

vengine

0.1.0

Generated by Doxygen 1.11.0



<b>1 vengine</b>	<b>1</b>
1.1 Description	1
1.2 Prerequisites	1
1.3 Usage	1
1.3.1 Build	1
1.3.2 Run	1
1.3.3 Key Bindings	1
1.3.4 Documentation	2
1.4 Commit Norms	2
1.5 License	2
1.6 Acknowledgements	2
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 myLib Namespace Reference	9
5.1.1 Variable Documentation	9
5.1.1.1 MICROSECONDS_PER_SECOND	9
5.1.1.2 MILLISECONDS_PER_SECOND	9
5.1.1.3 RANDOM_FLOAT_MAX	10
5.1.1.4 RANDOM_INT_MAX	10
5.1.1.5 RANDOM_INT_MIN	10
5.2 std Namespace Reference	10
5.2.1 Detailed Description	14
5.3 ven Namespace Reference	14
5.3.1 Function Documentation	16
5.3.1.1 hashCombine()	16
5.3.2 Variable Documentation	16
5.3.2.1 DEFAULT_CLEAR_COLOR	16
5.3.2.2 DEFAULT_CLEAR_DEPTH	17
5.3.2.3 DEFAULT_FAR	17
5.3.2.4 DEFAULT_FOV	17
5.3.2.5 DEFAULT_HEIGHT	17
5.3.2.6 DEFAULT_LIGHT_COLOR	17
5.3.2.7 DEFAULT_LIGHT_INTENSITY	17
5.3.2.8 DEFAULT_LIGHT_RADIUS	17
5.3.2.9 DEFAULT_LOOK_SPEED	18

5.3.2.10	DEFAULT_MOVE_SPEED	18
5.3.2.11	DEFAULT_NEAR	18
5.3.2.12	DEFAULT_POSITION	18
5.3.2.13	DEFAULT_ROTATION	18
5.3.2.14	DEFAULT_TITLE	18
5.3.2.15	DEFAULT_WIDTH	19
5.3.2.16	MAX_LIGHTS	19
5.3.2.17	SHADERS_BIN_PATH	19
<b>6</b>	<b>Class Documentation</b>	<b>21</b>
6.1	ven::Buffer Class Reference	21
6.1.1	Detailed Description	24
6.1.2	Constructor & Destructor Documentation	24
6.1.2.1	Buffer() [1/2]	24
6.1.2.2	~Buffer()	24
6.1.2.3	Buffer() [2/2]	24
6.1.3	Member Function Documentation	24
6.1.3.1	descriptorInfo()	24
6.1.3.2	descriptorInfoForIndex()	25
6.1.3.3	flush()	26
6.1.3.4	flushIndex()	26
6.1.3.5	getAlignment()	27
6.1.3.6	getAlignmentSize()	27
6.1.3.7	getBuffer()	28
6.1.3.8	getBufferSize()	28
6.1.3.9	getInstanceCount()	28
6.1.3.10	getInstanceSize()	28
6.1.3.11	getMappedMemory()	28
6.1.3.12	getMemoryPropertyFlags()	28
6.1.3.13	getUsageFlags()	29
6.1.3.14	invalidate()	29
6.1.3.15	invalidateIndex()	29
6.1.3.16	map()	30
6.1.3.17	operator=()	31
6.1.3.18	unmap()	31
6.1.3.19	writeToBuffer()	31
6.1.3.20	writeToIndex()	31
6.1.4	Member Data Documentation	32
6.1.4.1	m_alignmentSize	32
6.1.4.2	m_buffer	32
6.1.4.3	m_bufferSize	32
6.1.4.4	m_device	32

6.1.4.5 m_instanceCount . . . . .	33
6.1.4.6 m_instanceSize . . . . .	33
6.1.4.7 m_mapped . . . . .	33
6.1.4.8 m_memory . . . . .	33
6.1.4.9 m_memoryPropertyFlags . . . . .	33
6.1.4.10 m_usageFlags . . . . .	33
6.2 ven::DescriptorPool::Builder Class Reference . . . . .	34
6.2.1 Detailed Description . . . . .	35
6.2.2 Constructor & Destructor Documentation . . . . .	35
6.2.2.1 Builder() . . . . .	35
6.2.3 Member Function Documentation . . . . .	35
6.2.3.1 addPoolSize() . . . . .	35
6.2.3.2 build() . . . . .	36
6.2.3.3 setMaxSets() . . . . .	36
6.2.3.4 setPoolFlags() . . . . .	36
6.2.4 Member Data Documentation . . . . .	37
6.2.4.1 m_device . . . . .	37
6.2.4.2 m_maxSets . . . . .	37
6.2.4.3 m_poolFlags . . . . .	37
6.2.4.4 m_poolSizes . . . . .	37
6.3 ven::DescriptorSetLayout::Builder Class Reference . . . . .	38
6.3.1 Detailed Description . . . . .	39
6.3.2 Constructor & Destructor Documentation . . . . .	39
6.3.2.1 Builder() . . . . .	39
6.3.3 Member Function Documentation . . . . .	39
6.3.3.1 addBinding() . . . . .	39
6.3.3.2 build() . . . . .	40
6.3.4 Member Data Documentation . . . . .	40
6.3.4.1 m_bindings . . . . .	40
6.3.4.2 m_device . . . . .	40
6.4 ven::Model::Builder Struct Reference . . . . .	41
6.4.1 Detailed Description . . . . .	41
6.4.2 Member Function Documentation . . . . .	42
6.4.2.1 loadModel() . . . . .	42
6.4.3 Member Data Documentation . . . . .	42
6.4.3.1 indices . . . . .	42
6.4.3.2 vertices . . . . .	42
6.5 ven::Camera Class Reference . . . . .	43
6.5.1 Detailed Description . . . . .	44
6.5.2 Member Function Documentation . . . . .	44
6.5.2.1 getFar() . . . . .	44
6.5.2.2 getFov() . . . . .	44

6.5.2.3 <a href="#">getInverseView()</a>	45
6.5.2.4 <a href="#">getNear()</a>	45
6.5.2.5 <a href="#">getProjection()</a>	45
6.5.2.6 <a href="#">getView()</a>	45
6.5.2.7 <a href="#">setFar()</a>	46
6.5.2.8 <a href="#">setFov()</a>	46
6.5.2.9 <a href="#">setNear()</a>	46
6.5.2.10 <a href="#">setOrthographicProjection()</a>	47
6.5.2.11 <a href="#">setPerspectiveProjection()</a>	47
6.5.2.12 <a href="#">setViewDirection()</a>	47
6.5.2.13 <a href="#">setViewTarget()</a>	47
6.5.2.14 <a href="#">setViewXYZ()</a>	47
6.5.3 Member Data Documentation	48
6.5.3.1 <a href="#">m_far</a>	48
6.5.3.2 <a href="#">m_fov</a>	48
6.5.3.3 <a href="#">m_inverseViewMatrix</a>	48
6.5.3.4 <a href="#">m_near</a>	48
6.5.3.5 <a href="#">m_projectionMatrix</a>	48
6.5.3.6 <a href="#">m_viewMatrix</a>	48
6.6 <a href="#">myLib::Clock Class Reference</a>	49
6.6.1 Detailed Description	49
6.6.2 Constructor & Destructor Documentation	50
6.6.2.1 <a href="#">Clock()</a>	50
6.6.2.2 <a href="#">~Clock()</a>	50
6.6.3 Member Function Documentation	50
6.6.3.1 <a href="#">getElapsedTime()</a>	50
6.6.3.2 <a href="#">pause()</a>	50
6.6.3.3 <a href="#">restart()</a>	50
6.6.3.4 <a href="#">resume()</a>	51
6.6.4 Member Data Documentation	51
6.6.4.1 <a href="#">m_pause</a>	51
6.6.4.2 <a href="#">m_paused</a>	51
6.6.4.3 <a href="#">m_start</a>	51
6.7 <a href="#">ven::Colors Class Reference</a>	52
6.7.1 Detailed Description	53
6.7.2 Member Data Documentation	54
6.7.2.1 <a href="#">AQUA</a>	54
6.7.2.2 <a href="#">AQUA_V</a>	54
6.7.2.3 <a href="#">BLACK</a>	54
6.7.2.4 <a href="#">BLACK_V</a>	54
6.7.2.5 <a href="#">BLUE</a>	54
6.7.2.6 <a href="#">BLUE_V</a>	54

6.7.2.7 CLEAR_COLORS	55
6.7.2.8 COLOR_MAX	55
6.7.2.9 COLORS	55
6.7.2.10 CYAN	56
6.7.2.11 CYAN_V	56
6.7.2.12 FUCHSIA	56
6.7.2.13 FUCHSIA_V	56
6.7.2.14 GRAY	56
6.7.2.15 GRAY_V	56
6.7.2.16 GREEN	56
6.7.2.17 GREEN_V	57
6.7.2.18 LIME	57
6.7.2.19 LIME_V	57
6.7.2.20 MAGENTA	57
6.7.2.21 MAGENTA_V	57
6.7.2.22 MAROON	57
6.7.2.23 MAROON_V	57
6.7.2.24 NAVY	58
6.7.2.25 NAVY_V	58
6.7.2.26 NIGHT_BLUE	58
6.7.2.27 NIGHT_BLUE_V	58
6.7.2.28 NIGHT_MODE_V	58
6.7.2.29 OLIVE	58
6.7.2.30 OLIVE_V	58
6.7.2.31 RED	59
6.7.2.32 RED_V	59
6.7.2.33 SILVER	59
6.7.2.34 SILVER_V	59
6.7.2.35 SKY_BLUE	59
6.7.2.36 SKY_BLUE_V	59
6.7.2.37 SUNSET	59
6.7.2.38 SUNSET_V	60
6.7.2.39 TEAL	60
6.7.2.40 TEAL_V	60
6.7.2.41 WHITE	60
6.7.2.42 WHITE_V	60
6.7.2.43 YELLOW	60
6.7.2.44 YELLOW_V	60
6.8 ven::DescriptorPool Class Reference	61
6.8.1 Detailed Description	62
6.8.2 Constructor & Destructor Documentation	62
6.8.2.1 DescriptorPool() [1/2]	62

6.8.2.2 ~DescriptorPool()	63
6.8.2.3 DescriptorPool() [2/2]	63
6.8.3 Member Function Documentation	63
6.8.3.1 allocateDescriptor()	63
6.8.3.2 freeDescriptors()	63
6.8.3.3 getDescriptorPool()	64
6.8.3.4 operator=()	64
6.8.3.5 resetPool()	64
6.8.4 Friends And Related Symbol Documentation	64
6.8.4.1 DescriptorWriter	64
6.8.5 Member Data Documentation	64
6.8.5.1 m_descriptorPool	64
6.8.5.2 m_device	65
6.9 ven::DescriptorSetLayout Class Reference	65
6.9.1 Detailed Description	67
6.9.2 Constructor & Destructor Documentation	67
6.9.2.1 DescriptorSetLayout() [1/2]	67
6.9.2.2 ~DescriptorSetLayout()	68
6.9.2.3 DescriptorSetLayout() [2/2]	68
6.9.3 Member Function Documentation	68
6.9.3.1 getDescriptorSetLayout()	68
6.9.3.2 operator=()	68
6.9.4 Friends And Related Symbol Documentation	68
6.9.4.1 DescriptorWriter	68
6.9.5 Member Data Documentation	69
6.9.5.1 m_bindings	69
6.9.5.2 m_descriptorSetLayout	69
6.9.5.3 m_device	69
6.10 ven::DescriptorWriter Class Reference	69
6.10.1 Detailed Description	71
6.10.2 Constructor & Destructor Documentation	71
6.10.2.1 DescriptorWriter()	71
6.10.3 Member Function Documentation	71
6.10.3.1 build()	71
6.10.3.2 overwrite()	71
6.10.3.3 writeBuffer()	72
6.10.3.4 writImage()	72
6.10.4 Member Data Documentation	72
6.10.4.1 m_pool	72
6.10.4.2 m_setLayout	72
6.10.4.3 m_writes	73
6.11 ven::Device Class Reference	73



6.11.1 Detailed Description	76
6.11.2 Constructor & Destructor Documentation	76
6.11.2.1 Device() [1/3]	76
6.11.2.2 ~Device()	77
6.11.2.3 Device() [2/3]	77
6.11.2.4 Device() [3/3]	77
6.11.3 Member Function Documentation	77
6.11.3.1 beginSingleTimeCommands()	77
6.11.3.2 checkDeviceExtensionSupport()	77
6.11.3.3 checkValidationLayerSupport()	77
6.11.3.4 copyBuffer()	78
6.11.3.5 copyBufferToImage()	78
6.11.3.6 createBuffer()	78
6.11.3.7 createCommandPool()	79
6.11.3.8 createImageWithInfo()	79
6.11.3.9 createInstance()	79
6.11.3.10 createLogicalDevice()	80
6.11.3.11 createSurface()	80
6.11.3.12 device()	81
6.11.3.13 endSingleTimeCommands()	81
6.11.3.14 findMemoryType()	82
6.11.3.15 findPhysicalQueueFamilies()	82
6.11.3.16 findQueueFamilies()	82
6.11.3.17 findSupportedFormat()	83
6.11.3.18 getCommandPool()	83
6.11.3.19 getGraphicsQueue()	83
6.11.3.20 getPhysicalDevice()	83
6.11.3.21 getRequiredExtensions()	84
6.11.3.22 getSwapChainSupport()	84
6.11.3.23 graphicsQueue()	84
6.11.3.24 hasGlfwRequiredInstanceExtensions()	84
6.11.3.25 isDeviceSuitable()	85
6.11.3.26 operator=() [1/2]	85
6.11.3.27 operator=() [2/2]	85
6.11.3.28 pickPhysicalDevice()	85
6.11.3.29 populateDebugMessengerCreateInfo()	86
6.11.3.30 presentQueue()	86
6.11.3.31 querySwapChainSupport()	86
6.11.3.32 setupDebugMessenger()	87
6.11.3.33 surface()	87
6.11.4 Member Data Documentation	87
6.11.4.1 deviceExtensions	87

6.11.4.2 enableValidationLayers	88
6.11.4.3 m_commandPool	88
6.11.4.4 m_debugMessenger	88
6.11.4.5 m_device	88
6.11.4.6 m_graphicsQueue	88
6.11.4.7 m_instance	88
6.11.4.8 m_physicalDevice	89
6.11.4.9 m_presentQueue	89
6.11.4.10 m_surface	89
6.11.4.11 m_window	89
6.11.4.12 properties_	89
6.11.4.13 validationLayers	89
6.12 ven::Engine Class Reference	90
6.12.1 Detailed Description	91
6.12.2 Constructor & Destructor Documentation	91
6.12.2.1 Engine() [1/2]	91
6.12.2.2 ~Engine()	92
6.12.2.3 Engine() [2/2]	92
6.12.3 Member Function Documentation	92
6.12.3.1 createInstance()	92
6.12.3.2 createSurface()	93
6.12.3.3 loadObjects()	93
6.12.3.4 mainLoop()	94
6.12.3.5 operator=()	95
6.12.4 Member Data Documentation	95
6.12.4.1 m_device	95
6.12.4.2 m_globalPool	95
6.12.4.3 m_instance	95
6.12.4.4 m_objects	95
6.12.4.5 m_renderer	96
6.12.4.6 m_surface	96
6.12.4.7 m_window	96
6.13 ven::FrameInfo Struct Reference	96
6.13.1 Detailed Description	98
6.13.2 Member Data Documentation	98
6.13.2.1 camera	98
6.13.2.2 commandBuffer	98
6.13.2.3 frameIndex	98
6.13.2.4 frameTime	98
6.13.2.5 globalDescriptorSet	98
6.13.2.6 objects	99
6.14 ven::ImGuiWindowManager::funcs Struct Reference	99

6.14.1 Detailed Description	99
6.14.2 Member Function Documentation	100
6.14.2.1 IsLegacyNativeDupe()	100
6.15 ven::GlobalUbo Struct Reference	101
6.15.1 Detailed Description	101
6.15.2 Member Data Documentation	102
6.15.2.1 ambientLightColor	102
6.15.2.2 inverseView	102
6.15.2.3 numLights	102
6.15.2.4 pointLights	102
6.15.2.5 projection	102
6.15.2.6 view	102
6.16 std::hash< ven::Model::Vertex > Struct Reference	103
6.16.1 Detailed Description	103
6.16.2 Member Function Documentation	103
6.16.2.1 operator()()	103
6.17 ven::ImGuiWindowManager Class Reference	104
6.17.1 Detailed Description	104
6.17.2 Constructor & Destructor Documentation	105
6.17.2.1 ImGuiWindowManager() [1/2]	105
6.17.2.2 ~ImGuiWindowManager()	105
6.17.2.3 ImGuiWindowManager() [2/2]	105
6.17.3 Member Function Documentation	105
6.17.3.1 cleanup()	105
6.17.3.2 init()	106
6.17.3.3 operator=()	106
6.17.3.4 render()	107
6.18 ven::KeyboardController Class Reference	108
6.18.1 Detailed Description	109
6.18.2 Member Function Documentation	109
6.18.2.1 moveInPlaneXZ()	109
6.18.3 Member Data Documentation	109
6.18.3.1 m_keys	109
6.18.3.2 m_lookSpeed	109
6.18.3.3 m_moveSpeed	110
6.19 ven::KeyboardController::KeyMappings Struct Reference	110
6.19.1 Detailed Description	111
6.19.2 Member Data Documentation	111
6.19.2.1 lookDown	111
6.19.2.2 lookLeft	111
6.19.2.3 lookRight	111
6.19.2.4 lookUp	111

6.19.2.5 moveBackward	111
6.19.2.6 moveDown	112
6.19.2.7 moveForward	112
6.19.2.8 moveLeft	112
6.19.2.9 moveRight	112
6.19.2.10 moveUp	112
6.20 ven::Model Class Reference	113
6.20.1 Detailed Description	114
6.20.2 Constructor & Destructor Documentation	114
6.20.2.1 Model() [1/2]	114
6.20.2.2 ~Model()	115
6.20.2.3 Model() [2/2]	115
6.20.3 Member Function Documentation	115
6.20.3.1 bind()	115
6.20.3.2 createIndexBuffer()	115
6.20.3.3 createModelFromFile()	116
6.20.3.4 createVertexBuffer()	116
6.20.3.5 draw()	117
6.20.3.6 operator=()	117
6.20.4 Member Data Documentation	117
6.20.4.1 m_device	117
6.20.4.2 m_hasIndexBuffer	117
6.20.4.3 m_indexBuffer	117
6.20.4.4 m_indexCount	118
6.20.4.5 m_vertexBuffer	118
6.20.4.6 m_vertexCount	118
6.21 ven::Object Class Reference	118
6.21.1 Detailed Description	120
6.21.2 Member Typedef Documentation	120
6.21.2.1 Map	120
6.21.3 Constructor & Destructor Documentation	120
6.21.3.1 ~Object()	120
6.21.3.2 Object() [1/3]	121
6.21.3.3 Object() [2/3]	121
6.21.3.4 Object() [3/3]	121
6.21.4 Member Function Documentation	121
6.21.4.1 createObject()	121
6.21.4.2 getId()	122
6.21.4.3 makePointLight()	122
6.21.4.4 operator=() [1/2]	123
6.21.4.5 operator=() [2/2]	123
6.21.5 Member Data Documentation	123

6.21.5.1 color	123
6.21.5.2 m_objId	123
6.21.5.3 model	123
6.21.5.4 name	123
6.21.5.5 pointLight	124
6.21.5.6 transform3D	124
6.22 ven::PipelineConfigInfo Struct Reference	124
6.22.1 Detailed Description	125
6.22.2 Constructor & Destructor Documentation	125
6.22.2.1 PipelineConfigInfo() [1/2]	125
6.22.2.2 PipelineConfigInfo() [2/2]	125
6.22.3 Member Function Documentation	125
6.22.3.1 operator=()	125
6.22.4 Member Data Documentation	126
6.22.4.1 attributeDescriptions	126
6.22.4.2 bindingDescriptions	126
6.22.4.3 colorBlendAttachment	126
6.22.4.4 colorBlendInfo	126
6.22.4.5 depthStencilInfo	126
6.22.4.6 dynamicStateEnables	126
6.22.4.7 dynamicStateInfo	127
6.22.4.8 inputAssemblyInfo	127
6.22.4.9 multisampleInfo	127
6.22.4.10 pipelineLayout	127
6.22.4.11 rasterizationInfo	127
6.22.4.12 renderPass	127
6.22.4.13 subpass	128
6.23 ven::PointLight Struct Reference	128
6.23.1 Detailed Description	128
6.23.2 Member Data Documentation	128
6.23.2.1 color	128
6.23.2.2 position	129
6.24 ven::PointLightComponent Struct Reference	129
6.24.1 Detailed Description	129
6.24.2 Member Data Documentation	129
6.24.2.1 lightIntensity	129
6.25 PointLightPushConstants Struct Reference	130
6.25.1 Detailed Description	130
6.25.2 Member Data Documentation	130
6.25.2.1 color	130
6.25.2.2 position	130
6.25.2.3 radius	131

6.26 ven::PointLightSystem Class Reference	131
6.26.1 Detailed Description	133
6.26.2 Constructor & Destructor Documentation	133
6.26.2.1 PointLightSystem() [1/2]	133
6.26.2.2 ~PointLightSystem()	134
6.26.2.3 PointLightSystem() [2/2]	134
6.26.3 Member Function Documentation	134
6.26.3.1 createPipeline()	134
6.26.3.2 createPipelineLayout()	135
6.26.3.3 operator=()	135
6.26.3.4 render()	135
6.26.3.5 update()	136
6.26.4 Member Data Documentation	136
6.26.4.1 m_device	136
6.26.4.2 m_pipelineLayout	136
6.26.4.3 m_shaders	137
6.27 ven::QueueFamilyIndices Struct Reference	137
6.27.1 Detailed Description	137
6.27.2 Member Function Documentation	138
6.27.2.1 isComplete()	138
6.27.3 Member Data Documentation	138
6.27.3.1 graphicsFamily	138
6.27.3.2 graphicsFamilyHasValue	138
6.27.3.3 presentFamily	138
6.27.3.4 presentFamilyHasValue	139
6.28 myLib::Random Class Reference	139
6.28.1 Detailed Description	139
6.28.2 Member Function Documentation	140
6.28.2.1 randomFloat() [1/2]	140
6.28.2.2 randomFloat() [2/2]	140
6.28.2.3 randomInt() [1/2]	141
6.28.2.4 randomInt() [2/2]	141
6.29 ven::Renderer Class Reference	142
6.29.1 Detailed Description	143
6.29.2 Constructor & Destructor Documentation	143
6.29.2.1 Renderer() [1/2]	143
6.29.2.2 ~Renderer()	144
6.29.2.3 Renderer() [2/2]	144
6.29.3 Member Function Documentation	144
6.29.3.1 beginFrame()	144
6.29.3.2 beginSwapChainRenderPass()	144
6.29.3.3 createCommandBuffers()	145

6.29.3.4 endFrame()	145
6.29.3.5 endSwapChainRenderPass()	145
6.29.3.6 freeCommandBuffers()	146
6.29.3.7 getAspectRatio()	146
6.29.3.8 getClearColor()	146
6.29.3.9 getCurrentCommandBuffer()	147
6.29.3.10 getFrameIndex()	147
6.29.3.11 getSwapChainRenderPass()	148
6.29.3.12 isFrameInProgress()	148
6.29.3.13 operator=()	148
6.29.3.14 recreateSwapChain()	149
6.29.3.15 setClearColor()	149
6.29.4 Member Data Documentation	149
6.29.4.1 m_clearValues	149
6.29.4.2 m_commandBuffers	150
6.29.4.3 m_currentFrameIndex	150
6.29.4.4 m_currentImageIndex	150
6.29.4.5 m_device	150
6.29.4.6 m_isFrameStarted	150
6.29.4.7 m_swapChain	150
6.29.4.8 m_window	151
6.30 ven::RenderSystem Class Reference	151
6.30.1 Detailed Description	153
6.30.2 Constructor & Destructor Documentation	153
6.30.2.1 RenderSystem() [1/2]	153
6.30.2.2 ~RenderSystem()	154
6.30.2.3 RenderSystem() [2/2]	154
6.30.3 Member Function Documentation	154
6.30.3.1 createPipeline()	154
6.30.3.2 createPipelineLayout()	155
6.30.3.3 operator=()	155
6.30.3.4 renderObjects()	155
6.30.4 Member Data Documentation	156
6.30.4.1 m_device	156
6.30.4.2 m_pipelineLayout	156
6.30.4.3 m_shaders	156
6.31 ven::Shaders Class Reference	157
6.31.1 Detailed Description	158
6.31.2 Constructor & Destructor Documentation	159
6.31.2.1 Shaders() [1/2]	159
6.31.2.2 ~Shaders()	159
6.31.2.3 Shaders() [2/2]	159

6.31.3 Member Function Documentation	160
6.31.3.1 bind()	160
6.31.3.2 createGraphicsPipeline()	160
6.31.3.3 createShaderModule()	160
6.31.3.4 defaultPipelineConfigInfo()	161
6.31.3.5 operator=()	161
6.31.3.6 readFile()	161
6.31.4 Member Data Documentation	162
6.31.4.1 m_device	162
6.31.4.2 m_fragShaderModule	162
6.31.4.3 m_graphicsPipeline	162
6.31.4.4 m_vertShaderModule	162
6.32 ven::SimplePushConstantData Struct Reference	162
6.32.1 Detailed Description	163
6.32.2 Member Data Documentation	163
6.32.2.1 modelMatrix	163
6.32.2.2 normalMatrix	163
6.33 ven::SwapChain Class Reference	163
6.33.1 Detailed Description	166
6.33.2 Constructor & Destructor Documentation	166
6.33.2.1 SwapChain() [1/3]	166
6.33.2.2 SwapChain() [2/3]	166
6.33.2.3 ~SwapChain()	167
6.33.2.4 SwapChain() [3/3]	167
6.33.3 Member Function Documentation	167
6.33.3.1 acquireNextImage()	167
6.33.3.2 chooseSwapExtent()	167
6.33.3.3 chooseSwapPresentMode()	167
6.33.3.4 chooseSwapSurfaceFormat()	168
6.33.3.5 compareSwapFormats()	168
6.33.3.6 createDepthResources()	168
6.33.3.7 createFrameBuffers()	168
6.33.3.8 createImageViews()	168
6.33.3.9 createRenderPass()	168
6.33.3.10 createSwapChain()	168
6.33.3.11 createSyncObjects()	169
6.33.3.12 extentAspectRatio()	169
6.33.3.13 findDepthFormat()	169
6.33.3.14 getFrameBuffer()	169
6.33.3.15 getImageView()	169
6.33.3.16 getRenderPass()	169
6.33.3.17 getSwapChainExtent()	170



6.33.3.18 getSwapChainImageFormat()	170
6.33.3.19 height()	170
6.33.3.20 imageCount()	170
6.33.3.21 init()	170
6.33.3.22 operator=()	171
6.33.3.23 submitCommandBuffers()	171
6.33.3.24 width()	171
6.33.4 Member Data Documentation	171
6.33.4.1 currentFrame	171
6.33.4.2 depthImageMemory	171
6.33.4.3 depthImages	171
6.33.4.4 depthImageViews	172
6.33.4.5 device	172
6.33.4.6 imageAvailableSemaphores	172
6.33.4.7 imagesInFlight	172
6.33.4.8 inFlightFences	172
6.33.4.9 m_swapChainExtent	172
6.33.4.10 MAX_FRAMES_IN_FLIGHT	173
6.33.4.11 oldSwapChain	173
6.33.4.12 renderFinishedSemaphores	173
6.33.4.13 renderPass	173
6.33.4.14 swapChain	173
6.33.4.15 swapChainDepthFormat	173
6.33.4.16 swapChainFrameBuffers	174
6.33.4.17 swapChainImageFormat	174
6.33.4.18 swapChainImages	174
6.33.4.19 swapChainImageViews	174
6.33.4.20 windowExtent	174
6.34 ven::SwapChainSupportDetails Struct Reference	175
6.34.1 Detailed Description	175
6.34.2 Member Data Documentation	175
6.34.2.1 capabilities	175
6.34.2.2 formats	175
6.34.2.3 presentModes	176
6.35 myLib::Time Class Reference	176
6.35.1 Detailed Description	177
6.35.2 Constructor & Destructor Documentation	177
6.35.2.1 Time()	177
6.35.3 Member Function Documentation	177
6.35.3.1 asMicroseconds()	177
6.35.3.2 asMilliseconds()	177
6.35.3.3 asSeconds()	178

6.35.4 Member Data Documentation	178
6.35.4.1 m_seconds	178
6.36 ven::Transform3DComponent Struct Reference	178
6.36.1 Detailed Description	179
6.36.2 Member Function Documentation	179
6.36.2.1 mat4()	179
6.36.2.2 normalMatrix()	179
6.36.3 Member Data Documentation	179
6.36.3.1 rotation	179
6.36.3.2 scale	179
6.36.3.3 translation	180
6.37 ven::Model::Vertex Struct Reference	180
6.37.1 Detailed Description	181
6.37.2 Member Function Documentation	181
6.37.2.1 getAttributeDescriptions()	181
6.37.2.2 getBindingDescriptions()	181
6.37.2.3 operator==( )	181
6.37.3 Member Data Documentation	182
6.37.3.1 color	182
6.37.3.2 normal	182
6.37.3.3 position	182
6.37.3.4 uv	182
6.38 ven::Window Class Reference	183
6.38.1 Detailed Description	184
6.38.2 Constructor & Destructor Documentation	184
6.38.2.1 Window()	184
6.38.2.2 ~Window()	184
6.38.3 Member Function Documentation	184
6.38.3.1 createWindow()	184
6.38.3.2 createWindowSurface()	185
6.38.3.3 framebufferResizeCallback()	185
6.38.3.4 getExtent()	185
6.38.3.5 getGLFWWindow()	186
6.38.3.6 resetWindowResizedFlag()	186
6.38.3.7 wasWindowResized()	186
6.38.4 Member Data Documentation	186
6.38.4.1 m_framebufferResized	186
6.38.4.2 m_height	187
6.38.4.3 m_width	187
6.38.4.4 m_window	187

7.1 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference . . . . .	189
7.1.1 Detailed Description . . . . .	190
7.2 Buffer.hpp . . . . .	190
7.3 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference . . . . .	192
7.3.1 Detailed Description . . . . .	194
7.4 Camera.hpp . . . . .	194
7.5 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference . . . . .	195
7.6 Colors.hpp . . . . .	196
7.7 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference . . . . .	197
7.7.1 Detailed Description . . . . .	198
7.8 DescriptorPool.hpp . . . . .	198
7.9 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference . . . . .	199
7.9.1 Detailed Description . . . . .	200
7.10 DescriptorSetLayout.hpp . . . . .	201
7.11 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference . . . . .	201
7.11.1 Detailed Description . . . . .	203
7.12 DescriptorWriter.hpp . . . . .	203
7.13 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference . . . . .	203
7.13.1 Detailed Description . . . . .	204
7.14 Device.hpp . . . . .	205
7.15 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference . . . . .	206
7.15.1 Detailed Description . . . . .	207
7.16 Engine.hpp . . . . .	207
7.17 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference . . . . .	208
7.17.1 Detailed Description . . . . .	209
7.18 FrameInfo.hpp . . . . .	209
7.19 /home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp File Reference . . . . .	210
7.19.1 Detailed Description . . . . .	211
7.20 ImGuiWindowManager.hpp . . . . .	211
7.21 /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp File Reference . . . . .	212
7.22 KeyboardController.hpp . . . . .	213
7.23 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference . . . . .	214
7.23.1 Detailed Description . . . . .	215
7.24 Model.hpp . . . . .	215
7.25 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference . . . . .	216
7.25.1 Detailed Description . . . . .	218
7.26 Object.hpp . . . . .	218
7.27 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference . . . . .	219
7.27.1 Detailed Description . . . . .	220
7.28 Renderer.hpp . . . . .	220
7.29 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference . . . . .	221

7.29.1 Detailed Description	222
7.30 Shaders.hpp	222
7.31 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference	223
7.31.1 Detailed Description	225
7.32 SwapChain.hpp	225
7.33 /home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp File Reference	226
7.33.1 Detailed Description	227
7.34 PointLightSystem.hpp	227
7.35 /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp File Reference	228
7.35.1 Detailed Description	229
7.36 RenderSystem.hpp	229
7.37 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference	230
7.38 Utils.hpp	231
7.39 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference	231
7.39.1 Detailed Description	233
7.39.2 Macro Definition Documentation	233
7.39.2.1 GLFW_INCLUDE_VULKAN	233
7.40 Window.hpp	233
7.41 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference	234
7.41.1 Detailed Description	235
7.41.2 Typedef Documentation	235
7.41.2.1 TimePoint	235
7.42 Clock.hpp	235
7.43 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp File Reference	236
7.43.1 Detailed Description	237
7.44 Time.hpp	238
7.45 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp File Reference	238
7.45.1 Detailed Description	240
7.46 Random.hpp	240
7.47 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp File Reference	241
7.48 clock.cpp	241
7.49 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference	242
7.50 random.cpp	242
7.51 /home/runner/work/VEngine/VEngine/README.md File Reference	242
7.52 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference	242
7.53 buffer.cpp	243
7.54 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference	244
7.55 camera.cpp	245
7.56 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference	246
7.57 descriptorPool.cpp	246
7.58 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference	247

7.59 descriptorSetLayout.cpp . . . . .	248
7.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference . . . . .	248
7.61 descriptorWriter.cpp . . . . .	249
7.62 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference . . . . .	250
7.62.1 Function Documentation . . . . .	251
7.62.1.1 CreateDebugUtilsMessengerEXT() . . . . .	251
7.62.1.2 debugCallback() . . . . .	251
7.62.1.3 DestroyDebugUtilsMessengerEXT() . . . . .	252
7.63 device.cpp . . . . .	252
7.64 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference . . . . .	258
7.65 engine.cpp . . . . .	259
7.66 /home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp File Reference . . . . .	261
7.67 ImGuiWindowManager.cpp . . . . .	261
7.68 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp File Reference . . . . .	265
7.69 keyboardController.cpp . . . . .	265
7.70 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference . . . . .	266
7.70.1 Function Documentation . . . . .	266
7.70.1.1 main() . . . . .	266
7.71 main.cpp . . . . .	267
7.72 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference . . . . .	267
7.72.1 Macro Definition Documentation . . . . .	268
7.72.1.1 TINYOBJLOADER_IMPLEMENTATION . . . . .	268
7.73 model.cpp . . . . .	269
7.74 /home/runner/work/VEngine/VEngine/src/object.cpp File Reference . . . . .	271
7.75 object.cpp . . . . .	271
7.76 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference . . . . .	273
7.77 renderer.cpp . . . . .	273
7.78 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference . . . . .	275
7.79 shaders.cpp . . . . .	275
7.80 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference . . . . .	278
7.81 swapChain.cpp . . . . .	278
7.82 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference . . . . .	283
7.83 pointLightSystem.cpp . . . . .	283
7.84 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference . . . . .	285
7.85 renderSystem.cpp . . . . .	285
7.86 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference . . . . .	286
7.87 window.cpp . . . . .	286

## Index

289



# Chapter 1

## vengine

### 1.1 Description

#### ACTUALLY WORKING ON IT!

Welcome to **VEngine**, a graphics engine developed with Vulkan. This project aims to provide a robust foundation for game and application developers, focusing on the performance and flexibility offered by Vulkan.

### 1.2 Prerequisites

- CMake 3.27
- C++20
- Vulkan
- GLM
- assimp (unused ATM)

### 1.3 Usage

#### 1.3.1 Build

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

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

#### 1.3.2 Run

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

#### 1.3.3 Key Bindings

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

### 1.3.4 Documentation

The documentation is generated using [Doxygen](#). You can visualize it on the [GitHub Pages](#).

## 1.4 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

## 1.5 License

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

## 1.6 Acknowledgements

Thanks to [Brendan Galea](#).



## Chapter 2

# Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">myLib</a>	.....	<a href="#">9</a>
<a href="#">std</a>	STL namespace .....	<a href="#">10</a>
<a href="#">ven</a>	.....	<a href="#">14</a>



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">ven::Buffer</a>	
Class for buffer	21
<a href="#">ven::DescriptorPool::Builder</a>	34
<a href="#">ven::DescriptorSetLayout::Builder</a>	38
<a href="#">ven::Model::Builder</a>	41
<a href="#">ven::Camera</a>	
Class for camera	43
<a href="#">myLib::Clock</a>	
Class for time management	49
<a href="#">ven::Colors</a>	
Class for colors	52
<a href="#">ven::DescriptorPool</a>	
Class for descriptor pool	61
<a href="#">ven::DescriptorSetLayout</a>	
Class for descriptor set layout	65
<a href="#">ven::DescriptorWriter</a>	
Class for descriptor writer	69
<a href="#">ven::Device</a>	73
<a href="#">ven::Engine</a>	90
<a href="#">ven::FrameInfo</a>	96
<a href="#">ven::ImGuiWindowManager::funcs</a>	99
<a href="#">ven::GlobalUbo</a>	101
<a href="#">std::hash&lt; ven::Model::Vertex &gt;</a>	103
<a href="#">ven::ImGuiWindowManager</a>	
Class for ImGui window manager	104
<a href="#">ven::KeyboardController</a>	
Class for keyboard controller	108
<a href="#">ven::KeyboardController::KeyMappings</a>	110
<a href="#">ven::Model</a>	
Class for model	113
<a href="#">ven::Object</a>	
Class for object	118
<a href="#">ven::PipelineConfigInfo</a>	124
<a href="#">ven::PointLight</a>	128
<a href="#">ven::PointLightComponent</a>	129

<a href="#">PointLightPushConstants</a>	130
<a href="#">ven::PointLightSystem</a>	
Class for point light system	131
<a href="#">ven::QueueFamilyIndices</a>	137
<a href="#">myLib::Random</a>	
Class for random number generation	139
<a href="#">ven::Renderer</a>	142
<a href="#">ven::RenderSystem</a>	
Class for render system	151
<a href="#">ven::Shaders</a>	
Class for shaders	157
<a href="#">ven::SimplePushConstantData</a>	162
<a href="#">ven::SwapChain</a>	
Class for swap chain	163
<a href="#">ven::SwapChainSupportDetails</a>	175
<a href="#">myLib::Time</a>	
Class used for time management	176
<a href="#">ven::Transform3DComponent</a>	178
<a href="#">ven::Model::Vertex</a>	180
<a href="#">ven::Window</a>	
Class for window	183

## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp	
This file contains the Buffer class . . . . .	189
/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp	
This file contains the Camera class . . . . .	192
/home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp	195
/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp	
This file contains the Device class . . . . .	203
/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp	
This file contains the Engine class . . . . .	206
/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp	
This file contains the FrameInfo class . . . . .	208
/home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp	
This file contains the ImGuiWindowManager class . . . . .	210
/home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp	212
/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp	
This file contains the Model class . . . . .	214
/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp	
This file contains the Object class . . . . .	216
/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp	
This file contains the Renderer class . . . . .	219
/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp	
This file contains the Shader class . . . . .	221
/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp	
This file contains the Shader class . . . . .	223
/home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp	230
/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp	
This file contains the Window class . . . . .	231
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp	
This file contains the DescriptorPool class . . . . .	197
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp	
This file contains the DescriptorSetLayout class . . . . .	199
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp	
This file contains the DescriptorsWriter class . . . . .	201
/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp	
This file contains the PointLightSystem class . . . . .	226

/home/runner/work/VEngine/VEngine/include/VEngine/System/ <a href="#">RenderSystem.hpp</a>	
This file contains the RenderSystem class	228
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/ <a href="#">Random.hpp</a>	
Class for random number generation	238
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/ <a href="#">Clock.hpp</a>	
Clock class for time management	234
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/ <a href="#">Time.hpp</a>	
Class for time management	236
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/ <a href="#">clock.cpp</a>	241
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/ <a href="#">random.cpp</a>	242
/home/runner/work/VEngine/VEngine/src/ <a href="#">buffer.cpp</a>	242
/home/runner/work/VEngine/VEngine/src/ <a href="#">camera.cpp</a>	244
/home/runner/work/VEngine/VEngine/src/ <a href="#">device.cpp</a>	250
/home/runner/work/VEngine/VEngine/src/ <a href="#">engine.cpp</a>	258
/home/runner/work/VEngine/VEngine/src/ <a href="#">ImGuiWindowManager.cpp</a>	261
/home/runner/work/VEngine/VEngine/src/ <a href="#">keyboardController.cpp</a>	265
/home/runner/work/VEngine/VEngine/src/ <a href="#">main.cpp</a>	266
/home/runner/work/VEngine/VEngine/src/ <a href="#">model.cpp</a>	267
/home/runner/work/VEngine/VEngine/src/ <a href="#">object.cpp</a>	271
/home/runner/work/VEngine/VEngine/src/ <a href="#">renderer.cpp</a>	273
/home/runner/work/VEngine/VEngine/src/ <a href="#">shaders.cpp</a>	275
/home/runner/work/VEngine/VEngine/src/ <a href="#">swapChain.cpp</a>	278
/home/runner/work/VEngine/VEngine/src/ <a href="#">window.cpp</a>	286
/home/runner/work/VEngine/VEngine/src/descriptors/ <a href="#">descriptorPool.cpp</a>	246
/home/runner/work/VEngine/VEngine/src/descriptors/ <a href="#">descriptorSetLayout.cpp</a>	247
/home/runner/work/VEngine/VEngine/src/descriptors/ <a href="#">descriptorWriter.cpp</a>	248
/home/runner/work/VEngine/VEngine/src/system/ <a href="#">pointLightSystem.cpp</a>	283
/home/runner/work/VEngine/VEngine/src/system/ <a href="#">renderSystem.cpp</a>	285

## Chapter 5

# Namespace Documentation

### 5.1 myLib Namespace Reference

#### Classes

- class [Clock](#)  
*Class for time management.*
- class [Random](#)  
*Class for random number generation.*
- class [Time](#)  
*Class used for time management.*

#### Variables

- static constexpr unsigned int [MICROSECONDS\\_PER\\_SECOND](#) = 1000000
- static constexpr unsigned int [MILLISECONDS\\_PER\\_SECOND](#) = 1000
- static constexpr int [RANDOM\\_INT\\_MIN](#) = -1000
- static constexpr int [RANDOM\\_INT\\_MAX](#) = 1000
- static constexpr float [RANDOM\\_FLOAT\\_MAX](#) = 1000.0F

#### 5.1.1 Variable Documentation

##### 5.1.1.1 MICROSECONDS\_PER\_SECOND

```
unsigned int myLib::MICROSECONDS_PER_SECOND = 1000000 [static], [constexpr]
```

Definition at line 11 of file [Time.hpp](#).

Referenced by [myLib::Time::asMicroseconds\(\)](#).

##### 5.1.1.2 MILLISECONDS\_PER\_SECOND

```
unsigned int myLib::MILLISECONDS_PER_SECOND = 1000 [static], [constexpr]
```

Definition at line 12 of file [Time.hpp](#).

Referenced by [myLib::Time::asMilliseconds\(\)](#).

### 5.1.1.3 RANDOM\_FLOAT\_MAX

```
float myLib::RANDOM_FLOAT_MAX = 1000.0F [static], [constexpr]
```

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

Referenced by [myLib::Random::randomFloat\(\)](#).

### 5.1.1.4 RANDOM\_INT\_MAX

```
int myLib::RANDOM_INT_MAX = 1000 [static], [constexpr]
```

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

Referenced by [myLib::Random::randomFloat\(\)](#).

### 5.1.1.5 RANDOM\_INT\_MIN

```
int myLib::RANDOM_INT_MIN = -1000 [static], [constexpr]
```

Definition at line 13 of file [Random.hpp](#).

Referenced by [myLib::Random::randomFloat\(\)](#).

## 5.2 std Namespace Reference

STL namespace.

### Classes

- class **allocator**  
*STL class.*
- class **array**  
*STL class.*
- class **atomic**  
*STL class.*
- class **atomic\_ref**  
*STL class.*
- class **auto\_ptr**  
*STL class.*
- class **bad\_alloc**  
*STL class.*
- class **bad\_cast**  
*STL class.*
- class **bad\_exception**  
*STL class.*
- class **bad\_typeid**  
*STL class.*



- class **basic\_fstream**  
*STL class.*
- class **basic\_ifstream**  
*STL class.*
- class **basic\_ios**  
*STL class.*
- class **basic\_iostream**  
*STL class.*
- class **basic\_istream**  
*STL class.*
- class **basic\_istreamstream**  
*STL class.*
- class **basic\_ofstream**  
*STL class.*
- class **basic\_ostream**  
*STL class.*
- class **basic\_ostreamstream**  
*STL class.*
- class **basic\_string**  
*STL class.*
- class **basic\_string\_view**  
*STL class.*
- class **basic\_stringstream**  
*STL class.*
- class **bitset**  
*STL class.*
- class **complex**  
*STL class.*
- class **deque**  
*STL class.*
- class **domain\_error**  
*STL class.*
- class **error\_category**  
*STL class.*
- class **error\_code**  
*STL class.*
- class **error\_condition**  
*STL class.*
- class **exception**  
*STL class.*
- class **forward\_list**  
*STL class.*
- class **fstream**  
*STL class.*
- struct [hash< ven::Model::Vertex >](#)
- class **ifstream**  
*STL class.*
- class **invalid\_argument**  
*STL class.*
- class **ios**  
*STL class.*

- class **ios\_base**  
*STL class.*
- class **istream**  
*STL class.*
- class **istreamstream**  
*STL class.*
- class **jthread**  
*STL class.*
- class **length\_error**  
*STL class.*
- class **list**  
*STL class.*
- class **lock\_guard**  
*STL class.*
- class **logic\_error**  
*STL class.*
- class **map**  
*STL class.*
- class **multimap**  
*STL class.*
- class **multiset**  
*STL class.*
- class **mutex**  
*STL class.*
- class **ofstream**  
*STL class.*
- class **ostream**  
*STL class.*
- class **ostreamstream**  
*STL class.*
- class **out\_of\_range**  
*STL class.*
- class **overflow\_error**  
*STL class.*
- class **pair**  
*STL class.*
- class **priority\_queue**  
*STL class.*
- class **queue**  
*STL class.*
- class **range\_error**  
*STL class.*
- class **recursive\_mutex**  
*STL class.*
- class **recursive\_timed\_mutex**  
*STL class.*
- class **runtime\_error**  
*STL class.*
- class **set**  
*STL class.*
- class **shared\_lock**

- STL class.*
- class **shared\_mutex**
  - STL class.*
- class **shared\_ptr**
  - STL class.*
- class **shared\_timed\_mutex**
  - STL class.*
- class **smart\_ptr**
  - STL class.*
- class **span**
  - STL class.*
- class **stack**
  - STL class.*
- class **string**
  - STL class.*
- class **string\_view**
  - STL class.*
- class **stringstream**
  - STL class.*
- class **system\_error**
  - STL class.*
- class **thread**
  - STL class.*
- class **timed\_mutex**
  - STL class.*
- class **u16string**
  - STL class.*
- class **u16string\_view**
  - STL class.*
- class **u32string**
  - STL class.*
- class **u32string\_view**
  - STL class.*
- class **u8string**
  - STL class.*
- class **u8string\_view**
  - STL class.*
- class **underflow\_error**
  - STL class.*
- class **unique\_lock**
  - STL class.*
- class **unique\_ptr**
  - STL class.*
- class **unordered\_map**
  - STL class.*
- class **unordered\_multimap**
  - STL class.*
- class **unordered\_multiset**
  - STL class.*
- class **unordered\_set**
  - STL class.*

- class **valarray**  
*STL class.*
- class **vector**  
*STL class.*
- class **weak\_ptr**  
*STL class.*
- class **wfstream**  
*STL class.*
- class **wifstream**  
*STL class.*
- class **wios**  
*STL class.*
- class **wistream**  
*STL class.*
- class **wistringstream**  
*STL class.*
- class **wofstream**  
*STL class.*
- class **wostream**  
*STL class.*
- class **wostreamstream**  
*STL class.*
- class **wstring**  
*STL class.*
- class **wstring\_view**  
*STL class.*
- class **wstringstream**  
*STL class.*

### 5.2.1 Detailed Description

STL namespace.

## 5.3 ven Namespace Reference

### Classes

- 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 [Engine](#)
- struct [FrameInfo](#)
- struct [GlobalUbo](#)
- class [ImGuiWindowManager](#)

*Class for ImGui window manager.*

- class [KeyboardController](#)

*Class for keyboard controller.*

- class [Model](#)

*Class for model.*

- class [Object](#)

*Class for object.*

- struct [PipelineConfigInfo](#)
- struct [PointLight](#)
- struct [PointLightComponent](#)
- class [PointLightSystem](#)

*Class for point light system.*

- struct [QueueFamilyIndices](#)
- class [Renderer](#)
- class [RenderSystem](#)

*Class for render system.*

- class [Shaders](#)

*Class for shaders.*

- struct [SimplePushConstantData](#)
- class [SwapChain](#)

*Class for swap chain.*

- struct [SwapChainSupportDetails](#)
- struct [Transform3DComponent](#)
- class [Window](#)

*Class for window.*

## 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 std::size\_t [MAX\\_LIGHTS](#) = 10
- static constexpr float [DEFAULT\\_MOVE\\_SPEED](#) = 3.F
- static constexpr float [DEFAULT\\_LOOK\\_SPEED](#) = 1.5F
- static constexpr float [DEFAULT\\_LIGHT\\_INTENSITY](#) = .2F
- static constexpr float [DEFAULT\\_LIGHT\\_RADIUS](#) = 0.1F
- static constexpr glm::vec3 [DEFAULT\\_LIGHT\\_COLOR](#) = glm::vec3(1.F)
- 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](#) = "shaders/bin/"
- static constexpr uint32\_t [DEFAULT\\_WIDTH](#) = 1920
- static constexpr uint32\_t [DEFAULT\\_HEIGHT](#) = 1080
- static constexpr std::string\_view [DEFAULT\\_TITLE](#) = "VEngine"

### 5.3.1 Function Documentation

#### 5.3.1.1 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:



### 5.3.2 Variable Documentation

#### 5.3.2.1 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](#).

### 5.3.2.2 DEFAULT\_CLEAR\_DEPTH

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

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

### 5.3.2.3 DEFAULT\_FAR

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

### 5.3.2.4 DEFAULT\_FOV

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

### 5.3.2.5 DEFAULT\_HEIGHT

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

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

### 5.3.2.6 DEFAULT\_LIGHT\_COLOR

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

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

### 5.3.2.7 DEFAULT\_LIGHT\_INTENSITY

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

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

### 5.3.2.8 DEFAULT\_LIGHT\_RADIUS

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

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

#### 5.3.2.9 DEFAULT\_LOOK\_SPEED

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 5.3.2.10 DEFAULT\_MOVE\_SPEED

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 5.3.2.11 DEFAULT\_NEAR

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 5.3.2.12 DEFAULT\_POSITION

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

Definition at line 13 of file [Camera.hpp](#).

Referenced by [ven::Engine::mainLoop\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

#### 5.3.2.13 DEFAULT\_ROTATION

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 5.3.2.14 DEFAULT\_TITLE

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

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



#### 5.3.2.15 DEFAULT\_WIDTH

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

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

#### 5.3.2.16 MAX\_LIGHTS

```
std::size_t ven::MAX_LIGHTS = 10 [static], [constexpr]
```

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

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

#### 5.3.2.17 SHADERS\_BIN\_PATH

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

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

Referenced by [ven::PointLightSystem::createPipeline\(\)](#), and [ven::RenderSystem::createPipeline\(\)](#).



