#).

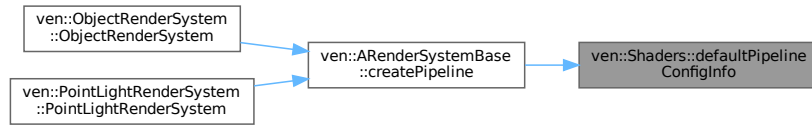
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions\(\)](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::ARenderSystemBase::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.3.5 operator=()

```
Shaders & ven::Shaders::operator= (
    const Shaders & ) [delete]
```

7.32.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

Definition at line 15 of file [shaders.cpp](#).

7.32.4 Member Data Documentation

7.32.4.1 m_device

```
Device& ven::Shaders::m_device [private]
```

Definition at line 63 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.32.4.2 m_fragShaderModule

```
VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]
```

Definition at line 66 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.32.4.3 m_graphicsPipeline

```
VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]
```

Definition at line 64 of file [Shaders.hpp](#).

Referenced by [bind\(\)](#), and [~Shaders\(\)](#).

7.32.4.4 m_vertShaderModule

```
VkShaderModule ven::Shaders::m_vertShaderModule {nullptr} [private]
```

Definition at line 65 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

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

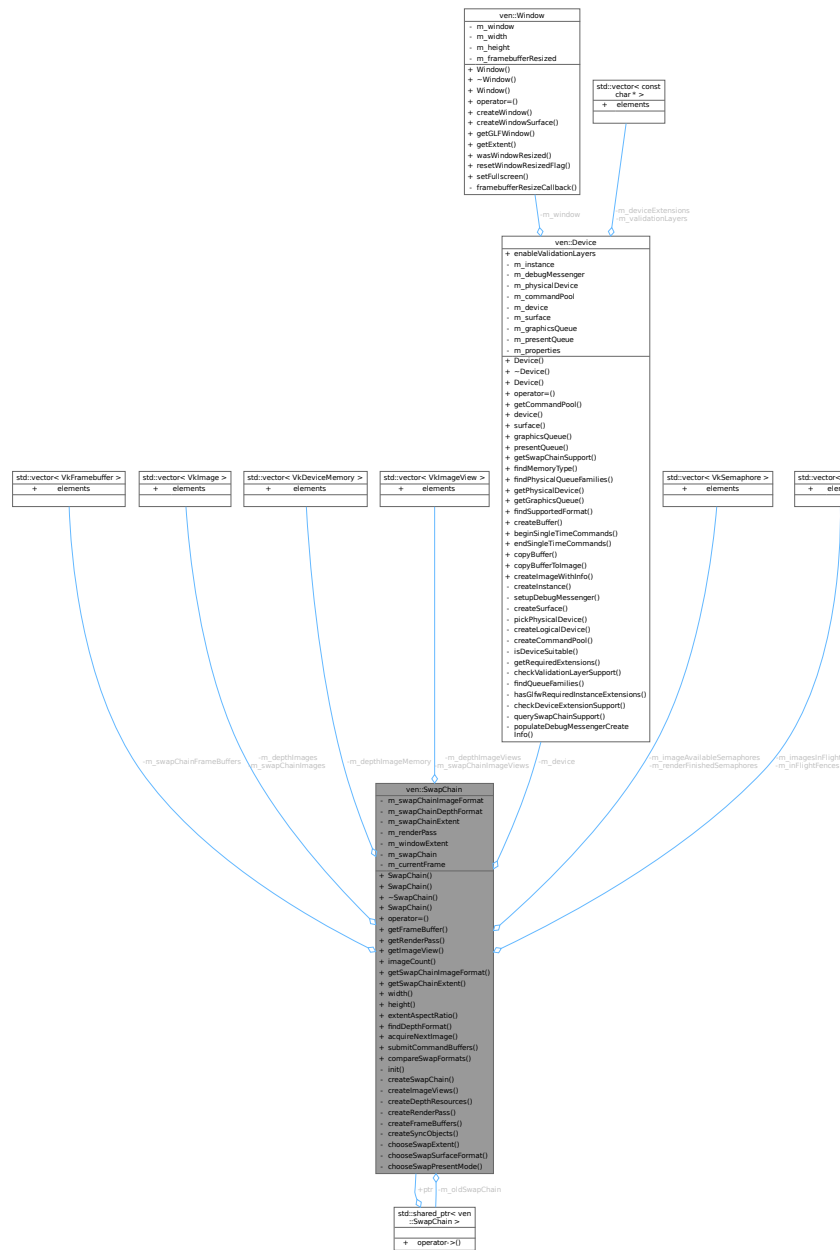
- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/shaders.cpp](#)

7.33 ven::SwapChain Class Reference

Class for swap chain.

```
#include <SwapChain.hpp>
```

Collaboration diagram for ven::SwapChain:



Public Member Functions

- [SwapChain](#) ([Device](#) &deviceRef, const [VkExtent2D](#) windowExtentRef)
- [SwapChain](#) ([Device](#) &deviceRef, const [VkExtent2D](#) windowExtentRef, std::shared_ptr< [SwapChain](#) > previous)
- [~SwapChain](#) ()
- [SwapChain](#) (const [SwapChain](#) &)=delete
- [SwapChain](#) & operator= (const [SwapChain](#) &)=delete
- [VkFramebuffer](#) [getFrameBuffer](#) (const unsigned long index) const
- [VkRenderPass](#) [getRenderPass](#) () const
- [VkImageView](#) [getImageView](#) (const int index) const
- [size_t](#) [imageCount](#) () const

- VkFormat [getSwapChainImageFormat](#) () const
- VkExtent2D [getSwapChainExtent](#) () const
- uint32_t [width](#) () const
- uint32_t [height](#) () const
- float [extentAspectRatio](#) () const
- VkFormat [findDepthFormat](#) () const
- VkResult [acquireNextImage](#) (uint32_t *imageIndex) const
- VkResult [submitCommandBuffers](#) (const VkCommandBuffer *buffers, const uint32_t *imageIndex)
- bool [compareSwapFormats](#) (const [SwapChain](#) &swapChain) const

Private Member Functions

- void [init](#) ()
- void [createSwapChain](#) ()
- void [createImageViews](#) ()
- void [createDepthResources](#) ()
- void [createRenderPass](#) ()
- void [createFrameBuffers](#) ()
- void [createSyncObjects](#) ()
- VkExtent2D [chooseSwapExtent](#) (const VkSurfaceCapabilitiesKHR &capabilities) const

Static Private Member Functions

- static VkSurfaceFormatKHR [chooseSwapSurfaceFormat](#) (const std::vector< VkSurfaceFormatKHR > &availableFormats)
- static VkPresentModeKHR [chooseSwapPresentMode](#) (const std::vector< VkPresentModeKHR > &availablePresentModes)

Private Attributes

- VkFormat [m_swapChainImageFormat](#) {}
- VkFormat [m_swapChainDepthFormat](#) {}
- VkExtent2D [m_swapChainExtent](#) {}
- std::vector< VkFramebuffer > [m_swapChainFrameBuffers](#)
- VkRenderPass [m_renderPass](#) {}
- std::vector< VkImage > [m_depthImages](#)
- std::vector< VkDeviceMemory > [m_depthImageMemory](#)
- std::vector< VkImageView > [m_depthImageViews](#)
- std::vector< VkImage > [m_swapChainImages](#)
- std::vector< VkImageView > [m_swapChainImageViews](#)
- [Device](#) & [m_device](#)
- VkExtent2D [m_windowExtent](#)
- VkSwapchainKHR [m_swapChain](#) {}
- std::shared_ptr< [SwapChain](#) > [m_oldSwapChain](#)
- std::vector< VkSemaphore > [m_imageAvailableSemaphores](#)
- std::vector< VkSemaphore > [m_renderFinishedSemaphores](#)
- std::vector< VkFence > [m_inFlightFences](#)
- std::vector< VkFence > [m_imagesInFlight](#)
- size_t [m_currentFrame](#) {0}

7.33.1 Detailed Description

Class for swap chain.

Definition at line 23 of file [SwapChain.hpp](#).

7.33.2 Constructor & Destructor Documentation

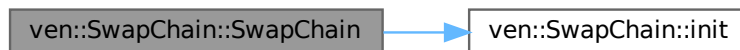
7.33.2.1 SwapChain() [1/3]

```
ven::SwapChain::SwapChain (  
    Device & deviceRef,  
    const VkExtent2D windowExtentRef) [inline]
```

Definition at line 27 of file [SwapChain.hpp](#).

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

Here is the call graph for this function:



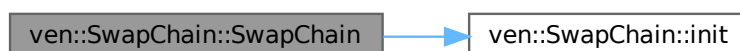
7.33.2.2 SwapChain() [2/3]

```
ven::SwapChain::SwapChain (  
    Device & deviceRef,  
    const VkExtent2D windowExtentRef,  
    std::shared_ptr< SwapChain > previous) [inline]
```

Definition at line 28 of file [SwapChain.hpp](#).

References [init\(\)](#), and [m_oldSwapChain](#).

Here is the call graph for this function:



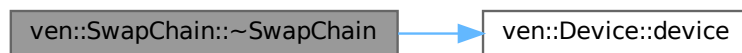
7.33.2.3 ~SwapChain()

```
ven::SwapChain::~~SwapChain ()
```

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

References [ven::Device::device\(\)](#), [m_depthImageMemory](#), [m_depthImages](#), [m_depthImageViews](#), [m_device](#), [m_imageAvailableSemaphores](#), [m_inFlightFences](#), [m_renderFinishedSemaphores](#), [m_renderPass](#), [m_swapChain](#), [m_swapChainFrameBuffers](#), [m_swapChainImageViews](#), and [ven::MAX_FRAMES_IN_FLIGHT](#).

Here is the call graph for this function:



7.33.2.4 SwapChain() [3/3]

```
ven::SwapChain::SwapChain (
    const SwapChain & ) [delete]
```

7.33.3 Member Function Documentation

7.33.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

7.33.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

7.33.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

7.33.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (  
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

Definition at line 331 of file [swapChain.cpp](#).

7.33.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (  
    const SwapChain & swapChain) const [inline], [nodiscard]
```

Definition at line 49 of file [SwapChain.hpp](#).

References [m_swapChainDepthFormat](#), and [m_swapChainImageFormat](#).

7.33.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

7.33.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

7.33.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

7.33.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

7.33.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

Definition at line 103 of file [swapChain.cpp](#).

7.33.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.33.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

Definition at line 43 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.33.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

7.33.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (  
    const unsigned long index) const [inline], [nodiscard]
```

Definition at line 34 of file [SwapChain.hpp](#).

References [m_swapChainFrameBuffers](#).

7.33.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (  
    const int index) const [inline], [nodiscard]
```

Definition at line 36 of file [SwapChain.hpp](#).

References [m_swapChainImageViews](#).

7.33.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

Definition at line 35 of file [SwapChain.hpp](#).

References [m_renderPass](#).

7.33.3.17 getSwapChainExtent()

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

Definition at line 39 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.33.3.18 getSwapChainImageFormat()

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

Definition at line 38 of file [SwapChain.hpp](#).

References [m_swapChainImageFormat](#).

7.33.3.19 height()

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

Definition at line 41 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.33.3.20 imageCount()

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

Definition at line 37 of file [SwapChain.hpp](#).

References [m_swapChainImages](#).

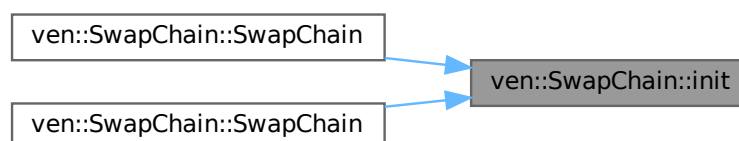
7.33.3.21 init()

```
void ven::SwapChain::init () [private]
```

Definition at line 39 of file [swapChain.cpp](#).

Referenced by [SwapChain\(\)](#), and [SwapChain\(\)](#).

Here is the caller graph for this function:



7.33.3.22 operator=()

```
SwapChain & ven::SwapChain::operator= (
    const SwapChain & ) [delete]
```

7.33.3.23 submitCommandBuffers()

```
VkResult ven::SwapChain::submitCommandBuffers (
    const VkCommandBuffer * buffers,
    const uint32_t * imageIndex)
```

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

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.33.3.24 width()

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

Definition at line 40 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.33.4 Member Data Documentation

7.33.4.1 m_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0} [private]
```

Definition at line 88 of file [SwapChain.hpp](#).

7.33.4.2 m_depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::m_depthImageMemory [private]
```

Definition at line 73 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.3 m_depthImages

```
std::vector<VkImage> ven::SwapChain::m_depthImages [private]
```

Definition at line 72 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.4 m_depthImageViews

```
std::vector<VkImageView> ven::SwapChain::m_depthImageViews [private]
```

Definition at line 74 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.5 m_device

```
Device& ven::SwapChain::m_device [private]
```

Definition at line 78 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.6 m_imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_imageAvailableSemaphores [private]
```

Definition at line 84 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.7 m_imagesInFlight

```
std::vector<VkFence> ven::SwapChain::m_imagesInFlight [private]
```

Definition at line 87 of file [SwapChain.hpp](#).

7.33.4.8 m_inFlightFences

```
std::vector<VkFence> ven::SwapChain::m_inFlightFences [private]
```

Definition at line 86 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.9 m_oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::m_oldSwapChain [private]
```

Definition at line 82 of file [SwapChain.hpp](#).

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

7.33.4.10 m_renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_renderFinishedSemaphores [private]
```

Definition at line 85 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.11 m_renderPass

```
VkRenderPass ven::SwapChain::m_renderPass {} [private]
```

Definition at line 70 of file [SwapChain.hpp](#).

Referenced by [getRenderPass\(\)](#), and [~SwapChain\(\)](#).

7.33.4.12 m_swapChain

```
VkSwapchainKHR ven::SwapChain::m_swapChain {} [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.33.4.13 m_swapChainDepthFormat

```
VkFormat ven::SwapChain::m_swapChainDepthFormat {} [private]
```

Definition at line 66 of file [SwapChain.hpp](#).

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

7.33.4.14 m_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

Definition at line 67 of file [SwapChain.hpp](#).

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

7.33.4.15 m_swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::m_swapChainFrameBuffers [private]
```

Definition at line 69 of file [SwapChain.hpp](#).

Referenced by [getFramebuffer\(\)](#), and [~SwapChain\(\)](#).

7.33.4.16 m_swapChainImageFormat

```
VkFormat ven::SwapChain::m_swapChainImageFormat {} [private]
```

Definition at line 65 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

7.33.4.17 m_swapChainImages

```
std::vector<VkImage> ven::SwapChain::m_swapChainImages [private]
```

Definition at line 75 of file [SwapChain.hpp](#).

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

7.33.4.18 m_swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::m_swapChainImageViews [private]
```

Definition at line 76 of file [SwapChain.hpp](#).

Referenced by [getImageView\(\)](#), and [~SwapChain\(\)](#).

7.33.4.19 m_windowExtent

```
VkExtent2D ven::SwapChain::m_windowExtent [private]
```

Definition at line 79 of file [SwapChain.hpp](#).

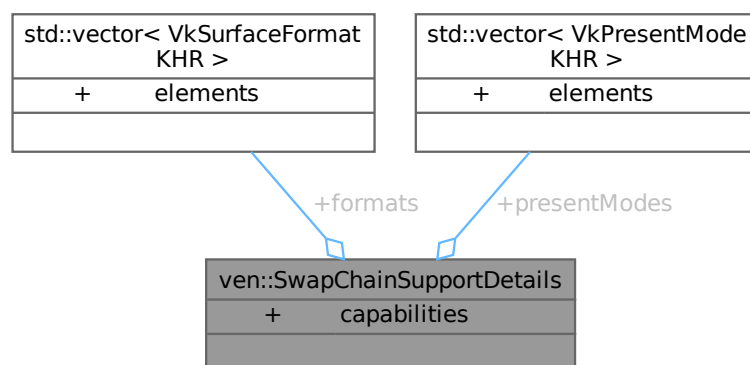
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/swapChain.cpp](#)

7.34 ven::SwapChainSupportDetails Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::SwapChainSupportDetails:



Public Attributes

- VkSurfaceCapabilitiesKHR [capabilities](#)
- std::vector< VkSurfaceFormatKHR > [formats](#)
- std::vector< VkPresentModeKHR > [presentModes](#)

7.34.1 Detailed Description

Definition at line 15 of file [Device.hpp](#).

7.34.2 Member Data Documentation

7.34.2.1 capabilities

```
VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities
```

Definition at line 16 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.34.2.2 formats

```
std::vector<VkSurfaceFormatKHR> ven::SwapChainSupportDetails::formats
```

Definition at line 17 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.34.2.3 presentModes

```
std::vector<VkPresentModeKHR> ven::SwapChainSupportDetails::presentModes
```

Definition at line 18 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

7.35 ven::Transform3DComponent Class Reference

Class for 3D transformation.

```
#include <Transform3DComponent.hpp>
```

Collaboration diagram for ven::Transform3DComponent:

ven::Transform3DComponent	
+	translation
+	scale
+	rotation
+	mat4()
+	normalMatrix()

Public Member Functions

- glm::mat4 [mat4](#) () const
- glm::mat3 [normalMatrix](#) () const

Public Attributes

- glm::vec3 [translation](#) {}
- glm::vec3 [scale](#) {1.F, 1.F, 1.F}
- glm::vec3 [rotation](#) {}

7.35.1 Detailed Description

Class for 3D transformation.

Definition at line 18 of file [Transform3DComponent.hpp](#).

7.35.2 Member Function Documentation

7.35.2.1 mat4()

```
glm::mat4 ven::Transform3DComponent::mat4 () const [nodiscard]
```

Definition at line 3 of file [transform3DComponent.cpp](#).

References [rotation](#), [scale](#), and [translation](#).

7.35.2.2 normalMatrix()

```
glm::mat3 ven::Transform3DComponent::normalMatrix () const [nodiscard]
```

Definition at line 38 of file [transform3DComponent.cpp](#).

7.35.3 Member Data Documentation

7.35.3.1 rotation

```
glm::vec3 ven::Transform3DComponent::rotation {}
```

Definition at line 24 of file [Transform3DComponent.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [mat4\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

7.35.3.2 scale

```
glm::vec3 ven::Transform3DComponent::scale {1.F, 1.F, 1.F}
```

Definition at line 23 of file [Transform3DComponent.hpp](#).

Referenced by [ven::Light::createLight\(\)](#), [ven::Engine::loadObjects\(\)](#), and [mat4\(\)](#).

7.35.3.3 translation

```
glm::vec3 ven::Transform3DComponent::translation {}
```

Definition at line 22 of file [Transform3DComponent.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::Engine::loadObjects\(\)](#), [mat4\(\)](#), and [ven::EventManager::moveCamera\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp](#)

7.36 ven::Model::Vertex Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for [ven::Model::Vertex](#):

ven::Model::Vertex
+ position
+ color
+ normal
+ uv
+ operator==()
+ getBindingDescriptions()
+ getAttributeDescriptions()

Public Member Functions

- bool `operator==` (const `Vertex` &other) const

Static Public Member Functions

- static std::vector< `VkVertexInputBindingDescription` > `getBindingDescriptions` ()
- static std::vector< `VkVertexInputAttributeDescription` > `getAttributeDescriptions` ()

Public Attributes

- glm::vec3 `position` {}
- glm::vec3 `color` {}
- glm::vec3 `normal` {}
- glm::vec2 `uv` {}

7.36.1 Detailed Description

Definition at line 29 of file `Model.hpp`.

7.36.2 Member Function Documentation

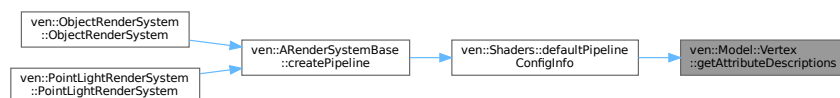
7.36.2.1 `getAttributeDescriptions()`

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions
() [static]
```

Definition at line 105 of file `model.cpp`.

Referenced by `ven::Shaders::defaultPipelineConfigInfo()`.

Here is the caller graph for this function:



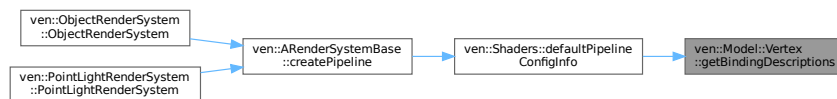
7.36.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()
[static]
```

Definition at line 96 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



7.36.2.3 operator==()

```
bool ven::Model::Vertex::operator== (
    const Vertex & other) const [inline]
```

Definition at line 38 of file [Model.hpp](#).

References [color](#), [normal](#), [position](#), and [uv](#).

7.36.3 Member Data Documentation

7.36.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 31 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.36.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 32 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.36.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 30 of file [Model.hpp](#).

Referenced by [operator==\(\)](#), and [ven::Model::Builder::processMesh\(\)](#).

7.36.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 33 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.37 ven::Window Class Reference

Class for window.

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

Collaboration diagram for ven::Window:

ven::Window
<ul style="list-style-type: none"> - m_window - m_width - m_height - m_framebufferResized
<ul style="list-style-type: none"> + Window() + ~Window() + Window() + operator=() + createWindow() + createWindowSurface() + getGLFWWindow() + getExtent() + wasWindowResized() + resetWindowResizedFlag() + setFullscreen() - framebufferResizeCallback()

Public Member Functions

- [Window](#) (const uint32_t width=[DEFAULT_WIDTH](#), const uint32_t height=[DEFAULT_HEIGHT](#), const std::string &title=[DEFAULT_TITLE](#).data())
- [~Window](#) ()
- [Window](#) (const [Window](#) &)=delete
- [Window](#) & [operator=](#) (const [Window](#) &)=delete
- GLFWwindow * [createWindow](#) (uint32_t width, uint32_t height, const std::string &title)
- void [createWindowSurface](#) (VkInstance instance, VkSurfaceKHR *surface) const
- GLFWwindow * [getGLFWWindow](#) () const
- VkExtent2D [getExtent](#) () const
- bool [wasWindowResized](#) () const
- void [resetWindowResizedFlag](#) ()
- void [setFullscreen](#) (bool fullscreen, uint32_t width, uint32_t height)

Static Private Member Functions

- static void [framebufferResizeCallback](#) (GLFWwindow *window, int width, int height)

Private Attributes

- GLFWwindow * [m_window](#) {nullptr}
- uint32_t [m_width](#) {[DEFAULT_WIDTH](#)}
- uint32_t [m_height](#) {[DEFAULT_HEIGHT](#)}
- bool [m_framebufferResized](#) = false

7.37.1 Detailed Description

Class for window.

Definition at line 26 of file [Window.hpp](#).

7.37.2 Constructor & Destructor Documentation

7.37.2.1 Window() [1/2]

```
ven::Window::Window (
    const uint32_t width = DEFAULT\_WIDTH,
    const uint32_t height = DEFAULT\_HEIGHT,
    const std::string & title = DEFAULT\_TITLE.data()) [inline], [explicit]
```

Definition at line 30 of file [Window.hpp](#).

7.37.2.2 ~Window()

```
ven::Window::~~Window () [inline]
```

Definition at line 31 of file [Window.hpp](#).

References [m_window](#).

7.37.2.3 Window() [2/2]

```
ven::Window::Window (  
    const Window & ) [delete]
```

7.37.3 Member Function Documentation

7.37.3.1 createWindow()

```
GLFWwindow * ven::Window::createWindow (  
    uint32_t width,  
    uint32_t height,  
    const std::string & title) [nodiscard]
```

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

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

Here is the call graph for this function:



7.37.3.2 createWindowSurface()

```
void ven::Window::createWindowSurface (  
    VkInstance instance,  
    VkSurfaceKHR * surface) const
```

Definition at line 24 of file [window.cpp](#).

Referenced by [ven::Device::createSurface\(\)](#).

Here is the caller graph for this function:



7.37.3.3 framebufferResizeCallback()

```
void ven::Window::framebufferResizeCallback (  
    GLFWwindow * window,  
    int width,  
    int height) [static], [private]
```

Definition at line 31 of file [window.cpp](#).

References [m_framebufferResized](#).

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

Here is the caller graph for this function:



7.37.3.4 getExtent()

```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

Definition at line 41 of file [Window.hpp](#).

References [m_height](#), and [m_width](#).

Referenced by [ven::Gui::renderSection\(\)](#).

Here is the caller graph for this function:



7.37.3.5 getGLFWWindow()

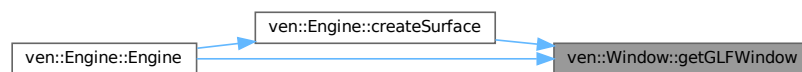
`GLFWwindow * ven::Window::getGLFWWindow () const [inline], [nodiscard]`

Definition at line 39 of file [Window.hpp](#).

References [m_window](#).

Referenced by [ven::Engine::createSurface\(\)](#), and [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.37.3.6 operator=()

```
Window & ven::Window::operator= (
    const Window & ) [delete]
```

7.37.3.7 resetWindowResizedFlag()

`void ven::Window::resetWindowResizedFlag () [inline]`

Definition at line 43 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.37.3.8 setFullscreen()

```
void ven::Window::setFullscreen (
    bool fullscreen,
    uint32_t width,
    uint32_t height)
```

Definition at line 39 of file [window.cpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.37.3.9 wasWindowResized()

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

Definition at line 42 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.37.4 Member Data Documentation

7.37.4.1 m_framebufferResized

```
bool ven::Window::m_framebufferResized = false [private]
```

Definition at line 55 of file [Window.hpp](#).

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).

7.37.4.2 m_height

```
uint32_t ven::Window::m_height {DEFAULT_HEIGHT} [private]
```

Definition at line 53 of file [Window.hpp](#).

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

7.37.4.3 m_width

```
uint32_t ven::Window::m_width {DEFAULT_WIDTH} [private]
```

Definition at line 52 of file [Window.hpp](#).

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

7.37.4.4 m_window

```
GLFWwindow* ven::Window::m_window {nullptr} [private]
```

Definition at line 51 of file [Window.hpp](#).

Referenced by [getGLFWWindow\(\)](#), and [~Window\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/window.cpp](#)

Chapter 8

File Documentation

8.1 `/home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag` File Reference ↩

8.2 `fragment_point_light.frag`

[Go to the documentation of this file.](#)

```
00001 #version 450
00002
00003 layout(location = 0) in vec2 fragOffset;
00004 layout(location = 0) out vec4 outColor;
00005
00006 struct PointLight {
00007     vec4 position; // ignore w
00008     vec4 color; // w is intensity
00009 };
00010
00011 layout(set = 0, binding = 0) uniform GlobalUbo {
00012     mat4 projection;
00013     mat4 view;
00014     mat4 invView;
00015     vec4 ambientLightColor; // w is intensity
00016     PointLight pointLights[10];
00017     int numLights;
00018 } ubo;
00019
00020 layout(push_constant) uniform Push {
00021     vec4 position;
00022     vec4 color;
00023     float radius;
00024 } push;
00025
00026 const float M_PI = 3.1415926538;
00027
00028 void main() {
00029     float dis = length(fragOffset);
00030     if (dis >= 1.0) {
00031         discard;
00032     }
00033
00034     float cosDis = 0.5 * (cos(dis * M_PI) + 1.0);
00035     outColor = vec4(push.color.rgb + 0.5 * cosDis, cosDis);
00036 }
```

8.3 `/home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag` File Reference ↩

8.4 `fragment_shader.frag`

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 fragColor;
00004 layout(location = 1) in vec3 fragPosWorld;
00005 layout(location = 2) in vec3 fragNormalWorld;
00006
00007 layout(location = 0) out vec4 outColor;
00008
00009 struct PointLight {
00010     vec4 position; // ignore w
00011     vec4 color; // w is intensity
00012 };
00013
00014 layout(set = 0, binding = 0) uniform GlobalUbo {
00015     mat4 projection;
00016     mat4 view;
00017     mat4 invView;
00018     vec4 ambientLightColor; // w is intensity
00019     PointLight pointLights[10];
00020     int numLights;
00021 } ubo;
00022
00023 layout(push_constant) uniform Push {
00024     mat4 modelMatrix;
00025     mat4 normalMatrix;
00026 } push;
00027
00028 void main() {
00029     vec3 specularLight = vec3(0.0);
00030     vec3 surfaceNormal = normalize(gl_FrontFacing ? fragNormalWorld : -fragNormalWorld);
00031     vec3 diffuseLight = ubo.ambientLightColor.rgb * ubo.ambientLightColor.a;
00032
00033     vec3 cameraPosWorld = ubo.invView[3].xyz;
00034     vec3 viewDirection = normalize(cameraPosWorld - fragPosWorld);
00035
00036     for (int i = 0; i < ubo.numLights; i++) {
00037         PointLight light = ubo.pointLights[i];
00038         vec3 directionToLight = light.position.xyz - fragPosWorld;
00039         float distanceSquared = dot(directionToLight, directionToLight);
00040         float attenuation = 1.0 / distanceSquared; // distance squared
00041         directionToLight = normalize(directionToLight);
00042
00043         float cosAngIncidence = max(dot(surfaceNormal, directionToLight), 0);
00044         vec3 intensity = light.color.rgb * light.color.a * attenuation;
00045         vec3 reflectionDirection = reflect(-directionToLight, surfaceNormal);
00046         float cosAngReflection = max(dot(viewDirection, reflectionDirection), 0);
00047
00048         // diffuse lighting
00049         diffuseLight += intensity * cosAngIncidence;
00050         // specular lighting
00051         float specular = pow(cosAngReflection, 512);
00052         specularLight += intensity * specular * step(0.0, cosAngIncidence) * step(0.0, cosAngReflection);
00053     }
00054
00055     outColor = vec4(fragColor * (diffuseLight + specularLight), 1.0);
00056 }

```

8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_↵ light.vert File Reference

8.6 vertex_point_light.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 const vec2 OFFSETS[6] = vec2[](
00004     vec2(-1.0, -1.0),
00005     vec2(-1.0, 1.0),
00006     vec2(1.0, -1.0),
00007     vec2(1.0, 1.0),
00008     vec2(-1.0, 1.0),
00009     vec2(1.0, 1.0)
00010 );
00011
00012 layout(location = 0) out vec2 fragOffset;
00013
00014 struct PointLight {

```

```

00015     vec4 position; // ignore w
00016     vec4 color; // w is intensity
00017 };
00018
00019 layout(set = 0, binding = 0) uniform GlobalUbo {
00020     mat4 projection;
00021     mat4 view;
00022     mat4 invView;
00023     vec4 ambientLightColor; // w is intensity
00024     PointLight pointLights[10];
00025     int numLights;
00026 } ubo;
00027
00028 layout(push_constant) uniform Push {
00029     vec4 position;
00030     vec4 color;
00031     float radius;
00032 } push;
00033
00034 void main() {
00035     fragOffset = OFFSETS[gl_VertexIndex];
00036     vec3 cameraRightWorld = vec3(ubo.view[0][0], ubo.view[1][0], ubo.view[2][0]);
00037     vec3 cameraUpWorld = vec3(ubo.view[0][1], ubo.view[1][1], ubo.view[2][1]);
00038
00039     vec3 positionWorld = push.position.xyz
00040     + push.radius * fragOffset.x * cameraRightWorld
00041     + push.radius * fragOffset.y * cameraUpWorld;
00042
00043     gl_Position = ubo.projection * ubo.view * vec4(positionWorld, 1.0);
00044 }

```

8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference

8.8 vertex_shader.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 position;
00004 layout(location = 1) in vec3 color;
00005 layout(location = 2) in vec3 normal;
00006 layout(location = 3) in vec2 uv;
00007
00008 layout(location = 0) out vec3 fragColor;
00009 layout(location = 1) out vec3 fragPosWorld;
00010 layout(location = 2) out vec3 fragNormalWorld;
00011
00012 struct PointLight {
00013     vec4 position; // ignore w
00014     vec4 color; // w is intensity
00015 };
00016
00017 layout(set = 0, binding = 0) uniform GlobalUbo {
00018     mat4 projection;
00019     mat4 view;
00020     mat4 invView;
00021     vec4 ambientLightColor; // w is intensity
00022     PointLight pointLights[10];
00023     int numLights;
00024 } ubo;
00025
00026 layout(push_constant) uniform Push {
00027     mat4 modelMatrix;
00028     mat4 normalMatrix;
00029 } push;
00030
00031 void main() {
00032     vec4 positionWorld = push.modelMatrix * vec4(position, 1.0);
00033     gl_Position = ubo.projection * ubo.view * positionWorld;
00034     fragNormalWorld = normalize(mat3(push.normalMatrix) * normal);
00035     fragPosWorld = positionWorld.xyz;
00036     fragColor = color;
00037 }

```


Namespaces

- namespace [ven](#)

8.9.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

8.10 Buffer.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Buffer.hpp
00003 /// @brief This file contains the Buffer class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Device.hpp"
00010
00011 namespace ven {
00012
00013     ///
00014     /// @class Buffer
00015     /// @brief Class for buffer
00016     /// @namespace ven
00017     ///
00018     class Buffer {
00019     public:
00020
00021         Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
00022             VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
00023             minOffsetAlignment = 1);
00024         ~Buffer();
00025
00026         Buffer(const Buffer&) = delete;
00027         Buffer& operator=(const Buffer&) = delete;
00028
00029         ///
00030         /// @brief Map a memory range of this buffer. If successful, mapped points to the
00031         /// specified buffer range.
00032         /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the
00033         /// complete buffer range.
00034         /// @param offset (Optional) Byte offset from beginning
00035         /// @return VkResult of the buffer mapping call
00036         ///
00037         VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00038
00039         ///
00040         /// @brief Unmap a mapped memory range
00041         ///
00042         /// @note Does not return a result as vkUnmapMemory can't fail
00043         void unmap();
00044
00045         ///
00046         /// @brief Copies the specified data to the mapped buffer. Default value writes whole
00047         /// buffer range
00048         /// @param data Pointer to the data to copy
00049         /// @param size (Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the
00050         /// complete buffer range.
00051         /// @param offset (Optional) Byte offset from beginning of mapped region
00052         void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
00053             offset = 0) const;
00054
00055     };

```

```

00055         /// @brief Flush a memory range of the buffer to make it visible to the device
00056         ///
00057         /// @note Only required for non-coherent memory
00058         ///
00059         /// @param size (Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush
the complete buffer range.
00060         /// @param offset (Optional) Byte offset from beginning
00061         ///
00062         /// @return VkResult of the flush call
00063         ///
00064         [[nodiscard]] VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0)
const;
00065
00066         ///
00067         /// @brief Create a buffer info descriptor
00068         ///
00069         /// @param size (Optional) Size of the memory range of the descriptor
00070         /// @param offset (Optional) Byte offset from beginning
00071         ///
00072         /// @return VkDescriptorBufferInfo of specified offset and range
00073         ///
00074         [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size =
VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset,
size, }; }
00075
00076         ///
00077         /// @brief Invalidate a memory range of the buffer to make it visible to the host
00078         ///
00079         /// @note Only required for non-coherent memory
00080         ///
00081         /// @param size (Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to
invalidate
00082         /// the complete buffer range.
00083         /// @param offset (Optional) Byte offset from beginning
00084         ///
00085         /// @return VkResult of the invalidate call
00086         ///
00087         [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset =
0) const;
00088
00089         ///
00090         /// Copies "instanceSize" bytes of data to the mapped buffer at an offset of index *
alignmentSize
00091         ///
00092         /// @param data Pointer to the data to copy
00093         /// @param index Used in offset calculation
00094         ///
00095         ///
00096         void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
m_instanceSize, index * m_alignmentSize); }
00097
00098         ///
00099         /// Flush the memory range at index * alignmentSize of the buffer to make it visible to
the device
00100         ///
00101         /// @param index Used in offset calculation
00102         ///
00103         [[nodiscard]] VkResult flushIndex(const VkDeviceSize index) const { return
flush(m_alignmentSize, index * m_alignmentSize); }
00104
00105         ///
00106         ///
00107         /// Create a buffer info descriptor
00108         ///
00109         /// @param index Specifies the region given by index * alignmentSize
00110         ///
00111         /// @return VkDescriptorBufferInfo for instance at index
00112         ///
00113         [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index)
const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }
00114
00115         ///
00116         /// Invalidate a memory range of the buffer to make it visible to the host
00117         ///
00118         /// @note Only required for non-coherent memory
00119         ///
00120         /// @param index Specifies the region to invalidate: index * alignmentSize
00121         ///
00122         /// @return VkResult of the invalidate call
00123         ///
00124         [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return
invalidate(m_alignmentSize, index * m_alignmentSize); }
00125
00126         [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
00127         [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
00128         [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
00129         [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }

