

vengine

0.1.0

Generated by Doxygen 1.11.0

1 vengine	1
1.1 VEngine - Vulkan Graphics Engine	1
1.1.1 Features	1
1.1.1.1 Planned Features:	1
1.1.2 Prerequisites	2
1.1.3 Usage	2
1.1.3.1 Build	2
1.1.3.2 Run	2
1.1.4 Key Bindings	2
1.1.5 Documentation	2
1.1.6 Commit Norms	2
1.1.7 License	3
1.1.8 Acknowledgements	3
2 Namespace Index	5
2.1 Namespace List	5
3 Hierarchical Index	7
3.1 Class Hierarchy	7
4 Class Index	9
4.1 Class List	9
5 File Index	11
5.1 File List	11
6 Namespace Documentation	13
6.1 myLib Namespace Reference	13
6.1.1 Variable Documentation	13
6.1.1.1 MICROSECONDS_PER_SECOND	13
6.1.1.2 MILLISECONDS_PER_SECOND	13
6.1.1.3 RANDOM_FLOAT_MAX	14
6.1.1.4 RANDOM_INT_MAX	14
6.1.1.5 RANDOM_INT_MIN	14
6.2 ven Namespace Reference	14
6.2.1 Function Documentation	16
6.2.1.1 hashCombine()	16
6.2.2 Variable Documentation	16
6.2.2.1 DEFAULT_AMBIENT_LIGHT_COLOR	16
6.2.2.2 DEFAULT_AMBIENT_LIGHT_INTENSITY	17
6.2.2.3 DEFAULT_CLEAR_COLOR	17
6.2.2.4 DEFAULT_CLEAR_DEPTH	17
6.2.2.5 DEFAULT_FAR	17
6.2.2.6 DEFAULT_FOV	17

6.2.2.7 DEFAULT_HEIGHT	17
6.2.2.8 DEFAULT_LIGHT_COLOR	17
6.2.2.9 DEFAULT_LIGHT_INTENSITY	18
6.2.2.10 DEFAULT_LIGHT_RADIUS	18
6.2.2.11 DEFAULT_LOOK_SPEED	18
6.2.2.12 DEFAULT_MOVE_SPEED	18
6.2.2.13 DEFAULT_NEAR	18
6.2.2.14 DEFAULT_POSITION	18
6.2.2.15 DEFAULT_ROTATION	19
6.2.2.16 DEFAULT_TITLE	19
6.2.2.17 DEFAULT_WIDTH	19
6.2.2.18 MAX_LIGHTS	19
6.2.2.19 SHADERS_BIN_PATH	19
7 Class Documentation	21
7.1 ven::ARenderSystemBase Class Reference	21
7.1.1 Detailed Description	23
7.1.2 Constructor & Destructor Documentation	23
7.1.2.1 ARenderSystemBase()	23
7.1.2.2 ~ARenderSystemBase()	23
7.1.3 Member Function Documentation	24
7.1.3.1 createPipeline()	24
7.1.3.2 createPipelineLayout()	24
7.1.3.3 getDevice()	25
7.1.3.4 getPipelineLayout()	25
7.1.3.5 getShaders()	26
7.1.4 Member Data Documentation	26
7.1.4.1 m_device	26
7.1.4.2 m_pipelineLayout	26
7.1.4.3 m_shaders	26
7.2 ven::Buffer Class Reference	27
7.2.1 Detailed Description	29
7.2.2 Constructor & Destructor Documentation	29
7.2.2.1 Buffer() [1/2]	29
7.2.2.2 ~Buffer()	29
7.2.2.3 Buffer() [2/2]	29
7.2.3 Member Function Documentation	29
7.2.3.1 descriptorInfo()	29
7.2.3.2 descriptorInfoForIndex()	30
7.2.3.3 flush()	31
7.2.3.4 flushIndex()	31
7.2.3.5 getAlignment()	32

7.2.3.6 getAlignmentSize()	32
7.2.3.7 getBuffer()	33
7.2.3.8 getBufferSize()	33
7.2.3.9 getInstanceCount()	33
7.2.3.10 getInstanceSize()	33
7.2.3.11 getMappedMemory()	33
7.2.3.12 getMemoryPropertyFlags()	33
7.2.3.13 getUsageFlags()	34
7.2.3.14 invalidate()	34
7.2.3.15 invalidateIndex()	34
7.2.3.16 map()	35
7.2.3.17 operator=()	36
7.2.3.18 unmap()	36
7.2.3.19 writeToBuffer()	36
7.2.3.20 writeToIndex()	36
7.2.4 Member Data Documentation	37
7.2.4.1 m_alignmentSize	37
7.2.4.2 m_buffer	37
7.2.4.3 m_bufferSize	37
7.2.4.4 m_device	38
7.2.4.5 m_instanceCount	38
7.2.4.6 m_instanceSize	38
7.2.4.7 m_mapped	38
7.2.4.8 m_memory	38
7.2.4.9 m_memoryPropertyFlags	38
7.2.4.10 m_usageFlags	39
7.3 ven::DescriptorPool::Builder Class Reference	39
7.3.1 Detailed Description	41
7.3.2 Constructor & Destructor Documentation	41
7.3.2.1 Builder()	41
7.3.3 Member Function Documentation	41
7.3.3.1 addPoolSize()	41
7.3.3.2 build()	42
7.3.3.3 setMaxSets()	42
7.3.3.4 setPoolFlags()	42
7.3.4 Member Data Documentation	43
7.3.4.1 m_device	43
7.3.4.2 m_maxSets	43
7.3.4.3 m_poolFlags	43
7.3.4.4 m_poolSizes	43
7.4 ven::DescriptorSetLayout::Builder Class Reference	44
7.4.1 Detailed Description	45

7.4.2 Constructor & Destructor Documentation	45
7.4.2.1 Builder()	45
7.4.3 Member Function Documentation	45
7.4.3.1 addBinding()	45
7.4.3.2 build()	46
7.4.4 Member Data Documentation	46
7.4.4.1 m_bindings	46
7.4.4.2 m_device	46
7.5 ven::Model::Builder Struct Reference	47
7.5.1 Detailed Description	47
7.5.2 Member Function Documentation	48
7.5.2.1 loadModel()	48
7.5.3 Member Data Documentation	48
7.5.3.1 indices	48
7.5.3.2 vertices	48
7.6 ven::Camera Class Reference	49
7.6.1 Detailed Description	50
7.6.2 Member Function Documentation	50
7.6.2.1 getFar()	50
7.6.2.2 getFov()	51
7.6.2.3 getInverseView()	51
7.6.2.4 getNear()	51
7.6.2.5 getProjection()	52
7.6.2.6 getView()	52
7.6.2.7 setFar()	52
7.6.2.8 setFov()	52
7.6.2.9 setNear()	53
7.6.2.10 setOrthographicProjection()	53
7.6.2.11 setPerspectiveProjection()	53
7.6.2.12 setViewDirection()	53
7.6.2.13 setViewTarget()	54
7.6.2.14 setViewYXZ()	54
7.6.3 Member Data Documentation	54
7.6.3.1 m_far	54
7.6.3.2 m_fov	54
7.6.3.3 m_inverseViewMatrix	54
7.6.3.4 m_near	54
7.6.3.5 m_projectionMatrix	55
7.6.3.6 m_viewMatrix	55
7.7 myLib::Clock Class Reference	55
7.7.1 Detailed Description	56
7.7.2 Constructor & Destructor Documentation	56

7.7.2.1 Clock()	56
7.7.2.2 ~Clock()	56
7.7.3 Member Function Documentation	56
7.7.3.1 getElapsedTime()	56
7.7.3.2 pause()	57
7.7.3.3 restart()	57
7.7.3.4 resume()	57
7.7.4 Member Data Documentation	57
7.7.4.1 m_pause	57
7.7.4.2 m_paused	57
7.7.4.3 m_start	58
7.8 ven::Colors Class Reference	58
7.8.1 Detailed Description	60
7.8.2 Member Data Documentation	61
7.8.2.1 AQUA	61
7.8.2.2 AQUA_V	61
7.8.2.3 BLACK	61
7.8.2.4 BLACK_V	61
7.8.2.5 BLUE	61
7.8.2.6 BLUE_V	61
7.8.2.7 CLEAR_COLORS	62
7.8.2.8 COLOR_MAX	62
7.8.2.9 COLORS	62
7.8.2.10 CYAN	63
7.8.2.11 CYAN_V	63
7.8.2.12 FUCHSIA	63
7.8.2.13 FUCHSIA_V	63
7.8.2.14 GRAY	63
7.8.2.15 GRAY_V	63
7.8.2.16 GREEN	63
7.8.2.17 GREEN_V	64
7.8.2.18 LIME	64
7.8.2.19 LIME_V	64
7.8.2.20 MAGENTA	64
7.8.2.21 MAGENTA_V	64
7.8.2.22 MAROON	64
7.8.2.23 MAROON_V	64
7.8.2.24 NAVY	65
7.8.2.25 NAVY_V	65
7.8.2.26 NIGHT_BLUE	65
7.8.2.27 NIGHT_BLUE_V	65
7.8.2.28 NIGHT_MODE_V	65

7.8.2.29 OLIVE	65
7.8.2.30 OLIVE_V	65
7.8.2.31 RED	66
7.8.2.32 RED_V	66
7.8.2.33 SILVER	66
7.8.2.34 SILVER_V	66
7.8.2.35 SKY_BLUE	66
7.8.2.36 SKY_BLUE_V	66
7.8.2.37 SUNSET	66
7.8.2.38 SUNSET_V	67
7.8.2.39 TEAL	67
7.8.2.40 TEAL_V	67
7.8.2.41 WHITE	67
7.8.2.42 WHITE_V	67
7.8.2.43 YELLOW	67
7.8.2.44 YELLOW_V	67
7.9 ven::DescriptorPool Class Reference	68
7.9.1 Detailed Description	69
7.9.2 Constructor & Destructor Documentation	69
7.9.2.1 DescriptorPool() [1/2]	69
7.9.2.2 ~DescriptorPool()	70
7.9.2.3 DescriptorPool() [2/2]	70
7.9.3 Member Function Documentation	70
7.9.3.1 allocateDescriptor()	70
7.9.3.2 freeDescriptors()	70
7.9.3.3 getDescriptorPool()	71
7.9.3.4 operator=()	71
7.9.3.5 resetPool()	71
7.9.4 Friends And Related Symbol Documentation	71
7.9.4.1 DescriptorWriter	71
7.9.5 Member Data Documentation	71
7.9.5.1 m_descriptorPool	71
7.9.5.2 m_device	72
7.10 ven::DescriptorSetLayout Class Reference	72
7.10.1 Detailed Description	74
7.10.2 Constructor & Destructor Documentation	74
7.10.2.1 DescriptorSetLayout() [1/2]	74
7.10.2.2 ~DescriptorSetLayout()	75
7.10.2.3 DescriptorSetLayout() [2/2]	75
7.10.3 Member Function Documentation	75
7.10.3.1 getDescriptorSetLayout()	75
7.10.3.2 operator=()	75

7.10.4 Friends And Related Symbol Documentation	75
7.10.4.1 DescriptorWriter	75
7.10.5 Member Data Documentation	76
7.10.5.1 m_bindings	76
7.10.5.2 m_descriptorSetLayout	76
7.10.5.3 m_device	76
7.11 ven::DescriptorWriter Class Reference	76
7.11.1 Detailed Description	78
7.11.2 Constructor & Destructor Documentation	78
7.11.2.1 DescriptorWriter()	78
7.11.3 Member Function Documentation	78
7.11.3.1 build()	78
7.11.3.2 overwrite()	78
7.11.3.3 writeBuffer()	79
7.11.3.4 writeImage()	79
7.11.4 Member Data Documentation	79
7.11.4.1 m_pool	79
7.11.4.2 m_setLayout	79
7.11.4.3 m_writes	80
7.12 ven::Device Class Reference	80
7.12.1 Detailed Description	83
7.12.2 Constructor & Destructor Documentation	83
7.12.2.1 Device() [1/3]	83
7.12.2.2 ~Device()	84
7.12.2.3 Device() [2/3]	84
7.12.2.4 Device() [3/3]	84
7.12.3 Member Function Documentation	84
7.12.3.1 beginSingleTimeCommands()	84
7.12.3.2 checkDeviceExtensionSupport()	84
7.12.3.3 checkValidationLayerSupport()	84
7.12.3.4 copyBuffer()	85
7.12.3.5 copyBufferToImage()	85
7.12.3.6 createBuffer()	85
7.12.3.7 createCommandPool()	86
7.12.3.8 createImageWithInfo()	86
7.12.3.9 createInstance()	86
7.12.3.10 createLogicalDevice()	87
7.12.3.11 createSurface()	87
7.12.3.12 device()	88
7.12.3.13 endSingleTimeCommands()	88
7.12.3.14 findMemoryType()	89
7.12.3.15 findPhysicalQueueFamilies()	89

7.12.3.16 findQueueFamilies()	89
7.12.3.17 findSupportedFormat()	90
7.12.3.18 getCommandPool()	90
7.12.3.19 getGraphicsQueue()	90
7.12.3.20 getPhysicalDevice()	90
7.12.3.21 getRequiredExtensions()	91
7.12.3.22 getSwapChainSupport()	91
7.12.3.23 graphicsQueue()	91
7.12.3.24 hasGlfwRequiredInstanceExtensions()	91
7.12.3.25 isDeviceSuitable()	92
7.12.3.26 operator=() [1/2]	92
7.12.3.27 operator=() [2/2]	92
7.12.3.28 pickPhysicalDevice()	92
7.12.3.29 populateDebugMessengerCreateInfo()	93
7.12.3.30 presentQueue()	93
7.12.3.31 querySwapChainSupport()	93
7.12.3.32 setupDebugMessenger()	94
7.12.3.33 surface()	94
7.12.4 Member Data Documentation	94
7.12.4.1 deviceExtensions	94
7.12.4.2 enableValidationLayers	95
7.12.4.3 m_commandPool	95
7.12.4.4 m_debugMessenger	95
7.12.4.5 m_device	95
7.12.4.6 m_graphicsQueue	95
7.12.4.7 m_instance	95
7.12.4.8 m_physicalDevice	96
7.12.4.9 m_presentQueue	96
7.12.4.10 m_properties	96
7.12.4.11 m_surface	96
7.12.4.12 m_window	96
7.12.4.13 validationLayers	96
7.13 ven::Engine Class Reference	97
7.13.1 Detailed Description	98
7.13.2 Constructor & Destructor Documentation	98
7.13.2.1 Engine() [1/2]	98
7.13.2.2 ~Engine()	99
7.13.2.3 Engine() [2/2]	99
7.13.3 Member Function Documentation	99
7.13.3.1 createInstance()	99
7.13.3.2 createSurface()	100
7.13.3.3 loadObjects()	100

7.13.3.4 mainLoop()	101
7.13.3.5 operator=()	102
7.13.4 Member Data Documentation	102
7.13.4.1 m_device	102
7.13.4.2 m_globalPool	103
7.13.4.3 m_instance	103
7.13.4.4 m_lights	103
7.13.4.5 m_objects	103
7.13.4.6 m_renderer	103
7.13.4.7 m_surface	103
7.13.4.8 m_window	104
7.14 ven::FrameInfo Struct Reference	104
7.14.1 Detailed Description	106
7.14.2 Member Data Documentation	106
7.14.2.1 camera	106
7.14.2.2 commandBuffer	106
7.14.2.3 frameIndex	106
7.14.2.4 frameTime	106
7.14.2.5 globalDescriptorSet	106
7.14.2.6 lights	107
7.14.2.7 objects	107
7.15 ven::ImGuiWindowManager::funcs Struct Reference	107
7.15.1 Detailed Description	107
7.15.2 Member Function Documentation	108
7.15.2.1 IsLegacyNativeDupe()	108
7.16 ven::GlobalUbo Struct Reference	109
7.16.1 Detailed Description	109
7.16.2 Member Data Documentation	110
7.16.2.1 ambientLightColor	110
7.16.2.2 inverseView	110
7.16.2.3 numLights	110
7.16.2.4 pointLights	110
7.16.2.5 projection	110
7.16.2.6 view	110
7.17 std::hash< ven::Model::Vertex > Struct Reference	111
7.17.1 Detailed Description	111
7.17.2 Member Function Documentation	111
7.17.2.1 operator>()	111
7.18 ven::ImGuiWindowManager Class Reference	112
7.18.1 Detailed Description	113
7.18.2 Constructor & Destructor Documentation	113
7.18.2.1 ImGuiWindowManager() [1/2]	113

7.18.2.2 ~ImGuiWindowManager()	113
7.18.2.3 ImGuiWindowManager() [2/2]	113
7.18.3 Member Function Documentation	113
7.18.3.1 cameraSection()	113
7.18.3.2 cleanup()	114
7.18.3.3 devicePropertiesSection()	114
7.18.3.4 init()	115
7.18.3.5 initStyle()	115
7.18.3.6 inputsSection()	116
7.18.3.7 lightsSection()	116
7.18.3.8 objectsSection()	116
7.18.3.9 operator=()	116
7.18.3.10 render()	117
7.18.3.11 rendererSection()	117
7.18.3.12 renderFrameWindow()	118
7.19 ven::KeyboardController Class Reference	118
7.19.1 Detailed Description	120
7.19.2 Member Function Documentation	120
7.19.2.1 moveInPlaneXZ() [1/2]	120
7.19.2.2 moveInPlaneXZ() [2/2]	120
7.19.3 Member Data Documentation	120
7.19.3.1 m_keys	120
7.19.3.2 m_lookSpeed	120
7.19.3.3 m_moveSpeed	121
7.20 ven::KeyboardController::KeyMappings Struct Reference	121
7.20.1 Detailed Description	122
7.20.2 Member Data Documentation	122
7.20.2.1 lookDown	122
7.20.2.2 lookLeft	122
7.20.2.3 lookRight	122
7.20.2.4 lookUp	122
7.20.2.5 moveBackward	122
7.20.2.6 moveDown	123
7.20.2.7 moveForward	123
7.20.2.8 moveLeft	123
7.20.2.9 moveRight	123
7.20.2.10 moveUp	123
7.21 ven::Light Class Reference	124
7.21.1 Detailed Description	125
7.21.2 Member Typedef Documentation	125
7.21.2.1 Map	125
7.21.3 Constructor & Destructor Documentation	125