## Chapter 6

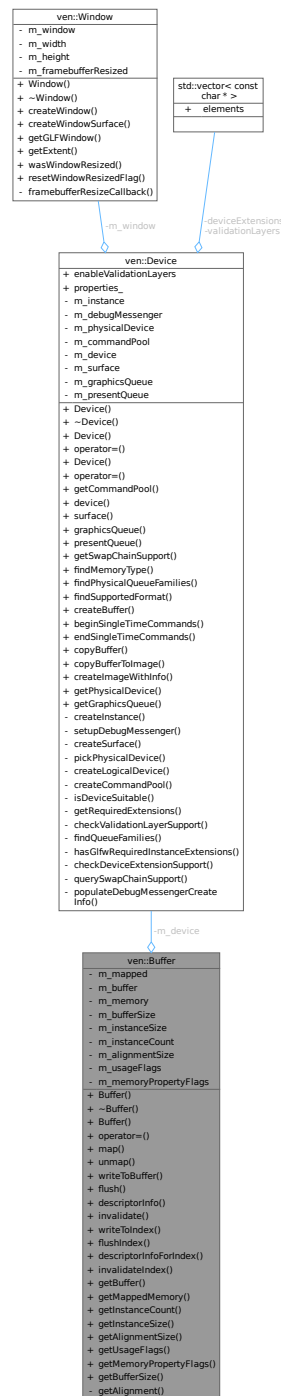
# Class Documentation

### 6.1 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`

### 6.1.1 Detailed Description

Class for buffer.

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

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 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:



#### 6.1.2.2 ~Buffer()

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

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

#### 6.1.2.3 Buffer() [2/2]

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

### 6.1.3 Member Function Documentation

#### 6.1.3.1 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

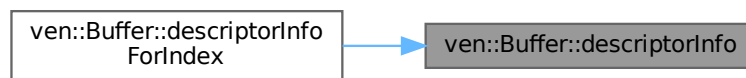
VkDescriptorBufferInfo of specified offset and range

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

References [m\\_buffer](#).

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

Here is the caller graph for this function:



## 6.1.3.2 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:



### 6.1.3.3 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:



### 6.1.3.4 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:



#### 6.1.3.5 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 <code>minUniformBufferOffsetAlignment</code> )

##### Returns

VkResult of the buffer mapping call

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

#### 6.1.3.6 getAlignmentSize()

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

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

References [m\\_instanceSize](#).

#### 6.1.3.7 getBuffer()

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

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

References [m\\_buffer](#).

#### 6.1.3.8 getBufferSize()

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

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

References [m\\_bufferSize](#).

#### 6.1.3.9 getInstanceCount()

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

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

References [m\\_instanceCount](#).

#### 6.1.3.10 getInstanceSize()

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

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

References [m\\_instanceSize](#).

#### 6.1.3.11 getMappedMemory()

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

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

References [m\\_mapped](#).

#### 6.1.3.12 getMemoryPropertyFlags()

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

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

References [m\\_memoryPropertyFlags](#).

#### 6.1.3.13 `getUsageFlags()`

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

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

References [m\\_usageFlags](#).

#### 6.1.3.14 `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:



#### 6.1.3.15 `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: $\text{index} * \text{alignmentSize}$
--------------	---

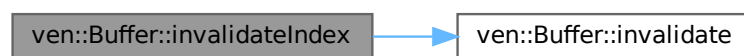
**Returns**

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:

**6.1.3.16 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 VK_WHOLE_SIZE to map the complete buffer range.
<i>offset</i>	(Optional) Byte offset from beginning

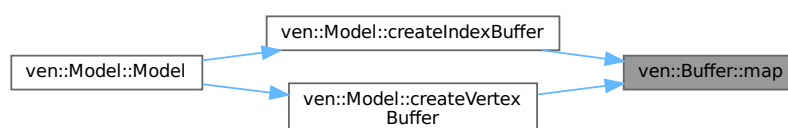
**Returns**

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:



### 6.1.3.17 operator=()

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

### 6.1.3.18 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](#).

### 6.1.3.19 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:



### 6.1.3.20 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:



## 6.1.4 Member Data Documentation

### 6.1.4.1 m\_alignmentSize

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

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

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

### 6.1.4.2 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\(\)](#).

### 6.1.4.3 m\_bufferSize

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

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

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

### 6.1.4.4 m\_device

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

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

#### 6.1.4.5 m\_instanceCount

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

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

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

#### 6.1.4.6 m\_instanceSize

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

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

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

#### 6.1.4.7 m\_mapped

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

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

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

#### 6.1.4.8 m\_memory

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

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

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

#### 6.1.4.9 m\_memoryPropertyFlags

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

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

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

#### 6.1.4.10 m\_usageFlags

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

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

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

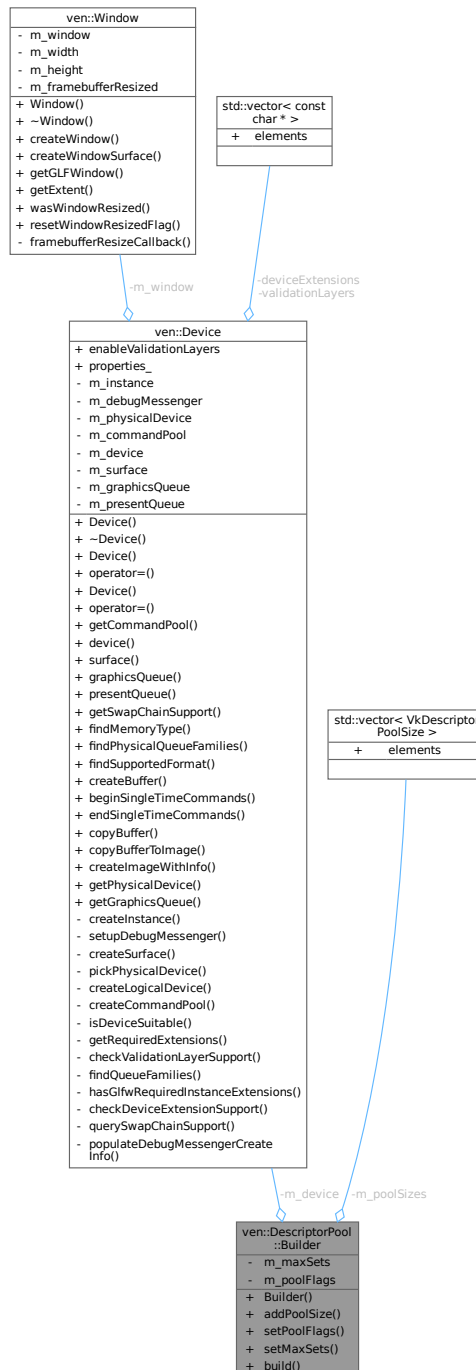
The documentation for this class was generated from the following files:

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

## 6.2 ven::DescriptorPool::Builder Class Reference

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

Collaboration diagram for ven::DescriptorPool::Builder:



### Public Member Functions

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



- [Builder](#) & [addPoolSize](#) (VkDescriptorType descriptorType, uint32\_t count)
- [Builder](#) & [setPoolFlags](#) (VkDescriptorPoolCreateFlags flags)
- [Builder](#) & [setMaxSets](#) (uint32\_t count)
- std::unique\_ptr< [DescriptorPool](#) > [build](#) () const

### Private Attributes

- [Device](#) & [m\\_device](#)
- std::vector< VkDescriptorPoolSize > [m\\_poolSizes](#)
- uint32\_t [m\\_maxSets](#) = 1000
- VkDescriptorPoolCreateFlags [m\\_poolFlags](#) = 0

## 6.2.1 Detailed Description

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

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 Builder()

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

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

## 6.2.3 Member Function Documentation

### 6.2.3.1 addPoolSize()

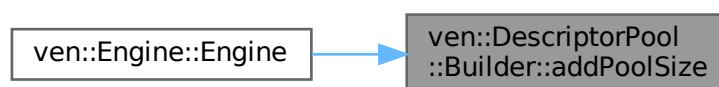
```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::addPoolSize (
    VkDescriptorType descriptorType,
    uint32_t count)
```

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

References [m\\_poolSizes](#).

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

Here is the caller graph for this function:



### 6.2.3.2 build()

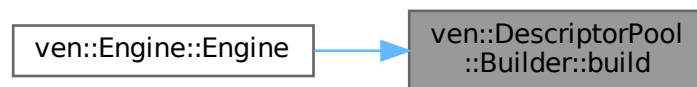
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],
[nodiscard]
```

Definition at line 34 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:



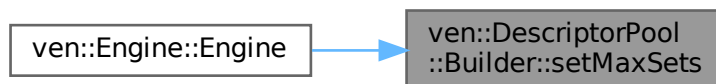
### 6.2.3.3 setMaxSets()

```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::setMaxSets (
    uint32_t count)
```

Definition at line 14 of file [descriptorPool.cpp](#).

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

Here is the caller graph for this function:



### 6.2.3.4 setPoolFlags()

```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::setPoolFlags (
    VkDescriptorPoolCreateFlags flags)
```

Definition at line 9 of file [descriptorPool.cpp](#).

## 6.2.4 Member Data Documentation

### 6.2.4.1 m\_device

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

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

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

### 6.2.4.2 m\_maxSets

`uint32_t ven::DescriptorPool::Builder::m_maxSets` = 1000 [private]

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

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

### 6.2.4.3 m\_poolFlags

`VkDescriptorPoolCreateFlags ven::DescriptorPool::Builder::m_poolFlags` = 0 [private]

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

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

### 6.2.4.4 m\_poolSizes

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

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

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

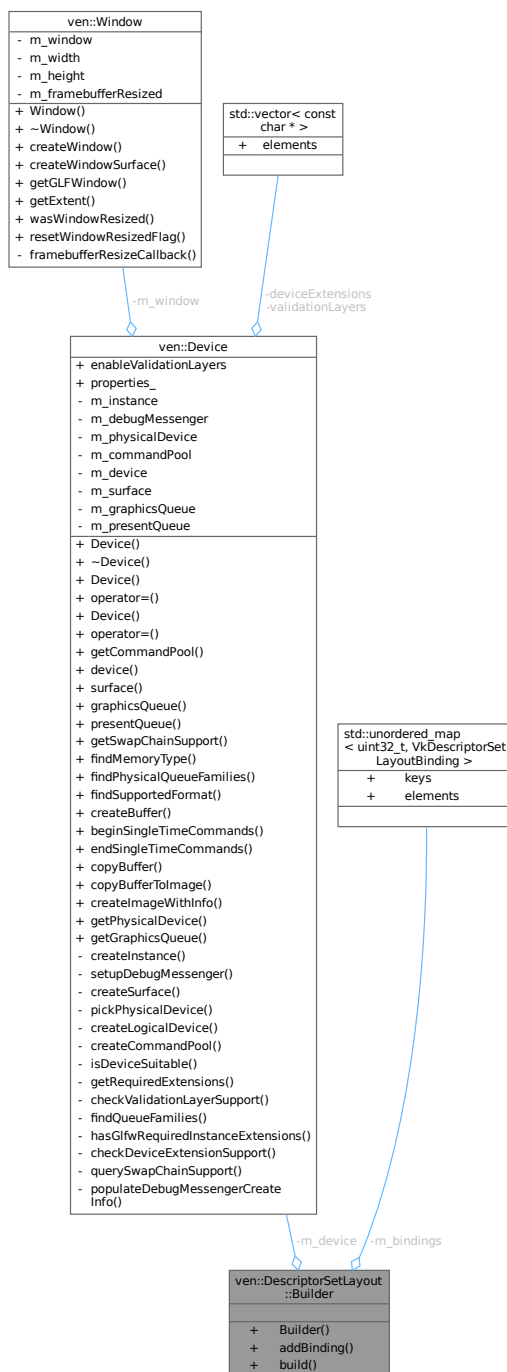
The documentation for this class was generated from the following files:

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

## 6.3 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](#)

## 6.3.1 Detailed Description

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

## 6.3.2 Constructor & Destructor Documentation

### 6.3.2.1 Builder()

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

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

## 6.3.3 Member Function Documentation

### 6.3.3.1 addBinding()

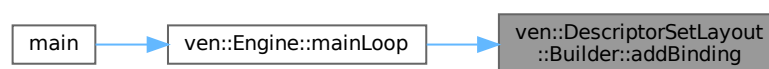
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

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

References [m\\_bindings](#).

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

Here is the caller graph for this function:



### 6.3.3.2 build()

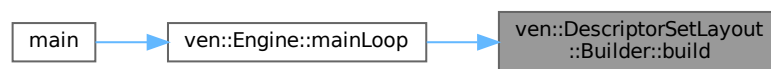
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

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:



## 6.3.4 Member Data Documentation

### 6.3.4.1 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\(\)](#).

### 6.3.4.2 m\_device

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

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

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

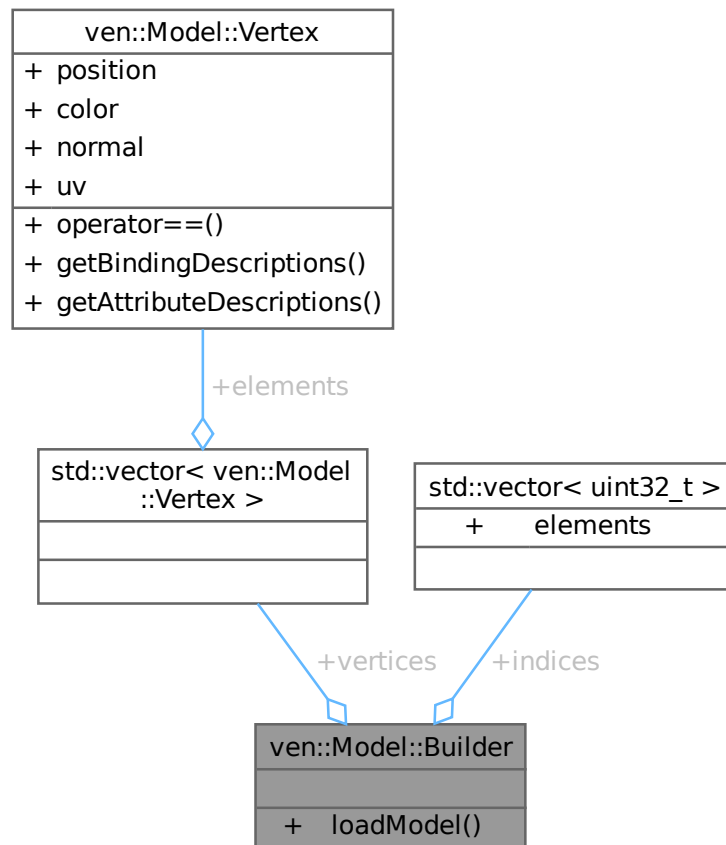
The documentation for this class was generated from the following files:

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

## 6.4 ven::Model::Builder Struct Reference

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

Collaboration diagram for ven::Model::Builder:



### Public Member Functions

- void [loadModel](#) (const std::string &filename)

### Public Attributes

- std::vector< [Vertex](#) > [vertices](#)
- std::vector< uint32\_t > [indices](#)

### 6.4.1 Detailed Description

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

## 6.4.2 Member Function Documentation

### 6.4.2.1 loadModel()

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

Definition at line 119 of file [model.cpp](#).

References [ven::Model::Vertex::position](#).

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

Here is the caller graph for this function:



## 6.4.3 Member Data Documentation

### 6.4.3.1 indices

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

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

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

### 6.4.3.2 vertices

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

Definition at line 40 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](#)



## 6.5 ven::Camera Class Reference

Class for camera.

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

Collaboration diagram for ven::Camera:

ven::Camera
<ul style="list-style-type: none"> <li>- m_fov</li> <li>- m_near</li> <li>- m_far</li> <li>- m_projectionMatrix</li> <li>- m_viewMatrix</li> <li>- m_inverseViewMatrix</li> </ul>
<ul style="list-style-type: none"> <li>+ setOrthographicProjection()</li> <li>+ setPerspectiveProjection()</li> <li>+ setViewDirection()</li> <li>+ setViewTarget()</li> <li>+ setViewYXZ()</li> <li>+ setFov()</li> <li>+ setNear()</li> <li>+ setFar()</li> <li>+ getProjection()</li> <li>+ getView()</li> <li>+ getInverseView()</li> <li>+ getFov()</li> <li>+ getNear()</li> <li>+ getFar()</li> </ul>

### Public Member Functions

- 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=glm::vec3{0.F, -1.F, 0.F})
- void [setViewTarget](#) (glm::vec3 position, glm::vec3 target, glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void [setViewYXZ](#) (glm::vec3 position, glm::vec3 rotation)
- void [setFov](#) (float fov)
- void [setNear](#) (float near)
- void [setFar](#) (float far)
- const glm::mat4 & [getProjection](#) () const
- const glm::mat4 & [getView](#) () const
- const glm::mat4 & [getInverseView](#) () const
- float [getFov](#) () const
- float [getNear](#) () const
- float [getFar](#) () const

### Private Attributes

- float [m\\_fov](#) {DEFAULT\_FOV}
- float [m\\_near](#) {DEFAULT\_NEAR}
- float [m\\_far](#) {DEFAULT\_FAR}
- glm::mat4 [m\\_projectionMatrix](#) {1.F}
- glm::mat4 [m\\_viewMatrix](#) {1.F}
- glm::mat4 [m\\_inverseViewMatrix](#) {1.F}

## 6.5.1 Detailed Description

Class for camera.

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

## 6.5.2 Member Function Documentation

### 6.5.2.1 getFar()

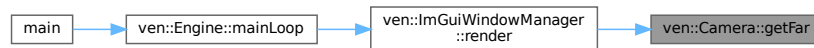
```
float ven::Camera::getFar () const [inline], [nodiscard]
```

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

References [m\\_far](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.5.2.2 getFov()

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

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

References [m\\_fov](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.5.2.3 getInverseView()

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

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

References [m\\_inverseViewMatrix](#).

### 6.5.2.4 getNear()

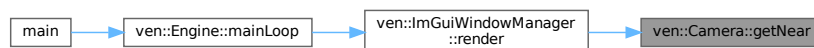
```
float ven::Camera::getNear () const [inline], [nodiscard]
```

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

References [m\\_near](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.5.2.5 getProjection()

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

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

References [m\\_projectionMatrix](#).

### 6.5.2.6 getView()

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

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

References [m\\_viewMatrix](#).

### 6.5.2.7 setFar()

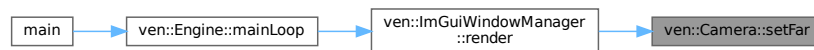
```
void ven::Camera::setFar (
    float far) [inline]
```

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

References [m\\_far](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.5.2.8 setFov()

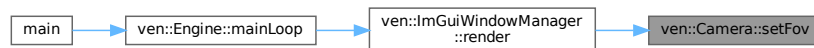
```
void ven::Camera::setFov (
    float fov) [inline]
```

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

References [m\\_fov](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.5.2.9 setNear()

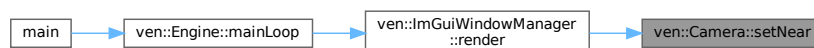
```
void ven::Camera::setNear (
    float near) [inline]
```

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

References [m\\_near](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



#### 6.5.2.10 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](#).

#### 6.5.2.11 setPerspectiveProjection()

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

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

#### 6.5.2.12 setViewDirection()

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

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

#### 6.5.2.13 setViewTarget()

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

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

#### 6.5.2.14 setViewYXZ()

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

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

## 6.5.3 Member Data Documentation

### 6.5.3.1 m\_far

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

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

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

### 6.5.3.2 m\_fov

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

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

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

### 6.5.3.3 m\_inverseViewMatrix

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

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

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

### 6.5.3.4 m\_near

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

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

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

### 6.5.3.5 m\_projectionMatrix

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

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

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

### 6.5.3.6 m\_viewMatrix

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

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

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

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](#)

## 6.6 myLib::Clock Class Reference

Class for time management.

```
#include <Clock.hpp>
```

Collaboration diagram for myLib::Clock:

myLib::Clock
<ul style="list-style-type: none"><li>- m_start</li><li>- m_pause</li><li>- m_paused</li></ul>
<ul style="list-style-type: none"><li>+ Clock()</li><li>+ ~Clock()</li><li>+ restart()</li><li>+ pause()</li><li>+ resume()</li><li>+ getElapsedTime()</li></ul>

### Public Member Functions

- [Clock](#) ()
- [~Clock](#) ()=default
- void [restart](#) ()  
*Restart the clock.*
- void [pause](#) ()  
*Pause the clock.*
- void [resume](#) ()  
*Resume the clock.*
- [Time getElapsedTime](#) () const  
*Get the elapsed time since the last restart.*

### Private Attributes

- [TimePoint m\\_start](#)
- [TimePoint m\\_pause](#)
- bool [m\\_paused](#) {false}

### 6.6.1 Detailed Description

Class for time management.

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

## 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 Clock()

```
myLib::Clock::Clock () [inline]
```

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

### 6.6.2.2 ~Clock()

```
myLib::Clock::~~Clock () [default]
```

## 6.6.3 Member Function Documentation

### 6.6.3.1 getElapsedTime()

```
myLib::Time myLib::Clock::getElapsedTime () const [nodiscard]
```

Get the elapsed time since the last restart.

#### Returns

[Time](#) The elapsed time

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

### 6.6.3.2 pause()

```
void myLib::Clock::pause ()
```

Pause the clock.

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

References [m\\_pause](#), and [m\\_paused](#).

### 6.6.3.3 restart()

```
void myLib::Clock::restart () [inline]
```

Restart the clock.

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

References [m\\_start](#).



#### 6.6.3.4 resume()

```
void myLib::Clock::resume ()
```

Resume the clock.

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

### 6.6.4 Member Data Documentation

#### 6.6.4.1 m\_pause

```
TimePoint myLib::Clock::m_pause [private]
```

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

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

#### 6.6.4.2 m\_paused

```
bool myLib::Clock::m_paused {false} [private]
```

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

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

#### 6.6.4.3 m\_start

```
TimePoint myLib::Clock::m_start [private]
```

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

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

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

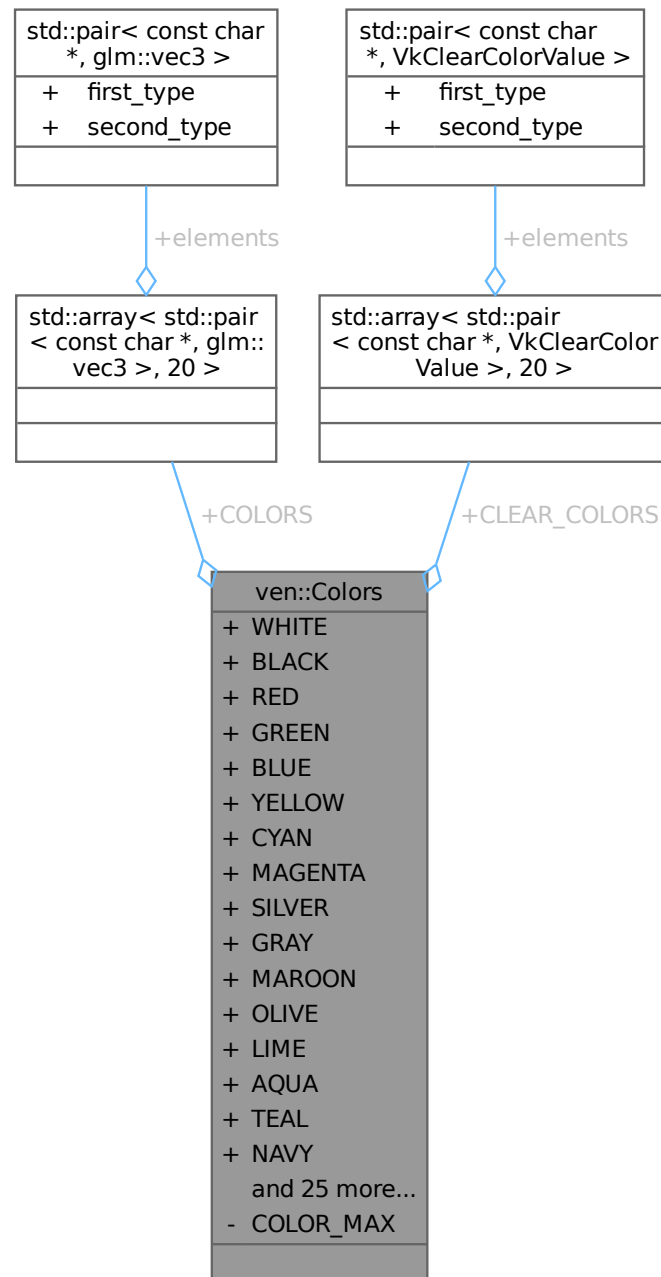
- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp](#)
- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp](#)

## 6.7 ven::Colors Class Reference

Class for colors.

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

Collaboration diagram for ven::Colors:



### Static Public Attributes

- static constexpr glm::vec3 **WHITE** = glm::vec3(COLOR\_MAX, COLOR\_MAX, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **BLACK** = glm::vec3(0.0F)
- static constexpr glm::vec3 **RED** = glm::vec3(COLOR\_MAX, 0.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **GREEN** = glm::vec3(0.0F, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **BLUE** = glm::vec3(0.0F, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **YELLOW** = glm::vec3(COLOR\_MAX, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **CYAN** = glm::vec3(0.0F, COLOR\_MAX, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **MAGENTA** = glm::vec3(COLOR\_MAX, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **SILVER** = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR\_MAX
- static constexpr glm::vec3 **GRAY** = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec3 **MAROON** = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **OLIVE** = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **LIME** = glm::vec3(0.0F, COLOR\_MAX, 0.0F) / COLOR\_MAX
- static constexpr glm::vec3 **AQUA** = glm::vec3(0.0F, COLOR\_MAX, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **TEAL** = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec3 **NAVY** = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR\_MAX
- static constexpr glm::vec3 **FUCHSIA** = glm::vec3(COLOR\_MAX, 0.0F, COLOR\_MAX) / COLOR\_MAX
- static constexpr glm::vec3 **NIGHT\_BLUE** = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR\_MAX
- static constexpr glm::vec3 **SKY\_BLUE** = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR\_MAX
- static constexpr glm::vec3 **SUNSET** = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR\_MAX
- static constexpr VkClearColorValue **WHITE\_V** = {{1.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **BLACK\_V** = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **RED\_V** = {{1.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **GREEN\_V** = {{0.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **BLUE\_V** = {{0.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **YELLOW\_V** = {{1.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **CYAN\_V** = {{0.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **MAGENTA\_V** = {{1.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **SILVER\_V** = {{0.75F, 0.75F, 0.75F, 1.0F}}
- static constexpr VkClearColorValue **GRAY\_V** = {{0.5F, 0.5F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue **MAROON\_V** = {{0.5F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **OLIVE\_V** = {{0.5F, 0.5F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **LIME\_V** = {{0.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **AQUA\_V** = {{0.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **TEAL\_V** = {{0.0F, 0.5F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue **NAVY\_V** = {{0.0F, 0.0F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue **FUCHSIA\_V** = {{1.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue **NIGHT\_BLUE\_V** = {{0.1F, 0.1F, 0.44F, 1.0F}}
- static constexpr VkClearColorValue **SKY\_BLUE\_V** = {{0.4F, 0.6F, 0.9F, 1.0F}}
- static constexpr VkClearColorValue **SUNSET\_V** = {{1.0F, 0.5F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue **NIGHT\_MODE\_V** = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr std::array< std::pair< const char \*, glm::vec3 >, 20 > **COLORS**
- static constexpr std::array< std::pair< const char \*, VkClearColorValue >, 20 > **CLEAR\_COLORS**

### Static Private Attributes

- static constexpr float **COLOR\_MAX** = 255.0F

## 6.7.1 Detailed Description

Class for colors.

Definition at line 24 of file [Colors.hpp](#).

## 6.7.2 Member Data Documentation

### 6.7.2.1 AQUA

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

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

### 6.7.2.2 AQUA\_V

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

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

### 6.7.2.3 BLACK

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

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

### 6.7.2.4 BLACK\_V

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

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

### 6.7.2.5 BLUE

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

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

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

### 6.7.2.6 BLUE\_V

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

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

### 6.7.2.7 CLEAR\_COLORS

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

#### Initial value:

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

### 6.7.2.8 COLOR\_MAX

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

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

### 6.7.2.9 COLORS

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

#### Initial value:

```
= {{
    {"White", Colors::WHITE},
    {"Black", Colors::BLACK},
    {"Red", Colors::RED},
    {"Green", Colors::GREEN},
    {"Blue", Colors::BLUE},
    {"Yellow", Colors::YELLOW},
    {"Cyan", Colors::CYAN},
    {"Magenta", Colors::MAGENTA},
    {"Silver", Colors::SILVER},
    {"Gray", Colors::GRAY},
    {"Maroon", Colors::MAROON},
    {"Olive", Colors::OLIVE},
    {"Lime", Colors::LIME},
    {"Aqua", Colors::AQUA},
    {"Teal", Colors::TEAL},
    {"Navy", Colors::NAVY},
    {"Fuchsia", Colors::FUCHSIA},
    {"Night Blue", ven::Colors::NIGHT_BLUE},
    {"Sky Blue", Colors::SKY_BLUE},
    {"Sunset", Colors::SUNSET}
}}
```

Definition at line 73 of file [Colors.hpp](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 6.7.2.10 CYAN

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

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

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

#### 6.7.2.11 CYAN\_V

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

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

#### 6.7.2.12 FUCHSIA

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

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

#### 6.7.2.13 FUCHSIA\_V

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

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

#### 6.7.2.14 GRAY

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

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

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

#### 6.7.2.15 GRAY\_V

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

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

#### 6.7.2.16 GREEN

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

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

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

#### 6.7.2.17 GREEN\_V

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

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

#### 6.7.2.18 LIME

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

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

#### 6.7.2.19 LIME\_V

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

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

#### 6.7.2.20 MAGENTA

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

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

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

#### 6.7.2.21 MAGENTA\_V

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

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

#### 6.7.2.22 MAROON

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

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

#### 6.7.2.23 MAROON\_V

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

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

#### 6.7.2.24 NAVY

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

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

#### 6.7.2.25 NAVY\_V

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

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

#### 6.7.2.26 NIGHT\_BLUE

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

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

#### 6.7.2.27 NIGHT\_BLUE\_V

```
VkClearColorValue ven::Colors::NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}} [static], [constexpr]
```

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

#### 6.7.2.28 NIGHT\_MODE\_V

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

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

#### 6.7.2.29 OLIVE

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

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

#### 6.7.2.30 OLIVE\_V

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

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



### 6.7.2.31 RED

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

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

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

### 6.7.2.32 RED\_V

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

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

### 6.7.2.33 SILVER

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

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

### 6.7.2.34 SILVER\_V

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

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

### 6.7.2.35 SKY\_BLUE

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

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

### 6.7.2.36 SKY\_BLUE\_V

```
VkClearColorValue ven::Colors::SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}} [static], [constexpr]
```

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

### 6.7.2.37 SUNSET

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

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

#### 6.7.2.38 SUNSET\_V

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

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

#### 6.7.2.39 TEAL

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

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

#### 6.7.2.40 TEAL\_V

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

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

#### 6.7.2.41 WHITE

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

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

#### 6.7.2.42 WHITE\_V

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

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

#### 6.7.2.43 YELLOW

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

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

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

#### 6.7.2.44 YELLOW\_V

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

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

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

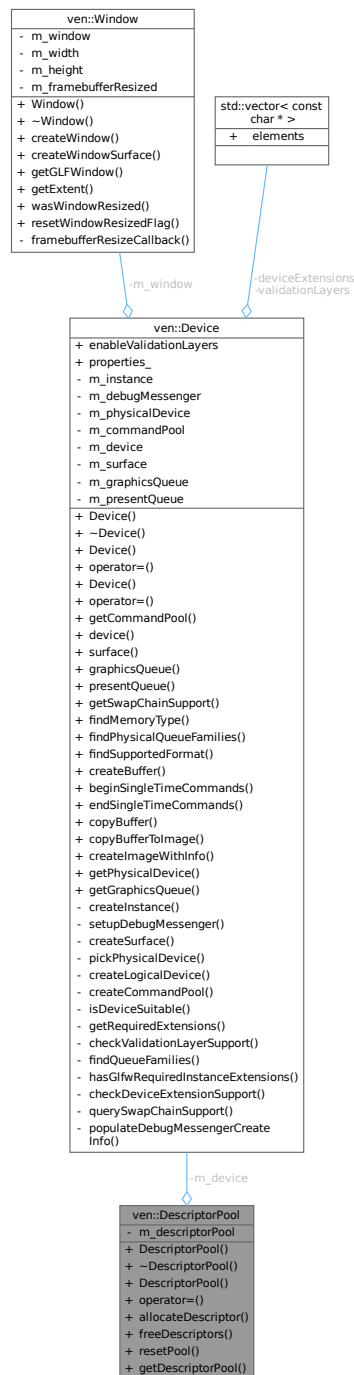
- [/home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp](#)

## 6.8 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](#)

## 6.8.1 Detailed Description

Class for descriptor pool.

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

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 DescriptorPool() [1/2]

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

Definition at line 20 of file [descriptorPool.cpp](#).

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



### 6.8.2.2 ~DescriptorPool()

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

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

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



### 6.8.2.3 DescriptorPool() [2/2]

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

## 6.8.3 Member Function Documentation

### 6.8.3.1 allocateDescriptor()

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

Definition at line 35 of file [descriptorPool.cpp](#).

### 6.8.3.2 freeDescriptors()

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

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

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



### 6.8.3.3 getDescriptorPool()

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

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

References [m\\_descriptorPool](#).

### 6.8.3.4 operator=()

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

### 6.8.3.5 resetPool()

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

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

References [ven::Device::device\(\)](#), [m\\_descriptorPool](#), and [m\\_device](#).

Here is the call graph for this function:



## 6.8.4 Friends And Related Symbol Documentation

### 6.8.4.1 DescriptorWriter

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

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

## 6.8.5 Member Data Documentation

### 6.8.5.1 m\_descriptorPool

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

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

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

### 6.8.5.2 m\_device

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

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

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

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

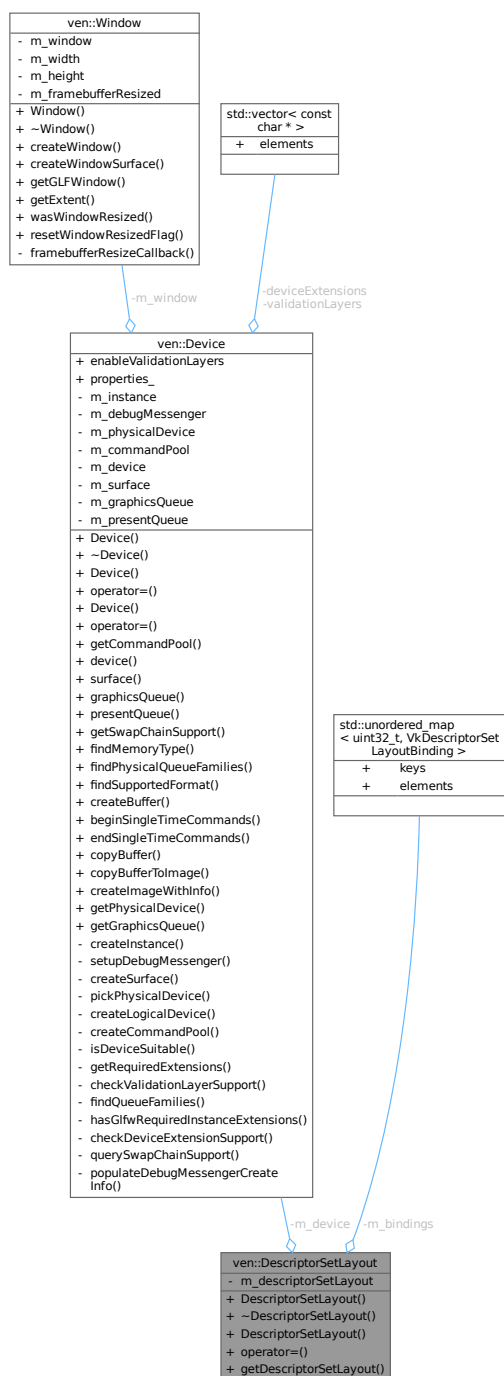
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp](#)

## 6.9 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](#)

## 6.9.1 Detailed Description

Class for descriptor set layout.

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

## 6.9.2 Constructor & Destructor Documentation

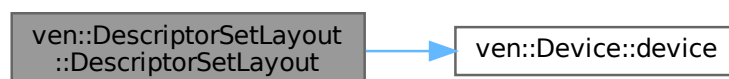
### 6.9.2.1 DescriptorSetLayout() [1/2]

```
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:



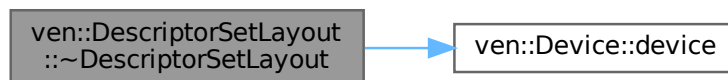
### 6.9.2.2 ~DescriptorSetLayout()

```
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:



### 6.9.2.3 DescriptorSetLayout() [2/2]

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

## 6.9.3 Member Function Documentation

### 6.9.3.1 getDescriptorSetLayout()

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

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

References [m\\_descriptorSetLayout](#).

### 6.9.3.2 operator=()

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

## 6.9.4 Friends And Related Symbol Documentation

### 6.9.4.1 DescriptorWriter

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

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

## 6.9.5 Member Data Documentation

### 6.9.5.1 m\_bindings

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

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

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

### 6.9.5.2 m\_descriptorSetLayout

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

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

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

### 6.9.5.3 m\_device

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

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

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

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

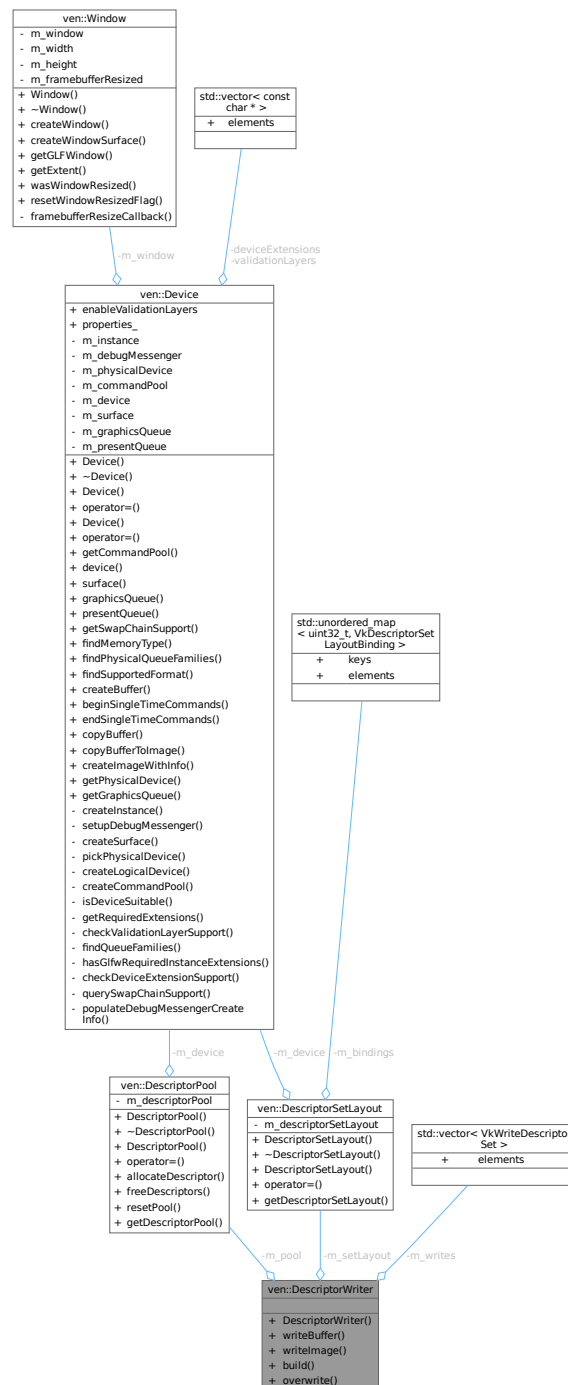
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#)

## 6.10 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 &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](#)

## 6.10.1 Detailed Description

Class for descriptor writer.

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

## 6.10.2 Constructor & Destructor Documentation

### 6.10.2.1 DescriptorWriter()

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

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

## 6.10.3 Member Function Documentation

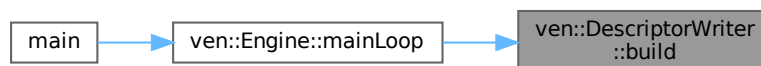
### 6.10.3.1 build()

```
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:



### 6.10.3.2 overwrite()

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

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

### 6.10.3.3 writeBuffer()

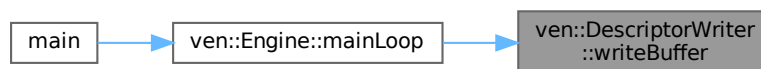
```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
    uint32_t binding,
    const VkDescriptorBufferInfo * bufferInfo)
```

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:



### 6.10.3.4 writelImage()

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

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

## 6.10.4 Member Data Documentation

### 6.10.4.1 m\_pool

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

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

### 6.10.4.2 m\_setLayout

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

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

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

#### 6.10.4.3 m\_writes

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

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

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

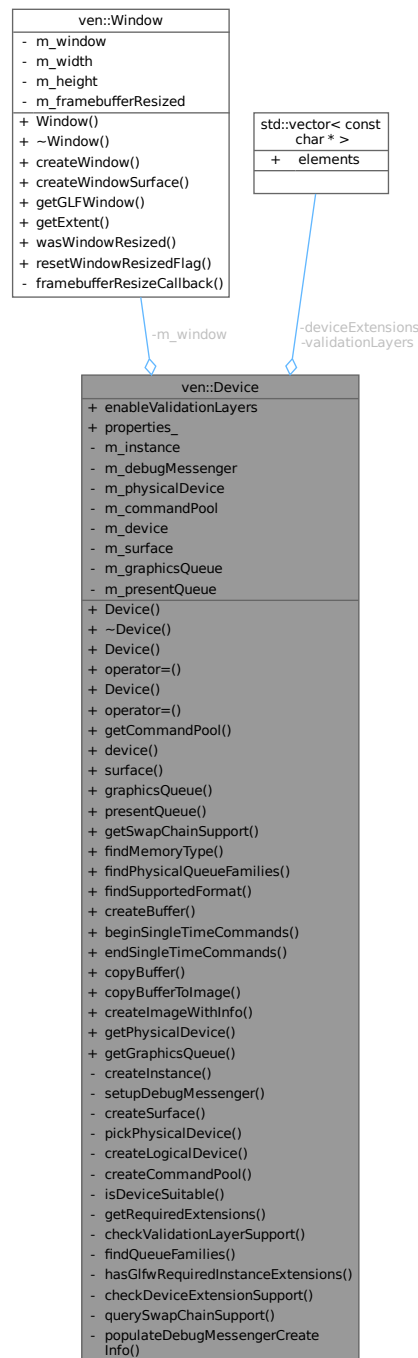
The documentation for this class was generated from the following files:

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

## 6.11 ven::Device Class Reference

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

Collaboration diagram for ven::Device:



## Public Member Functions

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



- [Device](#) & [operator=](#) ([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](#) propertiesp) const
- [QueueFamilyIndices](#) [findPhysicalQueueFamilies](#) () const
- [VkFormat](#) [findSupportedFormat](#) (const std::vector< [VkFormat](#) > &candidates, [VkImageTiling](#) tiling, [VkFormatFeatureFlags](#) features) const
- void [createBuffer](#) ([VkDeviceSize](#) size, [VkBufferUsageFlags](#) usage, [VkMemoryPropertyFlags](#) propertiesp, [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
- [VkPhysicalDevice](#) [getPhysicalDevice](#) () const
- [VkQueue](#) [getGraphicsQueue](#) () const

## Public Attributes

- const bool [enableValidationLayers](#) = true
- [VkPhysicalDeviceProperties](#) [properties\\_](#)

## 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](#)
- const std::vector< const char \* > [validationLayers](#) = {"VK\_LAYER\_KHRONOS\_validation"}
- const std::vector< const char \* > [deviceExtensions](#) = {VK\_KHR\_SWAPCHAIN\_EXTENSION\_NAME}

### 6.11.1 Detailed Description

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

### 6.11.2 Constructor & Destructor Documentation

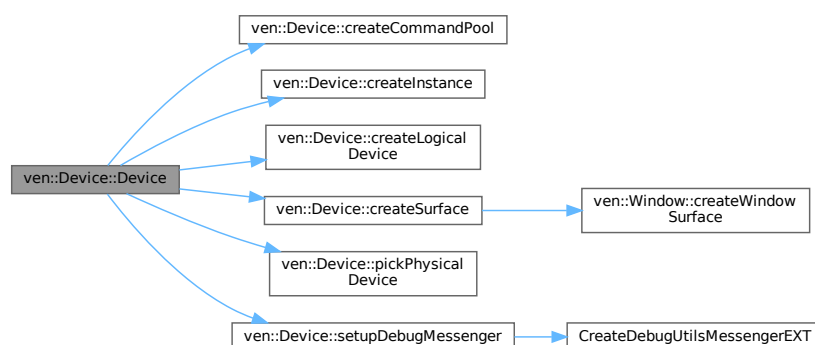
#### 6.11.2.1 Device() [1/3]

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

Definition at line 34 of file [device.cpp](#).

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

Here is the call graph for this function:



### 6.11.2.2 ~Device()

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

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

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

Here is the call graph for this function:



### 6.11.2.3 Device() [2/3]

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

### 6.11.2.4 Device() [3/3]

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

## 6.11.3 Member Function Documentation

### 6.11.3.1 beginSingleTimeCommands()

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

Definition at line 411 of file [device.cpp](#).

### 6.11.3.2 checkDeviceExtensionSupport()

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

Definition at line 289 of file [device.cpp](#).

### 6.11.3.3 checkValidationLayerSupport()

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

Definition at line 225 of file [device.cpp](#).

#### 6.11.3.4 copyBuffer()

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

Definition at line 445 of file [device.cpp](#).

#### 6.11.3.5 copyBufferToImage()

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

Definition at line 458 of file [device.cpp](#).

#### 6.11.3.6 createBuffer()

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

Definition at line 384 of file [device.cpp](#).

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

Here is the caller graph for this function:



### 6.11.3.7 createCommandPool()

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

Definition at line 171 of file [device.cpp](#).

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

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

Here is the caller graph for this function:



### 6.11.3.8 createImageWithInfo()

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

Definition at line 479 of file [device.cpp](#).

### 6.11.3.9 createInstance()

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

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

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

Here is the caller graph for this function:



### 6.11.3.10 createLogicalDevice()

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

Definition at line 124 of file [device.cpp](#).

References [ven::QueueFamilyIndices::graphicsFamily](#), and [ven::QueueFamilyIndices::presentFamily](#).

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

Here is the caller graph for this function:



### 6.11.3.11 createSurface()

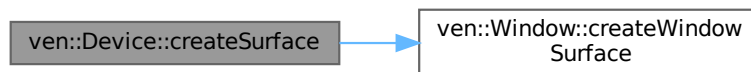
```
void ven::Device::createSurface () [inline], [private]
```

Definition at line 76 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:



### 6.11.3.12 device()

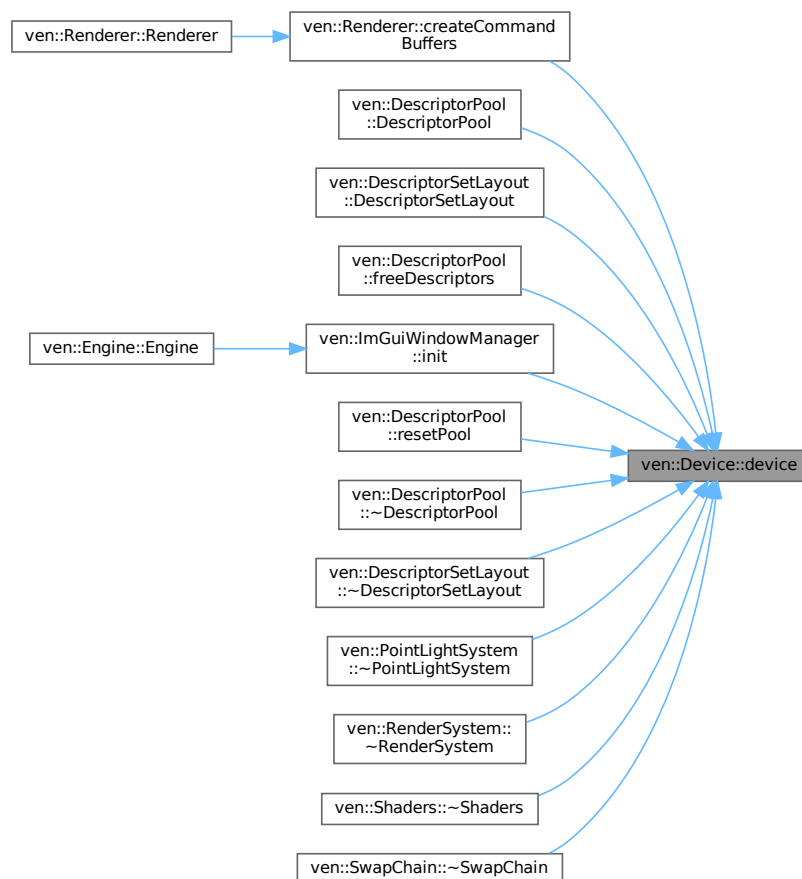
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

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

References [m\\_device](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::DescriptorPool::DescriptorPool\(\)](#), [ven::DescriptorSetLayout::DescriptorSetLayout\(\)](#), [ven::DescriptorPool::freeDescriptors\(\)](#), [ven::ImGuiWindowManager::init\(\)](#), [ven::DescriptorPool::resetPool\(\)](#), [ven::DescriptorPool::~DescriptorPool\(\)](#), [ven::DescriptorSetLayout::~DescriptorSetLayout\(\)](#), [ven::PointLightSystem::~PointLightSystem\(\)](#), [ven::RenderSystem::~RenderSystem\(\)](#), [ven::Shaders::~Shaders\(\)](#), and [ven::SwapChain::~SwapChain\(\)](#).

Here is the caller graph for this function:



### 6.11.3.13 endSingleTimeCommands()

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

Definition at line 430 of file [device.cpp](#).

### 6.11.3.14 findMemoryType()

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

Definition at line 369 of file [device.cpp](#).

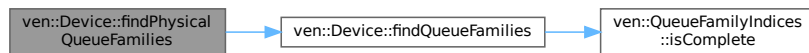
### 6.11.3.15 findPhysicalQueueFamilies()

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

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

References [findQueueFamilies\(\)](#), and [m\\_physicalDevice](#).

Here is the call graph for this function:



### 6.11.3.16 findQueueFamilies()

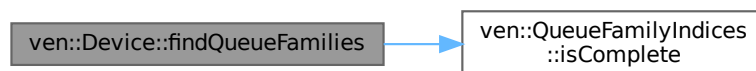
```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
    VkPhysicalDevice device) const [private]
```

Definition at line 305 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:





**6.11.3.17 findSupportedFormat()**

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

Definition at line 355 of file [device.cpp](#).

**6.11.3.18 getCommandPool()**

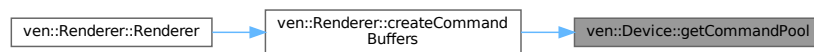
```
VkCommandPool ven::Device::getCommandPool () const [inline], [nodiscard]
```

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

References [m\\_commandPool](#).

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

Here is the caller graph for this function:

**6.11.3.19 getGraphicsQueue()**

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

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

References [m\\_graphicsQueue](#).

**6.11.3.20 getPhysicalDevice()**

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

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

References [m\\_physicalDevice](#).

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

Here is the caller graph for this function:



### 6.11.3.21 getRequiredExtensions()

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

Definition at line 250 of file [device.cpp](#).

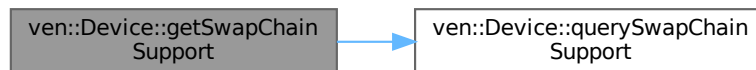
### 6.11.3.22 getSwapChainSupport()

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

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

References [m\\_physicalDevice](#), and [querySwapChainSupport\(\)](#).

Here is the call graph for this function:



### 6.11.3.23 graphicsQueue()

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

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

References [m\\_graphicsQueue](#).