```

```

00130         [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_alignmentSize; }
00131         [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00132         [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
m_memoryPropertyFlags; }
00133         [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }
00134
00135     private:
00136         ///
00137         /// Returns the minimum instance size required to be compatible with devices
minOffsetAlignment
00138         ///
00139         /// @param instanceSize The size of an instance
00140         /// @param minOffsetAlignment The minimum required alignment, in bytes, for the offset
member (eg
00141         /// minUniformBufferOffsetAlignment)
00142         ///
00143         /// @return VkResult of the buffer mapping call
00144         ///
00145         static VkDeviceSize getAlignment(VkDeviceSize instanceSize, VkDeviceSize
minOffsetAlignment);
00146
00147         Device& m_device;
00148         void* m_mapped = nullptr;
00149         VkBuffer m_buffer = VK_NULL_HANDLE;
00150         VkDeviceMemory m_memory = VK_NULL_HANDLE;
00151
00152         VkDeviceSize m_bufferSize;
00153         VkDeviceSize m_instanceSize;
00154         uint32_t m_instanceCount;
00155         VkDeviceSize m_alignmentSize;
00156         VkBufferUsageFlags m_usageFlags;
00157         VkMemoryPropertyFlags m_memoryPropertyFlags;
00158
00159     }; // class Buffer
00160
00161 } // namespace ven

```

8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

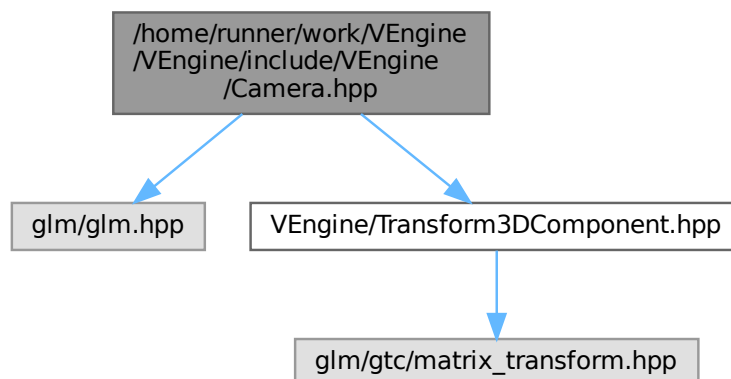
This file contains the Camera class.

```

#include <glm/glm.hpp>
#include "VEngine/Transform3DComponent.hpp"

```

Include dependency graph for Camera.hpp:




```

00013 namespace ven {
00014
00015     static constexpr glm::vec3 DEFAULT_POSITION{0.F, 0.F, -2.5F};
00016     static constexpr glm::vec3 DEFAULT_ROTATION{0.F, 0.F, 0.F};
00017
00018     static constexpr float DEFAULT_FOV = glm::radians(50.0F);
00019     static constexpr float DEFAULT_NEAR = 0.1F;
00020     static constexpr float DEFAULT_FAR = 100.F;
00021
00022     static constexpr float DEFAULT_MOVE_SPEED = 3.F;
00023     static constexpr float DEFAULT_LOOK_SPEED = 1.5F;
00024
00025     ///
00026     /// @class Camera
00027     /// @brief Class for camera
00028     /// @namespace ven
00029     ///
00030     class Camera {
00031
00032     public:
00033
00034         Camera() = default;
00035         ~Camera() = default;
00036
00037         Camera(const Camera&) = delete;
00038         Camera& operator=(const Camera&) = delete;
00039
00040         void setOrthographicProjection(float left, float right, float top, float bottom, float
near, float far);
00041         void setPerspectiveProjection(float aspect);
00042         void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up = {0.F, -1.F,
0.F});
00043         void setViewTarget(const glm::vec3 position, const glm::vec3 target, const glm::vec3 up =
{0.F, -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00044         void setViewXYZ(glm::vec3 position, glm::vec3 rotation);
00045         void setFov(const float fov) { m_fov = fov; }
00046         void setNear(const float near) { m_near = near; }
00047         void setFar(const float far) { m_far = far; }
00048         void setMoveSpeed(const float moveSpeed) { m_moveSpeed = moveSpeed; }
00049         void setLookSpeed(const float lookSpeed) { m_lookSpeed = lookSpeed; }
00050
00051         [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00052         [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00053         [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00054         [[nodiscard]] float getFov() const { return m_fov; }
00055         [[nodiscard]] float getNear() const { return m_near; }
00056         [[nodiscard]] float getFar() const { return m_far; }
00057         [[nodiscard]] float getMoveSpeed() const { return m_moveSpeed; }
00058         [[nodiscard]] float getLookSpeed() const { return m_lookSpeed; }
00059
00060         Transform3DComponent transform3D(DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION);
00061
00062     private:
00063
00064         float m_fov{DEFAULT_FOV};
00065         float m_near{DEFAULT_NEAR};
00066         float m_far{DEFAULT_FAR};
00067         float m_moveSpeed{DEFAULT_MOVE_SPEED};
00068         float m_lookSpeed{DEFAULT_LOOK_SPEED};
00069         glm::mat4 m_projectionMatrix{1.F};
00070         glm::mat4 m_viewMatrix{1.F};
00071         glm::mat4 m_inverseViewMatrix{1.F};
00072
00073     }; // class Camera
00074
00075 } // namespace ven

```

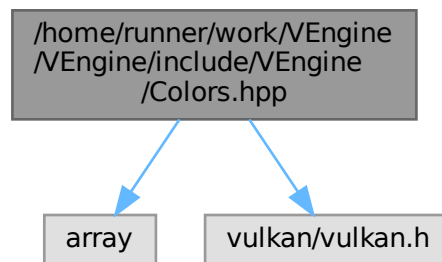
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference

```

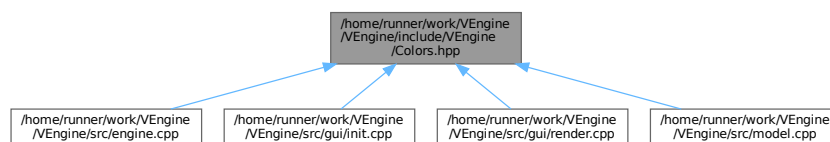
#include <array>
#include <vulkan/vulkan.h>

```

Include dependency graph for Colors.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Colors](#)
Class for colors.

Namespaces

- namespace [ven](#)

Variables

- static constexpr float [ven::COLOR_MAX](#) = 255.0F

8.14 Colors.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Colors.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
  
```

```

00008
00009 #include <array>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 namespace ven {
00014
00015     static constexpr float COLOR_MAX = 255.0F;
00016
00017     ///
00018     /// @class Colors
00019     /// @brief Class for colors
00020     /// @namespace ven
00021     ///
00022     class Colors {
00023
00024     public:
00025
00026         static constexpr glm::vec3 WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX;
00027         static constexpr glm::vec4 WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F };
00028         static constexpr VkClearColorValue WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } };
00029
00030         static constexpr glm::vec3 BLACK_3 = glm::vec3(0.0F);
00031         static constexpr glm::vec4 BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F };
00032         static constexpr VkClearColorValue BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } };
00033
00034         static constexpr glm::vec3 RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
00035         static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F };
00036         static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } };
00037
00038         static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00039         static constexpr glm::vec4 GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00040         static constexpr VkClearColorValue GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };
00041
00042         static constexpr glm::vec3 BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
00043         static constexpr glm::vec4 BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F };
00044         static constexpr VkClearColorValue BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } };
00045
00046         static constexpr glm::vec3 YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
00047         static constexpr glm::vec4 YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F };
00048         static constexpr VkClearColorValue YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } };
00049
00050         static constexpr glm::vec3 CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00051         static constexpr glm::vec4 CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00052         static constexpr VkClearColorValue CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };
00053
00054         static constexpr glm::vec3 MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00055         static constexpr glm::vec4 MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00056         static constexpr VkClearColorValue MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };
00057
00058         static constexpr glm::vec3 SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
00059         static constexpr glm::vec4 SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F };
00060         static constexpr VkClearColorValue SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } };
00061
00062         static constexpr glm::vec3 GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
00063         static constexpr glm::vec4 GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F };
00064         static constexpr VkClearColorValue GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } };
00065
00066         static constexpr glm::vec3 MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
00067         static constexpr glm::vec4 MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F };
00068         static constexpr VkClearColorValue MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } };
00069
00070         static constexpr glm::vec3 OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
00071         static constexpr glm::vec4 OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F };
00072         static constexpr VkClearColorValue OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } };
00073
00074         static constexpr glm::vec3 LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00075         static constexpr glm::vec4 LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00076         static constexpr VkClearColorValue LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };
00077
00078         static constexpr glm::vec3 AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00079         static constexpr glm::vec4 AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00080         static constexpr VkClearColorValue AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };
00081
00082         static constexpr glm::vec3 TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
00083         static constexpr glm::vec4 TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F };
00084         static constexpr VkClearColorValue TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } };
00085
00086         static constexpr glm::vec3 NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
00087         static constexpr glm::vec4 NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F };
00088         static constexpr VkClearColorValue NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } };
00089
00090         static constexpr glm::vec3 FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00091         static constexpr glm::vec4 FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00092         static constexpr VkClearColorValue FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };
00093
00094         static constexpr glm::vec3 NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;

```

```

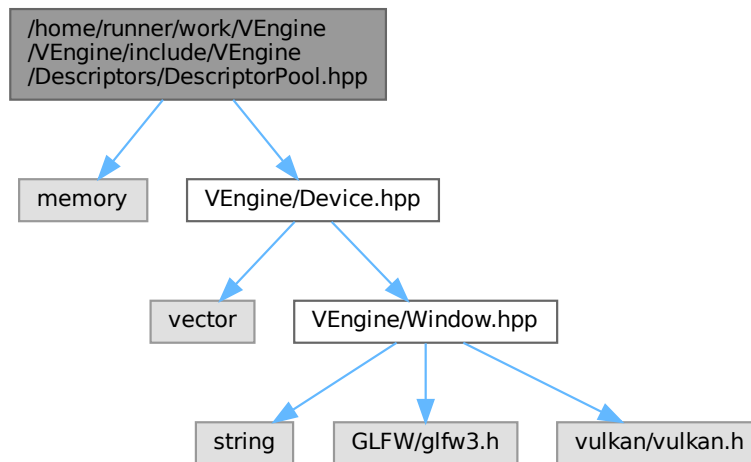
00095     static constexpr glm::vec4 NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F };
00096     static constexpr VkClearColorValue NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } };
00097
00098     static constexpr glm::vec3 SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
00099     static constexpr glm::vec4 SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F };
00100     static constexpr VkClearColorValue SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } };
00101
00102     static constexpr glm::vec3 SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
00103     static constexpr glm::vec4 SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F };
00104     static constexpr VkClearColorValue SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } };
00105
00106
00107     static constexpr std::array<std::pair<const char *, glm::vec3>, 20> COLOR_PRESETS_3 = {{
00108         {"White", WHITE_3},
00109         {"Black", BLACK_3},
00110         {"Red", RED_3},
00111         {"Green", GREEN_3},
00112         {"Blue", BLUE_3},
00113         {"Yellow", YELLOW_3},
00114         {"Cyan", CYAN_3},
00115         {"Magenta", MAGENTA_3},
00116         {"Silver", SILVER_3},
00117         {"Gray", GRAY_3},
00118         {"Maroon", MAROON_3},
00119         {"Olive", OLIVE_3},
00120         {"Lime", LIME_3},
00121         {"Aqua", AQUA_3},
00122         {"Teal", TEAL_3},
00123         {"Navy", NAVY_3},
00124         {"Fuchsia", FUCHSIA_3},
00125         {"Night Blue", NIGHT_BLUE_3},
00126         {"Sky Blue", SKY_BLUE_3},
00127         {"Sunset", SUNSET_3}
00128     }};
00129
00130     static constexpr std::array<std::pair<const char *, glm::vec4>, 20> COLOR_PRESETS_4 = {{
00131         {"White", WHITE_4},
00132         {"Black", BLACK_4},
00133         {"Red", RED_4},
00134         {"Green", GREEN_4},
00135         {"Blue", BLUE_4},
00136         {"Yellow", YELLOW_4},
00137         {"Cyan", CYAN_4},
00138         {"Magenta", MAGENTA_4},
00139         {"Silver", SILVER_4},
00140         {"Gray", GRAY_4},
00141         {"Maroon", MAROON_4},
00142         {"Olive", OLIVE_4},
00143         {"Lime", LIME_4},
00144         {"Aqua", AQUA_4},
00145         {"Teal", TEAL_4},
00146         {"Navy", NAVY_4},
00147         {"Fuchsia", FUCHSIA_4},
00148         {"Night Blue", NIGHT_BLUE_4},
00149         {"Sky Blue", SKY_BLUE_4},
00150         {"Sunset", SUNSET_4}
00151     }};
00152
00153     static constexpr std::array<std::pair<const char *, VkClearColorValue>, 20>
COLOR_PRESETS_VK = {{
00154         {"White", WHITE_V},
00155         {"Black", BLACK_V},
00156         {"Red", RED_V},
00157         {"Green", GREEN_V},
00158         {"Blue", BLUE_V},
00159         {"Yellow", YELLOW_V},
00160         {"Cyan", CYAN_V},
00161         {"Magenta", MAGENTA_V},
00162         {"Silver", SILVER_V},
00163         {"Gray", GRAY_V},
00164         {"Maroon", MAROON_V},
00165         {"Olive", OLIVE_V},
00166         {"Lime", LIME_V},
00167         {"Aqua", AQUA_V},
00168         {"Teal", TEAL_V},
00169         {"Navy", NAVY_V},
00170         {"Fuchsia", FUCHSIA_V},
00171         {"Night Blue", NIGHT_BLUE_V},
00172         {"Sky Blue", SKY_BLUE_V},
00173         {"Sunset", SUNSET_V}
00174     }};
00175
00176     }; // class Colors
00177
00178 } // namespace ven

```

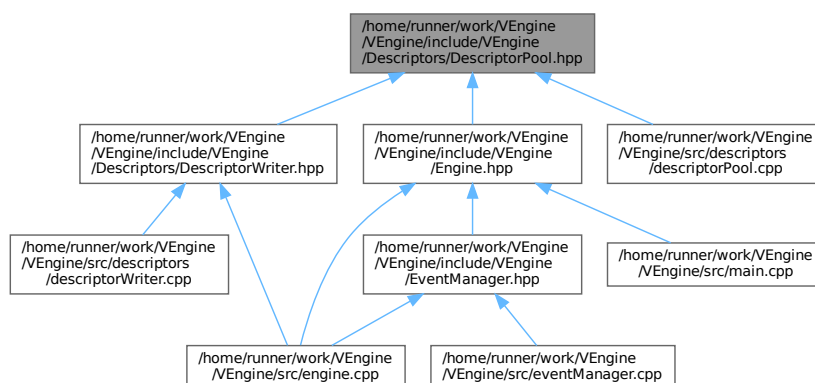
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference

This file contains the DescriptorPool class.

```
#include <memory>
#include "VEngine/Device.hpp"
Include dependency graph for DescriptorPool.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorPool](#)
Class for descriptor pool.
- class [ven::DescriptorPool::Builder](#)

Namespaces

- namespace [ven](#)

Variables

- static constexpr uint32_t [ven::DEFAULT_MAX_SETS](#) = 1000

8.15.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file [DescriptorPool.hpp](#).

8.16 DescriptorPool.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file DescriptorPool.hpp
00003 /// @brief This file contains the DescriptorPool class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr uint32_t DEFAULT_MAX_SETS = 1000;
00016
00017     ///
00018     /// @class DescriptorPool
00019     /// @brief Class for descriptor pool
00020     /// @namespace ven
00021     ///
00022     class DescriptorPool {
00023
00024     public:
00025
00026         class Builder {
00027
00028         public:
00029
00030             explicit Builder(Device &device) : m_device{device} {}
00031
00032             [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00033
00034             Builder &addPoolSize(const VkDescriptorType descriptorType, const uint32_t count)
{ m_poolSizes.push_back({descriptorType, count}); return *this; }
00035             Builder &setPoolFlags(const VkDescriptorPoolCreateFlags flags) { m_poolFlags =
flags; return *this; }
00036             Builder &setMaxSets(const uint32_t count) { m_maxSets = count; return *this; }
00037
00038         private:
00039
00040             Device &m_device;
00041             std::vector<VkDescriptorPoolSize> m_poolSizes;
00042             uint32_t m_maxSets{DEFAULT_MAX_SETS};
00043             VkDescriptorPoolCreateFlags m_poolFlags{0};
00044
00045         }; // class Builder
00046
00047         DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
const std::vector<VkDescriptorPoolSize> &poolSizes);
00048         ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
}
00049

```

```

00050         DescriptorPool(const DescriptorPool &) = delete;
00051         DescriptorPool &operator=(const DescriptorPool &) = delete;
00052
00053         bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
&descriptor) const;
00054         void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
descriptors.data()); }
00055         void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00056
00057         [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }
00058
00059     private:
00060
00061         Device &m_device;
00062         VkDescriptorPool m_descriptorPool;
00063         friend class DescriptorWriter;
00064
00065     }; // class DescriptorPool
00066
00067 } // namespace ven

```

8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference

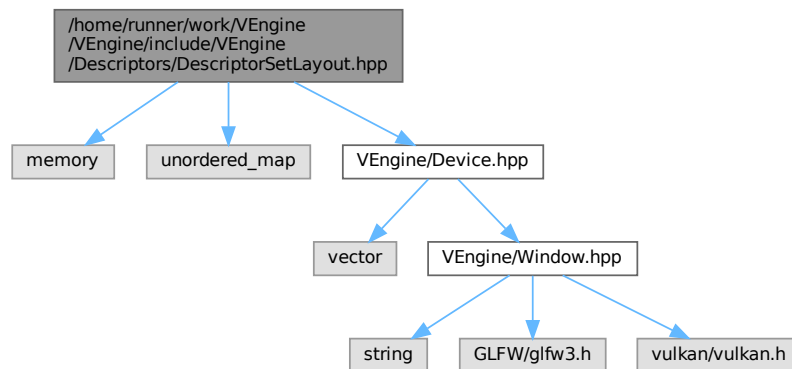
This file contains the DescriptorSetLayout class.

```

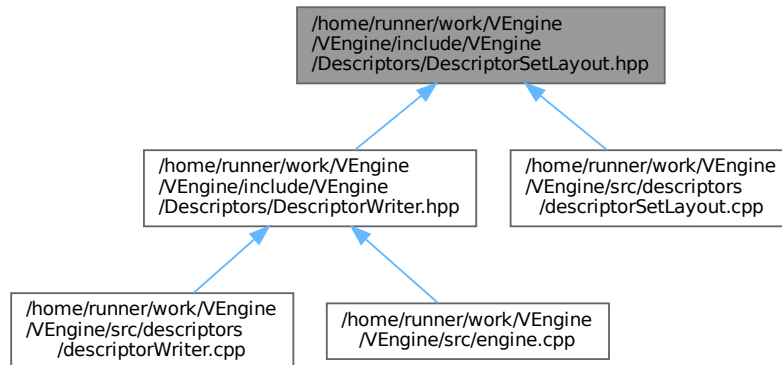
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"

```

Include dependency graph for DescriptorSetLayout.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorSetLayout](#)
Class for descriptor set layout.
- class [ven::DescriptorSetLayout::Builder](#)

Namespaces

- namespace [ven](#)

8.17.1 Detailed Description

This file contains the DescriptorSetLayout class.

Definition in file [DescriptorSetLayout.hpp](#).

8.18 DescriptorSetLayout.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file DescriptorSetLayout.hpp
00003 /// @brief This file contains the DescriptorSetLayout class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     ///
00017     /// @class DescriptorSetLayout
00018     /// @brief Class for descriptor set layout
  
```



```

00019     /// @namespace ven
00020     ///
00021     class DescriptorSetLayout {
00022     public:
00023
00024         class Builder {
00025         public:
00026
00027             explicit Builder(Device &device) : m_device{device} {}
00028
00029             Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType,
00030                               VkShaderStageFlags stageFlags, uint32_t count = 1);
00031             std::unique_ptr<DescriptorSetLayout> build() const { return
00032             std::make_unique<DescriptorSetLayout>(m_device, m_bindings); }
00033
00034     private:
00035
00036         Device &m_device;
00037         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00038
00039     }; // class Builder
00040
00041     DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
00042                               VkDescriptorSetLayoutBinding>& bindings);
00043     ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(),
00044                               m_descriptorSetLayout, nullptr); }
00045
00046     DescriptorSetLayout(const DescriptorSetLayout &) = delete;
00047     DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;
00048
00049     VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }
00050
00051     private:
00052
00053         Device &m_device;
00054         VkDescriptorSetLayout m_descriptorSetLayout;
00055         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00056
00057         friend class DescriptorWriter;
00058     }; // class DescriptorSetLayout
00059 } // namespace ven

```

8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference

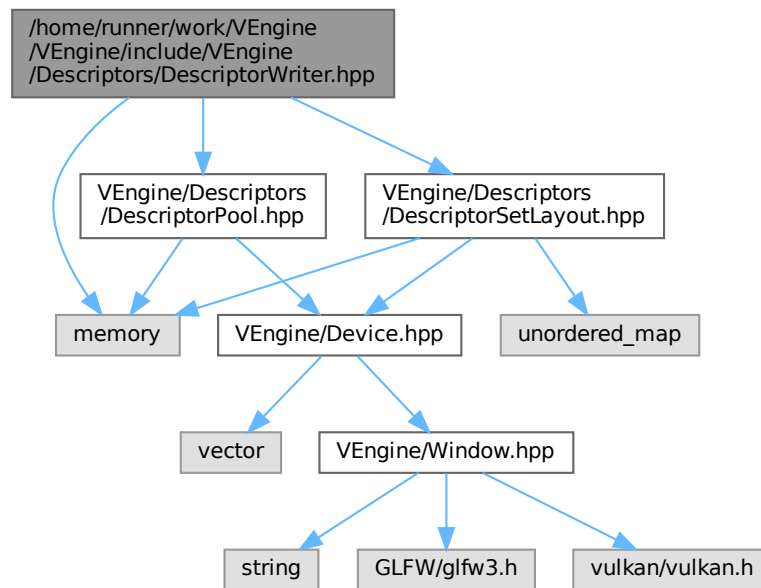
This file contains the DescriptorsWriter class.

```

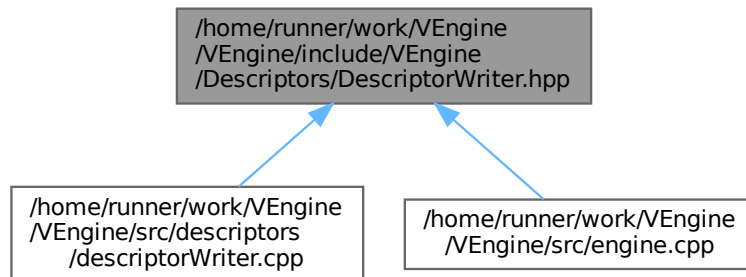
#include <memory>
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

Include dependency graph for DescriptorWriter.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorWriter](#)
Class for descriptor writer.

Namespaces

- namespace [ven](#)

8.19.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file [DescriptorWriter.hpp](#).

8.20 DescriptorWriter.hpp

[Go to the documentation of this file.](#)

```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <memory>  
00010 ///  
00011 #include "VEngine/Descriptors/DescriptorPool.hpp"  
00012 #include "VEngine/Descriptors/DescriptorSetLayout.hpp"  
00013 ///  
00014 namespace ven {  
00015     ///  
00016     ///  
00017     ///  
00018     ///  
00019     ///  
00020     ///  
00021     class DescriptorWriter {  
00022     public:  
00023         DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :  
00024             m_setLayout(setLayout), m_pool{pool} {}  
00025         ~DescriptorWriter() = default;  
00026         DescriptorWriter(const DescriptorWriter &) = delete;  
00027         DescriptorWriter &operator=(const DescriptorWriter &) = delete;  
00028         DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);  
00029         DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);  
00030         bool build(VkDescriptorSet &set);  
00031         void overwrite(const VkDescriptorSet &set);  
00032     private:  
00033         DescriptorSetLayout &m_setLayout;  
00034         DescriptorPool &m_pool;  
00035         std::vector<VkWriteDescriptorSet> m_writes;  
00036     }; // class DescriptorWriter  
00037 } // namespace ven
```

8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

This file contains the Device class.

```
#include <vector>  
#include "VEngine/Window.hpp"
```


8.22 Device.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Device.hpp
00003 /// @brief This file contains the Device class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vector>
00010
00011 #include "VEngine/Window.hpp"
00012
00013 namespace ven {
00014
00015     struct SwapChainSupportDetails {
00016         VkSurfaceCapabilitiesKHR capabilities;
00017         std::vector<VkSurfaceFormatKHR> formats;
00018         std::vector<VkPresentModeKHR> presentModes;
00019     };
00020
00021     struct QueueFamilyIndices {
00022         uint32_t graphicsFamily{};
00023         uint32_t presentFamily{};
00024         bool graphicsFamilyHasValue = false;
00025         bool presentFamilyHasValue = false;
00026         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
presentFamilyHasValue; }
00027     };
00028
00029     ///
00030     /// @class Device
00031     /// @brief Class for device
00032     /// @namespace ven
00033     ///
00034     class Device {
00035     public:
00036
00037         #ifdef NDEBUG
00038             const bool enableValidationLayers = false;
00039         #else
00040             const bool enableValidationLayers = true;
00041         #endif
00042
00043         explicit Device(Window &window);
00044         ~Device();
00045
00046         Device(const Device&) = delete;
00047         Device& operator=(const Device&) = delete;
00048
00049         [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
00050         [[nodiscard]] VkDevice device() const { return m_device; }
00051         [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
00052         [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
00053         [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }
00054
00055         [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return
querySwapChainSupport(m_physicalDevice); }
00056         [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags
properties) const;
00057         [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
findQueueFamilies(m_physicalDevice); }
00058         [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00059         [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00060         [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
VkImageTiling tiling, VkFormatFeatureFlags features) const;
00061
00062         // Buffer Helper Functions
00063         void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00064         [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00065         void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00066         void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00067         void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
uint32_t layerCount) const;
00068
00069         void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00070
00071     private:
00072
00073
00074

```

```

00075         void createInstance();
00076         void setupDebugMessenger();
00077         void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00078         void pickPhysicalDevice();
00079         void createLogicalDevice();
00080         void createCommandPool();
00081
00082         // helper functions
00083         bool isDeviceSuitable(VkPhysicalDevice device) const;
00084         [[nodiscard]] std::vector<const char*> getRequiredExtensions() const;
00085         [[nodiscard]] bool checkValidationLayerSupport() const;
00086         QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00087         static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
&createInfo);
00088         void hasGlfwRequiredInstanceExtensions() const;
00089         bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00090         SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00091
00092         VkInstance m_instance;
00093         VkDebugUtilsMessengerEXT m_debugMessenger;
00094         VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00095         Window &m_window;
00096         VkCommandPool m_commandPool;
00097
00098         VkDevice m_device;
00099         VkSurfaceKHR m_surface;
00100         VkQueue m_graphicsQueue;
00101         VkQueue m_presentQueue;
00102         VkPhysicalDeviceProperties m_properties;
00103
00104         const std::vector<const char*> m_validationLayers = {"VK_LAYER_KHRONOS_validation"};
00105         const std::vector<const char*> m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00106
00107     }; // class Device
00108
00109 } // namespace ven

```

8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

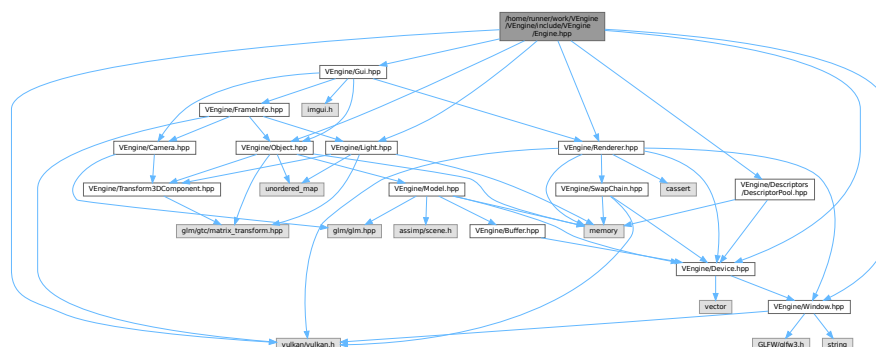
This file contains the Engine class.

```

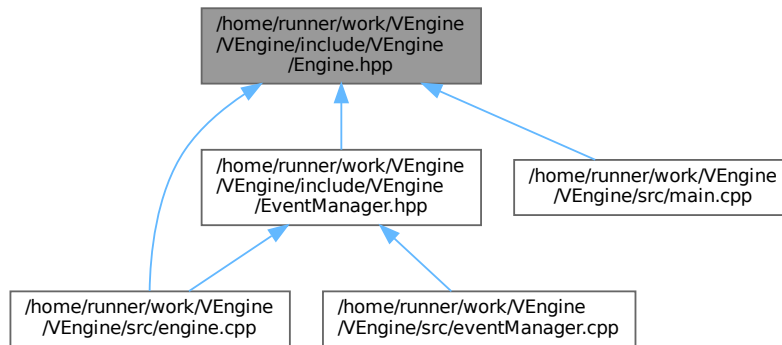
#include <vulkan/vulkan.h>
#include "VEngine/Gui.hpp"
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Light.hpp"

```

Include dependency graph for Engine.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Engine](#)
Class for engine.

Namespaces

- namespace [ven](#)

Enumerations

- enum [ven::ENGINE_STATE](#) : uint8_t { [ven::EDITOR](#) = 0 , [ven::GAME](#) = 1 , [ven::PAUSED](#) = 2 , [ven::EXIT](#) = 3 }

8.23.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

8.24 Engine.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Engine.hpp
00003 /// @brief This file contains the Engine class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Gui.hpp"
00012 #include "VEngine/Window.hpp"

```

```

00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Object.hpp"
00015 #include "VEngine/Renderer.hpp"
00016 #include "VEngine/Descriptors/DescriptorPool.hpp"
00017 #include "VEngine/Light.hpp"
00018
00019 namespace ven {
00020
00021     enum ENGINE_STATE : uint8_t {
00022         EDITOR = 0,
00023         GAME = 1,
00024         PAUSED = 2,
00025         EXIT = 3
00026     };
00027
00028     ///
00029     /// @class Engine
00030     /// @brief Class for engine
00031     /// @namespace ven
00032     ///
00033     class Engine {
00034
00035     public:
00036
00037         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string
&title = DEFAULT_TITLE.data());
00038         ~Engine() = default;
00039
00040         Engine(const Engine&) = delete;
00041         Engine operator=(const Engine&) = delete;
00042
00043         void mainLoop();
00044
00045     private:
00046
00047         void loadObjects();
00048
00049         ENGINE_STATE m_state{EXIT};
00050
00051         Window m_window;
00052         Device m_device{m_window};
00053         Renderer m_renderer{m_window, m_device};
00054         Gui m_gui;
00055         std::unique_ptr<DescriptorPool> m_globalPool;
00056         Object::Map m_objects;
00057         Light::Map m_lights;
00058
00059         VkInstance m_instance{nullptr};
00060         VkSurfaceKHR m_surface{nullptr};
00061
00062         void createInstance();
00063         void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWWindow(),
nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00064
00065     }; // class Engine
00066
00067 } // namespace ven

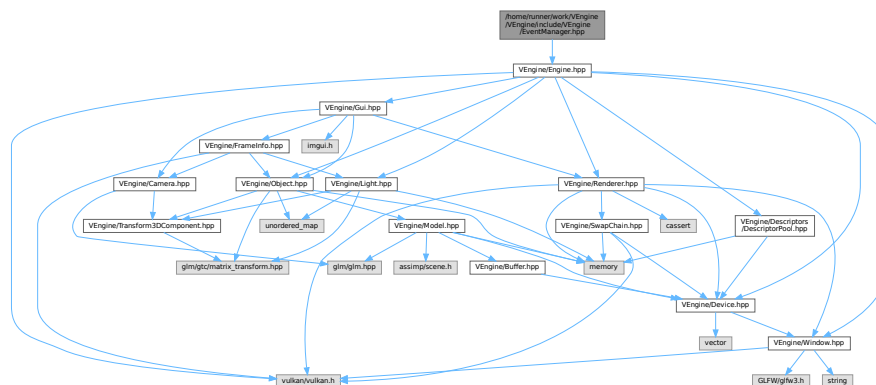
```

8.25 /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp File Reference

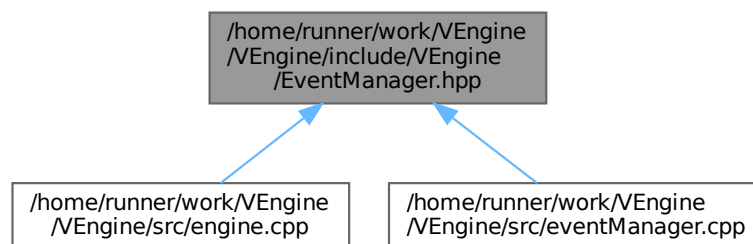
This file contains the EventManager class.