7.21.3.1 ~Light()	125
7.21.3.2 Light() [1/3]	126
7.21.3.3 Light() [2/3]	126
7.21.3.4 Light() [3/3]	126
7.21.4 Member Function Documentation	126
7.21.4.1 createLight()	126
7.21.4.2 getId()	126
7.21.4.3 getName()	127
7.21.4.4 operator=() [1/2]	127
7.21.4.5 operator=() [2/2]	127
7.21.4.6 setName()	127
7.21.5 Member Data Documentation	127
7.21.5.1 color	127
7.21.5.2 m_lightId	127
7.21.5.3 m_name	128
7.21.5.4 transform3D	128
7.22 ven::LightPushConstantData Struct Reference	128
7.22.1 Detailed Description	129
7.22.2 Member Data Documentation	129
7.22.2.1 color	129
7.22.2.2 position	129
7.22.2.3 radius	129
7.23 ven::Model Class Reference	129
7.23.1 Detailed Description	131
7.23.2 Constructor & Destructor Documentation	131
7.23.2.1 Model() [1/2]	131
7.23.2.2 ~Model()	132
7.23.2.3 Model() [2/2]	132
7.23.3 Member Function Documentation	132
7.23.3.1 bind()	132
7.23.3.2 createIndexBuffer()	132
7.23.3.3 createModelFromFile()	133
7.23.3.4 createVertexBuffer()	133
7.23.3.5 draw()	134
7.23.3.6 operator=()	134
7.23.4 Member Data Documentation	134
7.23.4.1 m_device	134
7.23.4.2 m_hasIndexBuffer	134
7.23.4.3 m_indexBuffer	134
7.23.4.4 m_indexCount	135
7.23.4.5 m_vertexBuffer	135
7.23.4.6 m_vertexCount	135

7.24 ven::Object Class Reference	135
7.24.1 Detailed Description	137
7.24.2 Member Typedef Documentation	137
7.24.2.1 Map	137
7.24.3 Constructor & Destructor Documentation	137
7.24.3.1 ~Object()	137
7.24.3.2 Object() [1/3]	138
7.24.3.3 Object() [2/3]	138
7.24.3.4 Object() [3/3]	138
7.24.4 Member Function Documentation	138
7.24.4.1 createObject()	138
7.24.4.2 getId()	139
7.24.4.3 getModel()	139
7.24.4.4 getName()	139
7.24.4.5 operator=() [1/2]	139
7.24.4.6 operator=() [2/2]	140
7.24.4.7 setModel()	140
7.24.4.8 setName()	140
7.24.5 Member Data Documentation	140
7.24.5.1 color	140
7.24.5.2 m_model	141
7.24.5.3 m_name	141
7.24.5.4 m_objId	141
7.24.5.5 transform3D	141
7.25 ven::ObjectPushConstantData Struct Reference	141
7.25.1 Detailed Description	142
7.25.2 Member Data Documentation	142
7.25.2.1 modelMatrix	142
7.25.2.2 normalMatrix	142
7.26 ven::PipelineConfigInfo Struct Reference	142
7.26.1 Detailed Description	143
7.26.2 Constructor & Destructor Documentation	143
7.26.2.1 PipelineConfigInfo() [1/2]	143
7.26.2.2 PipelineConfigInfo() [2/2]	143
7.26.3 Member Function Documentation	143
7.26.3.1 operator=()	143
7.26.4 Member Data Documentation	144
7.26.4.1 attributeDescriptions	144
7.26.4.2 bindingDescriptions	144
7.26.4.3 colorBlendAttachment	144
7.26.4.4 colorBlendInfo	144
7.26.4.5 depthStencilInfo	144

7.26.4.6 dynamicStateEnables	144
7.26.4.7 dynamicStateInfo	145
7.26.4.8 inputAssemblyInfo	145
7.26.4.9 multisampleInfo	145
7.26.4.10 pipelineLayout	145
7.26.4.11 rasterizationInfo	145
7.26.4.12 renderPass	145
7.26.4.13 subpass	146
7.27 ven::PointLightData Struct Reference	146
7.27.1 Detailed Description	146
7.27.2 Member Data Documentation	146
7.27.2.1 color	146
7.27.2.2 position	147
7.28 ven::PointLightSystem Class Reference	147
7.28.1 Detailed Description	149
7.28.2 Constructor & Destructor Documentation	149
7.28.2.1 PointLightSystem()	149
7.28.3 Member Function Documentation	150
7.28.3.1 render()	150
7.28.3.2 update()	150
7.29 ven::QueueFamilyIndices Struct Reference	151
7.29.1 Detailed Description	152
7.29.2 Member Function Documentation	152
7.29.2.1 isComplete()	152
7.29.3 Member Data Documentation	152
7.29.3.1 graphicsFamily	152
7.29.3.2 graphicsFamilyHasValue	152
7.29.3.3 presentFamily	152
7.29.3.4 presentFamilyHasValue	153
7.30 myLib::Random Class Reference	153
7.30.1 Detailed Description	153
7.30.2 Member Function Documentation	154
7.30.2.1 randomFloat() [1/2]	154
7.30.2.2 randomFloat() [2/2]	154
7.30.2.3 randomInt() [1/2]	155
7.30.2.4 randomInt() [2/2]	155
7.31 ven::Renderer Class Reference	156
7.31.1 Detailed Description	157
7.31.2 Constructor & Destructor Documentation	157
7.31.2.1 Renderer() [1/2]	157
7.31.2.2 ~Renderer()	158
7.31.2.3 Renderer() [2/2]	158

7.31.3 Member Function Documentation	158
7.31.3.1 beginFrame()	158
7.31.3.2 beginSwapChainRenderPass()	158
7.31.3.3 createCommandBuffers()	159
7.31.3.4 endFrame()	159
7.31.3.5 endSwapChainRenderPass()	159
7.31.3.6 freeCommandBuffers()	160
7.31.3.7 getAspectRatio()	160
7.31.3.8 getClearColor()	160
7.31.3.9 getCurrentCommandBuffer()	161
7.31.3.10 getFrameIndex()	161
7.31.3.11 getSwapChainRenderPass()	162
7.31.3.12 getWindow()	162
7.31.3.13 isFrameInProgress()	162
7.31.3.14 operator=()	163
7.31.3.15 recreateSwapChain()	163
7.31.3.16 setClearColor()	163
7.31.4 Member Data Documentation	164
7.31.4.1 m_clearValues	164
7.31.4.2 m_commandBuffers	164
7.31.4.3 m_currentFrameIndex	164
7.31.4.4 m_currentImageIndex	164
7.31.4.5 m_device	164
7.31.4.6 m_isFrameStarted	164
7.31.4.7 m_swapChain	165
7.31.4.8 m_window	165
7.32 ven::RenderSystem Class Reference	165
7.32.1 Detailed Description	167
7.32.2 Constructor & Destructor Documentation	167
7.32.2.1 RenderSystem() [1/2]	167
7.32.2.2 RenderSystem() [2/2]	168
7.32.3 Member Function Documentation	168
7.32.3.1 operator=()	168
7.32.3.2 renderObjects()	168
7.33 ven::Shaders Class Reference	169
7.33.1 Detailed Description	170
7.33.2 Constructor & Destructor Documentation	171
7.33.2.1 Shaders() [1/2]	171
7.33.2.2 ~Shaders()	171
7.33.2.3 Shaders() [2/2]	171
7.33.3 Member Function Documentation	172
7.33.3.1 bind()	172

7.33.3.2 createGraphicsPipeline()	172
7.33.3.3 createShaderModule()	172
7.33.3.4 defaultPipelineConfigInfo()	173
7.33.3.5 operator=()	173
7.33.3.6 readFile()	173
7.33.4 Member Data Documentation	174
7.33.4.1 m_device	174
7.33.4.2 m_fragShaderModule	174
7.33.4.3 m_graphicsPipeline	174
7.33.4.4 m_vertShaderModule	174
7.34 ven::SwapChain Class Reference	175
7.34.1 Detailed Description	177
7.34.2 Constructor & Destructor Documentation	177
7.34.2.1 SwapChain() [1/3]	177
7.34.2.2 SwapChain() [2/3]	178
7.34.2.3 ~SwapChain()	178
7.34.2.4 SwapChain() [3/3]	178
7.34.3 Member Function Documentation	179
7.34.3.1 acquireNextImage()	179
7.34.3.2 chooseSwapExtent()	179
7.34.3.3 chooseSwapPresentMode()	179
7.34.3.4 chooseSwapSurfaceFormat()	179
7.34.3.5 compareSwapFormats()	179
7.34.3.6 createDepthResources()	179
7.34.3.7 createFrameBuffers()	180
7.34.3.8 createImageViews()	180
7.34.3.9 createRenderPass()	180
7.34.3.10 createSwapChain()	180
7.34.3.11 createSyncObjects()	180
7.34.3.12 extentAspectRatio()	180
7.34.3.13 findDepthFormat()	180
7.34.3.14 getFramebuffer()	181
7.34.3.15 getImageView()	181
7.34.3.16 getRenderPass()	181
7.34.3.17 getSwapChainExtent()	181
7.34.3.18 getSwapChainImageFormat()	181
7.34.3.19 height()	181
7.34.3.20 imageCount()	182
7.34.3.21 init()	182
7.34.3.22 operator=()	182
7.34.3.23 submitCommandBuffers()	182
7.34.3.24 width()	182

7.34.4 Member Data Documentation	183
7.34.4.1 m_currentFrame	183
7.34.4.2 m_depthImageMemory	183
7.34.4.3 m_depthImages	183
7.34.4.4 m_depthImageViews	183
7.34.4.5 m_device	183
7.34.4.6 m_imageAvailableSemaphores	183
7.34.4.7 m_imagesInFlight	184
7.34.4.8 m_inFlightFences	184
7.34.4.9 m_oldSwapChain	184
7.34.4.10 m_renderFinishedSemaphores	184
7.34.4.11 m_renderPass	184
7.34.4.12 m_swapChain	184
7.34.4.13 m_swapChainDepthFormat	185
7.34.4.14 m_swapChainExtent	185
7.34.4.15 m_swapChainFrameBuffers	185
7.34.4.16 m_swapChainImageFormat	185
7.34.4.17 m_swapChainImages	185
7.34.4.18 m_swapChainImageViews	185
7.34.4.19 m_windowExtent	186
7.34.4.20 MAX_FRAMES_IN_FLIGHT	186
7.35 ven::SwapChainSupportDetails Struct Reference	186
7.35.1 Detailed Description	187
7.35.2 Member Data Documentation	187
7.35.2.1 capabilities	187
7.35.2.2 formats	187
7.35.2.3 presentModes	187
7.36 myLib::Time Class Reference	187
7.36.1 Detailed Description	188
7.36.2 Constructor & Destructor Documentation	188
7.36.2.1 Time()	188
7.36.3 Member Function Documentation	188
7.36.3.1 asMicroseconds()	188
7.36.3.2 asMilliseconds()	189
7.36.3.3 asSeconds()	189
7.36.4 Member Data Documentation	189
7.36.4.1 m_seconds	189
7.37 ven::Transform3DComponent Class Reference	190
7.37.1 Detailed Description	190
7.37.2 Member Function Documentation	190
7.37.2.1 mat4()	190
7.37.2.2 normalMatrix()	191

7.37.3 Member Data Documentation	191
7.37.3.1 rotation	191
7.37.3.2 scale	191
7.37.3.3 translation	191
7.38 ven::Model::Vertex Struct Reference	192
7.38.1 Detailed Description	192
7.38.2 Member Function Documentation	193
7.38.2.1 getAttributeDescriptions()	193
7.38.2.2 getBindingDescriptions()	193
7.38.2.3 operator==()	193
7.38.3 Member Data Documentation	194
7.38.3.1 color	194
7.38.3.2 normal	194
7.38.3.3 position	194
7.38.3.4 uv	194
7.39 ven::Window Class Reference	195
7.39.1 Detailed Description	196
7.39.2 Constructor & Destructor Documentation	196
7.39.2.1 Window()	196
7.39.2.2 ~Window()	196
7.39.3 Member Function Documentation	196
7.39.3.1 createWindow()	196
7.39.3.2 createWindowSurface()	197
7.39.3.3 framebufferResizeCallback()	197
7.39.3.4 getExtent()	198
7.39.3.5 getGLFWWindow()	198
7.39.3.6 resetWindowResizedFlag()	198
7.39.3.7 setFullscreen()	199
7.39.3.8 wasWindowResized()	199
7.39.4 Member Data Documentation	199
7.39.4.1 m_framebufferResized	199
7.39.4.2 m_height	199
7.39.4.3 m_width	200
7.39.4.4 m_window	200
8 File Documentation	201
8.1 /home/runner/work/VEngine/VEngine/include/VEngine/Abstraction/ARenderSystemBase.hpp File Reference	201
8.1.1 Detailed Description	202
8.2 ARenderSystemBase.hpp	202
8.3 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference	203
8.3.1 Detailed Description	204
8.4 Buffer.hpp	204

8.5 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference	206
8.5.1 Detailed Description	208
8.6 Camera.hpp	208
8.7 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference	209
8.8 Colors.hpp	210
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference	211
8.9.1 Detailed Description	212
8.10 DescriptorPool.hpp	212
8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference	213
8.11.1 Detailed Description	214
8.12 DescriptorSetLayout.hpp	215
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference	215
8.13.1 Detailed Description	217
8.14 DescriptorWriter.hpp	217
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference	217
8.15.1 Detailed Description	218
8.16 Device.hpp	219
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference	220
8.17.1 Detailed Description	221
8.18 Engine.hpp	221
8.19 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference	222
8.19.1 Detailed Description	223
8.20 FrameInfo.hpp	223
8.21 /home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp File Reference	224
8.21.1 Detailed Description	225
8.22 ImGuiWindowManager.hpp	225
8.23 /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp File Reference	226
8.24 KeyboardController.hpp	227
8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp File Reference	228
8.25.1 Detailed Description	229
8.26 Light.hpp	229
8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference	230
8.27.1 Detailed Description	232
8.28 Model.hpp	232
8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference	233
8.29.1 Detailed Description	234
8.30 Object.hpp	234
8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference	235
8.31.1 Detailed Description	236
8.32 Renderer.hpp	236
8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference	237

8.33.1 Detailed Description	238
8.34 Shaders.hpp	238
8.35 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference	239
8.35.1 Detailed Description	241
8.36 SwapChain.hpp	241
8.37 /home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp File Reference	242
8.37.1 Detailed Description	243
8.38 PointLightSystem.hpp	243
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp File Reference	244
8.39.1 Detailed Description	245
8.40 RenderSystem.hpp	245
8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp File Reference	246
8.41.1 Detailed Description	246
8.42 Transform3DComponent.hpp	247
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference	247
8.44 Utils.hpp	248
8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference	248
8.45.1 Detailed Description	250
8.45.2 Macro Definition Documentation	250
8.45.2.1 GLFW_INCLUDE_VULKAN	250
8.46 Window.hpp	250
8.47 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference	251
8.47.1 Detailed Description	252
8.47.2 Typedef Documentation	252
8.47.2.1 TimePoint	252
8.48 Clock.hpp	252
8.49 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp File Reference	253
8.49.1 Detailed Description	254
8.50 Time.hpp	255
8.51 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp File Reference	255
8.51.1 Detailed Description	257
8.52 Random.hpp	257
8.53 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp File Reference	258
8.54 clock.cpp	258
8.55 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference	259
8.56 random.cpp	259
8.57 /home/runner/work/VEngine/VEngine/README.md File Reference	260
8.58 /home/runner/work/VEngine/VEngine/src/Abstraction/renderSystemBase.cpp File Reference	260
8.59 renderSystemBase.cpp	260
8.60 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference	261
8.61 buffer.cpp	261

8.62 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference	262
8.63 camera.cpp	263
8.64 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference	265
8.65 descriptorPool.cpp	265
8.66 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference	266
8.67 descriptorSetLayout.cpp	266
8.68 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference	267
8.69 descriptorWriter.cpp	268
8.70 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference	268
8.70.1 Function Documentation	269
8.70.1.1 CreateDebugUtilsMessengerEXT()	269
8.70.1.2 debugCallback()	270
8.70.1.3 DestroyDebugUtilsMessengerEXT()	270
8.71 device.cpp	271
8.72 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference	277
8.73 engine.cpp	277
8.74 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference	279
8.74.1 Variable Documentation	280
8.74.1.1 DESCRIPTOR_COUNT	280
8.75 init.cpp	280
8.76 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference	281
8.77 render.cpp	282
8.78 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp File Reference	286
8.79 keyboardController.cpp	286
8.80 /home/runner/work/VEngine/VEngine/src/light.cpp File Reference	288
8.81 light.cpp	288
8.82 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference	288
8.82.1 Function Documentation	289
8.82.1.1 main()	289
8.83 main.cpp	289
8.84 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference	290
8.84.1 Macro Definition Documentation	290
8.84.1.1 GLM_ENABLE_EXPERIMENTAL	290
8.84.1.2 TINYOBJLOADER_IMPLEMENTATION	290
8.85 model.cpp	291
8.86 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference	293
8.87 renderer.cpp	293
8.88 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference	295
8.89 shaders.cpp	296
8.90 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference	298
8.91 swapChain.cpp	299
8.92 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference	304

8.93 pointLightSystem.cpp	304
8.94 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference	305
8.95 renderSystem.cpp	305
8.96 /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp File Reference	306
8.97 transform3DComponent.cpp	306
8.98 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference	307
8.99 window.cpp	307
Index	309

Chapter 1

vengine

1.1 VEngine - Vulkan Graphics Engine

WORK IN PROGRESS!