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

Here is the caller graph for this function:



### 6.11.3.24 hasGlfwRequiredInstanceExtensions()

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

Definition at line 265 of file [device.cpp](#).

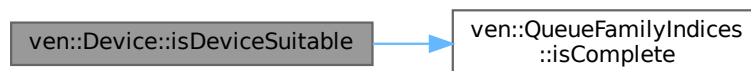
### 6.11.3.25 isDeviceSuitable()

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

Definition at line 185 of file [device.cpp](#).

References [ven::SwapChainSupportDetails::formats](#), [ven::QueueFamilyIndices::isComplete\(\)](#), and [ven::SwapChainSupportDetails::p](#)

Here is the call graph for this function:



### 6.11.3.26 operator=() [1/2]

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

### 6.11.3.27 operator=() [2/2]

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

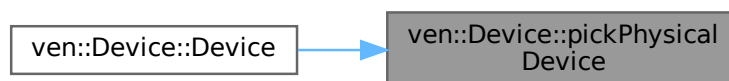
### 6.11.3.28 pickPhysicalDevice()

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

Definition at line 98 of file [device.cpp](#).

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

Here is the caller graph for this function:



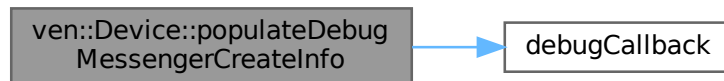
### 6.11.3.29 populateDebugMessengerCreateInfo()

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

Definition at line 202 of file [device.cpp](#).

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

Here is the call graph for this function:



### 6.11.3.30 presentQueue()

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

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

References [m\\_presentQueue](#).

### 6.11.3.31 querySwapChainSupport()

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

Definition at line 334 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:



### 6.11.3.32 setupDebugMessenger()

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

Definition at line 215 of file [device.cpp](#).

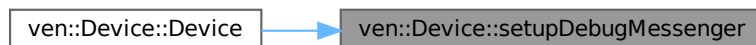
References [CreateDebugUtilsMessengerEXT\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.33 surface()

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

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

References [m\\_surface](#).

## 6.11.4 Member Data Documentation

### 6.11.4.1 deviceExtensions

```
const std::vector<const char *> ven::Device::deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_↵  
NAME} [private]
```

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

#### 6.11.4.2 enableValidationLayers

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

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

#### 6.11.4.3 m\_commandPool

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

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

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

#### 6.11.4.4 m\_debugMessenger

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

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

#### 6.11.4.5 m\_device

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

Definition at line 97 of file [Device.hpp](#).

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

#### 6.11.4.6 m\_graphicsQueue

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

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

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

#### 6.11.4.7 m\_instance

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

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

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

#### 6.11.4.8 m\_physicalDevice

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

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

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

#### 6.11.4.9 m\_presentQueue

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

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

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

#### 6.11.4.10 m\_surface

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

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

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

#### 6.11.4.11 m\_window

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

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

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

#### 6.11.4.12 properties\_

```
VkPhysicalDeviceProperties ven::Device::properties_
```

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

#### 6.11.4.13 validationLayers

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

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

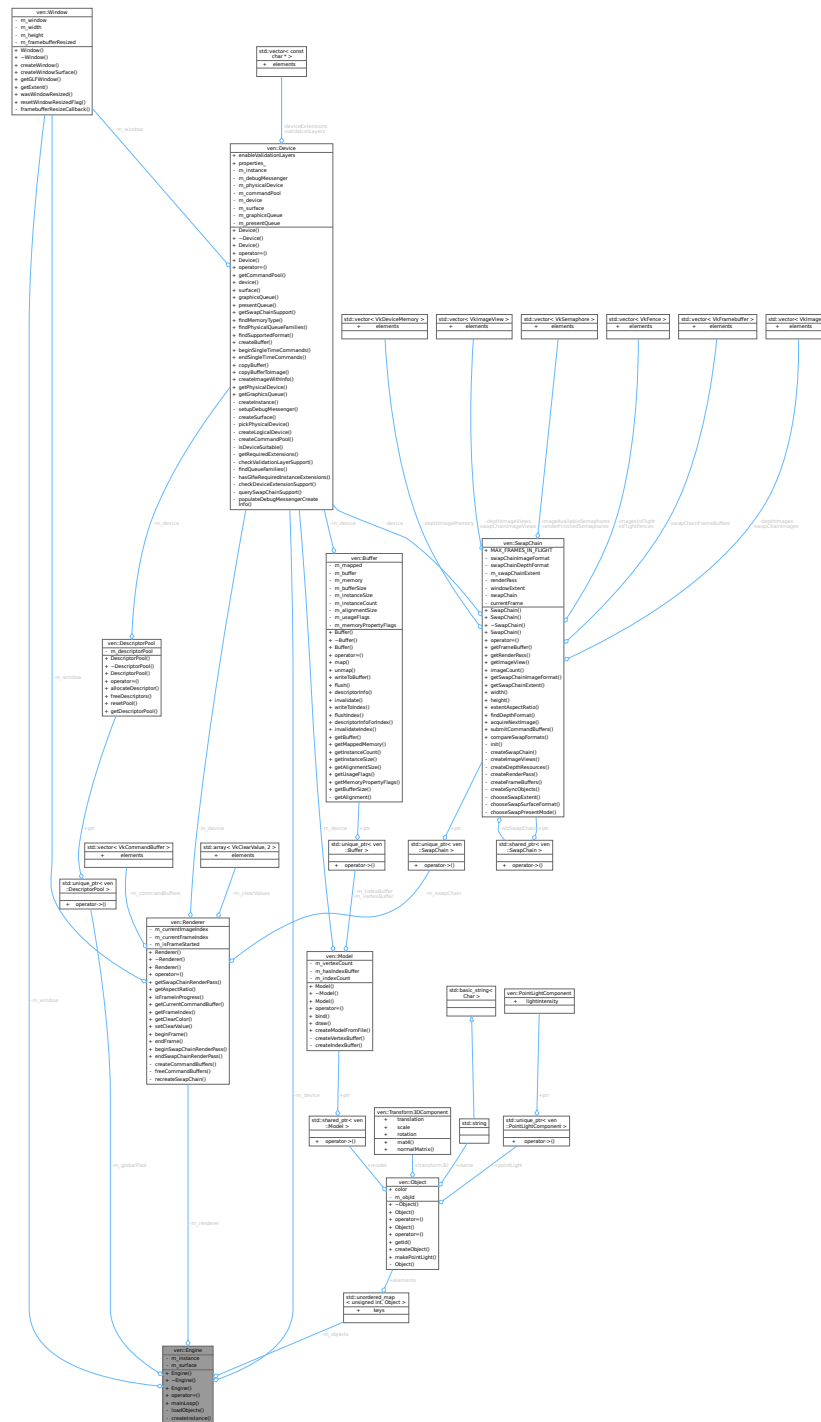
The documentation for this class was generated from the following files:

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

## 6.12 ven::Engine Class Reference

```
#include <Engine.hpp>
```

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

- [Window](#) [m\\_window](#)
- [Device](#) [m\\_device](#) {[m\\_window](#)}
- [Renderer](#) [m\\_renderer](#) {[m\\_window](#), [m\\_device](#)}
- std::unique\_ptr< [DescriptorPool](#) > [m\\_globalPool](#)
- [Object::Map](#) [m\\_objects](#)
- [VkInstance](#) [m\\_instance](#) {nullptr}
- [VkSurfaceKHR](#) [m\\_surface](#) {nullptr}

## 6.12.1 Detailed Description

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

## 6.12.2 Constructor & Destructor Documentation

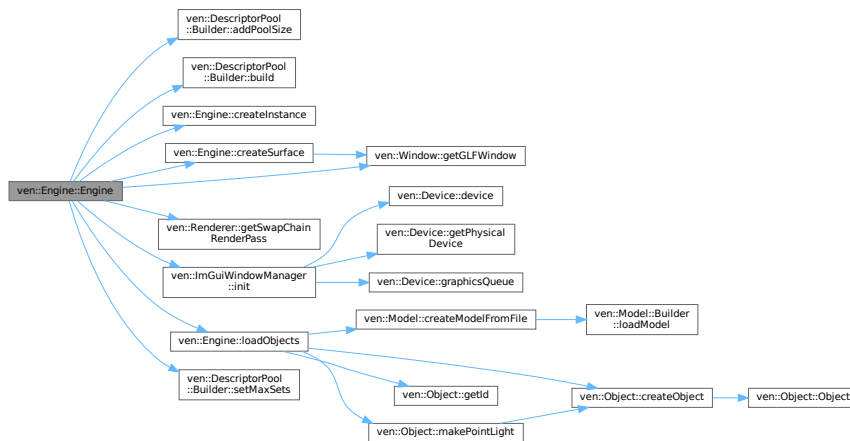
### 6.12.2.1 Engine() [1/2]

```
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::Window::getGLFWWindow\(\)](#), [ven::Renderer::getSwapChainRenderPass\(\)](#), [ven::ImGuiWindowManager::init\(\)](#), [loadObjects\(\)](#), [m\\_device](#), [m\\_globalPool](#), [m\\_instance](#), [m\\_renderer](#), [m\\_window](#), [ven::SwapChain::MAX\\_FRAMES\\_IN\\_FLIGHT](#), and [ven::DescriptorPool::Builder::setMaxSets\(\)](#).

Here is the call graph for this function:



### 6.12.2.2 ~Engine()

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

### 6.12.2.3 Engine() [2/2]

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

## 6.12.3 Member Function Documentation

### 6.12.3.1 createInstance()

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

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

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

Here is the caller graph for this function:



### 6.12.3.2 createSurface()

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

Definition at line 49 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:



### 6.12.3.3 loadObjects()

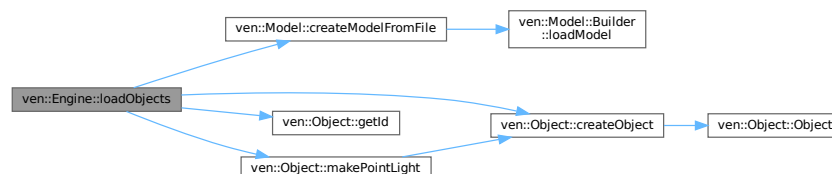
`void ven::Engine::loadObjects () [private]`

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

References [ven::Colors::BLUE](#), [ven::Object::color](#), [ven::Model::createModelFromFile\(\)](#), [ven::Object::createObject\(\)](#), [ven::Colors::CYAN](#), [ven::Object::getId\(\)](#), [ven::Colors::GREEN](#), [ven::Colors::MAGENTA](#), [ven::Object::makePointLight\(\)](#), [ven::Object::model](#), [ven::Object::name](#), [ven::Colors::RED](#), [ven::Transform3DComponent::scale](#), [ven::Object::transform3D](#), [ven::Transform3DComponent::translation](#), and [ven::Colors::YELLOW](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.3.4 mainLoop()

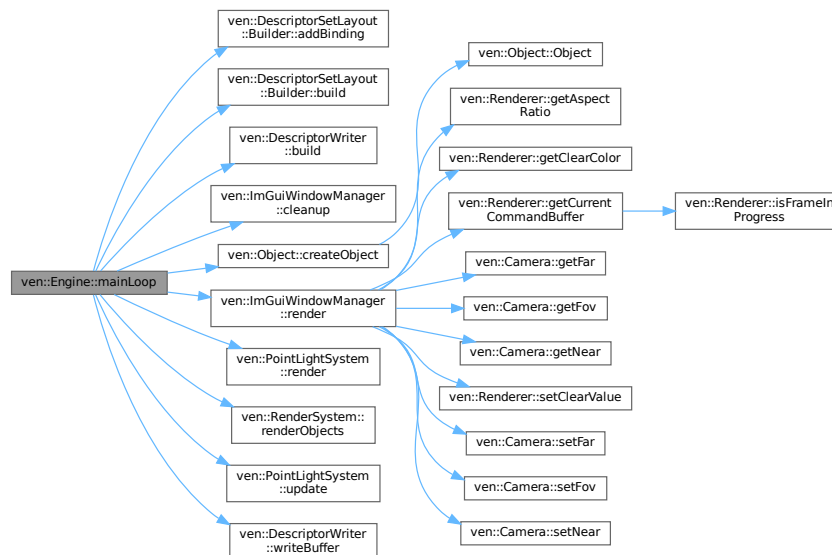
```
void ven::Engine::mainLoop ()
```

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

References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::DescriptorWriter::build\(\)](#), [ven::ImGuiWindowManager::cleanup\(\)](#), [ven::Object::createObject\(\)](#), [ven::DEFAULT\\_POSITION](#), [ven::SwapChain::MAX\\_FRAMES\\_IN\\_QUEUE](#), [ven::ImGuiWindowManager::render\(\)](#), [ven::PointLightSystem::render\(\)](#), [ven::RenderSystem::renderObjects\(\)](#), [ven::Transform3DComponent::rotation](#), [ven::Object::transform3D](#), [ven::Transform3DComponent::translation](#), [ven::PointLightSystem::update\(\)](#), and [ven::DescriptorWriter::writeBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.12.3.5 operator=()

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

## 6.12.4 Member Data Documentation

### 6.12.4.1 m\_device

```
Device ven::Engine::m_device {m_window} [private]
```

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

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

### 6.12.4.2 m\_globalPool

```
std::unique_ptr<DescriptorPool> ven::Engine::m_globalPool [private]
```

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

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

### 6.12.4.3 m\_instance

```
VkInstance ven::Engine::m_instance {nullptr} [private]
```

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

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

### 6.12.4.4 m\_objects

```
Object::Map ven::Engine::m_objects [private]
```

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

#### 6.12.4.5 m\_renderer

```
Renderer ven::Engine::m_renderer {m_window, m_device} [private]
```

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

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

#### 6.12.4.6 m\_surface

```
VkSurfaceKHR ven::Engine::m_surface {nullptr} [private]
```

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

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

#### 6.12.4.7 m\_window

```
Window ven::Engine::m_window [private]
```

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

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

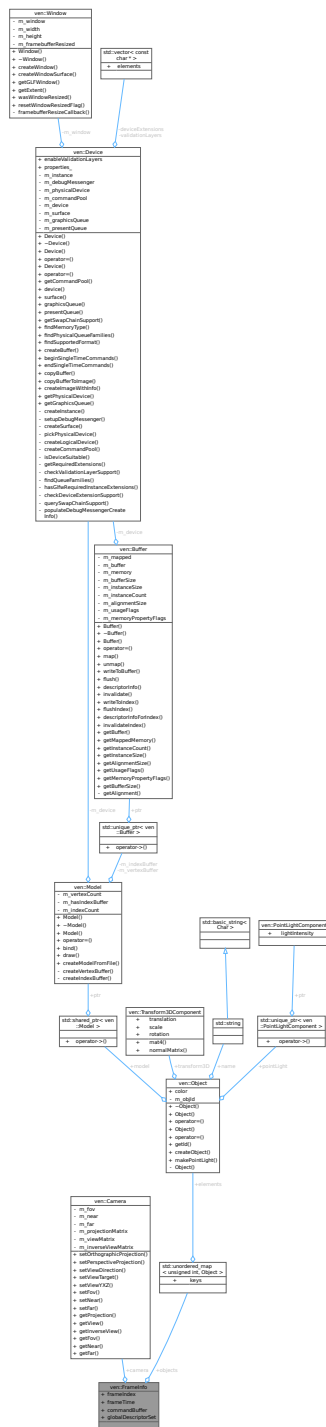
The documentation for this class was generated from the following files:

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

### 6.13 ven::FrameInfo Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::FrameInfo:



## Public Attributes

- int `frameIndex`
- float `frameTime`
- `VkCommandBuffer` `commandBuffer`
- `Camera` & `camera`
- `VkDescriptorSet` `globalDescriptorSet`
- `Object::Map` & `objects`

### 6.13.1 Detailed Description

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

### 6.13.2 Member Data Documentation

#### 6.13.2.1 camera

```
Camera& ven::FrameInfo::camera
```

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

#### 6.13.2.2 commandBuffer

```
VkCommandBuffer ven::FrameInfo::commandBuffer
```

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

Referenced by [ven::PointLightSystem::render\(\)](#), and [ven::RenderSystem::renderObjects\(\)](#).

#### 6.13.2.3 frameIndex

```
int ven::FrameInfo::frameIndex
```

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

#### 6.13.2.4 frameTime

```
float ven::FrameInfo::frameTime
```

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

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

#### 6.13.2.5 globalDescriptorSet

```
VkDescriptorSet ven::FrameInfo::globalDescriptorSet
```

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

Referenced by [ven::PointLightSystem::render\(\)](#), and [ven::RenderSystem::renderObjects\(\)](#).



### 6.13.2.6 objects

`Object::Map& ven::FrameInfo::objects`

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

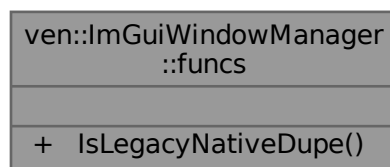
Referenced by [ven::PointLightSystem::render\(\)](#), [ven::RenderSystem::renderObjects\(\)](#), and [ven::PointLightSystem::update\(\)](#).

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

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

## 6.14 ven::ImGuiWindowManager::funcs Struct Reference

Collaboration diagram for `ven::ImGuiWindowManager::funcs`:



### Static Public Member Functions

- static bool [IsLegacyNativeDupe](#) (ImGuiKey key)

### 6.14.1 Detailed Description

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

## 6.14.2 Member Function Documentation

### 6.14.2.1 IsLegacyNativeDupe()

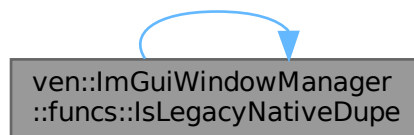
```
static bool ven::ImGuiWindowManager::funcs::IsLegacyNativeDupe (  
    ImGuiKey key) [inline], [static]
```

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

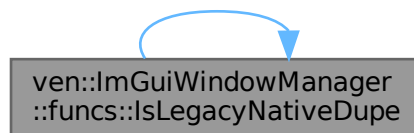
References [IsLegacyNativeDupe\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



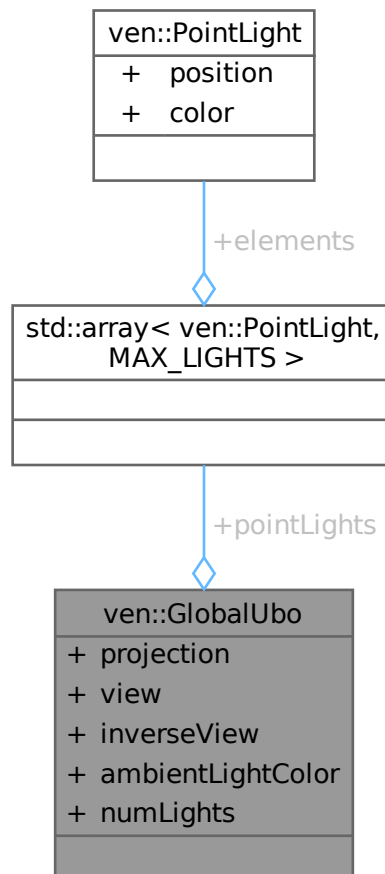
The documentation for this struct was generated from the following file:

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

## 6.15 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](#) {1.F, 1.F, 1.F, .02F}
- std::array< [PointLight](#), [MAX\\_LIGHTS](#) > [pointLights](#)
- int [numLights](#)

### 6.15.1 Detailed Description

Definition at line 24 of file [FrameInfo.hpp](#).

## 6.15.2 Member Data Documentation

### 6.15.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {1.F, 1.F, 1.F, .02F}
```

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

### 6.15.2.2 inverseView

```
glm::mat4 ven::GlobalUbo::inverseView {1.F}
```

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

### 6.15.2.3 numLights

```
int ven::GlobalUbo::numLights
```

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

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

### 6.15.2.4 pointLights

```
std::array<PointLight, MAX_LIGHTS> ven::GlobalUbo::pointLights
```

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

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

### 6.15.2.5 projection

```
glm::mat4 ven::GlobalUbo::projection {1.F}
```

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

### 6.15.2.6 view

```
glm::mat4 ven::GlobalUbo::view {1.F}
```

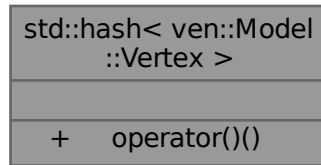
Definition at line 27 of file [FrameInfo.hpp](#).

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

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

## 6.16 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`

#### 6.16.1 Detailed Description

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

#### 6.16.2 Member Function Documentation

##### 6.16.2.1 operator()

```
size_t std::hash< ven::Model::Vertex >::operator() (
    ven::Model::Vertex const & vertex) const [inline]
```

Definition at line 16 of file [model.cpp](#).

References [ven::Model::Vertex::color](#), [ven::hashCombine\(\)](#), [ven::Model::Vertex::normal](#), [ven::Model::Vertex::position](#), and [ven::Model::Vertex::uv](#).

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](#)

## 6.17 ven::ImGuiWindowManager Class Reference

Class for ImGui window manager.

```
#include <ImGuiWindowManager.hpp>
```

Collaboration diagram for ven::ImGuiWindowManager:

ven::ImGuiWindowManager
<ul style="list-style-type: none"> <li>+ ImGuiWindowManager()</li> <li>+ ~ImGuiWindowManager()</li> <li>+ ImGuiWindowManager()</li> <li>+ operator=()</li> <li>+ init()</li> <li>+ render()</li> <li>+ cleanup()</li> </ul>

### Classes

- struct [funcs](#)

### Public Member Functions

- [ImGuiWindowManager](#) ()=default
- [~ImGuiWindowManager](#) ()=default
- [ImGuiWindowManager](#) (const [ImGuiWindowManager](#) &)=delete
- [ImGuiWindowManager](#) & [operator=](#) (const [ImGuiWindowManager](#) &)=delete

### Static Public Member Functions

- static void [init](#) (GLFWwindow \*window, VkInstance instance, [Device](#) \*device, VkRenderPass renderPass)
- static void [render](#) ([Renderer](#) \*renderer, std::unordered\_map< unsigned int, [Object](#) > &objects, ImGuiIO &io, [Object](#) &cameraObj, [Camera](#) &camera, [KeyboardController](#) &cameraController, VkPhysicalDevice physicalDevice)
- static void [cleanup](#) ()

### 6.17.1 Detailed Description

Class for ImGui window manager.

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

## 6.17.2 Constructor & Destructor Documentation

### 6.17.2.1 ImGuiWindowManager() [1/2]

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

### 6.17.2.2 ~ImGuiWindowManager()

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

### 6.17.2.3 ImGuiWindowManager() [2/2]

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

## 6.17.3 Member Function Documentation

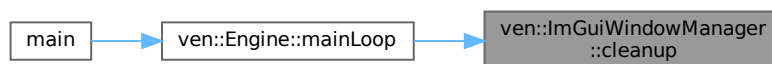
### 6.17.3.1 cleanup()

```
void ven::ImGuiWindowManager::cleanup () [static]
```

Definition at line 238 of file [ImGuiWindowManager.cpp](#).

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

Here is the caller graph for this function:



### 6.17.3.2 init()

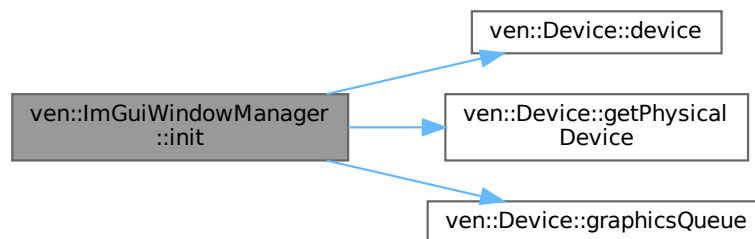
```
void ven::ImGuiWindowManager::init (
    GLFWwindow * window,
    VkInstance instance,
    Device * device,
    VkRenderPass renderPass) [static]
```

Definition at line 8 of file [ImGuiWindowManager.cpp](#).

References [ven::Device::device\(\)](#), [ven::Device::getPhysicalDevice\(\)](#), and [ven::Device::graphicsQueue\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.17.3.3 operator=()

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



## 6.17.3.4 render()

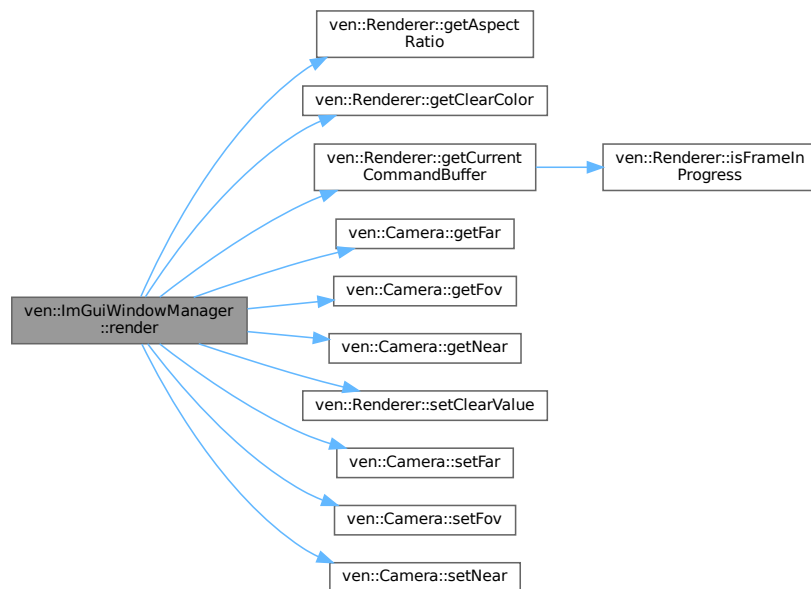
```
void ven::ImGuiWindowManager::render (
    Renderer * renderer,
    std::unordered_map< unsigned int, Object > & objects,
    ImGuiIO & io,
    Object & cameraObj,
    Camera & camera,
    KeyboardController & cameraController,
    VkPhysicalDevice physicalDevice) [static]
```

Definition at line 54 of file [ImGuiWindowManager.cpp](#).

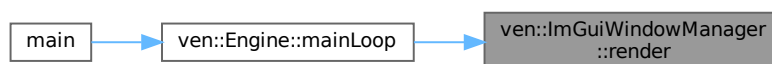
References [ven::Colors::CLEAR\\_COLORS](#), [ven::Colors::COLORS](#), [ven::DEFAULT\\_FAR](#), [ven::DEFAULT\\_FOV](#), [ven::DEFAULT\\_LOOK\\_SPEED](#), [ven::DEFAULT\\_MOVE\\_SPEED](#), [ven::DEFAULT\\_NEAR](#), [ven::DEFAULT\\_POSITION](#), [ven::DEFAULT\\_ROTATION](#), [ven::Renderer::getAspectRatio\(\)](#), [ven::Renderer::getClearColor\(\)](#), [ven::Renderer::getCurrentCommandBuffer](#), [ven::Camera::getFar\(\)](#), [ven::Camera::getFov\(\)](#), [ven::Camera::getNear\(\)](#), [ven::Colors::GRAY](#), [ven::KeyboardController::m\\_lookSpeed](#), [ven::KeyboardController::m\\_moveSpeed](#), [ven::Transform3DComponent::rotation](#), [ven::Renderer::setClearColor\(\)](#), [ven::Camera::setFar\(\)](#), [ven::Camera::setFov\(\)](#), [ven::Camera::setNear\(\)](#), [ven::Object::transform3D](#), and [ven::Transform3DComponent::rotation](#).

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:

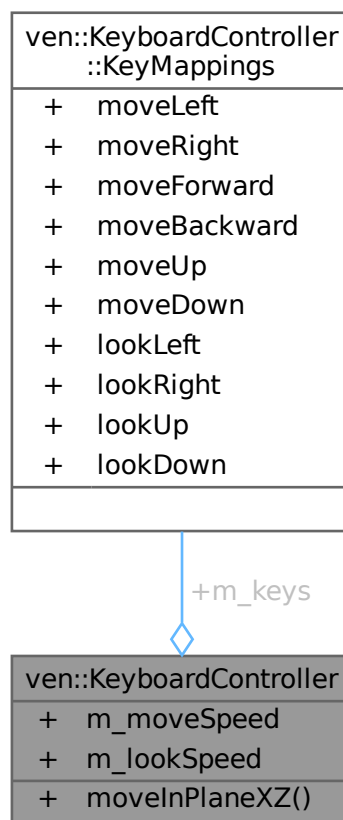
- [/home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp](#)

## 6.18 ven::KeyboardController Class Reference

Class for keyboard controller.

```
#include <KeyboardController.hpp>
```

Collaboration diagram for ven::KeyboardController:



### Classes

- struct [KeyMappings](#)

### Public Member Functions

- void [moveInPlaneXZ](#) (GLFWwindow \*window, float dt, [Object](#) &object, bool \*showDebugWindow) const

## Public Attributes

- [KeyMappings m\\_keys](#) {}
- float [m\\_moveSpeed](#) {DEFAULT\_MOVE\_SPEED}
- float [m\\_lookSpeed](#) {DEFAULT\_LOOK\_SPEED}

### 6.18.1 Detailed Description

Class for keyboard controller.

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

### 6.18.2 Member Function Documentation

#### 6.18.2.1 moveInPlaneXZ()

```
void ven::KeyboardController::moveInPlaneXZ (  
    GLFWwindow * window,  
    float dt,  
    Object & object,  
    bool * showDebugWindow) const
```

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

References [ven::KeyboardController::KeyMappings::lookDown](#), [ven::KeyboardController::KeyMappings::lookLeft](#), [ven::KeyboardController::KeyMappings::lookRight](#), [ven::KeyboardController::KeyMappings::lookUp](#), [m\\_keys](#), [m\\_lookSpeed](#), [m\\_moveSpeed](#), [ven::KeyboardController::KeyMappings::moveBackward](#), [ven::KeyboardController::KeyMappings::moveForward](#), [ven::KeyboardController::KeyMappings::moveLeft](#), [ven::KeyboardController::KeyMappings::moveRight](#), and [ven::KeyboardController::KeyMappings::moveUp](#).

### 6.18.3 Member Data Documentation

#### 6.18.3.1 m\_keys

```
KeyMappings ven::KeyboardController::m_keys {}
```

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

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

#### 6.18.3.2 m\_lookSpeed

```
float ven::KeyboardController::m_lookSpeed {DEFAULT_LOOK_SPEED}
```

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

Referenced by [moveInPlaneXZ\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

### 6.18.3.3 m\_moveSpeed

```
float ven::KeyboardController::m_moveSpeed {DEFAULT_MOVE_SPEED}
```

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

Referenced by [moveInPlaneXZ\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

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

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

## 6.19 ven::KeyboardController::KeyMappings Struct Reference

```
#include <KeyboardController.hpp>
```

Collaboration diagram for `ven::KeyboardController::KeyMappings`:

ven::KeyboardController ::KeyMappings	
+	moveLeft
+	moveRight
+	moveForward
+	moveBackward
+	moveUp
+	moveDown
+	lookLeft
+	lookRight
+	lookUp
+	lookDown

### Public Attributes

- int [moveLeft](#) = GLFW\_KEY\_A
- int [moveRight](#) = GLFW\_KEY\_D
- int [moveForward](#) = GLFW\_KEY\_W
- int [moveBackward](#) = GLFW\_KEY\_S
- int [moveUp](#) = GLFW\_KEY\_SPACE
- int [moveDown](#) = GLFW\_KEY\_LEFT\_SHIFT
- int [lookLeft](#) = GLFW\_KEY\_LEFT
- int [lookRight](#) = GLFW\_KEY\_RIGHT
- int [lookUp](#) = GLFW\_KEY\_UP
- int [lookDown](#) = GLFW\_KEY\_DOWN

### 6.19.1 Detailed Description

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

### 6.19.2 Member Data Documentation

#### 6.19.2.1 lookDown

```
int ven::KeyboardController::KeyMappings::lookDown = GLFW_KEY_DOWN
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.2 lookLeft

```
int ven::KeyboardController::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.3 lookRight

```
int ven::KeyboardController::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.4 lookUp

```
int ven::KeyboardController::KeyMappings::lookUp = GLFW_KEY_UP
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.5 moveBackward

```
int ven::KeyboardController::KeyMappings::moveBackward = GLFW_KEY_S
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.6 moveDown

```
int ven::KeyboardController::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.7 moveForward

```
int ven::KeyboardController::KeyMappings::moveForward = GLFW_KEY_W
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.8 moveLeft

```
int ven::KeyboardController::KeyMappings::moveLeft = GLFW_KEY_A
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.9 moveRight

```
int ven::KeyboardController::KeyMappings::moveRight = GLFW_KEY_D
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

#### 6.19.2.10 moveUp

```
int ven::KeyboardController::KeyMappings::moveUp = GLFW_KEY_SPACE
```

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

Referenced by [ven::KeyboardController::moveInPlaneXZ\(\)](#).

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

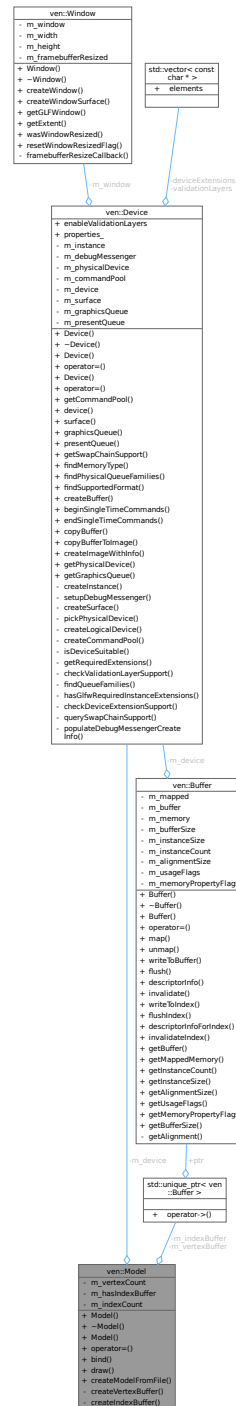
- [/home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp](#)

## 6.20 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](#) ()
- [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](#)

## 6.20.1 Detailed Description

Class for model.

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

## 6.20.2 Constructor & Destructor Documentation

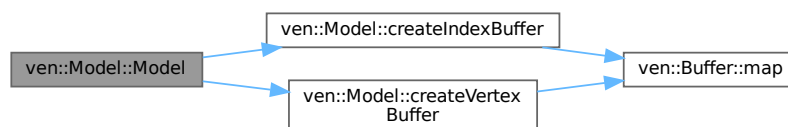
### 6.20.2.1 Model() [1/2]

```
ven::Model::Model (
    Device & device,
    const Builder & builder)
```

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

References [createIndexBuffer\(\)](#), [createVertexBuffer\(\)](#), [ven::Model::Builder::indices](#), and [ven::Model::Builder::vertices](#).

Here is the call graph for this function:





### 6.20.2.2 ~Model()

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

### 6.20.2.3 Model() [2/2]

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

## 6.20.3 Member Function Documentation

### 6.20.3.1 bind()

```
void ven::Model::bind (  
    VkCommandBuffer commandBuffer) const
```

Definition at line 80 of file [model.cpp](#).

### 6.20.3.2 createIndexBuffer()

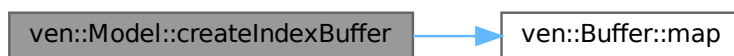
```
void ven::Model::createIndexBuffer (  
    const std::vector< uint32_t > & indices) [private]
```

Definition at line 49 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:



### 6.20.3.3 createModelFromFile()

```
std::unique_ptr< ven::Model > ven::Model::createModelFromFile (
    Device & device,
    const std::string & filename) [static]
```

Definition at line 91 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:



### 6.20.3.4 createVertexBuffer()

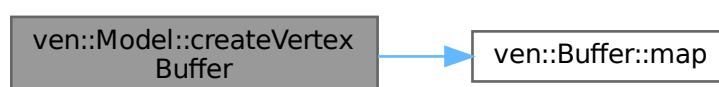
```
void ven::Model::createVertexBuffer (
    const std::vector< Vertex > & vertices) [private]
```

Definition at line 32 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:



### 6.20.3.5 draw()

```
void ven::Model::draw (
    VkCommandBuffer commandBuffer) const
```

Definition at line 71 of file [model.cpp](#).

### 6.20.3.6 operator=()

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

## 6.20.4 Member Data Documentation

### 6.20.4.1 m\_device

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

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

### 6.20.4.2 m\_hasIndexBuffer

```
bool ven::Model::m_hasIndexBuffer {false} [private]
```

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

### 6.20.4.3 m\_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

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

#### 6.20.4.4 m\_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

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

#### 6.20.4.5 m\_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

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

#### 6.20.4.6 m\_vertexCount

```
uint32_t ven::Model::m_vertexCount [private]
```

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

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

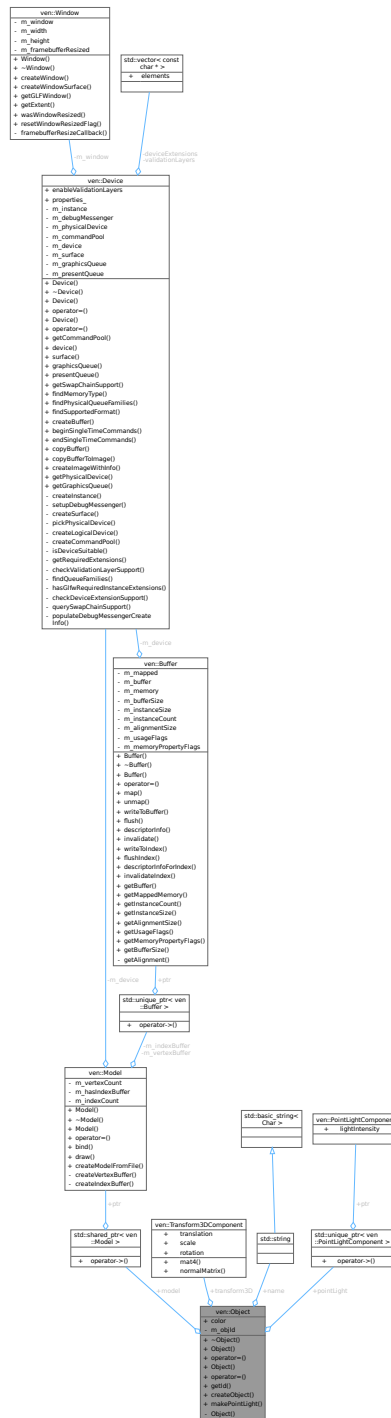
- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

## 6.21 ven::Object Class Reference

Class for object.

```
#include <Object.hpp>
```

Collaboration diagram for ven::Object:



## Public Types

- using [Map](#) = `std::unordered_map<unsigned int, Object>`

## Public Member Functions

- [~Object](#) ()=default

- [Object](#) (const [Object](#) &)=delete
- [Object](#) & [operator=](#) (const [Object](#) &)=delete
- [Object](#) ([Object](#) &&)=default
- [Object](#) & [operator=](#) ([Object](#) &&)=default
- unsigned int [getId](#) () const

### Static Public Member Functions

- static [Object](#) [createObject](#) ()
- static [Object](#) [makePointLight](#) (float intensity=[DEFAULT\\_LIGHT\\_INTENSITY](#), float radius=[DEFAULT\\_LIGHT\\_RADIUS](#), glm::vec3 [color](#)=[DEFAULT\\_LIGHT\\_COLOR](#))

### Public Attributes

- std::shared\_ptr< [Model](#) > [model](#) {}
- glm::vec3 [color](#) {}
- [Transform3DComponent](#) [transform3D](#) {}
- std::string [name](#) {""}
- std::unique\_ptr< [PointLightComponent](#) > [pointLight](#) = nullptr

### Private Member Functions

- [Object](#) (const unsigned int objId)

### Private Attributes

- unsigned int [m\\_objId](#)

## 6.21.1 Detailed Description

Class for object.

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

## 6.21.2 Member Typedef Documentation

### 6.21.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

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

## 6.21.3 Constructor & Destructor Documentation

### 6.21.3.1 ~Object()

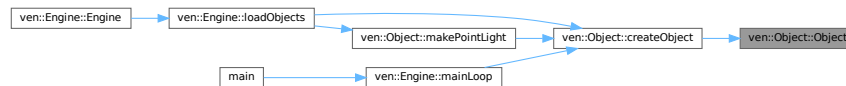
```
ven::Object::~Object () [default]
```

### 6.21.3.2 Object() [1/3]

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

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

Here is the caller graph for this function:



### 6.21.3.3 Object() [2/3]

```
ven::Object::Object (
    Object && ) [default]
```

### 6.21.3.4 Object() [3/3]

```
ven::Object::Object (
    const unsigned int objId) [inline], [explicit], [private]
```

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

## 6.21.4 Member Function Documentation

### 6.21.4.1 createObject()

```
static Object ven::Object::createObject () [inline], [static]
```

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

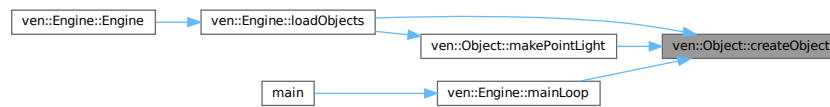
References [Object\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#), [ven::Engine::mainLoop\(\)](#), and [makePointLight\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.21.4.2 getId()

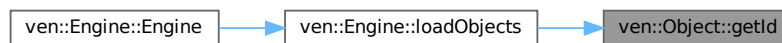
```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

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

References [m\\_objId](#).

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

Here is the caller graph for this function:



#### 6.21.4.3 makePointLight()

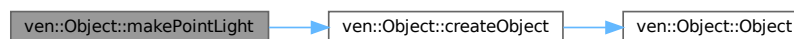
```
ven::Object ven::Object::makePointLight (
    float intensity = DEFAULT_LIGHT_INTENSITY,
    float radius = DEFAULT_LIGHT_RADIUS,
    glm::vec3 color = DEFAULT_LIGHT_COLOR) [static]
```

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

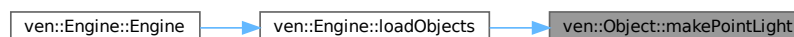
References [color](#), [createObject\(\)](#), [pointLight](#), [ven::Transform3DComponent::scale](#), and [transform3D](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:





#### 6.21.4.4 operator=() [1/2]

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

#### 6.21.4.5 operator=() [2/2]

```
Object & ven::Object::operator= (
    Object && ) [default]
```

### 6.21.5 Member Data Documentation

#### 6.21.5.1 color

```
glm::vec3 ven::Object::color {}
```

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

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

#### 6.21.5.2 m\_objId

```
unsigned int ven::Object::m_objId [private]
```

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

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

#### 6.21.5.3 model

```
std::shared_ptr<Model> ven::Object::model {}
```

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

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

#### 6.21.5.4 name

```
std::string ven::Object::name {""}
```

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

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

### 6.21.5.5 pointLight

```
std::unique_ptr<PointLightComponent> ven::Object::pointLight = nullptr
```

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

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

### 6.21.5.6 transform3D

```
Transform3DComponent ven::Object::transform3D {}
```

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

Referenced by [ven::Engine::loadObjects\(\)](#), [ven::Engine::mainLoop\(\)](#), [makePointLight\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

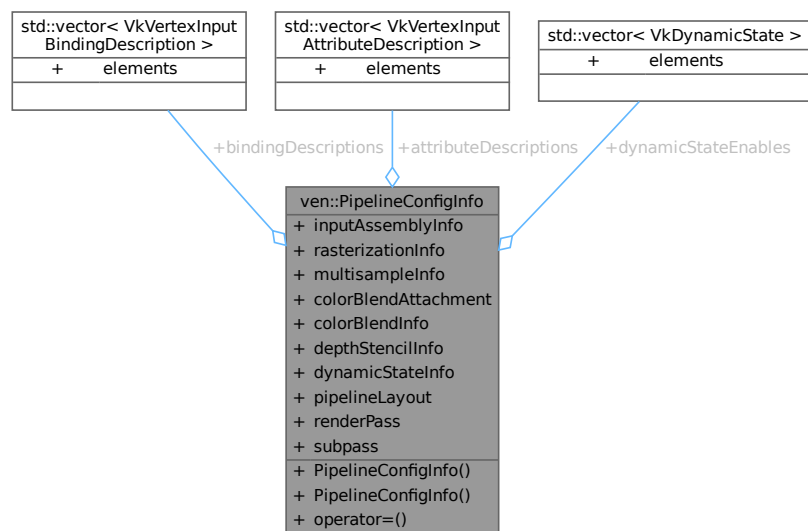
The documentation for this class was generated from the following files:

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

## 6.22 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

### 6.22.1 Detailed Description

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

### 6.22.2 Constructor & Destructor Documentation

#### 6.22.2.1 PipelineConfigInfo() [1/2]

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

#### 6.22.2.2 PipelineConfigInfo() [2/2]

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

### 6.22.3 Member Function Documentation

#### 6.22.3.1 operator=()

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

## 6.22.4 Member Data Documentation

### 6.22.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

### 6.22.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

### 6.22.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

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

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

### 6.22.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

### 6.22.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

### 6.22.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

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

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 6.22.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 6.22.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 6.22.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 6.22.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

#### 6.22.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

#### 6.22.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

### 6.22.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

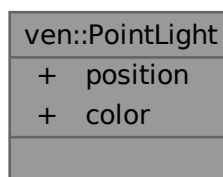
The documentation for this struct was generated from the following file:

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

## 6.23 ven::PointLight Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::PointLight:



### Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}

### 6.23.1 Detailed Description

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

### 6.23.2 Member Data Documentation

#### 6.23.2.1 color

```
glm::vec4 ven::PointLight::color {}
```

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

### 6.23.2.2 position

```
glm::vec4 ven::PointLight::position {}
```

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

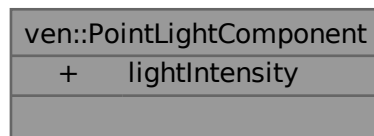
The documentation for this struct was generated from the following file:

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

## 6.24 ven::PointLightComponent Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::PointLightComponent:



### Public Attributes

- float [lightIntensity](#) = [DEFAULT\\_LIGHT\\_INTENSITY](#)

### 6.24.1 Detailed Description

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

### 6.24.2 Member Data Documentation

#### 6.24.2.1 lightIntensity

```
float ven::PointLightComponent::lightIntensity = DEFAULT\_LIGHT\_INTENSITY
```

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

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

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

## 6.25 PointLightPushConstants Struct Reference

Collaboration diagram for PointLightPushConstants:

PointLightPushConstants	
+	position
+	color
+	radius

### Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [radius](#)

### 6.25.1 Detailed Description

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

### 6.25.2 Member Data Documentation

#### 6.25.2.1 color

```
glm::vec4 PointLightPushConstants::color {}
```

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

#### 6.25.2.2 position

```
glm::vec4 PointLightPushConstants::position {}
```

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

Referenced by [ven::PointLightSystem::render\(\)](#).



### 6.25.2.3 radius

```
float PointLightPushConstants::radius
```

Definition at line 8 of file [pointLightSystem.cpp](#).

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

- [/home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp](#)

## 6.26 ven::PointLightSystem Class Reference

Class for point light system.

```
#include <PointLightSystem.hpp>
```

Collaboration diagram for `ven::PointLightSystem`:



## Public Member Functions

- [PointLightSystem](#) ([Device](#) &device, [VkRenderPass](#) renderPass, [VkDescriptorSetLayout](#) globalSetLayout)
- [~PointLightSystem](#) ()
- [PointLightSystem](#) (const [PointLightSystem](#) &)=delete
- [PointLightSystem](#) & operator= (const [PointLightSystem](#) &)=delete
- void [render](#) (const [FrameInfo](#) &frameInfo) const

## Static Public Member Functions

- static void [update](#) (const [FrameInfo](#) &frameInfo, [GlobalUbo](#) &ubo)

## Private Member Functions

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout)
- void [createPipeline](#) (VkRenderPass renderPass)

## Private Attributes

- [Device](#) & [m\\_device](#)
- std::unique\_ptr< [Shaders](#) > [m\\_shaders](#)
- VkPipelineLayout [m\\_pipelineLayout](#) {nullptr}

## 6.26.1 Detailed Description

Class for point light system.

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

## 6.26.2 Constructor & Destructor Documentation

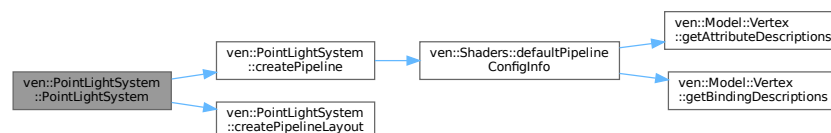
### 6.26.2.1 PointLightSystem() [1/2]

```
ven::PointLightSystem::PointLightSystem (
    Device & device,
    VkRenderPass renderPass,
    VkDescriptorSetLayout globalSetLayout) [explicit]
```

Definition at line 11 of file [pointLightSystem.cpp](#).

References [createPipeline\(\)](#), and [createPipelineLayout\(\)](#).

Here is the call graph for this function:



### 6.26.2.2 ~PointLightSystem()

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

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

References [ven::Device::device\(\)](#), [m\\_device](#), and [m\\_pipelineLayout](#).

Here is the call graph for this function:



### 6.26.2.3 PointLightSystem() [2/2]

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

## 6.26.3 Member Function Documentation

### 6.26.3.1 createPipeline()

```
void ven::PointLightSystem::createPipeline (
    VkRenderPass renderPass) [private]
```

Definition at line 38 of file [pointLightSystem.cpp](#).

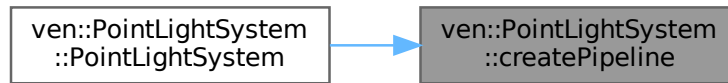
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#), and [ven::SHADERS\\_BIN\\_PATH](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



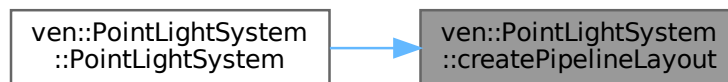
### 6.26.3.2 createPipelineLayout()

```
void ven::PointLightSystem::createPipelineLayout (
    VkDescriptorSetLayout globalSetLayout) [private]
```

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

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

Here is the caller graph for this function:



### 6.26.3.3 operator=()

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

### 6.26.3.4 render()

```
void ven::PointLightSystem::render (
    const FrameInfo & frameInfo) const
```

Definition at line 49 of file [pointLightSystem.cpp](#).

References [ven::FrameInfo::commandBuffer](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::FrameInfo::objects](#), and [PointLightPushConstants::position](#).

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

Here is the caller graph for this function:



#### 6.26.3.5 update()

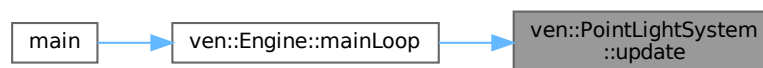
```
void ven::PointLightSystem::update (
    const FrameInfo & frameInfo,
    GlobalUbo & ubo) [static]
```

Definition at line 69 of file [pointLightSystem.cpp](#).

References [ven::FrameInfo::frameTime](#), [ven::MAX\\_LIGHTS](#), [ven::GlobalUbo::numLights](#), [ven::FrameInfo::objects](#), and [ven::GlobalUbo::pointLights](#).

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

Here is the caller graph for this function:



### 6.26.4 Member Data Documentation

#### 6.26.4.1 m\_device

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

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

Referenced by [~PointLightSystem\(\)](#).

#### 6.26.4.2 m\_pipelineLayout

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

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

Referenced by [~PointLightSystem\(\)](#).