```
#include "VEngine/Engine.hpp"
```

Include dependency graph for EventManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::KeyAction](#)
- struct [ven::KeyMappings](#)
- class [ven::EventManager](#)

Class for event manager.

Namespaces

- namespace [ven](#)

8.25.1 Detailed Description

This file contains the EventManager class.

Definition in file [EventManager.hpp](#).

8.26 EventManager.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file EventManager.hpp
00003 /// @brief This file contains the EventManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Engine.hpp"
00010
00011 namespace ven {
00012
00013     struct KeyAction {
00014         uint16_t key;
00015         glm::vec3* dir;
00016         glm::vec3 value;
00017     };
00018
00019     struct KeyMappings {
00020         uint16_t moveLeft = GLFW_KEY_A;
00021         uint16_t moveRight = GLFW_KEY_D;
00022         uint16_t moveForward = GLFW_KEY_W;
00023         uint16_t moveBackward = GLFW_KEY_S;
00024         uint16_t moveUp = GLFW_KEY_SPACE;
00025         uint16_t moveDown = GLFW_KEY_LEFT_SHIFT;
00026         uint16_t lookLeft = GLFW_KEY_LEFT;
00027         uint16_t lookRight = GLFW_KEY_RIGHT;
00028         uint16_t lookUp = GLFW_KEY_UP;
00029         uint16_t lookDown = GLFW_KEY_DOWN;
00030         uint16_t toggleGui = GLFW_KEY_F1;
00031     };
00032
00033     ///
00034     /// @class EventManager
00035     /// @brief Class for event manager
00036     /// @namespace ven
00037     ///
00038     class EventManager {
00039     public:
00040
00041         EventManager() = default;
00042         ~EventManager() = default;
00043
00044         EventManager(const EventManager&) = delete;
00045         EventManager& operator=(const EventManager&) = delete;
00046
00047         void handleEvents(GLFWwindow* window, ENGINE_STATE* engineState, Camera& camera, Gui& gui,
00048             float dt) const;
00049
00050     private:
00051
00052         void moveCamera(GLFWwindow* window, Camera& camera, Gui& gui, float dt) const;
00053         static void updateEngineState(ENGINE_STATE* engineState, const ENGINE_STATE newState) {
00054             *engineState = newState; }
00054         static bool isKeyJustPressed(GLFWwindow* window, int key, std::unordered_map<int, bool>&
00055             keyStates);
00055
00056         KeyMappings m_keys{};
00057         mutable std::unordered_map<int, bool> m_keyState;
00058
00059     }; // class EventManager
00060
00061 } // namespace ven

```

8.27 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference

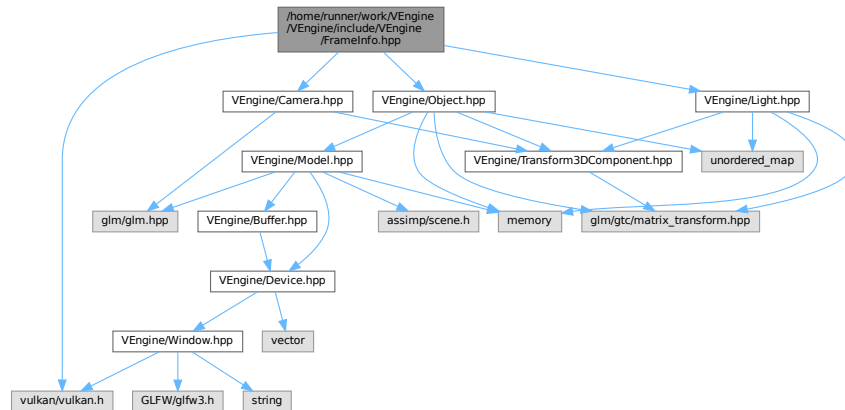
This file contains the FrameInfo class.

```

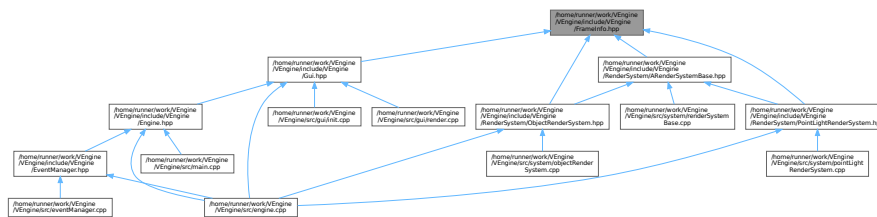
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"

```

```
#include "VEngine/Light.hpp"
Include dependency graph for FrameInfo.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `ven::PointLightData`
- struct `ven::GlobalUbo`
- struct `ven::FrameInfo`

Namespaces

- namespace **ven**

Variables

- static constexpr uint16_t ven::MAX_LIGHTS = 10
- static constexpr float ven::DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F
- static constexpr glm::vec4 ven::DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENSITY}

8.27.1 Detailed Description

This file contains the FrameInfo class.

Definition in file [FrameInfo.hpp](#).

8.28 FrameInfo.hpp

[Go to the documentation of this file.](#)

```

00001 ///  

00002 ///  

00003 ///  

00004 ///  

00005 ///  

00006 ///  

00007 #pragma once  

00008 ///  

00009 #include <vulkan/vulkan.h>  

00010 ///  

00011 #include "VEngine/Camera.hpp"  

00012 #include "VEngine/Object.hpp"  

00013 #include "VEngine/Light.hpp"  

00014 ///  

00015 namespace ven {  

00016 ///  

00017 static constexpr uint16_t MAX_LIGHTS = 10;  

00018 ///  

00019 static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F;  

00020 static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F),  

    DEFAULT_AMBIENT_LIGHT_INTENSITY};  

00021 ///  

00022 struct PointLightData  

00023 {  

00024     glm::vec4 position{};  

00025     glm::vec4 color{};  

00026 };  

00027 ///  

00028 struct GlobalUbo  

00029 {  

00030     glm::mat4 projection{1.F};  

00031     glm::mat4 view{1.F};  

00032     glm::mat4 inverseView{1.F};  

00033     glm::vec4 ambientLightColor{DEFAULT_AMBIENT_LIGHT_COLOR};  

00034     std::array<PointLightData, MAX_LIGHTS> pointLights;  

00035     uint16_t numLights;  

00036 };  

00037 ///  

00038 struct FrameInfo  

00039 {  

00040     unsigned long frameIndex;  

00041     float frameTime;  

00042     VkCommandBuffer commandBuffer;  

00043     Camera &camera;  

00044     VkDescriptorSet globalDescriptorSet;  

00045     Object::Map &objects;  

00046     Light::Map &lights;  

00047 };  

00048 ///  

00049 } // namespace ven

```

8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp File Reference

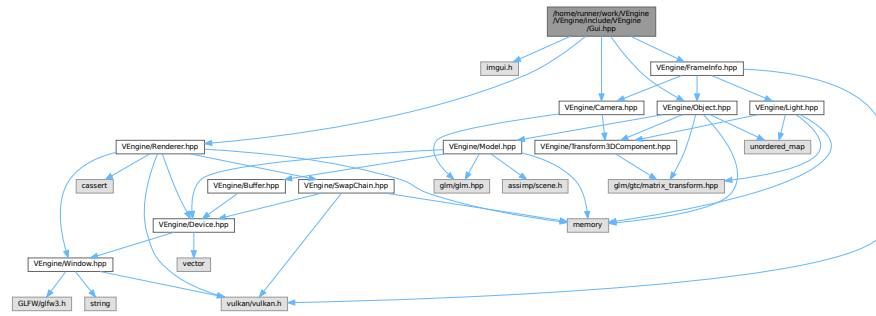
This file contains the ImGuiWindowManager class.

```

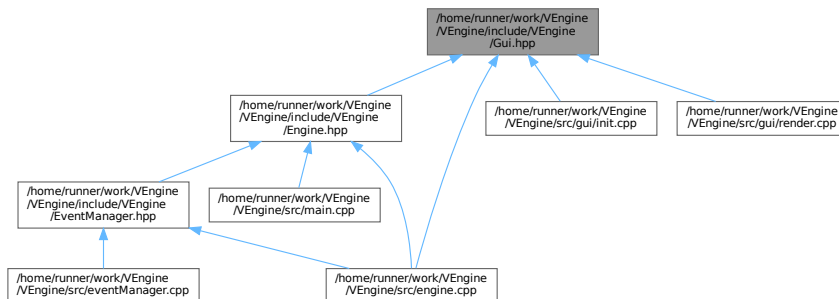
#include <imgui.h>
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/FrameInfo.hpp"

```

Include dependency graph for Gui.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Gui](#)
Class for *Gui*.
- struct [ven::Gui::funcs](#)

Namespaces

- namespace [ven](#)

Enumerations

- enum [ven::GUI_STATE](#) : `uint8_t` { [ven::VISIBLE](#) = 0 , [ven::HIDDEN](#) = 1 }

8.29.1 Detailed Description

This file contains the `ImGuiWindowManager` class.

Definition in file [Gui.hpp](#).

8.30 Gui.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Gui.hpp
00003 /// @brief This file contains the ImGuiWindowManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <imgui.h>
00010
00011 #include "VEngine/Object.hpp"
00012 #include "VEngine/Renderer.hpp"
00013 #include "VEngine/Camera.hpp"
00014 #include "VEngine/FrameInfo.hpp"
00015
00016 namespace ven {
00017
00018     enum GUI_STATE : uint8_t {
00019         VISIBLE = 0,
00020         HIDDEN = 1
00021     };
00022
00023     ///
00024     /// @class Gui
00025     /// @brief Class for Gui
00026     /// @namespace ven
00027     ///
00028     class Gui {
00029
00030     public:
00031
00032         Gui() = default;
00033         ~Gui() = default;
00034
00035         Gui(const Gui&) = delete;
00036         Gui& operator=(const Gui&) = delete;
00037
00038         static void init(GLFWwindow* window, VkInstance instance, const Device* device,
00039             VkRenderPass renderPass);
00040
00041         static void render(Renderer *renderer, std::unordered_map<unsigned int, Object>& objects,
00042             std::unordered_map<unsigned int, Light>& lights, Camera& camera, VkPhysicalDevice physicalDevice,
00043             GlobalUbo& ubo);
00044         static void cleanup();
00045
00046         void setState(const GUI_STATE state) { m_state = state; }
00047         [[nodiscard]] GUI_STATE getState() const { return m_state; }
00048
00049     private:
00050
00051         static void initStyle();
00052         static void renderFrameWindow();
00053         static void cameraSection(Camera& camera);
00054         static void inputsSection(const ImGuiIO *io);
00055         static void rendererSection(Renderer *renderer, GlobalUbo& ubo);
00056         static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
00057         static void objectsSection(std::unordered_map<unsigned int, Object>& objects);
00058         static void lightsSection(std::unordered_map<unsigned int, Light>& lights);
00059
00060         struct funcs { static bool IsLegacyNativeDupe(const ImGuiKey key) { return key >= 0 && key
00061             < 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist
00062
00063         static ImGuiIO* m_io;
00064         GUI_STATE m_state{VISIBLE};
00065     }; // class Gui
00066 } // namespace ven

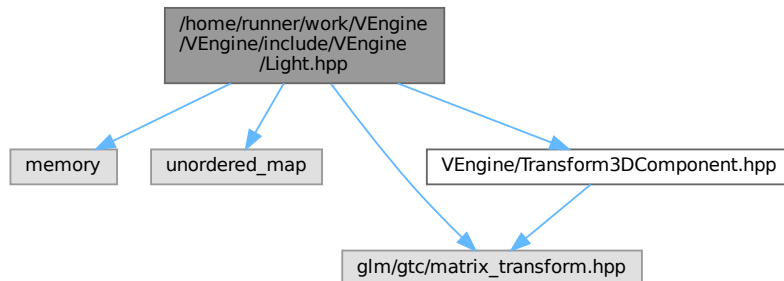
```

8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp

File Reference

This file contains the Light class.

Include dependency graph for Light.hpp:

[illegible]

- class `ven::Light`
Class for light.

- namespace **ven**

- static constexpr float `ven::DEFAULT_LIGHT_INTENSITY` = .2F
- static constexpr float `ven::DEFAULT_LIGHT_RADIUS` = 0.1F
- static constexpr glm::vec4 `ven::DEFAULT_LIGHT_COLOR` = {glm::vec3(1.F), `DEFAULT_LIGHT_INTENSITY`}

Definition in file [Light.hpp](#).

8.32 Light.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Light.hpp
00003 /// @brief This file contains the Light class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include <glm/gtc/matrix_transform.hpp>
00013
00014 #include "VEngine/Transform3DComponent.hpp"
00015
00016 namespace ven {
00017
00018     static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
00019     static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
00020     static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY};
00021
00022     ///
00023     /// @class Light
00024     /// @brief Class for light
00025     /// @namespace ven
00026     ///
00027     class Light {
00028     public:
00029
00030         using Map = std::unordered_map<unsigned int, Light>;
00031
00032         ~Light() = default;
00033
00034         Light(const Light&) = delete;
00035         Light& operator=(const Light&) = delete;
00036         Light(Light&&) = default;
00037         Light& operator=(Light&&) = default;
00038
00039         static Light createLight(float radius = DEFAULT_LIGHT_RADIUS, glm::vec4 color =
00040 DEFAULT_LIGHT_COLOR);
00041
00042         glm::vec4 color{DEFAULT_LIGHT_COLOR};
00043         Transform3DComponent transform3D{};
00044
00045         [[nodiscard]] unsigned int getId() const { return m_lightId; }
00046         [[nodiscard]] std::string getName() const { return m_name; }
00047
00048         void setName(const std::string &name) { m_name = name; }
00049
00050     private:
00051
00052         explicit Light(const unsigned int lightId) : m_lightId(lightId) {}
00053
00054         unsigned int m_lightId;
00055         std::string m_name{"point light"};
00056
00057     }; // class Light
00058
00059 } // namespace ven

```

8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp

File Reference

This file contains the Model class.

```

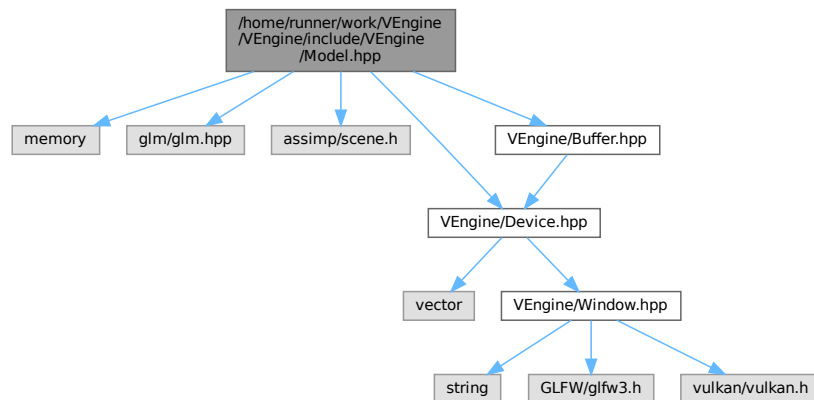
#include <memory>
#include <glm/glm.hpp>
#include <assimp/scene.h>
#include "VEngine/Device.hpp"

```

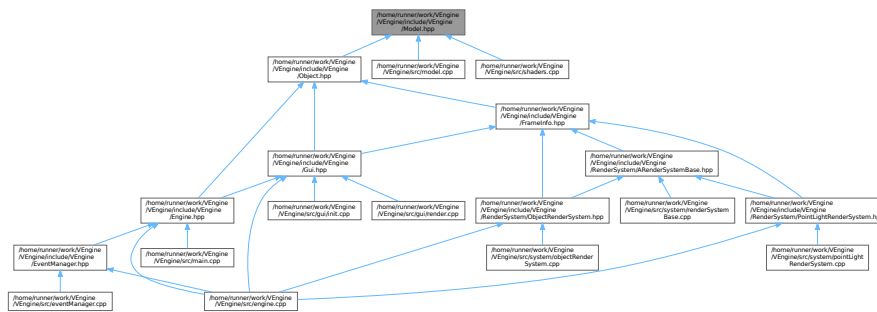


```
#include "VEngine/Buffer.hpp"
```

Include dependency graph for Model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Model](#)
Class for model.
- struct [ven::Model::Vertex](#)
- struct [ven::Model::Builder](#)

Namespaces

- namespace [ven](#)

8.33.1 Detailed Description

This file contains the Model class.

Definition in file [Model.hpp](#).

8.34 Model.hpp

[Go to the documentation of this file.](#)

```

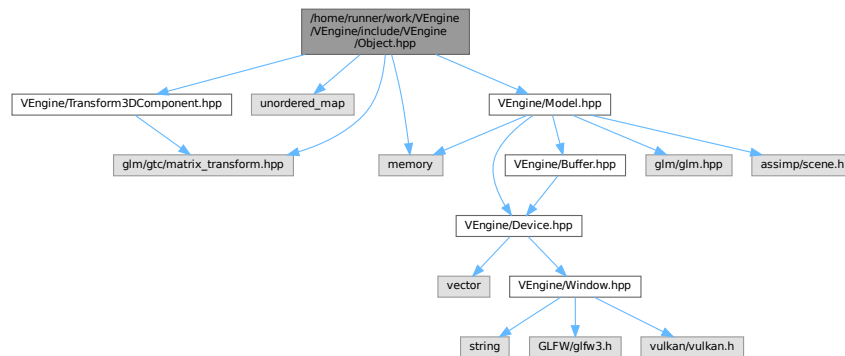
00001 ///
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include <glm/glm.hpp>
00012
00013 #include <assimp/scene.h>
00014
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/Buffer.hpp"
00017
00018 namespace ven {
00019
00020     ///
00021     /// @class Model
00022     /// @brief Class for model
00023     /// @namespace ven
00024     ///
00025     class Model {
00026
00027     public:
00028
00029         struct Vertex {
00030             glm::vec3 position{};
00031             glm::vec3 color{};
00032             glm::vec3 normal{};
00033             glm::vec2 uv{};
00034
00035             static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00036             static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00037
00038             bool operator==(const Vertex& other) const {
00039                 return position == other.position && color == other.color && normal ==
other.normal && uv == other.uv;
00040             }
00041         };
00042
00043         struct Builder {
00044             std::vector<Vertex> vertices;
00045             std::vector<uint32_t> indices;
00046
00047             void loadModel(const std::string &filename);
00048             void processNode(const aiNode* node, const aiScene* scene);
00049             void processMesh(const aiMesh* mesh, const aiScene* scene);
00050         };
00051
00052         Model(Device &device, const Builder &builder);
00053         ~Model() = default;
00054
00055         Model(const Model&) = delete;
00056         void operator=(const Model&) = delete;
00057
00058         static std::unique_ptr<Model> createModelFromFile(Device &device, const std::string
&filename);
00059
00060         void bind(VkCommandBuffer commandBuffer) const;
00061         void draw(VkCommandBuffer commandBuffer) const;
00062
00063     private:
00064
00065         void createVertexBuffer(const std::vector<Vertex>& vertices);
00066         void createIndexBuffer(const std::vector<uint32_t>& indices);
00067
00068         Device& m_device;
00069         std::unique_ptr<Buffer> m_vertexBuffer;
00070         uint32_t m_vertexCount;
00071
00072         bool m_hasIndexBuffer{false};
00073         std::unique_ptr<Buffer> m_indexBuffer;
00074         uint32_t m_indexCount;
00075
00076     }; // class Model
00077
00078 } // namespace ven

```

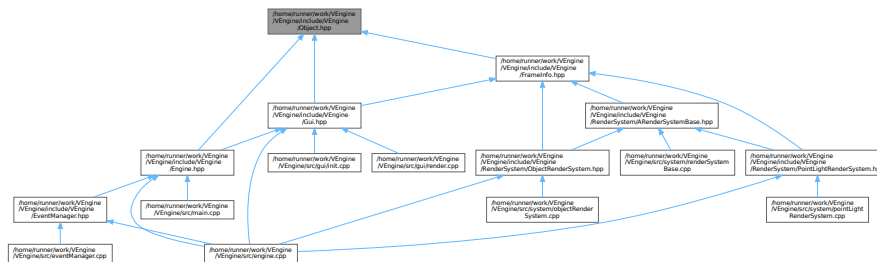
8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

This file contains the Object class.

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"
#include "VEngine/Transform3DComponent.hpp"
Include dependency graph for Object.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Object](#)
Class for object.

Namespaces

- namespace [ven](#)

8.35.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

8.36 Object.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Object.hpp
00003 /// @brief This file contains the Object class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include <glm/gtc/matrix_transform.hpp>
00013
00014 #include "VEngine/Model.hpp"
00015 #include "VEngine/Transform3DComponent.hpp"
00016
00017 namespace ven {
00018
00019     ///
00020     /// @class Object
00021     /// @brief Class for object
00022     /// @namespace ven
00023     ///
00024     class Object {
00025
00026     public:
00027
00028         using Map = std::unordered_map<unsigned int, Object>;
00029
00030         ~Object() = default;
00031
00032         Object(const Object&) = delete;
00033         Object& operator=(const Object&) = delete;
00034         Object(Object&&) = default;
00035         Object& operator=(Object&&) = default;
00036
00037         static Object createObject() { static unsigned int objId = 0; return Object(objId++); }
00038
00039         [[nodiscard]] unsigned int getId() const { return m_objId; }
00040         [[nodiscard]] std::string getName() const { return m_name; }
00041         [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }
00042
00043         void setName(const std::string &name) { m_name = name; }
00044         void setModel(const std::shared_ptr<Model> &model) { m_model = model; }
00045
00046         Transform3DComponent transform3D{};
00047
00048     private:
00049
00050         explicit Object(const unsigned int objId) : m_objId(objId) {}
00051
00052         unsigned int m_objId;
00053         std::string m_name;
00054         std::shared_ptr<Model> m_model;
00055
00056     }; // class Object
00057
00058 } // namespace ven

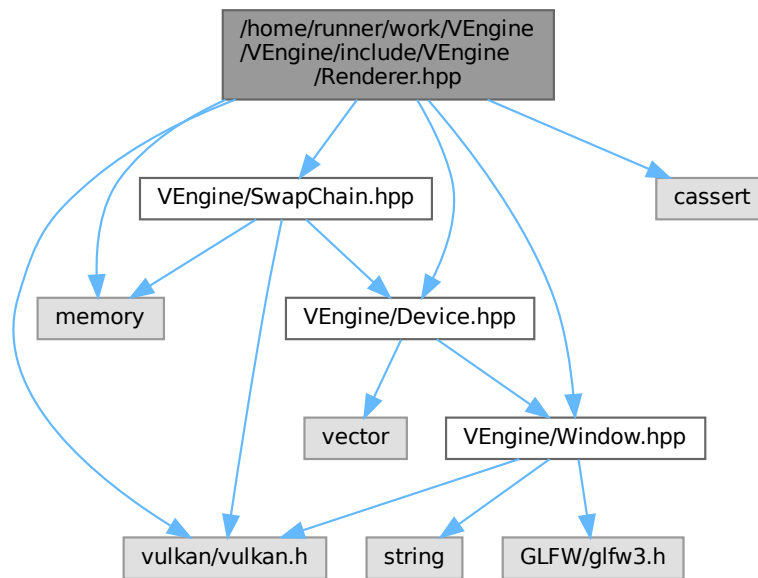
```

8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

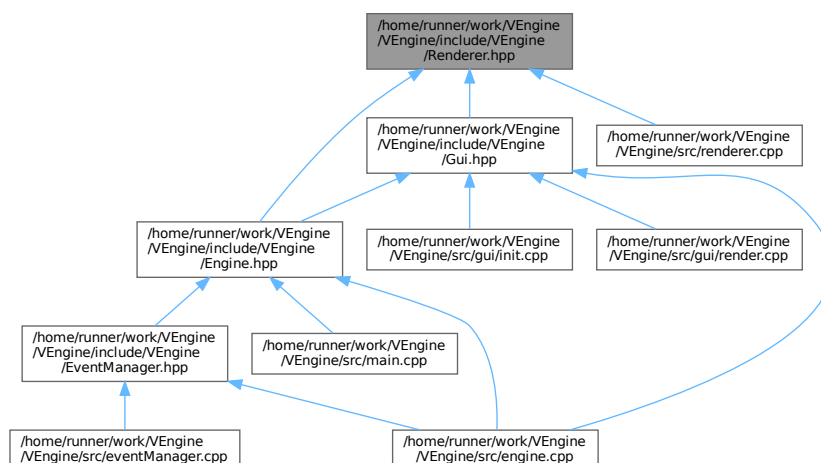
This file contains the Renderer class.

```
#include <memory>
#include <cassert>
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/SwapChain.hpp"
```

Include dependency graph for Renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Renderer](#)

Class for renderer.

Namespaces

- namespace [ven](#)

Variables

- static constexpr VkClearColorValue [ven::DEFAULT_CLEAR_COLOR](#) = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue [ven::DEFAULT_CLEAR_DEPTH](#) = {1.0F, 0}

8.37.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

8.38 Renderer.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Renderer.hpp
00003 /// @brief This file contains the Renderer class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <cassert>
00011
00012 #include <vulkan/vulkan.h>
00013
00014 #include "VEngine/Window.hpp"
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/SwapChain.hpp"
00017
00018 namespace ven {
00019
00020     static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}};
00021     static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0};
00022
00023     ///
00024     /// @class Renderer
00025     /// @brief Class for renderer
00026     /// @namespace ven
00027     ///
00028     class Renderer {
00029
00030     public:
00031
00032         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
00033             recreateSwapChain(); createCommandBuffers(); }
00034         ~Renderer() { freeCommandBuffers(); }
00035
00036         Renderer(const Renderer &) = delete;
00037         Renderer& operator=(const Renderer &) = delete;
00038
00039         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
00040             m_swapChain->getRenderPass(); }
00041         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00042         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00043         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress())
00044             && "cannot get command m_buffer when frame not in progress"; return
00045             m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
```

```

00043     [[nodiscard]] unsigned long getFrameIndex() const { assert(isFrameInProgress()) && "cannot
get frame index when frame not in progress"); return m_currentFrameIndex; }
00044     [[nodiscard]] std::array<float, 4> getClearColor() const { return {
00045         m_clearValues[0].color.float32[0],
00046         m_clearValues[0].color.float32[1],
00047         m_clearValues[0].color.float32[2],
00048         m_clearValues[0].color.float32[3]
00049     }; }
00050
00051     [[nodiscard]] Window& getWindow() const { return m_window; }
00052
00053     void setClearColor(const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR, const
VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00054     VkCommandBuffer beginFrame();
00055     void endFrame();
00056     void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00057     void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00058
00059     private:
00060
00061     void createCommandBuffers();
00062     void freeCommandBuffers();
00063     void recreateSwapChain();
00064
00065     Window &m_window;
00066     Device &m_device;
00067     std::unique_ptr<SwapChain> m_swapChain;
00068     std::vector<VkCommandBuffer> m_commandBuffers;
00069     std::array<VkClearValue, 2> m_clearValues{DEFAULT_CLEAR_COLOR, 1.0F, 0.F};
00070
00071     uint32_t m_currentImageIndex{0};
00072     unsigned long m_currentFrameIndex{0};
00073     bool m_isFrameStarted{false};
00074
00075 }; // class Renderer
00076
00077 } // namespace ven

```

8.39 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp File Reference

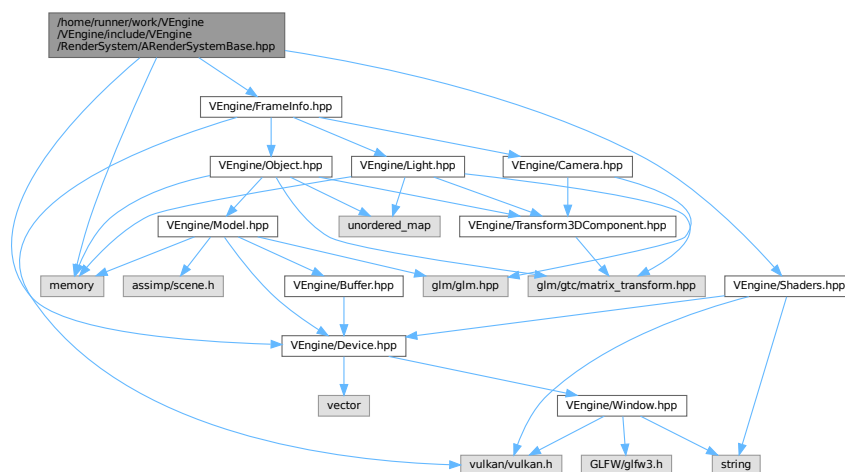
This file contains the ARenderSystemBase class.

```

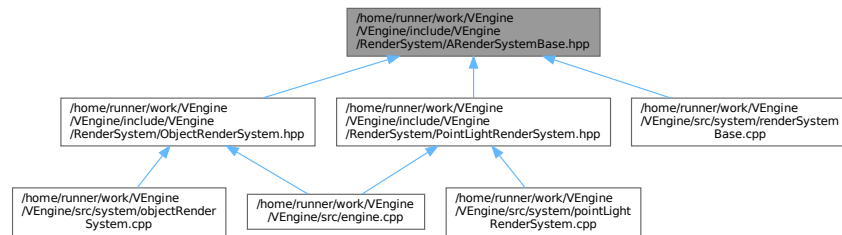
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

```

Include dependency graph for ARenderSystemBase.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::ARenderSystemBase](#)
Abstract class for render system base.

Namespaces

- namespace [ven](#)

8.39.1 Detailed Description

This file contains the ARenderSystemBase class.

Definition in file [ARenderSystemBase.hpp](#).

8.40 ARenderSystemBase.hpp

[Go to the documentation of this file.](#)

```

00001 ///  

00002 ///  

00003 ///  

00004 ///  

00005 ///  

00006 ///  

00007 #pragma once  

00008  

00009 #include <memory>  

00010  

00011 #include "VEngine/Device.hpp"  

00012 #include "VEngine/Shaders.hpp"  

00013 #include "VEngine/FrameInfo.hpp"  

00014  

00015 namespace ven {  

00016  

00017     ///  

00018     ///  

00019     ///  

00020     ///  

00021     ///  

00022     class ARenderSystemBase {  

00023  

00024     public:  

00025  

00026         explicit ARenderSystemBase(Device& device) : m_device{device} {}  

00027         virtual ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(),  

00028             m_pipelineLayout, nullptr); }  


```



```
00029         virtual void render(const FrameInfo &frameInfo) const = 0;
00030
00031     protected:
00032
00033         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout, uint32_t
pushConstantSize);
00034         void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
std::string &shadersFragPath, bool isLight);
00035
00036         [[nodiscard]] Device& getDevice() const { return m_device; }
00037         [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
00038         [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }
00039
00040     private:
00041
00042         Device &m_device;
00043         VkPipelineLayout m_pipelineLayout{nullptr};
00044         std::unique_ptr<Shaders> m_shaders;
00045
00046     }; // class ARenderSystemBase
00047
00048 } // namespace ven
```

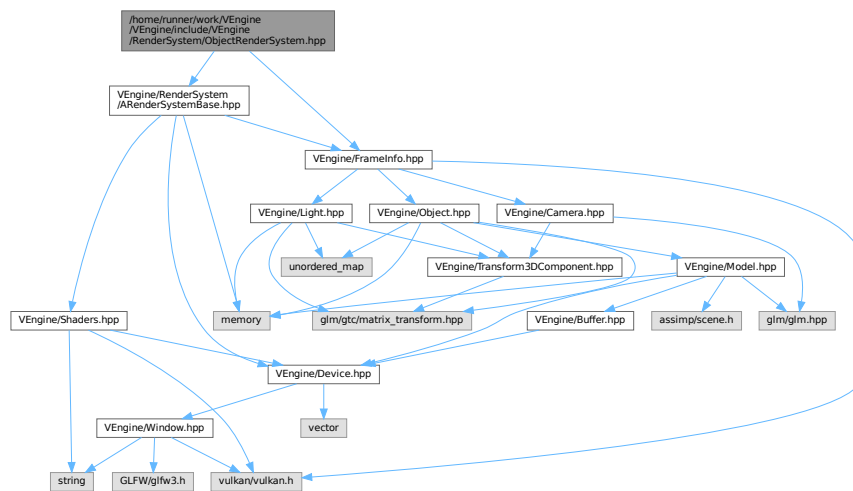
8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Render↵ System/ObjectRenderSystem.hpp File Reference

This file contains the `ObjectRenderSystem` class.

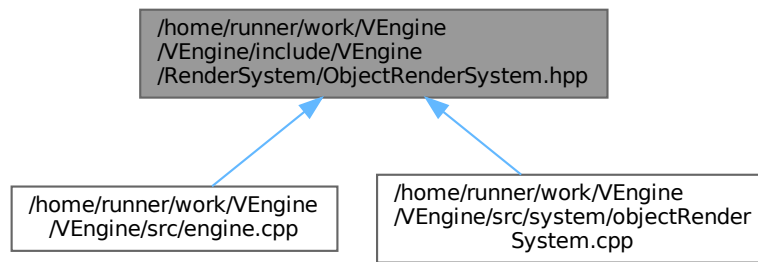
```
#include "VEngine/FrameInfo.hpp"
```

```
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
```

Include dependency graph for ObjectRenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::ObjectPushConstantData](#)
- class [ven::ObjectRenderSystem](#)
Class for object render system.

Namespaces

- namespace [ven](#)

8.41.1 Detailed Description

This file contains the ObjectRenderSystem class.

Definition in file [ObjectRenderSystem.hpp](#).

8.42 ObjectRenderSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file ObjectRenderSystem.hpp
00003 /// @brief This file contains the ObjectRenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/FrameInfo.hpp"
00010 #include "VEngine/RenderSystem/ARenderSystemBase.hpp"
00011
00012 namespace ven {
00013
00014     struct ObjectPushConstantData {
00015         glm::mat4 modelMatrix{};
00016         glm::mat4 normalMatrix{};
00017     };
00018
00019     ///
00020     /// @class ObjectRenderSystem
00021     /// @brief Class for object render system
00022     /// @namespace ven
  
```

```

00023     ///
00024     class ObjectRenderSystem final : public ARenderSystemBase {
00025     public:
00026
00027         explicit ObjectRenderSystem(Device& device, const VkRenderPass renderPass, const
00028         VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00029             createPipelineLayout(globalSetLayout, sizeof(ObjectPushConstantData));
00030             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_shader.spv",
00031             std::string(SHADERS_BIN_PATH) + "fragment_shader.spv", false);
00032         }
00033
00034         ObjectRenderSystem(const ObjectRenderSystem&) = delete;
00035         ObjectRenderSystem& operator=(const ObjectRenderSystem&) = delete;
00036
00037         void render(const FrameInfo &frameInfo) const override;
00038     }; // class ObjectRenderSystem
00039
00040 } // namespace ven

```

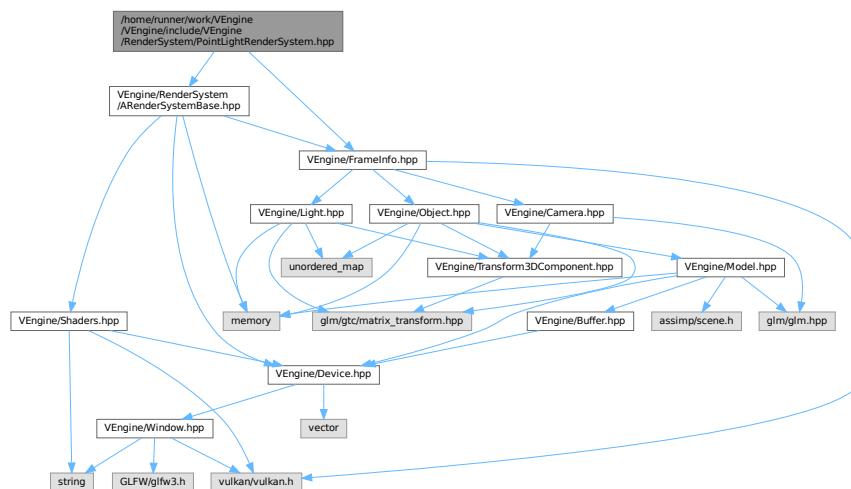
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp File Reference

This file contains the PointLightRenderSystem class.