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

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

1.1.1 Features

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

1.1.1.1 Planned Features:

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

1.1.2 Prerequisites

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

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

1.1.3 Usage

1.1.3.1 Build

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

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

1.1.3.2 Run

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

1.1.4 Key Bindings

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

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

1.1.5 Documentation

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

1.1.6 Commit Norms

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

1.1.7 License

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

1.1.8 Acknowledgements

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

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

myLib	13
ven	14

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ven::ARenderSystemBase	21
ven::PointLightSystem	147
ven::RenderSystem	165
ven::Buffer	27
ven::DescriptorPool::Builder	39
ven::DescriptorSetLayout::Builder	44
ven::Model::Builder	47
ven::Camera	49
myLib::Clock	55
ven::Colors	58
ven::DescriptorPool	68
ven::DescriptorSetLayout	72
ven::DescriptorWriter	76
ven::Device	80
ven::Engine	97
ven::FrameInfo	104
ven::ImGuiWindowManager::funcs	107
ven::GlobalUbo	109
std::hash< ven::Model::Vertex >	111
ven::ImGuiWindowManager	112
ven::KeyboardController	118
ven::KeyboardController::KeyMappings	121
ven::Light	124
ven::LightPushConstantData	128
ven::Model	129
ven::Object	135
ven::ObjectPushConstantData	141
ven::PipelineConfigInfo	142
ven::PointLightData	146
ven::QueueFamilyIndices	151
myLib::Random	153
ven::Renderer	156
ven::Shaders	169
ven::SwapChain	175
ven::SwapChainSupportDetails	186

myLib::Time	187
ven::Transform3DComponent	190
ven::Model::Vertex	192
ven::Window	195

Chapter 4

Class Index

4.1 Class List

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

ven::ARenderSystemBase	
Abstract class for render system base	21
ven::Buffer	
Class for buffer	27
ven::DescriptorPool::Builder	39
ven::DescriptorSetLayout::Builder	44
ven::Model::Builder	47
ven::Camera	
Class for camera	49
myLib::Clock	
Class for time management	55
ven::Colors	
Class for colors	58
ven::DescriptorPool	
Class for descriptor pool	68
ven::DescriptorSetLayout	
Class for descriptor set layout	72
ven::DescriptorWriter	
Class for descriptor writer	76
ven::Device	80
ven::Engine	97
ven::FrameInfo	104
ven::ImGuiWindowManager::funcs	107
ven::GlobalUbo	109
std::hash< ven::Model::Vertex >	111
ven::ImGuiWindowManager	
Class for ImGui window manager	112
ven::KeyboardController	
Class for keyboard controller	118
ven::KeyboardController::KeyMappings	121
ven::Light	
Class for light	124
ven::LightPushConstantData	128
ven::Model	
Class for model	129

ven::Object	
Class for object	135
ven::ObjectPushConstantData	141
ven::PipelineConfigInfo	142
ven::PointLightData	146
ven::PointLightSystem	
Class for point light system	147
ven::QueueFamilyIndices	151
myLib::Random	
Class for random number generation	153
ven::Renderer	156
ven::RenderSystem	
Class for render system	165
ven::Shaders	
Class for shaders	169
ven::SwapChain	
Class for swap chain	175
ven::SwapChainSupportDetails	186
myLib::Time	
Class used for time management	187
ven::Transform3DComponent	190
ven::Model::Vertex	192
ven::Window	
Class for window	195

Chapter 5

File Index

5.1 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	203
/home/runner/work/VEngine/VEngine/include/VEngine/ Camera.hpp	
This file contains the Camera class	206
/home/runner/work/VEngine/VEngine/include/VEngine/ Colors.hpp	209
/home/runner/work/VEngine/VEngine/include/VEngine/ Device.hpp	
This file contains the Device class	217
/home/runner/work/VEngine/VEngine/include/VEngine/ Engine.hpp	
This file contains the Engine class	220
/home/runner/work/VEngine/VEngine/include/VEngine/ FrameInfo.hpp	
This file contains the FrameInfo class	222
/home/runner/work/VEngine/VEngine/include/VEngine/ ImGuiWindowManager.hpp	
This file contains the ImGuiWindowManager class	224
/home/runner/work/VEngine/VEngine/include/VEngine/ KeyboardController.hpp	226
/home/runner/work/VEngine/VEngine/include/VEngine/ Light.hpp	
This file contains the Light class	228
/home/runner/work/VEngine/VEngine/include/VEngine/ Model.hpp	
This file contains the Model class	230
/home/runner/work/VEngine/VEngine/include/VEngine/ Object.hpp	
This file contains the Object class	233
/home/runner/work/VEngine/VEngine/include/VEngine/ Renderer.hpp	
This file contains the Renderer class	235
/home/runner/work/VEngine/VEngine/include/VEngine/ Shaders.hpp	
This file contains the Shader class	237
/home/runner/work/VEngine/VEngine/include/VEngine/ SwapChain.hpp	
This file contains the Shader class	239
/home/runner/work/VEngine/VEngine/include/VEngine/ Transform3DComponent.hpp	
This file contains the Transform3DComponent class	246
/home/runner/work/VEngine/VEngine/include/VEngine/ Utils.hpp	247
/home/runner/work/VEngine/VEngine/include/VEngine/ Window.hpp	
This file contains the Window class	248
/home/runner/work/VEngine/VEngine/include/VEngine/Abstraction/ ARenderSystemBase.hpp	
This file contains the ARenderSystemBase class	201
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/ DescriptorPool.hpp	
This file contains the DescriptorPool class	211

/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp	
This file contains the DescriptorSetLayout class	213
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp	
This file contains the DescriptorsWriter class	215
/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp	
This file contains the PointLightSystem class	242
/home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp	
This file contains the RenderSystem class	244
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp	
Class for random number generation	255
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp	
Clock class for time management	251
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp	
Class for time management	253
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp	258
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp	259
/home/runner/work/VEngine/VEngine/src/buffer.cpp	261
/home/runner/work/VEngine/VEngine/src/camera.cpp	262
/home/runner/work/VEngine/VEngine/src/device.cpp	268
/home/runner/work/VEngine/VEngine/src/engine.cpp	277
/home/runner/work/VEngine/VEngine/src/keyboardController.cpp	286
/home/runner/work/VEngine/VEngine/src/light.cpp	288
/home/runner/work/VEngine/VEngine/src/main.cpp	288
/home/runner/work/VEngine/VEngine/src/model.cpp	290
/home/runner/work/VEngine/VEngine/src/renderer.cpp	293
/home/runner/work/VEngine/VEngine/src/shaders.cpp	295
/home/runner/work/VEngine/VEngine/src/swapChain.cpp	298
/home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp	306
/home/runner/work/VEngine/VEngine/src/window.cpp	307
/home/runner/work/VEngine/VEngine/src/Abstraction/renderSystemBase.cpp	260
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp	265
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp	266
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp	267
/home/runner/work/VEngine/VEngine/src/gui/init.cpp	279
/home/runner/work/VEngine/VEngine/src/gui/render.cpp	281
/home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp	304
/home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp	305

Chapter 6

Namespace Documentation

6.1 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

6.1.1 Variable Documentation

6.1.1.1 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\(\)](#).

6.1.1.2 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\(\)](#).

6.1.1.3 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\(\)](#).

6.1.1.4 RANDOM_INT_MAX

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

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

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

6.1.1.5 RANDOM_INT_MIN

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

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

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

6.2 ven Namespace Reference

Classes

- class [ARenderSystemBase](#)
Abstract class for render system base.
- class [Buffer](#)
Class for buffer.
- class [Camera](#)
Class for camera.
- class [Colors](#)
Class for colors.
- class [DescriptorPool](#)
Class for descriptor pool.
- class [DescriptorSetLayout](#)
Class for descriptor set layout.
- class [DescriptorWriter](#)
Class for descriptor writer.
- class [Device](#)
- class [Engine](#)
- struct [FrameInfo](#)
- struct [GlobalUbo](#)
- class [ImGuiWindowManager](#)
Class for ImGui window manager.
- class [KeyboardController](#)

- Class for keyboard controller.*
- class [Light](#)
 - Class for light.*
- struct [LightPushConstantData](#)
- class [Model](#)
 - Class for model.*
- class [Object](#)
 - Class for object.*
- struct [ObjectPushConstantData](#)
- struct [PipelineConfigInfo](#)
- struct [PointLightData](#)
- class [PointLightSystem](#)
 - Class for point light system.*
- struct [QueueFamilyIndices](#)
- class [Renderer](#)
- class [RenderSystem](#)
 - Class for render system.*
- class [Shaders](#)
 - Class for shaders.*
- class [SwapChain](#)
 - Class for swap chain.*
- struct [SwapChainSupportDetails](#)
- class [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_AMBIENT_LIGHT_INTENSITY](#) = .2F
- static constexpr glm::vec4 [DEFAULT_AMBIENT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_AMBIENT_LIGHT_INTENSITY](#)}
- 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::vec4 [DEFAULT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_LIGHT_INTENSITY](#)}
- static constexpr VkClearColorValue [DEFAULT_CLEAR_COLOR](#) = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue [DEFAULT_CLEAR_DEPTH](#) = {1.0F, 0}
- static constexpr std::string_view [SHADERS_BIN_PATH](#) = "shaders/bin/"
- static constexpr uint32_t [DEFAULT_WIDTH](#) = 1920
- static constexpr uint32_t [DEFAULT_HEIGHT](#) = 1080
- static constexpr std::string_view [DEFAULT_TITLE](#) = "VEngine"

6.2.1 Function Documentation

6.2.1.1 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](#).

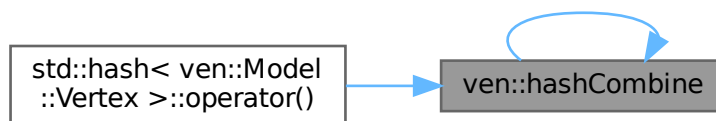
References [hashCombine\(\)](#).

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:



6.2.2 Variable Documentation

6.2.2.1 DEFAULT_AMBIENT_LIGHT_COLOR

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

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

6.2.2.2 DEFAULT_AMBIENT_LIGHT_INTENSITY

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

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

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

6.2.2.3 DEFAULT_CLEAR_COLOR

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

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

6.2.2.4 DEFAULT_CLEAR_DEPTH

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

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

6.2.2.5 DEFAULT_FAR

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

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

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

6.2.2.6 DEFAULT_FOV

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

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

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

6.2.2.7 DEFAULT_HEIGHT

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

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

6.2.2.8 DEFAULT_LIGHT_COLOR

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

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

6.2.2.9 DEFAULT_LIGHT_INTENSITY

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

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

Referenced by [ven::ImGuiWindowManager::lightsSection\(\)](#), and [ven::Engine::loadObjects\(\)](#).

6.2.2.10 DEFAULT_LIGHT_RADIUS

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

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

6.2.2.11 DEFAULT_LOOK_SPEED

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

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

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

6.2.2.12 DEFAULT_MOVE_SPEED

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

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

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

6.2.2.13 DEFAULT_NEAR

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

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

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

6.2.2.14 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::ImGuiWindowManager::cameraSection\(\)](#), and [ven::Engine::mainLoop\(\)](#).

6.2.2.15 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::cameraSection\(\)](#).

6.2.2.16 DEFAULT_TITLE

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

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

6.2.2.17 DEFAULT_WIDTH

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

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

6.2.2.18 MAX_LIGHTS

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

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

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

6.2.2.19 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::PointLightSystem\(\)](#), and [ven::RenderSystem::RenderSystem\(\)](#).

Chapter 7

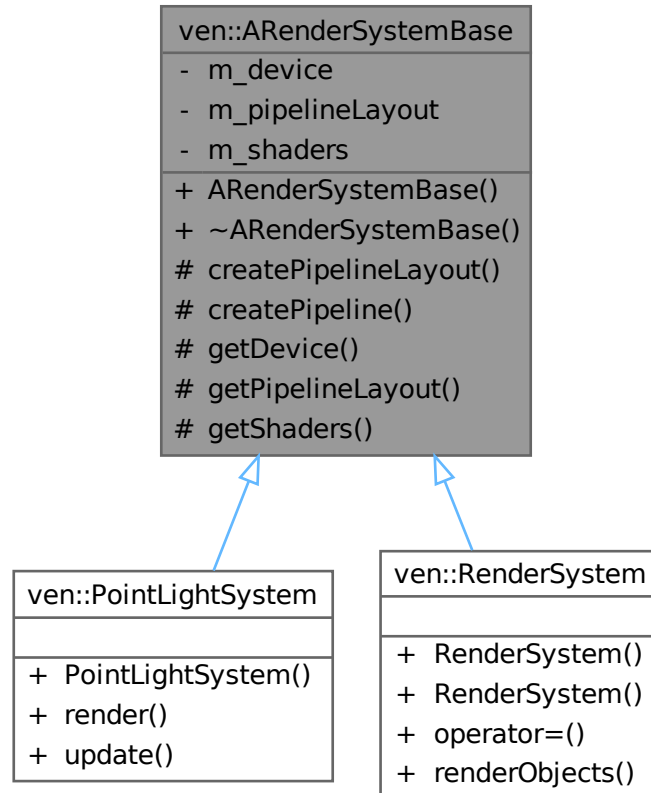
Class Documentation

7.1 ven::ARenderSystemBase Class Reference

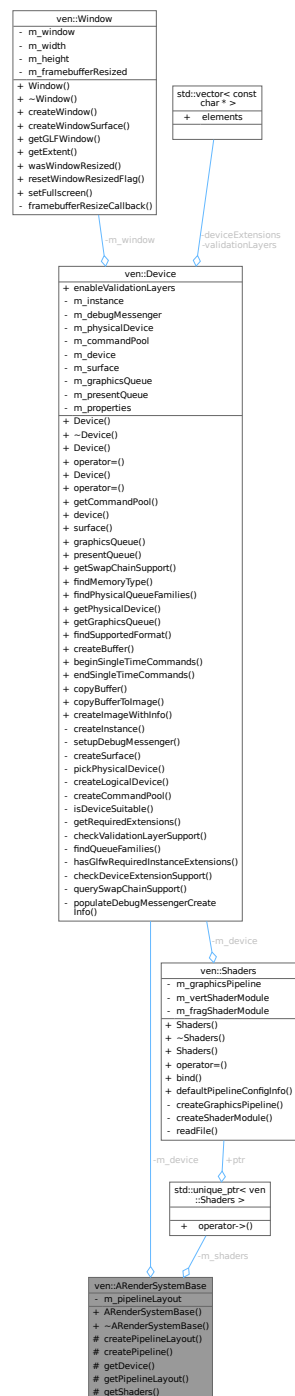
Abstract class for render system base.

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

Inheritance diagram for ven::ARenderSystemBase:



Collaboration diagram for `ven::ARenderSystemBase`:



Public Member Functions

- [ARenderSystemBase \(Device &device\)](#)
- [~ARenderSystemBase \(\)](#)

Protected Member Functions

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)

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

Private Attributes

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

7.1.1 Detailed Description

Abstract class for render system base.

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

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ARenderSystemBase()

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

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

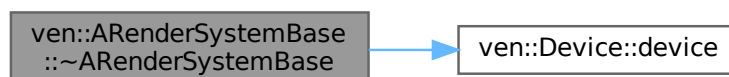
7.1.2.2 ~ARenderSystemBase()

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

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

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

Here is the call graph for this function:



7.1.3 Member Function Documentation

7.1.3.1 createPipeline()

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

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

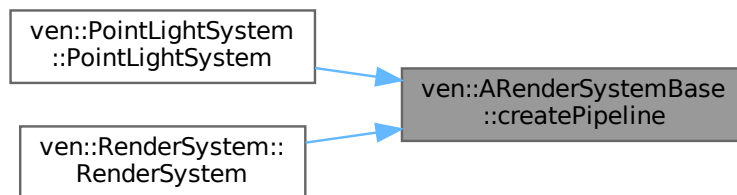
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.2 createPipelineLayout()

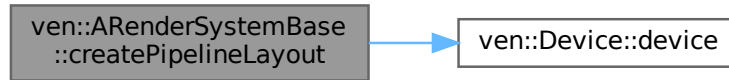
```
void ven::ARenderSystemBase::createPipelineLayout (
    VkDescriptorSetLayout globalSetLayout,
    uint32_t pushConstantSize) [protected]
```

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

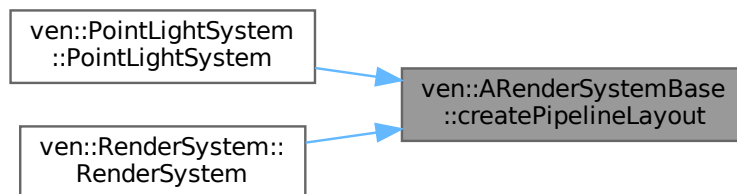
References [ven::Device::device\(\)](#), [m_device](#), and [m_pipelineLayout](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.3 getDevice()

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

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

References [m_device](#).