### 6.26.4.3 m\_shaders

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

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

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

- /home/runner/work/VEngine/VEngine/include/VEngine/System/[PointLightSystem.hpp](#)
- /home/runner/work/VEngine/VEngine/src/system/[pointLightSystem.cpp](#)

## 6.27 ven::QueueFamilyIndices Struct Reference

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

Collaboration diagram for ven::QueueFamilyIndices:

ven::QueueFamilyIndices
+ graphicsFamily
+ presentFamily
+ graphicsFamilyHasValue
+ presentFamilyHasValue
+ isComplete()

### Public Member Functions

- bool [isComplete](#) () const

### Public Attributes

- uint32\_t [graphicsFamily](#) {}
- uint32\_t [presentFamily](#) {}
- bool [graphicsFamilyHasValue](#) = false
- bool [presentFamilyHasValue](#) = false

### 6.27.1 Detailed Description

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

## 6.27.2 Member Function Documentation

### 6.27.2.1 isComplete()

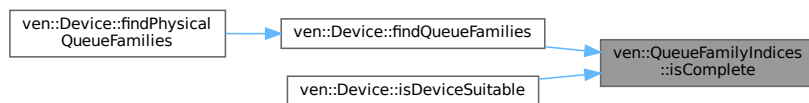
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

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:



## 6.27.3 Member Data Documentation

### 6.27.3.1 graphicsFamily

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

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

Referenced by [ven::Device::createCommandPool\(\)](#), [ven::Device::createLogicalDevice\(\)](#), [ven::SwapChain::createSwapChain\(\)](#), and [ven::Device::findQueueFamilies\(\)](#).

### 6.27.3.2 graphicsFamilyHasValue

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 24 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

### 6.27.3.3 presentFamily

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

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

Referenced by [ven::Device::createLogicalDevice\(\)](#), [ven::SwapChain::createSwapChain\(\)](#), and [ven::Device::findQueueFamilies\(\)](#).



#### 6.27.3.4 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:

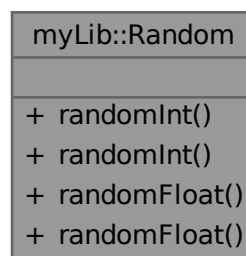
- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

## 6.28 myLib::Random Class Reference

Class for random number generation.

```
#include <Random.hpp>
```

Collaboration diagram for myLib::Random:



### Static Public Member Functions

- static int [randomInt](#) (int min, int max)  
*Generate a random integer between min and max.*
- static int [randomInt](#) ()
- static float [randomFloat](#) (float min, float max)
- static float [randomFloat](#) ()

### 6.28.1 Detailed Description

Class for random number generation.

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

## 6.28.2 Member Function Documentation

### 6.28.2.1 randomFloat() [1/2]

`static float myLib::Random::randomFloat () [inline], [static]`

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.28.2.2 randomFloat() [2/2]

```
float myLib::Random::randomFloat (
    float min,
    float max) [static]
```

#### Parameters

<i>min</i>	The minimum value
<i>max</i>	The maximum value

#### Returns

float The random float

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

References [myLib::RANDOM\\_FLOAT\\_MAX](#), [myLib::RANDOM\\_INT\\_MAX](#), and [myLib::RANDOM\\_INT\\_MIN](#).

**6.28.2.3** `randomInt()` [1/2]

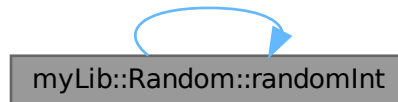
```
static int myLib::Random::randomInt () [inline], [static]
```

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

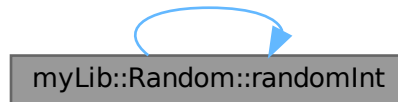
References [randomInt\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:

**6.28.2.4** `randomInt()` [2/2]

```
int myLib::Random::randomInt (
    int min,
    int max) [static]
```

Generate a random integer between min and max.

**Parameters**

<i>min</i>	The minimum value
<i>max</i>	The maximum value

**Returns**

int The random integer

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

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

- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp](#)
- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp](#)



- [~Renderer](#) ()
- [Renderer](#) (const [Renderer](#) &)=delete
- [Renderer](#) & [operator=](#) (const [Renderer](#) &)=delete
- [VkRenderPass](#) [getSwapChainRenderPass](#) () const
- [float](#) [getAspectRatio](#) () const
- [bool](#) [isFrameInProgress](#) () const
- [VkCommandBuffer](#) [getCurrentCommandBuffer](#) () const
- [int](#) [getFrameIndex](#) () const
- [std::array< float, 4 >](#) [getClearColor](#) () const
- [void](#) [setClearValue](#) ([VkClearColorValue](#) clearColorValue=[DEFAULT\\_CLEAR\\_COLOR](#), [VkClearDepthStencilValue](#) clearDepthValue=[DEFAULT\\_CLEAR\\_DEPTH](#))
- [VkCommandBuffer](#) [beginFrame](#) ()
- [void](#) [endFrame](#) ()
- [void](#) [beginSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer)
- [void](#) [endSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer)

### 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< VkClearValue, 2 >](#) [m\\_clearValues](#)
- [uint32\\_t](#) [m\\_currentImageIndex](#) {0}
- [int](#) [m\\_currentFrameIndex](#) {0}
- [bool](#) [m\\_isFrameStarted](#) {false}

## 6.29.1 Detailed Description

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

## 6.29.2 Constructor & Destructor Documentation

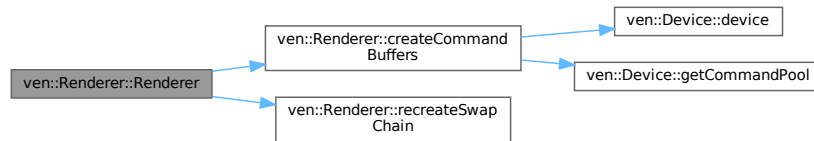
### 6.29.2.1 [Renderer\(\)](#) [1/2]

```
ven::Renderer::Renderer (
    Window & window,
    Device & device) [inline]
```

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

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



### 6.29.2.2 ~Renderer()

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

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

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

Here is the call graph for this function:



### 6.29.2.3 Renderer() [2/2]

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

## 6.29.3 Member Function Documentation

### 6.29.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

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

### 6.29.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (
    VkCommandBuffer commandBuffer)
```

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

### 6.29.3.3 createCommandBuffers()

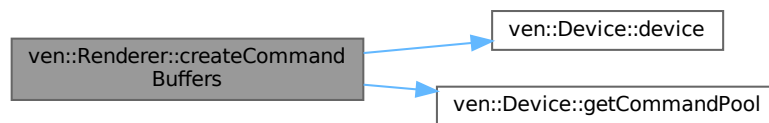
```
void ven::Renderer::createCommandBuffers () [private]
```

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

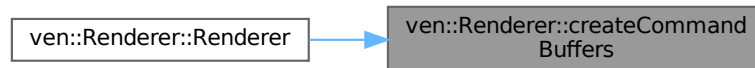
References [ven::Device::device\(\)](#), [ven::Device::getCommandPool\(\)](#), [m\\_commandBuffers](#), [m\\_device](#), and [ven::SwapChain::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.29.3.4 endFrame()

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::SwapChain::MAX\\_FRAMES\\_IN\\_FLIGHT](#).

### 6.29.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (  
    VkCommandBuffer commandBuffer)
```

Definition at line 120 of file [renderer.cpp](#).

### 6.29.3.6 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:



### 6.29.3.7 getAspectRatio()

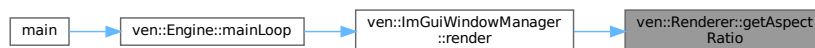
```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

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

References [m\\_swapChain](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



### 6.29.3.8 getClearColor()

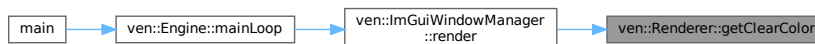
```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

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

References [m\\_clearValues](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:





### 6.29.3.9 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

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

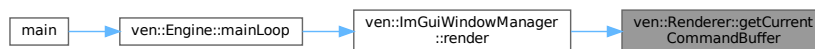
References [isFrameInProgress\(\)](#), [m\\_commandBuffers](#), and [m\\_currentFrameIndex](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



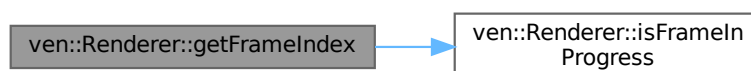
### 6.29.3.10 getFrameIndex()

```
int ven::Renderer::getFrameIndex () const [inline], [nodiscard]
```

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

References [isFrameInProgress\(\)](#), and [m\\_currentFrameIndex](#).

Here is the call graph for this function:



### 6.29.3.11 getSwapChainRenderPass()

```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

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

References [m\\_swapChain](#).

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

Here is the caller graph for this function:



### 6.29.3.12 isFrameInProgress()

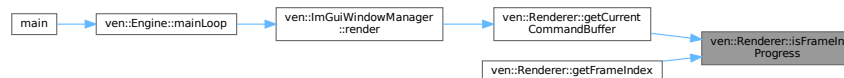
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

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

References [m\\_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



### 6.29.3.13 operator=()

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

### 6.29.3.14 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:



### 6.29.3.15 setClearValue()

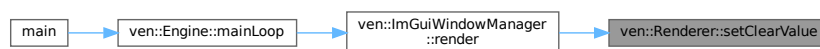
```
void ven::Renderer::setClearValue (
    VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

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

References [m\\_clearValues](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the caller graph for this function:



## 6.29.4 Member Data Documentation

### 6.29.4.1 m\_clearValues

```
std::array<VkClearValue, 2> ven::Renderer::m_clearValues [private]
```

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

Referenced by [getClearColor\(\)](#), and [setClearValue\(\)](#).

#### 6.29.4.2 m\_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

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

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

#### 6.29.4.3 m\_currentFrameIndex

```
int ven::Renderer::m_currentFrameIndex {0} [private]
```

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

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

#### 6.29.4.4 m\_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

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

#### 6.29.4.5 m\_device

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

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

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

#### 6.29.4.6 m\_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

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

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

#### 6.29.4.7 m\_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]
```

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

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

#### 6.29.4.8 m\_window

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

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

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

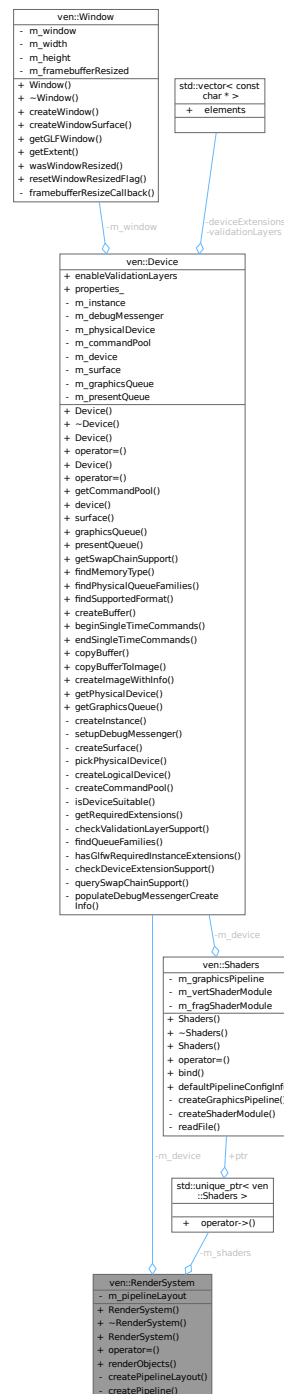
- [/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/renderer.cpp](#)

## 6.30 ven::RenderSystem Class Reference

Class for render system.

```
#include <RenderSystem.hpp>
```

Collaboration diagram for ven::RenderSystem:



## Public Member Functions

- [RenderSystem](#) ([Device](#) &device, [VkRenderPass](#) renderPass, [VkDescriptorSetLayout](#) globalSetLayout)
- [~RenderSystem](#) ()
- [RenderSystem](#) (const [RenderSystem](#) &)=delete
- [RenderSystem](#) & [operator=](#) (const [RenderSystem](#) &)=delete
- void [renderObjects](#) (const [FrameInfo](#) &frameInfo) const

## Private Member Functions

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout)
- void [createPipeline](#) (VkRenderPass renderPass)

## Private Attributes

- [Device](#) & [m\\_device](#)
- std::unique\_ptr< [Shaders](#) > [m\\_shaders](#)
- VkPipelineLayout [m\\_pipelineLayout](#) {nullptr}

### 6.30.1 Detailed Description

Class for render system.

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

### 6.30.2 Constructor & Destructor Documentation

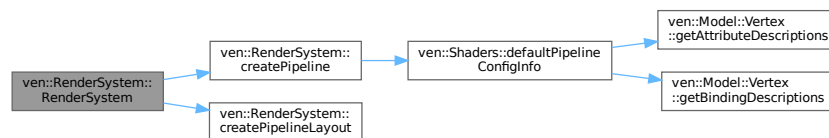
#### 6.30.2.1 RenderSystem() [1/2]

```
ven::RenderSystem::RenderSystem (
    Device & device,
    VkRenderPass renderPass,
    VkDescriptorSetLayout globalSetLayout) [explicit]
```

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

References [createPipeline\(\)](#), and [createPipelineLayout\(\)](#).

Here is the call graph for this function:



### 6.30.2.2 ~RenderSystem()

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

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

References [ven::Device::device\(\)](#), [m\\_device](#), and [m\\_pipelineLayout](#).

Here is the call graph for this function:



### 6.30.2.3 RenderSystem() [2/2]

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

## 6.30.3 Member Function Documentation

### 6.30.3.1 createPipeline()

```
void ven::RenderSystem::createPipeline (
    VkRenderPass renderPass) [private]
```

Definition at line 30 of file [renderSystem.cpp](#).

References [ven::Shaders::defaultPipelineConfigInfo\(\)](#), and [ven::SHADERS\\_BIN\\_PATH](#).

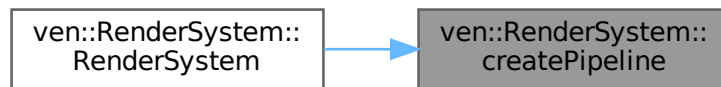
Referenced by [RenderSystem\(\)](#).

Here is the call graph for this function:





Here is the caller graph for this function:



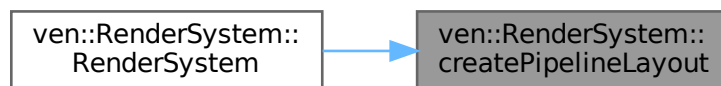
### 6.30.3.2 createPipelineLayout()

```
void ven::RenderSystem::createPipelineLayout (
    VkDescriptorSetLayout globalSetLayout) [private]
```

Definition at line 9 of file [renderSystem.cpp](#).

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

Here is the caller graph for this function:



### 6.30.3.3 operator=()

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

### 6.30.3.4 renderObjects()

```
void ven::RenderSystem::renderObjects (
    const FrameInfo & frameInfo) const
```

Definition at line 39 of file [renderSystem.cpp](#).

References [ven::FrameInfo::commandBuffer](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::SimplePushConstantData::modelMatrix](#), and [ven::FrameInfo::objects](#).

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

Here is the caller graph for this function:



## 6.30.4 Member Data Documentation

### 6.30.4.1 m\_device

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

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

Referenced by [~RenderSystem\(\)](#).

### 6.30.4.2 m\_pipelineLayout

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

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

Referenced by [~RenderSystem\(\)](#).

### 6.30.4.3 m\_shaders

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

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

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

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

## 6.31 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}

## 6.31.1 Detailed Description

Class for shaders.

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

## 6.31.2 Constructor & Destructor Documentation

### 6.31.2.1 Shaders() [1/2]

```
ven::Shaders::Shaders (
    Device & device,
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [inline]
```

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

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

Here is the call graph for this function:



### 6.31.2.2 ~Shaders()

```
ven::Shaders::~~Shaders ()
```

Definition at line 6 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:



### 6.31.2.3 Shaders() [2/2]

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

### 6.31.3 Member Function Documentation

#### 6.31.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

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

References [m\\_graphicsPipeline](#).

#### 6.31.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [private]
```

Definition at line 31 of file [shaders.cpp](#).

References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendEquation](#), [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:



#### 6.31.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

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

### 6.31.3.4 defaultPipelineConfigInfo()

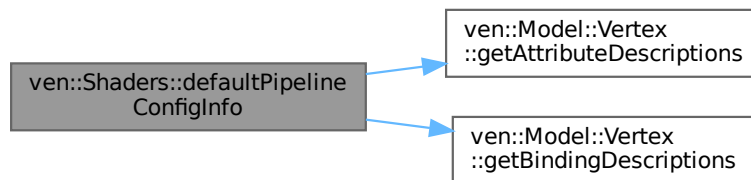
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 112 of file [shaders.cpp](#).

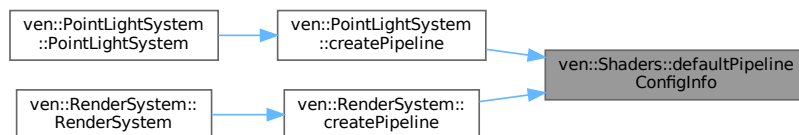
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::PointLightSystem::createPipeline\(\)](#), and [ven::RenderSystem::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.31.3.5 operator=()

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

### 6.31.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

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

### 6.31.4 Member Data Documentation

#### 6.31.4.1 m\_device

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

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

Referenced by [~Shaders\(\)](#).

#### 6.31.4.2 m\_fragShaderModule

`VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]`

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

Referenced by [~Shaders\(\)](#).

#### 6.31.4.3 m\_graphicsPipeline

`VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]`

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

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

#### 6.31.4.4 m\_vertShaderModule

`VkShaderModule ven::Shaders::m_vertShaderModule {nullptr} [private]`

Definition at line 67 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](#)

## 6.32 ven::SimplePushConstantData Struct Reference

`#include <RenderSystem.hpp>`

Collaboration diagram for `ven::SimplePushConstantData`:

ven::SimplePushConstantData	
+	modelMatrix
+	normalMatrix



## Public Attributes

- glm::mat4 [modelMatrix](#) {1.F}
- glm::mat4 [normalMatrix](#) {1.F}

### 6.32.1 Detailed Description

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

### 6.32.2 Member Data Documentation

#### 6.32.2.1 modelMatrix

```
glm::mat4 ven::SimplePushConstantData::modelMatrix {1.F}
```

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

Referenced by [ven::RenderSystem::renderObjects\(\)](#).

#### 6.32.2.2 normalMatrix

```
glm::mat4 ven::SimplePushConstantData::normalMatrix {1.F}
```

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

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

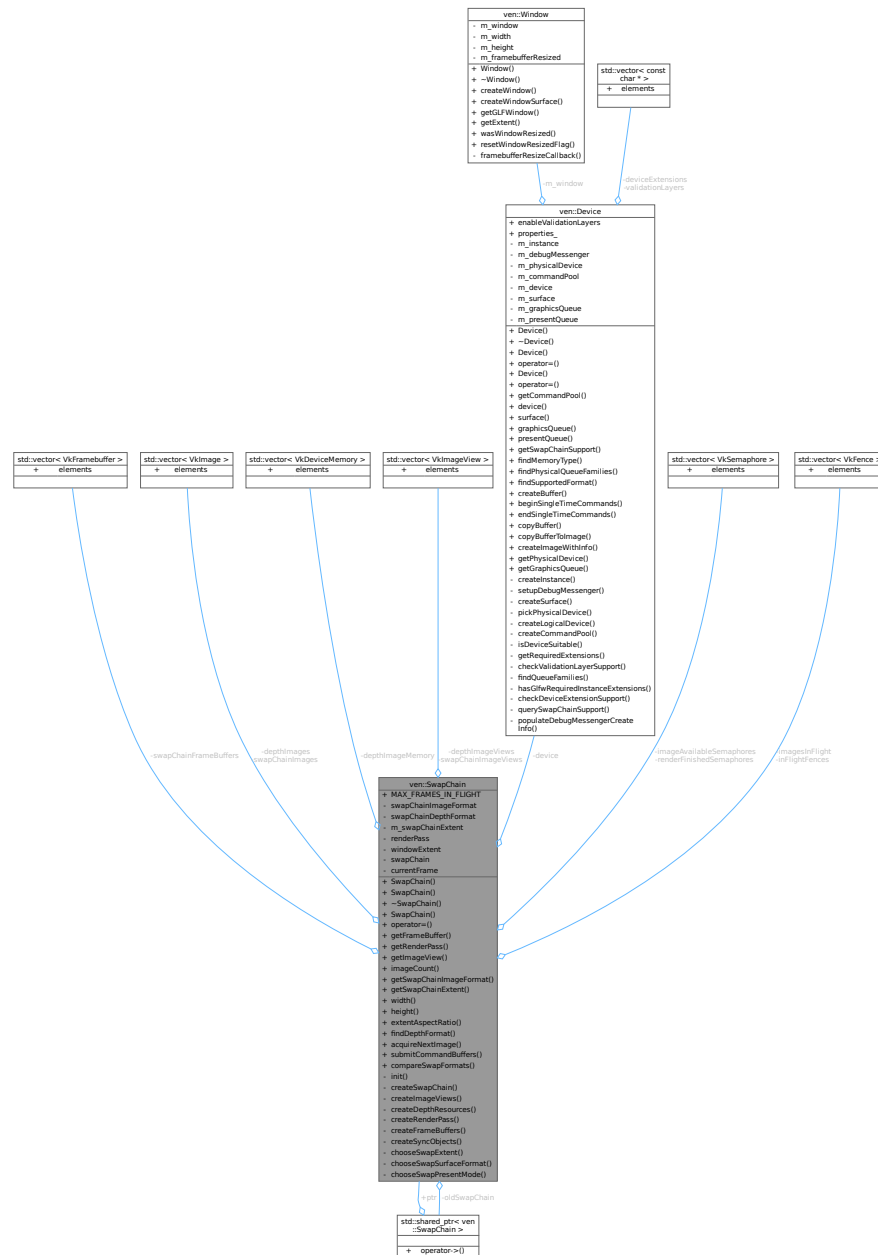
- [/home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp](#)

## 6.33 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](#) &swapChainp) const

### Static Public Attributes

- static constexpr int [MAX\\_FRAMES\\_IN\\_FLIGHT](#) = 2

### 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 [swapChainImageFormat](#) {}
- VkFormat [swapChainDepthFormat](#) {}
- VkExtent2D [m\\_swapChainExtent](#) {}
- std::vector< VkFramebuffer > [swapChainFrameBuffers](#)
- VkRenderPass [renderPass](#) {}
- std::vector< VkImage > [depthImages](#)
- std::vector< VkDeviceMemory > [depthImageMemory](#)
- std::vector< VkImageView > [depthImageViews](#)
- std::vector< VkImage > [swapChainImages](#)
- std::vector< VkImageView > [swapChainImageViews](#)
- [Device](#) & [device](#)
- VkExtent2D [windowExtent](#)
- VkSwapchainKHR [swapChain](#) {}
- std::shared\_ptr< [SwapChain](#) > [oldSwapChain](#)
- std::vector< VkSemaphore > [imageAvailableSemaphores](#)
- std::vector< VkSemaphore > [renderFinishedSemaphores](#)
- std::vector< VkFence > [inFlightFences](#)
- std::vector< VkFence > [imagesInFlight](#)
- size\_t [currentFrame](#) = 0

### 6.33.1 Detailed Description

Class for swap chain.

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

### 6.33.2 Constructor & Destructor Documentation

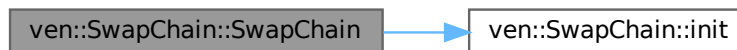
#### 6.33.2.1 SwapChain() [1/3]

```
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:



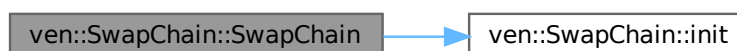
#### 6.33.2.2 SwapChain() [2/3]

```
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 [oldSwapChain](#).

Here is the call graph for this function:



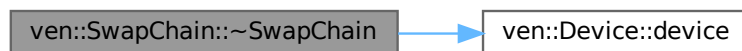
### 6.33.2.3 ~SwapChain()

```
ven::SwapChain::~~SwapChain ()
```

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

References [depthImageMemory](#), [depthImages](#), [depthImageViews](#), [ven::Device::device\(\)](#), [device](#), [imageAvailableSemaphores](#), [inFlightFences](#), [MAX\\_FRAMES\\_IN\\_FLIGHT](#), [renderFinishedSemaphores](#), [renderPass](#), [swapChain](#), [swapChainFrameBuffers](#), and [swapChainImageViews](#).

Here is the call graph for this function:



### 6.33.2.4 SwapChain() [3/3]

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

## 6.33.3 Member Function Documentation

### 6.33.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

### 6.33.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

### 6.33.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

#### 6.33.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (  
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

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

#### 6.33.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (  
    const SwapChain & swapChainp) const [inline], [nodiscard]
```

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

References [swapChainDepthFormat](#), and [swapChainImageFormat](#).

#### 6.33.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

#### 6.33.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

#### 6.33.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

#### 6.33.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

#### 6.33.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

Definition at line 103 of file [swapChain.cpp](#).

References [ven::SwapChainSupportDetails::capabilities](#), [ven::SwapChainSupportDetails::formats](#), [ven::QueueFamilyIndices::graphics](#), [ven::QueueFamilyIndices::presentFamily](#), and [ven::SwapChainSupportDetails::presentModes](#).

#### 6.33.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

#### 6.33.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

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

References [m\\_swapChainExtent](#).

#### 6.33.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

#### 6.33.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (  
    const unsigned long index) const [inline], [nodiscard]
```

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

References [swapChainFrameBuffers](#).

#### 6.33.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (  
    const int index) const [inline], [nodiscard]
```

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

References [swapChainImageViews](#).

#### 6.33.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

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

References [renderPass](#).

### 6.33.3.17 getSwapChainExtent()

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

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

References [m\\_swapChainExtent](#).

### 6.33.3.18 getSwapChainImageFormat()

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

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

References [swapChainImageFormat](#).

### 6.33.3.19 height()

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

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

References [m\\_swapChainExtent](#).

### 6.33.3.20 imageCount()

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

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

References [swapChainImages](#).

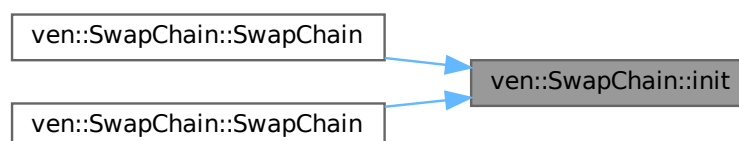
### 6.33.3.21 init()

```
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:





### 6.33.3.22 operator=()

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

### 6.33.3.23 submitCommandBuffers()

```
VkResult ven::SwapChain::submitCommandBuffers (
    const VkCommandBuffer * buffers,
    const uint32_t * imageIndex)
```

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

### 6.33.3.24 width()

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

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

References [m\\_swapChainExtent](#).

## 6.33.4 Member Data Documentation

### 6.33.4.1 currentFrame

```
size_t ven::SwapChain::currentFrame = 0 [private]
```

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

### 6.33.4.2 depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::depthImageMemory [private]
```

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

Referenced by [~SwapChain\(\)](#).

### 6.33.4.3 depthImages

```
std::vector<VkImage> ven::SwapChain::depthImages [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.4 depthImageViews

```
std::vector<VkImageView> ven::SwapChain::depthImageViews [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.5 device

```
Device& ven::SwapChain::device [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.6 imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::imageAvailableSemaphores [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.7 imagesInFlight

```
std::vector<VkFence> ven::SwapChain::imagesInFlight [private]
```

Definition at line 89 of file [SwapChain.hpp](#).

#### 6.33.4.8 inFlightFences

```
std::vector<VkFence> ven::SwapChain::inFlightFences [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.9 m\_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

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

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

#### 6.33.4.10 MAX\_FRAMES\_IN\_FLIGHT

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

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

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

#### 6.33.4.11 oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::oldSwapChain [private]
```

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

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

#### 6.33.4.12 renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::renderFinishedSemaphores [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.13 renderPass

```
VkRenderPass ven::SwapChain::renderPass {} [private]
```

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

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

#### 6.33.4.14 swapChain

```
VkSwapchainKHR ven::SwapChain::swapChain {} [private]
```

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

Referenced by [~SwapChain\(\)](#).

#### 6.33.4.15 swapChainDepthFormat

```
VkFormat ven::SwapChain::swapChainDepthFormat {} [private]
```

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

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

#### 6.33.4.16 swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::swapChainFrameBuffers [private]
```

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

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

#### 6.33.4.17 swapChainImageFormat

```
VkFormat ven::SwapChain::swapChainImageFormat {} [private]
```

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

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

#### 6.33.4.18 swapChainImages

```
std::vector<VkImage> ven::SwapChain::swapChainImages [private]
```

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

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

#### 6.33.4.19 swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::swapChainImageViews [private]
```

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

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

#### 6.33.4.20 windowExtent

```
VkExtent2D ven::SwapChain::windowExtent [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

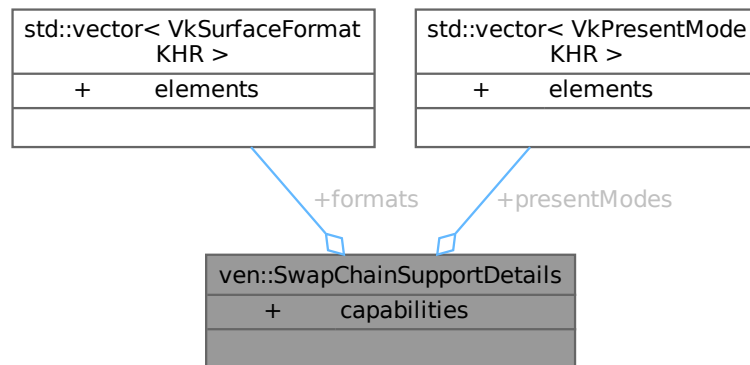
The documentation for this class was generated from the following files:

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

## 6.34 ven::SwapChainSupportDetails Struct Reference

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

Collaboration diagram for ven::SwapChainSupportDetails:



### Public Attributes

- VkSurfaceCapabilitiesKHR [capabilities](#)
- std::vector< VkSurfaceFormatKHR > [formats](#)
- std::vector< VkPresentModeKHR > [presentModes](#)

### 6.34.1 Detailed Description

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

### 6.34.2 Member Data Documentation

#### 6.34.2.1 capabilities

```
VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities
```

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

Referenced by [ven::SwapChain::createSwapChain\(\)](#), and [ven::Device::querySwapChainSupport\(\)](#).

#### 6.34.2.2 formats

```
std::vector<VkSurfaceFormatKHR> ven::SwapChainSupportDetails::formats
```

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

Referenced by [ven::SwapChain::createSwapChain\(\)](#), [ven::Device::isDeviceSuitable\(\)](#), and [ven::Device::querySwapChainSupport\(\)](#).

### 6.34.2.3 presentModes

```
std::vector<VkPresentModeKHR> ven::SwapChainSupportDetails::presentModes
```

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

Referenced by [ven::SwapChain::createSwapChain\(\)](#), [ven::Device::isDeviceSuitable\(\)](#), and [ven::Device::querySwapChainSupport\(\)](#).

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

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

## 6.35 myLib::Time Class Reference

Class used for time management.

```
#include <Time.hpp>
```

Collaboration diagram for myLib::Time:

myLib::Time
- m_seconds
+ Time()
+ asSeconds()
+ asMilliseconds()
+ asMicroseconds()

### Public Member Functions

- [Time](#) (const double seconds)  
*Construct a new [Time](#) object.*
- int [asSeconds](#) () const  
*Transform the time to seconds.*
- int [asMilliseconds](#) () const  
*Transform the time to milliseconds.*
- int [asMicroseconds](#) () const  
*Transform the time to microseconds.*

### Private Attributes

- double [m\\_seconds](#) {0.0F}

### 6.35.1 Detailed Description

Class used for time management.

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

### 6.35.2 Constructor & Destructor Documentation

#### 6.35.2.1 Time()

```
myLib::Time::Time (  
    const double seconds)    [inline], [explicit]
```

Construct a new [Time](#) object.

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

### 6.35.3 Member Function Documentation

#### 6.35.3.1 asMicroseconds()

```
int myLib::Time::asMicroseconds () const    [inline], [nodiscard]
```

Transform the time to microseconds.

##### Returns

int The time in microseconds

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

References [m\\_seconds](#), and [myLib::MICROSECONDS\\_PER\\_SECOND](#).

#### 6.35.3.2 asMilliseconds()

```
int myLib::Time::asMilliseconds () const    [inline], [nodiscard]
```

Transform the time to milliseconds.

##### Returns

int The time in milliseconds

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

References [m\\_seconds](#), and [myLib::MILLISECONDS\\_PER\\_SECOND](#).

### 6.35.3.3 asSeconds()

```
int myLib::Time::asSeconds () const [inline], [nodiscard]
```

Transform the time to seconds.

#### Returns

int The time in seconds

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

References [m\\_seconds](#).

## 6.35.4 Member Data Documentation

### 6.35.4.1 m\_seconds

```
double myLib::Time::m_seconds {0.0F} [private]
```

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

Referenced by [asMicroseconds\(\)](#), [asMilliseconds\(\)](#), and [asSeconds\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp](#)

## 6.36 ven::Transform3DComponent Struct Reference

```
#include <Object.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](#) {}

### 6.36.1 Detailed Description

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

### 6.36.2 Member Function Documentation

#### 6.36.2.1 mat4()

```
glm::mat4 ven::Transform3DComponent::mat4 () const [nodiscard]
```

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

References [rotation](#), [scale](#), and [translation](#).

#### 6.36.2.2 normalMatrix()

```
glm::mat3 ven::Transform3DComponent::normalMatrix () const [nodiscard]
```

Definition at line 38 of file [object.cpp](#).

### 6.36.3 Member Data Documentation

#### 6.36.3.1 rotation

```
glm::vec3 ven::Transform3DComponent::rotation {}
```

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

Referenced by [ven::Engine::mainLoop\(\)](#), [mat4\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

#### 6.36.3.2 scale

```
glm::vec3 ven::Transform3DComponent::scale {1.F, 1.F, 1.F}
```

Definition at line 24 of file [Object.hpp](#).

Referenced by [ven::Engine::loadObjects\(\)](#), [ven::Object::makePointLight\(\)](#), and [mat4\(\)](#).

### 6.36.3.3 translation

```
glm::vec3 ven::Transform3DComponent::translation {}
```

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

Referenced by [ven::Engine::loadObjects\(\)](#), [ven::Engine::mainLoop\(\)](#), [mat4\(\)](#), and [ven::ImGuiWindowManager::render\(\)](#).

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

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

## 6.37 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](#) {}

### 6.37.1 Detailed Description

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

### 6.37.2 Member Function Documentation

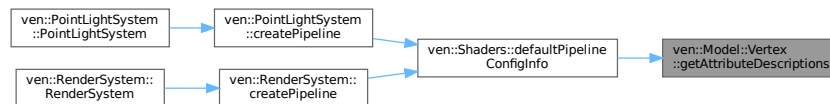
#### 6.37.2.1 getAttributeDescriptions()

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions
() [static]
```

Definition at line 107 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



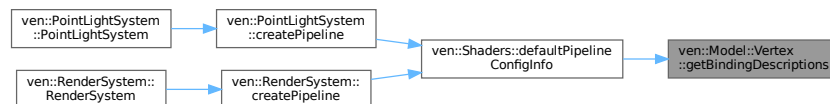
#### 6.37.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()
[static]
```

Definition at line 98 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



#### 6.37.2.3 operator==( )

```
bool ven::Model::Vertex::operator== (
    const Vertex & other) const [inline]
```

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

References [color](#), [normal](#), [position](#), and [uv](#).

### 6.37.3 Member Data Documentation

#### 6.37.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

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

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

#### 6.37.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

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

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

#### 6.37.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

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

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

#### 6.37.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

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

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

## 6.38 ven::Window Class Reference

Class for window.

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

Collaboration diagram for ven::Window:

ven::Window
<ul style="list-style-type: none"> <li>- m_window</li> <li>- m_width</li> <li>- m_height</li> <li>- m_framebufferResized</li> </ul>
<ul style="list-style-type: none"> <li>+ Window()</li> <li>+ ~Window()</li> <li>+ createWindow()</li> <li>+ createWindowSurface()</li> <li>+ getGLFWWindow()</li> <li>+ getExtent()</li> <li>+ wasWindowResized()</li> <li>+ resetWindowResizedFlag()</li> <li>- framebufferResizeCallback()</li> </ul>

### Public Member Functions

- [Window](#) (const uint32\_t width, const uint32\_t height, const std::string &title)
- [~Window](#) ()
- 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](#) ()

### Static Private Member Functions

- static void [framebufferResizeCallback](#) (GLFWwindow \*window, int width, int height)

### Private Attributes

- GLFWwindow \* [m\\_window](#) {nullptr}
- uint32\_t [m\\_width](#)
- uint32\_t [m\\_height](#)
- bool [m\\_framebufferResized](#) = false

### 6.38.1 Detailed Description

Class for window.

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

### 6.38.2 Constructor & Destructor Documentation

#### 6.38.2.1 Window()

```
ven::Window::Window (
    const uint32_t width,
    const uint32_t height,
    const std::string & title) [inline]
```

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

#### 6.38.2.2 ~Window()

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

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

References [m\\_window](#).

### 6.38.3 Member Function Documentation

#### 6.38.3.1 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:



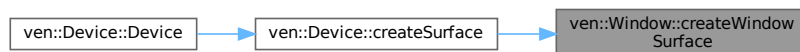
### 6.38.3.2 createWindowSurface()

```
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:



### 6.38.3.3 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:



### 6.38.3.4 getExtent()

```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

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

References [m\\_height](#), and [m\\_width](#).

### 6.38.3.5 getGLFWWindow()

```
GLFWwindow * ven::Window::getGLFWWindow () const [inline], [nodiscard]
```

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

References [m\\_window](#).

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

Here is the caller graph for this function:



### 6.38.3.6 resetWindowResizedFlag()

```
void ven::Window::resetWindowResizedFlag () [inline]
```

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

References [m\\_framebufferResized](#).

### 6.38.3.7 wasWindowResized()

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

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

References [m\\_framebufferResized](#).

## 6.38.4 Member Data Documentation

### 6.38.4.1 m\_framebufferResized

```
bool ven::Window::m_framebufferResized = false [private]
```

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

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).



#### 6.38.4.2 m\_height

```
uint32_t ven::Window::m_height [private]
```

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

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

#### 6.38.4.3 m\_width

```
uint32_t ven::Window::m_width [private]
```

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

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

#### 6.38.4.4 m\_window

```
GLFWwindow* ven::Window::m_window {nullptr} [private]
```

Definition at line 46 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 7

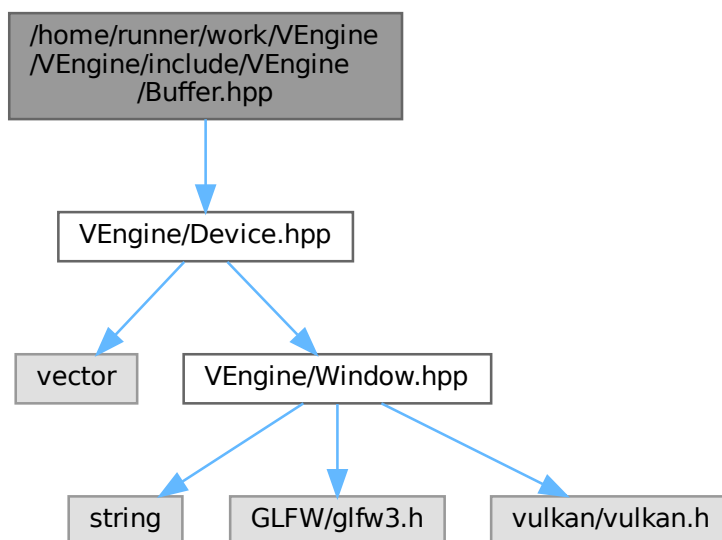
# File Documentation

### 7.1 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

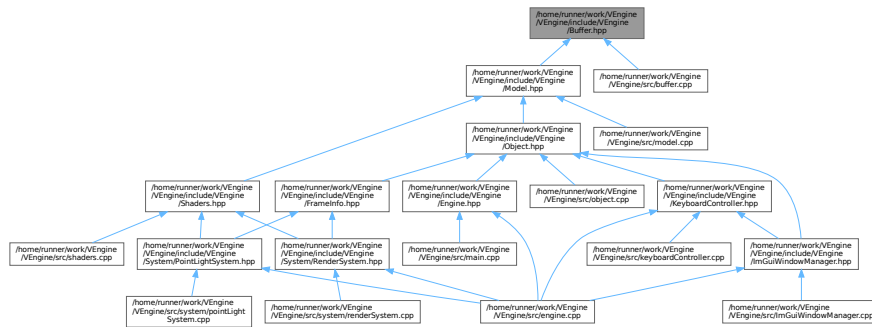
This file contains the Buffer class.

```
#include "VEngine/Device.hpp"
```

Include dependency graph for Buffer.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Buffer](#)  
*Class for buffer.*

## Namespaces

- namespace [ven](#)

### 7.1.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

## 7.2 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         Buffer(const Buffer&) = delete;
00026         Buffer& operator=(const Buffer&) = delete;
00027
00028         ///
00029         /// @brief Map a memory range of this buffer. If successful, mapped points to the
specified buffer range.
00030         ///
00031         /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the
complete buffer range.
00032         /// @param offset (Optional) Byte offset from beginning
00033         ///
00034         /// @return VkResult of the buffer mapping call
00035         ///
00036         VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00037
00038         ///
00039         /// @brief Unmap a mapped memory range
00040         ///
00041         /// @note Does not return a result as vkUnmapMemory can't fail
00042         ///
00043         void unmap();
00044
00045         ///
00046         /// @brief Copies the specified data to the mapped buffer. Default value writes whole
buffer range
00047         ///
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
complete buffer range.
00050         /// @param offset (Optional) Byte offset from beginning of mapped region
00051         ///
00052         void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
offset = 0) const;
00053
00054         ///
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         void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
m_instanceSize, index * m_alignmentSize); }
00096
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_instanceSize; }
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

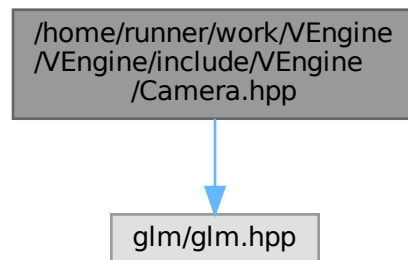
```

## 7.3 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

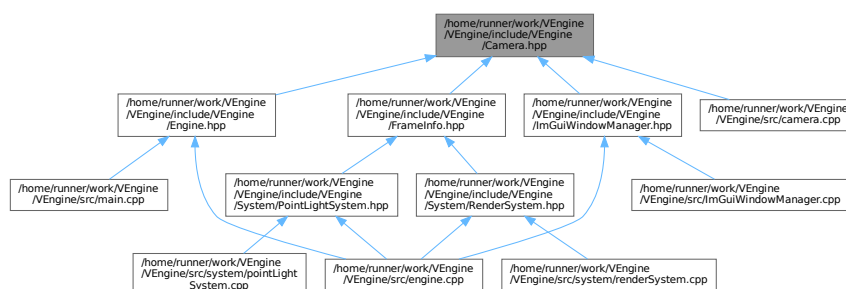
This file contains the Camera class.

```
#include <glm/glm.hpp>
```

Include dependency graph for Camera.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Camera](#)  
*Class for camera.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr glm::vec3 [ven::DEFAULT\\_POSITION](#) {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 [ven::DEFAULT\\_ROTATION](#) {0.F, 0.F, 0.F}
- static constexpr float [ven::DEFAULT\\_FOV](#) = glm::radians(50.0F)
- static constexpr float [ven::DEFAULT\\_NEAR](#) = 0.1F
- static constexpr float [ven::DEFAULT\\_FAR](#) = 100.F

### 7.3.1 Detailed Description

This file contains the Camera class.

This file contains the KeyboardController class.

Definition in file [Camera.hpp](#).

## 7.4 Camera.hpp

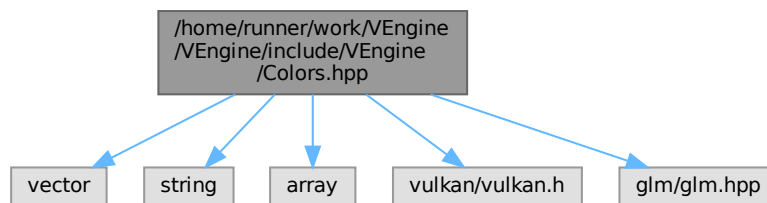
[Go to the documentation of this file.](#)

```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <glm/glm.hpp>  
00010 ///  
00011 namespace ven {  
00012     ///  
00013     static constexpr glm::vec3 DEFAULT_POSITION{0.F, 0.F, -2.5F};  
00014     static constexpr glm::vec3 DEFAULT_ROTATION{0.F, 0.F, 0.F};  
00015     ///  
00016     static constexpr float DEFAULT_FOV = glm::radians(50.0F);  
00017     static constexpr float DEFAULT_NEAR = 0.1F;  
00018     static constexpr float DEFAULT_FAR = 100.F;  
00019     ///  
00020     ///  
00021     ///  
00022     ///  
00023     ///  
00024     ///  
00025     class Camera {  
00026     public:  
00027         ///  
00028         ///  
00029         void setOrthographicProjection(float left, float right, float top, float bottom, float  
near, float far);  
00030         void setPerspectiveProjection(float aspect);  
00031         void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up =  
glm::vec3{0.F, -1.F, 0.F});  
00032         void setViewTarget(glm::vec3 position, glm::vec3 target, glm::vec3 up = glm::vec3{0.F,  
-1.F, 0.F}) { setViewDirection(position, target - position, up); }  
00033         void setViewYXZ(glm::vec3 position, glm::vec3 rotation);  
00034         void setFov(float fov) { m_fov = fov; }  
00035         void setNear(float near) { m_near = near; }  
00036         void setFar(float far) { m_far = far; }  
00037         ///  
00038         [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }  
00039         [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }  
00040         [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }  
00041         [[nodiscard]] float getFov() const { return m_fov; }  
00042         [[nodiscard]] float getNear() const { return m_near; }  
00043         [[nodiscard]] float getFar() const { return m_far; }  
00044     private:  
00045         ///  
00046         float m_fov{DEFAULT_FOV};  
00047         float m_near{DEFAULT_NEAR};  
00048         float m_far{DEFAULT_FAR};  
00049         glm::mat4 m_projectionMatrix{1.F};  
00050         glm::mat4 m_viewMatrix{1.F};  
00051         glm::mat4 m_inverseViewMatrix{1.F};  
00052     };  
00053     ///  
00054 }; // class Camera  
00055 ///  
00056 } // namespace ven
```

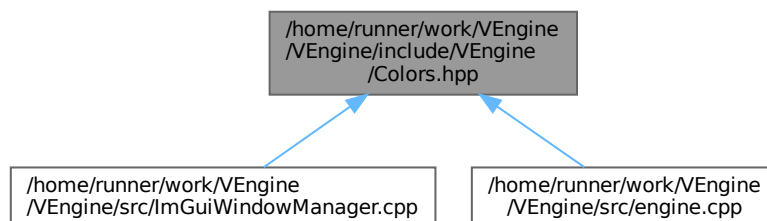


## 7.5 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference

```
#include <vector>
#include <string>
#include <array>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
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`

## 7.6 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 <vector>
00010 #include <string>
00011 #include <array>
00012
00013 #include <vulkan/vulkan.h>
00014
00015 #include <glm/glm.hpp>
00016
00017 namespace ven {
00018
00019     ///
00020     /// @class Colors
00021     /// @brief Class for colors
00022     /// @namespace ven
00023     ///
00024     class Colors {
00025
00026         static constexpr float COLOR_MAX = 255.0F;
00027
00028     public:
00029
00030         static constexpr glm::vec3 WHITE = glm::vec3(COLOR_MAX, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00031         static constexpr glm::vec3 BLACK = glm::vec3(0.0F);
00032         static constexpr glm::vec3 RED = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
00033         static constexpr glm::vec3 GREEN = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00034         static constexpr glm::vec3 BLUE = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
00035         static constexpr glm::vec3 YELLOW = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
00036         static constexpr glm::vec3 CYAN = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00037         static constexpr glm::vec3 MAGENTA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00038         static constexpr glm::vec3 SILVER = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
00039         static constexpr glm::vec3 GRAY = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
00040         static constexpr glm::vec3 MAROON = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
00041         static constexpr glm::vec3 OLIVE = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
00042         static constexpr glm::vec3 LIME = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00043         static constexpr glm::vec3 AQUA = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00044         static constexpr glm::vec3 TEAL = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
00045         static constexpr glm::vec3 NAVY = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
00046         static constexpr glm::vec3 FUCHSIA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00047         static constexpr glm::vec3 NIGHT_BLUE = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
00048         static constexpr glm::vec3 SKY_BLUE = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
00049         static constexpr glm::vec3 SUNSET = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
00050
00051         static constexpr VkClearColorValue WHITE_V = {{1.0F, 1.0F, 1.0F, 1.0F}};
00052         static constexpr VkClearColorValue BLACK_V = {{0.0F, 0.0F, 0.0F, 1.0F}};
00053         static constexpr VkClearColorValue RED_V = {{1.0F, 0.0F, 0.0F, 1.0F}};
00054         static constexpr VkClearColorValue GREEN_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
00055         static constexpr VkClearColorValue BLUE_V = {{0.0F, 0.0F, 1.0F, 1.0F}};
00056         static constexpr VkClearColorValue YELLOW_V = {{1.0F, 1.0F, 0.0F, 1.0F}};
00057         static constexpr VkClearColorValue CYAN_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
00058         static constexpr VkClearColorValue MAGENTA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
00059         static constexpr VkClearColorValue SILVER_V = {{0.75F, 0.75F, 0.75F, 1.0F}};
00060         static constexpr VkClearColorValue GRAY_V = {{0.5F, 0.5F, 0.5F, 1.0F}};
00061         static constexpr VkClearColorValue MAROON_V = {{0.5F, 0.0F, 0.0F, 1.0F}};
00062         static constexpr VkClearColorValue OLIVE_V = {{0.5F, 0.5F, 0.0F, 1.0F}};
00063         static constexpr VkClearColorValue LIME_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
00064         static constexpr VkClearColorValue AQUA_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
00065         static constexpr VkClearColorValue TEAL_V = {{0.0F, 0.5F, 0.5F, 1.0F}};
00066         static constexpr VkClearColorValue NAVY_V = {{0.0F, 0.0F, 0.5F, 1.0F}};
00067         static constexpr VkClearColorValue FUCHSIA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
00068         static constexpr VkClearColorValue NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}};
00069         static constexpr VkClearColorValue SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}};
00070         static constexpr VkClearColorValue SUNSET_V = {{1.0F, 0.5F, 0.0F, 1.0F}};
00071         static constexpr VkClearColorValue NIGHT_MODE_V = {{0.0F, 0.0F, 0.0F, 1.0F}};
00072
00073         static constexpr std::array<std::pair<const char*, glm::vec3>, 20> COLORS = {{
00074             {"White", Colors::WHITE},
00075             {"Black", Colors::BLACK},
00076             {"Red", Colors::RED},
00077             {"Green", Colors::GREEN},
00078             {"Blue", Colors::BLUE},
00079             {"Yellow", Colors::YELLOW},
00080             {"Cyan", Colors::CYAN},
00081             {"Magenta", Colors::MAGENTA},
00082             {"Silver", Colors::SILVER},
```

```

00083         {"Gray", Colors::GRAY},
00084         {"Maroon", Colors::MAROON},
00085         {"Olive", Colors::OLIVE},
00086         {"Lime", Colors::LIME},
00087         {"Aqua", Colors::AQUA},
00088         {"Teal", Colors::TEAL},
00089         {"Navy", Colors::NAVY},
00090         {"Fuchsia", Colors::FUCHSIA},
00091         {"Night Blue", ven::Colors::NIGHT_BLUE},
00092         {"Sky Blue", Colors::SKY_BLUE},
00093         {"Sunset", Colors::SUNSET}
00094     };
00095
00096     static constexpr std::array<std::pair<const char*, VkClearColorValue>, 20> CLEAR_COLORS = {{
00097         {"White", Colors::WHITE_V},
00098         {"Black", Colors::BLACK_V},
00099         {"Red", Colors::RED_V},
00100         {"Green", Colors::GREEN_V},
00101         {"Blue", Colors::BLUE_V},
00102         {"Yellow", Colors::YELLOW_V},
00103         {"Cyan", Colors::CYAN_V},
00104         {"Magenta", Colors::MAGENTA_V},
00105         {"Silver", Colors::SILVER_V},
00106         {"Gray", Colors::GRAY_V},
00107         {"Maroon", Colors::MAROON_V},
00108         {"Olive", Colors::OLIVE_V},
00109         {"Lime", Colors::LIME_V},
00110         {"Aqua", Colors::AQUA_V},
00111         {"Teal", Colors::TEAL_V},
00112         {"Navy", Colors::NAVY_V},
00113         {"Fuchsia", Colors::FUCHSIA_V},
00114         {"Night Blue", Colors::NIGHT_BLUE_V},
00115         {"Sky Blue", Colors::SKY_BLUE_V},
00116         {"Sunset", Colors::SUNSET_V}
00117     }};
00118
00119     }; // class Colors
00120
00121 } // namespace ven

```

## 7.7 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference

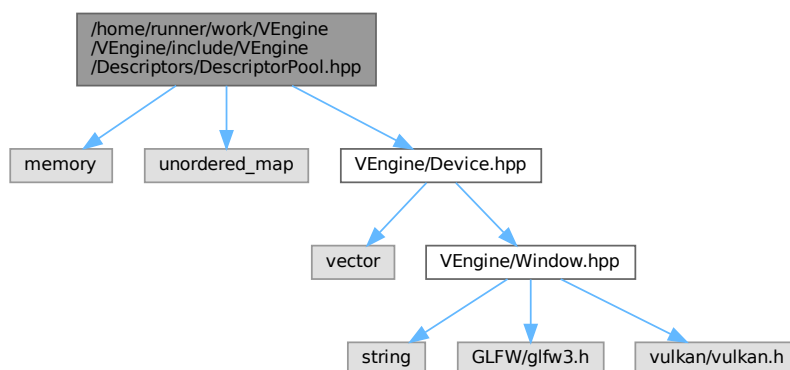
This file contains the DescriptorPool class.