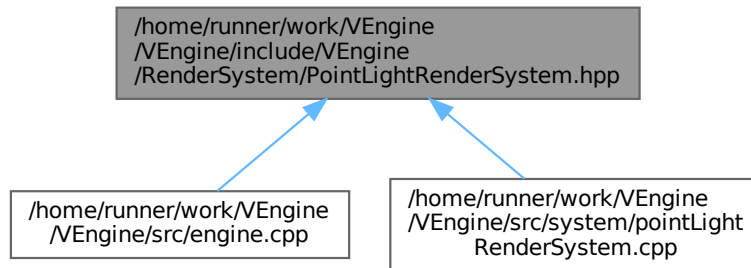
```
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
```

```
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for PointLightRenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::LightPushConstantData](#)
- class [ven::PointLightRenderSystem](#)

Class for point light system.

Namespaces

- namespace [ven](#)

8.43.1 Detailed Description

This file contains the PointLightRenderSystem class.

Definition in file [PointLightRenderSystem.hpp](#).

8.44 PointLightRenderSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file PointLightRenderSystem.hpp
00003 /// @brief This file contains the PointLightRenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/RenderSystem/ARenderSystemBase.hpp"
00010 #include "VEngine/FrameInfo.hpp"
00011
00012 namespace ven {
00013
00014     struct LightPushConstantData {
00015         glm::vec4 position{};
00016         glm::vec4 color{};
00017         float radius;
00018     };
00019
00020     ///
00021     /// @class PointLightRenderSystem
00022     /// @brief Class for point light system
  
```

```

00023     /// @namespace ven
00024     ///
00025     class PointLightRenderSystem final : public ARenderSystemBase {
00026     public:
00027
00028
00029         explicit PointLightRenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00030             createPipelineLayout(globalSetLayout, sizeof(LightPushConstantData));
00031             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_point_light.spv",
std::string(SHADERS_BIN_PATH) + "fragment_point_light.spv", true);
00032         }
00033
00034         PointLightRenderSystem(const PointLightRenderSystem&) = delete;
00035         PointLightRenderSystem& operator=(const PointLightRenderSystem&) = delete;
00036
00037         void render(const FrameInfo &frameInfo) const override;
00038
00039         static void update(const FrameInfo &frameInfo, GlobalUbo &ubo);
00040
00041     }; // class PointLightRenderSystem
00042
00043 } // namespace ven

```

8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference

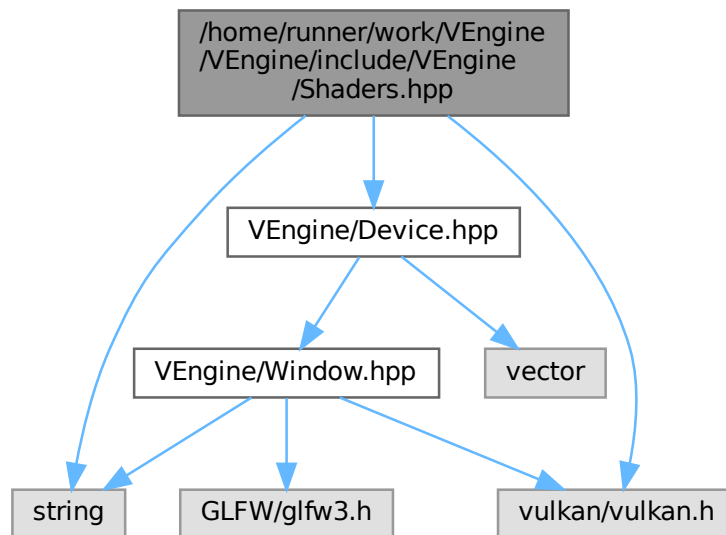
This file contains the Shader class.

```

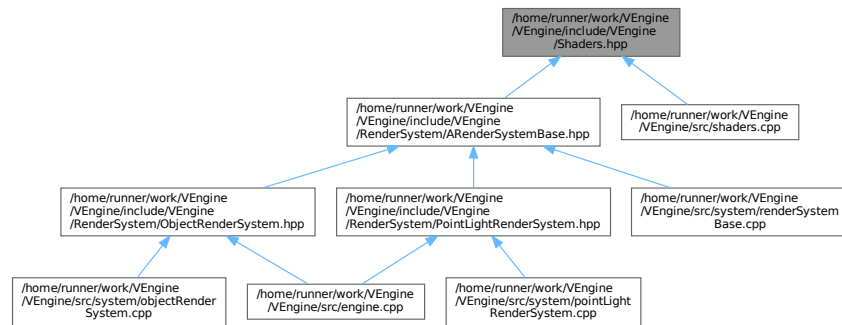
#include <string>
#include <vulkan/vulkan.h>
#include "VEngine/Device.hpp"

```

Include dependency graph for Shaders.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::PipelineConfigInfo](#)
- class [ven::Shaders](#)
Class for shaders.

Namespaces

- namespace [ven](#)

Variables

- static constexpr std::string_view [ven::SHADERS_BIN_PATH](#) = "build/shaders/"

8.45.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

8.46 Shaders.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Shaders.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 #include "VEngine/Device.hpp"
00014
00015 namespace ven {

```

```

00016
00017     static constexpr std::string_view SHADERS_BIN_PATH = "build/shaders/";
00018
00019     struct PipelineConfigInfo {
00020         PipelineConfigInfo() = default;
00021         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00022         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00023
00024         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00025         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00026         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00027         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00028         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00029         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00030         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00031         VkPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00032         std::vector<VkDynamicState> dynamicStateEnables;
00033         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00034         VkPipelineLayout pipelineLayout = nullptr;
00035         VkRenderPass renderPass = nullptr;
00036         uint32_t subpass = 0;
00037     };
00038
00039     ///
00040     /// @class Shaders
00041     /// @brief Class for shaders
00042     /// @namespace ven
00043     ///
00044     class Shaders {
00045
00046     public:
00047
00048         Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
00049 const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
00050 fragFilepath, configInfo); };
00051         ~Shaders();
00052
00053         Shaders(const Shaders&) = delete;
00054         Shaders& operator=(const Shaders&) = delete;
00055
00056         static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00057         void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
00058 VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00059
00060     private:
00061
00062         static std::vector<char> readFile(const std::string &filename);
00063         void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
00064 fragFilepath, const PipelineConfigInfo& configInfo);
00065         void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
00066 const;
00067
00068         Device& m_device;
00069         VkPipeline m_graphicsPipeline{nullptr};
00070         VkShaderModule m_vertShaderModule{nullptr};
00071         VkShaderModule m_fragShaderModule{nullptr};
00072     }; // class Shaders
00073 } // namespace ven

```

8.47 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

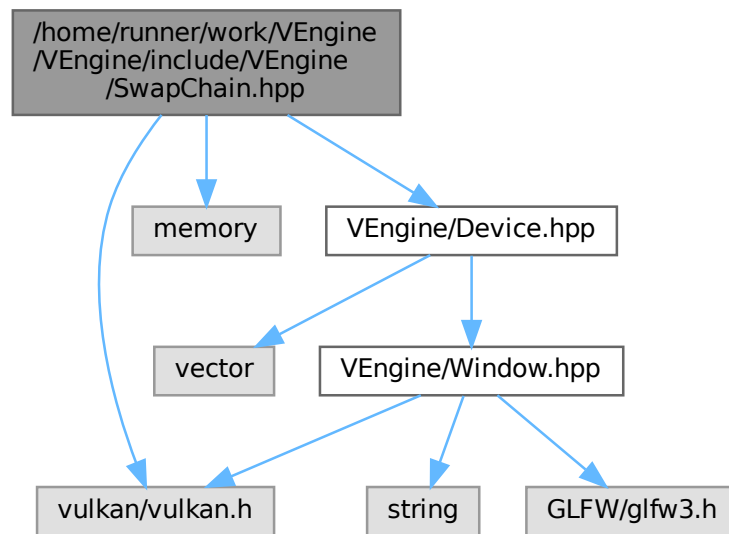
This file contains the Shader class.

```

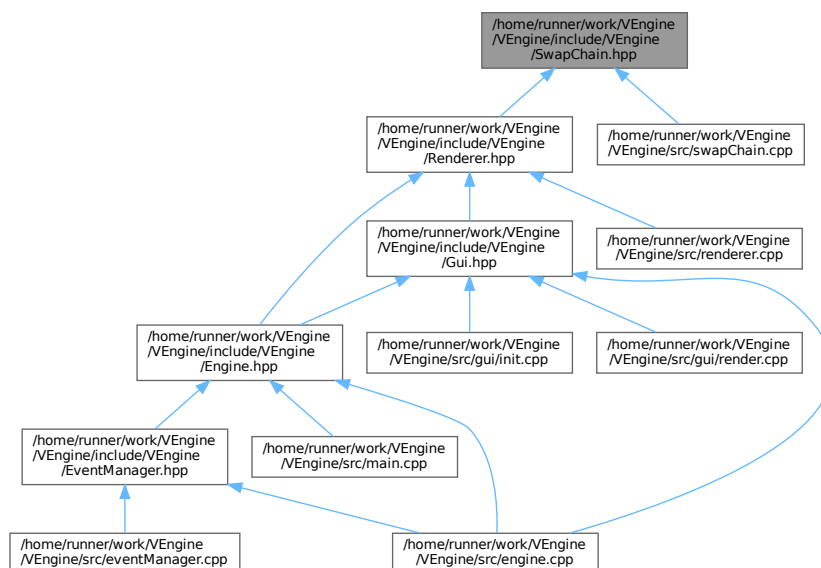
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"

```

Include dependency graph for SwapChain.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::SwapChain](#)
Class for swap chain.

Namespaces

- namespace [ven](#)

Variables

- static constexpr int [ven::MAX_FRAMES_IN_FLIGHT](#) = 2

8.47.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

8.48 SwapChain.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file SwapChain.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010 #include <memory>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00017
00018     ///
00019     /// @class SwapChain
00020     /// @brief Class for swap chain
00021     /// @namespace ven
00022     ///
00023     class SwapChain {
00024
00025     public:
00026
00027         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : m_device{deviceRef},
00028         m_windowExtent{windowExtentRef} { init(); }
00029         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
00030         previous) : m_device{deviceRef}, m_windowExtent{windowExtentRef}, m_oldSwapChain{std::move(previous)}
00031         { init(); m_oldSwapChain = nullptr; }
00032         ~SwapChain();
00033
00034         SwapChain(const SwapChain &) = delete;
00035         SwapChain& operator=(const SwapChain &) = delete;
00036
00037         [[nodiscard]] VkFramebuffer getFramebuffer(const unsigned long index) const { return
00038         m_swapChainFrameBuffers[index]; }
00039         [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }
00040         [[nodiscard]] VkImageView getImageView(const int index) const { return
00041         m_swapChainImageViews[static_cast<unsigned long>(index)]; }
00042         [[nodiscard]] size_t imageCount() const { return m_swapChainImages.size(); }
00043         [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_swapChainImageFormat; }
00044         [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00045         [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
00046         [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00047
00048         [[nodiscard]] float extentAspectRatio() const { return
00049         static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
00050         [[nodiscard]] VkFormat findDepthFormat() const;
00051
00052         VkResult acquireNextImage(uint32_t *imageIndex) const;
00053         VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00054     };
00055 }
```

```

00049         [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return
m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat ==
swapChain.m_swapChainDepthFormat; }
00050
00051     private:
00052
00053         void init();
00054         void createSwapChain();
00055         void createImageViews();
00056         void createDepthResources();
00057         void createRenderPass();
00058         void createFrameBuffers();
00059         void createSyncObjects();
00060
00061         static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats);
00062         static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes);
00063         [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
const;
00064
00065         VkFormat m_swapChainImageFormat{};
00066         VkFormat m_swapChainDepthFormat{};
00067         VkExtent2D m_swapChainExtent{};
00068
00069         std::vector<VkFramebuffer> m_swapChainFrameBuffers;
00070         VkRenderPass m_renderPass{};
00071
00072         std::vector<VkImage> m_depthImages;
00073         std::vector<VkDeviceMemory> m_depthImageMemory;
00074         std::vector<VkImageView> m_depthImageViews;
00075         std::vector<VkImage> m_swapChainImages;
00076         std::vector<VkImageView> m_swapChainImageViews;
00077
00078         Device &m_device;
00079         VkExtent2D m_windowExtent;
00080
00081         VkSwapchainKHR m_swapChain{};
00082         std::shared_ptr<SwapChain> m_oldSwapChain;
00083
00084         std::vector<VkSemaphore> m_imageAvailableSemaphores;
00085         std::vector<VkSemaphore> m_renderFinishedSemaphores;
00086         std::vector<VkFence> m_inFlightFences;
00087         std::vector<VkFence> m_imagesInFlight;
00088         size_t m_currentFrame{0};
00089
00090     }; // class SwapChain
00091
00092 } // namespace ven

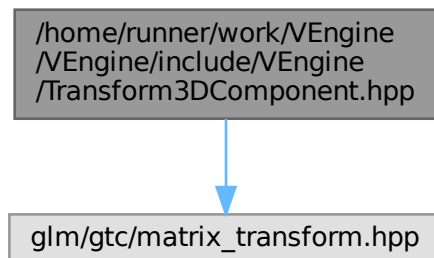
```

8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp File Reference

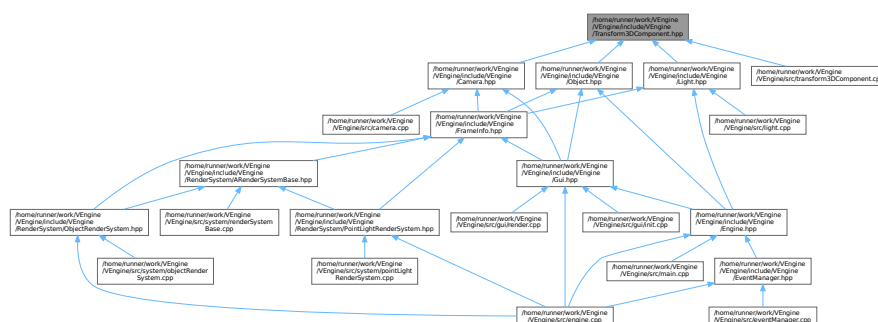
This file contains the Transform3DComponent class.

```
#include <glm/gtc/matrix_transform.hpp>
```

Include dependency graph for Transform3DComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Transform3DComponent](#)
Class for 3D transformation.

Namespaces

- namespace [ven](#)

8.49.1 Detailed Description

This file contains the Transform3DComponent class.

Definition in file [Transform3DComponent.hpp](#).

8.50 Transform3DComponent.hpp

[Go to the documentation of this file.](#)

```

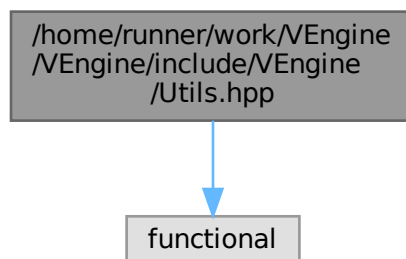
00001 ///
00002 /// @file Transform3DComponent.hpp
00003 /// @brief This file contains the Transform3DComponent class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <glm/gtc/matrix_transform.hpp>
00010
00011 namespace ven {
00012
00013     ///
00014     /// @class Transform3DComponent
00015     /// @brief Class for 3D transformation
00016     /// @namespace ven
00017     ///
00018     class Transform3DComponent {
00019
00020     public:
00021
00022         glm::vec3 translation{};
00023         glm::vec3 scale{1.F, 1.F, 1.F};
00024         glm::vec3 rotation{};
00025
00026         [[nodiscard]] glm::mat4 mat4() const;
00027         [[nodiscard]] glm::mat3 normalMatrix() const;
00028
00029     }; // class Transform3DComponent
00030
00031 } // namespace ven

```

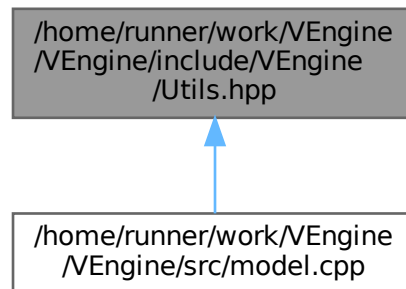
8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

```
#include <functional>
```

Include dependency graph for Utils.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `ven`

Functions

- `template<typename T, typename... Rest>`
`void ven::hashCombine (std::size_t &seed, const T &v, const Rest &... rest)`

8.52 Utils.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Utils.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018
00019 } // namespace ven
  
```

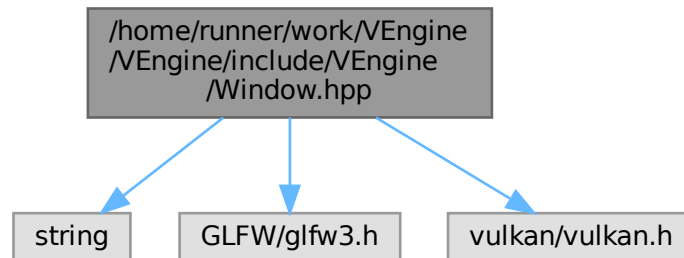
8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp

File Reference

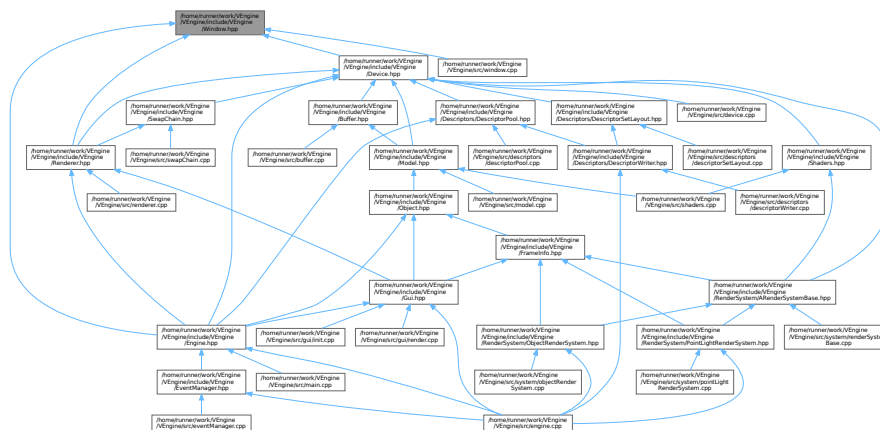
This file contains the Window class.

```
#include <string>
#include <GLFW/glfw3.h>
#include <vulkan/vulkan.h>
```

Include dependency graph for Window.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Window`
Class for window.

Namespaces

- namespace `ven`

Macros

- `#define` `GLFW_INCLUDE_VULKAN`

Variables

- static constexpr uint32_t [ven::DEFAULT_WIDTH](#) = 1920
- static constexpr uint32_t [ven::DEFAULT_HEIGHT](#) = 1080
- static constexpr std::string_view [ven::DEFAULT_TITLE](#) = "VEngine"

8.53.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

8.53.2 Macro Definition Documentation

8.53.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

8.54 Window.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00018     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00019     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00020
00021     ///
00022     /// @class Window
00023     /// @brief Class for window
00024     /// @namespace ven
00025     ///
00026     class Window {
00027     public:
00028
00029         explicit Window(const uint32_t width = DEFAULT_WIDTH, const uint32_t height =
00030             DEFAULT_HEIGHT, const std::string &title = DEFAULT_TITLE.data()) : m_window(createWindow(width,
00031             height, title)), m_width(width), m_height(height) {}
00032
00033         ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr; }
00034
00035         Window(const Window&) = delete;
00036         Window& operator=(const Window&) = delete;
00037
00038         [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
00039             &title);
00040         void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00041
00042         [[nodiscard]] GLFWwindow* getGLFWWindow() const { return m_window; }
```

```

00041     [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; }
00042     [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00043     void resetWindowResizedFlag() { m_framebufferResized = false; }
00044
00045     void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);
00046
00047     private:
00048
00049         static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00050
00051         GLFWwindow* m_window{nullptr};
00052         uint32_t m_width{DEFAULT_WIDTH};
00053         uint32_t m_height{DEFAULT_HEIGHT};
00054
00055         bool m_framebufferResized = false;
00056
00057     }; // class Window
00058
00059 } // namespace ven

```

8.55 /home/runner/work/VEngine/VEngine/README.md File Reference

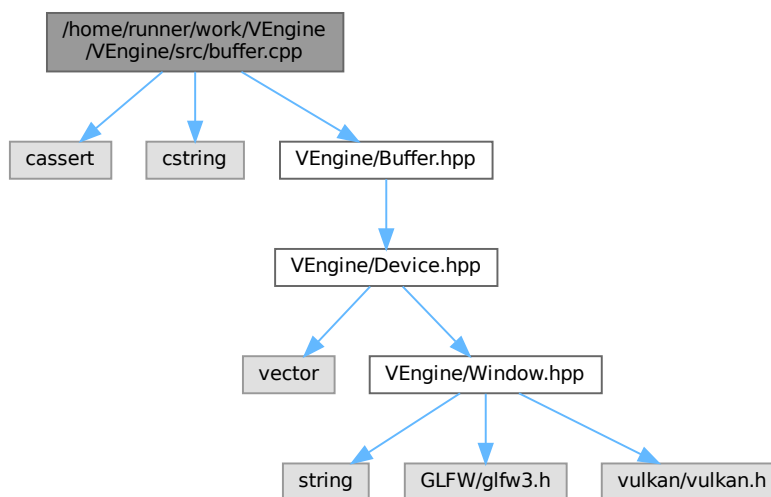
8.56 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```

#include <cassert>
#include <cstring>
#include "VEngine/Buffer.hpp"

```

Include dependency graph for buffer.cpp:



8.57 buffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <cstring>
00003
00004 #include "VEngine/Buffer.hpp"
00005

```



```

00006 VkDeviceSize ven::Buffer::getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
    minOffsetAlignment) {
00007     if (minOffsetAlignment > 0) {
00008         return (instanceSize + minOffsetAlignment - 1) & ~(minOffsetAlignment - 1);
00009     }
00010     return instanceSize;
00011 }
00012
00013 ven::Buffer::Buffer(Device &device, const VkDeviceSize instanceSize, const uint32_t instanceCount,
    const VkBufferUsageFlags usageFlags, const VkMemoryPropertyFlags memoryPropertyFlags, const
    VkDeviceSize minOffsetAlignment) : m_device{device}, m_instanceSize{instanceSize},
    m_instanceCount{instanceCount}, m_alignmentSize{getAlignment(instanceSize, minOffsetAlignment)},
    m_usageFlags{usageFlags}, m_memoryPropertyFlags{memoryPropertyFlags}
00014 {
00015     m_bufferSize = m_alignmentSize * m_instanceCount;
00016     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00017 }
00018
00019 ven::Buffer::~Buffer()
00020 {
00021     unmap();
00022     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00023     vkFreeMemory(m_device.device(), m_memory, nullptr);
00024 }
00025
00026 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00027 {
00028     assert(m_buffer && m_memory && "Called map on m_buffer before create");
00029     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00030 }
00031
00032 void ven::Buffer::unmap()
00033 {
00034     if (m_mapped != nullptr) {
00035         vkUnmapMemory(m_device.device(), m_memory);
00036         m_mapped = nullptr;
00037     }
00038 }
00039
00040 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
    const
00041 {
00042     assert(m_mapped && "Cannot copy to unmapped m_buffer");
00043
00044     if (size == VK_WHOLE_SIZE) {
00045         memcpy(m_mapped, data, m_bufferSize);
00046     } else {
00047         char *memOffset = static_cast<char *>(m_mapped);
00048         memOffset += offset;
00049         memcpy(memOffset, data, size);
00050     }
00051 }
00052
00053 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00054 {
00055     VkMappedMemoryRange mappedRange = {};
00056     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00057     mappedRange.memory = m_memory;
00058     mappedRange.offset = offset;
00059     mappedRange.size = size;
00060     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00061 }
00062
00063 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00064 {
00065     VkMappedMemoryRange mappedRange = {};
00066     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00067     mappedRange.memory = m_memory;
00068     mappedRange.offset = offset;
00069     mappedRange.size = size;
00070     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00071 }

```

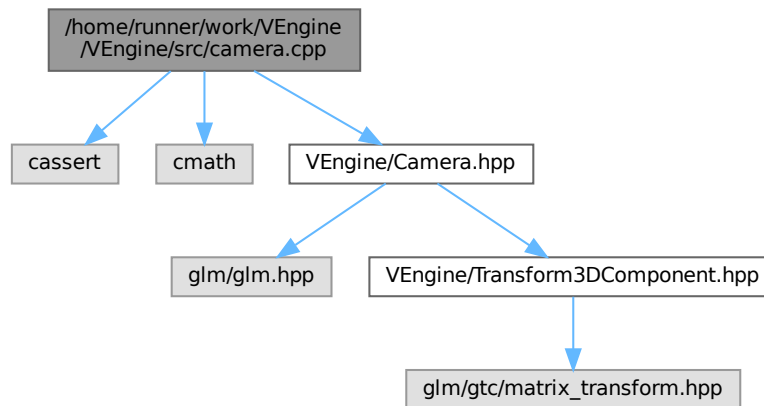
8.58 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```

#include <cassert>
#include <cmath>

```

```
#include "VEngine/Camera.hpp"
Include dependency graph for camera.cpp:
```



8.59 camera.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <cmath>
00003
00004 #include "VEngine/Camera.hpp"
00005
00006 void ven::Camera::setOrthographicProjection(const float left, const float right, const float top,
00007 const float bottom, const float near, const float far)
00008 {
00009     m_projectionMatrix = glm::mat4{1.0F};
00010     m_projectionMatrix[0][0] = 2.F / (right - left);
00011     m_projectionMatrix[1][1] = 2.F / (bottom - top);
00012     m_projectionMatrix[2][2] = 1.F / (far - near);
00013     m_projectionMatrix[3][0] = -(right + left) / (right - left);
00014     m_projectionMatrix[3][1] = -(bottom + top) / (bottom - top);
00015     m_projectionMatrix[3][2] = -near / (far - near);
00016 }
00017 void ven::Camera::setPerspectiveProjection(const float aspect)
00018 {
00019     assert(glm::abs(aspect - std::numeric_limits<float>::epsilon()) > 0.0F);
00020     const float tanHalfFov = std::tan(m_fov / 2.F);
00021     m_projectionMatrix = glm::mat4{0.0F};
00022     m_projectionMatrix[0][0] = 1.F / (aspect * tanHalfFov);
00023     m_projectionMatrix[1][1] = 1.F / (tanHalfFov);
00024     m_projectionMatrix[2][2] = m_far / (m_far - m_near);
00025     m_projectionMatrix[2][3] = 1.F;
00026     m_projectionMatrix[3][2] = -(m_far * m_near) / (m_far - m_near);
00027 }
00028
00029 void ven::Camera::setViewDirection(const glm::vec3 position, const glm::vec3 direction, const
00030 glm::vec3 up)
00031 {
00032     const glm::vec3 w{normalize(direction)};
00033     const glm::vec3 u{normalize(cross(w, up))};
00034     const glm::vec3 v{cross(w, u)};
00035
00036     m_viewMatrix = glm::mat4{1.F};
00037     m_viewMatrix[0][0] = u.x;
00038     m_viewMatrix[1][0] = u.y;
00039     m_viewMatrix[2][0] = u.z;
00040     m_viewMatrix[0][1] = v.x;
00041     m_viewMatrix[1][1] = v.y;
00042     m_viewMatrix[2][1] = v.z;
00043     m_viewMatrix[0][2] = w.x;
00044     m_viewMatrix[1][2] = w.y;
00045     m_viewMatrix[2][2] = w.z;
00046 }
```

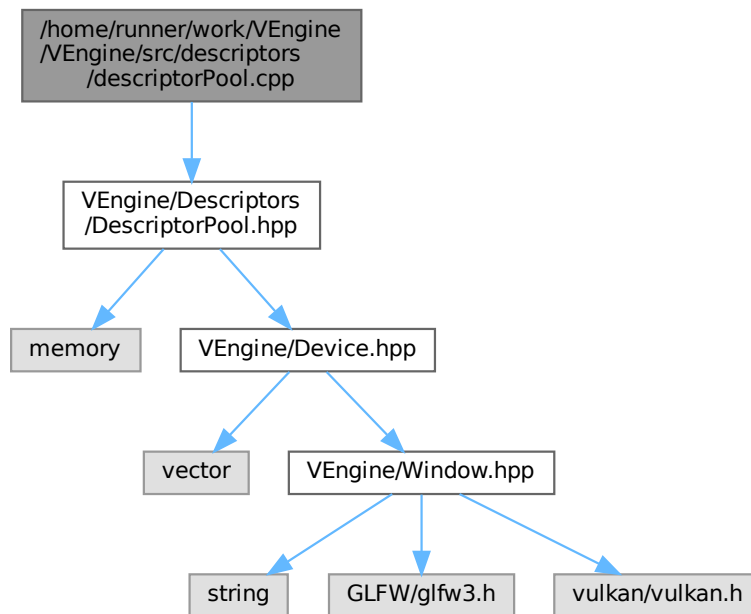
```

00044     m_viewMatrix[2][2] = w.z;
00045     m_viewMatrix[3][0] = -dot(u, position);
00046     m_viewMatrix[3][1] = -dot(v, position);
00047     m_viewMatrix[3][2] = -dot(w, position);
00048
00049     m_inverseViewMatrix = glm::mat4(1.F);
00050     m_inverseViewMatrix[0][0] = u.x;
00051     m_inverseViewMatrix[0][1] = u.y;
00052     m_inverseViewMatrix[0][2] = u.z;
00053     m_inverseViewMatrix[1][0] = v.x;
00054     m_inverseViewMatrix[1][1] = v.y;
00055     m_inverseViewMatrix[1][2] = v.z;
00056     m_inverseViewMatrix[2][0] = w.x;
00057     m_inverseViewMatrix[2][1] = w.y;
00058     m_inverseViewMatrix[2][2] = w.z;
00059     m_inverseViewMatrix[3][0] = position.x;
00060     m_inverseViewMatrix[3][1] = position.y;
00061     m_inverseViewMatrix[3][2] = position.z;
00062 }
00063
00064 void ven::Camera::setViewXYZ(const glm::vec3 position, const glm::vec3 rotation)
00065 {
00066     const float c3 = glm::cos(rotation.z);
00067     const float s3 = glm::sin(rotation.z);
00068     const float c2 = glm::cos(rotation.x);
00069     const float s2 = glm::sin(rotation.x);
00070     const float c1 = glm::cos(rotation.y);
00071     const float s1 = glm::sin(rotation.y);
00072     const glm::vec3 u{(c1 * c3 + s1 * s2 * s3), (c2 * s3), (c1 * s2 * s3 - c3 * s1)};
00073     const glm::vec3 v{(c3 * s1 * s2 - c1 * s3), (c2 * c3), (c1 * c3 * s2 + s1 * s3)};
00074     const glm::vec3 w{(c2 * s1), (-s2), (c1 * c2)};
00075     m_viewMatrix = glm::mat4(1.F);
00076     m_viewMatrix[0][0] = u.x;
00077     m_viewMatrix[1][0] = u.y;
00078     m_viewMatrix[2][0] = u.z;
00079     m_viewMatrix[0][1] = v.x;
00080     m_viewMatrix[1][1] = v.y;
00081     m_viewMatrix[2][1] = v.z;
00082     m_viewMatrix[0][2] = w.x;
00083     m_viewMatrix[1][2] = w.y;
00084     m_viewMatrix[2][2] = w.z;
00085     m_viewMatrix[3][0] = -dot(u, position);
00086     m_viewMatrix[3][1] = -dot(v, position);
00087     m_viewMatrix[3][2] = -dot(w, position);
00088
00089     m_inverseViewMatrix = glm::mat4(1.F);
00090     m_inverseViewMatrix[0][0] = u.x;
00091     m_inverseViewMatrix[0][1] = u.y;
00092     m_inverseViewMatrix[0][2] = u.z;
00093     m_inverseViewMatrix[1][0] = v.x;
00094     m_inverseViewMatrix[1][1] = v.y;
00095     m_inverseViewMatrix[1][2] = v.z;
00096     m_inverseViewMatrix[2][0] = w.x;
00097     m_inverseViewMatrix[2][1] = w.y;
00098     m_inverseViewMatrix[2][2] = w.z;
00099     m_inverseViewMatrix[3][0] = position.x;
00100     m_inverseViewMatrix[3][1] = position.y;
00101     m_inverseViewMatrix[3][2] = position.z;
00102 }

```

8.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference

#include "VEngine/Descriptors/DescriptorPool.hpp"
 Include dependency graph for descriptorPool.cpp:



8.61 descriptorPool.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Descriptors/DescriptorPool.hpp"
00002
00003 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device(device)
00004 {
00005     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00006     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00007     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00008     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00009     descriptorPoolInfo.maxSets = maxSets;
00010     descriptorPoolInfo.flags = poolFlags;
00011
00012     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) !=
00013         VK_SUCCESS) {
00014         throw std::runtime_error("failed to create descriptor pool!");
00015     }
00016 }
00017
00018 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorSetLayout,
    VkDescriptorSet &descriptor) const
00019 {
00020     VkDescriptorSetAllocateInfo allocInfo{};
00021     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00022     allocInfo.descriptorPool = m_descriptorPool;
00023     allocInfo.pSetLayouts = &descriptorSetLayout;
00024     allocInfo.descriptorSetCount = 1;
00025

```

```

00026 // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00027 // a new pool whenever an old pool fills up. But this is beyond our current scope
00028 return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00029 }

```

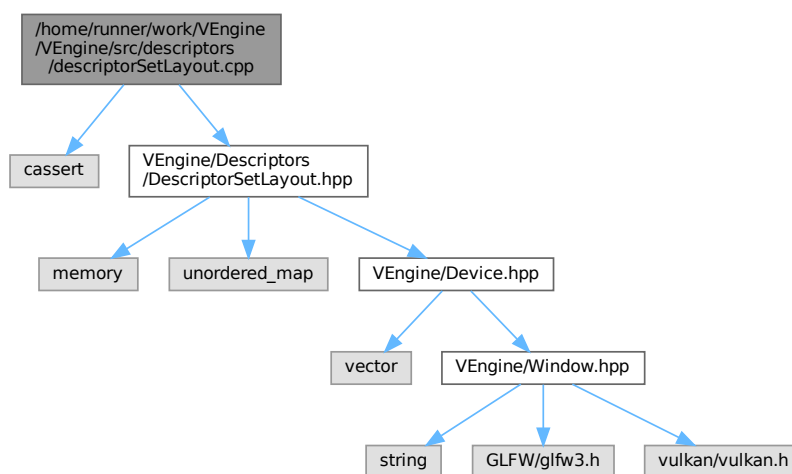
8.62 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

Include dependency graph for descriptorSetLayout.cpp:



8.63 descriptorSetLayout.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Descriptors/DescriptorSetLayout.hpp"
00004
00005 ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t
binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t
count)
00006 {
00007     assert(m_bindings.contains(binding) == 0 && "Binding already exists in layout");
00008     VkDescriptorSetLayoutBinding layoutBinding{};
00009     layoutBinding.binding = binding;
00010     layoutBinding.descriptorType = descriptorType;
00011     layoutBinding.descriptorCount = count;
00012     layoutBinding.stageFlags = stageFlags;
00013     m_bindings[binding] = layoutBinding;
00014     return *this;
00015 }
00016
00017 ven::DescriptorSetLayout::DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
00018 {
00019     std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
00020     setLayoutBindings.reserve(bindings.size());
00021     for (auto [fst, snd] : bindings) {
00022         setLayoutBindings.push_back(snd);
00023     }
00024 }

```

```

00025     VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
00026     descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
00027     descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
00028     descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();
00029
00030     if (vkCreateDescriptorSetLayout (
00031         m_device.device(),
00032         &descriptorSetLayoutInfo,
00033         nullptr,
00034         &m_descriptorSetLayout) != VK_SUCCESS) {
00035         throw std::runtime_error("failed to create descriptor set layout!");
00036     }
00037 }

```

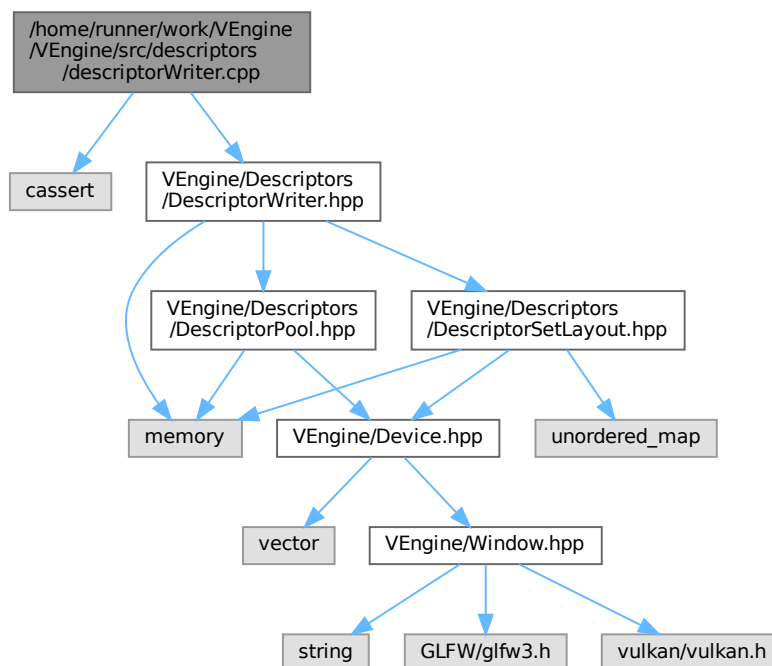
8.64 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorWriter.hpp"

```

Include dependency graph for descriptorWriter.cpp:



8.65 descriptorWriter.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Descriptors/DescriptorWriter.hpp"
00004
00005 ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const
    VkDescriptorBufferInfo *bufferInfo)
00006 {

```

```

00007     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00008
00009     const auto &bindingDescription = m_setLayout.m_bindings.at(binding);
00010
00011     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00012
00013     VkWriteDescriptorSet write{};
00014     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00015     write.descriptorType = bindingDescription.descriptorType;
00016     write.dstBinding = binding;
00017     write.pBufferInfo = bufferInfo;
00018     write.descriptorCount = 1;
00019
00020     m_writes.push_back(write);
00021     return *this;
00022 }
00023
00024 ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const
VkDescriptorImageInfo *imageInfo)
00025 {
00026     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00027
00028     const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings.at(binding);
00029
00030     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00031
00032     VkWriteDescriptorSet write{};
00033     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00034     write.descriptorType = bindingDescription.descriptorType;
00035     write.dstBinding = binding;
00036     write.pImageInfo = imageInfo;
00037     write.descriptorCount = 1;
00038
00039     m_writes.push_back(write);
00040     return *this;
00041 }
00042
00043 bool ven::DescriptorWriter::build(VkDescriptorSet &set)
00044 {
00045     if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
00046         return false;
00047     }
00048     overwrite(set);
00049     return true;
00050 }
00051
00052 void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set) {
00053     for (auto &[sType, pNext, dstSet, dstBinding, dstArrayElement, descriptorCount, descriptorType,
pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
00054         dstSet = set;
00055     }
00056     vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
m_writes.data(), 0, nullptr);
00057 }

```

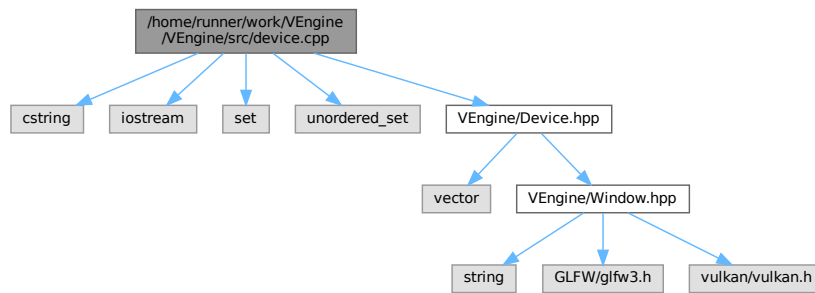
8.66 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

```

#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>
#include "VEngine/Device.hpp"

```

Include dependency graph for device.cpp:



Functions

- static VKAPI_ATTR VkBool32 VKAPI_CALL [debugCallback](#) (const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
- VkResult [CreateDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator, VkDebugUtilsMessengerEXT *pDebugMessenger)
- void [DestroyDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks *pAllocator)

8.66.1 Function Documentation

8.66.1.1 CreateDebugUtilsMessengerEXT()

```

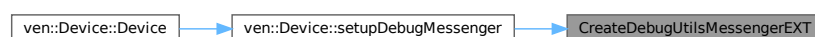
VkResult CreateDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerCreateInfoEXT * pCreateInfo,
    const VkAllocationCallbacks * pAllocator,
    VkDebugUtilsMessengerEXT * pDebugMessenger)

```

Definition at line 16 of file [device.cpp](#).