7.1.3.4 getPipelineLayout()

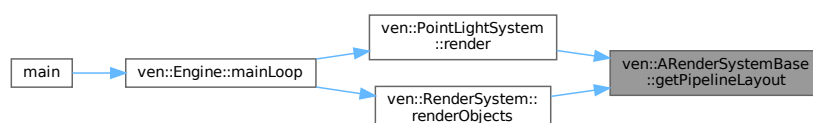
```
VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]
```

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

References [m_pipelineLayout](#).

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

Here is the caller graph for this function:



7.1.3.5 getShaders()

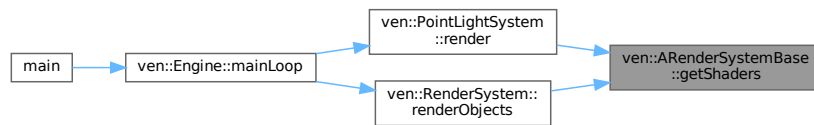
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],
[nodiscard], [protected]
```

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

References [m_shaders](#).

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

Here is the caller graph for this function:



7.1.4 Member Data Documentation

7.1.4.1 m_device

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

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

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

7.1.4.2 m_pipelineLayout

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

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

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

7.1.4.3 m_shaders

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

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

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

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

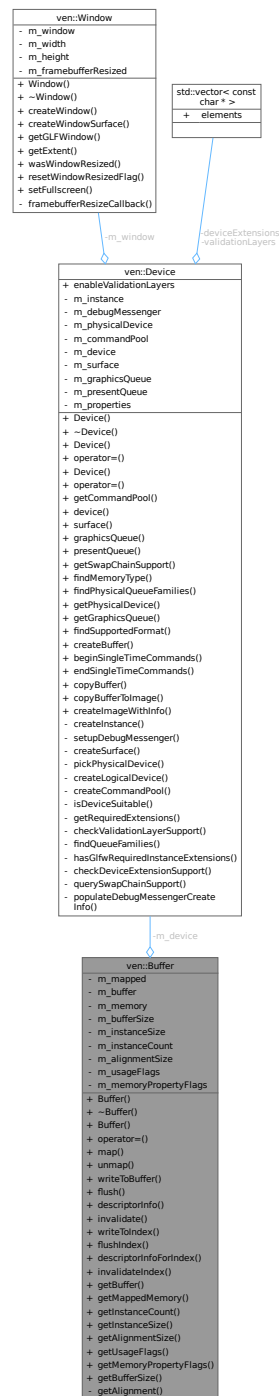
- [/home/runner/work/VEngine/VEngine/include/VEngine/Abstraction/ARenderSystemBase.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/Abstraction/renderSystemBase.cpp](#)

7.2 ven::Buffer Class Reference

Class for buffer.

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

Collaboration diagram for ven::Buffer:



Public Member Functions

- [Buffer](#) ([Device](#) &device, VkDeviceSize instanceSize, uint32_t instanceCount, VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize minOffsetAlignment=1)
- [~Buffer](#) ()
- [Buffer](#) (const [Buffer](#) &)=delete
- [Buffer](#) & [operator=](#) (const [Buffer](#) &)=delete
- VkResult [map](#) (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0)
Map a memory range of this buffer.
- void [unmap](#) ()
Unmap a mapped memory range.
- void [writeToBuffer](#) (const void *data, VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const
Copies the specified data to the mapped buffer.
- VkResult [flush](#) (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const
Flush a memory range of the buffer to make it visible to the device.
- VkDescriptorBufferInfo [descriptorInfo](#) (const VkDeviceSize size=VK_WHOLE_SIZE, const VkDeviceSize offset=0) const
Create a buffer info descriptor.
- VkResult [invalidate](#) (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const
Invalidate a memory range of the buffer to make it visible to the host.
- void [writeToIndex](#) (const void *data, const VkDeviceSize index) const
*Copies "instanceSize" bytes of data to the mapped buffer at an offset of index * alignmentSize.*
- VkResult [flushIndex](#) (const VkDeviceSize index) const
*Flush the memory range at index * alignmentSize of the buffer to make it visible to the device.*
- VkDescriptorBufferInfo [descriptorInfoForIndex](#) (const VkDeviceSize index) const
Create a buffer info descriptor.
- VkResult [invalidateIndex](#) (const VkDeviceSize index) const
Invalidate a memory range of the buffer to make it visible to the host.
- VkBuffer [getBuffer](#) () const
- void * [getMappedMemory](#) () const
- uint32_t [getInstanceCount](#) () const
- VkDeviceSize [getInstanceSize](#) () const
- VkDeviceSize [getAlignmentSize](#) () const
- VkBufferUsageFlags [getUsageFlags](#) () const
- VkMemoryPropertyFlags [getMemoryPropertyFlags](#) () const
- VkDeviceSize [getBufferSize](#) () const

Static Private Member Functions

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

Private Attributes

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

7.2.1 Detailed Description

Class for buffer.

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

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Buffer() [1/2]

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

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

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

Here is the call graph for this function:



7.2.2.2 ~Buffer()

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

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

7.2.2.3 Buffer() [2/2]

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

7.2.3 Member Function Documentation

7.2.3.1 descriptorInfo()

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

Create a buffer info descriptor.

Parameters

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

Returns

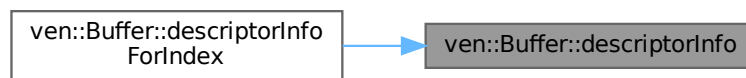
VkDescriptorBufferInfo of specified offset and range

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

References [m_buffer](#).

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

Here is the caller graph for this function:



7.2.3.2 descriptorInfoForIndex()

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

Create a buffer info descriptor.

Parameters

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

Returns

VkDescriptorBufferInfo for instance at index

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

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

Here is the call graph for this function:



7.2.3.3 flush()

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

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

Note

Only required for non-coherent memory

Parameters

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

Returns

VkResult of the flush call

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

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

Here is the caller graph for this function:



7.2.3.4 flushIndex()

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

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

Parameters

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

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

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

Here is the call graph for this function:



7.2.3.5 getAlignment()

```

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

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

Parameters

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

Returns

VkResult of the buffer mapping call

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

7.2.3.6 getAlignmentSize()

```

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

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

References [m_alignmentSize](#).

7.2.3.7 getBuffer()

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

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

References [m_buffer](#).

7.2.3.8 getBufferSize()

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

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

References [m_bufferSize](#).

7.2.3.9 getInstanceCount()

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

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

References [m_instanceCount](#).

7.2.3.10 getInstanceSize()

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

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

References [m_instanceSize](#).

7.2.3.11 getMappedMemory()

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

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

References [m_mapped](#).

7.2.3.12 getMemoryPropertyFlags()

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

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

References [m_memoryPropertyFlags](#).

7.2.3.13 `getUsageFlags()`

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

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

References [m_usageFlags](#).

7.2.3.14 `invalidate()`

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

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

Note

Only required for non-coherent memory

Parameters

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

Returns

VkResult of the invalidate call

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

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

Here is the caller graph for this function:



7.2.3.15 `invalidateIndex()`

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

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

Note

Only required for non-coherent memory

Parameters

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

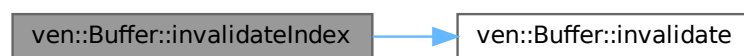
Returns

VkResult of the invalidate call

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

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

Here is the call graph for this function:



7.2.3.16 map()

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

Map a memory range of this buffer.

If successful, mapped points to the specified buffer range.

Parameters

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

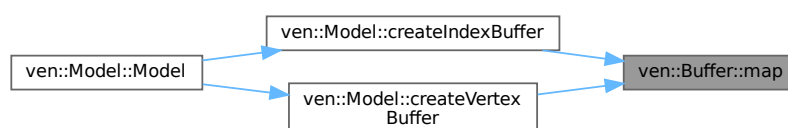
Returns

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:



7.2.3.17 operator=()

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

7.2.3.18 unmap()

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

Unmap a mapped memory range.

Note

Does not return a result as vkUnmapMemory can't fail

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

7.2.3.19 writeToBuffer()

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

Copies the specified data to the mapped buffer.

Default value writes whole buffer range

Parameters

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

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

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

Here is the caller graph for this function:



7.2.3.20 writeToIndex()

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

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

Parameters

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

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

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

Here is the call graph for this function:



7.2.4 Member Data Documentation

7.2.4.1 m_alignmentSize

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

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

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

7.2.4.2 m_buffer

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

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

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

7.2.4.3 m_bufferSize

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

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

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

7.2.4.4 m_device

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

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

7.2.4.5 m_instanceCount

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

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

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

7.2.4.6 m_instanceSize

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

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

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

7.2.4.7 m_mapped

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

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

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

7.2.4.8 m_memory

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

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

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

7.2.4.9 m_memoryPropertyFlags

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

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

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

7.2.4.10 m_usageFlags

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

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

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

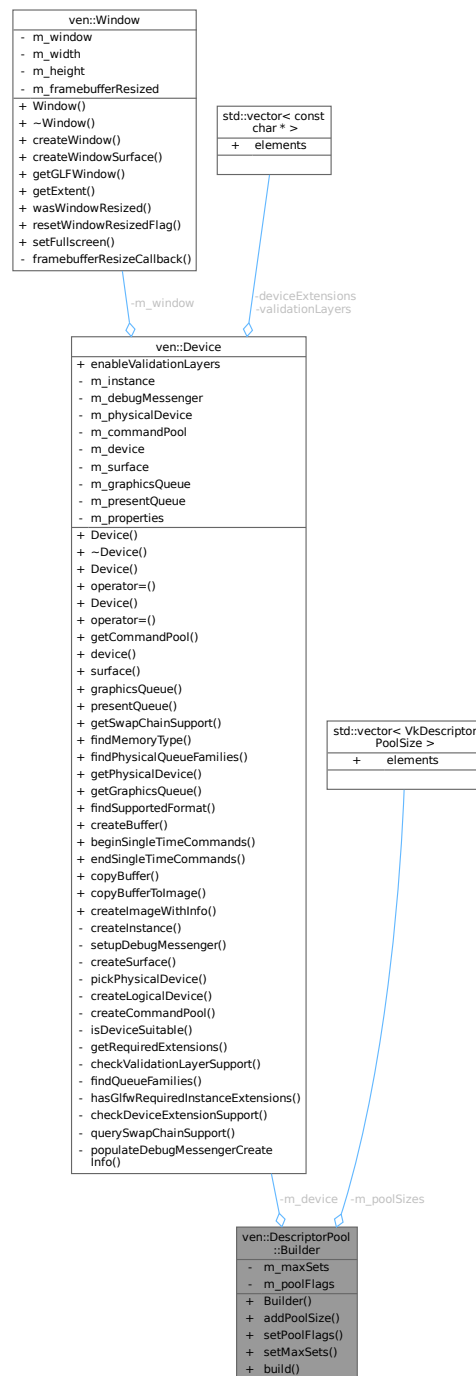
The documentation for this class was generated from the following files:

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

7.3 ven::DescriptorPool::Builder Class Reference

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

Collaboration diagram for `ven::DescriptorPool::Builder`:



Public Member Functions

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

7.3.1 Detailed Description

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Builder()

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

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

7.3.3 Member Function Documentation

7.3.3.1 addPoolSize()

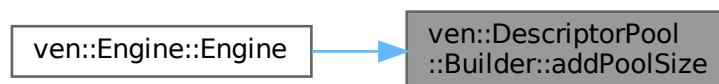
```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::addPoolSize (
    VkDescriptorType descriptorType,
    uint32_t count)
```

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

References [m_poolSizes](#).

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

Here is the caller graph for this function:



7.3.3.2 build()

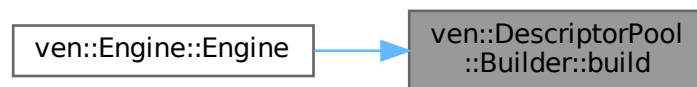
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],
[nodiscard]
```

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

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

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

Here is the caller graph for this function:



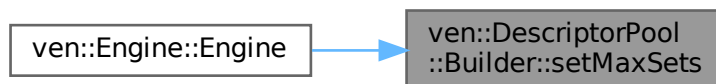
7.3.3.3 setMaxSets()

```
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:



7.3.3.4 setPoolFlags()

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

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

7.3.4 Member Data Documentation

7.3.4.1 m_device

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

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

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

7.3.4.2 m_maxSets

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

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

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

7.3.4.3 m_poolFlags

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

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

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

7.3.4.4 m_poolSizes

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

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

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

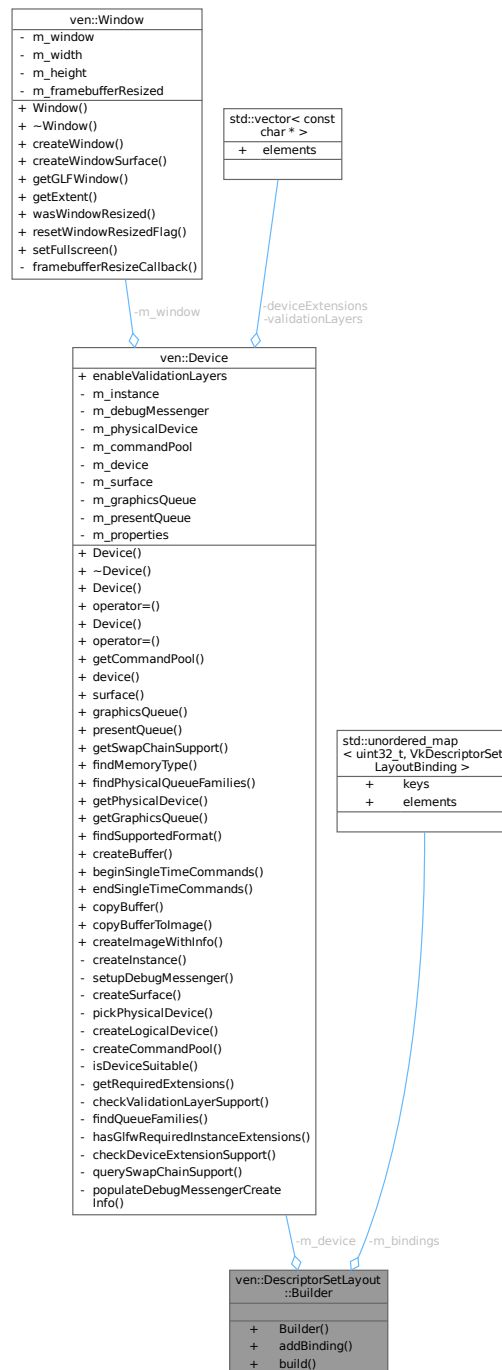
The documentation for this class was generated from the following files:

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

7.4 ven::DescriptorSetLayout::Builder Class Reference

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

Collaboration diagram for ven::DescriptorSetLayout::Builder:



Public Member Functions

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

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

Private Attributes

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

7.4.1 Detailed Description

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Builder()

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

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

7.4.3 Member Function Documentation

7.4.3.1 addBinding()

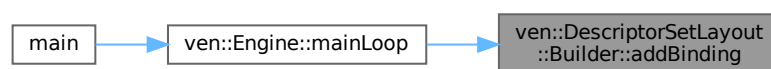
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

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

References [m_bindings](#).

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

Here is the caller graph for this function:



7.4.3.2 build()

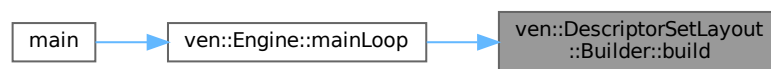
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

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

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

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

Here is the caller graph for this function:



7.4.4 Member Data Documentation

7.4.4.1 m_bindings

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

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

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

7.4.4.2 m_device

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

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

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

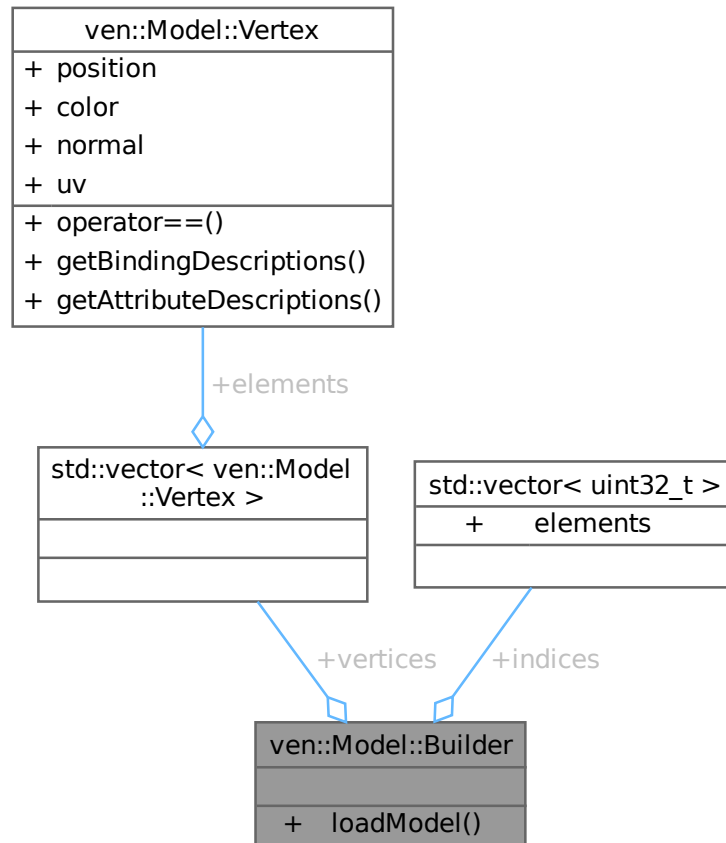
The documentation for this class was generated from the following files:

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

7.5 ven::Model::Builder Struct Reference

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

Collaboration diagram for ven::Model::Builder:



Public Member Functions

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

Public Attributes

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

7.5.1 Detailed Description

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

7.5.2 Member Function Documentation

7.5.2.1 loadModel()

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

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

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

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

Here is the caller graph for this function:



7.5.3 Member Data Documentation

7.5.3.1 indices

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

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

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

7.5.3.2 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](#)

7.6 ven::Camera Class Reference

Class for camera.

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

Collaboration diagram for ven::Camera:

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

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](#) (const glm::vec3 position, const glm::vec3 target, const glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void [setViewYXZ](#) (glm::vec3 position, glm::vec3 rotation)
- void [setFov](#) (const float fov)
- void [setNear](#) (const float near)
- void [setFar](#) (const 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}`

7.6.1 Detailed Description

Class for camera.

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

7.6.2 Member Function Documentation

7.6.2.1 getFar()

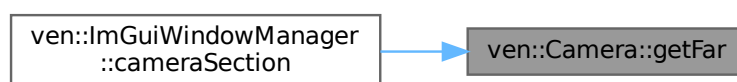
```
float ven::Camera::getFar () const [inline], [nodiscard]
```

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

References [m_far](#).

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

Here is the caller graph for this function:



7.6.2.2 getFov()

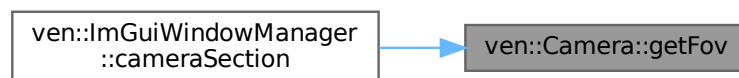
```
float ven::Camera::getFov () const [inline], [nodiscard]
```

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

References [m_fov](#).

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

Here is the caller graph for this function:



7.6.2.3 getInverseView()

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

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

References [m_inverseViewMatrix](#).

7.6.2.4 getNear()

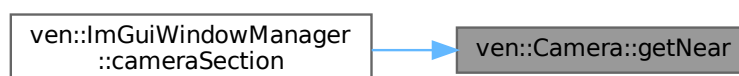
```
float ven::Camera::getNear () const [inline], [nodiscard]
```

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

References [m_near](#).

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

Here is the caller graph for this function:



7.6.2.5 getProjection()

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

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

References [m_projectionMatrix](#).

7.6.2.6 getView()

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

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

References [m_viewMatrix](#).

7.6.2.7 setFar()

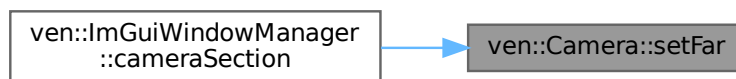
```
void ven::Camera::setFar (  
    const float far) [inline]
```

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

References [m_far](#).

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

Here is the caller graph for this function:



7.6.2.8 setFov()

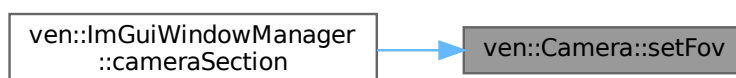
```
void ven::Camera::setFov (  
    const float fov) [inline]
```

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

References [m_fov](#).

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

Here is the caller graph for this function:



7.6.2.9 setNear()

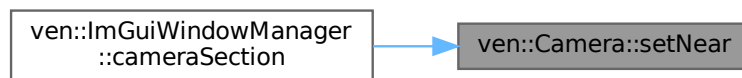
```
void ven::Camera::setNear (
    const float near) [inline]
```

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

References [m_near](#).

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

Here is the caller graph for this function:



7.6.2.10 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](#).

7.6.2.11 setPerspectiveProjection()

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

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

7.6.2.12 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](#).

7.6.2.13 `setViewTarget()`

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

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

7.6.2.14 `setViewYXZ()`

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

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

7.6.3 Member Data Documentation

7.6.3.1 `m_far`

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

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

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

7.6.3.2 `m_fov`

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

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

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

7.6.3.3 `m_inverseViewMatrix`

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

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

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

7.6.3.4 `m_near`

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

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

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

7.6.3.5 m_projectionMatrix

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

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

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

7.6.3.6 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](#)

7.7 myLib::Clock Class Reference

Class for time management.

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

Collaboration diagram for myLib::Clock:

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

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}

7.7.1 Detailed Description

Class for time management.

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 [Clock](#)()

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

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

7.7.2.2 [~Clock](#)()

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

7.7.3 Member Function Documentation

7.7.3.1 [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](#).

7.7.3.2 pause()

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

Pause the clock.

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

References [m_pause](#), and [m_paused](#).

7.7.3.3 restart()

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

Restart the clock.

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

References [m_start](#).

7.7.3.4 resume()

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

Resume the clock.

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

7.7.4 Member Data Documentation

7.7.4.1 m_pause

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

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

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

7.7.4.2 m_paused

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

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

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

7.7.4.3 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:

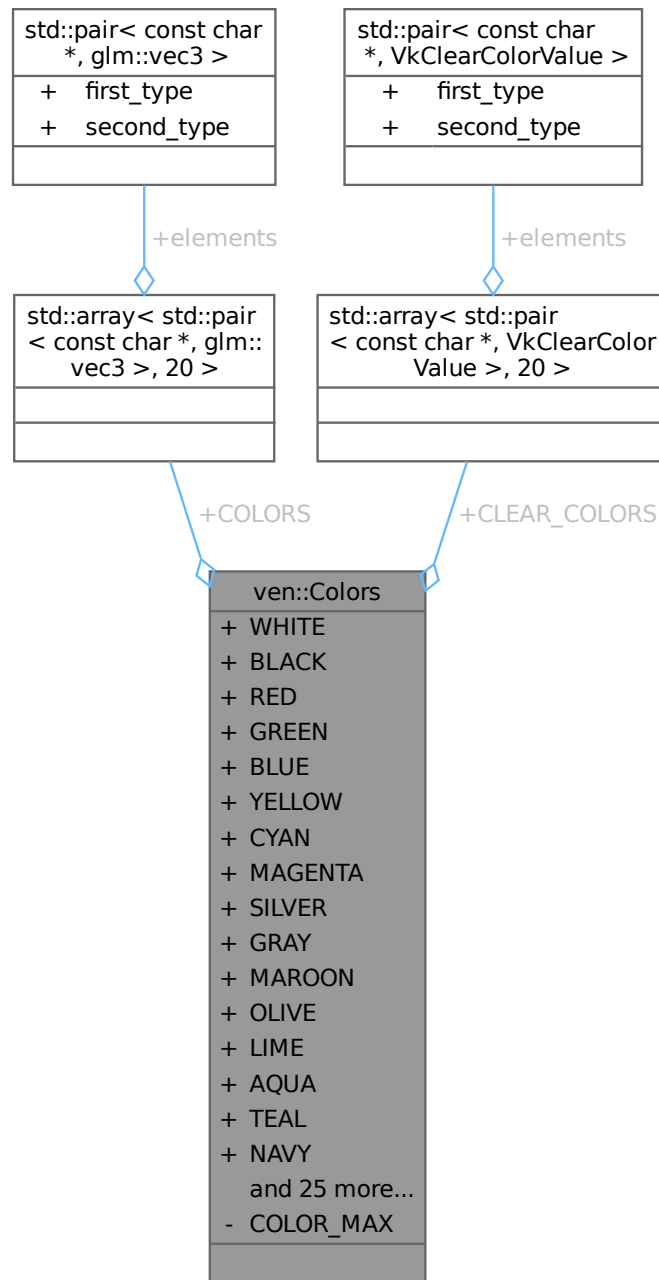
- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp](#)
- [/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp](#)

7.8 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

7.8.1 Detailed Description

Class for colors.

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

7.8.2 Member Data Documentation

7.8.2.1 AQUA

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

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

7.8.2.2 AQUA_V

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

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

7.8.2.3 BLACK

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

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

7.8.2.4 BLACK_V

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

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

7.8.2.5 BLUE

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

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

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

7.8.2.6 BLUE_V

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

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

7.8.2.7 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 92 of file [Colors.hpp](#).

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

7.8.2.8 COLOR_MAX

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

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

7.8.2.9 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 69 of file [Colors.hpp](#).

Referenced by [ven::ImGuiWindowManager::lightsSection\(\)](#), and [ven::ImGuiWindowManager::rendererSection\(\)](#).

7.8.2.10 CYAN

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

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

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

7.8.2.11 CYAN_V

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

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

7.8.2.12 FUCHSIA

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

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

7.8.2.13 FUCHSIA_V

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

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

7.8.2.14 GRAY

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

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

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

7.8.2.15 GRAY_V

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

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

7.8.2.16 GREEN

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

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

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

7.8.2.17 GREEN_V

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

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

7.8.2.18 LIME

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

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

7.8.2.19 LIME_V

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

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

7.8.2.20 MAGENTA

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

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

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

7.8.2.21 MAGENTA_V

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

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

7.8.2.22 MAROON

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

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

7.8.2.23 MAROON_V

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

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

7.8.2.24 NAVY

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

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

7.8.2.25 NAVY_V

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

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

7.8.2.26 NIGHT_BLUE

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

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

7.8.2.27 NIGHT_BLUE_V

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

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

7.8.2.28 NIGHT_MODE_V

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

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

7.8.2.29 OLIVE

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

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

7.8.2.30 OLIVE_V

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

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

7.8.2.31 RED

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

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

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

7.8.2.32 RED_V

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

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

7.8.2.33 SILVER

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

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

7.8.2.34 SILVER_V

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

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

7.8.2.35 SKY_BLUE

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

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

7.8.2.36 SKY_BLUE_V

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

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

7.8.2.37 SUNSET

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

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

7.8.2.38 SUNSET_V

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

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

7.8.2.39 TEAL

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

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

7.8.2.40 TEAL_V

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

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

7.8.2.41 WHITE

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

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

7.8.2.42 WHITE_V

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

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

7.8.2.43 YELLOW

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

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

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

7.8.2.44 YELLOW_V

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

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

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

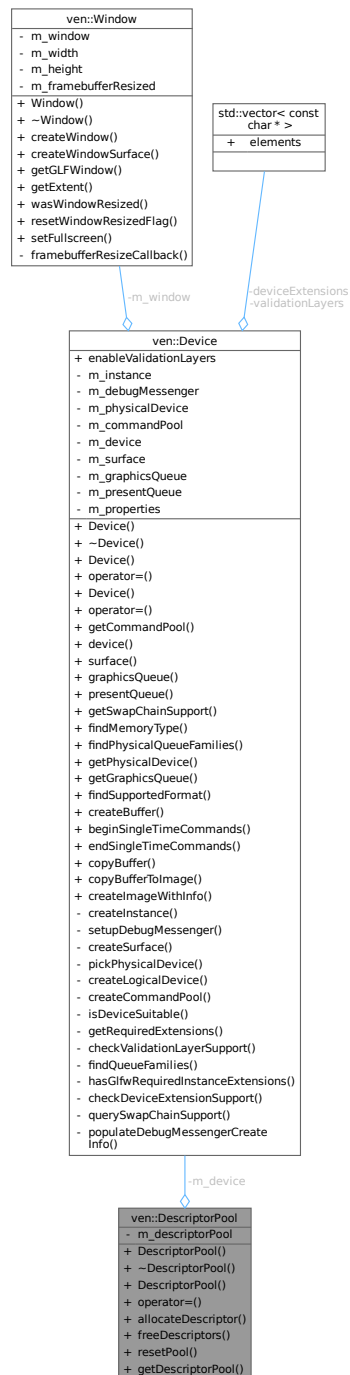
- [/home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp](#)

7.9 ven::DescriptorPool Class Reference

Class for descriptor pool.

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

Collaboration diagram for ven::DescriptorPool:



Classes

- class [Builder](#)

Public Member Functions

- [DescriptorPool](#) ([Device](#) &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags, const std::vector< VkDescriptorPoolSize > &poolSizes)
- [~DescriptorPool](#) ()
- [DescriptorPool](#) (const [DescriptorPool](#) &)=delete
- [DescriptorPool](#) & [operator=](#) (const [DescriptorPool](#) &)=delete
- bool [allocateDescriptor](#) (VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet &descriptor) const
- void [freeDescriptors](#) (const std::vector< VkDescriptorSet > &descriptors) const
- void [resetPool](#) () const
- VkDescriptorPool [getDescriptorPool](#) () const

Private Attributes

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

Friends

- class [DescriptorWriter](#)

7.9.1 Detailed Description

Class for descriptor pool.

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

7.9.2 Constructor & Destructor Documentation

7.9.2.1 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:



7.9.2.2 ~DescriptorPool()

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

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

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

Here is the call graph for this function:



7.9.2.3 DescriptorPool() [2/2]

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

7.9.3 Member Function Documentation

7.9.3.1 allocateDescriptor()

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

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

7.9.3.2 freeDescriptors()

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

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

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

Here is the call graph for this function:



7.9.3.3 getDescriptorPool()

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

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

References [m_descriptorPool](#).

7.9.3.4 operator=()

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

7.9.3.5 resetPool()

```
void ven::DescriptorPool::resetPool () 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:



7.9.4 Friends And Related Symbol Documentation

7.9.4.1 DescriptorWriter

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

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

7.9.5 Member Data Documentation

7.9.5.1 m_descriptorPool

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

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

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

7.9.5.2 m_device

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

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

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

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

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

7.10 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

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


Collaboration diagram for ven::DescriptorSetLayout:



Classes

- class [Builder](#)

Public Member Functions

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

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

Private Attributes

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

Friends

- class [DescriptorWriter](#)

7.10.1 Detailed Description

Class for descriptor set layout.

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

7.10.2 Constructor & Destructor Documentation

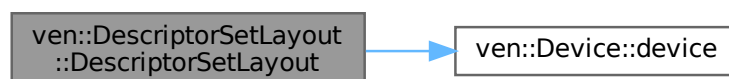
7.10.2.1 DescriptorSetLayout() [1/2]

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

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

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

Here is the call graph for this function:



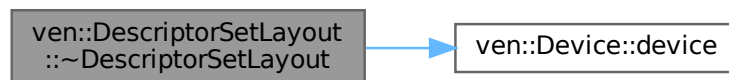
7.10.2.2 ~DescriptorSetLayout()

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

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

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

Here is the call graph for this function:



7.10.2.3 DescriptorSetLayout() [2/2]

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

7.10.3 Member Function Documentation

7.10.3.1 getDescriptorSetLayout()

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

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

References [m_descriptorSetLayout](#).

7.10.3.2 operator=()

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

7.10.4 Friends And Related Symbol Documentation

7.10.4.1 DescriptorWriter

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

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

7.10.5 Member Data Documentation

7.10.5.1 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\(\)](#).

7.10.5.2 m_descriptorSetLayout

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

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

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

7.10.5.3 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:

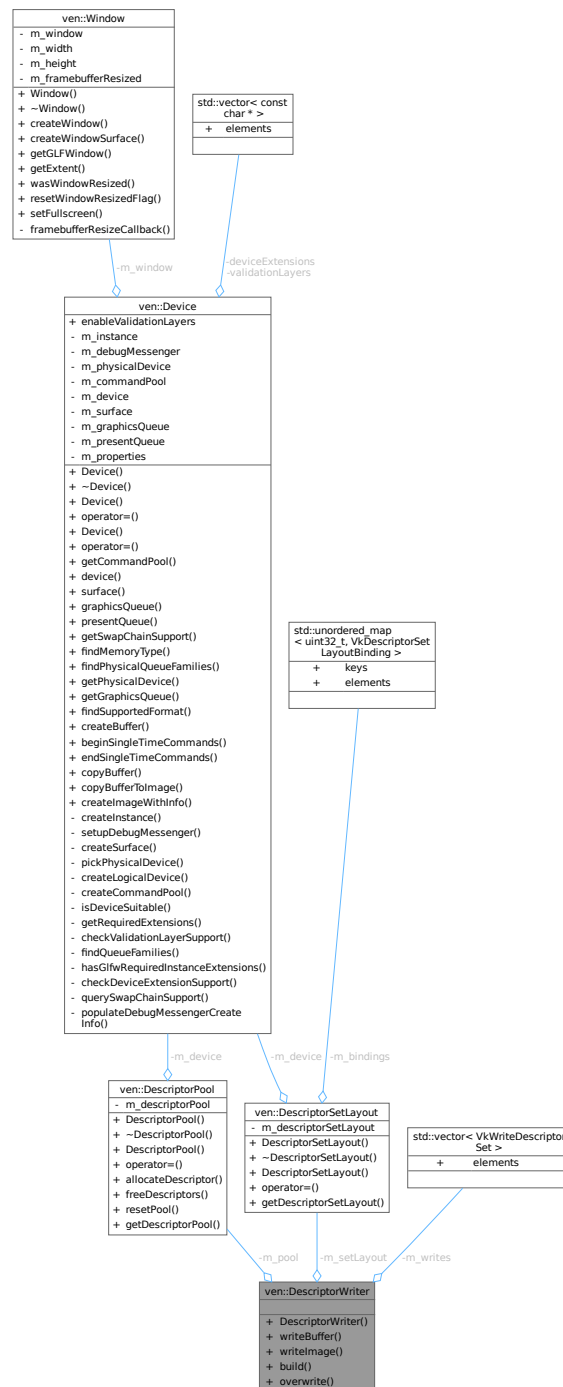
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#)

7.11 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](#)

7.11.1 Detailed Description

Class for descriptor writer.