```

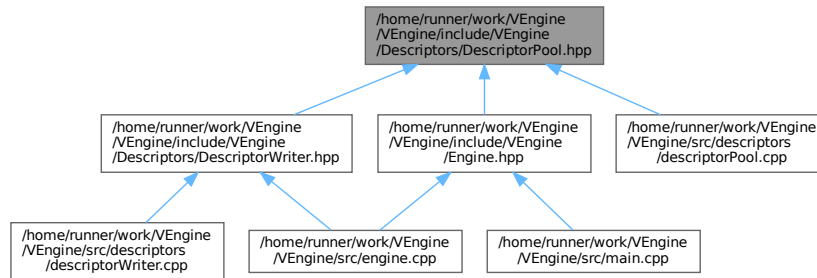
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"

```

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](#)

### 7.7.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file [DescriptorPool.hpp](#).

## 7.8 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 #include <unordered_map>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     ///
00017     /// @class DescriptorPool
00018     /// @brief Class for descriptor pool
00019     /// @namespace ven
00020     ///
00021     class DescriptorPool {
00022
00023     public:
00024
  
```

```

00025     class Builder {
00026     public:
00027         explicit Builder(Device &device) : m_device{device} {}
00028
00029         Builder &addPoolSize(VkDescriptorType descriptorType, uint32_t count);
00030         Builder &setPoolFlags(VkDescriptorPoolCreateFlags flags);
00031         Builder &setMaxSets(uint32_t count);
00032         [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
00033             std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00034
00035     private:
00036         Device &m_device;
00037         std::vector<VkDescriptorPoolSize> m_poolSizes;
00038         uint32_t m_maxSets = 1000;
00039         VkDescriptorPoolCreateFlags m_poolFlags = 0;
00040
00041     }; // class Builder
00042
00043     DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
00044         const std::vector<VkDescriptorPoolSize> &poolSizes);
00045     ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr); }
00046
00047     DescriptorPool(const DescriptorPool &) = delete;
00048     DescriptorPool &operator=(const DescriptorPool &) = delete;
00049
00050     bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
00051         &descriptor) const;
00052     void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
00053         vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
00054             descriptors.data()); }
00055     void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00056     [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }
00057
00058     private:
00059         Device &m_device;
00060         VkDescriptorPool m_descriptorPool;
00061         friend class DescriptorWriter;
00062     }; // class DescriptorPool
00063
00064 } // namespace ven

```

## 7.9 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference

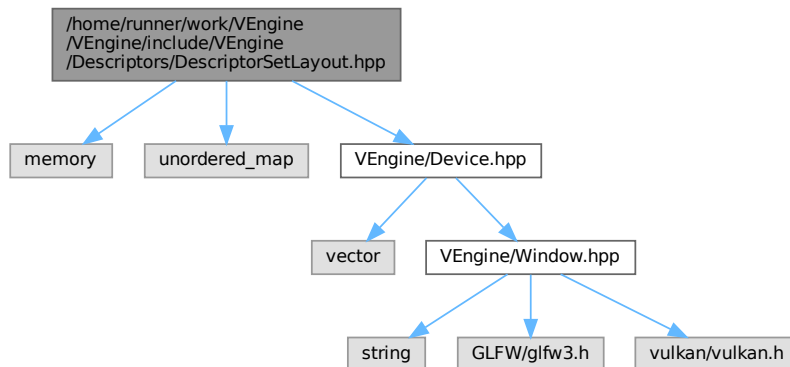
This file contains the DescriptorSetLayout class.

```

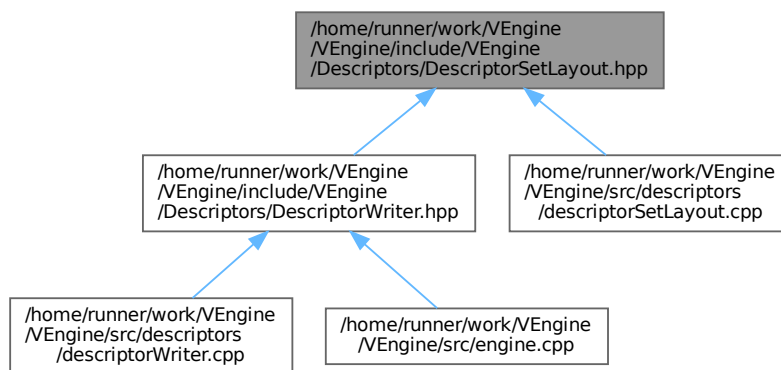
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"

```

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](#)

### 7.9.1 Detailed Description

This file contains the `DescriptorSetLayout` class.

Definition in file [DescriptorSetLayout.hpp](#).

## 7.10 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
00023     public:
00024
00025         class Builder {
00026
00027         public:
00028
00029             explicit Builder(Device &device) : m_device{device} {}
00030
00031             Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType,
00032                               VkShaderStageFlags stageFlags, uint32_t count = 1);
00033             std::unique_ptr<DescriptorSetLayout> build() const { return
00034             std::make_unique<DescriptorSetLayout>(m_device, m_bindings); }
00035
00036         private:
00037             Device &m_device;
00038             std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00039
00040         }; // class Builder
00041
00042         DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
00043                               VkDescriptorSetLayoutBinding>& bindings);
00044         ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(),
00045                               m_descriptorSetLayout, nullptr); }
00046         DescriptorSetLayout(const DescriptorSetLayout &) = delete;
00047         DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;
00048
00049         VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }
00050
00051     private:
00052         Device &m_device;
00053         VkDescriptorSetLayout m_descriptorSetLayout;
00054         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00055
00056         friend class DescriptorWriter;
00057
00058     }; // class DescriptorSetLayout
00059 } // namespace ven

```

## 7.11 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference

This file contains the DescriptorsWriter class.

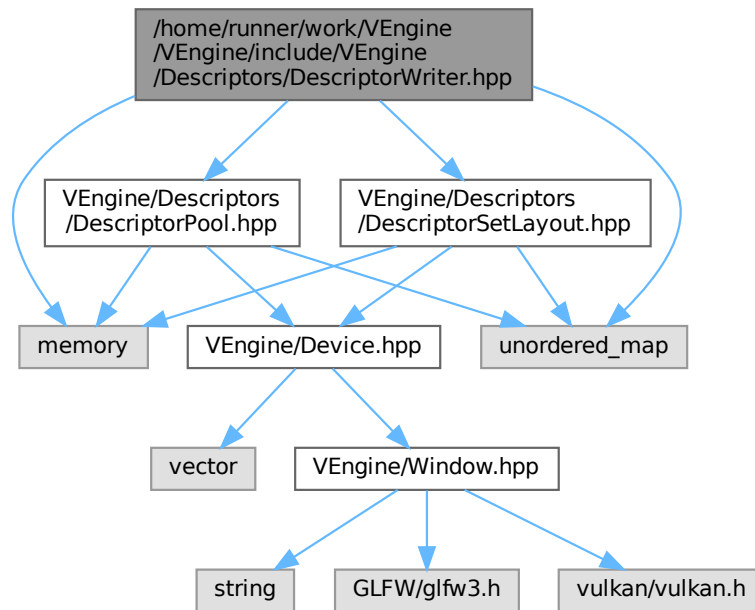
```

#include <memory>
#include <unordered_map>
#include "VEngine/Descriptors/DescriptorPool.hpp"

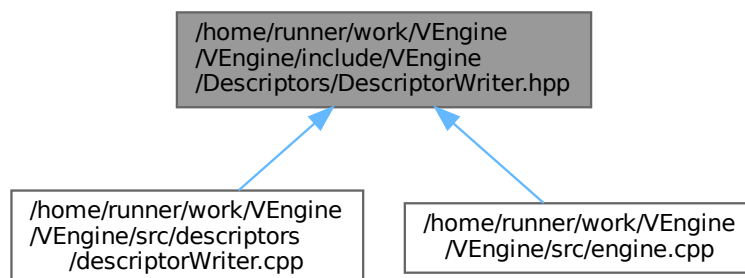
```

```
#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](#)



### 7.11.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file [DescriptorWriter.hpp](#).

## 7.12 DescriptorWriter.hpp

[Go to the documentation of this file.](#)

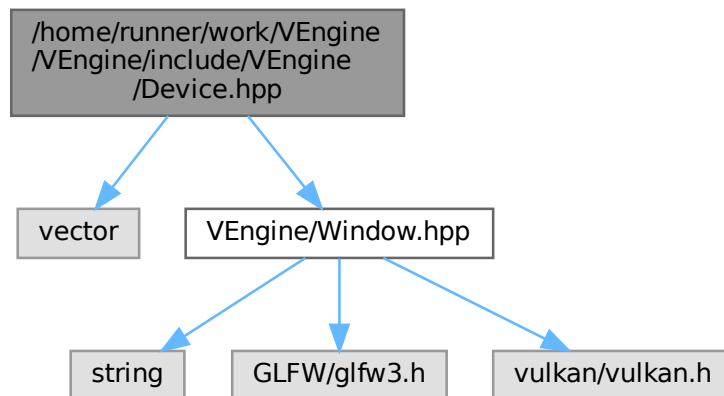
```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <memory>  
00010 #include <unordered_map>  
00011 ///  
00012 #include "VEngine/Descriptors/DescriptorPool.hpp"  
00013 #include "VEngine/Descriptors/DescriptorSetLayout.hpp"  
00014 ///  
00015 namespace ven {  
00016 ///  
00017     ///  
00018     ///  
00019     ///  
00020     ///  
00021     ///  
00022     class DescriptorWriter {  
00023     public:  
00024     public:  
00025     public:  
00026         DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :  
00027             m_setLayout(setLayout), m_pool{pool} {}  
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     private:  
00034     private:  
00035         DescriptorSetLayout &m_setLayout;  
00036         DescriptorPool &m_pool;  
00037         std::vector<VkWriteDescriptorSet> m_writes;  
00038     }; // class DescriptorWriter  
00039     };  
00040     };  
00041     };  
00042 } // namespace ven
```

## 7.13 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

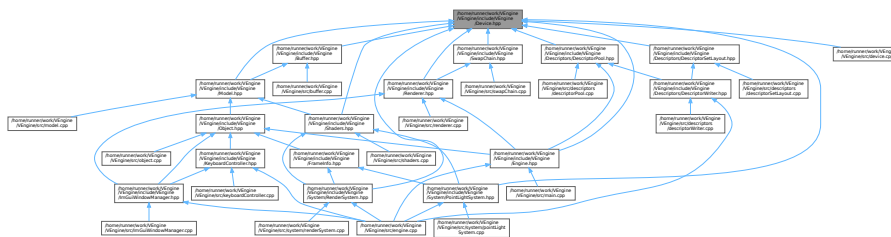
This file contains the Device class.

```
#include <vector>  
#include "VEngine/Window.hpp"
```

Include dependency graph for Device.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [ven::SwapChainSupportDetails](#)
- struct [ven::QueueFamilyIndices](#)
- class [ven::Device](#)

## Namespaces

- namespace [ven](#)

### 7.13.1 Detailed Description

This file contains the Device class.

Definition in file [Device.hpp](#).

## 7.14 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     class Device {
00030     public:
00031
00032         #ifdef NDEBUG
00033             const bool enableValidationLayers = false;
00034         #else
00035             const bool enableValidationLayers = true;
00036         #endif
00037
00038         explicit Device(Window &window);
00039         ~Device();
00040
00041         Device(const Device &) = delete;
00042         Device& operator=(const Device &) = delete;
00043         Device(Device &&) = delete;
00044         Device &operator=(Device &&) = delete;
00045
00046         [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
00047         [[nodiscard]] VkDevice device() const { return m_device; }
00048         [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
00049         [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
00050         [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }
00051
00052         [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return
querySwapChainSupport(m_physicalDevice); }
00053         [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags
properties) const;
00054         [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
findQueueFamilies(m_physicalDevice); }
00055         [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
VkImageTiling tiling, VkFormatFeatureFlags features) const;
00056
00057         // Buffer Helper Functions
00058         void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00059         [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00060         void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00061         void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00062         void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
uint32_t layerCount) const;
00063
00064         void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00065
00066         [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00067         [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00068
00069         VkPhysicalDeviceProperties properties_;
00070
00071     private:
00072         void createInstance();
00073
00074

```

```

00075         void setupDebugMessenger();
00076         void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00077         void pickPhysicalDevice();
00078         void createLogicalDevice();
00079         void createCommandPool();
00080
00081         // helper functions
00082         bool isDeviceSuitable(VkPhysicalDevice device) const;
00083         [[nodiscard]] std::vector<const char*> getRequiredExtensions() const;
00084         [[nodiscard]] bool checkValidationLayerSupport() const;
00085         QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00086         static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
&createInfo);
00087         void hasGlfwRequiredInstanceExtensions() const;
00088         bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00089         SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00090
00091         VkInstance m_instance;
00092         VkDebugUtilsMessengerEXT m_debugMessenger;
00093         VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00094         Window &m_window;
00095         VkCommandPool m_commandPool;
00096
00097         VkDevice m_device;
00098         VkSurfaceKHR m_surface;
00099         VkQueue m_graphicsQueue;
00100         VkQueue m_presentQueue;
00101
00102         const std::vector<const char*> validationLayers = {"VK_LAYER_KHRONOS_validation"};
00103         const std::vector<const char*> deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00104
00105     }; // class Device
00106
00107 } // namespace ven

```

## 7.15 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

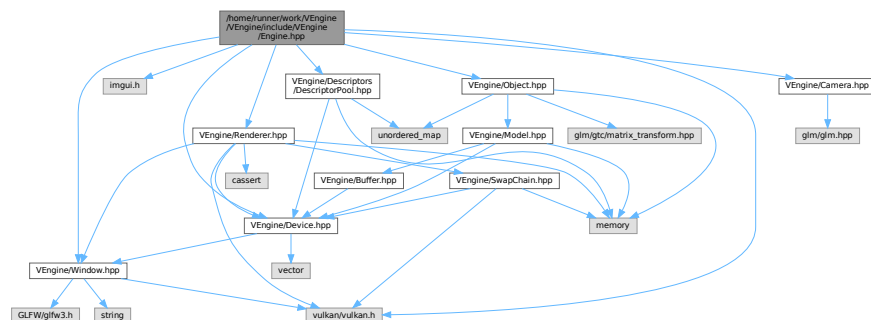
This file contains the Engine class.

```

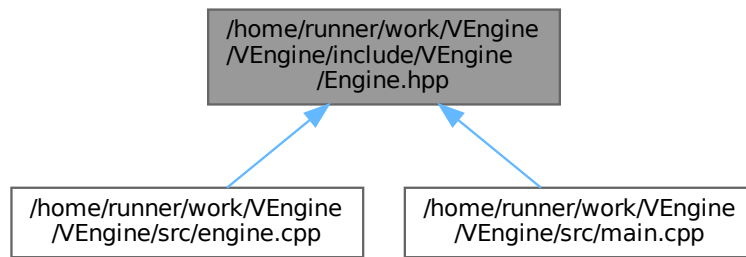
#include <vulkan/vulkan.h>
#include <imgui.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Camera.hpp"

```

Include dependency graph for Engine.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::Engine](#)

## Namespaces

- namespace [ven](#)

### 7.15.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

## 7.16 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 <imgui.h>
00012
00013 #include "VEngine/Window.hpp"
00014 #include "VEngine/Device.hpp"
00015 #include "VEngine/Object.hpp"
00016 #include "VEngine/Renderer.hpp"
00017 #include "VEngine/Descriptors/DescriptorPool.hpp"
00018 #include "VEngine/Camera.hpp"
00019
00020 namespace ven {
00021
00022     class Engine {
00023     public:
00024
00025 
```

```

00026     explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string &title
= DEFAULT_TITLE.data());
00027     ~Engine() = default;
00028
00029     Engine(const Engine &) = delete;
00030     Engine operator=(const Engine &) = delete;
00031
00032     void mainLoop();
00033
00034 private:
00035
00036     void loadObjects();
00037
00038     Window m_window;
00039     Device m_device{m_window};
00040     Renderer m_renderer{m_window, m_device};
00041
00042     std::unique_ptr<DescriptorPool> m_globalPool;
00043     Object::Map m_objects;
00044
00045     VkInstance m_instance{nullptr};
00046     VkSurfaceKHR m_surface{nullptr};
00047
00048     void createInstance();
00049     void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWWindow(),
nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00050
00051 }; // class Engine
00052
00053 } // namespace ven

```

## 7.17 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference

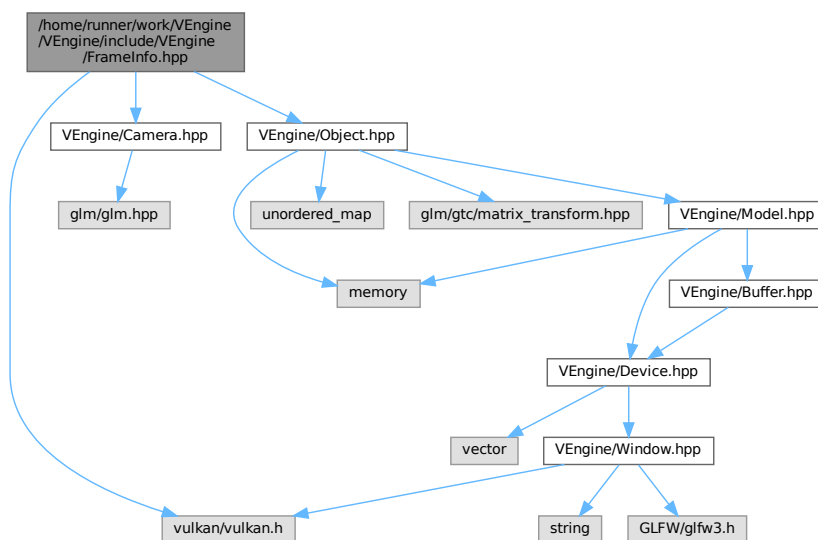
This file contains the FrameInfo class.

```

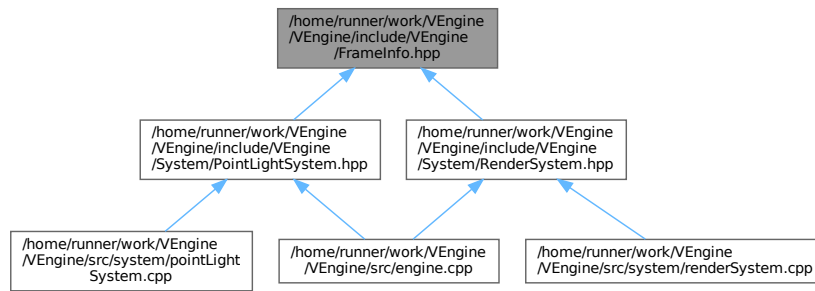
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"

```

Include dependency graph for FrameInfo.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [ven::PointLight](#)
- struct [ven::GlobalUbo](#)
- struct [ven::FrameInfo](#)

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr std::size\_t [ven::MAX\\_LIGHTS](#) = 10

### 7.17.1 Detailed Description

This file contains the FrameInfo class.

Definition in file [FrameInfo.hpp](#).

## 7.18 FrameInfo.hpp

[Go to the documentation of this file.](#)

```

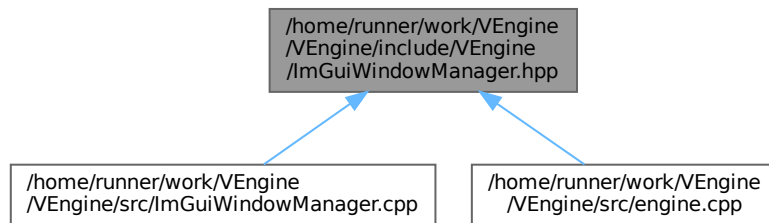
00001 ///
00002 /// @file FrameInfo.hpp
00003 /// @brief This file contains the FrameInfo class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Camera.hpp"
00012 #include "VEngine/Object.hpp"
00013
00014 namespace ven {
00015
00016 static constexpr std::size_t MAX_LIGHTS = 10;

```





This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::ImGuiWindowManager](#)  
*Class for ImGui window manager.*
- struct [ven::ImGuiWindowManager::funcs](#)

## Namespaces

- namespace [ven](#)

### 7.19.1 Detailed Description

This file contains the ImGuiWindowManager class.

Definition in file [ImGuiWindowManager.hpp](#).

## 7.20 ImGuiWindowManager.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file ImGuiWindowManager.hpp
00003 /// @brief This file contains the ImGuiWindowManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <imgui_impl_glfw.h>
00010 #include <imgui_impl_vulkan.h>
00011 #include <imgui.h>
00012 #include <imgui_internal.h>
00013
00014 #include "VEngine/Object.hpp"
00015 #include "VEngine/Renderer.hpp"
00016 #include "VEngine/Camera.hpp"
00017 #include "VEngine/KeyboardController.hpp"
00018
00019 namespace ven {
00020
00021     ///
00022     /// @class ImGuiWindowManager
00023     /// @brief Class for ImGui window manager
00024     /// @namespace ven
  
```

```

00025     ///
00026     class ImGuiWindowManager {
00027     public:
00028
00029         ImGuiWindowManager() = default;
00030         ~ImGuiWindowManager() = default;
00031
00032         ImGuiWindowManager(const ImGuiWindowManager&) = delete;
00033         ImGuiWindowManager& operator=(const ImGuiWindowManager&) = delete;
00034
00035         static void init(GLFWwindow* window, VkInstance instance, Device* device, VkRenderPass
00036 renderPass);
00037         static void render(Renderer *renderer, std::unordered_map<unsigned int, Object>& objects,
00038 ImGuiIO& io, Object& cameraObj, Camera& camera, KeyboardController& cameraController, VkPhysicalDevice
00039 physicalDevice);
00040         static void cleanup();
00041     private:
00042         struct funcs { static bool IsLegacyNativeDupe(ImGuiKey key) { return key >= 0 && key < 512
00043 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist
00044     }; // class ImGuiWindowManager
00045
00046 } // namespace ven

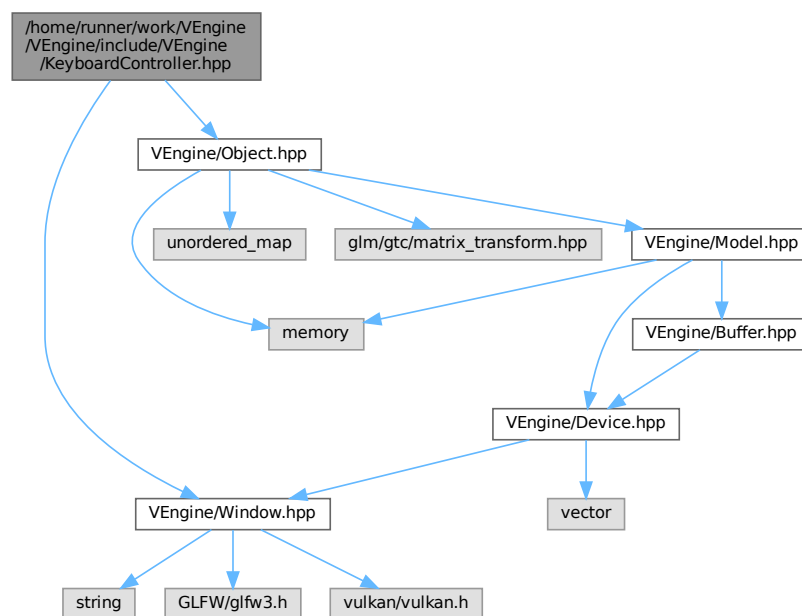
```

## 7.21 /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp File Reference

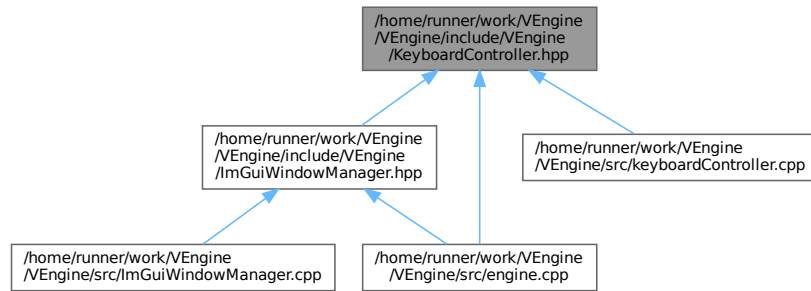
```
#include "VEngine/Window.hpp"
```

```
#include "VEngine/Object.hpp"
```

Include dependency graph for KeyboardController.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `ven::KeyboardController`  
*Class for keyboard controller.*
- struct `ven::KeyboardController::KeyMappings`

## Namespaces

- namespace `ven`

## Variables

- static constexpr float `ven::DEFAULT_MOVE_SPEED` = 3.F
- static constexpr float `ven::DEFAULT_LOOK_SPEED` = 1.5F

## 7.22 KeyboardController.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Camera.hpp
00003 /// @brief This file contains the KeyboardController class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Window.hpp"
00010 #include "VEngine/Object.hpp"
00011
00012 namespace ven {
00013
00014     static constexpr float DEFAULT_MOVE_SPEED = 3.F;
00015     static constexpr float DEFAULT_LOOK_SPEED = 1.5F;
00016
00017     ///
00018     /// @class KeyboardController
00019     /// @brief Class for keyboard controller
00020     /// @namespace ven
00021     ///
00022     class KeyboardController {
00023
00024     public:
00025
00026         struct KeyMappings {

```

```

00027         int moveLeft = GLFW_KEY_A;
00028         int moveRight = GLFW_KEY_D;
00029         int moveForward = GLFW_KEY_W;
00030         int moveBackward = GLFW_KEY_S;
00031         int moveUp = GLFW_KEY_SPACE;
00032         int moveDown = GLFW_KEY_LEFT_SHIFT;
00033         int lookLeft = GLFW_KEY_LEFT;
00034         int lookRight = GLFW_KEY_RIGHT;
00035         int lookUp = GLFW_KEY_UP;
00036         int lookDown = GLFW_KEY_DOWN;
00037     };
00038
00039     void moveInPlaneXZ(GLFWwindow* window, float dt, Object& object, bool* showDebugWindow)
00040     const;
00041
00041     KeyMappings m_keys{};
00042     float m_moveSpeed{DEFAULT_MOVE_SPEED};
00043     float m_lookSpeed{DEFAULT_LOOK_SPEED};
00044
00045     }; // class KeyboardController
00046
00047 } // namespace ven

```

## 7.23 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference

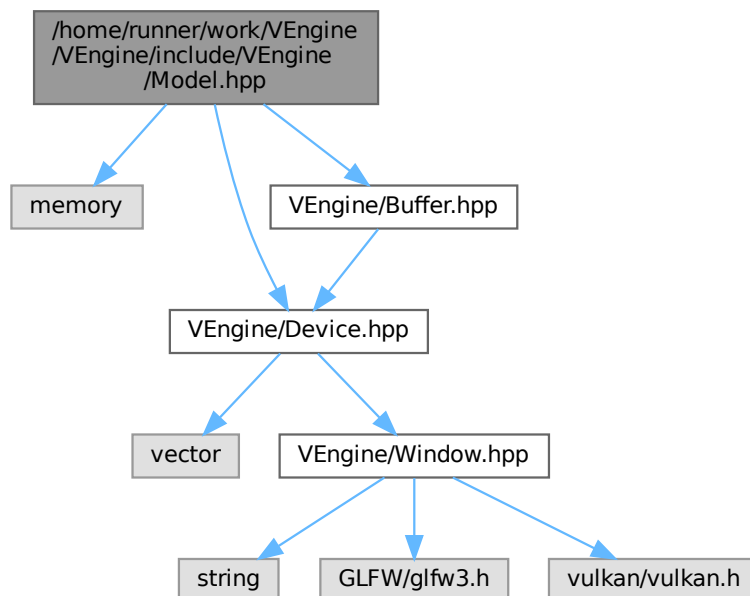
This file contains the Model class.

```

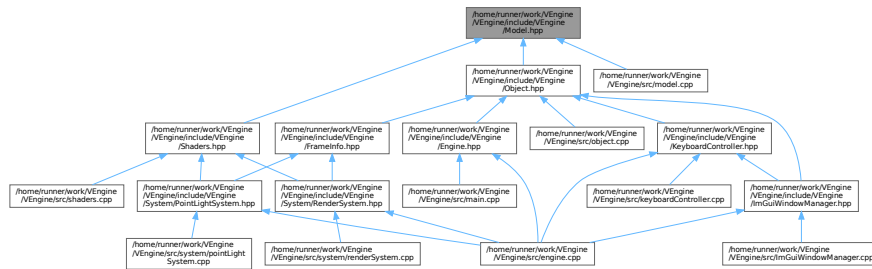
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"

```

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`

### 7.23.1 Detailed Description

This file contains the Model class.

Definition in file [Model.hpp](#).

## 7.24 Model.hpp

[Go to the documentation of this file.](#)

```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 #include <memory>  
00009 #include "VEngine/Device.hpp"  
00010 #include "VEngine/Buffer.hpp"  
00011 namespace ven {  
00012     ///  
00013     ///  
00014     ///  
00015     ///  
00016     ///  
00017     ///  
00018     ///  
00019     ///  
00020     ///  
00021     class Model {  
00022     public:  
00023         struct Vertex {  
00024             glm::vec3 position{};  
00025         }  
00026     };
```

```

00027         glm::vec3 color{};
00028         glm::vec3 normal{};
00029         glm::vec2 uv{};
00030
00031         static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00032         static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00033
00034         bool operator==(const Vertex& other) const {
00035             return position == other.position && color == other.color && normal ==
other.normal && uv == other.uv;
00036         }
00037     };
00038
00039     struct Builder {
00040         std::vector<Vertex> vertices;
00041         std::vector<uint32_t> indices;
00042
00043         void loadModel(const std::string &filename);
00044     };
00045
00046     Model(Device &device, const Builder &builder);
00047     ~Model();
00048
00049     Model(const Model&) = delete;
00050     void operator=(const Model&) = delete;
00051
00052     static std::unique_ptr<Model> createModelFromFile(Device &device, const std::string
&filename);
00053
00054     void bind(VkCommandBuffer commandBuffer) const;
00055     void draw(VkCommandBuffer commandBuffer) const;
00056
00057     private:
00058
00059         void createVertexBuffer(const std::vector<Vertex>& vertices);
00060         void createIndexBuffer(const std::vector<uint32_t>& indices);
00061
00062         Device& m_device;
00063         std::unique_ptr<Buffer> m_vertexBuffer;
00064         uint32_t m_vertexCount;
00065
00066         bool m_hasIndexBuffer{false};
00067         std::unique_ptr<Buffer> m_indexBuffer;
00068         uint32_t m_indexCount;
00069
00070     }; // class Model
00071
00072 } // namespace ven

```

## 7.25 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

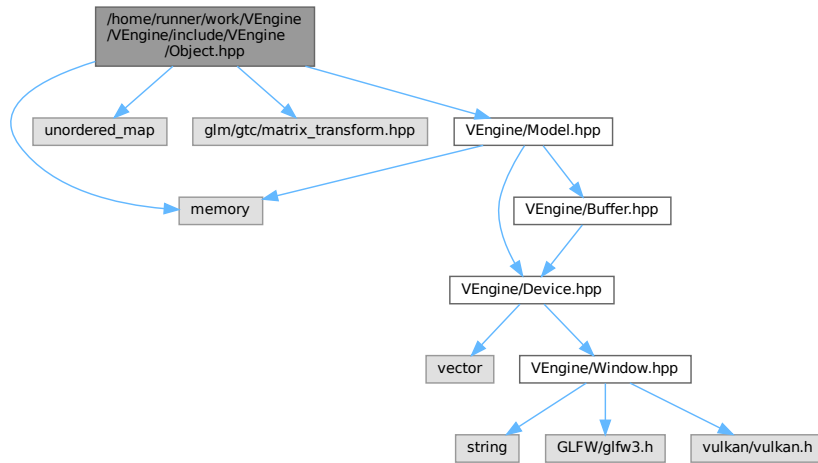
This file contains the Object class.

```

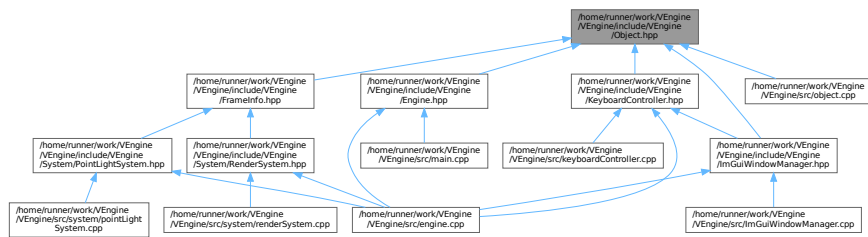
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"

```

Include dependency graph for Object.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [ven::Transform3DComponent](#)
- struct [ven::PointLightComponent](#)
- class [ven::Object](#)

*Class for object.*

## Namespaces

- namespace [ven](#)

## Variables

- static constexpr float [ven::DEFAULT\\_LIGHT\\_INTENSITY](#) = .2F
- static constexpr float [ven::DEFAULT\\_LIGHT\\_RADIUS](#) = 0.1F
- static constexpr glm::vec3 [ven::DEFAULT\\_LIGHT\\_COLOR](#) = glm::vec3(1.F)

### 7.25.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

## 7.26 Object.hpp

[Go to the documentation of this file.](#)

```

00001 ///  

00002 ///  

00003 ///  

00004 ///  

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 ///  

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::vec3 DEFAULT_LIGHT_COLOR = glm::vec3(1.F);  

00021 ///  

00022     struct Transform3DComponent {  

00023         glm::vec3 translation{};  

00024         glm::vec3 scale{1.F, 1.F, 1.F};  

00025         glm::vec3 rotation{};  

00026 ///  

00027         [[nodiscard]] glm::mat4 mat4() const;  

00028         [[nodiscard]] glm::mat3 normalMatrix() const;  

00029     };  

00030 ///  

00031     struct PointLightComponent {  

00032         float lightIntensity = DEFAULT_LIGHT_INTENSITY;  

00033     };  

00034 ///  

00035     ///  

00036     ///  

00037     ///  

00038     ///  

00039     ///  

00040     class Object {  

00041     public:  

00042         using Map = std::unordered_map<unsigned int, Object>;  

00043         ~Object() = default;  

00044         Object(const Object&) = delete;  

00045         Object& operator=(const Object&) = delete;  

00046         Object(Object&&) = default;  

00047         Object& operator=(Object&&) = default;  

00048         static Object createObject() { static unsigned int objId = 0; return Object(objId++); }  

00049         static Object makePointLight(float intensity = DEFAULT_LIGHT_INTENSITY, float radius =  

00050         DEFAULT_LIGHT_RADIUS, glm::vec3 color = DEFAULT_LIGHT_COLOR);  

00051         [[nodiscard]] unsigned int getId() const { return m_objId; }  

00052         std::shared_ptr<Model> model{};  

00053         glm::vec3 color{};  

00054         Transform3DComponent transform3D{};  

00055         std::string name{};  

00056         std::unique_ptr<PointLightComponent> pointLight = nullptr;  

00057     private:  

00058         explicit Object(const unsigned int objId) : m_objId(objId) {}  

00059         unsigned int m_objId;  

00060     }; // class Object  

00061 } // namespace ven

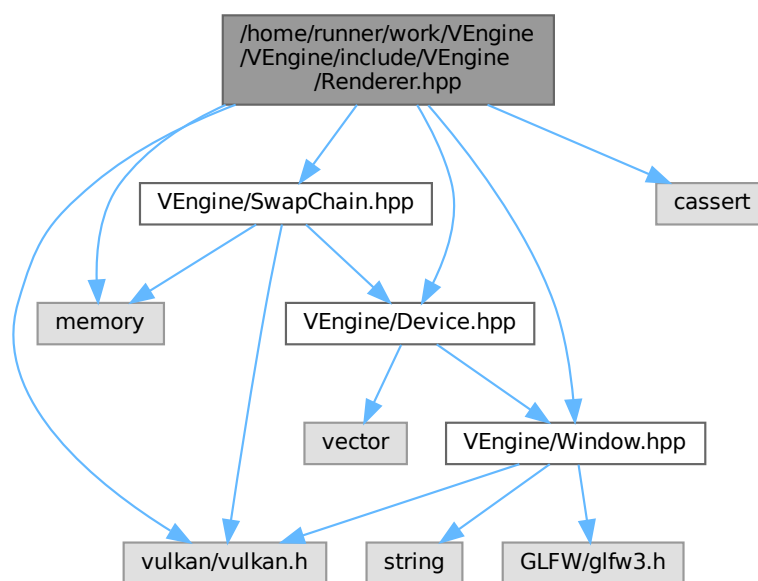
```



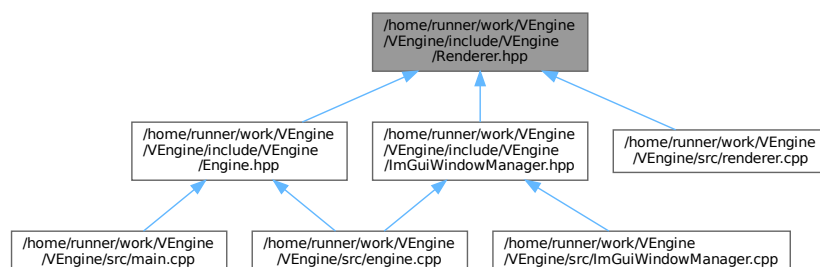
## 7.27 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

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`

## 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}

### 7.27.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

## 7.28 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     class Renderer {
00024     public:
00025
00026         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
00027             recreateSwapChain(); createCommandBuffers(); }
00028         ~Renderer() { freeCommandBuffers(); }
00029
00030         Renderer(const Renderer &) = delete;
00031         Renderer& operator=(const Renderer &) = delete;
00032
00033         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
00034             m_swapChain->getRenderPass(); }
00035         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00036         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00037         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress()
00038             && "cannot get command m_buffer when frame not in progress"); return
00039             m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00040
00041         [[nodiscard]] int getFrameIndex() const { assert(isFrameInProgress() && "cannot get frame
00042             index when frame not in progress"); return m_currentFrameIndex; }
00043         [[nodiscard]] std::array<float, 4> getClearColor() const { return {
00044             m_clearValues[0].color.float32[0],
00045             m_clearValues[0].color.float32[1],
00046             m_clearValues[0].color.float32[2],
00047             m_clearValues[0].color.float32[3]
00048         }; }
00049     };
00050 }
```

```

00046         void setClearColor(VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00047         VkCommandBuffer beginFrame();
00048         void endFrame();
00049         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer);
00050         void endSwapChainRenderPass(VkCommandBuffer commandBuffer);
00051
00052     private:
00053
00054         void createCommandBuffers();
00055         void freeCommandBuffers();
00056         void recreateSwapChain();
00057
00058         Window &m_window;
00059         Device &m_device;
00060         std::unique_ptr<SwapChain> m_swapChain;
00061         std::vector<VkCommandBuffer> m_commandBuffers;
00062         std::array<VkClearColorValue, 2> m_clearValues;
00063
00064         uint32_t m_currentImageIndex{0};
00065         int m_currentFrameIndex{0};
00066         bool m_isFrameStarted{false};
00067
00068     }; // class Renderer
00069
00070 } // namespace ven

```

## 7.29 /home/runner/work/VEngine/VEngine/include/VEngine/Shader.h File Reference

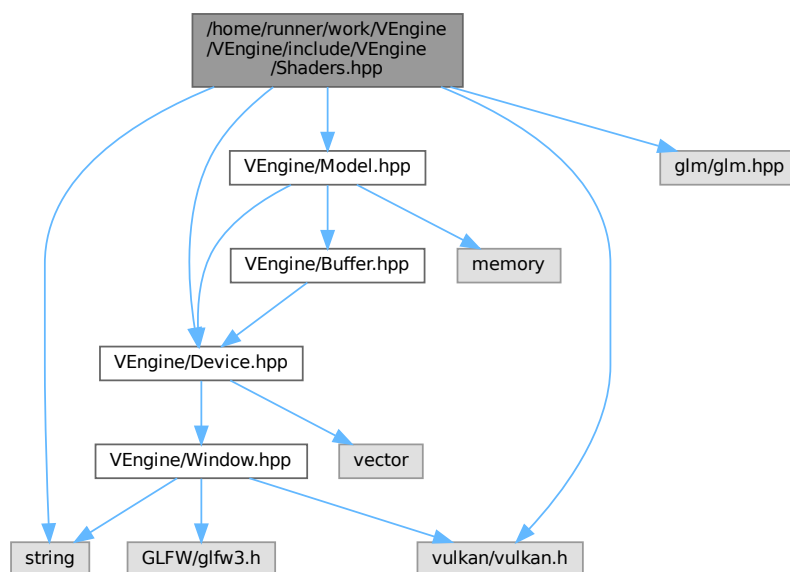
This file contains the Shader class.

```

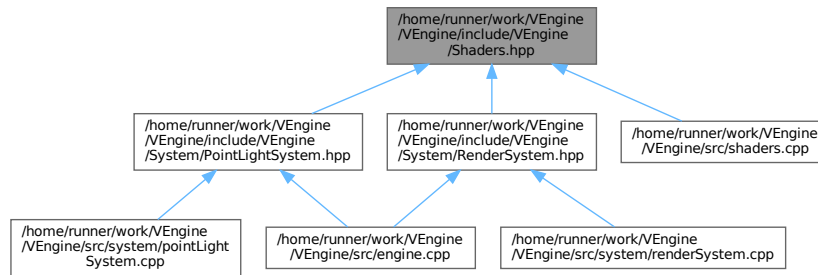
#include <string>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
#include "VEngine/Device.hpp"
#include "VEngine/Model.hpp"

```

Include dependency graph for Shader.h:



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](#) = "shaders/bin/"

## 7.29.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

## 7.30 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 #include <glm/glm.hpp>
00013
00014 #include "VEngine/Device.hpp"
00015 #include "VEngine/Model.hpp"
00016
00017 namespace ven {

```

```

00018
00019     static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/";
00020
00021     struct PipelineConfigInfo {
00022         PipelineConfigInfo() = default;
00023         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00024         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00025
00026         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00027         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00028         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00029         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00030         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00031         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00032         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00033         VkPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00034         std::vector<VkDynamicState> dynamicStateEnables;
00035         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00036         VkPipelineLayout pipelineLayout = nullptr;
00037         VkRenderPass renderPass = nullptr;
00038         uint32_t subpass = 0;
00039     };
00040
00041     ///
00042     /// @class Shaders
00043     /// @brief Class for shaders
00044     /// @namespace ven
00045     ///
00046     class Shaders {
00047
00048     public:
00049
00050         Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
00051 const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
00052 fragFilepath, configInfo); };
00053         ~Shaders();
00054
00055         Shaders(const Shaders&) = delete;
00056         Shaders& operator=(const Shaders&) = delete;
00057
00058         static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00059         void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
00060 VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00061
00062     private:
00063
00064         static std::vector<char> readFile(const std::string &filename);
00065         void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
00066 fragFilepath, const PipelineConfigInfo& configInfo);
00067         void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
00068 const;
00069
00070         Device& m_device;
00071         VkPipeline m_graphicsPipeline{nullptr};
00072         VkShaderModule m_vertShaderModule{nullptr};
00073         VkShaderModule m_fragShaderModule{nullptr};
00074     }; // class Shaders
00075 } // namespace ven

```

## 7.31 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

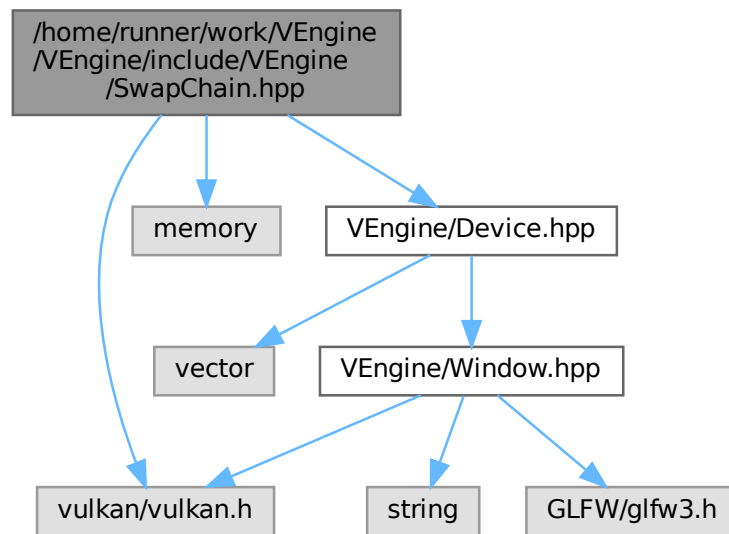
This file contains the Shader class.

```

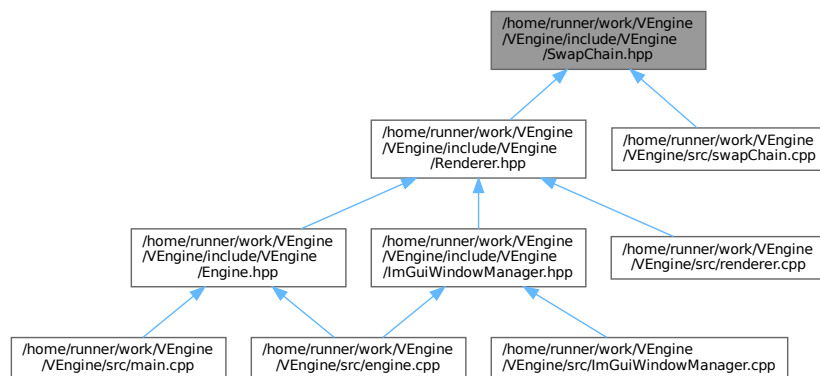
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"

```

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](#)

### 7.31.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

## 7.32 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     ///
00017     /// @class SwapChain
00018     /// @brief Class for swap chain
00019     /// @namespace ven
00020     ///
00021     class SwapChain {
00022
00023     public:
00024
00025         static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00026
00027         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : device{deviceRef},
00028 windowExtent{windowExtentRef} { init(); }
00029         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
00030 previous) : device{deviceRef}, windowExtent{windowExtentRef}, oldSwapChain{std::move(previous)} {
00031 init(); 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 swapChainFrameBuffers[index]; }
00039         [[nodiscard]] VkRenderPass getRenderPass() const { return renderPass; }
00040         [[nodiscard]] VkImageView getImageView(const int index) const { return
00041 swapChainImageViews[static_cast<unsigned long>(index)]; }
00042         [[nodiscard]] size_t imageCount() const { return swapChainImages.size(); }
00043         [[nodiscard]] VkFormat getSwapChainImageFormat() const { return 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         [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChainp) const {
00056 return swapChainImageFormat == swapChainp.swapChainImageFormat && swapChainDepthFormat
00057 == swapChainp.swapChainDepthFormat;
00058 }
00059
00060     private:
00061
00062         void init();
00063         void createSwapChain();
00064         void createImageViews();
00065         void createDepthResources();
00066         void createRenderPass();
00067         void createFrameBuffers();
00068         void createSyncObjects();

```

```

00063         static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats);
00064         static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes);
00065         [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
const;
00066
00067         VkFormat swapChainImageFormat{};
00068         VkFormat swapChainDepthFormat{};
00069         VkExtent2D m_swapChainExtent{};
00070
00071         std::vector<VkFramebuffer> swapChainFrameBuffers;
00072         VkRenderPass renderPass{};
00073
00074         std::vector<VkImage> depthImages;
00075         std::vector<VkDeviceMemory> depthImageMemory;
00076         std::vector<VkImageView> depthImageViews;
00077         std::vector<VkImage> swapChainImages;
00078         std::vector<VkImageView> swapChainImageViews;
00079
00080         Device &device;
00081         VkExtent2D windowExtent;
00082
00083         VkSwapchainKHR swapChain{};
00084         std::shared_ptr<SwapChain> oldSwapChain;
00085
00086         std::vector<VkSemaphore> imageAvailableSemaphores;
00087         std::vector<VkSemaphore> renderFinishedSemaphores;
00088         std::vector<VkFence> inFlightFences;
00089         std::vector<VkFence> imagesInFlight;
00090         size_t currentFrame = 0;
00091
00092     }; // class SwapChain
00093
00094 } // namespace ven

```

## 7.33 /home/runner/work/VEngine/VEngine/include/VEngine/System/↵ PointLightSystem.hpp File Reference

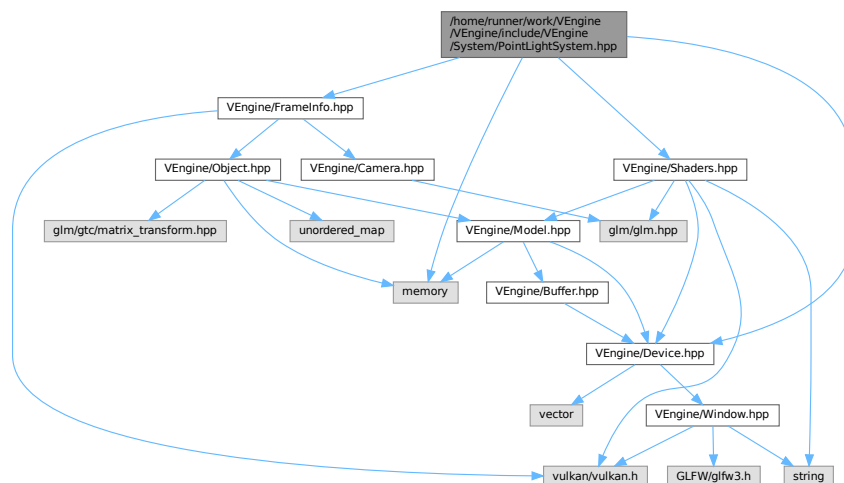
This file contains the PointLightSystem class.