Referenced by [ven::Device::setupDebugMessenger\(\)](#).

Here is the caller graph for this function:



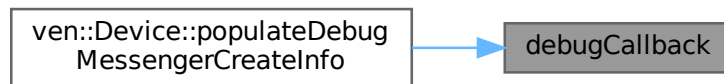
8.66.1.2 debugCallback()

```
static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (
    const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity,
    const VkDebugUtilsMessageTypeFlagsEXT messageType,
    const VkDebugUtilsMessengerCallbackDataEXT * pCallbackData,
    void * pUserData) [static]
```

Definition at line 8 of file [device.cpp](#).

Referenced by [ven::Device::populateDebugMessengerCreateInfo\(\)](#).

Here is the caller graph for this function:



8.66.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerEXT debugMessenger,
    const VkAllocationCallbacks * pAllocator)
```

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

Referenced by [ven::Device::~~Device\(\)](#).

Here is the caller graph for this function:



8.67 device.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cstring>
00002 #include <iostream>
00003 #include <set>
00004 #include <unordered_set>
00005
00006 #include "VEngine/Device.hpp"
00007
00008 static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT
messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const
VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
00009 {
00010     (void) pUserData; (void) messageSeverity; (void) messageType;
00011
00012     std::cerr << "validation layer: " << pCallbackData->pMessage << '\n';
00013     return VK_FALSE;
00014 }
00015
00016 VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const
VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator,
VkDebugUtilsMessengerEXT *pDebugMessenger)
00017 {
00018     if (const auto func =
reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkCreateDebugUtilsMessengerEXT")); func != nullptr) {
00019         return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00020     }
00021
00022     return VK_ERROR_EXTENSION_NOT_PRESENT;
00023 }
00024
00025 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
debugMessenger, const VkAllocationCallbacks *pAllocator)
00026 {
00027     if (const auto func =
reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkDestroyDebugUtilsMessengerEXT")); func != nullptr) {
00028         func(instance, debugMessenger, pAllocator);
00029     }
00030 }
00031
00032 ven::Device::Device(Window &window) : m_window{window}
00033 {
00034     createInstance();
00035     setupDebugMessenger();
00036     createSurface();
00037     pickPhysicalDevice();
00038     createLogicalDevice();
00039     createCommandPool();
00040 }
00041
00042 ven::Device::~Device()
00043 {
00044     vkDestroyCommandPool(m_device, m_commandPool, nullptr);
00045     vkDestroyDevice(m_device, nullptr);
00046
00047     if (enableValidationLayers) {
00048         DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
00049     }
00050
00051     vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
00052     vkDestroyInstance(m_instance, nullptr);
00053 }
00054
00055 void ven::Device::createInstance()
00056 {
00057     if (enableValidationLayers && !checkValidationLayerSupport()) {
00058         throw std::runtime_error("validation layers requested, but not available!");
00059     }
00060
00061     VkApplicationInfo appInfo = {};
00062     appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
00063     appInfo.pApplicationName = "LittleVulkanEngine App";
00064     appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
00065     appInfo.pEngineName = "No Engine";
00066     appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
00067     appInfo.apiVersion = VK_API_VERSION_1_0;
00068
00069     VkInstanceCreateInfo createInfo = {};
00070     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00071     createInfo.pApplicationInfo = &appInfo;
00072
00073     const std::vector<const char *> extensions = getRequiredExtensions();

```

```

00074     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00075     createInfo.ppEnabledExtensionNames = extensions.data();
00076
00077     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00078     if (enableValidationLayers) {
00079         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00080         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00081
00082         populateDebugMessengerCreateInfo(debugCreateInfo);
00083         createInfo.pNext = &debugCreateInfo;
00084     } else {
00085         createInfo.enabledLayerCount = 0;
00086         createInfo.pNext = nullptr;
00087     }
00088
00089     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
00090         throw std::runtime_error("failed to create instance!");
00091     }
00092
00093     hasGlfwRequiredInstanceExtensions();
00094 }
00095
00096 void ven::Device::pickPhysicalDevice()
00097 {
00098     uint32_t deviceCount = 0;
00099     vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
00100     if (deviceCount == 0) {
00101         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00102     }
00103     std::cout << "Device count: " << deviceCount << '\n';
00104     std::vector<VkPhysicalDevice> devices(deviceCount);
00105     vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());
00106
00107     for (const auto &device : devices) {
00108         if (isDeviceSuitable(device)) {
00109             m_physicalDevice = device;
00110             break;
00111         }
00112     }
00113
00114     if (m_physicalDevice == VK_NULL_HANDLE) {
00115         throw std::runtime_error("failed to find a suitable GPU!");
00116     }
00117
00118     vkGetPhysicalDeviceProperties(m_physicalDevice, &m_properties);
00119     std::cout << "physical device: " << m_properties.deviceName << '\n';
00120 }
00121
00122 void ven::Device::createLogicalDevice()
00123 {
00124     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
        findQueueFamilies(m_physicalDevice);
00125
00126     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00127     const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, presentFamily};
00128     float queuePriority = 1.0F;
00129
00130     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00131         VkDeviceQueueCreateInfo queueCreateInfo = {};
00132         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00133         queueCreateInfo.queueFamilyIndex = queueFamily;
00134         queueCreateInfo.queueCount = 1;
00135         queueCreateInfo.pQueuePriorities = &queuePriority;
00136         queueCreateInfos.push_back(queueCreateInfo);
00137     }
00138
00139     VkPhysicalDeviceFeatures deviceFeatures = {};
00140     deviceFeatures.samplerAnisotropy = VK_TRUE;
00141
00142     VkDeviceCreateInfo createInfo = {};
00143     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00144
00145     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00146     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00147
00148     createInfo.pEnabledFeatures = &deviceFeatures;
00149     createInfo.enabledExtensionCount = static_cast<uint32_t>(m_deviceExtensions.size());
00150     createInfo.ppEnabledExtensionNames = m_deviceExtensions.data();
00151
00152     // might not really be necessary anymore because device specific validation layers
00153     // have been deprecated
00154     if (enableValidationLayers) {
00155         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00156         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00157     } else {
00158         createInfo.enabledLayerCount = 0;
00159     }

```

```

00160
00161     if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
00162         throw std::runtime_error("failed to create logical device!");
00163     }
00164
00165     vkGetDeviceQueue(m_device, graphicsFamily, 0, &m_graphicsQueue);
00166     vkGetDeviceQueue(m_device, presentFamily, 0, &m_presentQueue);
00167 }
00168
00169 void ven::Device::createCommandPool()
00170 {
00171     const QueueFamilyIndices queueFamilyIndices = findPhysicalQueueFamilies();
00172
00173     VkCommandPoolCreateInfo poolInfo = {};
00174     poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
00175     poolInfo.queueFamilyIndex = queueFamilyIndices.graphicsFamily;
00176     poolInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
00177
00178     if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
00179         throw std::runtime_error("failed to create command pool!");
00180     }
00181 }
00182
00183 bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00184 {
00185     const QueueFamilyIndices indices = findQueueFamilies(device);
00186     const bool extensionsSupported = checkDeviceExtensionSupport(device);
00187     bool swapChainAdequate = false;
00188
00189     if (extensionsSupported) {
00190         auto [capabilities, formats, presentModes] = querySwapChainSupport(device);
00191         swapChainAdequate = !formats.empty() && !presentModes.empty();
00192     }
00193
00194     VkPhysicalDeviceFeatures supportedFeatures;
00195     vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00196
00197     return indices.isComplete() && extensionsSupported && swapChainAdequate &&
(supportedFeatures.samplerAnisotropy != 0U);
00198 }
00199
00200 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00201 {
00202     createInfo = {};
00203     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00204     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00205     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00206     createInfo.pfnUserCallback = debugCallback;
00207     createInfo.pUserData = nullptr; // Optional
00208 }
00209
00210 void ven::Device::setupDebugMessenger()
00211 {
00212     if (!enableValidationLayers) { return; }
00213     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00214     populateDebugMessengerCreateInfo(createInfo);
00215     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
VK_SUCCESS) {
00216         throw std::runtime_error("failed to set up debug messenger!");
00217     }
00218 }
00219
00220 bool ven::Device::checkValidationLayerSupport() const
00221 {
00222     uint32_t layerCount = 0;
00223     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00224
00225     std::vector<VkLayerProperties> availableLayers(layerCount);
00226     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00227
00228     for (const char *validationLayer : m_validationLayers) {
00229         bool layerFound = false;
00230
00231         for (const auto &[layerName, specVersion, implementationVersion, description] :
availableLayers) {
00232             if (strcmp(layerName, validationLayer) == 0) {
00233                 layerFound = true;
00234                 break;
00235             }
00236         }
00237         if (!layerFound) {
00238             return false;
00239         }
00240     }
00241 }
00242

```

```

00243     }
00244
00245     return true;
00246 }
00247
00248 std::vector<const char *> ven::Device::getRequiredExtensions() const
00249 {
00250     uint32_t glfwExtensionCount = 0;
00251     const char **glfwExtensions = nullptr;
00252     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00253
00254     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00255
00256     if (enableValidationLayers) {
00257         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00258     }
00259
00260     return extensions;
00261 }
00262
00263 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00264 {
00265     uint32_t extensionCount = 0;
00266     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
00267     std::vector<VkExtensionProperties> extensions(extensionCount);
00268     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00269
00270     std::cout << "available extensions:\n";
00271     std::unordered_set<std::string> available;
00272     for (const auto &[extensionName, specVersion] : extensions) {
00273         std::cout << '\t' << extensionName << '\n';
00274         available.insert(extensionName);
00275     }
00276
00277     std::cout << "required extensions:\n";
00278     for (const std::vector<const char *> requiredExtensions = getRequiredExtensions(); const auto
&required : requiredExtensions) {
00279         std::cout << "\t" << required << '\n';
00280         if (!available.contains(required)) {
00281             throw std::runtime_error("Missing required glfw extension");
00282         }
00283     }
00284 }
00285
00286 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00287 {
00288     uint32_t extensionCount = 0;
00289     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00290
00291     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00292     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
availableExtensions.data());
00293
00294     std::set<std::string> requiredExtensions(m_deviceExtensions.begin(), m_deviceExtensions.end());
00295     for (const auto &[extensionName, specVersion] : availableExtensions) {
00296         requiredExtensions.erase(extensionName);
00297     }
00298
00299     return requiredExtensions.empty();
00300 }
00301
00302 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00303 {
00304     QueueFamilyIndices indices;
00305
00306     uint32_t queueFamilyCount = 0;
00307     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00308     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00309     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00310     uint32_t index = 0;
00311
00312     for (const auto &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
queueFamilies) {
00313         if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00314             indices.graphicsFamily = index;
00315             indices.graphicsFamilyHasValue = true;
00316         }
00317         VkBool32 presentSupport = 0U;
00318         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00319         if (queueCount > 0 && (presentSupport != 0U)) {
00320             indices.presentFamily = index;
00321             indices.presentFamilyHasValue = true;
00322         }
00323         if (indices.isComplete()) {
00324             break;
00325         }
00326         index++;

```

```

00327     }
00328     return indices;
00329 }
00330
00331 ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00332 {
00333     SwapChainSupportDetails details;
00334     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
00335     uint32_t formatCount = 0;
00336
00337     vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
00338     if (formatCount != 0) {
00339         details.formats.resize(formatCount);
00340         vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
00341     }
00342     uint32_t presentModeCount = 0;
00343     vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
00344     if (presentModeCount != 0) {
00345         details.presentModes.resize(presentModeCount);
00346         vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount,
00347             details.presentModes.data());
00348     }
00349     return details;
00350 }
00351
00352 VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
00353     tiling, const VkFormatFeatureFlags features) const
00354 {
00355     for (const VkFormat format : candidates) {
00356         VkFormatProperties props;
00357         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00358         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00359             return format;
00360         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
00361             features) {
00362             return format;
00363         }
00364     }
00365     throw std::runtime_error("failed to find supported format!");
00366 }
00367
00368 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
00369     properties) const
00370 {
00371     VkPhysicalDeviceMemoryProperties memProperties;
00372     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
00373
00374     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00375         if (((typeFilter & (1 << i)) != 0U) &&
00376             (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00377             return i;
00378         }
00379     }
00380     throw std::runtime_error("failed to find suitable m_memory type!");
00381 }
00382
00383 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
00384     VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00385 {
00386     VkBufferCreateInfo bufferInfo{};
00387     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00388     bufferInfo.size = size;
00389     bufferInfo.usage = usage;
00390     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00391
00392     if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00393         throw std::runtime_error("failed to create vertex m_buffer!");
00394     }
00395
00396     VkMemoryRequirements memRequirements;
00397     vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);
00398
00399     VkMemoryAllocateInfo allocInfo{};
00400     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00401     allocInfo.allocationSize = memRequirements.size;
00402     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00403
00404     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00405         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00406     }
00407
00408     vkBindBufferMemory(m_device, buffer, bufferMemory, 0);
00409 }
00410
00411 VkCommandBuffer ven::Device::beginSingleTimeCommands() const

```

```

00409 {
00410     VkCommandBufferAllocateInfo allocInfo{};
00411     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00412     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00413     allocInfo.commandPool = m_commandPool;
00414     allocInfo.commandBufferCount = 1;
00415
00416     VkCommandBuffer commandBuffer = nullptr;
00417     vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);
00418
00419     VkCommandBufferBeginInfo beginInfo{};
00420     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00421     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00422
00423     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00424     return commandBuffer;
00425 }
00426
00427 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00428 {
00429     vkEndCommandBuffer(commandBuffer);
00430
00431     VkSubmitInfo submitInfo{};
00432     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00433     submitInfo.commandBufferCount = 1;
00434     submitInfo.pCommandBuffers = &commandBuffer;
00435
00436     vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
00437     vkQueueWaitIdle(m_graphicsQueue);
00438
00439     vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
00440 }
00441
00442 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
size) const
00443 {
00444     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00445
00446     VkBufferCopy copyRegion{};
00447     copyRegion.srcOffset = 0; // Optional
00448     copyRegion.dstOffset = 0; // Optional
00449     copyRegion.size = size;
00450     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00451
00452     endSingleTimeCommands(commandBuffer);
00453 }
00454
00455 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
const uint32_t height, const uint32_t layerCount) const
00456 {
00457     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00458
00459     VkBufferImageCopy region{};
00460     region.bufferOffset = 0;
00461     region.bufferRowLength = 0;
00462     region.bufferImageHeight = 0;
00463
00464     region.imageSubresource.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00465     region.imageSubresource.mipLevel = 0;
00466     region.imageSubresource.baseArrayLayer = 0;
00467     region.imageSubresource.layerCount = layerCount;
00468
00469     region.imageOffset = {.x=0, .y=0, .z=0};
00470     region.imageExtent = {.width=width, .height=height, .depth=1};
00471
00472     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
&region);
00473     endSingleTimeCommands(commandBuffer);
00474 }
00475
00476 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const
00477 {
00478     if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00479         throw std::runtime_error("failed to create image!");
00480     }
00481
00482     VkMemoryRequirements memRequirements;
00483     vkGetImageMemoryRequirements(m_device, image, &memRequirements);
00484
00485     VkMemoryAllocateInfo allocInfo{};
00486     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00487     allocInfo.allocationSize = memRequirements.size;
00488     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00489
00490     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00491         throw std::runtime_error("failed to allocate image memory!");

```



```

00023 void ven::Engine::createInstance()
00024 {
00025     uint32_t glfwExtensionCount = 0;
00026     const char** glfwExtensions = nullptr;
00027     VkInstanceCreateInfo createInfo{};
00028     constexpr VkApplicationInfo appInfo{ .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO, .pNext =
    nullptr, .pApplicationName = "VEngine App", .applicationVersion = VK_MAKE_API_VERSION(0, 1, 0, 0),
    .pEngineName = "VEngine", .engineVersion = VK_MAKE_API_VERSION(0, 1, 0, 0), .apiVersion =
    VK_API_VERSION_1_0 };
00029
00030     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00031     createInfo.pApplicationInfo = &appInfo;
00032     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00033     createInfo.enabledExtensionCount = glfwExtensionCount;
00034     createInfo.ppEnabledExtensionNames = glfwExtensions;
00035
00036     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS)
00037     {
00038         throw std::runtime_error("Failed to create Vulkan instance");
00039     }
00040 }
00041
00042 void ven::Engine::loadObjects()
00043 {
00044     constexpr std::array lightColors{
00045         Colors::RED_4,
00046         Colors::GREEN_4,
00047         Colors::BLUE_4,
00048         Colors::YELLOW_4,
00049         Colors::CYAN_4,
00050         Colors::MAGENTA_4
00051     };
00052     std::shared_ptr model = Model::createModelFromFile(m_device, "assets/models/quad.obj");
00053
00054     Object quad = Object::createObject();
00055     quad.setName("quad");
00056     quad.setModel(model);
00057     quad.transform3D.translation = {0.F, .5F, 0.F};
00058     quad.transform3D.scale = {3.F, 1.F, 3.F};
00059     m_objects.emplace(quad.getId(), std::move(quad));
00060
00061     model = Model::createModelFromFile(m_device, "assets/models/flat_vase.obj");
00062     Object flatVase = Object::createObject();
00063     flatVase.setName("flat vase");
00064     flatVase.setModel(model);
00065     flatVase.transform3D.translation = {-0.5F, .5F, 0.F};
00066     flatVase.transform3D.scale = {3.F, 1.5F, 3.F};
00067     m_objects.emplace(flatVase.getId(), std::move(flatVase));
00068
00069     model = Model::createModelFromFile(m_device, "assets/models/smooth_vase.obj");
00070     Object smoothVase = Object::createObject();
00071     smoothVase.setName("smooth vase");
00072     smoothVase.setModel(model);
00073     smoothVase.transform3D.translation = {.5F, .5F, 0.F};
00074     smoothVase.transform3D.scale = {3.F, 1.5F, 3.F};
00075     m_objects.emplace(smoothVase.getId(), std::move(smoothVase));
00076
00077     for (std::size_t i = 0; i < lightColors.size(); i++)
00078     {
00079         Light pointLight = Light::createLight();
00080         pointLight.color = lightColors.at(i);
00081         glm::mat4 rotateLight = rotate(glm::mat4(1.F), (static_cast<float>(i) * glm::two_pi<float>())
    / static_cast<float>(lightColors.size()), {0.F, -1.F, 0.F});
00082         pointLight.transform3D.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F,
    1.F));
00083         m_lights.emplace(pointLight.getId(), std::move(pointLight));
00084     }
00085 }
00086
00087 void ven::Engine::mainLoop()
00088 {
00089     GlobalUbo ubo{};
00090     Camera camera{};
00091     EventManager eventManager{};
00092     std::chrono::duration<float> deltaTime{};
00093     VkCommandBuffer_T *commandBuffer = nullptr;
00094     float frameTime = NAN;
00095     unsigned long frameIndex = 0;
00096     std::chrono::time_point<std::chrono::system_clock> newTime;
00097     std::chrono::time_point<std::chrono::system_clock> currentTime =
    std::chrono::high_resolution_clock::now();
00098     std::unique_ptr<DescriptorSetLayout> globalSetLayout =
    DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    VK_SHADER_STAGE_ALL_GRAPHICS).build();
00099     std::vector<std::unique_ptr<Buffer>> uboBuffers(MAX_FRAMES_IN_FLIGHT);
00100     std::vector<VkDescriptorSet> globalDescriptorSets(MAX_FRAMES_IN_FLIGHT);
00101     ObjectRenderSystem objectRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),

```

```

    globalSetLayout->getDescriptorSetLayout();
00102     PointLightRenderSystem pointLightRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
    globalSetLayout->getDescriptorSetLayout());
00103     VkDescriptorBufferInfo bufferInfo{};
00104
00105     for (auto& uboBuffer : uboBuffers)
00106     {
00107         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00108         uboBuffer->map();
00109     }
00110     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00111         bufferInfo = uboBuffers[i]->descriptorInfo();
00112         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
&bufferInfo).build(globalDescriptorSets[i]);
00113     }
00114
00115     while (m_state != EXIT)
00116     {
00117         glfwPollEvents();
00118         eventManager.handleEvents(m_window.getGLFWWindow(), &m_state, camera, m_gui, frameTime);
00119         newTime = std::chrono::high_resolution_clock::now();
00120         deltaTime = newTime - currentTime;
00121         currentTime = newTime;
00122         frameTime = deltaTime.count();
00123         commandBuffer = m_renderer.beginFrame();
00124
00125         camera.setViewXYZ(camera.transform3D.translation, camera.transform3D.rotation);
00126         camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00127
00128         if (commandBuffer != nullptr) {
00129             frameIndex = m_renderer.getFrameIndex();
00130             FrameInfo frameInfo{.frameIndex=frameIndex, .frameTime=frameTime,
.commandBuffer=commandBuffer, .camera=camera, .globalDescriptorSet=globalDescriptorSets[frameIndex],
.objects=m_objects, .lights=m_lights};
00131             ubo.projection = camera.getProjection();
00132             ubo.view = camera.getView();
00133             ubo.inverseView = camera.getInverseView();
00134             PointLightRenderSystem::update(frameInfo, ubo);
00135             uboBuffers[frameIndex]->writeToBuffer(&ubo);
00136             uboBuffers[frameIndex]->flush();
00137
00138             m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00139             objectRenderSystem.render(frameInfo);
00140             pointLightRenderSystem.render(frameInfo);
00141
00142             if (m_gui.getState() == VISIBLE) { Gui::render(&m_renderer, m_objects, m_lights, camera,
m_device.getPhysicalDevice(), ubo); }
00143
00144             m_renderer.endSwapChainRenderPass(commandBuffer);
00145             m_renderer.endFrame();
00146             commandBuffer = nullptr;
00147         }
00148     }
00149     Gui::cleanup();
00150     vkDeviceWaitIdle(m_device.device());
00151 }

```

8.70 /home/runner/work/VEngine/VEngine/src/eventManager.cpp File Reference

```

#include <glm/gtx/norm.hpp>
#include "VEngine/EventManager.hpp"

```

[Go to the documentation of this file.](#)

Generated by Doxygen

8.73 init.cpp

[Go to the documentation of this file.](#)

```
00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include "VEngine/Gui.hpp"
00005 #include "VEngine/Colors.hpp"
00006
00007 static constexpr uint32_t DESCRIPTOR_COUNT = 1000;
00008
00009 ImGuiIO *ven::Gui::m_io = nullptr;
00010
00011 void ven::Gui::init(GLFWwindow* window, const VkInstance instance, const Device* device, const
VkRenderPass renderPass)
00012 {
00013     VkDescriptorPool pool = nullptr;
00014
00015     ImGui::CreateContext();
00016     m_io = &ImGui::GetIO();
00017     m_io->IniFilename = "assets/imgui-config.txt";
00018
00019     // ImGui::StyleColorsDark();
00020
00021     std::array<VkDescriptorPoolSize, 11> pool_sizes = {{
00022         { .type=VK_DESCRIPTOR_TYPE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00023         { .type=VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00024         { .type=VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00025         { .type=VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00026         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00027         { .type=VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00028         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00029         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00030         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00031         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00032         { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
00033     }};
00034     const VkDescriptorPoolCreateInfo pool_info = {
00035         VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00036         nullptr,
00037         VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00038         DESCRIPTOR_COUNT,
00039         std::size(pool_sizes),
00040         pool_sizes.data()
00041     };
00042
00043     if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00044         throw std::runtime_error("Failed to create ImGui descriptor pool");
00045     }
00046     ImGui_ImplVulkan_InitInfo init_info = {
00047         .Instance = instance,
00048         .PhysicalDevice = device->getPhysicalDevice(),
00049         .Device = device->device(),
00050         .Queue = device->graphicsQueue(),
00051         .DescriptorPool = pool,
00052         .MinImageCount = 3,
00053         .ImageCount = 3,
00054         .MSAASamples = VK_SAMPLE_COUNT_1_BIT
00055     };
00056
00057     ImGui_ImplGlfw_InitForVulkan(window, true);
00058     ImGui_ImplVulkan_Init(&init_info, renderPass);
00059     initStyle();
00060 }
00061
00062 void ven::Gui::initStyle()
00063 {
00064     ImGuiStyle& style = ImGui::GetStyle();
00065     style.Alpha = 1.0;
00066     style.WindowRounding = 3;
00067     style.GrabRounding = 1;
00068     style.GrabMinSize = 20;
00069     style.FrameRounding = 3;
00070
00071     style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00072     style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
00073     style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
00074     style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
00075     style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
00076     style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
00077     style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
00078     style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
00079     style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
00080     style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
00081     style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
```

```

00082 style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
00083 style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
00084 style.Colors[ImGuiCol_ScrollbarGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.44F);
00085 style.Colors[ImGuiCol_ScrollbarGrabHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00086 style.Colors[ImGuiCol_ScrollbarGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00087 style.Colors[ImGuiCol_CheckMark] = ImVec4(0.00F, 1.00F, 1.00F, 0.68F);
00088 style.Colors[ImGuiCol_SliderGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.36F);
00089 style.Colors[ImGuiCol_SliderGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.76F);
00090 style.Colors[ImGuiCol_Button] = ImVec4(0.00F, 0.65F, 0.65F, 0.46F);
00091 style.Colors[ImGuiCol_ButtonHovered] = ImVec4(0.01F, 1.00F, 1.00F, 0.43F);
00092 style.Colors[ImGuiCol_ButtonActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.62F);
00093 style.Colors[ImGuiCol_Header] = ImVec4(0.00F, 1.00F, 1.00F, 0.33F);
00094 style.Colors[ImGuiCol_HeaderHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.42F);
00095 style.Colors[ImGuiCol_HeaderActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00096 style.Colors[ImGuiCol_ResizeGrip] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00097 style.Colors[ImGuiCol_ResizeGripHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00098 style.Colors[ImGuiCol_ResizeGripActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00099 style.Colors[ImGuiCol_PlotLines] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00100 style.Colors[ImGuiCol_PlotLinesHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00101 style.Colors[ImGuiCol_PlotHistogram] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00102 style.Colors[ImGuiCol_PlotHistogramHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00103 style.Colors[ImGuiCol_TextSelectedBg] = ImVec4(0.00F, 1.00F, 1.00F, 0.22F);
00104 }

```

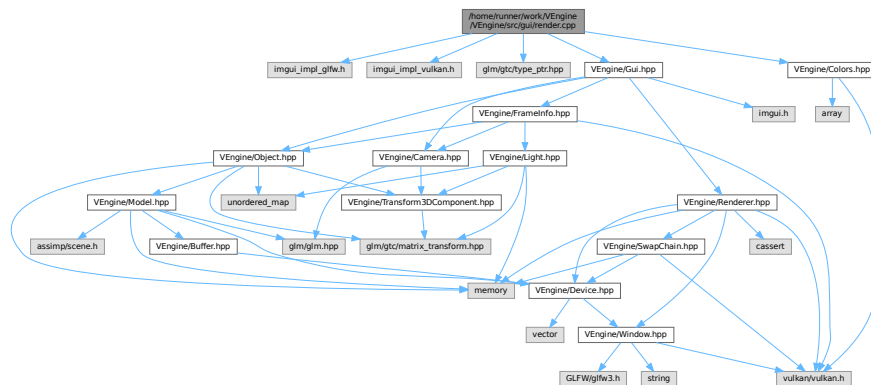
8.74 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference

```

#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"

```

Include dependency graph for render.cpp:



8.75 render.cpp

[Go to the documentation of this file.](#)

```

00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include <glm/gtc/type_ptr.hpp>
00005
00006 #include "VEngine/Gui.hpp"
00007 #include "VEngine/Colors.hpp"
00008
00009 void ven::Gui::cleanup()
00010 {
00011     ImGui_ImplVulkan_Shutdown();

```

```

00012     ImGui_ImplGlfw_Shutdown();
00013     ImGui::DestroyContext();
00014 }
00015
00016 void ven::Gui::render(Renderer* renderer, std::unordered_map<unsigned int, Object>& objects,
    std::unordered_map<unsigned int, Light>& lights, Camera& camera, const VkPhysicalDevice
    physicalDevice, GlobalUbo& ubo)
00017 {
00018     VkPhysicalDeviceProperties deviceProperties;
00019     vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
00020
00021     ImGui_ImplVulkan_NewFrame();
00022     ImGui_ImplGlfw_NewFrame();
00023     ImGui::NewFrame();
00024
00025     renderFrameWindow();
00026
00027     ImGui::Begin("Debug Window");
00028     rendererSection(renderer, ubo);
00029     cameraSection(camera);
00030     objectsSection(objects);
00031     lightsSection(lights);
00032     inputsSection(m_io);
00033     devicePropertiesSection(deviceProperties);
00034
00035     ImGui::End();
00036     ImGui::Render();
00037     ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
00038 }
00039
00040 void ven::Gui::renderFrameWindow()
00041 {
00042     const float framerate = m_io->Framerate;
00043
00044     ImGui::SetNextWindowPos(ImVec2(0.0F, 0.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
00045     ImGui::Begin("Application Info", nullptr, ImGuiWindowFlags_NoDecoration | ImGuiWindowFlags_NoMove
    | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoSavedSettings | ImGuiWindowFlags_NoFocusOnAppearing |
    ImGuiWindowFlags_NoNav);
00046     ImGui::Text("FPS: %.1f", framerate);
00047     ImGui::Text("Frame time: %.3fms", 1000.0F / framerate);
00048     ImGui::End();
00049 }
00050
00051 void ven::Gui::rendererSection(Renderer *renderer, GlobalUbo& ubo)
00052 {
00053     if (ImGui::CollapsingHeader("Renderer")) {
00054         ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());
00055
00056         if (ImGui::BeginTable("ClearColorTable", 2)) {
00057             ImGui::TableNextColumn();
00058             std::array<float, 4> clearColor = renderer->getClearColor();
00059
00060             if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
00061                 const VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1],
    clearColor[2], clearColor[3]}};
00062                 renderer->setClearColorValue(clearColorValue);
00063             }
00064
00065             ImGui::TableNextColumn();
00066             static int item_current = 0;
00067
00068             if (ImGui::Combo("Color Presets##clearColor",
00069                             &item_current,
00070                             [](void*, const int idx, const char** out_text) -> bool {
00071                                 if (idx < 0 || idx >=
00072 static_cast<int>(std::size(Colors::COLOR_PRESETS_VK))) { return false; }
00073                                 *out_text = Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
    long>(idx)).first;
00074                                 return true;
00075                             },
00076                             nullptr,
00077                             std::size(Colors::COLOR_PRESETS_VK))) {
00078                 renderer->setClearColorValue(Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
    long>(item_current)).second);
00079             }
00080
00081             ImGui::TableNextColumn();
00082             ImGui::ColorEdit4("Ambient Light Color", glm::value_ptr(ubo.ambientLightColor));
00083             ImGui::TableNextColumn();
00084             if (ImGui::Combo("Color Presets##ambientColor",
00085                             &item_current,
00086                             [](void*, const int idx, const char** out_text) -> bool {
00087                                 if (idx < 0 || idx >=
00088 static_cast<int>(std::size(Colors::COLOR_PRESETS_4))) { return false; }
00089                                 *out_text = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
    long>(idx)).first;
00090                                 return true;
00091                             },
00092                             nullptr,
00093                             std::size(Colors::COLOR_PRESETS_4))) {
00094                 renderer->setAmbientLightColor(Colors::COLOR_PRESETS_4.at(static_cast<unsigned
    long>(item_current)).second);
00095             }
00096         }
00097     }
00098 }