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

7.11.2 Constructor & Destructor Documentation

7.11.2.1 DescriptorWriter()

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

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

7.11.3 Member Function Documentation

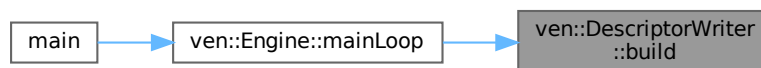
7.11.3.1 build()

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

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

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

Here is the caller graph for this function:



7.11.3.2 overwrite()

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

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

7.11.3.3 writeBuffer()

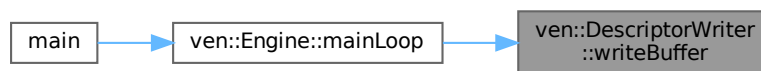
```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
    uint32_t binding,
    const VkDescriptorBufferInfo * bufferInfo)
```

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

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

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

Here is the caller graph for this function:



7.11.3.4 writelImage()

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

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

7.11.4 Member Data Documentation

7.11.4.1 m_pool

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

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

7.11.4.2 m_setLayout

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

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

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

7.11.4.3 m_writes

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

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

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

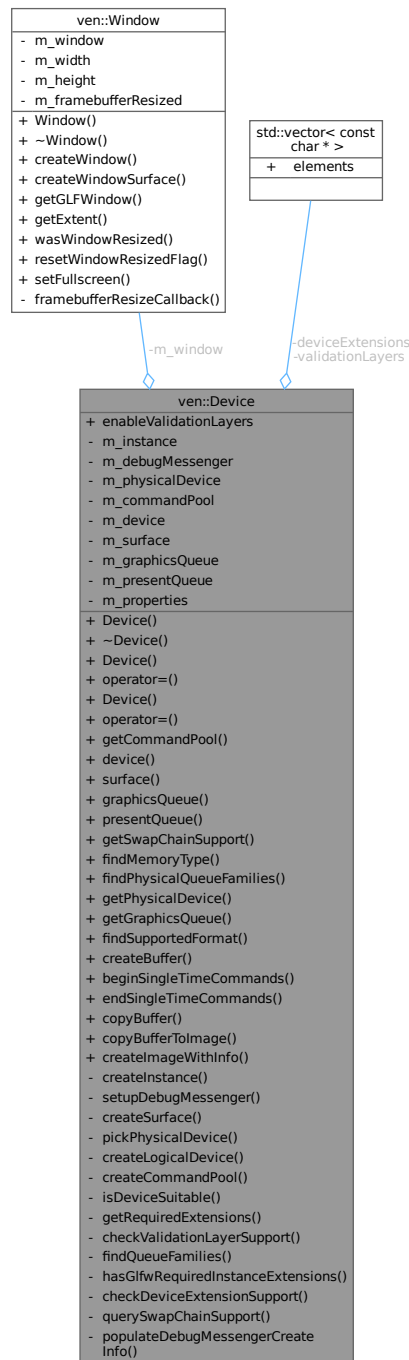
The documentation for this class was generated from the following files:

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

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

Public Attributes

- const bool [enableValidationLayers](#) = true

Private Member Functions

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

Static Private Member Functions

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

Private Attributes

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

7.12.1 Detailed Description

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

7.12.2 Constructor & Destructor Documentation

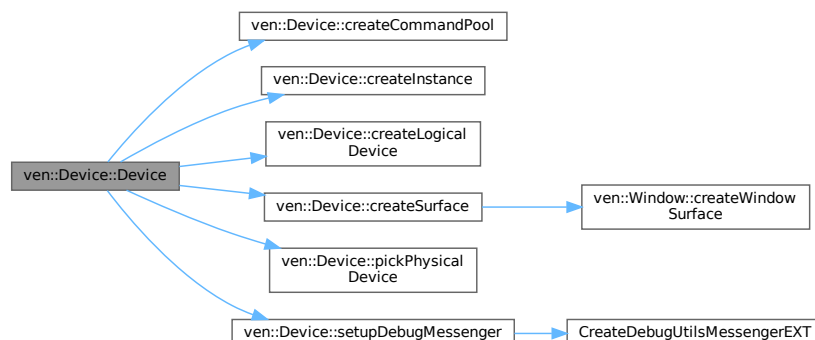
7.12.2.1 Device() [1/3]

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

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

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

Here is the call graph for this function:



7.12.2.2 ~Device()

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

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

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

Here is the call graph for this function:



7.12.2.3 Device() [2/3]

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

7.12.2.4 Device() [3/3]

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

7.12.3 Member Function Documentation

7.12.3.1 beginSingleTimeCommands()

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

Definition at line 409 of file [device.cpp](#).

7.12.3.2 checkDeviceExtensionSupport()

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

Definition at line 287 of file [device.cpp](#).

7.12.3.3 checkValidationLayerSupport()

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

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

7.12.3.4 copyBuffer()

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

Definition at line 443 of file [device.cpp](#).

7.12.3.5 copyBufferToImage()

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

Definition at line 456 of file [device.cpp](#).

7.12.3.6 createBuffer()

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

Definition at line 382 of file [device.cpp](#).

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

Here is the caller graph for this function:



7.12.3.7 createCommandPool()

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

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

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

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

Here is the caller graph for this function:



7.12.3.8 createImageWithInfo()

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

Definition at line 477 of file [device.cpp](#).

7.12.3.9 createInstance()

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

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

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

Here is the caller graph for this function:



7.12.3.10 createLogicalDevice()

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

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

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

Here is the caller graph for this function:



7.12.3.11 createSurface()

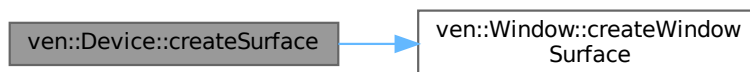
```
void ven::Device::createSurface () [inline], [private]
```

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.12 device()

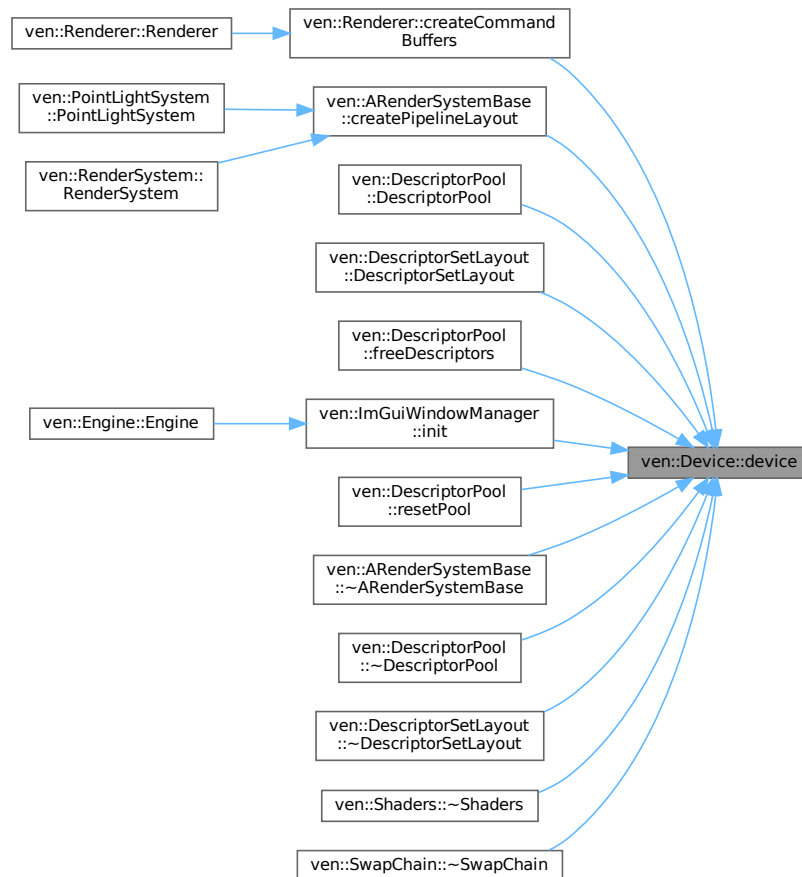
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

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

References [m_device](#).

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

Here is the caller graph for this function:



7.12.3.13 endSingleTimeCommands()

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

Definition at line 428 of file [device.cpp](#).

7.12.3.14 findMemoryType()

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

Definition at line 367 of file [device.cpp](#).

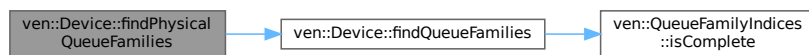
7.12.3.15 findPhysicalQueueFamilies()

```
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:



7.12.3.16 findQueueFamilies()

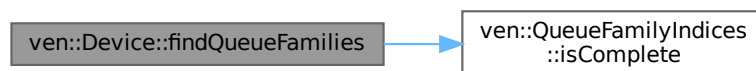
```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
    VkPhysicalDevice device) const [private]
```

Definition at line 303 of file [device.cpp](#).

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.17 findSupportedFormat()

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

Definition at line 353 of file [device.cpp](#).

7.12.3.18 getCommandPool()

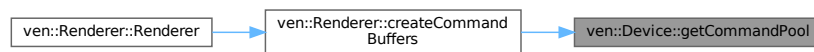
```
VkCommandPool ven::Device::getCommandPool () const [inline], [nodiscard]
```

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

References [m_commandPool](#).

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

Here is the caller graph for this function:



7.12.3.19 getGraphicsQueue()

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

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

References [m_graphicsQueue](#).

7.12.3.20 getPhysicalDevice()

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

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

References [m_physicalDevice](#).

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

Here is the caller graph for this function:



7.12.3.21 getRequiredExtensions()

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

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

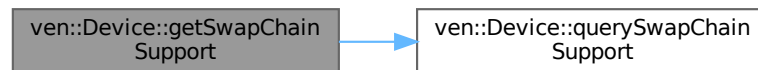
7.12.3.22 getSwapChainSupport()

```
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:

**7.12.3.23 graphicsQueue()**

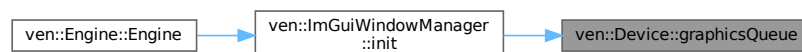
```
VkQueue ven::Device::graphicsQueue () const [inline], [nodiscard]
```

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

References [m_graphicsQueue](#).

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

Here is the caller graph for this function:

**7.12.3.24 hasGlfwRequiredInstanceExtensions()**

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

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

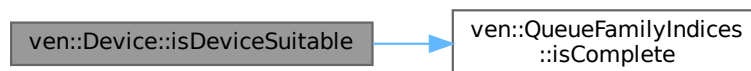
7.12.3.25 isDeviceSuitable()

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

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

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

Here is the call graph for this function:



7.12.3.26 operator=() [1/2]

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

7.12.3.27 operator=() [2/2]

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

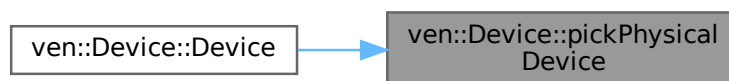
7.12.3.28 pickPhysicalDevice()

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

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

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

Here is the caller graph for this function:



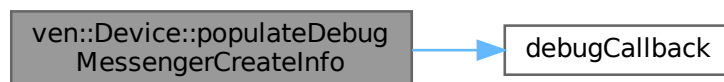
7.12.3.29 populateDebugMessengerCreateInfo()

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

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

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

Here is the call graph for this function:



7.12.3.30 presentQueue()

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

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

References [m_presentQueue](#).

7.12.3.31 querySwapChainSupport()

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

Definition at line 332 of file [device.cpp](#).

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

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

Here is the caller graph for this function:



7.12.3.32 `setupDebugMessenger()`

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

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

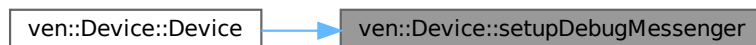
References [CreateDebugUtilsMessengerEXT\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.3.33 `surface()`

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

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

References [m_surface](#).

7.12.4 Member Data Documentation

7.12.4.1 `deviceExtensions`

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

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

7.12.4.2 enableValidationLayers

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

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

7.12.4.3 m_commandPool

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

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

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

7.12.4.4 m_debugMessenger

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

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

7.12.4.5 m_device

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

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

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

7.12.4.6 m_graphicsQueue

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

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

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

7.12.4.7 m_instance

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

Definition at line 89 of file [Device.hpp](#).

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

7.12.4.8 m_physicalDevice

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

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

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

7.12.4.9 m_presentQueue

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

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

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

7.12.4.10 m_properties

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

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

7.12.4.11 m_surface

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

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

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

7.12.4.12 m_window

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

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

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

7.12.4.13 validationLayers

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

Definition at line 101 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](#)


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

[illegible]

- **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](#)
- [Light::Map](#) [m_lights](#)
- [VkInstance](#) [m_instance](#) {nullptr}
- [VkSurfaceKHR](#) [m_surface](#) {nullptr}

7.13.1 Detailed Description

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

7.13.2 Constructor & Destructor Documentation

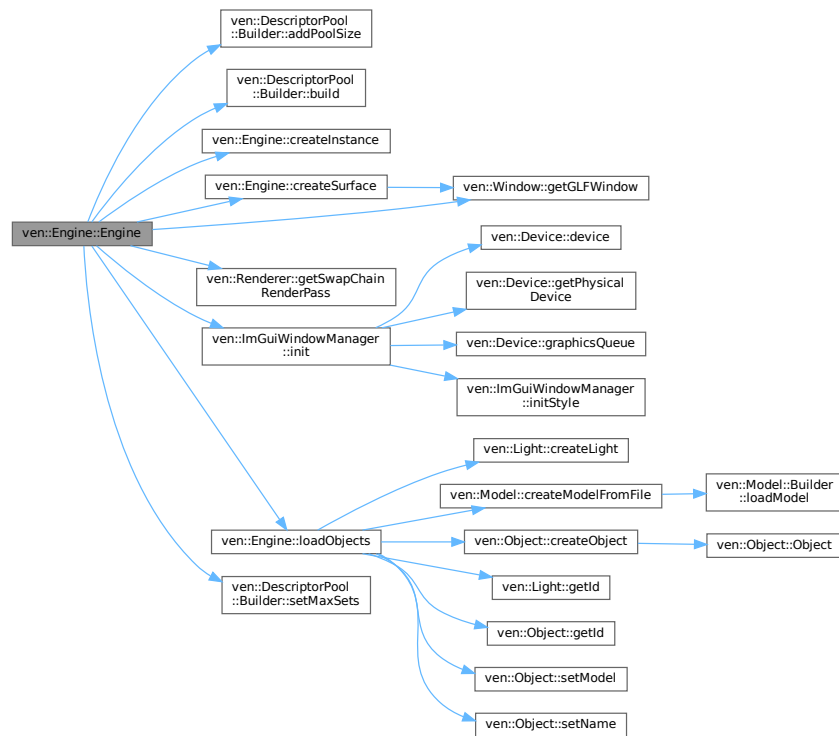
7.13.2.1 Engine() [1/2]

```
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:



7.13.2.2 ~Engine()

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

7.13.2.3 Engine() [2/2]

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

7.13.3 Member Function Documentation

7.13.3.1 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:



7.13.3.2 createSurface()

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.3 loadObjects()

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

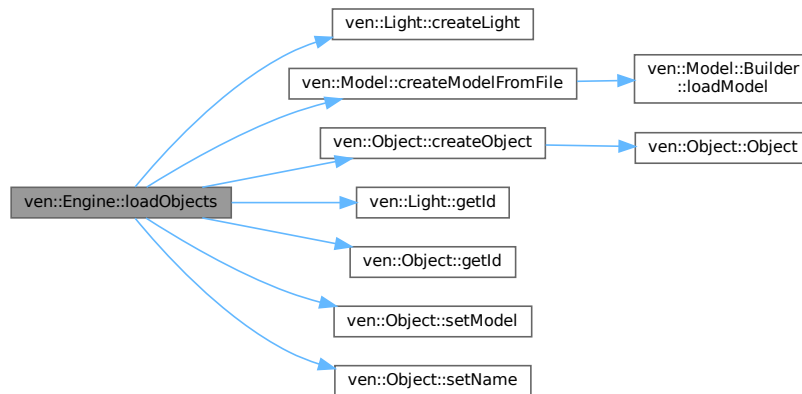
Definition at line 43 of file [engine.cpp](#).

References [ven::Colors::BLUE](#), [ven::Light::color](#), [ven::Light::createLight\(\)](#), [ven::Model::createModelFromFile\(\)](#), [ven::Object::createObject\(\)](#), [ven::Colors::CYAN](#), [ven::DEFAULT_LIGHT_INTENSITY](#), [ven::Light::getId\(\)](#), [ven::Object::getId\(\)](#),

[ven::Colors::GREEN](#), [ven::Colors::MAGENTA](#), [ven::Colors::RED](#), [ven::Transform3DComponent::scale](#), [ven::Object::setModel\(\)](#), [ven::Object::setName\(\)](#), [ven::Light::transform3D](#), [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:



7.13.3.4 mainLoop()

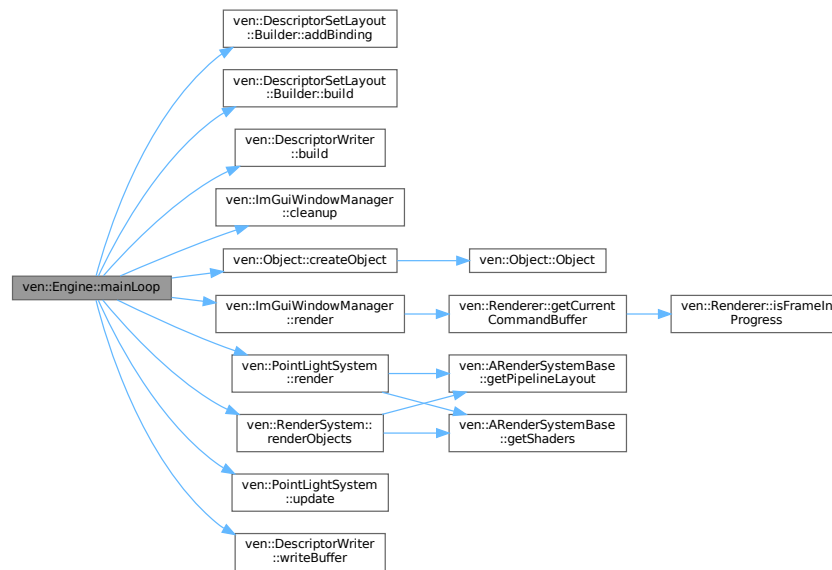
```
void ven::Engine::mainLoop ()
```

Definition at line 89 of file [engine.cpp](#).

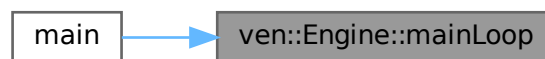
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::DescriptorWriter::build\(\)](#), [ven::ImGuiWindowManager::cleanup\(\)](#), [ven::Object::createObject\(\)](#), [ven::DEFAULT_POSITION](#), [ven::FrameInfo::frameIndex](#), [ven::SwapChain::MAX_FRAMES_IN_FLIGHT](#), [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:



7.13.3.5 operator=()