```

#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

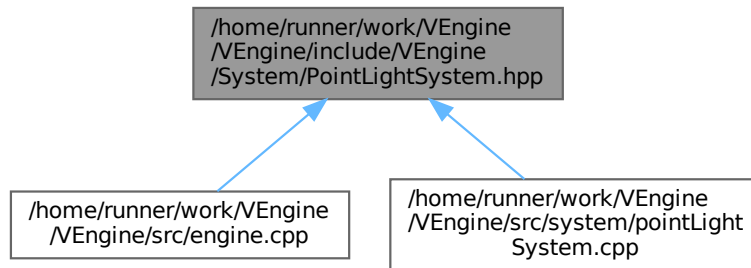
```

Include dependency graph for PointLightSystem.hpp:





This graph shows which files directly or indirectly include this file:



## Classes

- class [ven::PointLightSystem](#)  
*Class for point light system.*

## Namespaces

- namespace [ven](#)

### 7.33.1 Detailed Description

This file contains the PointLightSystem class.

Definition in file [PointLightSystem.hpp](#).

## 7.34 PointLightSystem.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 PointLightSystem {  

00023

```

```

00024     public:
00025
00026         explicit PointLightSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
globalSetLayout);
00027         ~PointLightSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout,
nullptr); }
00028
00029         PointLightSystem(const PointLightSystem&) = delete;
00030         PointLightSystem& operator=(const PointLightSystem&) = delete;
00031
00032         static void update(const FrameInfo &frameInfo, GlobalUbo &ubo);
00033         void render(const FrameInfo &frameInfo) const;
00034
00035     private:
00036
00037         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00038         void createPipeline(VkRenderPass renderPass);
00039
00040         Device &m_device;
00041
00042         std::unique_ptr<Shaders> m_shaders;
00043         VkPipelineLayout m_pipelineLayout{nullptr};
00044
00045 }; // class PointLightSystem
00046
00047 } // namespace ven

```

## 7.35 /home/runner/work/VEngine/VEngine/include/VEngine/System/↵ RenderSystem.hpp File Reference

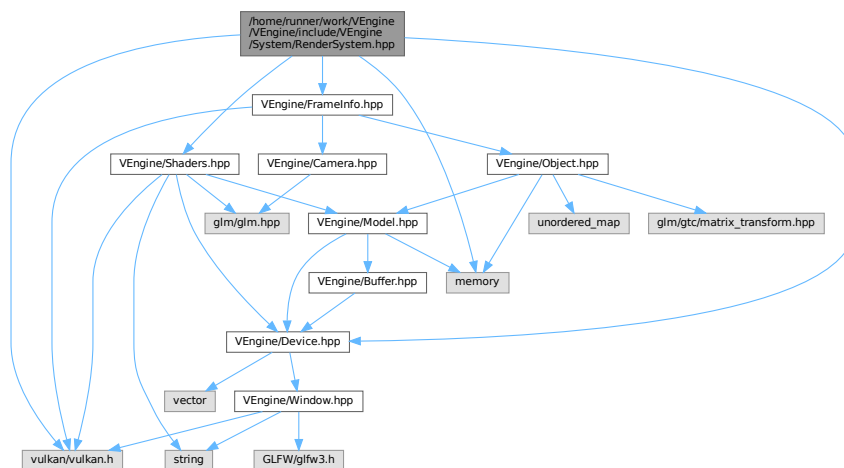
This file contains the RenderSystem class.

```

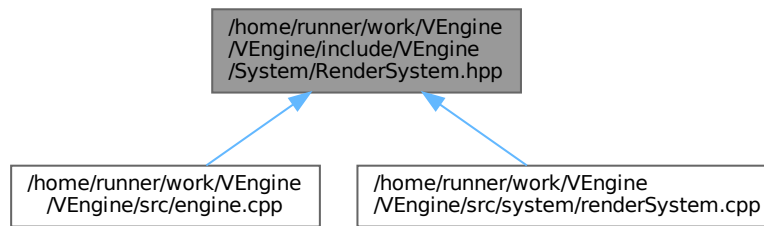
#include <memory>
#include <vulkan/vulkan.h>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

```

Include dependency graph for RenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [ven::SimplePushConstantData](#)
- class [ven::RenderSystem](#)  
*Class for render system.*

## Namespaces

- namespace [ven](#)

### 7.35.1 Detailed Description

This file contains the RenderSystem class.

Definition in file [RenderSystem.hpp](#).

## 7.36 RenderSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file RenderSystem.hpp
00003 /// @brief This file contains the RenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Shaders.hpp"
00015 #include "VEngine/FrameInfo.hpp"
00016
00017 namespace ven {
00018
00019     struct SimplePushConstantData {
00020         glm::mat4 modelMatrix{1.F};
00021         glm::mat4 normalMatrix{1.F};
00022     };
00023
00024     ///
  
```

```

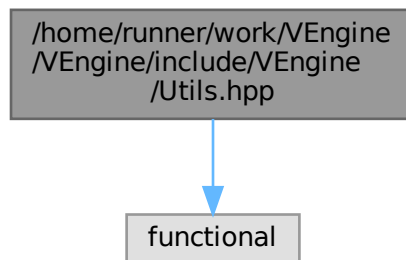
00025     /// @class RenderSystem
00026     /// @brief Class for render system
00027     /// @namespace ven
00028     ///
00029     class RenderSystem {
00030
00031     public:
00032
00033         explicit RenderSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
globalSetLayout);
00034         ~RenderSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout, nullptr); }
00035
00036         RenderSystem(const RenderSystem&) = delete;
00037         RenderSystem& operator=(const RenderSystem&) = delete;
00038
00039         void renderObjects(const FrameInfo &frameInfo) const;
00040
00041     private:
00042
00043         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00044         void createPipeline(VkRenderPass renderPass);
00045
00046         Device &m_device;
00047         std::unique_ptr<Shaders> m_shaders;
00048         VkPipelineLayout m_pipelineLayout{nullptr};
00049
00050     }; // class RenderSystem
00051
00052 } // namespace ven

```

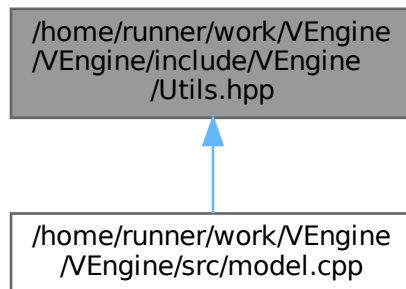
## 7.37 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

#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)`

## 7.38 Utils.hpp

[Go to the documentation of this file.](#)

```

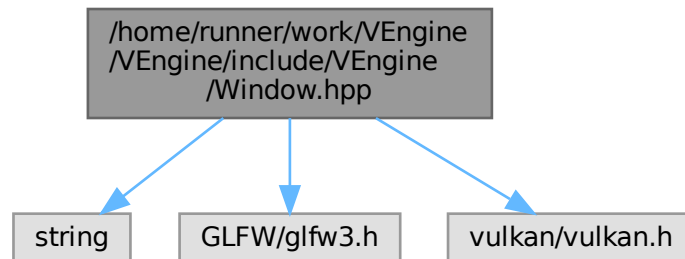
00001 ///
00002 /// @file Utils.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018
00019 } // namespace ven
  
```

## 7.39 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp

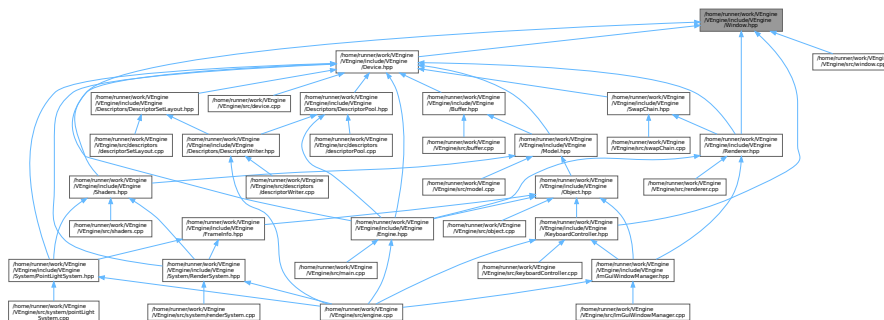
### File Reference

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"

### 7.39.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

### 7.39.2 Macro Definition Documentation

#### 7.39.2.1 GLFW\_INCLUDE\_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

## 7.40 Window.hpp

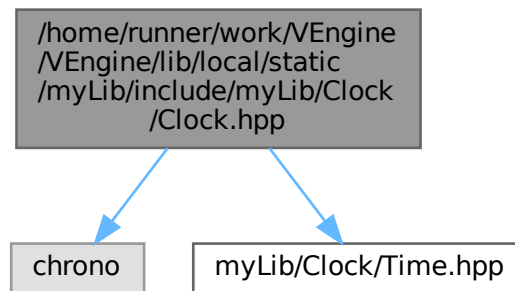
[Go to the documentation of this file.](#)

```
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
00028     public:
00029
00030         Window(const uint32_t width, const uint32_t height, const std::string &title) :
00031             m_window(createWindow(width, height, title)), m_width(width), m_height(height) {};
00032         ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr; };
00033
00034         [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
00035         &title);
00036         void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00037
00038         [[nodiscard]] GLFWwindow* getGLFWWindow() const { return m_window; };
00039
00040         [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; };
00041         [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00042         void resetWindowResizedFlag() { m_framebufferResized = false; }
00043
00044     private:
00045
00046         static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00047
00048         GLFWwindow* m_window{nullptr};
00049         uint32_t m_width;
00050         uint32_t m_height;
00051
00052         bool m_framebufferResized = false;
00053
00054     }; // class Window
00055 } // namespace ven
```

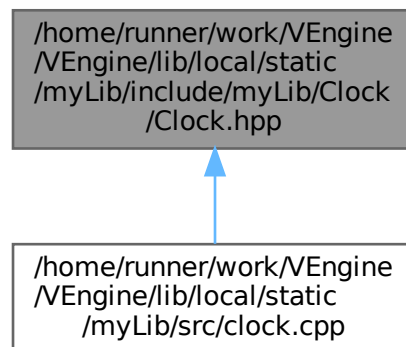
## 7.41 `/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp` File Reference

Clock class for time management.

```
#include <chrono>
#include "myLib/Clock/Time.hpp"
Include dependency graph for Clock.hpp:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class `myLib::Clock`  
*Class for time management.*



## Namespaces

- namespace [myLib](#)

## Typedefs

- using [TimePoint](#) = std::chrono::time\_point<std::chrono::high\_resolution\_clock>  
*TimePoint is a type alias for a time point which is a very long and complicated type in the standard library.*

### 7.41.1 Detailed Description

Clock class for time management.

Definition in file [Clock.hpp](#).

### 7.41.2 Typedef Documentation

#### 7.41.2.1 TimePoint

```
using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>
```

TimePoint is a type alias for a time point which is a very long and complicated type in the standard library.

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

## 7.42 Clock.hpp

[Go to the documentation of this file.](#)

```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <chrono>  
00010 ///  
00011 #include "myLib/Clock/Time.hpp"  
00012 ///  
00013 ///  
00014 ///  
00015 ///  
00016 using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;  
00017 ///  
00018 namespace myLib {  
00019     ///  
00020     ///  
00021     ///  
00022     ///  
00023     class Clock {  
00024     public:  
00025     public:  
00026     public:  
00027         Clock() : m_start(std::chrono::high_resolution_clock::now()) {};  
00028         ~Clock() = default;  
00029     public:  
00030     public:  
00031     public:  
00032     public: // @brief Restart the clock
```

```

00033         ///  

00034         void restart() { m_start = std::chrono::high_resolution_clock::now(); };  

00035         ///  

00036         ///  

00037         ///  

00038         ///  

00039         void pause();  

00040         ///  

00041         ///  

00042         ///  

00043         ///  

00044         void resume();  

00045         ///  

00046         ///  

00047         ///  

00048         ///  

00049         ///  

00050         [[nodiscard]] Time getElapsedTime() const;  

00051     private:  

00052     private:  

00053     private:  

00054         ///  

00055         ///  

00056         ///  

00057         TimePoint m_start;  

00058         ///  

00059         ///  

00060         ///  

00061         ///  

00062         TimePoint m_pause;  

00063         ///  

00064         ///  

00065         ///  

00066         ///  

00067         bool m_paused{false};  

00068     }; // Clock  

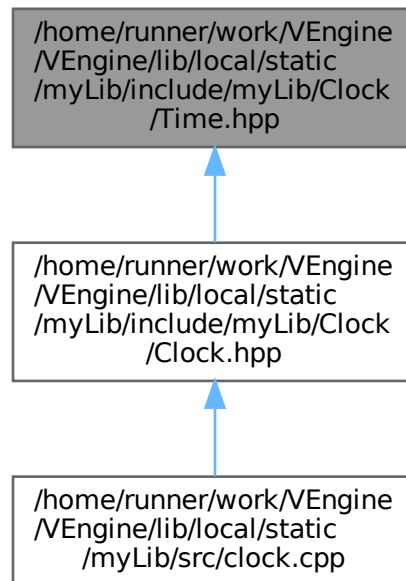
00070 } // namespace myLib

```

## 7.43 /home/runner/work/VEngine/VEngine/lib/local/static/my↵ Lib/include/myLib/Clock/Time.hpp File Reference

Class for time management.

This graph shows which files directly or indirectly include this file:



## Classes

- class [myLib::Time](#)  
*Class used for time management.*

## Namespaces

- namespace [myLib](#)

## Variables

- static constexpr unsigned int [myLib::MICROSECONDS\\_PER\\_SECOND](#) = 1000000
- static constexpr unsigned int [myLib::MILLISECONDS\\_PER\\_SECOND](#) = 1000

### 7.43.1 Detailed Description

Class for time management.

Definition in file [Time.hpp](#).

## 7.44 Time.hpp

[Go to the documentation of this file.](#)

```

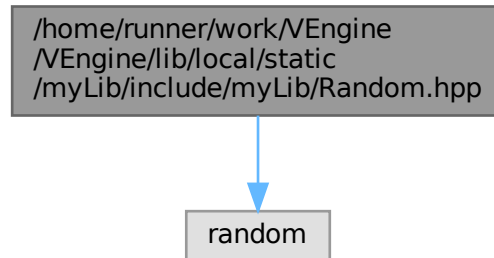
00001 ///
00002 /// @file Time.hpp
00003 /// @brief Class for time management
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 namespace myLib {
00010
00011     static constexpr unsigned int MICROSECONDS_PER_SECOND = 1000000;
00012     static constexpr unsigned int MILLISECONDS_PER_SECOND = 1000;
00013
00014     ///
00015     /// @class Time
00016     /// @brief Class used for time management
00017     ///
00018     class Time {
00019     public:
00020
00021         ///
00022         /// @brief Construct a new Time object
00023         ///
00024         explicit Time(const double seconds) : m_seconds(seconds) {};
00025
00026         ///
00027         /// @brief Transform the time to seconds
00028         /// @return int The time in seconds
00029         ///
00030         [[nodiscard]] int asSeconds() const { return static_cast<int>(m_seconds); };
00031
00032         ///
00033         /// @brief Transform the time to milliseconds
00034         /// @return int The time in milliseconds
00035         ///
00036         [[nodiscard]] int asMilliseconds() const { return static_cast<int>(m_seconds *
MILLISECONDS_PER_SECOND); };
00037
00038         ///
00039         /// @brief Transform the time to microseconds
00040         /// @return int The time in microseconds
00041         ///
00042         [[nodiscard]] int asMicroseconds() const { return static_cast<int>(m_seconds *
MICROSECONDS_PER_SECOND); };
00043
00044     private:
00045
00046         ///
00047         /// @property The time in seconds
00048         ///
00049         double m_seconds{0.0F};
00050
00051     }; // Time
00052
00053 } // namespace myLib
00054
```

## 7.45 /home/runner/work/VEngine/VEngine/lib/local/static/my↵ Lib/include/myLib/Random.hpp File Reference

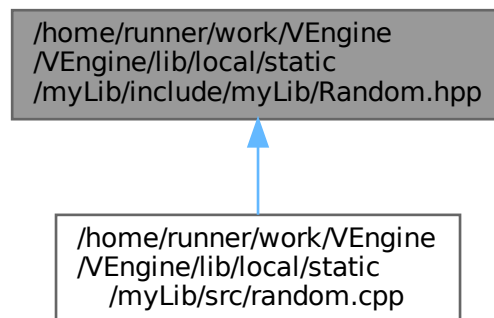
Class for random number generation.

```
#include <random>
```

Include dependency graph for Random.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `myLib::Random`  
*Class for random number generation.*

## Namespaces

- namespace `myLib`

## Variables

- static constexpr int `myLib::RANDOM_INT_MIN` = -1000
- static constexpr int `myLib::RANDOM_INT_MAX` = 1000
- static constexpr float `myLib::RANDOM_FLOAT_MAX` = 1000.0F

### 7.45.1 Detailed Description

Class for random number generation.

Definition in file [Random.hpp](#).

## 7.46 Random.hpp

[Go to the documentation of this file.](#)

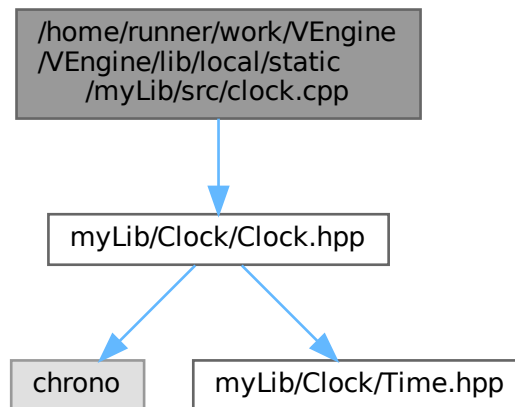
```

00001 ///
00002 /// @file Random.hpp
00003 /// @brief Class for random number generation
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 #include <random>
00010
00011 namespace myLib {
00012
00013     static constexpr int RANDOM_INT_MIN = -1000;
00014     static constexpr int RANDOM_INT_MAX = 1000;
00015     static constexpr float RANDOM_FLOAT_MAX = 1000.0F;
00016
00017     ///
00018     /// @class Random
00019     /// @brief Class for random number generation
00020     ///
00021     class Random {
00022     public:
00023
00024         ///
00025         /// @brief Generate a random integer between min and max
00026         /// @param min The minimum value
00027         /// @param max The maximum value
00028         /// @return int The random integer
00029         ///
00030         static int randomInt(int min, int max);
00031         static int randomInt() { return randomInt(-1000, 1000); };
00032
00033         ///
00034         /// @param min The minimum value
00035         /// @param max The maximum value
00036         /// @return float The random float
00037         ///
00038         static float randomFloat(float min, float max);
00039         static float randomFloat() { return randomFloat(-1.0F, 1.0F); };
00040
00041     }; // class Random
00042
00043 } // namespace myLib
00044
```

## 7.47 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp File Reference

```
#include "myLib/Clock/Clock.hpp"
```

Include dependency graph for clock.cpp:



## 7.48 clock.cpp

[Go to the documentation of this file.](#)

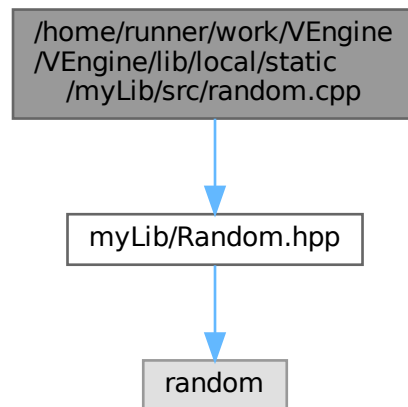
```

00001 #include "myLib/Clock/Clock.hpp"
00002
00003 void myLib::Clock::pause()
00004 {
00005     if (m_paused) {
00006         return;
00007     }
00008     m_pause = std::chrono::high_resolution_clock::now();
00009     m_paused = true;
00010 }
00011
00012 void myLib::Clock::resume()
00013 {
00014     if (!m_paused) {
00015         return;
00016     }
00017
00018     m_start += std::chrono::high_resolution_clock::now() - m_pause;
00019     m_paused = false;
00020 }
00021
00022 myLib::Time myLib::Clock::getElapsedTime() const
00023 {
00024     TimePoint now = std::chrono::high_resolution_clock::now();
00025     std::chrono::duration<float> elapsed_time{};
00026     if (m_paused) {
00027         elapsed_time = m_pause - m_start;
00028     } else {
00029         elapsed_time = now - m_start;
00030     }
00031     return Time(elapsed_time.count());
00032 }

```

## 7.49 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference

```
#include "myLib/Random.hpp"
Include dependency graph for random.cpp:
```



## 7.50 random.cpp

[Go to the documentation of this file.](#)

```
00001 #include "myLib/Random.hpp"
00002
00003 int myLib::Random::randomInt(const int min, const int max)
00004 {
00005     std::mt19937 gen(std::random_device{}());
00006     std::uniform_int_distribution<> dis(min, max);
00007     return dis(gen);
00008 }
00009
00010 float myLib::Random::randomFloat(const float min, const float max)
00011 {
00012     return min + (static_cast<float>(randomInt(RANDOM_INT_MIN, RANDOM_INT_MAX)) / RANDOM_FLOAT_MAX *
00013         (max - min));
00013 }
```

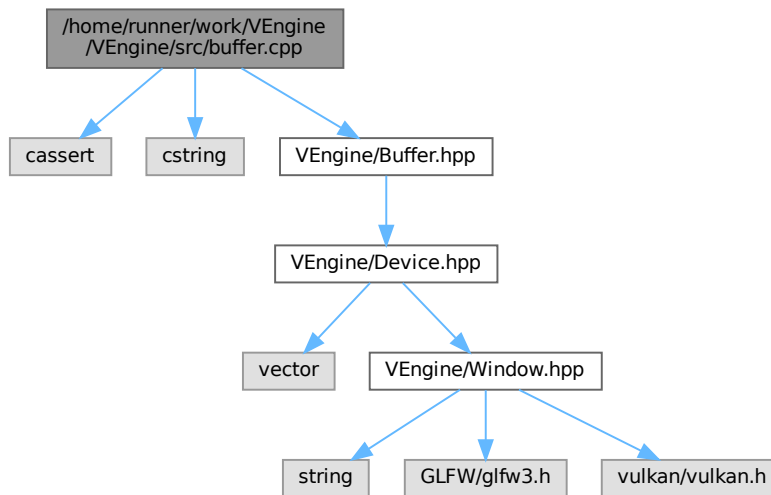
## 7.51 /home/runner/work/VEngine/VEngine/README.md File Reference

## 7.52 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```
#include <cassert>
#include <cstring>
```



#include "VEngine/Buffer.hpp"  
 Include dependency graph for buffer.cpp:



## 7.53 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 }

```

```

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 }

```

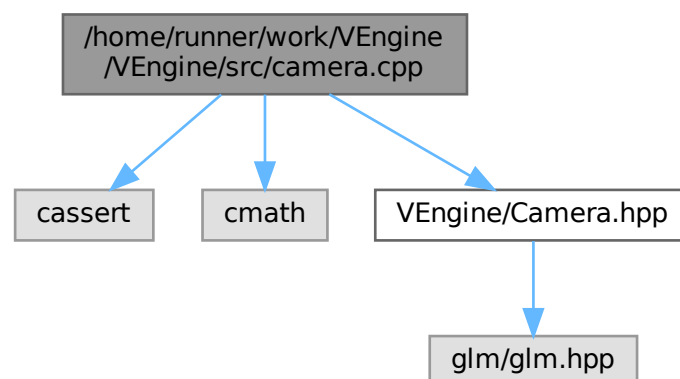
## 7.54 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```

#include <cassert>
#include <cmath>
#include "VEngine/Camera.hpp"

```

Include dependency graph for camera.cpp:



## 7.55 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,
const float bottom, const float near, const float far)
00007 {
00008     m_projectionMatrix = glm::mat4{1.0F};
00009     m_projectionMatrix[0][0] = 2.F / (right - left);
00010     m_projectionMatrix[1][1] = 2.F / (bottom - top);
00011     m_projectionMatrix[2][2] = 1.F / (far - near);
00012     m_projectionMatrix[3][0] = -(right + left) / (right - left);
00013     m_projectionMatrix[3][1] = -(bottom + top) / (bottom - top);
00014     m_projectionMatrix[3][2] = -near / (far - near);
00015 }
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
glm::vec3 up)
00030 {
00031     const glm::vec3 w(normalize(direction));
00032     const glm::vec3 u(normalize(cross(w, up)));
00033     const glm::vec3 v(cross(w, u));
00034
00035     m_viewMatrix = glm::mat4{1.F};
00036     m_viewMatrix[0][0] = u.x;
00037     m_viewMatrix[1][0] = u.y;
00038     m_viewMatrix[2][0] = u.z;
00039     m_viewMatrix[0][1] = v.x;
00040     m_viewMatrix[1][1] = v.y;
00041     m_viewMatrix[2][1] = v.z;
00042     m_viewMatrix[0][2] = w.x;
00043     m_viewMatrix[1][2] = w.y;
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;

```

```

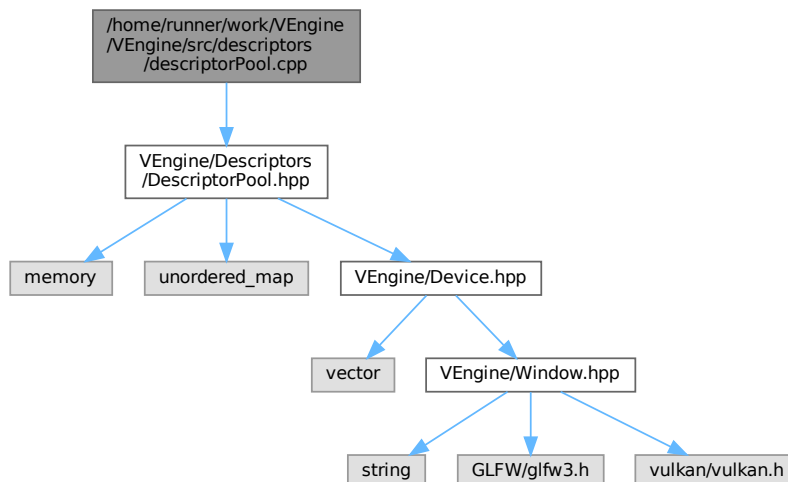
00081     m_viewMatrix[2][1] = v.z;
00082     m_viewMatrix[0][2] = w.x;
00083     m_viewMatrix[1][2] = w.y;
00084     m_viewMatrix[2][2] = w.z;
00085     m_viewMatrix[3][0] = -dot(u, position);
00086     m_viewMatrix[3][1] = -dot(v, position);
00087     m_viewMatrix[3][2] = -dot(w, position);
00088
00089     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 }

```

## 7.56 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference

#include "VEngine/Descriptors/DescriptorPool.hpp"

Include dependency graph for descriptorPool.cpp:



## 7.57 descriptorPool.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Descriptors/DescriptorPool.hpp"
00002
00003 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::addPoolSize(const VkDescriptorType
    descriptorType, const uint32_t count)
00004 {
00005     m_poolSizes.push_back({descriptorType, count});
00006     return *this;
00007 }
00008

```

```

00009 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setPoolFlags(const
    VkDescriptorPoolCreateFlags flags)
00010 {
00011     m_poolFlags = flags;
00012     return *this;
00013 }
00014 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setMaxSets(const uint32_t count)
00015 {
00016     m_maxSets = count;
00017     return *this;
00018 }
00019
00020 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device(device)
00021 {
00022     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00023     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00024     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00025     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00026     descriptorPoolInfo.maxSets = maxSets;
00027     descriptorPoolInfo.flags = poolFlags;
00028
00029     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) !=
00030         VK_SUCCESS) {
00031         throw std::runtime_error("failed to create descriptor pool!");
00032     }
00033 }
00034
00035 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorSetLayout,
    VkDescriptorSet &descriptor) const
00036 {
00037     VkDescriptorSetAllocateInfo allocInfo{};
00038     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00039     allocInfo.descriptorPool = m_descriptorPool;
00040     allocInfo.pSetLayouts = &descriptorSetLayout;
00041     allocInfo.descriptorSetCount = 1;
00042
00043     // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00044     // a new pool whenever an old pool fills up. But this is beyond our current scope
00045     return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00046 }

```

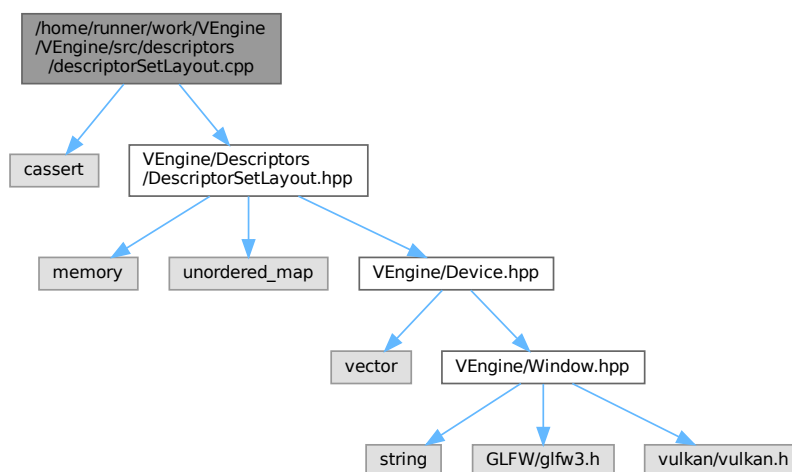
## 7.58 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

Include dependency graph for descriptorSetLayout.cpp:



## 7.59 descriptorSetLayout.cpp

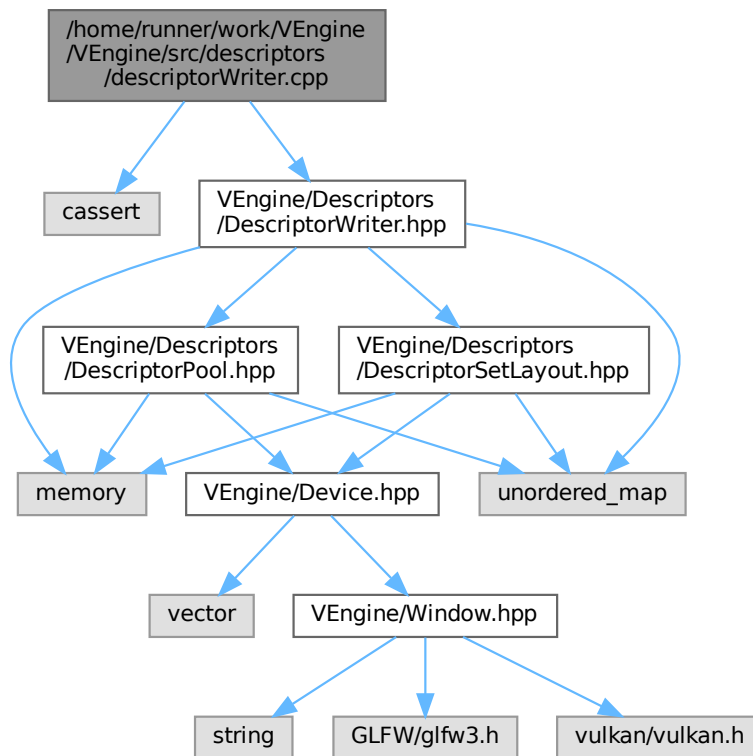
[Go to the documentation of this file.](#)

```
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 kv : bindings) {
00022         setLayoutBindings.push_back(kv.second);
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 }
```

## 7.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference

```
#include <cassert>
#include "VEngine/Descriptors/DescriptorWriter.hpp"
```

Include dependency graph for descriptorWriter.cpp:



## 7.61 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[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[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 {
00054     for (auto &write : m_writes) {
00055         write.dstSet = set;
00056     }
00057     vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
    m_writes.data(), 0, nullptr);
00058 }

```

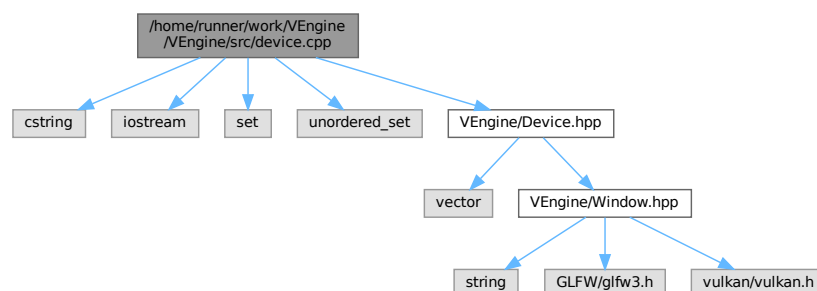
## 7.62 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

```

#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)



## 7.62.1 Function Documentation

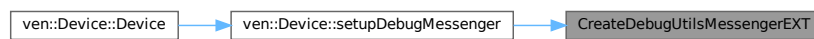
### 7.62.1.1 CreateDebugUtilsMessengerEXT()

```
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:



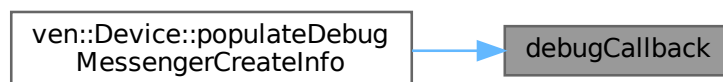
### 7.62.1.2 debugCallback()

```
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:



### 7.62.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerEXT debugMessenger,
    const VkAllocationCallbacks * pAllocator)
```

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

Referenced by [ven::Device::~~Device\(\)](#).

Here is the caller graph for this function:



## 7.63 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     auto func = reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkCreateDebugUtilsMessengerEXT"));
00019     if (func != nullptr) {
00020         return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00021     }
00022     return VK_ERROR_EXTENSION_NOT_PRESENT;
00023 }
00024
00025 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
debugMessenger, const VkAllocationCallbacks *pAllocator)
00026 {
00027     auto func = reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkDestroyDebugUtilsMessengerEXT"));
00028     if (func != nullptr) {
00029         func(instance, debugMessenger, pAllocator);
00030     }
00031 }
00032
00033
00034 ven::Device::Device(Window &window) : m_window{window}
00035 {
00036     createInstance();
00037     setupDebugMessenger();
00038     createSurface();
00039     pickPhysicalDevice();
```

```

00040     createLogicalDevice();
00041     createCommandPool();
00042 }
00043
00044 ven::Device::~Device()
00045 {
00046     vkDestroyCommandPool(m_device, m_commandPool, nullptr);
00047     vkDestroyDevice(m_device, nullptr);
00048
00049     if (enableValidationLayers) {
00050         DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
00051     }
00052
00053     vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
00054     vkDestroyInstance(m_instance, nullptr);
00055 }
00056
00057 void ven::Device::createInstance()
00058 {
00059     if (enableValidationLayers && !checkValidationLayerSupport()) {
00060         throw std::runtime_error("validation layers requested, but not available!");
00061     }
00062
00063     VkApplicationInfo appInfo = {};
00064     appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
00065     appInfo.pApplicationName = "LittleVulkanEngine App";
00066     appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
00067     appInfo.pEngineName = "No Engine";
00068     appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
00069     appInfo.apiVersion = VK_API_VERSION_1_0;
00070
00071     VkInstanceCreateInfo createInfo = {};
00072     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00073     createInfo.pApplicationInfo = &appInfo;
00074
00075     std::vector<const char*> extensions = getRequiredExtensions();
00076     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00077     createInfo.ppEnabledExtensionNames = extensions.data();
00078
00079     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00080     if (enableValidationLayers) {
00081         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00082         createInfo.ppEnabledLayerNames = validationLayers.data();
00083
00084         populateDebugMessengerCreateInfo(debugCreateInfo);
00085         createInfo.pNext = &debugCreateInfo;
00086     } else {
00087         createInfo.enabledLayerCount = 0;
00088         createInfo.pNext = nullptr;
00089     }
00090
00091     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
00092         throw std::runtime_error("failed to create instance!");
00093     }
00094
00095     hasGlfwRequiredInstanceExtensions();
00096 }
00097
00098 void ven::Device::pickPhysicalDevice()
00099 {
00100     uint32_t deviceCount = 0;
00101     vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
00102     if (deviceCount == 0) {
00103         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00104     }
00105     std::cout << "Device count: " << deviceCount << '\n';
00106     std::vector<VkPhysicalDevice> devices(deviceCount);
00107     vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());
00108
00109     for (const auto &device : devices) {
00110         if (isDeviceSuitable(device)) {
00111             m_physicalDevice = device;
00112             break;
00113         }
00114     }
00115
00116     if (m_physicalDevice == VK_NULL_HANDLE) {
00117         throw std::runtime_error("failed to find a suitable GPU!");
00118     }
00119
00120     vkGetPhysicalDeviceProperties(m_physicalDevice, &properties_);
00121     std::cout << "physical device: " << properties_.deviceName << '\n';
00122 }
00123
00124 void ven::Device::createLogicalDevice()
00125 {
00126     const QueueFamilyIndices indices = findQueueFamilies(m_physicalDevice);

```

```

00127
00128     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00129     const std::set<uint32_t> uniqueQueueFamilies = {indices.graphicsFamily, indices.presentFamily};
00130     float queuePriority = 1.0F;
00131
00132     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00133         VkDeviceQueueCreateInfo queueCreateInfo = {};
00134         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00135         queueCreateInfo.queueFamilyIndex = queueFamily;
00136         queueCreateInfo.queueCount = 1;
00137         queueCreateInfo.pQueuePriorities = &queuePriority;
00138         queueCreateInfos.push_back(queueCreateInfo);
00139     }
00140
00141     VkPhysicalDeviceFeatures deviceFeatures = {};
00142     deviceFeatures.samplerAnisotropy = VK_TRUE;
00143
00144     VkDeviceCreateInfo createInfo = {};
00145     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00146
00147     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00148     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00149
00150     createInfo.pEnabledFeatures = &deviceFeatures;
00151     createInfo.enabledExtensionCount = static_cast<uint32_t>(deviceExtensions.size());
00152     createInfo.ppEnabledExtensionNames = deviceExtensions.data();
00153
00154     // might not really be necessary anymore because device specific validation layers
00155     // have been deprecated
00156     if (enableValidationLayers) {
00157         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00158         createInfo.ppEnabledLayerNames = validationLayers.data();
00159     } else {
00160         createInfo.enabledLayerCount = 0;
00161     }
00162
00163     if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
00164         throw std::runtime_error("failed to create logical device!");
00165     }
00166
00167     vkGetDeviceQueue(m_device, indices.graphicsFamily, 0, &m_graphicsQueue);
00168     vkGetDeviceQueue(m_device, indices.presentFamily, 0, &m_presentQueue);
00169 }
00170
00171 void ven::Device::createCommandPool()
00172 {
00173     const QueueFamilyIndices queueFamilyIndices = findPhysicalQueueFamilies();
00174
00175     VkCommandPoolCreateInfo poolInfo = {};
00176     poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
00177     poolInfo.queueFamilyIndex = queueFamilyIndices.graphicsFamily;
00178     poolInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
00179         VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
00179
00180     if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
00181         throw std::runtime_error("failed to create command pool!");
00182     }
00183 }
00184
00185 bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00186 {
00187     const QueueFamilyIndices indices = findQueueFamilies(device);
00188     const bool extensionsSupported = checkDeviceExtensionSupport(device);
00189     bool swapChainAdequate = false;
00190
00191     if (extensionsSupported) {
00192         SwapChainSupportDetails swapChainSupport = querySwapChainSupport(device);
00193         swapChainAdequate = !swapChainSupport.formats.empty() &&
00194             !swapChainSupport.presentModes.empty();
00195     }
00196
00197     VkPhysicalDeviceFeatures supportedFeatures;
00198     vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00199
00200     return indices.isComplete() && extensionsSupported && swapChainAdequate &&
00201         (supportedFeatures.samplerAnisotropy != 0U);
00202 }
00203
00204 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00205 {
00206     createInfo = {};
00207     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00208     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
00209         VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00210     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
00211         VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
00212         VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;

```

```

00211     createInfo.pfnUserCallback = debugCallback;
00212     createInfo.pUserData = nullptr; // Optional
00213 }
00214
00215 void ven::Device::setupDebugMessenger()
00216 {
00217     if (!enableValidationLayers) { return; }
00218     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00219     populateDebugMessengerCreateInfo(createInfo);
00220     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
    VK_SUCCESS) {
00221         throw std::runtime_error("failed to set up debug messenger!");
00222     }
00223 }
00224
00225 bool ven::Device::checkValidationLayerSupport() const
00226 {
00227     uint32_t layerCount = 0;
00228     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00229
00230     std::vector<VkLayerProperties> availableLayers(layerCount);
00231     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00232
00233     for (const char *layerName : validationLayers) {
00234         bool layerFound = false;
00235
00236         for (const auto &layerProperties : availableLayers) {
00237             if (strcmp(layerName, layerProperties.layerName) == 0) {
00238                 layerFound = true;
00239                 break;
00240             }
00241         }
00242         if (!layerFound) {
00243             return false;
00244         }
00245     }
00246
00247     return true;
00248 }
00249
00250 std::vector<const char *> ven::Device::getRequiredExtensions() const
00251 {
00252     uint32_t glfwExtensionCount = 0;
00253     const char **glfwExtensions = nullptr;
00254     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00255
00256     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00257
00258     if (enableValidationLayers) {
00259         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00260     }
00261
00262     return extensions;
00263 }
00264
00265 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00266 {
00267     uint32_t extensionCount = 0;
00268     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
00269     std::vector<VkExtensionProperties> extensions(extensionCount);
00270     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00271
00272     std::cout << "available extensions:\n";
00273     std::unordered_set<std::string> available;
00274     for (const auto &extension : extensions) {
00275         std::cout << '\t' << extension.extensionName << '\n';
00276         available.insert(extension.extensionName);
00277     }
00278
00279     std::cout << "required extensions:\n";
00280     const std::vector<const char *> requiredExtensions = getRequiredExtensions();
00281     for (const auto &required : requiredExtensions) {
00282         std::cout << "\t" << required << '\n';
00283         if (available.find(required) == available.end()) {
00284             throw std::runtime_error("Missing required glfw extension");
00285         }
00286     }
00287 }
00288
00289 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00290 {
00291     uint32_t extensionCount = 0;
00292     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00293
00294     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00295     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
    availableExtensions.data());

```

```

00296
00297     std::set<std::string> requiredExtensions(deviceExtensions.begin(), deviceExtensions.end());
00298     for (const auto &extension : availableExtensions) {
00299         requiredExtensions.erase(extension.extensionName);
00300     }
00301
00302     return requiredExtensions.empty();
00303 }
00304
00305 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00306 {
00307     QueueFamilyIndices indices;
00308
00309     uint32_t queueFamilyCount = 0;
00310     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00311     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00312     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00313     uint32_t index = 0;
00314
00315     for (const auto &queueFamily : queueFamilies) {
00316         if (queueFamily.queueCount > 0 && ((queueFamily.queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00317             indices.graphicsFamily = index;
00318             indices.graphicsFamilyHasValue = true;
00319         }
00320         VkBool32 presentSupport = 0U;
00321         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00322         if (queueFamily.queueCount > 0 && (presentSupport != 0U)) {
00323             indices.presentFamily = index;
00324             indices.presentFamilyHasValue = true;
00325         }
00326         if (indices.isComplete()) {
00327             break;
00328         }
00329         index++;
00330     }
00331     return indices;
00332 }
00333
00334 ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00335 {
00336     SwapChainSupportDetails details;
00337     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
00338     uint32_t formatCount = 0;
00339
00340     vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
00341     if (formatCount != 0) {
00342         details.formats.resize(formatCount);
00343         vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
00344     }
00345     uint32_t presentModeCount = 0;
00346     vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
00347     if (presentModeCount != 0) {
00348         details.presentModes.resize(presentModeCount);
00349         vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount,
00350         details.presentModes.data());
00351     }
00352     return details;
00353 }
00354
00355 VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
00356 tiling, const VkFormatFeatureFlags features) const
00357 {
00358     for (const VkFormat format : candidates) {
00359         VkFormatProperties props;
00360         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00361         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00362             return format;
00363         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
00364         features) {
00365             return format;
00366         }
00367     }
00368     throw std::runtime_error("failed to find supported format!");
00369 }
00370
00371 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
00372 properties) const
00373 {
00374     VkPhysicalDeviceMemoryProperties memProperties;
00375     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
00376
00377     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00378         if (((typeFilter & (1 << i)) != 0U) &&
00379         (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00380             return i;
00381         }
00382     }

```

```

00379     }
00380
00381     throw std::runtime_error("failed to find suitable m_memory type!");
00382 }
00383
00384 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
VkMemoryPropertyFlags propertiesp, 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, propertiesp);
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
00412 {
00413     VkCommandBufferAllocateInfo allocInfo{};
00414     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00415     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00416     allocInfo.commandPool = m_commandPool;
00417     allocInfo.commandBufferCount = 1;
00418
00419     VkCommandBuffer commandBuffer = nullptr;
00420     vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);
00421
00422     VkCommandBufferBeginInfo beginInfo{};
00423     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00424     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00425
00426     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00427     return commandBuffer;
00428 }
00429
00430 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00431 {
00432     vkEndCommandBuffer(commandBuffer);
00433
00434     VkSubmitInfo submitInfo{};
00435     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00436     submitInfo.commandBufferCount = 1;
00437     submitInfo.pCommandBuffers = &commandBuffer;
00438
00439     vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
00440     vkQueueWaitIdle(m_graphicsQueue);
00441
00442     vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
00443 }
00444
00445 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
size) const
00446 {
00447     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00448
00449     VkBufferCopy copyRegion{};
00450     copyRegion.srcOffset = 0; // Optional
00451     copyRegion.dstOffset = 0; // Optional
00452     copyRegion.size = size;
00453     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00454
00455     endSingleTimeCommands(commandBuffer);
00456 }
00457
00458 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
const uint32_t height, const uint32_t layerCount) const
00459 {
00460     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00461
00462     VkBufferImageCopy region{};

```

```

00463     region.bufferOffset = 0;
00464     region.bufferRowLength = 0;
00465     region.bufferImageHeight = 0;
00466
00467     region.imageSubresource.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00468     region.imageSubresource.mipLevel = 0;
00469     region.imageSubresource.baseArrayLayer = 0;
00470     region.imageSubresource.layerCount = layerCount;
00471
00472     region.imageOffset = {0, 0, 0};
00473     region.imageExtent = {width, height, 1};
00474
00475     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
00476     &region);
00477     endSingleTimeCommands(commandBuffer);
00478 }
00479
00479 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const
00480 {
00481     if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00482         throw std::runtime_error("failed to create image!");
00483     }
00484
00485     VkMemoryRequirements memRequirements;
00486     vkGetImageMemoryRequirements(m_device, image, &memRequirements);
00487
00488     VkMemoryAllocateInfo allocInfo{};
00489     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00490     allocInfo.allocationSize = memRequirements.size;
00491     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00492
00493     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00494         throw std::runtime_error("failed to allocate image m_memory!");
00495     }
00496
00497     if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
00498         throw std::runtime_error("failed to bind image m_memory!");
00499     }
00500 }

```

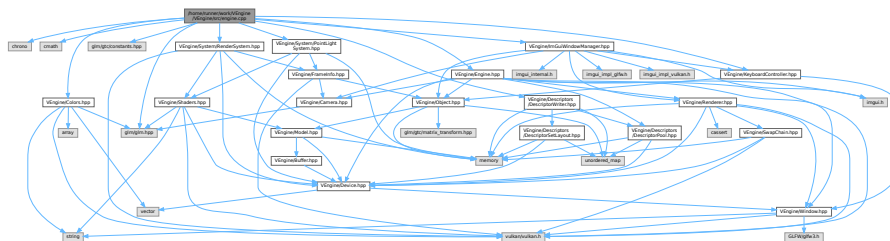
## 7.64 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference

```

#include <chrono>
#include <cmath>
#include <glm/glm.hpp>
#include <glm/gtc/constants.hpp>
#include "VEngine/Engine.hpp"
#include "VEngine/KeyboardController.hpp"
#include "VEngine/System/RenderSystem.hpp"
#include "VEngine/System/PointLightSystem.hpp"
#include "VEngine/Descriptors/DescriptorWriter.hpp"
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"

```

Include dependency graph for engine.cpp:





## 7.65 engine.cpp

[Go to the documentation of this file.](#)

```

00001 #include <chrono>
00002 #include <cmath>
00003
00004 #include <glm/glm.hpp>
00005 #include <glm/gtc/constants.hpp>
00006
00007 #include "VEngine/Engine.hpp"
00008 #include "VEngine/KeyboardController.hpp"
00009 #include "VEngine/System/RenderSystem.hpp"
00010 #include "VEngine/System/PointLightSystem.hpp"
00011 #include "VEngine/Descriptors/DescriptorWriter.hpp"
00012 #include "VEngine/ImGuiWindowManager.hpp"
00013 #include "VEngine/Colors.hpp"
00014
00015 ven::Engine::Engine(const uint32_t width, const uint32_t height, const std::string &title) :
    m_window(width, height, title)
00016 {
00017     createInstance();
00018     createSurface();
00019     ImGuiWindowManager::init(m_window.getGLFWWindow(), m_instance, &m_device,
    m_renderer.getSwapChainRenderPass());
00020     m_globalPool =
    DescriptorPool::Builder(m_device).setMaxSets(SwapChain::MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    SwapChain::MAX_FRAMES_IN_FLIGHT).build();
00021     loadObjects();
00022 }
00023
00024 void ven::Engine::createInstance()
00025 {
00026     uint32_t glfwExtensionCount = 0;
00027     const char** glfwExtensions = nullptr;
00028     VkInstanceCreateInfo createInfo{};
00029     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 };
00030
00031     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00032     createInfo.pApplicationInfo = &appInfo;
00033     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00034     createInfo.enabledExtensionCount = glfwExtensionCount;
00035     createInfo.ppEnabledExtensionNames = glfwExtensions;
00036
00037     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS)
00038     {
00039         throw std::runtime_error("Failed to create Vulkan instance");
00040     }
00041 }
00042
00043 void ven::Engine::loadObjects()
00044 {
00045     std::shared_ptr model = Model::createModelFromFile(m_device, "models/flat_vase.obj");
00046
00047     Object flatVase = Object::createObject();
00048     flatVase.name = "flat vase";
00049     flatVase.model = model;
00050     flatVase.transform3D.translation = {-0.5F, 0.5F, 0.F};
00051     flatVase.transform3D.scale = {3.F, 1.5F, 3.F};
00052     m_objects.emplace(flatVase.getId(), std::move(flatVase));
00053
00054     model = Model::createModelFromFile(m_device, "models/smooth_vase.obj");
00055     Object smoothVase = Object::createObject();
00056     smoothVase.name = "smooth vase";
00057     smoothVase.model = model;
00058     smoothVase.transform3D.translation = {0.5F, 0.5F, 0.F};
00059     smoothVase.transform3D.scale = {3.F, 1.5F, 3.F};
00060     m_objects.emplace(smoothVase.getId(), std::move(smoothVase));
00061
00062     model = Model::createModelFromFile(m_device, "models/quad.obj");
00063     Object floor = Object::createObject();
00064     floor.name = "floor";
00065     floor.model = model;
00066     floor.transform3D.translation = {0.F, 0.5F, 0.F};
00067     floor.transform3D.scale = {3.F, 1.F, 3.F};
00068     m_objects.emplace(floor.getId(), std::move(floor));
00069
00070     const std::vector<glm::vec3> lightColors{
00071         {Colors::RED},
00072         {Colors::GREEN},
00073         {Colors::BLUE},
00074         {Colors::YELLOW},
00075         {Colors::CYAN},
00076         {Colors::MAGENTA}
    }

```