```

```

00089             },
00090             nullptr,
00091             std::size(Colors::COLOR_PRESETS_4)) {
00092         ubo.ambientLightColor = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
long>(item_current)).second;
00093     }
00094
00095     ImGui::TableNextColumn();
00096     ImGui::SliderFloat(("Intensity##" + std::to_string(0)).c_str(), &ubo.ambientLightColor.a,
0.0F, 1.0F);
00097     ImGui::TableNextColumn();
00098     if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a =
DEFAULT_AMBIENT_LIGHT_INTENSITY; }
00099
00100     ImGui::EndTable();
00101 }
00102
00103 static bool fullscreen = false;
00104 if (ImGui::Checkbox("Fullscreen", &fullscreen)) {
00105     renderer->getWindow().setFullscreen(fullscreen, renderer->getWindow().getExtent().width,
renderer->getWindow().getExtent().height);
00106 }
00107 }
00108 }
00109
00110 void ven::Gui::cameraSection(Camera &camera)
00111 {
00112     if (ImGui::CollapsingHeader("Camera")) {
00113         float fov = camera.getFov();
00114         float near = camera.getNear();
00115         float far = camera.getFar();
00116         if (ImGui::BeginTable("CameraTable", 2)) {
00117             ImGui::TableNextColumn();
00118             ImGui::DragFloat3("Position", glm::value_ptr(camera.transform3D.translation), 0.1F);
00119             ImGui::TableNextColumn();
00120             if (ImGui::Button("Reset##position")) { camera.transform3D.translation = DEFAULT_POSITION; }
00121         }
00122         ImGui::TableNextColumn();
00123         ImGui::DragFloat3("Rotation", glm::value_ptr(camera.transform3D.rotation), 0.1F);
00124         ImGui::TableNextColumn();
00125         if (ImGui::Button("Reset##rotation")) { camera.transform3D.rotation = DEFAULT_ROTATION; }
00126
00127         ImGui::TableNextColumn();
00128         if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
camera.setFov(fov); }
00129         ImGui::TableNextColumn();
00130         if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }
00131
00132         ImGui::TableNextColumn();
00133         if (ImGui::SliderFloat("Near", &near, 0.001F, 10.0F)) { camera.setNear(near); }
00134         ImGui::TableNextColumn();
00135         if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }
00136
00137         ImGui::TableNextColumn();
00138         if (ImGui::SliderFloat("Far", &far, 1.F, 1000.0F)) { camera.setFar(far); }
00139         ImGui::TableNextColumn();
00140         if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }
00141
00142         ImGui::TableNextColumn();
00143         float moveSpeed = camera.getMoveSpeed();
00144         if (ImGui::SliderFloat("Move speed", &moveSpeed, 0.1F, 10.0F)) {
camera.setMoveSpeed(moveSpeed); }
00145         ImGui::TableNextColumn();
00146         if (ImGui::Button("Reset##moveSpeed")) { camera.setMoveSpeed(DEFAULT_MOVE_SPEED); }
00147
00148         ImGui::TableNextColumn();
00149         float lookSpeed = camera.getLookSpeed();
00150         if (ImGui::SliderFloat("Look speed", &lookSpeed, 0.1F, 10.0F)) {
camera.setLookSpeed(lookSpeed); }
00151         ImGui::TableNextColumn();
00152         if (ImGui::Button("Reset##lookSpeed")) { camera.setLookSpeed(DEFAULT_LOOK_SPEED); }
00153
00154         ImGui::EndTable();
00155     }
00156 }
00157 }
00158
00159 void ven::Gui::objectsSection(std::unordered_map<unsigned int, Object>& objects)
00160 {
00161     if (ImGui::CollapsingHeader("Objects")) {
00162         bool open = false;
00163         for (auto& [id, object] : objects) {
00164             ImGui::PushStyleColor(ImGuiCol_Text, { Colors::GRAY_4.r, Colors::GRAY_4.g,
Colors::GRAY_4.b, 1.0F });
00165             open = ImGui::TreeNode(std::string(object.getName() + " [" +
std::to_string(object.getId()) + "]").c_str());

```



```

00166         ImGui::PopStyleColor(1);
00167         if (open) {
00168             ImGui::DragFloat3(("Position##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.translation), 0.1F);
00169             ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.rotation), 0.1F);
00170             ImGui::DragFloat3(("Scale##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.scale), 0.1F);
00171             ImGui::Text("Address: %p", &object);
00172             ImGui::TreePop();
00173         }
00174     }
00175 }
00176 }
00177
00178 void ven::Gui::lightsSection(std::unordered_map<unsigned int, Light> &lights)
00179 {
00180     if (ImGui::CollapsingHeader("Lights")) {
00181         bool open = false;
00182
00183         for (auto& [id, light] : lights) {
00184             ImGui::PushStyleColor(ImGuiCol_Text, {light.color.r, light.color.g, light.color.b, 1.0F});
00185             open = ImGui::TreeNode(std::string(light.getName() + " [" + std::to_string(light.getId())
+ "]").c_str());
00186             ImGui::PopStyleColor(1);
00187             if (open) {
00188                 ImGui::Text("Address: %p", &light);
00189                 ImGui::DragFloat3(("Position##" + std::to_string(light.getId()).c_str(),
glm::value_ptr(light.transform3D.translation), 0.1F);
00190                 ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId()).c_str(),
glm::value_ptr(light.transform3D.rotation), 0.1F);
00191                 ImGui::DragFloat3(("Scale##" + std::to_string(light.getId()).c_str(),
glm::value_ptr(light.transform3D.scale), 0.1F);
00192                 if (ImGui::BeginTable("ColorTable", 2)) {
00193                     ImGui::TableNextColumn(); ImGui::ColorEdit4(("Color##" +
std::to_string(light.getId()).c_str(), glm::value_ptr(light.color));
00194
00195                     ImGui::TableNextColumn();
00196                     static int item_current = 0;
00197                     if (ImGui::Combo("Color Presets",
&item_current,
[])(void*, const int idx, const char** out_text) -> bool {
00200                         if (idx < 0 || idx >=
static_cast<int>(std::size(Colors::COLOR_PRESETS_3))) { return false; }
00201                         *out_text = Colors::COLOR_PRESETS_3.at(static_cast<unsigned
long>(idx)).first;
00202                         return true;
00203                     },
00204                     nullptr,
00205                     std::size(Colors::COLOR_PRESETS_3)) {
00206                         light.color = {Colors::COLOR_PRESETS_3.at(static_cast<unsigned
long>(item_current)).second, light.color.a};
00207                     }
00208
00209                     ImGui::TableNextColumn();
00210                     ImGui::SliderFloat(("Intensity##" + std::to_string(light.getId()).c_str(),
&light.color.a, 0.0F, 5.F);
00211                     ImGui::TableNextColumn();
00212                     if (ImGui::Button(("Reset##" + std::to_string(light.getId()).c_str())) {
light.color.a = DEFAULT_LIGHT_INTENSITY; }
00213
00214                     ImGui::EndTable();
00215                 }
00216                 ImGui::TreePop();
00217             }
00218         }
00219     }
00220 }
00221
00222 void ven::Gui::inputsSection(const ImGuiIO* io)
00223 {
00224     if (ImGui::CollapsingHeader("Input")) {
00225         ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io->MousePos.x, io->MousePos.y)
: ImGui::Text("Mouse pos: <INVALID>");
00226         ImGui::Text("Mouse delta: (%g, %g)", io->MouseDelta.x, io->MouseDelta.y);
00227         ImGui::Text("Mouse down:");
00228         for (int i = 0; i < static_cast<int>(std::size(io->MouseDown)); i++) {
00229             if (ImGui::IsMouseDown(i)) {
00230                 ImGui::SameLine();
00231                 ImGui::Text("%b%d (%.02f secs)", i, io->MouseDownDuration[i]);
00232             }
00233         }
00234         ImGui::Text("Mouse wheel: %.1f", io->MouseWheel);
00235         ImGui::Text("Keys down:");
00236         for (auto key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
static_cast<ImGuiKey>(key + 1)) {
00237             if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }

```

```

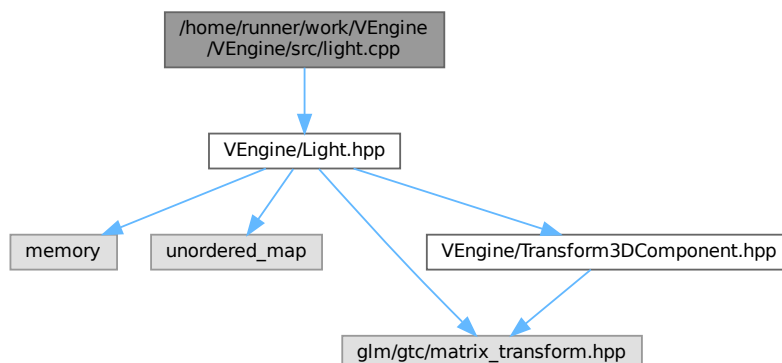
00238         ImGui::SameLine();
00239         ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\\\"%s\\\" : "\\\"%s\\\" %d",
ImGui::GetKeyName(key), key);
00240     }
00241 }
00242 }
00243
00244 void ven::Gui::devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties)
00245 {
00246     if (ImGui::CollapsingHeader("Device Properties")) {
00247         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00248
00249             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00250             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion));
00251             ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00252             ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00253             ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00254             ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00255             ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00256             ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantsSize);
00257             ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00258             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00259             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",
deviceProperties.limits.maxImageDimension2D);
00260             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00261             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00262             ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00263             ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00264             ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00265             ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00266             ImGui::EndTable();
00267         }
00268     }
00269 }

```

8.76 /home/runner/work/VEngine/VEngine/src/light.cpp File Reference

#include "VEngine/Light.hpp"

Include dependency graph for light.cpp:



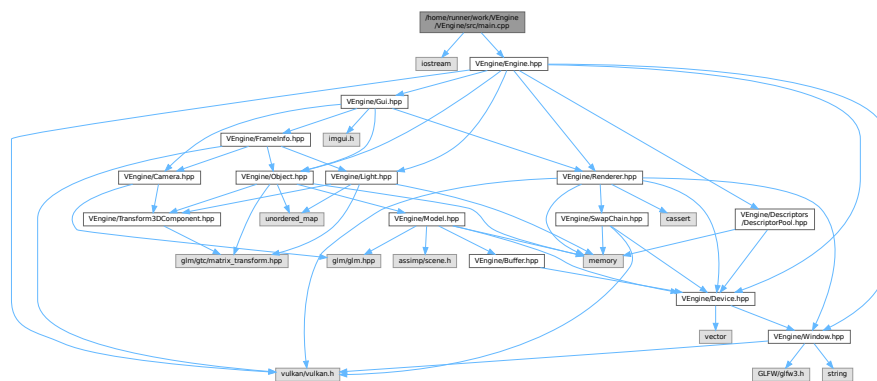
8.77 light.cpp

[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Light.hpp"
00002
00003 ven::Light ven::Light::createLight(const float radius, const glm::vec4 color)
00004 {
00005     static unsigned int objId = 0;
00006     Light light(objId++);
00007
00008     light.color = color;
00009     light.transform3D.scale.x = radius;
00010
00011     return light;
00012 }
```

8.78 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```
#include <iostream>
#include "VEngine/Engine.hpp"
Include dependency graph for main.cpp:
```



Functions

- `int main ()`

8.78.1 Function Documentation

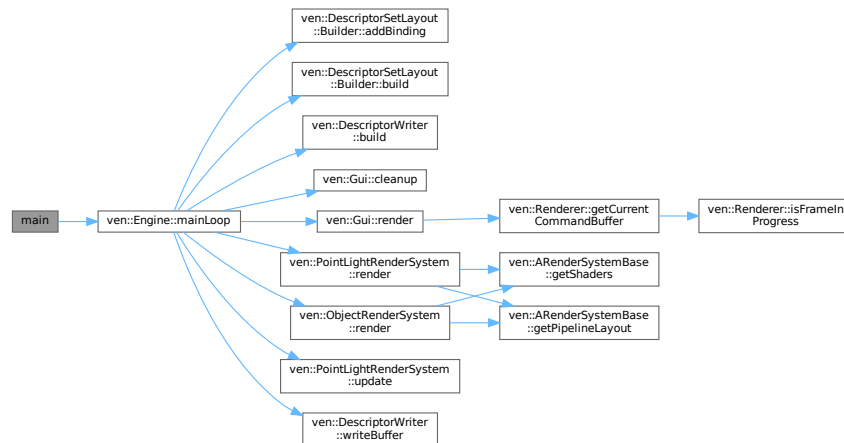
8.78.1.1 main()

```
int main ()
```

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

References [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



8.79 main.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002
00003 #include "VEngine/Engine.hpp"
00004
00005 using namespace ven;
00006
00007 int main()
00008 {
00009     try {
00010         Engine engine{};
00011         engine.mainLoop();
00012     } catch (const std::exception &e) {
00013         std::cerr << "std exception: " << e.what() << '\n';
00014         return EXIT_FAILURE;
00015     } catch (...) {
00016         std::cerr << "Unknown error\n";
00017         return EXIT_SUCCESS;
00018     }
00019     return EXIT_SUCCESS;
00020 }

```

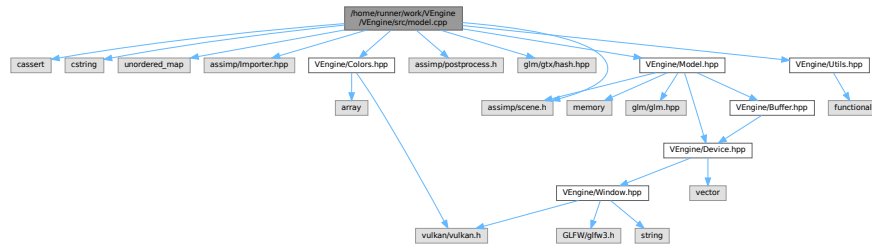
8.80 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference

```

#include <cassert>
#include <cstring>
#include <unordered_map>
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Colors.hpp"
#include "VEngine/Model.hpp"

```

#include "VEngine/Utils.hpp"
 Include dependency graph for model.cpp:



Classes

- struct `std::hash< ven::Model::Vertex >`

Macros

- #define `GLM_ENABLE_EXPERIMENTAL`

8.80.1 Macro Definition Documentation

8.80.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 9 of file [model.cpp](#).

8.81 model.cpp

[Go to the documentation of this file.](#)

```
00001 #include <cassert>
00002 #include <cstring>
00003 #include <unordered_map>
00004
00005 #include <assimp/Importer.hpp>
00006 #include <assimp/scene.h>
00007 #include <assimp/postprocess.h>
00008
00009 #define GLM_ENABLE_EXPERIMENTAL
00010 #include <glm/gtx/hash.hpp>
00011
00012 #include "VEngine/Colors.hpp"
00013 #include "VEngine/Model.hpp"
00014 #include "VEngine/Utils.hpp"
00015
00016 template<>
00017 struct std::hash<ven::Model::Vertex> {
00018     size_t operator() (ven::Model::Vertex const &vertex) const noexcept {
00019         size_t seed = 0;
00020         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00021         return seed;
00022     }
00023 };
00024
00025 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
    m_indexCount(0)
```

```

00026 {
00027     createVertexBuffer(builder.vertices);
00028     createIndexBuffer(builder.indices);
00029 }
00030
00031 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00032 {
00033     m_vertexCount = static_cast<uint32_t>(vertices.size());
00034     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00035     const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
00036     uint32_t vertexSize = sizeof(vertices[0]);
00037
00038     Buffer stagingBuffer{m_device, vertexSize, m_vertexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00039
00040     stagingBuffer.map();
00041     stagingBuffer.writeToBuffer(vertices.data());
00042
00043     m_vertexBuffer = std::make_unique<Buffer>(m_device, vertexSize, m_vertexCount,
VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00044
00045     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00046 }
00047
00048 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00049 {
00050     m_indexCount = static_cast<uint32_t>(indices.size());
00051     m_hasIndexBuffer = m_indexCount > 0;
00052
00053     if (!m_hasIndexBuffer) {
00054         return;
00055     }
00056
00057     uint32_t indexSize = sizeof(indices[0]);
00058
00059     Buffer stagingBuffer{m_device, indexSize, m_indexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00060
00061     stagingBuffer.map();
00062     stagingBuffer.writeToBuffer(indices.data());
00063
00064     m_indexBuffer = std::make_unique<Buffer>(m_device, indexSize, m_indexCount,
VK_BUFFER_USAGE_INDEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00065
00066     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), sizeof(indices[0]) *
m_indexCount);
00067 }
00068
00069 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00070 {
00071     if (m_hasIndexBuffer) {
00072         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00073     } else {
00074         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00075     }
00076 }
00077
00078 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00079 {
00080     const std::array buffers{m_vertexBuffer->getBuffer()};
00081     constexpr std::array<VkDeviceSize, 1> offsets{0};
00082     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets.data());
00083
00084     if (m_hasIndexBuffer) {
00085         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00086     }
00087 }
00088
00089 std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
&filename)
00090 {
00091     Builder builder{};
00092     builder.loadModel(filename);
00093     return std::make_unique<Model>(device, builder);
00094 }
00095
00096 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00097 {
00098     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00099     bindingDescriptions[0].binding = 0;
00100     bindingDescriptions[0].stride = sizeof(Vertex);
00101     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00102     return bindingDescriptions;
00103 }
00104

```

```

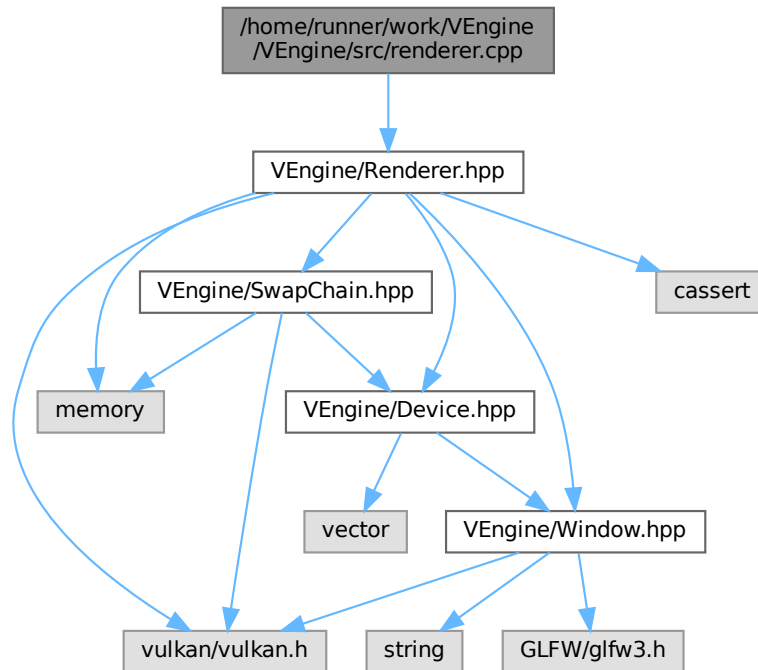
00105 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00106 {
00107     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00108
00109     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00110     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00111     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00112     attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00113
00114     return attributeDescriptions;
00115 }
00116
00117 void ven::Model::Builder::loadModel(const std::string &filename) {
00118     Assimp::Importer importer;
00119
00120     const aiScene* scene = importer.ReadFile(filename, aiProcess_Triangulate | aiProcess_FlipUVs |
00121     aiProcess_CalcTangentSpace | aiProcess_GenNormals);
00122
00123     if ((scene == nullptr) || ((scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE) != 0U) ||
00124     !scene->mRootNode) {
00125         throw std::runtime_error("Failed to load model with Assimp: " +
00126         std::string(importer.GetErrorString()));
00127     }
00128
00129     vertices.clear();
00130     indices.clear();
00131
00132     processNode(scene->mRootNode, scene);
00133 }
00134
00135 void ven::Model::Builder::processNode(const aiNode* node, const aiScene* scene) {
00136     for (unsigned int i = 0; i < node->mNumMeshes; i++) {
00137         const aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
00138         processMesh(mesh, scene);
00139     }
00140
00141     for (unsigned int i = 0; i < node->mNumChildren; i++) {
00142         processNode(node->mChildren[i], scene);
00143     }
00144 }
00145
00146 void ven::Model::Builder::processMesh(const aiMesh* mesh, const aiScene* scene) {
00147     std::unordered_map<Vertex, uint32_t> uniqueVertices;
00148
00149     for (unsigned int i = 0; i < mesh->mNumVertices; i++) {
00150         Vertex vertex{};
00151
00152         vertex.position = glm::vec3(
00153             mesh->mVertices[i].x,
00154             mesh->mVertices[i].y,
00155             mesh->mVertices[i].z
00156         );
00157
00158         if (mesh->HasNormals()) {
00159             vertex.normal = glm::vec3(
00160                 mesh->mNormals[i].x,
00161                 mesh->mNormals[i].y,
00162                 mesh->mNormals[i].z
00163             );
00164         }
00165
00166         if (mesh->mTextureCoords[0] != nullptr) {
00167             vertex.uv = glm::vec2(
00168                 mesh->mTextureCoords[0][i].x,
00169                 mesh->mTextureCoords[0][i].y
00170             );
00171         } else {
00172             vertex.uv = glm::vec2(0.0F, 0.0F);
00173         }
00174
00175         if (vertex.color == Colors::BLACK_3) {
00176             vertex.color = Colors::WHITE_3;
00177         }
00178
00179         if (!uniqueVertices.contains(vertex)) {
00180             uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00181             vertices.push_back(vertex);
00182         }
00183     }
00184
00185     indices.push_back(uniqueVertices[vertex]);
00186 }

```

8.82 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

```
#include "VEngine/Renderer.hpp"
```

Include dependency graph for renderer.cpp:



8.83 renderer.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Renderer.hpp"
00002
00003 void ven::Renderer::createCommandBuffers()
00004 {
00005     m_commandBuffers.resize(MAX_FRAMES_IN_FLIGHT);
00006     VkCommandBufferAllocateInfo allocInfo{};
00007     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00008     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00009     allocInfo.commandPool = m_device.getCommandPool();
00010     allocInfo.commandBufferCount = static_cast<uint32_t>(m_commandBuffers.size());
00011
00012     if (vkAllocateCommandBuffers(m_device.device(), &allocInfo, m_commandBuffers.data()) !=
VK_SUCCESS) {
00013         throw std::runtime_error("Failed to allocate command buffers");
00014     }
00015 }
00016
00017 void ven::Renderer::freeCommandBuffers()
00018 {
00019     vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(),
static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
00020     m_commandBuffers.clear();
00021 }
00022
00023 void ven::Renderer::recreateSwapChain()
00024 {

```



```

00025     VkExtent2D extent = m_window.getExtent();
00026     while (extent.width == 0 || extent.height == 0) {
00027         extent = m_window.getExtent();
00028         glfwWaitEvents();
00029     }
00030     vkDeviceWaitIdle(m_device.device());
00031     if (m_swapChain == nullptr) {
00032         m_swapChain = std::make_unique<SwapChain>(m_device, extent);
00033     } else {
00034         std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
00035         m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
00036         if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
00037             throw std::runtime_error("Swap chain image/depth format changed");
00038         }
00039     }
00040     // well be back
00041 }
00042
00043 VkCommandBuffer ven::Renderer::beginFrame()
00044 {
00045     assert(!m_isFrameStarted && "Can't start new frame while previous one is still in progress");
00046
00047     const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
00048     if (result == VK_ERROR_OUT_OF_DATE_KHR) {
00049         recreateSwapChain();
00050         return nullptr;
00051     }
00052
00053     if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
00054         throw std::runtime_error("Failed to acquire swap chain image");
00055     }
00056
00057     m_isFrameStarted = true;
00058
00059     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00060     VkCommandBufferBeginInfo beginInfo{};
00061     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00062
00063     if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
00064         throw std::runtime_error("Failed to begin recording command m_buffer");
00065     }
00066     return commandBuffer;
00067 }
00068
00069 void ven::Renderer::endFrame()
00070 {
00071     assert(m_isFrameStarted && "Can't end frame that hasn't been started");
00072
00073     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00074     if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
00075         throw std::runtime_error("Failed to record command buffer");
00076     }
00077     if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer,
&m_currentImageIndex); result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
m_window.wasWindowResized()) {
00078         m_window.resetWindowResizedFlag();
00079         recreateSwapChain();
00080     }
00081     else if (result != VK_SUCCESS) {
00082         throw std::runtime_error("Failed to submit command buffer");
00083     }
00084
00085     m_isFrameStarted = false;
00086     m_currentFrameIndex = (m_currentFrameIndex + 1) % MAX_FRAMES_IN_FLIGHT;
00087 }
00088
00089 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00090 {
00091     assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");
00092     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
from a different frame");
00093
00094     VkRenderPassBeginInfo renderPassInfo{};
00095     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00096     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00097     renderPassInfo.framebuffer = m_swapChain->getFrameBuffer(m_currentImageIndex);
00098
00099     renderPassInfo.renderArea.offset = {.x=0, .y=0};
00100     renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();
00101
00102     renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
00103     renderPassInfo.pClearValues = m_clearValues.data();
00104
00105     vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00106
00107     VkViewport viewport{};
00108     viewport.x = 0.0F;

```

```

00109     viewport.y = 0.0F;
00110     viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00111     viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00112     viewport.minDepth = 0.0F;
00113     viewport.maxDepth = 1.0F;
00114     const VkRect2D scissor{{0, 0}, m_swapChain->getSwapChainExtent()};
00115     vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00116     vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00117 }
00118
00119 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00120 {
00121     assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
00122     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
    from a different frame");
00123
00124     vkCmdEndRenderPass(commandBuffer);
00125 }

```

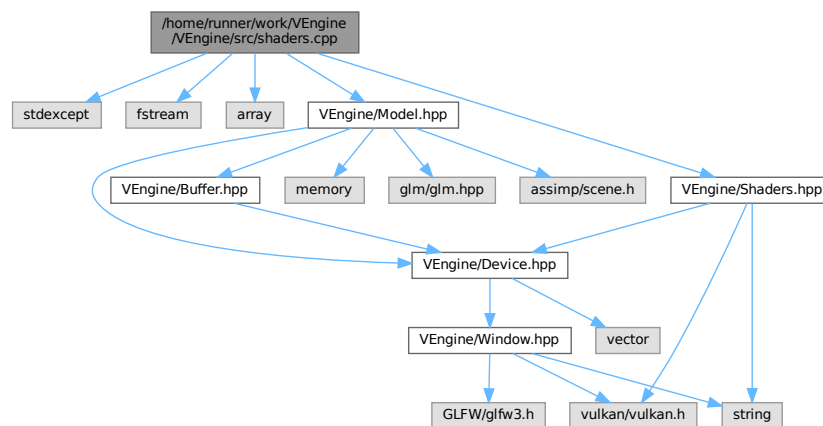
8.84 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```

#include <stdexcept>
#include <fstream>
#include <array>
#include "VEngine/Model.hpp"
#include "VEngine/Shaders.hpp"

```

Include dependency graph for shaders.cpp:



8.85 shaders.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002 #include <fstream>
00003 #include <array>
00004
00005 #include "VEngine/Model.hpp"
00006 #include "VEngine/Shaders.hpp"
00007
00008 ven::Shaders::~Shaders()
00009 {
00010     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00011     vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00012     vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);

```

```

00013 }
00014
00015 std::vector<char> ven::Shaders::readFile(const std::string &filename) {
00016     std::ifstream file(filename, std::ios::binary | std::ios::ate);
00017     if (!file.is_open()) {
00018         throw std::runtime_error("failed to open file!");
00019     }
00020
00021     const long int fileSize = file.tellg();
00022     std::vector<char> buffer(static_cast<long unsigned int>(fileSize));
00023     file.seekg(0);
00024     file.read(buffer.data(), fileSize);
00025     return buffer;
00026 }
00027
00028 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
    fragFilepath, const PipelineConfigInfo& configInfo)
00029 {
00030     const std::vector<char> vertCode = readFile(vertFilepath);
00031     const std::vector<char> fragCode = readFile(fragFilepath);
00032
00033     createShaderModule(vertCode, &m_vertShaderModule);
00034     createShaderModule(fragCode, &m_fragShaderModule);
00035
00036     std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
00037     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00038     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00039     shaderStages[0].module = m_vertShaderModule;
00040     shaderStages[0].pName = "main";
00041     shaderStages[0].flags = 0;
00042     shaderStages[0].pNext = nullptr;
00043     shaderStages[0].pSpecializationInfo = nullptr;
00044
00045     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00046     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00047     shaderStages[1].module = m_fragShaderModule;
00048     shaderStages[1].pName = "main";
00049     shaderStages[1].flags = 0;
00050     shaderStages[1].pNext = nullptr;
00051     shaderStages[1].pSpecializationInfo = nullptr;
00052
00053     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00054     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00055     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00056     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00057     vertexInputInfo.vertexAttributeDescriptionCount =
        static_cast<uint32_t>(attributeDescriptions.size());
00058     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00059     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00060     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00061
00062     VkPipelineViewportStateCreateInfo viewportInfo{};
00063     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00064     viewportInfo.viewportCount = 1;
00065     viewportInfo.pViewports = nullptr;
00066     viewportInfo.scissorCount = 1;
00067     viewportInfo.pScissors = nullptr;
00068
00069     VkGraphicsPipelineCreateInfo pipelineInfo{};
00070     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00071     pipelineInfo.stageCount = 2;
00072     pipelineInfo.pStages = shaderStages.data();
00073     pipelineInfo.pVertexInputState = &vertexInputInfo;
00074     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00075     pipelineInfo.pViewportState = &viewportInfo;
00076     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00077     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00078
00079     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00080     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00081     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00082
00083     pipelineInfo.layout = configInfo.pipelineLayout;
00084     pipelineInfo.renderPass = configInfo.renderPass;
00085     pipelineInfo.subpass = configInfo.subpass;
00086
00087     pipelineInfo.basePipelineIndex = -1;
00088     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00089
00090     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
        &m_graphicsPipeline) != VK_SUCCESS) {
00091         throw std::runtime_error("failed to create graphics pipeline");
00092     }
00093 }
00094
00095 }
00096

```

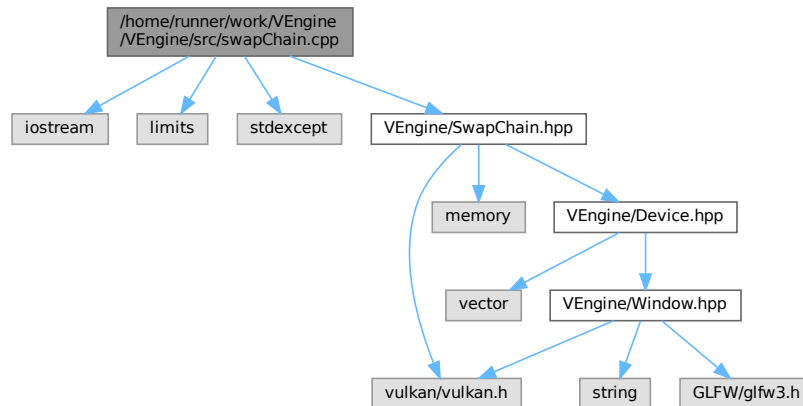
```

00097 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
00098 {
00099     VkShaderModuleCreateInfo createInfo{};
00100     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00101     createInfo.codeSize = code.size();
00102     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00103
00104     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00105         throw std::runtime_error("failed to create shader module");
00106     }
00107 }
00108
00109 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00110 {
00111     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00112     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00113     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00114
00115     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00116     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00117     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00118     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00119     configInfo.rasterizationInfo.lineWidth = 1.0F;
00120     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE; // to enable later
00121     configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00122     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00123     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00124     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00125     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00126
00127     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00128     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00129     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00130     configInfo.multisampleInfo.minSampleShading = 1.0F;
00131     configInfo.multisampleInfo.pSampleMask = nullptr;
00132     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00133     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00134
00135     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00136     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00137     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00138     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00139     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00140     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00141     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00142     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00143
00144     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00145     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00146     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00147     configInfo.colorBlendInfo.attachmentCount = 1;
00148     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00149     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00150     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00151     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00152     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00153
00154     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00155     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00156     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00157     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00158     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00159     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00160     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00161     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00162     configInfo.depthStencilInfo.front = {};
00163     configInfo.depthStencilInfo.back = {};
00164
00165     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00166     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00167     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00168     configInfo.dynamicStateInfo.dynamicStateCount =
static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00169     configInfo.dynamicStateInfo.flags = 0;
00170     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();
00171     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00172 }

```

8.86 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference

```
#include <iostream>
#include <limits>
#include <stdexcept>
#include "VEngine/SwapChain.hpp"
Include dependency graph for swapChain.cpp:
```



8.87 swapChain.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002 #include <limits>
00003 #include <stdexcept>
00004
00005 #include "VEngine/SwapChain.hpp"
00006
00007 ven::SwapChain::~SwapChain()
00008 {
00009     for (VkImageView_T *imageView : m_swapChainImageViews) {
00010         vkDestroyImageView(m_device.device(), imageView, nullptr);
00011     }
00012     m_swapChainImageViews.clear();
00013
00014     if (m_swapChain != nullptr) {
00015         vkDestroySwapchainKHR(m_device.device(), m_swapChain, nullptr);
00016         m_swapChain = nullptr;
00017     }
00018
00019     for (size_t i = 0; i < m_depthImages.size(); i++) {
00020         vkDestroyImageView(m_device.device(), m_depthImageViews[i], nullptr);
00021         vkDestroyImage(m_device.device(), m_depthImages[i], nullptr);
00022         vkFreeMemory(m_device.device(), m_depthImageMemory[i], nullptr);
00023     }
00024
00025     for (VkFramebuffer_T *framebuffer : m_swapChainFrameBuffers) {
00026         vkDestroyFramebuffer(m_device.device(), framebuffer, nullptr);
00027     }
00028
00029     vkDestroyRenderPass(m_device.device(), m_renderPass, nullptr);
00030
00031     // cleanup synchronization objects
00032     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00033         vkDestroySemaphore(m_device.device(), m_renderFinishedSemaphores[i], nullptr);
00034         vkDestroySemaphore(m_device.device(), m_imageAvailableSemaphores[i], nullptr);
00035         vkDestroyFence(m_device.device(), m_inFlightFences[i], nullptr);
00036     }

```