```

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

7.13.4 Member Data Documentation

7.13.4.1 m_device

```

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

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

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

7.13.4.2 m_globalPool

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

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

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

7.13.4.3 m_instance

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

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

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

7.13.4.4 m_lights

```
Light::Map ven::Engine::m_lights [private]
```

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

7.13.4.5 m_objects

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

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

7.13.4.6 m_renderer

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

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

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

7.13.4.7 m_surface

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

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

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

7.13.4.8 m_window

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

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

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

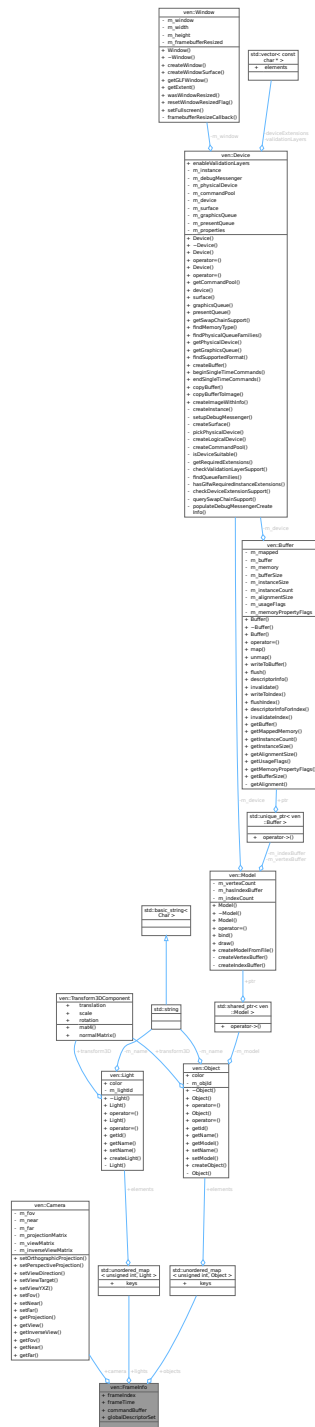
The documentation for this class was generated from the following files:

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

7.14 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**
- **Light::Map** & **lights**

7.14.1 Detailed Description

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

7.14.2 Member Data Documentation

7.14.2.1 camera

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

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

7.14.2.2 commandBuffer

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

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

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

7.14.2.3 frameIndex

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

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

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

7.14.2.4 frameTime

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

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

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

7.14.2.5 globalDescriptorSet

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

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

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

7.14.2.6 lights

`Light::Map& ven::FrameInfo::lights`

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

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

7.14.2.7 objects

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

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

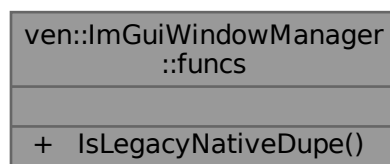
Referenced by [ven::RenderSystem::renderObjects\(\)](#).

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

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

7.15 ven::ImGuiWindowManager::funcs Struct Reference

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



Static Public Member Functions

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

7.15.1 Detailed Description

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

7.15.2 Member Function Documentation

7.15.2.1 IsLegacyNativeDupe()

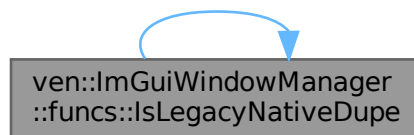
```
static bool ven::ImGuiWindowManager::funcs::IsLegacyNativeDupe (  
    const ImGuiKey key) [inline], [static]
```

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

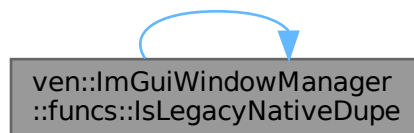
References [IsLegacyNativeDupe\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



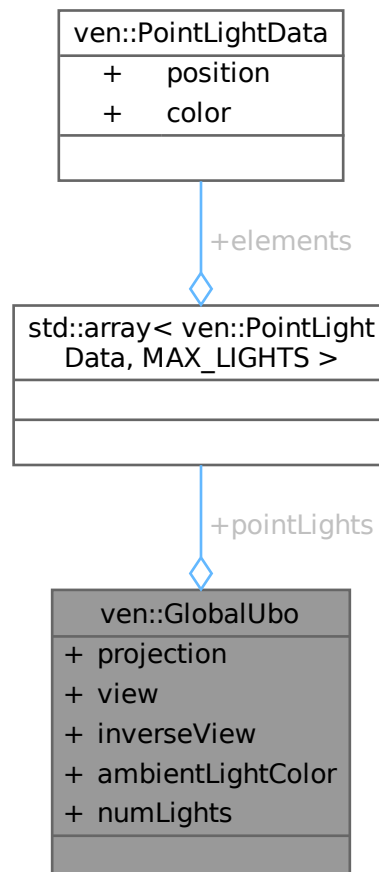
The documentation for this struct was generated from the following file:

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

7.16 ven::GlobalUbo Struct Reference

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

Collaboration diagram for ven::GlobalUbo:



Public Attributes

- glm::mat4 [projection](#) {1.F}
- glm::mat4 [view](#) {1.F}
- glm::mat4 [inverseView](#) {1.F}
- glm::vec4 [ambientLightColor](#) {DEFAULT_AMBIENT_LIGHT_COLOR}
- std::array< [PointLightData](#), MAX_LIGHTS > [pointLights](#)
- int [numLights](#)

7.16.1 Detailed Description

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

7.16.2 Member Data Documentation

7.16.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {DEFAULT_AMBIENT_LIGHT_COLOR}
```

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

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

7.16.2.2 inverseView

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

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

7.16.2.3 numLights

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

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

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

7.16.2.4 pointLights

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

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

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

7.16.2.5 projection

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

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

7.16.2.6 view

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

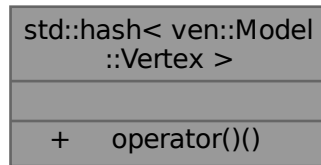
Definition at line 31 of file [FrameInfo.hpp](#).

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

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

7.17 std::hash< ven::Model::Vertex > Struct Reference

Collaboration diagram for std::hash< ven::Model::Vertex >:



Public Member Functions

- `size_t operator() (ven::Model::Vertex const &vertex) const noexcept`

7.17.1 Detailed Description

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

7.17.2 Member Function Documentation

7.17.2.1 operator()()

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

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

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

Here is the call graph for this function:



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

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

7.18 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"> + ImGuiWindowManager() + ~ImGuiWindowManager() + ImGuiWindowManager() + operator=() + init() + render() + cleanup() - initStyle() - renderFrameWindow() - cameraSection() - inputsSection() - rendererSection() - devicePropertiesSection() - objectsSection() - lightsSection()

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, const [Device](#) *device, VkRenderPass render↔ Pass)
- static void [render](#) ([Renderer](#) *renderer, std::unordered_map< unsigned int, [Object](#) > &objects, std↔ ::unordered_map< unsigned int, [Light](#) > &lights, const ImGuiIO &io, [Object](#) &cameraObj, [Camera](#) &camera, [KeyboardController](#) &cameraController, VkPhysicalDevice physicalDevice, [GlobalUbo](#) &ubo)
- static void [cleanup](#) ()

Static Private Member Functions

- static void [initStyle](#) ()
- static void [renderFrameWindow](#) (const ImGuiIO &io)
- static void [cameraSection](#) ([Object](#) &cameraObj, [Camera](#) &camera, [KeyboardController](#) &cameraController)
- static void [inputsSection](#) (const ImGuiIO &io)
- static void [rendererSection](#) ([Renderer](#) *renderer, [GlobalUbo](#) &ubo)
- static void [devicePropertiesSection](#) (VkPhysicalDeviceProperties deviceProperties)
- static void [objectsSection](#) (std::unordered_map< unsigned int, [Object](#) > &objects)
- static void [lightsSection](#) (std::unordered_map< unsigned int, [Light](#) > &lights)

7.18.1 Detailed Description

Class for ImGui window manager.

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

7.18.2 Constructor & Destructor Documentation

7.18.2.1 ImGuiWindowManager() [1/2]

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

7.18.2.2 ~ImGuiWindowManager()

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

7.18.2.3 ImGuiWindowManager() [2/2]

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

7.18.3 Member Function Documentation

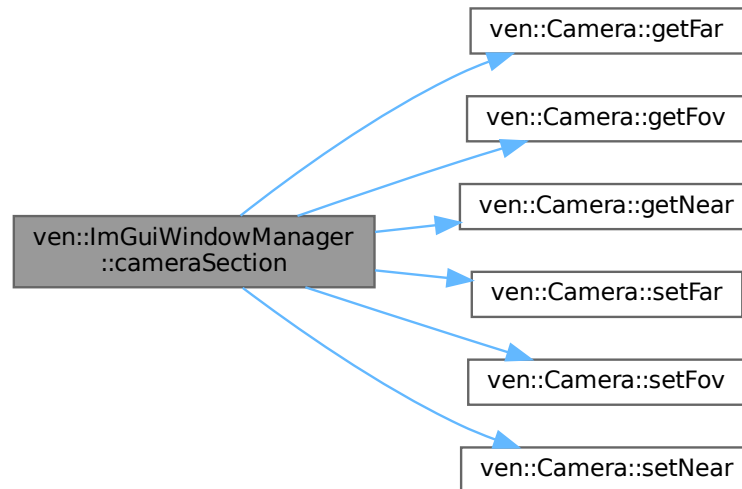
7.18.3.1 cameraSection()

```
void ven::ImGuiWindowManager::cameraSection (
    Object & cameraObj,
    Camera & camera,
    KeyboardController & cameraController) [static], [private]
```

Definition at line 112 of file [render.cpp](#).

References [ven::DEFAULT_FAR](#), [ven::DEFAULT_FOV](#), [ven::DEFAULT_LOOK_SPEED](#), [ven::DEFAULT_MOVE_SPEED](#), [ven::DEFAULT_NEAR](#), [ven::DEFAULT_POSITION](#), [ven::DEFAULT_ROTATION](#), [ven::Camera::getFar\(\)](#), [ven::Camera::getFov\(\)](#), [ven::Camera::getNear\(\)](#), [ven::KeyboardController::m_lookSpeed](#), [ven::KeyboardController::m_moveSpeed](#), [ven::Transform3DComponent::rotation](#), [ven::Camera::setFar\(\)](#), [ven::Camera::setFov\(\)](#), [ven::Camera::setNear\(\)](#), [ven::Object::transform3D](#), and [ven::Transform3DComponent::translation](#).

Here is the call graph for this function:



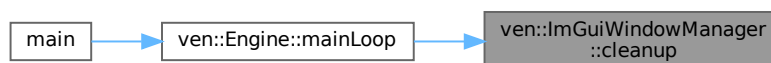
7.18.3.2 cleanup()

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

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

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

Here is the caller graph for this function:



7.18.3.3 devicePropertiesSection()

```
void ven::ImGuiWindowManager::devicePropertiesSection (
    VkPhysicalDeviceProperties deviceProperties) [static], [private]
```

Definition at line 252 of file [render.cpp](#).

7.18.3.4 init()

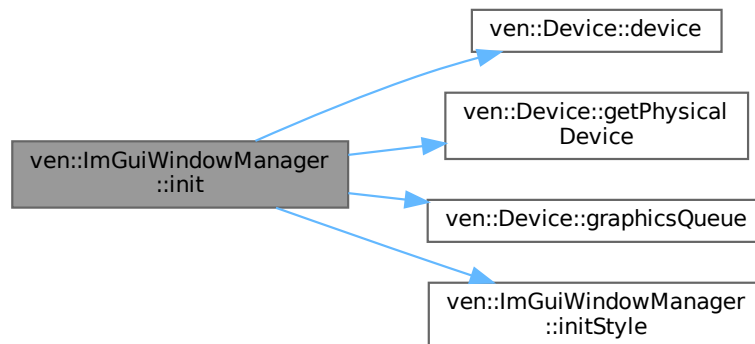
```
void ven::ImGuiWindowManager::init (
    GLFWwindow * window,
    VkInstance instance,
    const Device * device,
    VkRenderPass renderPass) [static]
```

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.18.3.5 initStyle()

```
void ven::ImGuiWindowManager::initStyle () [static], [private]
```

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

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

Here is the caller graph for this function:



7.18.3.6 inputsSection()

```
void ven::ImGuiWindowManager::inputsSection (
    const ImGuiIO & io) [static], [private]
```

Definition at line 230 of file [render.cpp](#).

7.18.3.7 lightsSection()

```
void ven::ImGuiWindowManager::lightsSection (
    std::unordered_map< unsigned int, Light > & lights) [static], [private]
```

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

References [ven::Colors::COLORS](#), and [ven::DEFAULT_LIGHT_INTENSITY](#).

7.18.3.8 objectsSection()

```
void ven::ImGuiWindowManager::objectsSection (
    std::unordered_map< unsigned int, Object > & objects) [static], [private]
```

Definition at line 159 of file [render.cpp](#).

References [ven::Colors::GRAY](#).

7.18.3.9 operator=()

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

7.18.3.10 render()

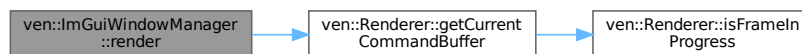
```
void ven::ImGuiWindowManager::render (
    Renderer * renderer,
    std::unordered_map< unsigned int, Object > & objects,
    std::unordered_map< unsigned int, Light > & lights,
    const ImGuiIO & io,
    Object & cameraObj,
    Camera & camera,
    KeyboardController & cameraController,
    VkPhysicalDevice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

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

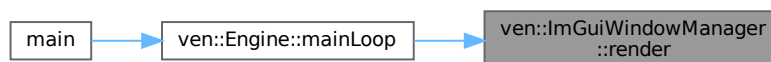
References [ven::Renderer::getCurrentCommandBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



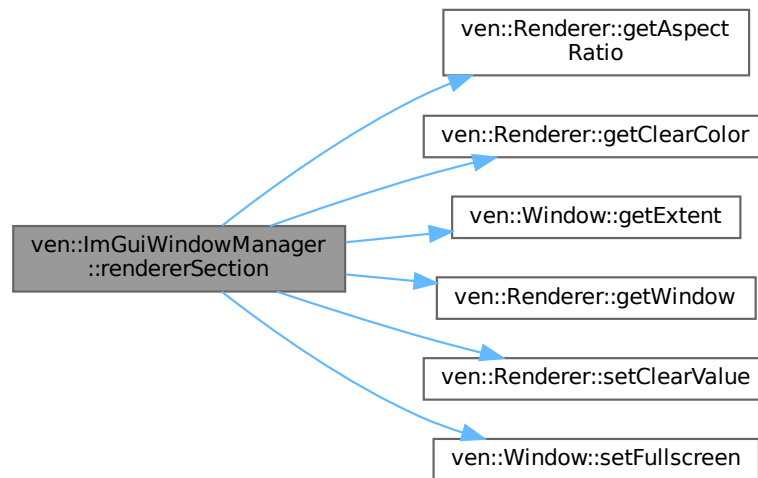
7.18.3.11 rendererSection()

```
void ven::ImGuiWindowManager::rendererSection (
    Renderer * renderer,
    GlobalUbo & ubo) [static], [private]
```

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

References [ven::GlobalUbo::ambientLightColor](#), [ven::Colors::CLEAR_COLORS](#), [ven::Colors::COLORS](#), [ven::DEFAULT_AMBIENT_L](#), [ven::Renderer::getAspectRatio\(\)](#), [ven::Renderer::getClearColor\(\)](#), [ven::Window::getExtent\(\)](#), [ven::Renderer::getWindow\(\)](#), [ven::Renderer::setClearColor\(\)](#), and [ven::Window::setFullscreen\(\)](#).

Here is the call graph for this function:



7.18.3.12 renderFrameWindow()

```
void ven::ImGuiWindowManager::renderFrameWindow (
    const ImGuiIO & io) [static], [private]
```

Definition at line 41 of file [render.cpp](#).

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

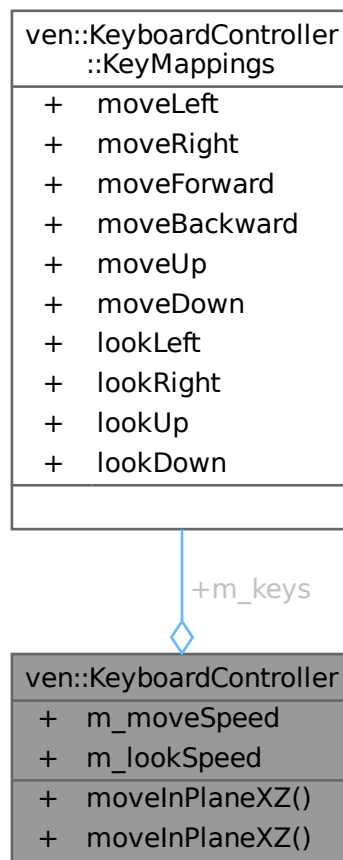
- [/home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/init.cpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/render.cpp](#)

7.19 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
- void [moveInPlaneXZ](#) (GLFWwindow *window, float dt, [Light](#) &light, bool *showDebugWindow) const

Public Attributes

- [KeyMappings](#) m_keys {}
- float [m_moveSpeed](#) {DEFAULT_MOVE_SPEED}
- float [m_lookSpeed](#) {DEFAULT_LOOK_SPEED}

7.19.1 Detailed Description

Class for keyboard controller.

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

7.19.2 Member Function Documentation

7.19.2.1 moveInPlaneXZ() [1/2]

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

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

References [ven::Transform3DComponent::rotation](#), [ven::Light::transform3D](#), and [ven::Transform3DComponent::translation](#).

7.19.2.2 moveInPlaneXZ() [2/2]

```
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](#), [ven::KeyboardController::KeyMappings::moveUp](#), and [ven::KeyboardController::KeyMappings::moveDown](#).