```

00077     };
00078
00079     for (std::size_t i = 0; i < lightColors.size(); i++)
00080     {
00081         Object pointLight = Object::makePointLight();
00082         pointLight.name = "point light " + std::to_string(i);
00083         pointLight.color = lightColors[i];
00084         auto 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});
00085         pointLight.transform3D.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F,
1.F));
00086         m_objects.emplace(pointLight.getId(), std::move(pointLight));
00087     }
00088 }
00089
00090 void ven::Engine::mainLoop()
00091 {
00092     GlobalUbo ubo{};
00093     Camera camera{};
00094     KeyboardController cameraController{};
00095     std::chrono::duration<float> deltaTime{};
00096     VkCommandBuffer_T *commandBuffer = nullptr;
00097     bool showDebugWindow = true;
00098     float frameTime = NAN;
00099     int frameIndex = 0;
00100     Object viewerObject = Object::createObject();
00101     std::chrono::time_point<std::chrono::system_clock> newTime;
00102     std::chrono::time_point<std::chrono::system_clock> currentTime =
std::chrono::high_resolution_clock::now();
00103     std::unique_ptr<DescriptorSetLayout> globalSetLayout =
DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
VK_SHADER_STAGE_ALL_GRAPHICS).build();
00104     std::vector<std::unique_ptr<Buffer>> uboBuffers(SwapChain::MAX_FRAMES_IN_FLIGHT);
00105     std::vector<VkDescriptorSet> globalDescriptorSets(SwapChain::MAX_FRAMES_IN_FLIGHT);
00106     RenderSystem renderSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorSetLayout());
00107     PointLightSystem pointLightSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorSetLayout());
00108     ImGuiIO &io = ImGui::GetIO();
00109     VkDescriptorBufferInfo bufferInfo{};
00110
00111     io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard | ImGuiConfigFlags_NavEnableGamepad;
00112
00113     for (auto& uboBuffer : uboBuffers)
00114     {
00115         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00116         uboBuffer->map();
00117     }
00118     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00119         bufferInfo = uboBuffers[i]->descriptorInfo();
00120         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
&bufferInfo).build(globalDescriptorSets[i]);
00121     }
00122     camera.setViewTarget(glm::vec3(-1.F, -2.F, -2.F), glm::vec3(0.F, 0.F, 2.5F));
00123     viewerObject.transform3D.translation.z = DEFAULT_POSITION[2];
00124
00125     m_renderer.setClearColor();
00126
00127     while (glfwWindowShouldClose(m_window.getGLFWWindow()) == 0)
00128     {
00129         glfwPollEvents();
00130
00131         newTime = std::chrono::high_resolution_clock::now();
00132         deltaTime = newTime - currentTime;
00133         currentTime = newTime;
00134         frameTime = deltaTime.count();
00135         commandBuffer = m_renderer.beginFrame();
00136
00137         cameraController.moveInPlaneXZ(m_window.getGLFWWindow(), frameTime, viewerObject,
&showDebugWindow);
00138         camera.setViewXYZ(viewerObject.transform3D.translation, viewerObject.transform3D.rotation);
00139         camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00140
00141         if (commandBuffer != nullptr) {
00142             frameIndex = m_renderer.getFrameIndex();
00143             FrameInfo frameInfo{frameIndex, frameTime, commandBuffer, camera,
globalDescriptorSets[static_cast<unsigned long>(frameIndex)], m_objects};
00144             ubo.projection = camera.getProjection();
00145             ubo.view = camera.getView();
00146             ubo.inverseView = camera.getInverseView();
00147             PointLightSystem::update(frameInfo, ubo);
00148             uboBuffers[static_cast<unsigned long>(frameIndex)]->writeToBuffer(&ubo);
00149             uboBuffers[static_cast<unsigned long>(frameIndex)]->flush();
00150
00151             m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00152             renderSystem.renderObjects(frameInfo);

```

```

00153         pointLightSystem.render(frameInfo);
00154
00155         if (showDebugWindow) { ImGuiWindowManager::render(&m_renderer, m_objects, io,
viewerObject, camera, cameraController, m_device.getPhysicalDevice()); }
00156
00157         m_renderer.endSwapChainRenderPass(commandBuffer);
00158         m_renderer.endFrame();
00159         commandBuffer = nullptr;
00160     }
00161 }
00162 ImGuiWindowManager::cleanup();
00163 vkDeviceWaitIdle(m_device.device());
00164 }

```

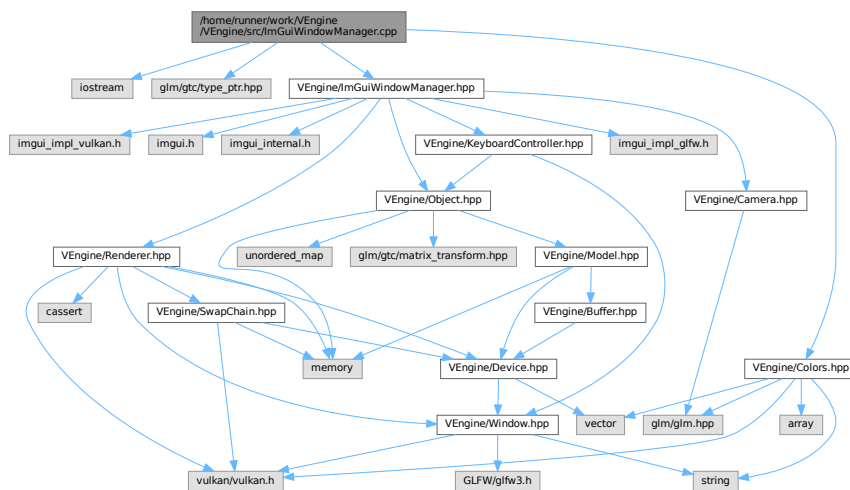
## 7.66 /home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp File Reference

```

#include <iostream>
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"

```

Include dependency graph for ImGuiWindowManager.cpp:



## 7.67 ImGuiWindowManager.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002
00003 #include <glm/gtc/type_ptr.hpp>
00004
00005 #include "VEngine/ImGuiWindowManager.hpp"
00006 #include "VEngine/Colors.hpp"
00007
00008 void ven::ImGuiWindowManager::init(GLFWwindow* window, VkInstance instance, Device* device,
VkRenderPass renderPass)
00009 {
00010     VkDescriptorPool pool = nullptr;
00011
00012     ImGui::CreateContext();
00013     // ImGui::StyleColorsDark();
00014     VkDescriptorPoolSize pool_sizes[] = {
00015         { VK_DESCRIPTOR_TYPE_SAMPLER, 1000 },

```

```

00016         { VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1000 },
00017         { VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, 1000 },
00018         { VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, 1000 },
00019         { VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, 1000 },
00020         { VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, 1000 },
00021         { VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1000 },
00022         { VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, 1000 },
00023         { VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, 1000 },
00024         { VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, 1000 },
00025         { VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, 1000 }
00026     };
00027     VkDescriptorPoolCreateInfo pool_info = {
00028         VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00029         nullptr,
00030         VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00031         1000,
00032         std::size(pool_sizes),
00033         pool_sizes
00034     };
00035
00036     if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00037         throw std::runtime_error("Failed to create ImGui descriptor pool");
00038     }
00039     ImGui_ImplVulkan_InitInfo init_info = {
00040         .Instance = instance,
00041         .PhysicalDevice = device->getPhysicalDevice(),
00042         .Device = device->device(),
00043         .Queue = device->graphicsQueue(),
00044         .DescriptorPool = pool,
00045         .MinImageCount = 3,
00046         .ImageCount = 3,
00047         .MSAASamples = VK_SAMPLE_COUNT_1_BIT
00048     };
00049
00050     ImGui_ImplGlfw_InitForVulkan(window, true);
00051     ImGui_ImplVulkan_Init(&init_info, renderPass);
00052 }
00053
00054 void ven::ImGuiWindowManager::render(Renderer* renderer, std::unordered_map<unsigned int, Object>&
objects, ImGuiIO& io, Object& cameraObj, Camera& camera, KeyboardController& cameraController,
VkPhysicalDevice physicalDevice)
00055 {
00056     const float framerate = io.Framerate;
00057     VkPhysicalDeviceProperties deviceProperties;
00058     vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
00059
00060     ImGui_ImplVulkan_NewFrame();
00061     ImGui_ImplGlfw_NewFrame();
00062     ImGui::NewFrame();
00063
00064     ImGui::Begin("Application Info");
00065     ImGui::Text("FPS: %.1f", framerate);
00066     ImGui::Text("Frame time: %.3fms", 1000.0f / framerate);
00067     ImGui::End();
00068
00069     ImGui::Begin("Debug Window");
00070
00071     if (ImGui::CollapsingHeader("Camera")) {
00072         float fov = camera.getFov();
00073         float near = camera.getNear();
00074         float far = camera.getFar();
00075         if (ImGui::BeginTable("CameraTable", 2)) {
00076             ImGui::TableNextColumn();
00077             ImGui::DragFloat3("Position", glm::value_ptr(cameraObj.transform3D.translation), 0.1F);
00078             ImGui::TableNextColumn();
00079             if (ImGui::Button("Reset##position")) { cameraObj.transform3D.translation =
DEFAULT_POSITION; }
00080
00081             ImGui::TableNextColumn();
00082             ImGui::DragFloat3("Rotation", glm::value_ptr(cameraObj.transform3D.rotation), 0.1F);
00083             ImGui::TableNextColumn();
00084             if (ImGui::Button("Reset##rotation")) { cameraObj.transform3D.rotation = DEFAULT_ROTATION; }
00085         }
00086         ImGui::TableNextColumn();
00087         if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
camera.setFov(fov); }
00088         ImGui::TableNextColumn();
00089         if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }
00090
00091         ImGui::TableNextColumn();
00092         if (ImGui::SliderFloat("Near", &near, 0.001F, 10.0F)) { camera.setNear(near); }
00093         ImGui::TableNextColumn();
00094         if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }
00095
00096         ImGui::TableNextColumn();
00097         if (ImGui::SliderFloat("Far", &far, 1.F, 1000.0F)) { camera.setFar(far); }

```

```

00098         ImGui::TableNextColumn();
00099         if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }
00100
00101         ImGui::TableNextColumn();
00102         ImGui::SliderFloat("Move Speed", &cameraController.m_moveSpeed, 0.1F, 10.0F);
00103         ImGui::TableNextColumn();
00104         if (ImGui::Button("Reset##moveSpeed")) { cameraController.m_moveSpeed =
DEFAULT_MOVE_SPEED; }
00105
00106         ImGui::TableNextColumn();
00107         ImGui::SliderFloat("Look Speed", &cameraController.m_lookSpeed, 0.1F, 10.0F);
00108         ImGui::TableNextColumn();
00109         if (ImGui::Button("Reset##lookSpeed")) { cameraController.m_lookSpeed =
DEFAULT_LOOK_SPEED; }
00110
00111         ImGui::EndTable();
00112     }
00113 }
00114
00115     if (ImGui::CollapsingHeader("Input")) {
00116         ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io.MousePos.x, io.MousePos.y) :
ImGui::Text("Mouse pos: <INVALID>");
00117         ImGui::Text("Mouse delta: (%g, %g)", io.MouseDelta.x, io.MouseDelta.y);
00118         ImGui::Text("Mouse down:");
00119         for (int i = 0; i < static_cast<int>(std::size(io.MouseDown)); i++) {
00120             if (ImGui::IsMouseDown(i)) {
00121                 ImGui::SameLine();
00122                 ImGui::Text("%b%d (%.02f secs)", i, io.MouseDownDuration[i]);
00123             }
00124         }
00125         ImGui::Text("Mouse wheel: %.1f", io.MouseWheel);
00126         ImGui::Text("Keys down:");
00127         for (ImGuiKey key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
static_cast<ImGuiKey>(key + 1)) {
00128             if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
00129             ImGui::SameLine();
00130             ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\\%s\\" : "\\%s\\ %d",
ImGui::GetKeyName(key), key);
00131         }
00132     }
00133
00134     if (ImGui::CollapsingHeader("Render Settings")) {
00135         ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());
00136
00137         if (ImGui::BeginTable("ClearColorTable", 2)) {
00138             ImGui::TableNextColumn();
00139             std::array<float, 4> clearColor = renderer->getClearColor();
00140
00141             if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
00142                 VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1], clearColor[2],
clearColor[3]}};
00143                 renderer->setClearColorValue(clearColorValue);
00144             }
00145
00146             ImGui::TableNextColumn();
00147             static int item_current = 0;
00148
00149             if (ImGui::Combo("Color Presets",
&item_current,
[] (void*, int idx, const char** out_text) -> bool {
00150                 if (idx < 0 || idx >=
static_cast<int>(std::size(Colors::CLEAR_COLORS))) { return false; }
00151                 *out_text = Colors::CLEAR_COLORS.at(static_cast<unsigned
long>(idx)).first;
00152                 return true;
00153             },
nullptr,
std::size(Colors::CLEAR_COLORS))) {
00154                 renderer->setClearColorValue(Colors::CLEAR_COLORS.at(static_cast<unsigned
long>(item_current)).second);
00155             }
00156             ImGui::EndTable();
00157         }
00158     }
00159
00160     if (ImGui::CollapsingHeader("Device Properties")) {
00161         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00162             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00163             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion));
00164             ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00165             ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00166             ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00167         }
00168     }
00169 }
00170
00171

```

```

00172         ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00173         ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00174         ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantsSize);
00175         ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00176         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00177         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",
deviceProperties.limits.maxImageDimension2D);
00178         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00179         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00180         ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00181         ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00182         ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00183         ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00184         ImGui::EndTable();
00185     }
00186 }
00187
00188     if (ImGui::CollapsingHeader("Objects")) {
00189         ImVec4 color;
00190         bool open = false;
00191
00192         for (auto& [id, object] : objects) {
00193             if (object.color.r == 0.0F && object.color.g == 0.0F && object.color.b == 0.0F) {
00194                 color = { Colors::GRAY.r, Colors::GRAY.g, Colors::GRAY.b, 1.0F };
00195             } else {
00196                 color = { object.color.r, object.color.g, object.color.b, 1.0F };
00197             }
00198             ImGui::PushStyleColor(ImGuiCol_Text, color);
00199             open = ImGui::TreeNode(std::string(object.name + " [" + std::to_string(object.getId()) +
"]").c_str());
00200             ImGui::PopStyleColor(1);
00201             if (open) {
00202                 ImGui::Text("Address: %p", &object);
00203                 ImGui::DragFloat3(("Position##" + object.name).c_str(),
glm::value_ptr(object.transform3D.translation), 0.1F);
00204                 ImGui::DragFloat3(("Rotation##" + object.name).c_str(),
glm::value_ptr(object.transform3D.rotation), 0.1F);
00205                 ImGui::DragFloat3(("Scale##" + object.name).c_str(),
glm::value_ptr(object.transform3D.scale), 0.1F);
00206                 if (ImGui::BeginTable("ColorTable", 2)) {
00207                     ImGui::TableNextColumn(); ImGui::ColorEdit3(("Color##" + object.name).c_str(),
glm::value_ptr(object.color));
00208
00209                     ImGui::TableNextColumn();
00210                     static int item_current = 0;
00211                     if (ImGui::Combo("Color Presets",
&item_current,
00212                         [](void*, int idx, const char** out_text) -> bool {
00213                             if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLORS))) { return
false; }
00214
00215                             *out_text = Colors::COLORS.at(static_cast<unsigned long>(idx)).first;
00216                             return true;
00217                         },
00218                         nullptr,
00219                         std::size(Colors::COLORS))) {
00220                         object.color = Colors::COLORS.at(static_cast<unsigned
long>(item_current)).second;
00221                     }
00222                 }
00223                 ImGui::EndTable();
00224             }
00225             if (object.pointLight != nullptr) {
00226                 ImGui::SliderFloat(("Intensity##" + object.name).c_str(),
&object.pointLight->lightIntensity, 0.0F, 10.0F);
00227             }
00228             ImGui::TreePop();
00229         }
00230     }
00231 }
00232
00233 ImGui::End();
00234 ImGui::Render();
00235 ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
00236 }
00237
00238 void ven::ImGuiWindowManager::cleanup()
00239 {

```

```

00240     ImGui_ImplVulkan_Shutdown();
00241     ImGui_ImplGlfw_Shutdown();
00242     ImGui::DestroyContext();
00243 }

```

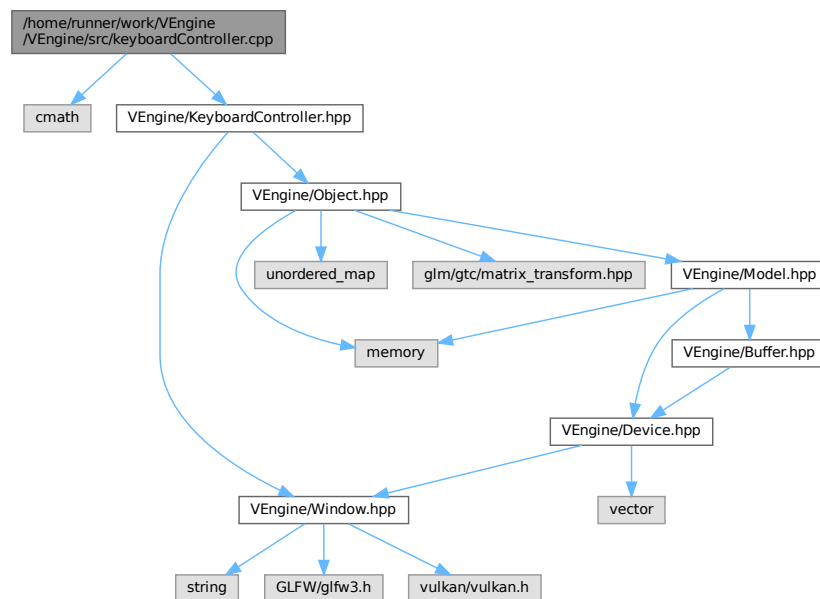
## 7.68 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp File Reference

```

#include <cmath>
#include "VEngine/KeyboardController.hpp"

```

Include dependency graph for keyboardController.cpp:



## 7.69 keyboardController.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cmath>
00002
00003 #include "VEngine/KeyboardController.hpp"
00004
00005 void ven::KeyboardController::moveInPlaneXZ(GLFWwindow* window, float dt, Object& object, bool*
showDebugWindow) const
00006 {
00007     glm::vec3 rotate{0};
00008     if (glfwGetKey(window, m_keys.lookLeft) == GLFW_PRESS) { rotate.y -= 1.F; }
00009     if (glfwGetKey(window, m_keys.lookRight) == GLFW_PRESS) { rotate.y += 1.F; }
00010     if (glfwGetKey(window, m_keys.lookUp) == GLFW_PRESS) { rotate.x += 1.F; }
00011     if (glfwGetKey(window, m_keys.lookDown) == GLFW_PRESS) { rotate.x -= 1.F; }
00012
00013     if (dot(rotate, rotate) > std::numeric_limits<float>::epsilon()) {
00014         object.transform3D.rotation += m_lookSpeed * dt * normalize(rotate);
00015     }
00016
00017     object.transform3D.rotation.x = glm::clamp(object.transform3D.rotation.x, -1.5F, 1.5F);
00018     object.transform3D.rotation.y = glm::mod(object.transform3D.rotation.y, glm::two_pi<float>());
00019
00020     float yaw = object.transform3D.rotation.y;
00021     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};

```

```

00022     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00023     constexpr glm::vec3 upDir{0.F, -1.F, 0.F};
00024
00025     glm::vec3 moveDir{0.F};
00026     if (glfwGetKey(window, m_keys.moveForward) == GLFW_PRESS) {moveDir += forwardDir;}
00027     if (glfwGetKey(window, m_keys.moveBackward) == GLFW_PRESS) {moveDir -= forwardDir;}
00028     if (glfwGetKey(window, m_keys.moveRight) == GLFW_PRESS) {moveDir += rightDir;}
00029     if (glfwGetKey(window, m_keys.moveLeft) == GLFW_PRESS) {moveDir -= rightDir;}
00030     if (glfwGetKey(window, m_keys.moveUp) == GLFW_PRESS) {moveDir += upDir;}
00031     if (glfwGetKey(window, m_keys.moveDown) == GLFW_PRESS) {moveDir -= upDir;}
00032
00033     if (dot(moveDir, moveDir) > std::numeric_limits<float>::epsilon()) {
00034         object.transform3D.translation += m_moveSpeed * dt * normalize(moveDir);
00035     }
00036
00037     // ImGui debug window
00038     if (glfwGetKey(window, GLFW_KEY_F1) == GLFW_PRESS) {
00039         *showDebugWindow = !*showDebugWindow;
00040     }
00041 }

```

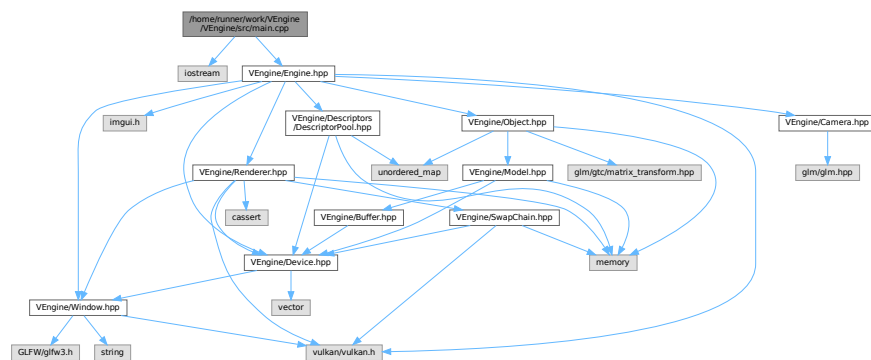
## 7.70 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```

#include <iostream>
#include "VEngine/Engine.hpp"

```

Include dependency graph for main.cpp:



### Functions

- `int main ()`

### 7.70.1 Function Documentation

#### 7.70.1.1 main()

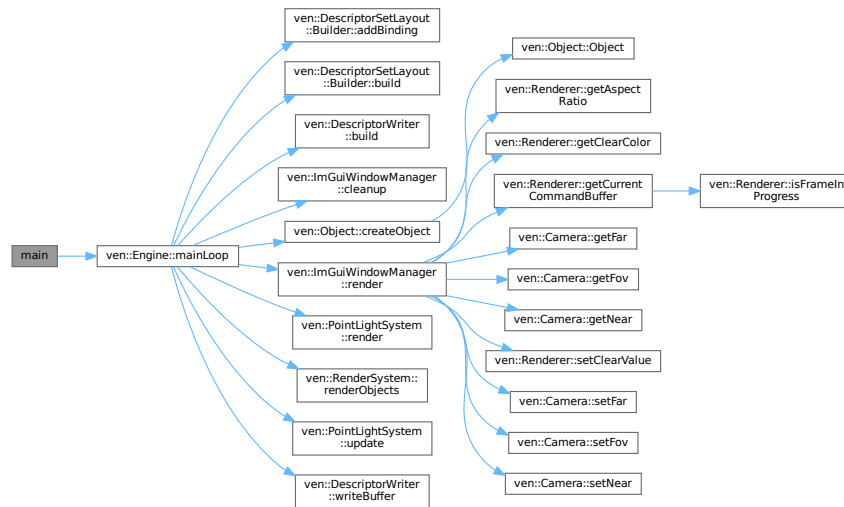
```
int main ()
```

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

References [ven::Engine::mainLoop\(\)](#).



Here is the call graph for this function:



## 7.71 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 }

```

## 7.72 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference

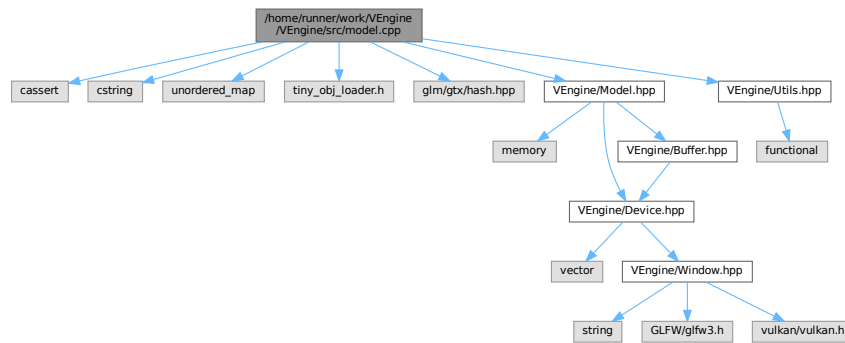
```

#include <cassert>
#include <cstring>
#include <unordered_map>
#include <tiny_obj_loader.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Model.hpp"

```

```
#include "VEngine/Utils.hpp"
```

Include dependency graph for model.cpp:



## Classes

- struct [std::hash< ven::Model::Vertex >](#)

## Namespaces

- namespace [std](#)  
*STL namespace.*

## Macros

- [#define TINYOBJLOADER\\_IMPLEMENTATION](#)

## 7.72.1 Macro Definition Documentation

### 7.72.1.1 TINYOBJLOADER\_IMPLEMENTATION

```
#define TINYOBJLOADER_IMPLEMENTATION
```

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

## 7.73 model.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <cstring>
00003 #include <unordered_map>
00004
00005 #define TINYOBJLOADER_IMPLEMENTATION
00006 #include <tiny_obj_loader.h>
00007
00008 #include <glm/gtx/hash.hpp>
00009
00010 #include "VEngine/Model.hpp"
00011 #include "VEngine/Utils.hpp"
00012
00013 namespace std {
00014     template<>
00015     struct hash<ven::Model::Vertex> {
00016         size_t operator()(ven::Model::Vertex const &vertex) const {
00017             size_t seed = 0;
00018             ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00019             return seed;
00020         }
00021     };
00022 }
00023
00024 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
    m_indexCount(0)
00025 {
00026     createVertexBuffer(builder.vertices);
00027     createIndexBuffer(builder.indices);
00028 }
00029
00030 ven::Model::~~Model() = default;
00031
00032 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00033 {
00034     m_vertexCount = static_cast<uint32_t>(vertices.size());
00035     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00036     const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
00037     uint32_t vertexSize = sizeof(vertices[0]);
00038
00039     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};
00040
00041     stagingBuffer.map();
00042     stagingBuffer.writeToBuffer(vertices.data());
00043
00044     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);
00045
00046     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00047 }
00048
00049 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00050 {
00051     m_indexCount = static_cast<uint32_t>(indices.size());
00052     m_hasIndexBuffer = m_indexCount > 0;
00053
00054     if (!m_hasIndexBuffer) {
00055         return;
00056     }
00057
00058     const VkDeviceSize bufferSize = sizeof(indices[0]) * m_indexCount;
00059     uint32_t indexSize = sizeof(indices[0]);
00060
00061     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};
00062
00063     stagingBuffer.map();
00064     stagingBuffer.writeToBuffer(indices.data());
00065
00066     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);
00067
00068     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), bufferSize);
00069 }
00070
00071 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00072 {
00073     if (m_hasIndexBuffer) {
00074         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00075     } else {

```

```

00076         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00077     }
00078 }
00079
00080 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00081 {
00082     const VkBuffer buffers[] = {m_vertexBuffer->getBuffer()};
00083     constexpr VkDeviceSize offsets[] = {0};
00084     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers, offsets);
00085
00086     if (m_hasIndexBuffer) {
00087         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00088     }
00089 }
00090
00091 std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
&filename)
00092 {
00093     Builder builder{};
00094     builder.loadModel(filename);
00095     return std::make_unique<Model>(device, builder);
00096 }
00097
00098 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00099 {
00100     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00101     bindingDescriptions[0].binding = 0;
00102     bindingDescriptions[0].stride = sizeof(Vertex);
00103     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00104     return bindingDescriptions;
00105 }
00106
00107 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00108 {
00109     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00110
00111     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00112     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00113     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00114     attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00115
00116     return attributeDescriptions;
00117 }
00118
00119 void ven::Model::Builder::loadModel(const std::string &filename)
00120 {
00121     tinyobj::attrib_t attrib;
00122     std::vector<tinyobj::shape_t> shapes;
00123     std::vector<tinyobj::material_t> materials;
00124     std::string warn;
00125     std::string err;
00126
00127     if (!LoadObj(&attrib, &shapes, &materials, &warn, &err, filename.c_str()))
00128     {
00129         throw std::runtime_error(warn + err);
00130     }
00131
00132     vertices.clear();
00133     indices.clear();
00134
00135     std::unordered_map<Vertex, uint32_t> uniqueVertices{};
00136     for (const auto &shape : shapes) {
00137         for (const auto &index : shape.mesh.indices) {
00138             Vertex vertex{};
00139             if (index.vertex_index >= 0) {
00140                 vertex.position = {
00141                     attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 0],
00142                     attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 1],
00143                     attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 2]
00144                 };
00145
00146                 vertex.color = {
00147                     attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 0],
00148                     attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 1],
00149                     attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 2]
00150                 };
00151             }
00152
00153             if (index.normal_index >= 0) {
00154                 vertex.normal = {
00155                     attrib.normals[3 * static_cast<size_t>(index.normal_index) + 0],
00156                     attrib.normals[3 * static_cast<size_t>(index.normal_index) + 1],
00157                     attrib.normals[3 * static_cast<size_t>(index.normal_index) + 2]
00158                 };
00159             }
00160
00161             if (index.texcoord_index >= 0) {

```

```

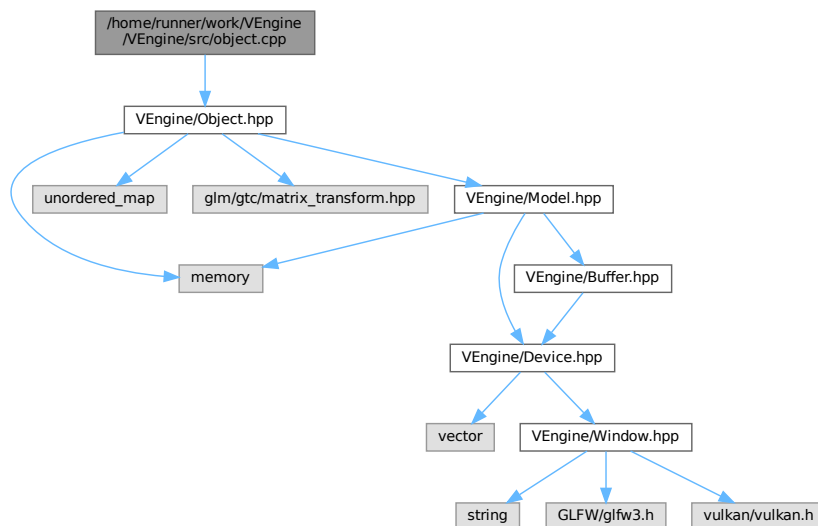
00162         vertex.uv = {
00163             attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 0],
00164             attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 1]
00165         };
00166     }
00167
00168     if (!uniqueVertices.contains(vertex)) {
00169         uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00170         vertices.push_back(vertex);
00171     }
00172     indices.push_back(uniqueVertices[vertex]);
00173 }
00174 }
00175 }

```

## 7.74 /home/runner/work/VEngine/VEngine/src/object.cpp File Reference

#include "VEngine/Object.hpp"

Include dependency graph for object.cpp:



## 7.75 object.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Object.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),

```

```

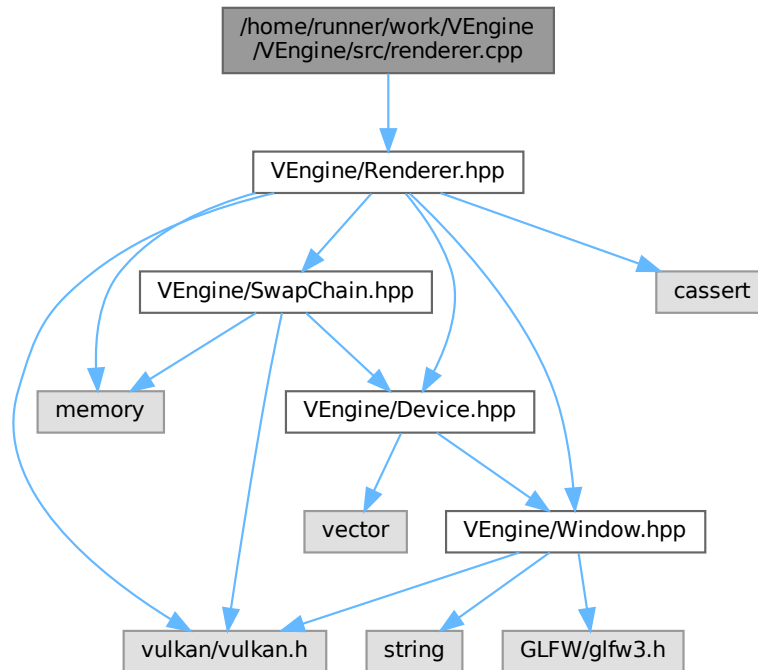
00020         scale.y * (c1 * c3 * s2 + s1 * s3),
00021         0.0F,
00022     },
00023     {
00024         scale.z * (c2 * s1),
00025         scale.z * (-s2),
00026         scale.z * (c1 * c2),
00027         0.0F,
00028     },
00029     {
00030         translation.x,
00031         translation.y,
00032         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 }
00066
00067 ven::Object ven::Object::makePointLight(const float intensity, const float radius, const glm::vec3
    color)
00068 {
00069     Object obj = Object::createObject();
00070     obj.color = color;
00071     obj.transform3D.scale.x = radius;
00072     obj.pointLight = std::make_unique<PointLightComponent>();
00073     obj.pointLight->lightIntensity = intensity;
00074     return obj;
00075 }

```

## 7.76 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

```
#include "VEngine/Renderer.hpp"
```

Include dependency graph for renderer.cpp:



## 7.77 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(SwapChain::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     VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer, &m_currentImageIndex);
00078     if (result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
00079         m_window.wasWindowResized()) {
00079         m_window.resetWindowResizedFlag();
00080         recreateSwapChain();
00081     }
00082     else if (result != VK_SUCCESS) {
00083         throw std::runtime_error("Failed to submit command buffer");
00084     }
00085
00086     m_isFrameStarted = false;
00087     m_currentFrameIndex = (m_currentFrameIndex + 1) % SwapChain::MAX_FRAMES_IN_FLIGHT;
00088 }
00089
00090 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer)
00091 {
00092     assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");
00093     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
00094         from a different frame");
00095
00096     VkRenderPassBeginInfo renderPassInfo{};
00097     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00098     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00099     renderPassInfo.framebuffer = m_swapChain->getFramebuffer(m_currentImageIndex);
00100
00101     renderPassInfo.renderArea.offset = {0, 0};
00102     renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();
00103
00104     renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
00105     renderPassInfo.pClearValues = m_clearValues.data();
00106
00107     vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00108
00109     VkViewport viewport{};
00110     viewport.x = 0.0F;

```



```

00110     viewport.y = 0.0F;
00111     viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00112     viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00113     viewport.minDepth = 0.0F;
00114     viewport.maxDepth = 1.0F;
00115     const VkRect2D scissor{{0, 0}, m_swapChain->getSwapChainExtent()};
00116     vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00117     vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00118 }
00119
00120 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer)
00121 {
00122     assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
00123     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
    from a different frame");
00124
00125     vkCmdEndRenderPass(commandBuffer);
00126 }

```

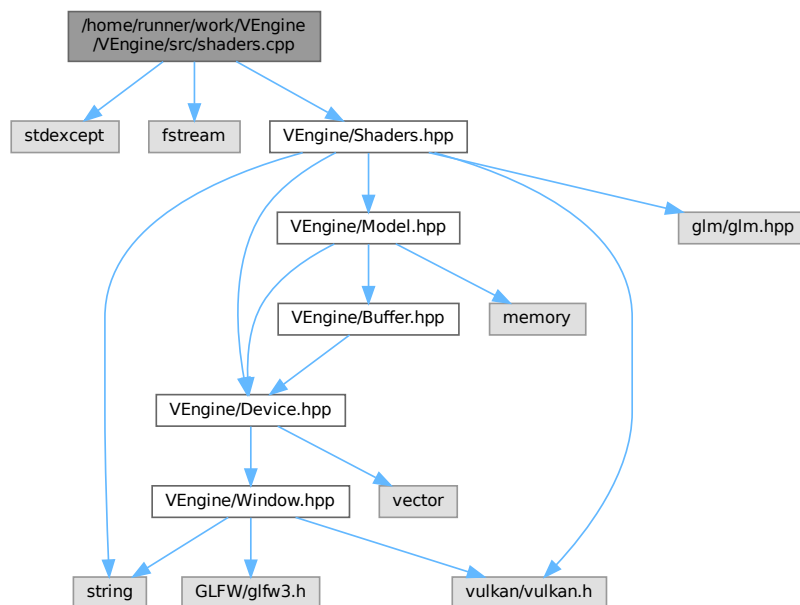
## 7.78 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```

#include <stdexcept>
#include <fstream>
#include "VEngine/Shaders.hpp"

```

Include dependency graph for shaders.cpp:



## 7.79 shaders.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002 #include <fstream>
00003
00004 #include "VEngine/Shaders.hpp"
00005

```

```

00006 ven::Shaders::~Shaders()
00007 {
00008     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00009     vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00010     vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
00011 }
00012
00013 std::vector<char> ven::Shaders::readFile(const std::string &filename)
00014 {
00015     std::ifstream file(filename, std::ios::ate | std::ios::binary);
00016
00017     if (!file.is_open()) {
00018         throw std::runtime_error("failed to open file!");
00019     }
00020
00021     const std::streamsize fileSize = file.tellg();
00022     std::vector<char> buffer(static_cast<unsigned long>(fileSize));
00023
00024     file.seekg(0);
00025     file.read(buffer.data(), fileSize);
00026
00027     file.close();
00028     return buffer;
00029 }
00030
00031 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo)
00032 {
00033     const std::vector<char> vertCode = readFile(vertFilepath);
00034     const std::vector<char> fragCode = readFile(fragFilepath);
00035
00036     createShaderModule(vertCode, &m_vertShaderModule);
00037     createShaderModule(fragCode, &m_fragShaderModule);
00038
00039     VkPipelineShaderStageCreateInfo shaderStages[2];
00040     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00041     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00042     shaderStages[0].module = m_vertShaderModule;
00043     shaderStages[0].pName = "main";
00044     shaderStages[0].flags = 0;
00045     shaderStages[0].pNext = nullptr;
00046     shaderStages[0].pSpecializationInfo = nullptr;
00047
00048     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00049     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00050     shaderStages[1].module = m_fragShaderModule;
00051     shaderStages[1].pName = "main";
00052     shaderStages[1].flags = 0;
00053     shaderStages[1].pNext = nullptr;
00054     shaderStages[1].pSpecializationInfo = nullptr;
00055
00056     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00057     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00058     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00059     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00060     vertexInputInfo.vertexAttributeDescriptionCount =
static_cast<uint32_t>(attributeDescriptions.size());
00061     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00062     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00063     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00064
00065
00066     VkPipelineViewportStateCreateInfo viewportInfo{};
00067     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00068     viewportInfo.viewportCount = 1;
00069     viewportInfo.pViewports = nullptr;
00070     viewportInfo.scissorCount = 1;
00071     viewportInfo.pScissors = nullptr;
00072
00073
00074     VkGraphicsPipelineCreateInfo pipelineInfo{};
00075     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00076     pipelineInfo.stageCount = 2;
00077     pipelineInfo.pStages = shaderStages;
00078     pipelineInfo.pVertexInputState = &vertexInputInfo;
00079     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00080     pipelineInfo.pViewportState = &viewportInfo;
00081     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00082     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00083
00084     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00085     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00086     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00087
00088     pipelineInfo.layout = configInfo.pipelineLayout;
00089     pipelineInfo.renderPass = configInfo.renderPass;
00090     pipelineInfo.subpass = configInfo.subpass;

```

```

00091
00092     pipelineInfo.basePipelineIndex = -1;
00093     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00094
00095     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
00096     &m_graphicsPipeline) != VK_SUCCESS) {
00097         throw std::runtime_error("failed to create graphics pipeline");
00098     }
00099 }
00100
00101 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
00102 const
00103 {
00104     VkShaderModuleCreateInfo createInfo{};
00105     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00106     createInfo.codeSize = code.size();
00107     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00108
00109     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00110         throw std::runtime_error("failed to create shader module");
00111     }
00112 }
00113
00114 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00115 {
00116     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00117     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00118     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00119
00120     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00121     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00122     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00123     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00124     configInfo.rasterizationInfo.lineWidth = 1.0F;
00125     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE;
00126     configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00127     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00128     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00129     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00130     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00131
00132     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00133     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00134     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00135     configInfo.multisampleInfo.minSampleShading = 1.0F;
00136     configInfo.multisampleInfo.pSampleMask = nullptr;
00137     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00138     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00139
00140     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
00141     VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00142     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00143     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00144     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00145     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00146     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00147     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00148     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00149
00150     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00151     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00152     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00153     configInfo.colorBlendInfo.attachmentCount = 1;
00154     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00155     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00156     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00157     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00158     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00159
00160     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00161     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00162     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00163     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00164     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00165     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00166     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00167     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00168     configInfo.depthStencilInfo.front = {};
00169     configInfo.depthStencilInfo.back = {};
00170
00171     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00172     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00173     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00174     configInfo.dynamicStateInfo.dynamicStateCount =
00175     static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00176     configInfo.dynamicStateInfo.flags = 0;
00177     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();

```

```

00174     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00175 }

```

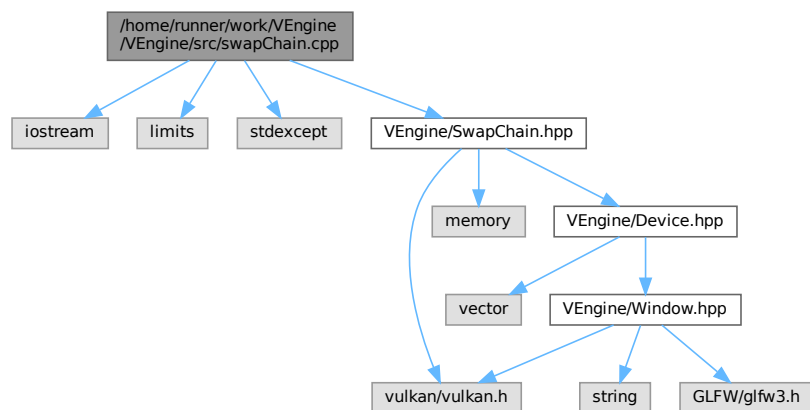
## 7.80 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference

```

#include <iostream>
#include <limits>
#include <stdexcept>
#include "VEngine/SwapChain.hpp"

```

Include dependency graph for swapChain.cpp:



## 7.81 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 : swapChainImageViews) {
00010         vkDestroyImageView(device.device(), imageView, nullptr);
00011     }
00012     swapChainImageViews.clear();
00013
00014     if (swapChain != nullptr) {
00015         vkDestroySwapchainKHR(device.device(), swapChain, nullptr);
00016         swapChain = nullptr;
00017     }
00018
00019     for (size_t i = 0; i < depthImages.size(); i++) {
00020         vkDestroyImageView(device.device(), depthImageViews[i], nullptr);
00021         vkDestroyImage(device.device(), depthImages[i], nullptr);
00022         vkFreeMemory(device.device(), depthImageMemory[i], nullptr);
00023     }
00024
00025     for (VkFramebuffer_T *framebuffer : swapChainFrameBuffers) {
00026         vkDestroyFramebuffer(device.device(), framebuffer, nullptr);
00027     }
00028
00029     vkDestroyRenderPass(device.device(), renderPass, nullptr);

```

```

00030
00031 // cleanup synchronization objects
00032 for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00033     vkDestroySemaphore(device.device(), renderFinishedSemaphores[i], nullptr);
00034     vkDestroySemaphore(device.device(), imageAvailableSemaphores[i], nullptr);
00035     vkDestroyFence(device.device(), 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(device.device(), 1, &inFlightFences[currentFrame], VK_TRUE,
00052         std::numeric_limits<uint64_t>::max());
00053     return vkAcquireNextImageKHR(device.device(), swapChain, std::numeric_limits<uint64_t>::max(),
00054         imageAvailableSemaphores[currentFrame], VK_NULL_HANDLE, imageIndex);
00055 }
00056
00057 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
00058     *imageIndex)
00059 {
00060     if (imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00061         vkWaitForFences(device.device(), 1, &imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00062     }
00063     imagesInFlight[*imageIndex] = inFlightFences[currentFrame];
00064
00065     VkSubmitInfo submitInfo = {};
00066     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00067
00068     const VkSemaphore waitSemaphores[] = {imageAvailableSemaphores[currentFrame]};
00069     const VkPipelineStageFlags waitStages[] = {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00070     submitInfo.waitSemaphoreCount = 1;
00071     submitInfo.pWaitSemaphores = waitSemaphores;
00072     submitInfo.pWaitDstStageMask = waitStages;
00073
00074     submitInfo.commandBufferCount = 1;
00075     submitInfo.pCommandBuffers = buffers;
00076
00077     const VkSemaphore signalSemaphores[] = {renderFinishedSemaphores[currentFrame]};
00078     submitInfo.signalSemaphoreCount = 1;
00079     submitInfo.pSignalSemaphores = signalSemaphores;
00080
00081     vkResetFences(device.device(), 1, &inFlightFences[currentFrame]);
00082     if (vkQueueSubmit(device.graphicsQueue(), 1, &submitInfo, inFlightFences[currentFrame]) !=
00083         VK_SUCCESS) {
00084         throw std::runtime_error("failed to submit draw command m_buffer!");
00085     }
00086
00087     VkPresentInfoKHR presentInfo = {};
00088     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00089
00090     presentInfo.waitSemaphoreCount = 1;
00091     presentInfo.pWaitSemaphores = signalSemaphores;
00092
00093     const VkSwapchainKHR swapChains[] = {swapChain};
00094     presentInfo.swapchainCount = 1;
00095     presentInfo.pSwapchains = swapChains;
00096
00097     presentInfo.pImageIndices = imageIndex;
00098
00099     const VkResult result = vkQueuePresentKHR(device.presentQueue(), &presentInfo);
00100
00101     currentFrame = (currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00102
00103     return result;
00104 }
00105
00106 void ven::SwapChain::createSwapChain()
00107 {
00108     const SwapChainSupportDetails swapChainSupport = device.getSwapChainSupport();
00109
00110     const VkSurfaceFormatKHR surfaceFormat = chooseSwapSurfaceFormat(swapChainSupport.formats);
00111     const VkPresentModeKHR presentMode = chooseSwapPresentMode(swapChainSupport.presentModes);
00112     const VkExtent2D extent = chooseSwapExtent(swapChainSupport.capabilities);
00113
00114     uint32_t imageCount = swapChainSupport.capabilities.minImageCount + 1;
00115     if (swapChainSupport.capabilities.maxImageCount > 0 && imageCount >

```

```

        swapChainSupport.capabilities.maxImageCount) {
00113         imageCount = swapChainSupport.capabilities.maxImageCount;
00114     }
00115
00116     VkSwapchainCreateInfoKHR createInfo = {};
00117     createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
00118     createInfo.surface = device.surface();
00119
00120     createInfo.minImageCount = imageCount;
00121     createInfo.imageFormat = surfaceFormat.format;
00122     createInfo.imageColorSpace = surfaceFormat.colorSpace;
00123     createInfo.imageExtent = extent;
00124     createInfo.imageArrayLayers = 1;
00125     createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
00126
00127     const QueueFamilyIndices indices = device.findPhysicalQueueFamilies();
00128     const uint32_t queueFamilyIndices[] = {indices.graphicsFamily, indices.presentFamily};
00129
00130     if (indices.graphicsFamily != indices.presentFamily) {
00131         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00132         createInfo.queueFamilyIndexCount = 2;
00133         createInfo.pQueueFamilyIndices = queueFamilyIndices;
00134     } else {
00135         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00136         createInfo.queueFamilyIndexCount = 0; // Optional
00137         createInfo.pQueueFamilyIndices = nullptr; // Optional
00138     }
00139
00140     createInfo.preTransform = swapChainSupport.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 = oldSwapChain == nullptr ? VK_NULL_HANDLE : oldSwapChain->swapChain;
00147
00148     if (vkCreateSwapchainKHR(device.device(), &createInfo, nullptr, &swapChain) != VK_SUCCESS) {
00149         throw std::runtime_error("failed to create swap chain!");
00150     }
00151
00152     vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, nullptr);
00153     swapChainImages.resize(imageCount);
00154     vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, swapChainImages.data());
00155
00156     swapChainImageFormat = surfaceFormat.format;
00157     m_swapChainExtent = extent;
00158 }
00159
00160 void ven::SwapChain::createImageViews()
00161 {
00162     swapChainImageViews.resize(swapChainImages.size());
00163     for (size_t i = 0; i < swapChainImages.size(); i++) {
00164         VkImageViewCreateInfo viewInfo{};
00165         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00166         viewInfo.image = swapChainImages[i];
00167         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00168         viewInfo.format = 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(device.device(), &viewInfo, nullptr, &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(device.device(), &renderPassInfo, nullptr, &renderPass) != VK_SUCCESS) {
00236         throw std::runtime_error("failed to create render pass!");
00237     }
00238 }
00239
00240 void ven::SwapChain::createFrameBuffers()
00241 {
00242     swapChainFrameBuffers.resize(imageCount());
00243     for (size_t i = 0; i < imageCount(); i++) {
00244         std::array<VkImageView, 2> attachments = {swapChainImageViews[i], depthImageViews[i]};
00245
00246         const VkExtent2D swapChainExtent = getSwapChainExtent();
00247         VkFramebufferCreateInfo framebufferInfo = {};
00248         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00249         framebufferInfo.renderPass = renderPass;
00250         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00251         framebufferInfo.pAttachments = attachments.data();
00252         framebufferInfo.width = swapChainExtent.width;
00253         framebufferInfo.height = swapChainExtent.height;
00254         framebufferInfo.layers = 1;
00255
00256         if (vkCreateFramebuffer(device.device(), &framebufferInfo, nullptr, &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 VkExtent2D swapChainExtent = getSwapChainExtent();
00266
00267     swapChainDepthFormat = depthFormat;
00268     depthImages.resize(imageCount());
00269     depthImageMemory.resize(imageCount());
00270     depthImageViews.resize(imageCount());
00271
00272     for (size_t i = 0; i < 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 = swapChainExtent.width;
00277         imageInfo.extent.height = swapChainExtent.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         device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, depthImages[i],
depthImageMemory[i]);
00290
00291         VkImageViewCreateInfo viewInfo{};
00292         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00293         viewInfo.image = 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(device.device(), &viewInfo, nullptr, &depthImageViews[i]) != VK_SUCCESS)
00303         {
00304             throw std::runtime_error("failed to create texture image view!");
00305         }
00306     }
00307
00308 void ven::SwapChain::createSyncObjects()
00309 {
00310     imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00311     renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00312     inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
00313     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(device.device(), &semaphoreInfo, nullptr, &imageAvailableSemaphores[i])
!= VK_SUCCESS ||
00324             vkCreateSemaphore(device.device(), &semaphoreInfo, nullptr, &renderFinishedSemaphores[i])
!= VK_SUCCESS ||
00325             vkCreateFence(device.device(), &fenceInfo, nullptr, &inFlightFences[i]) != VK_SUCCESS) {
00326             throw std::runtime_error("failed to create synchronization objects for a frame!");
00327         }
00328     }
00329 }
00330
00331 VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats)
00332 {
00333     for (const auto &availableFormat : availableFormats) {
00334         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00335             return availableFormat;
00336         }
00337     }
00338     return availableFormats[0];
00339 }
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     std::cout << "Present mode: V-Sync\n";
00358     return VK_PRESENT_MODE_FIFO_KHR;
00359 }
00360 }

```



```

00361
00362 VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00363 {
00364     if (capabilities.currentExtent.width != std::numeric_limits<uint32_t>::max()) {
00365         return capabilities.currentExtent;
00366     }
00367     VkExtent2D actualExtent = windowExtent;
00368     actualExtent.width = std::max(capabilities.minImageExtent.width,
00369     std::min(capabilities.maxImageExtent.width, actualExtent.width));
00369     actualExtent.height = std::max(capabilities.minImageExtent.height,
00370     std::min(capabilities.maxImageExtent.height, actualExtent.height));
00370
00371     return actualExtent;
00372 }
00373
00374 VkFormat ven::SwapChain::findDepthFormat() const
00375 {
00376     return device.findSupportedFormat(
00377     {VK_FORMAT_D32_SFLOAT, VK_FORMAT_D32_SFLOAT_S8_UINT, VK_FORMAT_D24_UNORM_S8_UINT},
00378     VK_IMAGE_TILING_OPTIMAL,
00379     VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT);
00380 }

```

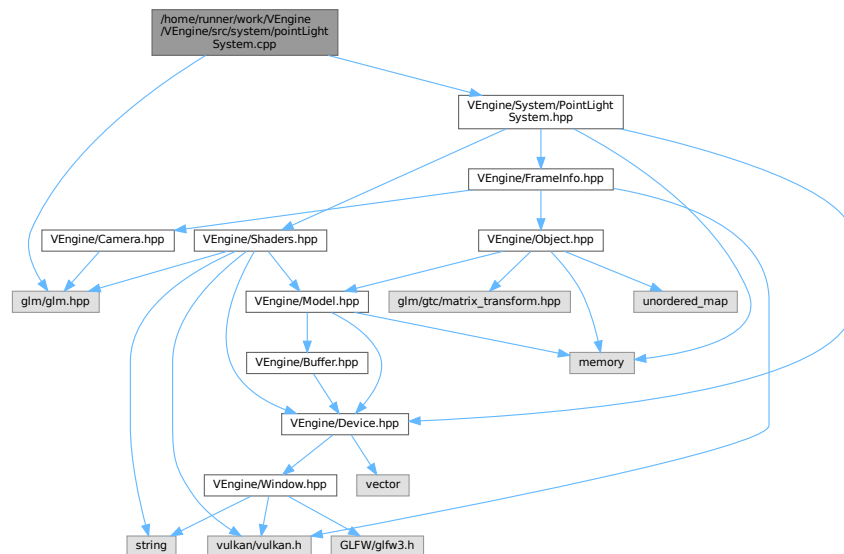
## 7.82 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference

```

#include <glm/glm.hpp>
#include "VEngine/System/PointLightSystem.hpp"

```

Include dependency graph for pointLightSystem.cpp:



### Classes

- struct [PointLightPushConstants](#)

## 7.83 pointLightSystem.cpp

[Go to the documentation of this file.](#)

```

00001 #include <glm/glm.hpp>
00002
00003 #include "VEngine/System/PointLightSystem.hpp"
00004
00005 struct PointLightPushConstants {
00006     glm::vec4 position{};
00007     glm::vec4 color{};
00008     float radius;
00009 };
00010
00011 ven::PointLightSystem::PointLightSystem(Device& device, const VkRenderPass renderPass, const
    VkDescriptorSetLayout globalSetLayout) : m_device(device)
00012 {
00013     createPipelineLayout(globalSetLayout);
00014     createPipeline(renderPass);
00015 }
00016
00017 void ven::PointLightSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
00018 {
00019     VkPushConstantRange pushConstantRange{};
00020     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00021     pushConstantRange.offset = 0;
00022     pushConstantRange.size = sizeof(PointLightPushConstants);
00023
00024     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};
00025
00026     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00027     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00028     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00029     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00030     pipelineLayoutInfo.pushConstantRangeCount = 1;
00031     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00032     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
        VK_SUCCESS)
00033     {
00034         throw std::runtime_error("Failed to create pipeline layout");
00035     }
00036 }
00037
00038 void ven::PointLightSystem::createPipeline(const VkRenderPass renderPass)
00039 {
00040     PipelineConfigInfo pipelineConfig{};
00041     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00042     pipelineConfig.attributeDescriptions.clear();
00043     pipelineConfig.bindingDescriptions.clear();
00044     pipelineConfig.renderPass = renderPass;
00045     pipelineConfig.pipelineLayout = m_pipelineLayout;
00046     m_shaders = std::make_unique<Shaders>(m_device, std::string(SHADERS_BIN_PATH) +
        "point_light_vert.spv", std::string(SHADERS_BIN_PATH) + "point_light_frag.spv", pipelineConfig);
00047 }
00048
00049 void ven::PointLightSystem::render(const FrameInfo &frameInfo) const
00050 {
00051     m_shaders->bind(frameInfo.commandBuffer);
00052
00053     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
        m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00054
00055     for (auto &kv : frameInfo.objects)
00056     {
00057         Object &object = kv.second;
00058         if (object.pointLight == nullptr) continue;
00059         PointLightPushConstants push{};
00060         push.position = glm::vec4(object.transform3D.translation, 1.F);
00061         push.color = glm::vec4(object.color, object.pointLight->lightIntensity);
00062         push.radius = object.transform3D.scale.x;
00063         vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT |
            VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(PointLightPushConstants), &push);
00064         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00065     }
00066 }
00067
00068
00069 void ven::PointLightSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
00070 {
00071     const auto rotateLight = rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});
00072     unsigned long lightIndex = 0;
00073     for (auto &kv : frameInfo.objects)
00074     {
00075         Object &object = kv.second;
00076         if (object.pointLight == nullptr) continue;
00077         assert(lightIndex < MAX_LIGHTS && "Too many lights");
00078         object.transform3D.translation = glm::vec3(rotateLight *
            glm::vec4(object.transform3D.translation, 1.F));
00079         ubo.pointLights[lightIndex].position = glm::vec4(object.transform3D.translation, 1.F);
00080         ubo.pointLights[lightIndex].color = glm::vec4(object.color,
            object.pointLight->lightIntensity);

```

```

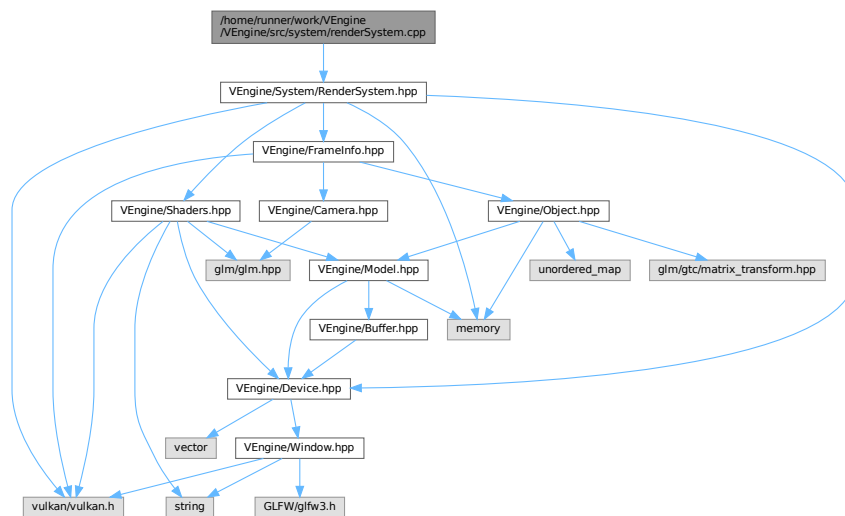
00081         lightIndex++;
00082     }
00083     ubo.numLights = static_cast<int>(lightIndex);
00084 }

```

## 7.84 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference

#include "VEngine/System/RenderSystem.hpp"

Include dependency graph for renderSystem.cpp:



## 7.85 renderSystem.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/System/RenderSystem.hpp"
00002
00003 ven::RenderSystem::RenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : m_device(device)
00004 {
00005     createPipelineLayout(globalSetLayout);
00006     createPipeline(renderPass);
00007 }
00008
00009 void ven::RenderSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
00010 {
00011     VkPushConstantRange pushConstantRange{};
00012     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00013     pushConstantRange.offset = 0;
00014     pushConstantRange.size = sizeof(SimplePushConstantData);
00015
00016     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};
00017
00018     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00019     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00020     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00021     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00022     pipelineLayoutInfo.pushConstantRangeCount = 1;
00023     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00024     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
VK_SUCCESS)
00025     {
00026         throw std::runtime_error("Failed to create pipeline layout");

```

```

00027     }
00028 }
00029
00030 void ven::RenderSystem::createPipeline(const VkRenderPass renderPass)
00031 {
00032     PipelineConfigInfo pipelineConfig{};
00033     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00034     pipelineConfig.renderPass = renderPass;
00035     pipelineConfig.pipelineLayout = m_pipelineLayout;
00036     m_shaders = std::make_unique<Shaders>(m_device, std::string(SHADERS_BIN_PATH) + "shader_vert.spv",
std::string(SHADERS_BIN_PATH) + "shader_frag.spv", pipelineConfig);
00037 }
00038
00039 void ven::RenderSystem::renderObjects(const FrameInfo &frameInfo) const
00040 {
00041     m_shaders->bind(frameInfo.commandBuffer);
00042
00043     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00044
00045     for (auto &kv : frameInfo.objects)
00046     {
00047         Object &object = kv.second;
00048         if (object.model == nullptr) continue;
00049         SimplePushConstantData push{};
00050         push.modelMatrix = object.transform3D.mat4();
00051         push.normalMatrix = object.transform3D.normalMatrix();
00052         vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(SimplePushConstantData), &push);
00053         object.model->bind(frameInfo.commandBuffer);
00054         object.model->draw(frameInfo.commandBuffer);
00055     }
00056 }

```

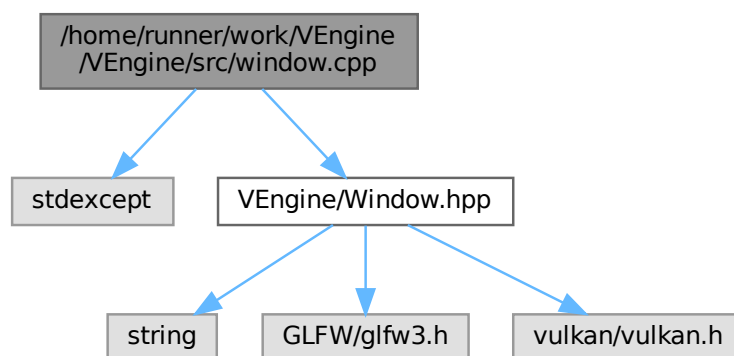
## 7.86 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

```

#include <stdexcept>
#include "VEngine/Window.hpp"

```

Include dependency graph for window.cpp:



## 7.87 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 }
```



# Index

[/home/runner/work/VEngine/VEngine/README.md](#),  
[242](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp](#),  
[189, 190](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp](#),  
[192, 194](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Color.hpp](#),  
[195, 196](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#),  
[197, 198](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#),  
[199, 201](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp](#),  
[201, 203](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#),  
[203, 205](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp](#),  
[206, 207](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Framerate.hpp](#),  
[208, 209](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp](#),  
[210, 211](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp](#),  
[212, 213](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#),  
[214, 215](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#),  
[216, 218](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp](#),  
[219, 220](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Shader.hpp](#),  
[221, 222](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp](#),  
[223, 225](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp](#),  
[226, 227](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp](#),  
[228, 229](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp](#),  
[230, 231](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp](#),  
[231, 233](#)

[/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp](#),  
[234, 235](#)

[/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp](#),  
[236, 238](#)

[/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/DescriptorPool.hpp](#),  
[238, 240](#)

[/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp](#),  
[241](#)

[/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp](#),  
[242](#)

[/home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp](#),  
[261](#)

[/home/runner/work/VEngine/VEngine/src/buffer.cpp](#),  
[242, 243](#)

[/home/runner/work/VEngine/VEngine/src/camera.cpp](#),  
[244, 245](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp](#),  
[246](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#),  
[247, 248](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp](#),  
[248, 249](#)

[/home/runner/work/VEngine/VEngine/src/device.cpp](#),  
[250, 252](#)

[/home/runner/work/VEngine/VEngine/src/engine.cpp](#),  
[258, 259](#)

[/home/runner/work/VEngine/VEngine/src/keyboardController.cpp](#),  
[265](#)

[/home/runner/work/VEngine/VEngine/src/main.cpp](#),  
[266, 267](#)

[/home/runner/work/VEngine/VEngine/src/model.cpp](#),  
[267, 269](#)

[/home/runner/work/VEngine/VEngine/src/object.cpp](#),  
[271](#)

[/home/runner/work/VEngine/VEngine/src/renderer.cpp](#),  
[273](#)

[/home/runner/work/VEngine/VEngine/src/shaders.cpp](#),  
[275](#)

[/home/runner/work/VEngine/VEngine/src/swapChain.cpp](#),  
[278](#)

[/home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp](#),  
[283](#)

[/home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp](#),  
[285](#)

[/home/runner/work/VEngine/VEngine/src/window.cpp](#),  
[286](#)

[Buffer](#),  
[ven::Buffer](#), [24](#)

[Clock](#),  
[myLib::Clock](#), [50](#)

[DescriptorPool](#),  
[ven::DescriptorPool](#), [62](#)

[DescriptorSetLayout](#),  
[ven::DescriptorSetLayout](#), [67](#)

[~Device](#)

- ven::Device, 76
- ~Engine
  - ven::Engine, 92
- ~ImGuiWindowManager
  - ven::ImGuiWindowManager, 105
- ~Model
  - ven::Model, 114
- ~Object
  - ven::Object, 120
- ~PointLightSystem
  - ven::PointLightSystem, 133
- ~RenderSystem
  - ven::RenderSystem, 153
- ~Renderer
  - ven::Renderer, 144
- ~Shaders
  - ven::Shaders, 159
- ~SwapChain
  - ven::SwapChain, 166
- ~Window
  - ven::Window, 184
- acquireNextImage
  - ven::SwapChain, 167
- addBinding
  - ven::DescriptorSetLayout::Builder, 39
- addPoolSize
  - ven::DescriptorPool::Builder, 35
- allocateDescriptor
  - ven::DescriptorPool, 63
- ambientLightColor
  - ven::GlobalUbo, 102
- AQUA
  - ven::Colors, 54
- AQUA\_V
  - ven::Colors, 54
- asMicroseconds
  - myLib::Time, 177
- asMilliseconds
  - myLib::Time, 177
- asSeconds
  - myLib::Time, 177
- attributeDescriptions
  - ven::PipelineConfigInfo, 126
- beginFrame
  - ven::Renderer, 144
- beginSingleTimeCommands
  - ven::Device, 77
- beginSwapChainRenderPass
  - ven::Renderer, 144
- bind
  - ven::Model, 115
  - ven::Shaders, 160
- bindingDescriptions
  - ven::PipelineConfigInfo, 126
- BLACK
  - ven::Colors, 54
- BLACK\_V
  - ven::Colors, 54
- BLUE
  - ven::Colors, 54
- BLUE\_V
  - ven::Colors, 54
- Buffer
  - ven::Buffer, 24
- build
  - ven::DescriptorPool::Builder, 35
  - ven::DescriptorSetLayout::Builder, 39
  - ven::DescriptorWriter, 71
- Builder
  - ven::DescriptorPool::Builder, 35
  - ven::DescriptorSetLayout::Builder, 39
- camera
  - ven::FrameInfo, 98
- capabilities
  - ven::SwapChainSupportDetails, 175
- checkDeviceExtensionSupport
  - ven::Device, 77
- checkValidationLayerSupport
  - ven::Device, 77
- chooseSwapExtent
  - ven::SwapChain, 167
- chooseSwapPresentMode
  - ven::SwapChain, 167
- chooseSwapSurfaceFormat
  - ven::SwapChain, 167
- cleanup
  - ven::ImGuiWindowManager, 105
- CLEAR\_COLORS
  - ven::Colors, 54
- Clock
  - myLib::Clock, 50
- Clock.hpp
  - TimePoint, 235
- color
  - PointLightPushConstants, 130
  - ven::Model::Vertex, 182
  - ven::Object, 123
  - ven::PointLight, 128
- COLOR\_MAX
  - ven::Colors, 55
- colorBlendAttachment
  - ven::PipelineConfigInfo, 126
- colorBlendInfo
  - ven::PipelineConfigInfo, 126
- COLORS
  - ven::Colors, 55
- commandBuffer
  - ven::FrameInfo, 98
- compareSwapFormats
  - ven::SwapChain, 168
- copyBuffer
  - ven::Device, 77
- copyBufferToImage
  - ven::Device, 78
- createBuffer



- ven::Device, [78](#)
- createCommandBuffers
  - ven::Renderer, [144](#)
- createCommandPool
  - ven::Device, [78](#)
- CreateDebugUtilsMessengerEXT
  - device.cpp, [251](#)
- createDepthResources
  - ven::SwapChain, [168](#)
- createFrameBuffers
  - ven::SwapChain, [168](#)
- createGraphicsPipeline
  - ven::Shaders, [160](#)
- createImageViews
  - ven::SwapChain, [168](#)
- createImageWithInfo
  - ven::Device, [79](#)
- createIndexBuffer
  - ven::Model, [115](#)
- createInstance
  - ven::Device, [79](#)
  - ven::Engine, [92](#)
- createLogicalDevice
  - ven::Device, [79](#)
- createModelFromFile
  - ven::Model, [115](#)
- createObject
  - ven::Object, [121](#)
- createPipeline
  - ven::PointLightSystem, [134](#)
  - ven::RenderSystem, [154](#)
- createPipelineLayout
  - ven::PointLightSystem, [135](#)
  - ven::RenderSystem, [155](#)
- createRenderPass
  - ven::SwapChain, [168](#)
- createShaderModule
  - ven::Shaders, [160](#)
- createSurface
  - ven::Device, [80](#)
  - ven::Engine, [92](#)
- createSwapChain
  - ven::SwapChain, [168](#)
- createSyncObjects
  - ven::SwapChain, [168](#)
- createVertexBuffer
  - ven::Model, [116](#)
- createWindow
  - ven::Window, [184](#)
- createWindowSurface
  - ven::Window, [184](#)
- currentFrame
  - ven::SwapChain, [171](#)
- CYAN
  - ven::Colors, [55](#)
- CYAN\_V
  - ven::Colors, [56](#)
- debugCallback
  - device.cpp, [251](#)
- DEFAULT\_CLEAR\_COLOR
  - ven, [16](#)
- DEFAULT\_CLEAR\_DEPTH
  - ven, [16](#)
- DEFAULT\_FAR
  - ven, [17](#)
- 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\_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, [160](#)
- depthImageMemory
  - ven::SwapChain, [171](#)
- depthImages
  - ven::SwapChain, [171](#)
- depthImageViews
  - ven::SwapChain, [171](#)
- depthStencilInfo
  - ven::PipelineConfigInfo, [126](#)
- descriptorInfo
  - ven::Buffer, [24](#)
- descriptorInfoForIndex
  - ven::Buffer, [25](#)
- DescriptorPool
  - ven::DescriptorPool, [62](#), [63](#)
- DescriptorSetLayout
  - ven::DescriptorSetLayout, [67](#), [68](#)
- DescriptorWriter
  - ven::DescriptorPool, [64](#)
  - ven::DescriptorSetLayout, [68](#)
  - ven::DescriptorWriter, [71](#)
- DestroyDebugUtilsMessengerEXT
  - device.cpp, [251](#)
- Device
  - ven::Device, [76](#), [77](#)
- device

- ven::Device, 80
- ven::SwapChain, 172
- device.cpp
  - CreateDebugUtilsMessengerEXT, 251
  - debugCallback, 251
  - DestroyDebugUtilsMessengerEXT, 251
- deviceExtensions
  - ven::Device, 87
- draw
  - ven::Model, 117
- dynamicStateEnables
  - ven::PipelineConfigInfo, 126
- dynamicStateInfo
  - ven::PipelineConfigInfo, 126
- enableValidationLayers
  - ven::Device, 87
- endFrame
  - ven::Renderer, 145
- endSingleTimeCommands
  - ven::Device, 81
- endSwapChainRenderPass
  - ven::Renderer, 145
- Engine
  - ven::Engine, 91, 92
- extentAspectRatio
  - ven::SwapChain, 169
- findDepthFormat
  - ven::SwapChain, 169
- findMemoryType
  - ven::Device, 81
- findPhysicalQueueFamilies
  - ven::Device, 82
- findQueueFamilies
  - ven::Device, 82
- findSupportedFormat
  - ven::Device, 82
- flush
  - ven::Buffer, 25
- flushIndex
  - ven::Buffer, 26
- formats
  - ven::SwapChainSupportDetails, 175
- framebufferResizeCallback
  - ven::Window, 185
- frameIndex
  - ven::FrameInfo, 98
- frameTime
  - ven::FrameInfo, 98
- freeCommandBuffers
  - ven::Renderer, 145
- freeDescriptors
  - ven::DescriptorPool, 63
- FUCHSIA
  - ven::Colors, 56
- FUCHSIA\_V
  - ven::Colors, 56
- getAlignment
  - ven::Buffer, 27
- getAlignmentSize
  - ven::Buffer, 27
- getAspectRatio
  - ven::Renderer, 146
- getAttributeDescriptions
  - ven::Model::Vertex, 181
- getBindingDescriptions
  - ven::Model::Vertex, 181
- getBuffer
  - ven::Buffer, 27
- getBufferSize
  - ven::Buffer, 28
- getClearColor
  - ven::Renderer, 146
- getCommandPool
  - ven::Device, 83
- getCurrentCommandBuffer
  - ven::Renderer, 146
- getDescriptorPool
  - ven::DescriptorPool, 63
- getDescriptorSetLayout
  - ven::DescriptorSetLayout, 68
- getElapsedTime
  - myLib::Clock, 50
- getExtent
  - ven::Window, 185
- getFar
  - ven::Camera, 44
- getFov
  - ven::Camera, 44
- getFramebuffer
  - ven::SwapChain, 169
- getFrameIndex
  - ven::Renderer, 147
- getGLFWWindow
  - ven::Window, 185
- getGraphicsQueue
  - ven::Device, 83
- getId
  - ven::Object, 122
- getImageView
  - ven::SwapChain, 169
- getInstanceCount
  - ven::Buffer, 28
- getInstanceSize
  - ven::Buffer, 28
- getInverseView
  - ven::Camera, 44
- getMappedMemory
  - ven::Buffer, 28
- getMemoryPropertyFlags
  - ven::Buffer, 28
- getNear
  - ven::Camera, 45
- getPhysicalDevice
  - ven::Device, 83

- getProjection
  - ven::Camera, [45](#)
- getRenderPass
  - ven::SwapChain, [169](#)
- getRequiredExtensions
  - ven::Device, [83](#)
- getSwapChainExtent
  - ven::SwapChain, [169](#)
- getSwapChainImageFormat
  - ven::SwapChain, [170](#)
- getSwapChainRenderPass
  - ven::Renderer, [147](#)
- getSwapChainSupport
  - ven::Device, [84](#)
- getUsageFlags
  - ven::Buffer, [28](#)
- getView
  - ven::Camera, [45](#)
- GLFW\_INCLUDE\_VULKAN
  - Window.hpp, [233](#)
- globalDescriptorSet
  - ven::FrameInfo, [98](#)
- graphicsFamily
  - ven::QueueFamilyIndices, [138](#)
- graphicsFamilyHasValue
  - ven::QueueFamilyIndices, [138](#)
- graphicsQueue
  - ven::Device, [84](#)
- GRAY
  - ven::Colors, [56](#)
- GRAY\_V
  - ven::Colors, [56](#)
- GREEN
  - ven::Colors, [56](#)
- GREEN\_V
  - ven::Colors, [56](#)
- hasGlfwRequiredInstanceExtensions
  - ven::Device, [84](#)
- hashCombine
  - ven, [16](#)
- height
  - ven::SwapChain, [170](#)
- imageAvailableSemaphores
  - ven::SwapChain, [172](#)
- imageCount
  - ven::SwapChain, [170](#)
- imagesInFlight
  - ven::SwapChain, [172](#)
- ImGuiWindowManager
  - ven::ImGuiWindowManager, [105](#)
- indices
  - ven::Model::Builder, [42](#)
- inFlightFences
  - ven::SwapChain, [172](#)
- init
  - ven::ImGuiWindowManager, [105](#)
  - ven::SwapChain, [170](#)
- inputAssemblyInfo
  - ven::PipelineConfigInfo, [127](#)
- invalidate
  - ven::Buffer, [29](#)
- invalidateIndex
  - ven::Buffer, [29](#)
- inverseView
  - ven::GlobalUbo, [102](#)
- isComplete
  - ven::QueueFamilyIndices, [138](#)
- isDeviceSuitable
  - ven::Device, [84](#)
- isFrameInProgress
  - ven::Renderer, [148](#)
- IsLegacyNativeDupe
  - ven::ImGuiWindowManager::funcs, [100](#)
- lightIntensity
  - ven::PointLightComponent, [129](#)
- LIME
  - ven::Colors, [57](#)
- LIME\_V
  - ven::Colors, [57](#)
- loadModel
  - ven::Model::Builder, [42](#)
- loadObjects
  - ven::Engine, [93](#)
- lookDown
  - ven::KeyboardController::KeyMappings, [111](#)
- lookLeft
  - ven::KeyboardController::KeyMappings, [111](#)
- lookRight
  - ven::KeyboardController::KeyMappings, [111](#)
- lookUp
  - ven::KeyboardController::KeyMappings, [111](#)
- m\_alignmentSize
  - ven::Buffer, [32](#)
- m\_bindings
  - ven::DescriptorSetLayout, [69](#)
  - ven::DescriptorSetLayout::Builder, [40](#)
- m\_buffer
  - ven::Buffer, [32](#)
- m\_bufferSize
  - ven::Buffer, [32](#)
- m\_clearValues
  - ven::Renderer, [149](#)
- m\_commandBuffers
  - ven::Renderer, [149](#)
- m\_commandPool
  - ven::Device, [88](#)
- m\_currentFrameIndex
  - ven::Renderer, [150](#)
- m\_currentImageIndex
  - ven::Renderer, [150](#)
- m\_debugMessenger
  - ven::Device, [88](#)
- m\_descriptorPool
  - ven::DescriptorPool, [64](#)

- m\_descriptorSetLayout
  - ven::DescriptorSetLayout, 69
- m\_device
  - ven::Buffer, 32
  - ven::DescriptorPool, 64
  - ven::DescriptorPool::Builder, 37
  - ven::DescriptorSetLayout, 69
  - ven::DescriptorSetLayout::Builder, 40
  - ven::Device, 88
  - ven::Engine, 95
  - ven::Model, 117
  - ven::PointLightSystem, 136
  - ven::Renderer, 150
  - ven::RenderSystem, 156
  - ven::Shaders, 162
- m\_far
  - ven::Camera, 48
- m\_fov
  - ven::Camera, 48
- m\_fragShaderModule
  - ven::Shaders, 162
- m\_framebufferResized
  - ven::Window, 186
- m\_globalPool
  - ven::Engine, 95
- m\_graphicsPipeline
  - ven::Shaders, 162
- m\_graphicsQueue
  - ven::Device, 88
- m\_hasIndexBuffer
  - ven::Model, 117
- m\_height
  - ven::Window, 186
- m\_indexBuffer
  - ven::Model, 117
- m\_indexCount
  - ven::Model, 117
- m\_instance
  - ven::Device, 88
  - ven::Engine, 95
- m\_instanceCount
  - ven::Buffer, 32
- m\_instanceSize
  - ven::Buffer, 33
- m\_inverseViewMatrix
  - ven::Camera, 48
- m\_isFrameStarted
  - ven::Renderer, 150
- m\_keys
  - ven::KeyboardController, 109
- m\_lookSpeed
  - ven::KeyboardController, 109
- m\_mapped
  - ven::Buffer, 33
- m\_maxSets
  - ven::DescriptorPool::Builder, 37
- m\_memory
  - ven::Buffer, 33
- m\_memoryPropertyFlags
  - ven::Buffer, 33
- m\_moveSpeed
  - ven::KeyboardController, 109
- m\_near
  - ven::Camera, 48
- m\_objects
  - ven::Engine, 95
- m\_objId
  - ven::Object, 123
- m\_pause
  - myLib::Clock, 51
- m\_paused
  - myLib::Clock, 51
- m\_physicalDevice
  - ven::Device, 88
- m\_pipelineLayout
  - ven::PointLightSystem, 136
  - ven::RenderSystem, 156
- m\_pool
  - ven::DescriptorWriter, 72
- m\_poolFlags
  - ven::DescriptorPool::Builder, 37
- m\_poolSizes
  - ven::DescriptorPool::Builder, 37
- m\_presentQueue
  - ven::Device, 89
- m\_projectionMatrix
  - ven::Camera, 48
- m\_renderer
  - ven::Engine, 95
- m\_seconds
  - myLib::Time, 178
- m\_setLayout
  - ven::DescriptorWriter, 72
- m\_shaders
  - ven::PointLightSystem, 136
  - ven::RenderSystem, 156
- m\_start
  - myLib::Clock, 51
- m\_surface
  - ven::Device, 89
  - ven::Engine, 96
- m\_swapChain
  - ven::Renderer, 150
- m\_swapChainExtent
  - ven::SwapChain, 172
- m\_usageFlags
  - ven::Buffer, 33
- m\_vertexBuffer
  - ven::Model, 118
- m\_vertexCount
  - ven::Model, 118
- m\_vertShaderModule
  - ven::Shaders, 162
- m\_viewMatrix
  - ven::Camera, 48
- m\_width

- ven::Window, [187](#)
- m\_window
  - ven::Device, [89](#)
  - ven::Engine, [96](#)
  - ven::Renderer, [150](#)
  - ven::Window, [187](#)
- m\_writes
  - ven::DescriptorWriter, [72](#)
- MAGENTA
  - ven::Colors, [57](#)
- MAGENTA\_V
  - ven::Colors, [57](#)
- main
  - main.cpp, [266](#)
- main.cpp
  - main, [266](#)
- mainLoop
  - ven::Engine, [94](#)
- makePointLight
  - ven::Object, [122](#)
- Map
  - ven::Object, [120](#)
- map
  - ven::Buffer, [30](#)
- MAROON
  - ven::Colors, [57](#)
- MAROON\_V
  - ven::Colors, [57](#)
- mat4
  - ven::Transform3DComponent, [179](#)
- MAX\_FRAMES\_IN\_FLIGHT
  - ven::SwapChain, [172](#)
- MAX\_LIGHTS
  - ven, [19](#)
- MICROSECONDS\_PER\_SECOND
  - myLib, [9](#)
- MILLISECONDS\_PER\_SECOND
  - myLib, [9](#)
- Model
  - ven::Model, [114](#), [115](#)
- model
  - ven::Object, [123](#)
- model.cpp
  - TINYOBJLOADER\_IMPLEMENTATION, [268](#)
- modelMatrix
  - ven::SimplePushConstantData, [163](#)
- moveBackward
  - ven::KeyboardController::KeyMappings, [111](#)
- moveDown
  - ven::KeyboardController::KeyMappings, [111](#)
- moveForward
  - ven::KeyboardController::KeyMappings, [112](#)
- moveInPlaneXZ
  - ven::KeyboardController, [109](#)
- moveLeft
  - ven::KeyboardController::KeyMappings, [112](#)
- moveRight
  - ven::KeyboardController::KeyMappings, [112](#)
- moveUp
  - ven::KeyboardController::KeyMappings, [112](#)
- multisampleInfo
  - ven::PipelineConfigInfo, [127](#)
- myLib, [9](#)
  - MICROSECONDS\_PER\_SECOND, [9](#)
  - MILLISECONDS\_PER\_SECOND, [9](#)
  - RANDOM\_FLOAT\_MAX, [9](#)
  - RANDOM\_INT\_MAX, [10](#)
  - RANDOM\_INT\_MIN, [10](#)
- myLib::Clock, [49](#)
  - ~Clock, [50](#)
  - Clock, [50](#)
  - getElapsedTime, [50](#)
  - m\_pause, [51](#)
  - m\_paused, [51](#)
  - m\_start, [51](#)
  - pause, [50](#)
  - restart, [50](#)
  - resume, [50](#)
- myLib::Random, [139](#)
  - randomFloat, [140](#)
  - randomInt, [140](#), [141](#)
- myLib::Time, [176](#)
  - asMicroseconds, [177](#)
  - asMilliseconds, [177](#)
  - asSeconds, [177](#)
  - m\_seconds, [178](#)
  - Time, [177](#)
- name
  - ven::Object, [123](#)
- NAVY
  - ven::Colors, [57](#)
- NAVY\_V
  - ven::Colors, [58](#)
- NIGHT\_BLUE
  - ven::Colors, [58](#)
- NIGHT\_BLUE\_V
  - ven::Colors, [58](#)
- NIGHT\_MODE\_V
  - ven::Colors, [58](#)
- normal
  - ven::Model::Vertex, [182](#)
- normalMatrix
  - ven::SimplePushConstantData, [163](#)
  - ven::Transform3DComponent, [179](#)
- numLights
  - ven::GlobalUbo, [102](#)
- Object
  - ven::Object, [120](#), [121](#)
- objects
  - ven::FrameInfo, [98](#)
- oldSwapChain
  - ven::SwapChain, [173](#)
- OLIVE
  - ven::Colors, [58](#)
- OLIVE\_V

- ven::Colors, 58
- operator()
  - std::hash< ven::Model::Vertex >, 103
- operator=
  - ven::Buffer, 30
  - ven::DescriptorPool, 64
  - ven::DescriptorSetLayout, 68
  - ven::Device, 85
  - ven::Engine, 95
  - ven::ImGuiWindowManager, 106
  - ven::Model, 117
  - ven::Object, 122, 123
  - ven::PipelineConfigInfo, 125
  - ven::PointLightSystem, 135
  - ven::Renderer, 148
  - ven::RenderSystem, 155
  - ven::Shaders, 161
  - ven::SwapChain, 170
- operator==
  - ven::Model::Vertex, 181
- overwrite
  - ven::DescriptorWriter, 71
- pause
  - myLib::Clock, 50
- pickPhysicalDevice
  - ven::Device, 85
- PipelineConfigInfo
  - ven::PipelineConfigInfo, 125
- pipelineLayout
  - ven::PipelineConfigInfo, 127
- pointLight
  - ven::Object, 123
- PointLightPushConstants, 130
  - color, 130
  - position, 130
  - radius, 130
- pointLights
  - ven::GlobalUbo, 102
- PointLightSystem
  - ven::PointLightSystem, 133, 134
- populateDebugMessengerCreateInfo
  - ven::Device, 85
- position
  - PointLightPushConstants, 130
  - ven::Model::Vertex, 182
  - ven::PointLight, 128
- presentFamily
  - ven::QueueFamilyIndices, 138
- presentFamilyHasValue
  - ven::QueueFamilyIndices, 138
- presentModes
  - ven::SwapChainSupportDetails, 175
- presentQueue
  - ven::Device, 86
- projection
  - ven::GlobalUbo, 102
- properties\_
  - ven::Device, 89
- querySwapChainSupport
  - ven::Device, 86
- radius
  - PointLightPushConstants, 130
- RANDOM\_FLOAT\_MAX
  - myLib, 9
- RANDOM\_INT\_MAX
  - myLib, 10
- RANDOM\_INT\_MIN
  - myLib, 10
- randomFloat
  - myLib::Random, 140
- randomInt
  - myLib::Random, 140, 141
- rasterizationInfo
  - ven::PipelineConfigInfo, 127
- readFile
  - ven::Shaders, 161
- recreateSwapChain
  - ven::Renderer, 148
- RED
  - ven::Colors, 58
- RED\_V
  - ven::Colors, 59
- render
  - ven::ImGuiWindowManager, 106
  - ven::PointLightSystem, 135
- Renderer
  - ven::Renderer, 143, 144
- renderFinishedSemaphores
  - ven::SwapChain, 173
- renderObjects
  - ven::RenderSystem, 155
- renderPass
  - ven::PipelineConfigInfo, 127
  - ven::SwapChain, 173
- RenderSystem
  - ven::RenderSystem, 153, 154
- resetPool
  - ven::DescriptorPool, 64
- resetWindowResizedFlag
  - ven::Window, 186
- restart
  - myLib::Clock, 50
- resume
  - myLib::Clock, 50
- rotation
  - ven::Transform3DComponent, 179
- scale
  - ven::Transform3DComponent, 179
- setClearValue
  - ven::Renderer, 149
- setFar
  - ven::Camera, 45
- setFov
  - ven::Camera, 46
- setMaxSets

- ven::DescriptorPool::Builder, 36
- setNear
  - ven::Camera, 46
- setOrthographicProjection
  - ven::Camera, 46
- setPerspectiveProjection
  - ven::Camera, 47
- setPoolFlags
  - ven::DescriptorPool::Builder, 36
- setupDebugMessenger
  - ven::Device, 86
- setViewDirection
  - ven::Camera, 47
- setViewTarget
  - ven::Camera, 47
- setViewYXZ
  - ven::Camera, 47
- Shaders
  - ven::Shaders, 159
- SHADERS\_BIN\_PATH
  - ven, 19
- SILVER
  - ven::Colors, 59
- SILVER\_V
  - ven::Colors, 59
- SKY\_BLUE
  - ven::Colors, 59
- SKY\_BLUE\_V
  - ven::Colors, 59
- std, 10
- std::hash< ven::Model::Vertex >, 103
  - operator(), 103
- submitCommandBuffers
  - ven::SwapChain, 171
- subpass
  - ven::PipelineConfigInfo, 127
- SUNSET
  - ven::Colors, 59
- SUNSET\_V
  - ven::Colors, 59
- surface
  - ven::Device, 87
- SwapChain
  - ven::SwapChain, 166, 167
- swapChain
  - ven::SwapChain, 173
- swapChainDepthFormat
  - ven::SwapChain, 173
- swapChainFrameBuffers
  - ven::SwapChain, 173
- swapChainImageFormat
  - ven::SwapChain, 174
- swapChainImages
  - ven::SwapChain, 174
- swapChainImageViews
  - ven::SwapChain, 174
- TEAL
  - ven::Colors, 60
- TEAL\_V
  - ven::Colors, 60
- Time
  - myLib::Time, 177
- TimePoint
  - Clock.hpp, 235
- TINYOBJLOADER\_IMPLEMENTATION
  - model.cpp, 268
- transform3D
  - ven::Object, 124
- translation
  - ven::Transform3DComponent, 179
- unmap
  - ven::Buffer, 31
- update
  - ven::PointLightSystem, 136
- uv
  - ven::Model::Vertex, 182
- validationLayers
  - ven::Device, 89
- ven, 14
  - DEFAULT\_CLEAR\_COLOR, 16
  - DEFAULT\_CLEAR\_DEPTH, 16
  - DEFAULT\_FAR, 17
  - DEFAULT\_FOV, 17
  - DEFAULT\_HEIGHT, 17
  - DEFAULT\_LIGHT\_COLOR, 17
  - DEFAULT\_LIGHT\_INTENSITY, 17
  - DEFAULT\_LIGHT\_RADIUS, 17
  - DEFAULT\_LOOK\_SPEED, 17
  - DEFAULT\_MOVE\_SPEED, 18
  - DEFAULT\_NEAR, 18
  - DEFAULT\_POSITION, 18
  - DEFAULT\_ROTATION, 18
  - DEFAULT\_TITLE, 18
  - DEFAULT\_WIDTH, 18
  - hashCombine, 16
  - MAX\_LIGHTS, 19
  - SHADERS\_BIN\_PATH, 19
- ven::Buffer, 21
  - ~Buffer, 24
  - Buffer, 24
  - descriptorInfo, 24
  - descriptorInfoForIndex, 25
  - flush, 25
  - flushIndex, 26
  - getAlignment, 27
  - getAlignmentSize, 27
  - getBuffer, 27
  - getBufferSize, 28
  - getInstanceCount, 28
  - getInstanceSize, 28
  - getMappedMemory, 28
  - getMemoryPropertyFlags, 28
  - getUsageFlags, 28
  - invalidate, 29
  - invalidateIndex, 29

- m\_alignmentSize, 32
- m\_buffer, 32
- m\_bufferSize, 32
- m\_device, 32
- m\_instanceCount, 32
- m\_instanceSize, 33
- m\_mapped, 33
- m\_memory, 33
- m\_memoryPropertyFlags, 33
- m\_usageFlags, 33
- map, 30
- operator=, 30
- unmap, 31
- writeToBuffer, 31
- writeToIndex, 31
- ven::Camera, 43
  - getFar, 44
  - getFov, 44
  - getInverseView, 44
  - getNear, 45
  - getProjection, 45
  - getView, 45
  - m\_far, 48
  - m\_fov, 48
  - m\_inverseViewMatrix, 48
  - m\_near, 48
  - m\_projectionMatrix, 48
  - m\_viewMatrix, 48
  - setFar, 45
  - setFov, 46
  - setNear, 46
  - setOrthographicProjection, 46
  - setPerspectiveProjection, 47
  - setViewDirection, 47
  - setViewTarget, 47
  - setViewYXZ, 47
- ven::Colors, 52
  - AQUA, 54
  - AQUA\_V, 54
  - BLACK, 54
  - BLACK\_V, 54
  - BLUE, 54
  - BLUE\_V, 54
  - CLEAR\_COLORS, 54
  - COLOR\_MAX, 55
  - COLORS, 55
  - CYAN, 55
  - CYAN\_V, 56
  - FUCHSIA, 56
  - FUCHSIA\_V, 56
  - GRAY, 56
  - GRAY\_V, 56
  - GREEN, 56
  - GREEN\_V, 56
  - LIME, 57
  - LIME\_V, 57
  - MAGENTA, 57
  - MAGENTA\_V, 57
- MAROON, 57
- MAROON\_V, 57
- NAVY, 57
- NAVY\_V, 58
- NIGHT\_BLUE, 58
- NIGHT\_BLUE\_V, 58
- NIGHT\_MODE\_V, 58
- OLIVE, 58
- OLIVE\_V, 58
- RED, 58
- RED\_V, 59
- SILVER, 59
- SILVER\_V, 59
- SKY\_BLUE, 59
- SKY\_BLUE\_V, 59
- SUNSET, 59
- SUNSET\_V, 59
- TEAL, 60
- TEAL\_V, 60
- WHITE, 60
- WHITE\_V, 60
- YELLOW, 60
- YELLOW\_V, 60
- ven::DescriptorPool, 61
  - ~DescriptorPool, 62
  - allocateDescriptor, 63
  - DescriptorPool, 62, 63
  - DescriptorWriter, 64
  - freeDescriptors, 63
  - getDescriptorPool, 63
  - m\_descriptorPool, 64
  - m\_device, 64
  - operator=, 64
  - resetPool, 64
- ven::DescriptorPool::Builder, 34
  - addPoolSize, 35
  - build, 35
  - Builder, 35
  - m\_device, 37
  - m\_maxSets, 37
  - m\_poolFlags, 37
  - m\_poolSizes, 37
  - setMaxSets, 36
  - setPoolFlags, 36
- ven::DescriptorSetLayout, 65
  - ~DescriptorSetLayout, 67
  - DescriptorSetLayout, 67, 68
  - DescriptorWriter, 68
  - getDescriptorSetLayout, 68
  - m\_bindings, 69
  - m\_descriptorSetLayout, 69
  - m\_device, 69
  - operator=, 68
- ven::DescriptorSetLayout::Builder, 38
  - addBinding, 39
  - build, 39
  - Builder, 39
  - m\_bindings, 40



- m\_device, 40
- ven::DescriptorWriter, 69
  - build, 71
  - DescriptorWriter, 71
  - m\_pool, 72
  - m\_setLayout, 72
  - m\_writes, 72
  - overwrite, 71
  - writeBuffer, 71
  - writelnImage, 72
- ven::Device, 73
  - ~Device, 76
  - beginSingleTimeCommands, 77
  - checkDeviceExtensionSupport, 77
  - checkValidationLayerSupport, 77
  - copyBuffer, 77
  - copyBufferToImage, 78
  - createBuffer, 78
  - createCommandPool, 78
  - createImageWithInfo, 79
  - createInstance, 79
  - createLogicalDevice, 79
  - createSurface, 80
  - Device, 76, 77
  - device, 80
  - deviceExtensions, 87
  - enableValidationLayers, 87
  - endSingleTimeCommands, 81
  - findMemoryType, 81
  - findPhysicalQueueFamilies, 82
  - findQueueFamilies, 82
  - findSupportedFormat, 82
  - getCommandPool, 83
  - getGraphicsQueue, 83
  - getPhysicalDevice, 83
  - getRequiredExtensions, 83
  - getSwapChainSupport, 84
  - graphicsQueue, 84
  - hasGlfwRequiredInstanceExtensions, 84
  - isDeviceSuitable, 84
  - m\_commandPool, 88
  - m\_debugMessenger, 88
  - m\_device, 88
  - m\_graphicsQueue, 88
  - m\_instance, 88
  - m\_physicalDevice, 88
  - m\_presentQueue, 89
  - m\_surface, 89
  - m\_window, 89
  - operator=, 85
  - pickPhysicalDevice, 85
  - populateDebugMessengerCreateInfo, 85
  - presentQueue, 86
  - properties\_, 89
  - querySwapChainSupport, 86
  - setupDebugMessenger, 86
  - surface, 87
  - validationLayers, 89
- ven::Engine, 90
  - ~Engine, 92
  - createInstance, 92
  - createSurface, 92
  - Engine, 91, 92
  - loadObjects, 93
  - m\_device, 95
  - m\_globalPool, 95
  - m\_instance, 95
  - m\_objects, 95
  - m\_renderer, 95
  - m\_surface, 96
  - m\_window, 96
  - mainLoop, 94
  - operator=, 95
- ven::FrameInfo, 96
  - camera, 98
  - commandBuffer, 98
  - frameIndex, 98
  - frameTime, 98
  - globalDescriptorSet, 98
  - objects, 98
- ven::GlobalUbo, 101
  - ambientLightColor, 102
  - inverseView, 102
  - numLights, 102
  - pointLights, 102
  - projection, 102
  - view, 102
- ven::ImGuiWindowManager, 104
  - ~ImGuiWindowManager, 105
  - cleanup, 105
  - ImGuiWindowManager, 105
  - init, 105
  - operator=, 106
  - render, 106
- ven::ImGuiWindowManager::funcs, 99
  - IsLegacyNativeDupe, 100
- ven::KeyboardController, 108
  - m\_keys, 109
  - m\_lookSpeed, 109
  - m\_moveSpeed, 109
  - moveInPlaneXZ, 109
- ven::KeyboardController::KeyMappings, 110
  - lookDown, 111
  - lookLeft, 111
  - lookRight, 111
  - lookUp, 111
  - moveBackward, 111
  - moveDown, 111
  - moveForward, 112
  - moveLeft, 112
  - moveRight, 112
  - moveUp, 112
- ven::Model, 113
  - ~Model, 114
  - bind, 115
  - createIndexBuffer, 115

- createModelFromFile, 115
- createVertexBuffer, 116
- draw, 117
- m\_device, 117
- m\_hasIndexBuffer, 117
- m\_indexBuffer, 117
- m\_indexCount, 117
- m\_vertexBuffer, 118
- m\_vertexCount, 118
- Model, 114, 115
- operator=, 117
- ven::Model::Builder, 41
  - indices, 42
  - loadModel, 42
  - vertices, 42
- ven::Model::Vertex, 180
  - color, 182
  - getAttributeDescriptions, 181
  - getBindingDescriptions, 181
  - normal, 182
  - operator==, 181
  - position, 182
  - uv, 182
- ven::Object, 118
  - ~Object, 120
  - color, 123
  - createObject, 121
  - getId, 122
  - m\_objId, 123
  - makePointLight, 122
  - Map, 120
  - model, 123
  - name, 123
  - Object, 120, 121
  - operator=, 122, 123
  - pointLight, 123
  - transform3D, 124
- ven::PipelineConfigInfo, 124
  - attributeDescriptions, 126
  - bindingDescriptions, 126
  - colorBlendAttachment, 126
  - colorBlendInfo, 126
  - depthStencilInfo, 126
  - dynamicStateEnables, 126
  - dynamicStateInfo, 126
  - inputAssemblyInfo, 127
  - multisampleInfo, 127
  - operator=, 125
  - PipelineConfigInfo, 125
  - pipelineLayout, 127
  - rasterizationInfo, 127
  - renderPass, 127
  - subpass, 127
- ven::PointLight, 128
  - color, 128
  - position, 128
- ven::PointLightComponent, 129
  - lightIntensity, 129
- ven::PointLightSystem, 131
  - ~PointLightSystem, 133
  - createPipeline, 134
  - createPipelineLayout, 135
  - m\_device, 136
  - m\_pipelineLayout, 136
  - m\_shaders, 136
  - operator=, 135
  - PointLightSystem, 133, 134
  - render, 135
  - update, 136
- ven::QueueFamilyIndices, 137
  - graphicsFamily, 138
  - graphicsFamilyHasValue, 138
  - isComplete, 138
  - presentFamily, 138
  - presentFamilyHasValue, 138
- ven::Renderer, 142
  - ~Renderer, 144
  - beginFrame, 144
  - beginSwapChainRenderPass, 144
  - createCommandBuffers, 144
  - endFrame, 145
  - endSwapChainRenderPass, 145
  - freeCommandBuffers, 145
  - getAspectRatio, 146
  - getClearColor, 146
  - getCurrentCommandBuffer, 146
  - getFrameIndex, 147
  - getSwapChainRenderPass, 147
  - isFrameInProgress, 148
  - m\_clearValues, 149
  - m\_commandBuffers, 149
  - m\_currentFrameIndex, 150
  - m\_currentImageIndex, 150
  - m\_device, 150
  - m\_isFrameStarted, 150
  - m\_swapChain, 150
  - m\_window, 150
  - operator=, 148
  - recreateSwapChain, 148
  - Renderer, 143, 144
  - setClearValue, 149
- ven::RenderSystem, 151
  - ~RenderSystem, 153
  - createPipeline, 154
  - createPipelineLayout, 155
  - m\_device, 156
  - m\_pipelineLayout, 156
  - m\_shaders, 156
  - operator=, 155
  - renderObjects, 155
  - RenderSystem, 153, 154
- ven::Shaders, 157
  - ~Shaders, 159
  - bind, 160
  - createGraphicsPipeline, 160
  - createShaderModule, 160

- defaultPipelineConfigInfo, 160
- m\_device, 162
- m\_fragShaderModule, 162
- m\_graphicsPipeline, 162
- m\_vertShaderModule, 162
- operator=, 161
- readFile, 161
- Shaders, 159
- ven::SimplePushConstantData, 162
  - modelMatrix, 163
  - normalMatrix, 163
- ven::SwapChain, 163
  - ~SwapChain, 166
  - acquireNextImage, 167
  - chooseSwapExtent, 167
  - chooseSwapPresentMode, 167
  - chooseSwapSurfaceFormat, 167
  - compareSwapFormats, 168
  - createDepthResources, 168
  - createFrameBuffers, 168
  - createImageViews, 168
  - createRenderPass, 168
  - createSwapChain, 168
  - createSyncObjects, 168
  - currentFrame, 171
  - depthImageMemory, 171
  - depthImages, 171
  - depthImageViews, 171
  - device, 172
  - extentAspectRatio, 169
  - findDepthFormat, 169
  - getFrameBuffer, 169
  - getImageView, 169
  - getRenderPass, 169
  - getSwapChainExtent, 169
  - getSwapChainImageFormat, 170
  - height, 170
  - imageAvailableSemaphores, 172
  - imageCount, 170
  - imagesInFlight, 172
  - inFlightFences, 172
  - init, 170
  - m\_swapChainExtent, 172
  - MAX\_FRAMES\_IN\_FLIGHT, 172
  - oldSwapChain, 173
  - operator=, 170
  - renderFinishedSemaphores, 173
  - renderPass, 173
  - submitCommandBuffers, 171
  - SwapChain, 166, 167
  - swapChain, 173
  - swapChainDepthFormat, 173
  - swapChainFrameBuffers, 173
  - swapChainImageFormat, 174
  - swapChainImages, 174
  - swapChainImageViews, 174
  - width, 171
  - windowExtent, 174
- ven::SwapChainSupportDetails, 175
  - capabilities, 175
  - formats, 175
  - presentModes, 175
- ven::Transform3DComponent, 178
  - mat4, 179
  - normalMatrix, 179
  - rotation, 179
  - scale, 179
  - translation, 179
- ven::Window, 183
  - ~Window, 184
  - createWindow, 184
  - createWindowSurface, 184
  - framebufferResizeCallback, 185
  - getExtent, 185
  - getGLFWWindow, 185
  - m\_framebufferResized, 186
  - m\_height, 186
  - m\_width, 187
  - m\_window, 187
  - resetWindowResizedFlag, 186
  - wasWindowResized, 186
  - Window, 184
- vengine, 1
- vertices
  - ven::Model::Builder, 42
- view
  - ven::GlobalUbo, 102
- wasWindowResized
  - ven::Window, 186
- WHITE
  - ven::Colors, 60
- WHITE\_V
  - ven::Colors, 60
- width
  - ven::SwapChain, 171
- Window
  - ven::Window, 184
- Window.hpp
  - GLFW\_INCLUDE\_VULKAN, 233
- windowExtent
  - ven::SwapChain, 174
- writeBuffer
  - ven::DescriptorWriter, 71
- writelImage
  - ven::DescriptorWriter, 72
- writeToBuffer
  - ven::Buffer, 31
- writeToIndex
  - ven::Buffer, 31
- YELLOW
  - ven::Colors, 60
- YELLOW\_V
  - ven::Colors, 60