```

00037 }
00038
00039 void ven::SwapChain::init()
00040 {
00041     createSwapChain();
00042     createImageViews();
00043     createRenderPass();
00044     createDepthResources();
00045     createFrameBuffers();
00046     createSyncObjects();
00047 }
00048
00049 VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
00050 {
00051     vkWaitForFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame], VK_TRUE,
std::numeric_limits<uint64_t>::max());
00052
00053     return vkAcquireNextImageKHR(m_device.device(), m_swapChain, std::numeric_limits<uint64_t>::max(),
m_imageAvailableSemaphores[m_currentFrame], VK_NULL_HANDLE, imageIndex);
00054 }
00055
00056 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
*imageIndex)
00057 {
00058     if (m_imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00059         vkWaitForFences(m_device.device(), 1, &m_imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00060     }
00061     m_imagesInFlight[*imageIndex] = m_inFlightFences[m_currentFrame];
00062
00063     VkSubmitInfo submitInfo = {};
00064     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00065
00066     const std::array<VkSemaphore, 1> waitSemaphores = {m_imageAvailableSemaphores[m_currentFrame]};
00067     constexpr std::array<VkPipelineStageFlags, 1> waitStages =
{VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00068     submitInfo.waitSemaphoreCount = 1;
00069     submitInfo.pWaitSemaphores = waitSemaphores.data();
00070     submitInfo.pWaitDstStageMask = waitStages.data();
00071
00072     submitInfo.commandBufferCount = 1;
00073     submitInfo.pCommandBuffers = buffers;
00074
00075     const std::array<VkSemaphore, 1> signalSemaphores = {m_renderFinishedSemaphores[m_currentFrame]};
00076     submitInfo.signalSemaphoreCount = 1;
00077     submitInfo.pSignalSemaphores = signalSemaphores.data();
00078
00079     vkResetFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame]);
00080     if (vkQueueSubmit(m_device.graphicsQueue(), 1, &submitInfo, m_inFlightFences[m_currentFrame]) !=
VK_SUCCESS) {
00081         throw std::runtime_error("failed to submit draw command m_buffer!");
00082     }
00083
00084     VkPresentInfoKHR presentInfo = {};
00085     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00086
00087     presentInfo.waitSemaphoreCount = 1;
00088     presentInfo.pWaitSemaphores = signalSemaphores.data();
00089
00090     const std::array<VkSwapchainKHR, 1> swapChains = {m_swapChain};
00091     presentInfo.swapchainCount = 1;
00092     presentInfo.pSwapchains = swapChains.data();
00093
00094     presentInfo.pImageIndices = imageIndex;
00095
00096     const VkResult result = vkQueuePresentKHR(m_device.presentQueue(), &presentInfo);
00097
00098     m_currentFrame = (m_currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00099
00100     return result;
00101 }
00102
00103 void ven::SwapChain::createSwapChain()
00104 {
00105     const auto [capabilities, formats, presentModes] = m_device.getSwapChainSupport();
00106
00107     const auto [format, colorSpace] = chooseSwapSurfaceFormat(formats);
00108     const VkPresentModeKHR presentMode = chooseSwapPresentMode(presentModes);
00109     const VkExtent2D extent = chooseSwapExtent(capabilities);
00110
00111     uint32_t imageCount = capabilities.minImageCount + 1;
00112     if (capabilities.maxImageCount > 0 && imageCount > capabilities.maxImageCount) {
00113         imageCount = capabilities.maxImageCount;
00114     }
00115
00116     VkSwapchainCreateInfoKHR createInfo = {};
00117     createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
00118     createInfo.surface = m_device.surface();

```

```

00119
00120     createInfo.minImageCount = imageCount;
00121     createInfo.imageFormat = format;
00122     createInfo.imageColorSpace = colorSpace;
00123     createInfo.imageExtent = extent;
00124     createInfo.imageArrayLayers = 1;
00125     createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
00126
00127     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
m_device.findPhysicalQueueFamilies();
00128     const std::array<uint32_t, 2> queueFamilyIndices = {graphicsFamily, presentFamily};
00129
00130     if (graphicsFamily != presentFamily) {
00131         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00132         createInfo.queueFamilyIndexCount = 2;
00133         createInfo.pQueueFamilyIndices = queueFamilyIndices.data();
00134     } else {
00135         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00136         createInfo.queueFamilyIndexCount = 0; // Optional
00137         createInfo.pQueueFamilyIndices = nullptr; // Optional
00138     }
00139
00140     createInfo.preTransform = capabilities.currentTransform;
00141     createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
00142
00143     createInfo.presentMode = presentMode;
00144     createInfo.clipped = VK_TRUE;
00145
00146     createInfo.oldSwapchain = m_oldSwapChain == nullptr ? VK_NULL_HANDLE :
m_oldSwapChain->m_swapChain;
00147
00148     if (vkCreateSwapchainKHR(m_device.device(), &createInfo, nullptr, &m_swapChain) != VK_SUCCESS) {
00149         throw std::runtime_error("failed to create swap chain!");
00150     }
00151
00152     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, nullptr);
00153     m_swapChainImages.resize(imageCount);
00154     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, m_swapChainImages.data());
00155
00156     m_swapChainImageFormat = format;
00157     m_swapChainExtent = extent;
00158 }
00159
00160 void ven::SwapChain::createImageViews()
00161 {
00162     m_swapChainImageViews.resize(m_swapChainImages.size());
00163     for (size_t i = 0; i < m_swapChainImages.size(); i++) {
00164         VkImageViewCreateInfo viewInfo{};
00165         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00166         viewInfo.image = m_swapChainImages[i];
00167         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00168         viewInfo.format = m_swapChainImageFormat;
00169         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00170         viewInfo.subresourceRange.baseMipLevel = 0;
00171         viewInfo.subresourceRange.levelCount = 1;
00172         viewInfo.subresourceRange.baseArrayLayer = 0;
00173         viewInfo.subresourceRange.layerCount = 1;
00174
00175         if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_swapChainImageViews[i]) !=
VK_SUCCESS) {
00176             throw std::runtime_error("failed to create texture image view!");
00177         }
00178     }
00179 }
00180
00181 void ven::SwapChain::createRenderPass()
00182 {
00183     VkAttachmentDescription depthAttachment{};
00184     depthAttachment.format = findDepthFormat();
00185     depthAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00186     depthAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00187     depthAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00188     depthAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00189     depthAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00190     depthAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00191     depthAttachment.finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00192
00193     VkAttachmentReference depthAttachmentRef{};
00194     depthAttachmentRef.attachment = 1;
00195     depthAttachmentRef.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00196
00197     VkAttachmentDescription colorAttachment = {};
00198     colorAttachment.format = getSwapChainImageFormat();
00199     colorAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00200     colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00201     colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
00202     colorAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;

```

```

00203     colorAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00204     colorAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00205     colorAttachment.finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
00206
00207     VkAttachmentReference colorAttachmentRef = {};
00208     colorAttachmentRef.attachment = 0;
00209     colorAttachmentRef.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00210
00211     VkSubpassDescription subpass = {};
00212     subpass.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
00213     subpass.colorAttachmentCount = 1;
00214     subpass.pColorAttachments = &colorAttachmentRef;
00215     subpass.pDepthStencilAttachment = &depthAttachmentRef;
00216
00217     VkSubpassDependency dependency = {};
00218     dependency.srcSubpass = VK_SUBPASS_EXTERNAL;
00219     dependency.srcAccessMask = 0;
00220     dependency.srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00221     dependency.dstSubpass = 0;
00222     dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00223     dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00224
00225     const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
00226     VkRenderPassCreateInfo renderPassInfo = {};
00227     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
00228     renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00229     renderPassInfo.pAttachments = attachments.data();
00230     renderPassInfo.subpassCount = 1;
00231     renderPassInfo.pSubpasses = &subpass;
00232     renderPassInfo.dependencyCount = 1;
00233     renderPassInfo.pDependencies = &dependency;
00234
00235     if (vkCreateRenderPass(m_device.device(), &renderPassInfo, nullptr, &m_renderPass) != VK_SUCCESS)
00236     {
00237         throw std::runtime_error("failed to create render pass!");
00238     }
00239
00240 void ven::SwapChain::createFrameBuffers()
00241 {
00242     m_swapChainFrameBuffers.resize(imageCount());
00243     for (size_t i = 0; i < imageCount(); i++) {
00244         std::array<VkImageView, 2> attachments = {m_swapChainImageViews[i], m_depthImageViews[i]};
00245
00246         const auto [width, height] = getSwapChainExtent();
00247         VkFramebufferCreateInfo framebufferInfo = {};
00248         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00249         framebufferInfo.renderPass = m_renderPass;
00250         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00251         framebufferInfo.pAttachments = attachments.data();
00252         framebufferInfo.width = width;
00253         framebufferInfo.height = height;
00254         framebufferInfo.layers = 1;
00255
00256         if (vkCreateFramebuffer(m_device.device(), &framebufferInfo, nullptr,
&m_swapChainFrameBuffers[i]) != VK_SUCCESS) {
00257             throw std::runtime_error("failed to create framebuffer!");
00258         }
00259     }
00260 }
00261
00262 void ven::SwapChain::createDepthResources()
00263 {
00264     const VkFormat depthFormat = findDepthFormat();
00265     const auto [width, height] = getSwapChainExtent();
00266
00267     m_swapChainDepthFormat = depthFormat;
00268     m_depthImages.resize(imageCount());
00269     m_depthImageMemory.resize(imageCount());
00270     m_depthImageViews.resize(imageCount());
00271
00272     for (size_t i = 0; i < m_depthImages.size(); i++) {
00273         VkImageCreateInfo imageInfo{};
00274         imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00275         imageInfo.imageType = VK_IMAGE_TYPE_2D;
00276         imageInfo.extent.width = width;
00277         imageInfo.extent.height = height;
00278         imageInfo.extent.depth = 1;
00279         imageInfo.mipLevels = 1;
00280         imageInfo.arrayLayers = 1;
00281         imageInfo.format = depthFormat;
00282         imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00283         imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00284         imageInfo.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;

```



```

00285         imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00286         imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00287         imageInfo.flags = 0;
00288
00289         m_device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_depthImages[i],
m_depthImageMemory[i]);
00290
00291         VkImageViewCreateInfo viewInfo{};
00292         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00293         viewInfo.image = m_depthImages[i];
00294         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00295         viewInfo.format = depthFormat;
00296         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00297         viewInfo.subresourceRange.baseMipLevel = 0;
00298         viewInfo.subresourceRange.levelCount = 1;
00299         viewInfo.subresourceRange.baseArrayLayer = 0;
00300         viewInfo.subresourceRange.layerCount = 1;
00301
00302         if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_depthImageViews[i]) !=
VK_SUCCESS) {
00303             throw std::runtime_error("failed to create texture image view!");
00304         }
00305     }
00306 }
00307
00308 void ven::SwapChain::createSyncObjects()
00309 {
00310     m_imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00311     m_renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00312     m_inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
00313     m_imagesInFlight.resize(imageCount(), VK_NULL_HANDLE);
00314
00315     VkSemaphoreCreateInfo semaphoreInfo = {};
00316     semaphoreInfo.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
00317
00318     VkFenceCreateInfo fenceInfo = {};
00319     fenceInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
00320     fenceInfo.flags = VK_FENCE_CREATE_SIGNALED_BIT;
00321
00322     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00323         if (vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
&m_imageAvailableSemaphores[i]) != VK_SUCCESS ||
00324             vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
&m_renderFinishedSemaphores[i]) != VK_SUCCESS ||
00325             vkCreateFence(m_device.device(), &fenceInfo, nullptr, &m_inFlightFences[i]) != VK_SUCCESS)
00326         {
00327             throw std::runtime_error("failed to create synchronization objects for a frame!");
00328         }
00329     }
00330 }
00331
00332 VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats)
00333 {
00334     for (const auto &availableFormat : availableFormats) {
00335         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00336             return availableFormat;
00337         }
00338     }
00339     return availableFormats[0];
00340 }
00341
00342 VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes)
00343 {
00344     for (const auto &availablePresentMode : availablePresentModes) {
00345         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00346             std::cout << "Present mode: Mailbox\n";
00347             return availablePresentMode;
00348         }
00349     }
00350
00351     for (const auto &availablePresentMode : availablePresentModes) {
00352         if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00353             std::cout << "Present mode: Immediate" << '\n';
00354             return availablePresentMode;
00355         }
00356     }
00357
00358     std::cout << "Present mode: V-Sync\n";
00359     return VK_PRESENT_MODE_FIFO_KHR;
00360 }
00361
00362 VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00363 {

```


8.90 /home/runner/work/VEngine/VEngine/src/system/pointLight↵ RenderSystem.cpp File Reference

Include dependency graph for pointLightRenderSystem.cpp:



Generated by Doxygen

```

00016         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
00017         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00018     }
00019 }
00020
00021 void ven::PointLightRenderSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
00022 {
00023     const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});
00024     uint16_t lightIndex = 0;
00025
00026     for (Light &light : frameInfo.lights | std::views::values) {
00027         assert(lightIndex < MAX_LIGHTS && "Too many lights");
00028         light.transform3D.translation = glm::vec3(rotateLight *
glm::vec4(light.transform3D.translation, 1.F));
00029         ubo.pointLights.at(lightIndex).position = glm::vec4(light.transform3D.translation, 1.F);
00030         ubo.pointLights.at(lightIndex).color = light.color;
00031         lightIndex++;
00032     }
00033     ubo.numLights = lightIndex;
00034 }

```

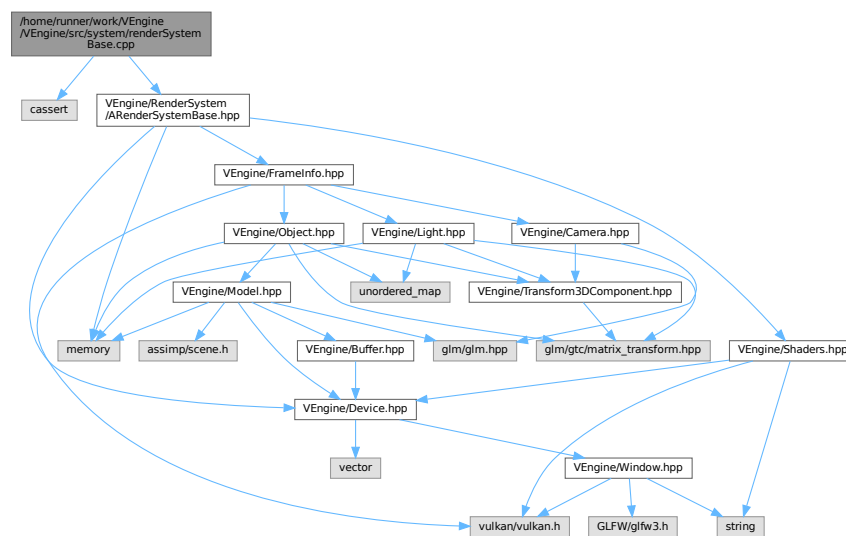
8.92 /home/runner/work/VEngine/VEngine/src/system/renderSystem↩ Base.cpp File Reference

```

#include <cassert>
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"

```

Include dependency graph for renderSystemBase.cpp:



8.93 renderSystemBase.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/RenderSystem/ARenderSystemBase.hpp"
00004
00005 void ven::ARenderSystemBase::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout, const
uint32_t pushConstantSize)
00006 {
00007     VkPushConstantRange pushConstantRange{};
00008     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;

```

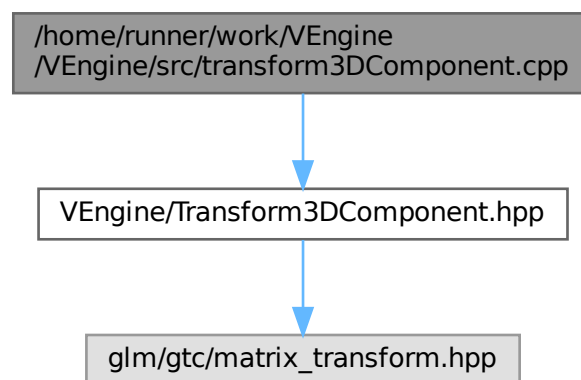
```

00009     pushConstantRange.offset = 0;
00010     pushConstantRange.size = pushConstantSize;
00011
00012     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts(globalSetLayout);
00013
00014     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00015     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00016     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00017     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00018     pipelineLayoutInfo.pushConstantRangeCount = 1;
00019     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00020     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
VK_SUCCESS)
00021     {
00022         throw std::runtime_error("Failed to create pipeline layout");
00023     }
00024 }
00025
00026 void ven::ARenderSystemBase::createPipeline(const VkRenderPass renderPass, const std::string
&shadersVertPath, const std::string &shadersFragPath, const bool isLight)
00027 {
00028     assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
00029     PipelineConfigInfo pipelineConfig{};
00030     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00031     if (isLight) {
00032         pipelineConfig.attributeDescriptions.clear();
00033         pipelineConfig.bindingDescriptions.clear();
00034     }
00035     pipelineConfig.renderPass = renderPass;
00036     pipelineConfig.pipelineLayout = m_pipelineLayout;
00037     m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
00038 }

```

8.94 /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp File Reference

#include "VEngine/Transform3DComponent.hpp"
Include dependency graph for transform3DComponent.cpp:



8.95 transform3DComponent.cpp

[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Transform3DComponent.hpp"
```

```

00002
00003 glm::mat4 ven::Transform3DComponent::mat4() const {
00004     const float c3 = glm::cos(rotation.z);
00005     const float s3 = glm::sin(rotation.z);
00006     const float c2 = glm::cos(rotation.x);
00007     const float s2 = glm::sin(rotation.x);
00008     const float c1 = glm::cos(rotation.y);
00009     const float s1 = glm::sin(rotation.y);
00010     return glm::mat4{
00011         {
00012             scale.x * (c1 * c3 + s1 * s2 * s3),
00013             scale.x * (c2 * s3),
00014             scale.x * (c1 * s2 * s3 - c3 * s1),
00015             0.0F,
00016         },
00017         {
00018             scale.y * (c3 * s1 * s2 - c1 * s3),
00019             scale.y * (c2 * c3),
00020             scale.y * (c1 * c3 * s2 + s1 * s3),
00021             0.0F,
00022         },
00023         {
00024             scale.z * (c2 * s1),
00025             scale.z * (-s2),
00026             scale.z * (c1 * c2),
00027             0.0F,
00028         },
00029         {
00030             translation.x,
00031             translation.y,
00032             translation.z,
00033             1.0F
00034         }
00035     };
00036 }
00037
00038 glm::mat3 ven::Transform3DComponent::normalMatrix() const
00039 {
00040     const float c3 = glm::cos(rotation.z);
00041     const float s3 = glm::sin(rotation.z);
00042     const float c2 = glm::cos(rotation.x);
00043     const float s2 = glm::sin(rotation.x);
00044     const float c1 = glm::cos(rotation.y);
00045     const float s1 = glm::sin(rotation.y);
00046     const glm::vec3 invScale = 1.0F / scale;
00047
00048     return glm::mat3{
00049         {
00050             invScale.x * (c1 * c3 + s1 * s2 * s3),
00051             invScale.x * (c2 * s3),
00052             invScale.x * (c1 * s2 * s3 - c3 * s1)
00053         },
00054         {
00055             invScale.y * (c3 * s1 * s2 - c1 * s3),
00056             invScale.y * (c2 * c3),
00057             invScale.y * (c1 * c3 * s2 + s1 * s3)
00058         },
00059         {
00060             invScale.z * (c2 * s1),
00061             invScale.z * (-s2),
00062             invScale.z * (c1 * c2)
00063         }
00064     };
00065 }

```

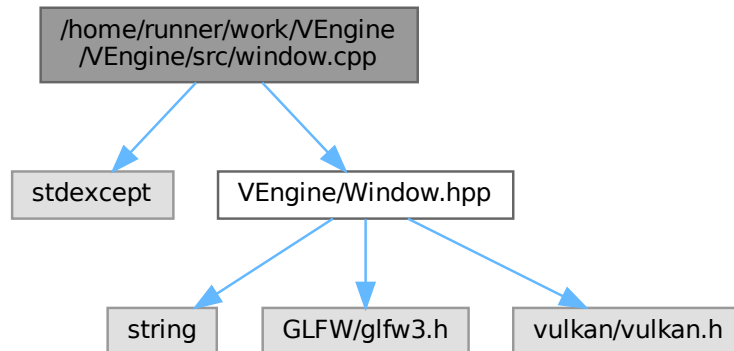
8.96 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

```

#include <stdexcept>
#include "VEngine/Window.hpp"

```

Include dependency graph for window.cpp:



8.97 window.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002
00003 #include "VEngine/Window.hpp"
00004
00005 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string
&title)
00006 {
00007     if (glfwInit() == GLFW_FALSE) {
00008         throw std::runtime_error("Failed to initialize GLFW");
00009     }
00010
00011     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
00012     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00013
00014     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
title.c_str(), nullptr, nullptr);
00015     if (window == nullptr) {
00016         glfwTerminate();
00017         throw std::runtime_error("Failed to create window");
00018     }
00019     glfwSetWindowUserPointer(window, this);
00020     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
00021     return window;
00022 }
00023
00024 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
00025 {
00026     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
00027         throw std::runtime_error("Failed to create window surface");
00028     }
00029 }
00030
00031 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
00032 {
00033     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
00034     app->m_framebufferResized = true;
00035     app->m_width = static_cast<uint32_t>(width);
00036     app->m_height = static_cast<uint32_t>(height);
00037 }
00038
00039 void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
00040 {
00041     GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
00042     const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);
00043
00044     /*
00045     if (fullscreen) {

```

```
00046         glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
mode->refreshRate);
00047     } else {
00048         // To restore a window that was originally windowed to its original size and position,
00049         // save these before making it full screen and then pass them in as above
00050         glfwSetWindowMonitor(m_window, nullptr, 0, 0, static_cast<int>(width),
static_cast<int>(height), mode->refreshRate);
00051
00052     }
00053
00054     m_width = width;
00055     m_height = height;
00056     */
00057 }
```


Index

[/home/runner/work/VEngine/VEngine/README.md](#), [264](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/fragment/pointLight.frag](#), [209](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/fragment/skinned.frag](#), [209](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/vertex/pointLight.vert](#), [210](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/vertex/skinned.vert](#), [211](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp](#), [212, 213](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp](#), [215, 216](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Color.hpp](#), [217, 218](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#), [221, 222](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSet.hpp](#), [223, 224](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp](#), [225, 227](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#), [227, 229](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp](#), [230, 231](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp](#), [232, 234](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Framerate.hpp](#), [234, 236](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp](#), [236, 238](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp](#), [238, 240](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#), [240, 242](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#), [243, 244](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/RenderSystemBase.hpp](#), [247, 248](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp](#), [249, 250](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp](#), [251, 252](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/RenderSystemBase.hpp](#), [244, 246](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Shader.hpp](#), [253, 254](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp](#), [255, 257](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp](#), [258, 260](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp](#), [260, 261](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp](#), [261, 263](#)

[/home/runner/work/VEngine/VEngine/src/buffer.cpp](#), [264](#)

[/home/runner/work/VEngine/VEngine/src/camera.cpp](#), [265, 266](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp](#), [268](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#), [269](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp](#), [270](#)

[/home/runner/work/VEngine/VEngine/src/device.cpp](#), [271, 274](#)

[/home/runner/work/VEngine/VEngine/src/engine.cpp](#), [280](#)

[/home/runner/work/VEngine/VEngine/src/eventManager.cpp](#), [282, 283](#)

[/home/runner/work/VEngine/VEngine/src/gui/init.cpp](#), [284, 285](#)

[/home/runner/work/VEngine/VEngine/src/gui/render.cpp](#), [286](#)

[/home/runner/work/VEngine/VEngine/src/light.cpp](#), [290, 291](#)

[/home/runner/work/VEngine/VEngine/src/main.cpp](#), [291, 292](#)

[/home/runner/work/VEngine/VEngine/src/model.cpp](#), [292, 293](#)

[/home/runner/work/VEngine/VEngine/src/renderer.cpp](#), [296](#)

[/home/runner/work/VEngine/VEngine/src/shaders.cpp](#), [298](#)

[/home/runner/work/VEngine/VEngine/src/swapChain.cpp](#), [301](#)

[/home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp](#), [306](#)

[/home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp](#), [307](#)

[/home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp](#), [308](#)

[/home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp](#), [309](#)

- /home/runner/work/VEngine/VEngine/src/window.cpp, 310, 311
- ~ARenderSystemBase
 - ven::ARenderSystemBase, 23
- ~Buffer
 - ven::Buffer, 30
- ~Camera
 - ven::Camera, 52
- ~DescriptorPool
 - ven::DescriptorPool, 75
- ~DescriptorSetLayout
 - ven::DescriptorSetLayout, 80
- ~DescriptorWriter
 - ven::DescriptorWriter, 84
- ~Device
 - ven::Device, 89
- ~Engine
 - ven::Engine, 105
- ~EventManager
 - ven::EventManager, 112
- ~Gui
 - ven::Gui, 123
- ~Light
 - ven::Light, 137
- ~Model
 - ven::Model, 143
- ~Object
 - ven::Object, 149
- ~Renderer
 - ven::Renderer, 172
- ~Shaders
 - ven::Shaders, 182
- ~SwapChain
 - ven::SwapChain, 189
- ~Window
 - ven::Window, 204
- acquireNextImage
 - ven::SwapChain, 190
- addBinding
 - ven::DescriptorSetLayout::Builder, 46
- addPoolSize
 - ven::DescriptorPool::Builder, 42
- allocateDescriptor
 - ven::DescriptorPool, 76
- ambientLightColor
 - ven::GlobalUbo, 121
- AQUA_3
 - ven::Colors, 63
- AQUA_4
 - ven::Colors, 63
- AQUA_V
 - ven::Colors, 63
- ARenderSystemBase
 - ven::ARenderSystemBase, 23
- attributeDescriptions
 - ven::PipelineConfigInfo, 160
- beginFrame
 - ven::Renderer, 173
- beginSingleTimeCommands
 - ven::Device, 90
- beginSwapChainRenderPass
 - ven::Renderer, 173
- bind
 - ven::Model, 144
 - ven::Shaders, 183
- bindingDescriptions
 - ven::PipelineConfigInfo, 160
- BLACK_3
 - ven::Colors, 63
- BLACK_4
 - ven::Colors, 63
- BLACK_V
 - ven::Colors, 63
- BLUE_3
 - ven::Colors, 63
- BLUE_4
 - ven::Colors, 64
- BLUE_V
 - ven::Colors, 64
- Buffer
 - ven::Buffer, 30
- build
 - ven::DescriptorPool::Builder, 42
 - ven::DescriptorSetLayout::Builder, 46
 - ven::DescriptorWriter, 84
- Builder
 - ven::DescriptorPool::Builder, 42
 - ven::DescriptorSetLayout::Builder, 46
- Camera
 - ven::Camera, 52
- camera
 - ven::FrameInfo, 117
- cameraSection
 - ven::Gui, 124
- capabilities
 - ven::SwapChainSupportDetails, 198
- checkDeviceExtensionSupport
 - ven::Device, 90
- checkValidationLayerSupport
 - ven::Device, 90
- chooseSwapExtent
 - ven::SwapChain, 190
- chooseSwapPresentMode
 - ven::SwapChain, 190
- chooseSwapSurfaceFormat
 - ven::SwapChain, 190
- cleanup
 - ven::Gui, 124
- color
 - ven::Light, 139
 - ven::LightPushConstantData, 141
 - ven::Model::Vertex, 202
 - ven::PointLightData, 163
- COLOR_MAX
 - ven, 16

- COLOR_PRESETS_3
 - ven::Colors, [64](#)
- COLOR_PRESETS_4
 - ven::Colors, [64](#)
- COLOR_PRESETS_VK
 - ven::Colors, [65](#)
- colorBlendAttachment
 - ven::PipelineConfigInfo, [160](#)
- colorBlendInfo
 - ven::PipelineConfigInfo, [160](#)
- commandBuffer
 - ven::FrameInfo, [117](#)
- compareSwapFormats
 - ven::SwapChain, [191](#)
- copyBuffer
 - ven::Device, [90](#)
- copyBufferToImage
 - ven::Device, [91](#)
- createBuffer
 - ven::Device, [91](#)
- createCommandBuffers
 - ven::Renderer, [173](#)
- createCommandPool
 - ven::Device, [91](#)
- CreateDebugUtilsMessengerEXT
 - device.cpp, [272](#)
- createDepthResources
 - ven::SwapChain, [191](#)
- createFrameBuffers
 - ven::SwapChain, [191](#)
- createGraphicsPipeline
 - ven::Shaders, [183](#)
- createImageViews
 - ven::SwapChain, [191](#)
- createImageWithInfo
 - ven::Device, [92](#)
- createIndexBuffer
 - ven::Model, [144](#)
- createInstance
 - ven::Device, [92](#)
 - ven::Engine, [105](#)
- createLight
 - ven::Light, [138](#)
- createLogicalDevice
 - ven::Device, [92](#)
- createModelFromFile
 - ven::Model, [144](#)
- createObject
 - ven::Object, [150](#)
- createPipeline
 - ven::ARenderSystemBase, [24](#)
- createPipelineLayout
 - ven::ARenderSystemBase, [24](#)
- createRenderPass
 - ven::SwapChain, [191](#)
- createShaderModule
 - ven::Shaders, [184](#)
- createSurface
 - ven::Device, [93](#)
 - ven::Engine, [106](#)
- createSwapChain
 - ven::SwapChain, [191](#)
- createSyncObjects
 - ven::SwapChain, [191](#)
- createVertexBuffer
 - ven::Model, [145](#)
- createWindow
 - ven::Window, [205](#)
- createWindowSurface
 - ven::Window, [205](#)
- CYAN_3
 - ven::Colors, [65](#)
- CYAN_4
 - ven::Colors, [65](#)
- CYAN_V
 - ven::Colors, [66](#)
- debugCallback
 - device.cpp, [272](#)
- DEFAULT_AMBIENT_LIGHT_COLOR
 - ven, [16](#)
- DEFAULT_AMBIENT_LIGHT_INTENSITY
 - ven, [16](#)
- DEFAULT_CLEAR_COLOR
 - ven, [16](#)
- DEFAULT_CLEAR_DEPTH
 - ven, [16](#)
- DEFAULT_FAR
 - ven, [16](#)
- DEFAULT_FOV
 - ven, [17](#)
- DEFAULT_HEIGHT
 - ven, [17](#)
- DEFAULT_LIGHT_COLOR
 - ven, [17](#)
- DEFAULT_LIGHT_INTENSITY
 - ven, [17](#)
- DEFAULT_LIGHT_RADIUS
 - ven, [17](#)
- DEFAULT_LOOK_SPEED
 - ven, [17](#)
- DEFAULT_MAX_SETS
 - ven, [17](#)
- DEFAULT_MOVE_SPEED
 - ven, [18](#)
- DEFAULT_NEAR
 - ven, [18](#)
- DEFAULT_POSITION
 - ven, [18](#)
- DEFAULT_ROTATION
 - ven, [18](#)
- DEFAULT_TITLE
 - ven, [18](#)
- DEFAULT_WIDTH
 - ven, [18](#)
- defaultPipelineConfigInfo
 - ven::Shaders, [184](#)

- depthStencilInfo
 - ven::PipelineConfigInfo, 161
- DESCRIPTOR_COUNT
 - init.cpp, 284
- descriptorInfo
 - ven::Buffer, 30
- descriptorInfoForIndex
 - ven::Buffer, 31
- DescriptorPool
 - ven::DescriptorPool, 75, 76
- DescriptorSetLayout
 - ven::DescriptorSetLayout, 80, 81
- DescriptorWriter
 - ven::DescriptorPool, 77
 - ven::DescriptorSetLayout, 81
 - ven::DescriptorWriter, 84
- DestroyDebugUtilsMessengerEXT
 - device.cpp, 273
- Device
 - ven::Device, 89, 90
- device
 - ven::Device, 93
- device.cpp
 - CreateDebugUtilsMessengerEXT, 272
 - debugCallback, 272
 - DestroyDebugUtilsMessengerEXT, 273
- devicePropertiesSection
 - ven::Gui, 125
- dir
 - ven::KeyAction, 132
- draw
 - ven::Model, 146
- dynamicStateEnables
 - ven::PipelineConfigInfo, 161
- dynamicStateInfo
 - ven::PipelineConfigInfo, 161
- EDITOR
 - ven, 15
- enableValidationLayers
 - ven::Device, 100
- endFrame
 - ven::Renderer, 174
- endSingleTimeCommands
 - ven::Device, 94
- endSwapChainRenderPass
 - ven::Renderer, 174
- Engine
 - ven::Engine, 104, 105
- ENGINE_STATE
 - ven, 15
- EventManager
 - ven::EventManager, 112
- EXIT
 - ven, 15
- extentAspectRatio
 - ven::SwapChain, 192
- findDepthFormat
 - ven::SwapChain, 192
- findMemoryType
 - ven::Device, 94
- findPhysicalQueueFamilies
 - ven::Device, 95
- findQueueFamilies
 - ven::Device, 95
- findSupportedFormat
 - ven::Device, 95
- flush
 - ven::Buffer, 31
- flushIndex
 - ven::Buffer, 32
- formats
 - ven::SwapChainSupportDetails, 198
- framebufferResizeCallback
 - ven::Window, 205
- frameIndex
 - ven::FrameInfo, 117
- frameTime
 - ven::FrameInfo, 117
- freeCommandBuffers
 - ven::Renderer, 174
- freeDescriptors
 - ven::DescriptorPool, 76
- FUCHSIA_3
 - ven::Colors, 66
- FUCHSIA_4
 - ven::Colors, 66
- FUCHSIA_V
 - ven::Colors, 66
- GAME
 - ven, 15
- getAlignment
 - ven::Buffer, 33
- getAlignmentSize
 - ven::Buffer, 33
- getAspectRatio
 - ven::Renderer, 175
- getAttributeDescriptions
 - ven::Model::Vertex, 201
- getBindingDescriptions
 - ven::Model::Vertex, 201
- getBuffer
 - ven::Buffer, 33
- getBufferSize
 - ven::Buffer, 34
- getClearColor
 - ven::Renderer, 175
- getCommandPool
 - ven::Device, 96
- getCurrentCommandBuffer
 - ven::Renderer, 175
- getDescriptorPool
 - ven::DescriptorPool, 76
- getDescriptorSetLayout
 - ven::DescriptorSetLayout, 81
- getDevice

- ven::ARenderSystemBase, 25
- getExtent
 - ven::Window, 206
- getFar
 - ven::Camera, 53
- getFov
 - ven::Camera, 53
- getFramebuffer
 - ven::SwapChain, 192
- getFrameIndex
 - ven::Renderer, 176
- getGLFWWindow
 - ven::Window, 206
- getGraphicsQueue
 - ven::Device, 96
- getId
 - ven::Light, 138
 - ven::Object, 150
- getImageView
 - ven::SwapChain, 192
- getInstanceCount
 - ven::Buffer, 34
- getInstanceSize
 - ven::Buffer, 34
- getInverseView
 - ven::Camera, 53
- getLookSpeed
 - ven::Camera, 54
- getMappedMemory
 - ven::Buffer, 34
- getMemoryPropertyFlags
 - ven::Buffer, 34
- getModel
 - ven::Object, 151
- getMoveSpeed
 - ven::Camera, 54
- getName
 - ven::Light, 138
 - ven::Object, 151
- getNear
 - ven::Camera, 54
- getPhysicalDevice
 - ven::Device, 96
- getPipelineLayout
 - ven::ARenderSystemBase, 25
- getProjection
 - ven::Camera, 55
- getRenderPass
 - ven::SwapChain, 192
- getRequiredExtensions
 - ven::Device, 96
- getShaders
 - ven::ARenderSystemBase, 25
- getState
 - ven::Gui, 125
- getSwapChainExtent
 - ven::SwapChain, 192
- getSwapChainImageFormat
 - ven::SwapChain, 193
- getSwapChainRenderPass
 - ven::Renderer, 176
- getSwapChainSupport
 - ven::Device, 97
- getUsageFlags
 - ven::Buffer, 34
- getView
 - ven::Camera, 55
- getWindow
 - ven::Renderer, 177
- GLFW_INCLUDE_VULKAN
 - Window.hpp, 263
- GLM_ENABLE_EXPERIMENTAL
 - model.cpp, 293
- globalDescriptorSet
 - ven::FrameInfo, 117
- graphicsFamily
 - ven::QueueFamilyIndices, 169
- graphicsFamilyHasValue
 - ven::QueueFamilyIndices, 169
- graphicsQueue
 - ven::Device, 97
- GRAY_3
 - ven::Colors, 66
- GRAY_4
 - ven::Colors, 66
- GRAY_V
 - ven::Colors, 66
- GREEN_3
 - ven::Colors, 67
- GREEN_4
 - ven::Colors, 67
- GREEN_V
 - ven::Colors, 67
- Gui
 - ven::Gui, 123
- GUI_STATE
 - ven, 15
- handleEvents
 - ven::EventManager, 112
- hasGlfwRequiredInstanceExtensions
 - ven::Device, 97
- hashCombine
 - ven, 15
- height
 - ven::SwapChain, 193
- HIDDEN
 - ven, 15
- imageCount
 - ven::SwapChain, 193
- indices
 - ven::Model::Builder, 49
- init
 - ven::Gui, 125
 - ven::SwapChain, 193
- init.cpp

- DESCRIPTOR_COUNT, 284
- initStyle
 - ven::Gui, 126
- inputAssemblyInfo
 - ven::PipelineConfigInfo, 161
- inputsSection
 - ven::Gui, 127
- invalidate
 - ven::Buffer, 35
- invalidateIndex
 - ven::Buffer, 35
- inverseView
 - ven::GlobalUbo, 121
- isComplete
 - ven::QueueFamilyIndices, 169
- isDeviceSuitable
 - ven::Device, 97
- isFrameInProgress
 - ven::Renderer, 177
- isKeyJustPressed
 - ven::EventManager, 113
- IsLegacyNativeDupe
 - ven::Gui::funcs, 119
- key
 - ven::KeyAction, 132
- Light
 - ven::Light, 137, 138
- lights
 - ven::FrameInfo, 117
- lightsSection
 - ven::Gui, 127
- LIME_3
 - ven::Colors, 67
- LIME_4
 - ven::Colors, 67
- LIME_V
 - ven::Colors, 67
- loadModel
 - ven::Model::Builder, 49
- loadObjects
 - ven::Engine, 106
- lookDown
 - ven::KeyMappings, 134
- lookLeft
 - ven::KeyMappings, 134
- lookRight
 - ven::KeyMappings, 134
- lookUp
 - ven::KeyMappings, 134
- m_alignmentSize
 - ven::Buffer, 38
- m_bindings
 - ven::DescriptorSetLayout, 82
 - ven::DescriptorSetLayout::Builder, 47
- m_buffer
 - ven::Buffer, 38
- m_bufferSize
 - ven::Buffer, 38
- m_clearValues
 - ven::Renderer, 179
- m_commandBuffers
 - ven::Renderer, 179
- m_commandPool
 - ven::Device, 100
- m_currentFrame
 - ven::SwapChain, 194
- m_currentFrameIndex
 - ven::Renderer, 179
- m_currentImageIndex
 - ven::Renderer, 179
- m_debugMessenger
 - ven::Device, 100
- m_depthImageMemory
 - ven::SwapChain, 194
- m_depthImages
 - ven::SwapChain, 194
- m_depthImageViews
 - ven::SwapChain, 194
- m_descriptorPool
 - ven::DescriptorPool, 77
- m_descriptorSetLayout
 - ven::DescriptorSetLayout, 82
- m_device
 - ven::ARenderSystemBase, 26
 - ven::Buffer, 38
 - ven::DescriptorPool, 77
 - ven::DescriptorPool::Builder, 44
 - ven::DescriptorSetLayout, 82
 - ven::DescriptorSetLayout::Builder, 47
 - ven::Device, 101
 - ven::Engine, 108
 - ven::Model, 146
 - ven::Renderer, 179
 - ven::Shaders, 185
 - ven::SwapChain, 195
- m_deviceExtensions
 - ven::Device, 101
- m_far
 - ven::Camera, 59
- m_fov
 - ven::Camera, 59
- m_fragShaderModule
 - ven::Shaders, 185
- m_framebufferResized
 - ven::Window, 208
- m_globalPool
 - ven::Engine, 108
- m_graphicsPipeline
 - ven::Shaders, 185
- m_graphicsQueue
 - ven::Device, 101
- m_gui
 - ven::Engine, 109
- m_hasIndexBuffer

- ven::Model, 146
- m_height
 - ven::Window, 208
- m_imageAvailableSemaphores
 - ven::SwapChain, 195
- m_imagesInFlight
 - ven::SwapChain, 195
- m_indexBuffer
 - ven::Model, 146
- m_indexCount
 - ven::Model, 146
- m_inFlightFences
 - ven::SwapChain, 195
- m_instance
 - ven::Device, 101
 - ven::Engine, 109
- m_instanceCount
 - ven::Buffer, 39
- m_instanceSize
 - ven::Buffer, 39
- m_inverseViewMatrix
 - ven::Camera, 59
- m_io
 - ven::Gui, 130
- m_isFrameStarted
 - ven::Renderer, 179
- m_keys
 - ven::EventManager, 114
- m_keyState
 - ven::EventManager, 114
- m_lightId
 - ven::Light, 139
- m_lights
 - ven::Engine, 109
- m_lookSpeed
 - ven::Camera, 59
- m_mapped
 - ven::Buffer, 39
- m_maxSets
 - ven::DescriptorPool::Builder, 44
- m_memory
 - ven::Buffer, 39
- m_memoryPropertyFlags
 - ven::Buffer, 39
- m_model
 - ven::Object, 152
- m_moveSpeed
 - ven::Camera, 59
- m_name
 - ven::Light, 139
 - ven::Object, 152
- m_near
 - ven::Camera, 60
- m_objects
 - ven::Engine, 109
- m_objId
 - ven::Object, 153
- m_oldSwapChain
 - ven::SwapChain, 195
- m_physicalDevice
 - ven::Device, 101
- m_pipelineLayout
 - ven::ARenderSystemBase, 26
- m_pool
 - ven::DescriptorWriter, 85
- m_poolFlags
 - ven::DescriptorPool::Builder, 44
- m_poolSizes
 - ven::DescriptorPool::Builder, 44
- m_presentQueue
 - ven::Device, 101
- m_projectionMatrix
 - ven::Camera, 60
- m_properties
 - ven::Device, 102
- m_renderer
 - ven::Engine, 109
- m_renderFinishedSemaphores
 - ven::SwapChain, 195
- m_renderPass
 - ven::SwapChain, 196
- m_setLayout
 - ven::DescriptorWriter, 85
- m_shaders
 - ven::ARenderSystemBase, 26
- m_state
 - ven::Engine, 109
 - ven::Gui, 130
- m_surface
 - ven::Device, 102
 - ven::Engine, 109
- m_swapChain
 - ven::Renderer, 179
 - ven::SwapChain, 196
- m_swapChainDepthFormat
 - ven::SwapChain, 196
- m_swapChainExtent
 - ven::SwapChain, 196
- m_swapChainFrameBuffers
 - ven::SwapChain, 196
- m_swapChainImageFormat
 - ven::SwapChain, 196
- m_swapChainImages
 - ven::SwapChain, 197
- m_swapChainImageViews
 - ven::SwapChain, 197
- m_usageFlags
 - ven::Buffer, 39
- m_validationLayers
 - ven::Device, 102
- m_vertexBuffer
 - ven::Model, 147
- m_vertexCount
 - ven::Model, 147
- m_vertShaderModule
 - ven::Shaders, 185

- m_viewMatrix
 - ven::Camera, 60
- m_width
 - ven::Window, 208
- m_window
 - ven::Device, 102
 - ven::Engine, 110
 - ven::Renderer, 180
 - ven::Window, 208
- m_windowExtent
 - ven::SwapChain, 197
- m_writes
 - ven::DescriptorWriter, 86
- MAGENTA_3
 - ven::Colors, 67
- MAGENTA_4
 - ven::Colors, 68
- MAGENTA_V
 - ven::Colors, 68
- main
 - main.cpp, 291
- main.cpp
 - main, 291
- mainLoop
 - ven::Engine, 107
- Map
 - ven::Light, 137
 - ven::Object, 149
- map
 - ven::Buffer, 36
- MAROON_3
 - ven::Colors, 68
- MAROON_4
 - ven::Colors, 68
- MAROON_V
 - ven::Colors, 68
- mat4
 - ven::Transform3DComponent, 199
- MAX_FRAMES_IN_FLIGHT
 - ven, 18
- MAX_LIGHTS
 - ven, 19
- Model
 - ven::Model, 143, 144
- model.cpp
 - GLM_ENABLE_EXPERIMENTAL, 293
- modelMatrix
 - ven::ObjectPushConstantData, 154
- moveBackward
 - ven::KeyMappings, 134
- moveCamera
 - ven::EventManager, 113
- moveDown
 - ven::KeyMappings, 134
- moveForward
 - ven::KeyMappings, 134
- moveLeft
 - ven::KeyMappings, 134
- moveRight
 - ven::KeyMappings, 135
- moveUp
 - ven::KeyMappings, 135
- multisampleInfo
 - ven::PipelineConfigInfo, 161
- NAVY_3
 - ven::Colors, 68
- NAVY_4
 - ven::Colors, 68
- NAVY_V
 - ven::Colors, 69
- NIGHT_BLUE_3
 - ven::Colors, 69
- NIGHT_BLUE_4
 - ven::Colors, 69
- NIGHT_BLUE_V
 - ven::Colors, 69
- normal
 - ven::Model::Vertex, 202
- normalMatrix
 - ven::ObjectPushConstantData, 154
 - ven::Transform3DComponent, 199
- numLights
 - ven::GlobalUbo, 121
- Object
 - ven::Object, 149, 150
- ObjectRenderSystem
 - ven::ObjectRenderSystem, 157
- objects
 - ven::FrameInfo, 118
- objectsSection
 - ven::Gui, 127
- OLIVE_3
 - ven::Colors, 69
- OLIVE_4
 - ven::Colors, 69
- OLIVE_V
 - ven::Colors, 69
- operator()
 - std::hash< ven::Model::Vertex >, 131
- operator=
 - ven::Buffer, 36
 - ven::Camera, 55
 - ven::DescriptorPool, 77
 - ven::DescriptorSetLayout, 81
 - ven::DescriptorWriter, 84
 - ven::Device, 98
 - ven::Engine, 108
 - ven::EventManager, 114
 - ven::Gui, 127
 - ven::Light, 139
 - ven::Model, 146
 - ven::Object, 151
 - ven::ObjectRenderSystem, 158
 - ven::PipelineConfigInfo, 160
 - ven::PointLightRenderSystem, 167

- ven::Renderer, 177
- ven::Shaders, 185
- ven::SwapChain, 193
- ven::Window, 207
- operator==
 - ven::Model::Vertex, 202
- overwrite
 - ven::DescriptorWriter, 85
- PAUSED
 - ven, 15
- pickPhysicalDevice
 - ven::Device, 98
- PipelineConfigInfo
 - ven::PipelineConfigInfo, 160
- pipelineLayout
 - ven::PipelineConfigInfo, 161
- PointLightRenderSystem
 - ven::PointLightRenderSystem, 166
- pointLights
 - ven::GlobalUbo, 121
- populateDebugMessengerCreateInfo
 - ven::Device, 98
- position
 - ven::LightPushConstantData, 141
 - ven::Model::Vertex, 202
 - ven::PointLightData, 163
- presentFamily
 - ven::QueueFamilyIndices, 169
- presentFamilyHasValue
 - ven::QueueFamilyIndices, 170
- presentModes
 - ven::SwapChainSupportDetails, 198
- presentQueue
 - ven::Device, 99
- processMesh
 - ven::Model::Builder, 49
- processNode
 - ven::Model::Builder, 49
- projection
 - ven::GlobalUbo, 121
- querySwapChainSupport
 - ven::Device, 99
- radius
 - ven::LightPushConstantData, 141
- rasterizationInfo
 - ven::PipelineConfigInfo, 162
- readFile
 - ven::Shaders, 185
- recreateSwapChain
 - ven::Renderer, 178
- RED_3
 - ven::Colors, 70
- RED_4
 - ven::Colors, 70
- RED_V
 - ven::Colors, 70
- render
 - ven::ARenderSystemBase, 26
 - ven::Gui, 127
 - ven::ObjectRenderSystem, 158
 - ven::PointLightRenderSystem, 167
- Renderer
 - ven::Renderer, 172, 173
- rendererSection
 - ven::Gui, 128
- renderFrameWindow
 - ven::Gui, 129
- renderPass
 - ven::PipelineConfigInfo, 162
- resetPool
 - ven::DescriptorPool, 77
- resetWindowResizedFlag
 - ven::Window, 207
- rotation
 - ven::Transform3DComponent, 200
- scale
 - ven::Transform3DComponent, 200
- setClearValue
 - ven::Renderer, 178
- setFar
 - ven::Camera, 55
- setFov
 - ven::Camera, 56
- setFullscreen
 - ven::Window, 207
- setLookSpeed
 - ven::Camera, 56
- setMaxSets
 - ven::DescriptorPool::Builder, 43
- setModel
 - ven::Object, 151
- setMoveSpeed
 - ven::Camera, 57
- setName
 - ven::Light, 139
 - ven::Object, 152
- setNear
 - ven::Camera, 57
- setOrthographicProjection
 - ven::Camera, 58
- setPerspectiveProjection
 - ven::Camera, 58
- setPoolFlags
 - ven::DescriptorPool::Builder, 43
- setState
 - ven::Gui, 129
- setupDebugMessenger
 - ven::Device, 99
- setViewDirection
 - ven::Camera, 58
- setViewTarget
 - ven::Camera, 58
- setViewXYZ
 - ven::Camera, 59

- Shaders
 - ven::Shaders, [182](#), [183](#)
- SHADERS_BIN_PATH
 - ven, [19](#)
- SILVER_3
 - ven::Colors, [70](#)
- SILVER_4
 - ven::Colors, [70](#)
- SILVER_V
 - ven::Colors, [70](#)
- SKY_BLUE_3
 - ven::Colors, [70](#)
- SKY_BLUE_4
 - ven::Colors, [71](#)
- SKY_BLUE_V
 - ven::Colors, [71](#)
- std::hash< ven::Model::Vertex >, [130](#)
 - operator(), [131](#)
- submitCommandBuffers
 - ven::SwapChain, [194](#)
- subpass
 - ven::PipelineConfigInfo, [162](#)
- SUNSET_3
 - ven::Colors, [71](#)
- SUNSET_4
 - ven::Colors, [71](#)
- SUNSET_V
 - ven::Colors, [71](#)
- surface
 - ven::Device, [100](#)
- SwapChain
 - ven::SwapChain, [189](#), [190](#)
- TEAL_3
 - ven::Colors, [71](#)
- TEAL_4
 - ven::Colors, [71](#)
- TEAL_V
 - ven::Colors, [72](#)
- toggleGui
 - ven::KeyMappings, [135](#)
- transform3D
 - ven::Camera, [60](#)
 - ven::Light, [140](#)
 - ven::Object, [153](#)
- translation
 - ven::Transform3DComponent, [200](#)
- unmap
 - ven::Buffer, [37](#)
- update
 - ven::PointLightRenderSystem, [167](#)
- updateEngineState
 - ven::EventManager, [114](#)
- uv
 - ven::Model::Vertex, [203](#)
- value
 - ven::KeyAction, [132](#)
- ven, [13](#)
 - COLOR_MAX, [16](#)
 - DEFAULT_AMBIENT_LIGHT_COLOR, [16](#)
 - DEFAULT_AMBIENT_LIGHT_INTENSITY, [16](#)
 - DEFAULT_CLEAR_COLOR, [16](#)
 - DEFAULT_CLEAR_DEPTH, [16](#)
 - DEFAULT_FAR, [16](#)
 - DEFAULT_FOV, [17](#)
 - DEFAULT_HEIGHT, [17](#)
 - DEFAULT_LIGHT_COLOR, [17](#)
 - DEFAULT_LIGHT_INTENSITY, [17](#)
 - DEFAULT_LIGHT_RADIUS, [17](#)
 - DEFAULT_LOOK_SPEED, [17](#)
 - DEFAULT_MAX_SETS, [17](#)
 - DEFAULT_MOVE_SPEED, [18](#)
 - DEFAULT_NEAR, [18](#)
 - DEFAULT_POSITION, [18](#)
 - DEFAULT_ROTATION, [18](#)
 - DEFAULT_TITLE, [18](#)
 - DEFAULT_WIDTH, [18](#)
 - EDITOR, [15](#)
 - ENGINE_STATE, [15](#)
 - EXIT, [15](#)
 - GAME, [15](#)
 - GUI_STATE, [15](#)
 - hashCombine, [15](#)
 - HIDDEN, [15](#)
 - MAX_FRAMES_IN_FLIGHT, [18](#)
 - MAX_LIGHTS, [19](#)
 - PAUSED, [15](#)
 - SHADERS_BIN_PATH, [19](#)
 - VISIBLE, [15](#)
- ven::ARenderSystemBase, [21](#)
 - ~ARenderSystemBase, [23](#)
 - ARenderSystemBase, [23](#)
 - createPipeline, [24](#)
 - createPipelineLayout, [24](#)
 - getDevice, [25](#)
 - getPipelineLayout, [25](#)
 - getShaders, [25](#)
 - m_device, [26](#)
 - m_pipelineLayout, [26](#)
 - m_shaders, [26](#)
 - render, [26](#)
- ven::Buffer, [27](#)
 - ~Buffer, [30](#)
 - Buffer, [30](#)
 - descriptorInfo, [30](#)
 - descriptorInfoForIndex, [31](#)
 - flush, [31](#)
 - flushIndex, [32](#)
 - getAlignment, [33](#)
 - getAlignmentSize, [33](#)
 - getBuffer, [33](#)
 - getBufferSize, [34](#)
 - getInstanceCount, [34](#)
 - getInstanceSize, [34](#)
 - getMappedMemory, [34](#)

- getMemoryPropertyFlags, 34
- getUsageFlags, 34
- invalidate, 35
- invalidateIndex, 35
- m_alignmentSize, 38
- m_buffer, 38
- m_bufferSize, 38
- m_device, 38
- m_instanceCount, 39
- m_instanceSize, 39
- m_mapped, 39
- m_memory, 39
- m_memoryPropertyFlags, 39
- m_usageFlags, 39
- map, 36
- operator=, 36
- unmap, 37
- writeToBuffer, 37
- writeToIndex, 37
- ven::Camera, 50
 - ~Camera, 52
 - Camera, 52
 - getFar, 53
 - getFov, 53
 - getInverseView, 53
 - getLookSpeed, 54
 - getMoveSpeed, 54
 - getNear, 54
 - getProjection, 55
 - getView, 55
 - m_far, 59
 - m_fov, 59
 - m_inverseViewMatrix, 59
 - m_lookSpeed, 59
 - m_moveSpeed, 59
 - m_near, 60
 - m_projectionMatrix, 60
 - m_viewMatrix, 60
 - operator=, 55
 - setFar, 55
 - setFov, 56
 - setLookSpeed, 56
 - setMoveSpeed, 57
 - setNear, 57
 - setOrthographicProjection, 58
 - setPerspectiveProjection, 58
 - setViewDirection, 58
 - setViewTarget, 58
 - setViewXYZ, 59
 - transform3D, 60
- ven::Colors, 61
 - AQUA_3, 63
 - AQUA_4, 63
 - AQUA_V, 63
 - BLACK_3, 63
 - BLACK_4, 63
 - BLACK_V, 63
 - BLUE_3, 63
 - BLUE_4, 64
 - BLUE_V, 64
 - COLOR_PRESETS_3, 64
 - COLOR_PRESETS_4, 64
 - COLOR_PRESETS_VK, 65
 - CYAN_3, 65
 - CYAN_4, 65
 - CYAN_V, 66
 - FUCHSIA_3, 66
 - FUCHSIA_4, 66
 - FUCHSIA_V, 66
 - GRAY_3, 66
 - GRAY_4, 66
 - GRAY_V, 66
 - GREEN_3, 67
 - GREEN_4, 67
 - GREEN_V, 67
 - LIME_3, 67
 - LIME_4, 67
 - LIME_V, 67
 - MAGENTA_3, 67
 - MAGENTA_4, 68
 - MAGENTA_V, 68
 - MAROON_3, 68
 - MAROON_4, 68
 - MAROON_V, 68
 - NAVY_3, 68
 - NAVY_4, 68
 - NAVY_V, 69
 - NIGHT_BLUE_3, 69
 - NIGHT_BLUE_4, 69
 - NIGHT_BLUE_V, 69
 - OLIVE_3, 69
 - OLIVE_4, 69
 - OLIVE_V, 69
 - RED_3, 70
 - RED_4, 70
 - RED_V, 70
 - SILVER_3, 70
 - SILVER_4, 70
 - SILVER_V, 70
 - SKY_BLUE_3, 70
 - SKY_BLUE_4, 71
 - SKY_BLUE_V, 71
 - SUNSET_3, 71
 - SUNSET_4, 71
 - SUNSET_V, 71
 - TEAL_3, 71
 - TEAL_4, 71
 - TEAL_V, 72
 - WHITE_3, 72
 - WHITE_4, 72
 - WHITE_V, 72
 - YELLOW_3, 72
 - YELLOW_4, 72
 - YELLOW_V, 72
- ven::DescriptorPool, 73
 - ~DescriptorPool, 75

- allocateDescriptor, 76
- DescriptorPool, 75, 76
- DescriptorWriter, 77
- freeDescriptors, 76
- getDescriptorPool, 76
- m_descriptorPool, 77
- m_device, 77
- operator=, 77
- resetPool, 77
- ven::DescriptorPool::Builder, 40
 - addPoolSize, 42
 - build, 42
 - Builder, 42
 - m_device, 44
 - m_maxSets, 44
 - m_poolFlags, 44
 - m_poolSizes, 44
 - setMaxSets, 43
 - setPoolFlags, 43
- ven::DescriptorSetLayout, 78
 - ~DescriptorSetLayout, 80
 - DescriptorSetLayout, 80, 81
 - DescriptorWriter, 81
 - getDescriptorSetLayout, 81
 - m_bindings, 82
 - m_descriptorSetLayout, 82
 - m_device, 82
 - operator=, 81
- ven::DescriptorSetLayout::Builder, 45
 - addBinding, 46
 - build, 46
 - Builder, 46
 - m_bindings, 47
 - m_device, 47
- ven::DescriptorWriter, 82
 - ~DescriptorWriter, 84
 - build, 84
 - DescriptorWriter, 84
 - m_pool, 85
 - m_setLayout, 85
 - m_writes, 86
 - operator=, 84
 - overwrite, 85
 - writeBuffer, 85
 - writImage, 85
- ven::Device, 86
 - ~Device, 89
 - beginSingleTimeCommands, 90
 - checkDeviceExtensionSupport, 90
 - checkValidationLayerSupport, 90
 - copyBuffer, 90
 - copyBufferToImage, 91
 - createBuffer, 91
 - createCommandPool, 91
 - createImageWithInfo, 92
 - createInstance, 92
 - createLogicalDevice, 92
 - createSurface, 93
 - Device, 89, 90
 - device, 93
 - enableValidationLayers, 100
 - endSingleTimeCommands, 94
 - findMemoryType, 94
 - findPhysicalQueueFamilies, 95
 - findQueueFamilies, 95
 - findSupportedFormat, 95
 - getCommandPool, 96
 - getGraphicsQueue, 96
 - getPhysicalDevice, 96
 - getRequiredExtensions, 96
 - getSwapChainSupport, 97
 - graphicsQueue, 97
 - hasGlfwRequiredInstanceExtensions, 97
 - isDeviceSuitable, 97
 - m_commandPool, 100
 - m_debugMessenger, 100
 - m_device, 101
 - m_deviceExtensions, 101
 - m_graphicsQueue, 101
 - m_instance, 101
 - m_physicalDevice, 101
 - m_presentQueue, 101
 - m_properties, 102
 - m_surface, 102
 - m_validationLayers, 102
 - m_window, 102
 - operator=, 98
 - pickPhysicalDevice, 98
 - populateDebugMessengerCreateInfo, 98
 - presentQueue, 99
 - querySwapChainSupport, 99
 - setupDebugMessenger, 99
 - surface, 100
- ven::Engine, 103
 - ~Engine, 105
 - createInstance, 105
 - createSurface, 106
 - Engine, 104, 105
 - loadObjects, 106
 - m_device, 108
 - m_globalPool, 108
 - m_gui, 109
 - m_instance, 109
 - m_lights, 109
 - m_objects, 109
 - m_renderer, 109
 - m_state, 109
 - m_surface, 109
 - m_window, 110
 - mainLoop, 107
 - operator=, 108
- ven::EventManager, 110
 - ~EventManager, 112
 - EventManager, 112
 - handleEvents, 112
 - isKeyJustPressed, 113

- [m_keys](#), 114
 - [m_keyState](#), 114
 - [moveCamera](#), 113
 - [operator=](#), 114
 - [updateEngineState](#), 114
- [ven::FrameInfo](#), 115
 - [camera](#), 117
 - [commandBuffer](#), 117
 - [frameIndex](#), 117
 - [frameTime](#), 117
 - [globalDescriptorSet](#), 117
 - [lights](#), 117
 - [objects](#), 118
- [ven::GlobalUbo](#), 120
 - [ambientLightColor](#), 121
 - [inverseView](#), 121
 - [numLights](#), 121
 - [pointLights](#), 121
 - [projection](#), 121
 - [view](#), 121
- [ven::Gui](#), 122
 - [~Gui](#), 123
 - [cameraSection](#), 124
 - [cleanup](#), 124
 - [devicePropertiesSection](#), 125
 - [getState](#), 125
 - [Gui](#), 123
 - [init](#), 125
 - [initStyle](#), 126
 - [inputsSection](#), 127
 - [lightsSection](#), 127
 - [m_io](#), 130
 - [m_state](#), 130
 - [objectsSection](#), 127
 - [operator=](#), 127
 - [render](#), 127
 - [rendererSection](#), 128
 - [renderFrameWindow](#), 129
 - [setState](#), 129
- [ven::Gui::funcs](#), 118
 - [IsLegacyNativeDupe](#), 119
- [ven::KeyAction](#), 131
 - [dir](#), 132
 - [key](#), 132
 - [value](#), 132
- [ven::KeyMappings](#), 133
 - [lookDown](#), 134
 - [lookLeft](#), 134
 - [lookRight](#), 134
 - [lookUp](#), 134
 - [moveBackward](#), 134
 - [moveDown](#), 134
 - [moveForward](#), 134
 - [moveLeft](#), 134
 - [moveRight](#), 135
 - [moveUp](#), 135
 - [toggleGui](#), 135
- [ven::Light](#), 135
 - [~Light](#), 137
 - [color](#), 139
 - [createLight](#), 138
 - [getId](#), 138
 - [getName](#), 138
 - [Light](#), 137, 138
 - [m_lightId](#), 139
 - [m_name](#), 139
 - [Map](#), 137
 - [operator=](#), 139
 - [setName](#), 139
 - [transform3D](#), 140
- [ven::LightPushConstantData](#), 140
 - [color](#), 141
 - [position](#), 141
 - [radius](#), 141
- [ven::Model](#), 141
 - [~Model](#), 143
 - [bind](#), 144
 - [createIndexBuffer](#), 144
 - [createModelFromFile](#), 144
 - [createVertexBuffer](#), 145
 - [draw](#), 146
 - [m_device](#), 146
 - [m_hasIndexBuffer](#), 146
 - [m_indexBuffer](#), 146
 - [m_indexCount](#), 146
 - [m_vertexBuffer](#), 147
 - [m_vertexCount](#), 147
 - [Model](#), 143, 144
 - [operator=](#), 146
- [ven::Model::Builder](#), 48
 - [indices](#), 49
 - [loadModel](#), 49
 - [processMesh](#), 49
 - [processNode](#), 49
 - [vertices](#), 49
- [ven::Model::Vertex](#), 200
 - [color](#), 202
 - [getAttributeDescriptions](#), 201
 - [getBindingDescriptions](#), 201
 - [normal](#), 202
 - [operator==](#), 202
 - [position](#), 202
 - [uv](#), 203
- [ven::Object](#), 147
 - [~Object](#), 149
 - [createObject](#), 150
 - [getId](#), 150
 - [getModel](#), 151
 - [getName](#), 151
 - [m_model](#), 152
 - [m_name](#), 152
 - [m_objId](#), 153
 - [Map](#), 149
 - [Object](#), 149, 150
 - [operator=](#), 151
 - [setModel](#), 151

- setName, 152
 - transform3D, 153
- ven::ObjectPushConstantData, 153
 - modelMatrix, 154
 - normalMatrix, 154
- ven::ObjectRenderSystem, 154
 - ObjectRenderSystem, 157
 - operator=, 158
 - render, 158
- ven::PipelineConfigInfo, 159
 - attributeDescriptions, 160
 - bindingDescriptions, 160
 - colorBlendAttachment, 160
 - colorBlendInfo, 160
 - depthStencilInfo, 161
 - dynamicStateEnables, 161
 - dynamicStateInfo, 161
 - inputAssemblyInfo, 161
 - multisampleInfo, 161
 - operator=, 160
 - PipelineConfigInfo, 160
 - pipelineLayout, 161
 - rasterizationInfo, 162
 - renderPass, 162
 - subpass, 162
- ven::PointLightData, 162
 - color, 163
 - position, 163
- ven::PointLightRenderSystem, 163
 - operator=, 167
 - PointLightRenderSystem, 166
 - render, 167
 - update, 167
- ven::QueueFamilyIndices, 168
 - graphicsFamily, 169
 - graphicsFamilyHasValue, 169
 - isComplete, 169
 - presentFamily, 169
 - presentFamilyHasValue, 170
- ven::Renderer, 170
 - ~Renderer, 172
 - beginFrame, 173
 - beginSwapChainRenderPass, 173
 - createCommandBuffers, 173
 - endFrame, 174
 - endSwapChainRenderPass, 174
 - freeCommandBuffers, 174
 - getAspectRatio, 175
 - getClearColor, 175
 - getCurrentCommandBuffer, 175
 - getFrameIndex, 176
 - getSwapChainRenderPass, 176
 - getWindow, 177
 - isFrameInProgress, 177
 - m_clearValues, 179
 - m_commandBuffers, 179
 - m_currentFrameIndex, 179
 - m_currentImageIndex, 179
 - m_device, 179
 - m_isFrameStarted, 179
 - m_swapChain, 179
 - m_window, 180
 - operator=, 177
 - recreateSwapChain, 178
 - Renderer, 172, 173
 - setClearValue, 178
- ven::Shaders, 180
 - ~Shaders, 182
 - bind, 183
 - createGraphicsPipeline, 183
 - createShaderModule, 184
 - defaultPipelineConfigInfo, 184
 - m_device, 185
 - m_fragShaderModule, 185
 - m_graphicsPipeline, 185
 - m_vertShaderModule, 185
 - operator=, 185
 - readFile, 185
 - Shaders, 182, 183
- ven::SwapChain, 186
 - ~SwapChain, 189
 - acquireNextImage, 190
 - chooseSwapExtent, 190
 - chooseSwapPresentMode, 190
 - chooseSwapSurfaceFormat, 190
 - compareSwapFormats, 191
 - createDepthResources, 191
 - createFrameBuffers, 191
 - createImageViews, 191
 - createRenderPass, 191
 - createSwapChain, 191
 - createSyncObjects, 191
 - extentAspectRatio, 192
 - findDepthFormat, 192
 - getFrameBuffer, 192
 - getImageView, 192
 - getRenderPass, 192
 - getSwapChainExtent, 192
 - getSwapChainImageFormat, 193
 - height, 193
 - imageCount, 193
 - init, 193
 - m_currentFrame, 194
 - m_depthImageMemory, 194
 - m_depthImages, 194
 - m_depthImageViews, 194
 - m_device, 195
 - m_imageAvailableSemaphores, 195
 - m_imagesInFlight, 195
 - m_inFlightFences, 195
 - m_oldSwapChain, 195
 - m_renderFinishedSemaphores, 195
 - m_renderPass, 196
 - m_swapChain, 196
 - m_swapChainDepthFormat, 196
 - m_swapChainExtent, 196

- [m_swapChainFrameBuffers](#), [196](#)
 - [m_swapChainImageFormat](#), [196](#)
 - [m_swapChainImages](#), [197](#)
 - [m_swapChainImageViews](#), [197](#)
 - [m_windowExtent](#), [197](#)
 - [operator=](#), [193](#)
 - [submitCommandBuffers](#), [194](#)
 - [SwapChain](#), [189](#), [190](#)
 - [width](#), [194](#)
- [ven::SwapChainSupportDetails](#), [197](#)
 - [capabilities](#), [198](#)
 - [formats](#), [198](#)
 - [presentModes](#), [198](#)
- [ven::Transform3DComponent](#), [199](#)
 - [mat4](#), [199](#)
 - [normalMatrix](#), [199](#)
 - [rotation](#), [200](#)
 - [scale](#), [200](#)
 - [translation](#), [200](#)
- [ven::Window](#), [203](#)
 - [~Window](#), [204](#)
 - [createWindow](#), [205](#)
 - [createWindowSurface](#), [205](#)
 - [framebufferResizeCallback](#), [205](#)
 - [getExtent](#), [206](#)
 - [getGLFWWindow](#), [206](#)
 - [m_framebufferResized](#), [208](#)
 - [m_height](#), [208](#)
 - [m_width](#), [208](#)
 - [m_window](#), [208](#)
 - [operator=](#), [207](#)
 - [resetWindowResizedFlag](#), [207](#)
 - [setFullscreen](#), [207](#)
 - [wasWindowResized](#), [207](#)
 - [Window](#), [204](#)
- [vengine](#), [1](#)
- [vertices](#)
 - [ven::Model::Builder](#), [49](#)
- [view](#)
 - [ven::GlobalUbo](#), [121](#)
- [VISIBLE](#)
 - [ven](#), [15](#)
- [wasWindowResized](#)
 - [ven::Window](#), [207](#)
- [WHITE_3](#)
 - [ven::Colors](#), [72](#)
- [WHITE_4](#)
 - [ven::Colors](#), [72](#)
- [WHITE_V](#)
 - [ven::Colors](#), [72](#)
- [width](#)
 - [ven::SwapChain](#), [194](#)
- [Window](#)
 - [ven::Window](#), [204](#)
- [Window.hpp](#)
 - [GLFW_INCLUDE_VULKAN](#), [263](#)
- [writeBuffer](#)
 - [ven::DescriptorWriter](#), [85](#)
- [writeImage](#)
 - [ven::DescriptorWriter](#), [85](#)
- [writeToBuffer](#)
 - [ven::Buffer](#), [37](#)
- [writeToIndex](#)
 - [ven::Buffer](#), [37](#)
- [YELLOW_3](#)
 - [ven::Colors](#), [72](#)
- [YELLOW_4](#)
 - [ven::Colors](#), [72](#)
- [YELLOW_V](#)
 - [ven::Colors](#), [72](#)