7.19.3 Member Data Documentation

7.19.3.1 m_keys

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

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

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

7.19.3.2 m_lookSpeed

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

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

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

7.19.3.3 m_moveSpeed

float ven::KeyboardController::m_moveSpeed {DEFAULT_MOVE_SPEED}

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

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

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](#)

7.20 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

7.20.1 Detailed Description

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

7.20.2 Member Data Documentation

7.20.2.1 lookDown

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

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

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

7.20.2.2 lookLeft

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

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

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

7.20.2.3 lookRight

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

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

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

7.20.2.4 lookUp

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

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

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

7.20.2.5 moveBackward

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

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

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

7.20.2.6 moveDown

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

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

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

7.20.2.7 moveForward

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

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

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

7.20.2.8 moveLeft

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

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

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

7.20.2.9 moveRight

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

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

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

7.20.2.10 moveUp

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

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

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

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

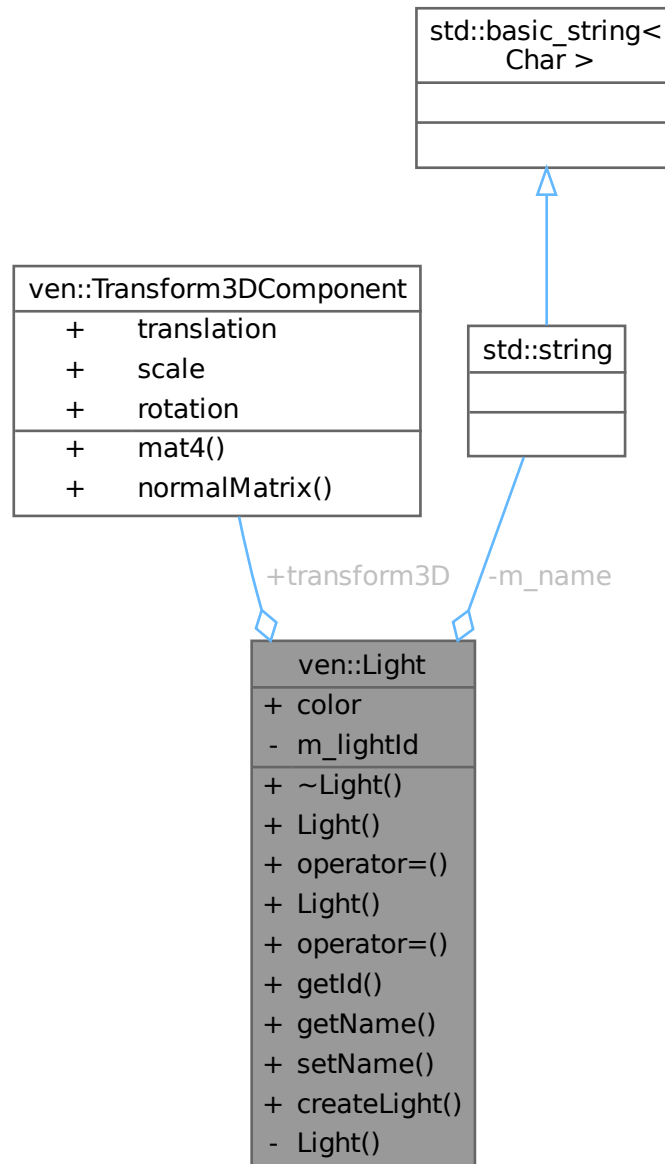
- [/home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp](#)

7.21 ven::Light Class Reference

Class for light.

```
#include <Light.hpp>
```

Collaboration diagram for ven::Light:



Public Types

- using `Map` = `std::unordered_map<unsigned int, Light>`

Public Member Functions

- [~Light](#) ()=default
- [Light](#) (const [Light](#) &)=delete
- [Light](#) & [operator=](#) (const [Light](#) &)=delete
- [Light](#) ([Light](#) &&)=default
- [Light](#) & [operator=](#) ([Light](#) &&)=default
- unsigned int [getId](#) () const
- std::string [getName](#) () const
- void [setName](#) (const std::string &name)

Static Public Member Functions

- static [Light](#) [createLight](#) (float radius=[DEFAULT_LIGHT_RADIUS](#), glm::vec4 [color](#)=[DEFAULT_LIGHT_COLOR](#))

Public Attributes

- glm::vec4 [color](#) {[DEFAULT_LIGHT_COLOR](#)}
- [Transform3DComponent](#) [transform3D](#) {}

Private Member Functions

- [Light](#) (const unsigned int [lightId](#))

Private Attributes

- unsigned int [m_lightId](#)
- std::string [m_name](#) {"point light"}

7.21.1 Detailed Description

Class for light.

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

7.21.2 Member Typedef Documentation

7.21.2.1 Map

```
using ven::Light::Map = std::unordered_map<unsigned int, Light>
```

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

7.21.3 Constructor & Destructor Documentation

7.21.3.1 ~Light()

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

7.21.3.2 Light() [1/3]

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

7.21.3.3 Light() [2/3]

```
ven::Light::Light (
    Light && ) [default]
```

7.21.3.4 Light() [3/3]

```
ven::Light::Light (
    const unsigned int lightId) [inline], [explicit], [private]
```

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

7.21.4 Member Function Documentation

7.21.4.1 createLight()

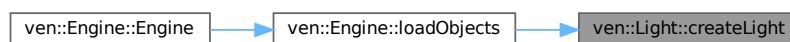
```
ven::Light ven::Light::createLight (
    float radius = DEFAULT\_LIGHT\_RADIUS,
    glm::vec4 color = DEFAULT\_LIGHT\_COLOR) [static]
```

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

References [color](#), [ven::Transform3DComponent::scale](#), and [transform3D](#).

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

Here is the caller graph for this function:



7.21.4.2 getId()

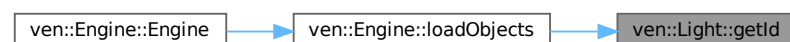
```
unsigned int ven::Light::getId () const [inline], [nodiscard]
```

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

References [m_lightId](#).

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

Here is the caller graph for this function:



7.21.4.3 getName()

```
std::string ven::Light::getName () const [inline], [nodiscard]
```

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

References [m_name](#).

7.21.4.4 operator=() [1/2]

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

7.21.4.5 operator=() [2/2]

```
Light & ven::Light::operator= (  
    Light && ) [default]
```

7.21.4.6 setName()

```
void ven::Light::setName (  
    const std::string & name) [inline]
```

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

References [m_name](#).

7.21.5 Member Data Documentation

7.21.5.1 color

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

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

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

7.21.5.2 m_lightId

```
unsigned int ven::Light::m_lightId [private]
```

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

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

7.21.5.3 m_name

```
std::string ven::Light::m_name {"point light"} [private]
```

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

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.21.5.4 transform3D

```
Transform3DComponent ven::Light::transform3D {}
```

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

Referenced by [createLight\(\)](#), [ven::Engine::loadObjects\(\)](#), and [ven::KeyboardController::moveInPlaneXZ\(\)](#).

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

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

7.22 ven::LightPushConstantData Struct Reference

```
#include <PointLightSystem.hpp>
```

Collaboration diagram for `ven::LightPushConstantData`:

ven::LightPushConstantData	
+	position
+	color
+	radius

Public Attributes

- `glm::vec4` [position](#) {}
- `glm::vec4` [color](#) {}
- `float` [radius](#)

7.22.1 Detailed Description

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

7.22.2 Member Data Documentation

7.22.2.1 color

```
glm::vec4 ven::LightPushConstantData::color {}
```

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

7.22.2.2 position

```
glm::vec4 ven::LightPushConstantData::position {}
```

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

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

7.22.2.3 radius

```
float ven::LightPushConstantData::radius
```

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

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

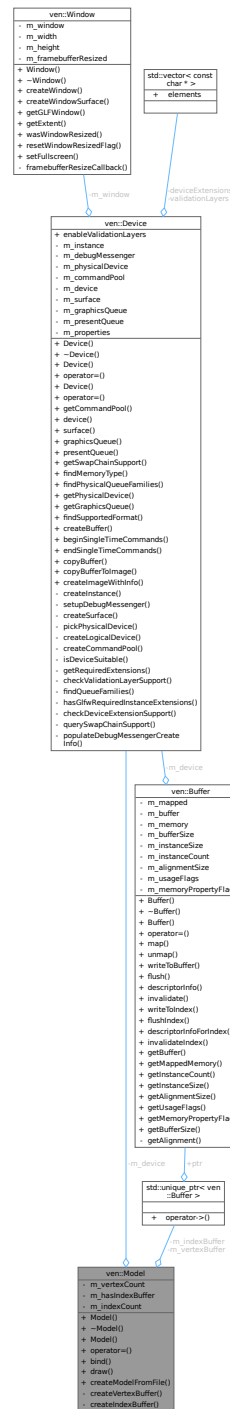
- [/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp](#)

7.23 ven::Model Class Reference

Class for model.

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

Collaboration diagram for `ven::Model`:



Classes

- struct [Builder](#)
- struct [Vertex](#)

Public Member Functions

- [Model](#) ([Device](#) &device, const [Builder](#) &builder)

- [~Model](#) ()
- [Model](#) (const [Model](#) &)=delete
- void [operator=](#) (const [Model](#) &)=delete
- void [bind](#) (VkCommandBuffer commandBuffer) const
- void [draw](#) (VkCommandBuffer commandBuffer) const

Static Public Member Functions

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

Private Member Functions

- void [createVertexBuffer](#) (const std::vector< [Vertex](#) > &vertices)
- void [createIndexBuffer](#) (const std::vector< uint32_t > &indices)

Private Attributes

- [Device](#) & [m_device](#)
- std::unique_ptr< [Buffer](#) > [m_vertexBuffer](#)
- uint32_t [m_vertexCount](#)
- bool [m_hasIndexBuffer](#) {false}
- std::unique_ptr< [Buffer](#) > [m_indexBuffer](#)
- uint32_t [m_indexCount](#)

7.23.1 Detailed Description

Class for model.

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

7.23.2 Constructor & Destructor Documentation

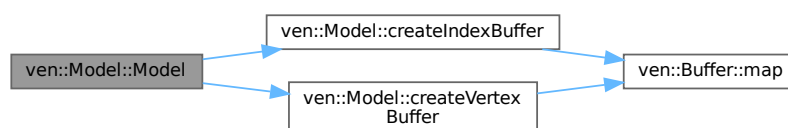
7.23.2.1 Model() [1/2]

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

Definition at line 23 of file [model.cpp](#).

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

Here is the call graph for this function:



7.23.2.2 ~Model()

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

7.23.2.3 Model() [2/2]

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

7.23.3 Member Function Documentation

7.23.3.1 bind()

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

Definition at line 78 of file [model.cpp](#).

7.23.3.2 createIndexBuffer()

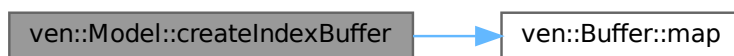
```
void ven::Model::createIndexBuffer (
    const std::vector< uint32_t > & indices) [private]
```

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

References [ven::Buffer::map\(\)](#).

Referenced by [Model\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.3 createModelFromFile()

```
std::unique_ptr< ven::Model > ven::Model::createModelFromFile (
    Device & device,
    const std::string & filename) [static]
```

Definition at line 89 of file [model.cpp](#).

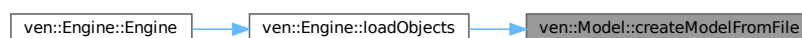
References [ven::Model::Builder::loadModel\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.4 createVertexBuffer()

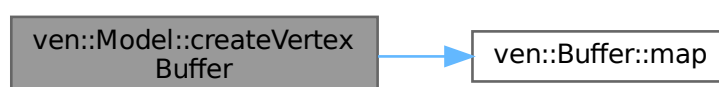
```
void ven::Model::createVertexBuffer (
    const std::vector< Vertex > & vertices) [private]
```

Definition at line 31 of file [model.cpp](#).

References [ven::Buffer::map\(\)](#).

Referenced by [Model\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.5 draw()

```
void ven::Model::draw (
    VkCommandBuffer commandBuffer) const
```

Definition at line 69 of file [model.cpp](#).

7.23.3.6 operator=()

```
void ven::Model::operator= (
    const Model & ) [delete]
```

7.23.4 Member Data Documentation

7.23.4.1 m_device

```
Device& ven::Model::m_device [private]
```

Definition at line 62 of file [Model.hpp](#).

7.23.4.2 m_hasIndexBuffer

```
bool ven::Model::m_hasIndexBuffer {false} [private]
```

Definition at line 66 of file [Model.hpp](#).

7.23.4.3 m_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

Definition at line 67 of file [Model.hpp](#).

7.23.4.4 m_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

Definition at line 68 of file [Model.hpp](#).

7.23.4.5 m_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

Definition at line 63 of file [Model.hpp](#).

7.23.4.6 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:

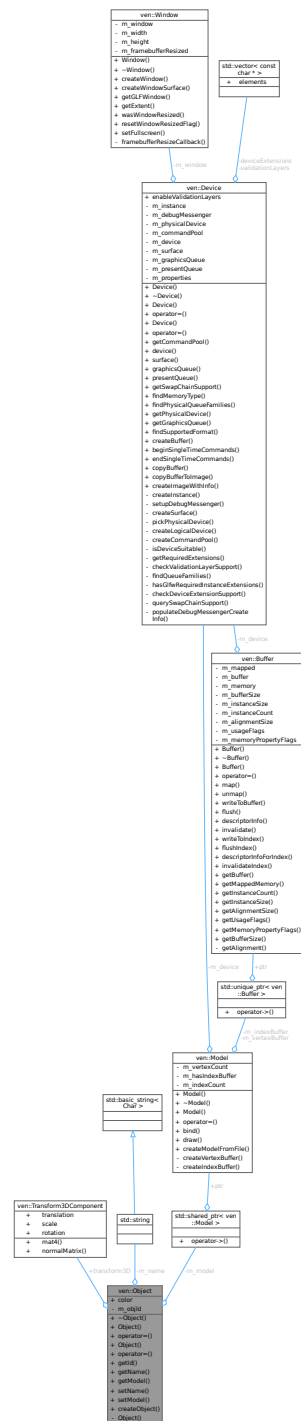
- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.24 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
- std::string [getName](#) () const
- std::shared_ptr< [Model](#) > [getModel](#) () const
- void [setName](#) (const std::string &name)
- void [setModel](#) (const std::shared_ptr< [Model](#) > &model)

Static Public Member Functions

- static [Object](#) [createObject](#) ()

Public Attributes

- glm::vec3 [color](#) {}
- [Transform3DComponent](#) [transform3D](#) {}

Private Member Functions

- [Object](#) (const unsigned int objId)

Private Attributes

- unsigned int [m_objId](#)
- std::string [m_name](#) {}
- std::shared_ptr< [Model](#) > [m_model](#) {}

7.24.1 Detailed Description

Class for object.

Definition at line 24 of file [Object.hpp](#).

7.24.2 Member Typedef Documentation

7.24.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

Definition at line 28 of file [Object.hpp](#).

7.24.3 Constructor & Destructor Documentation

7.24.3.1 ~Object()

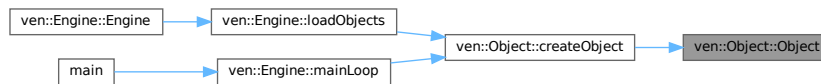
```
ven::Object::~Object () [default]
```

7.24.3.2 `Object()` [1/3]

```
ven::Object::Object (
    const Object & ) [delete]
```

Referenced by [createObject\(\)](#).

Here is the caller graph for this function:



7.24.3.3 `Object()` [2/3]

```
ven::Object::Object (
    Object && ) [default]
```

7.24.3.4 `Object()` [3/3]

```
ven::Object::Object (
    const unsigned int objId) [inline], [explicit], [private]
```

Definition at line 51 of file [Object.hpp](#).

7.24.4 Member Function Documentation

7.24.4.1 `createObject()`

```
static Object ven::Object::createObject () [inline], [static]
```

Definition at line 37 of file [Object.hpp](#).

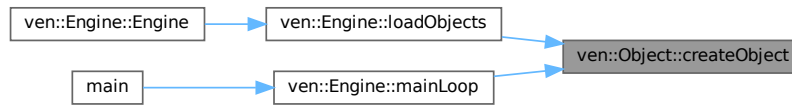
References [Object\(\)](#).

Referenced by [ven::Engine::loadObjects\(\)](#), and [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.4.2 getId()

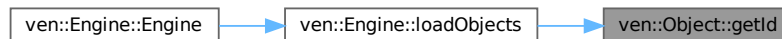
```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

Definition at line 39 of file [Object.hpp](#).

References [m_objId](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.4.3 getModel()

```
std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]
```

Definition at line 41 of file [Object.hpp](#).

References [m_model](#).

7.24.4.4 getName()

```
std::string ven::Object::getName () const [inline], [nodiscard]
```

Definition at line 40 of file [Object.hpp](#).

References [m_name](#).

7.24.4.5 operator=() [1/2]

```
Object & ven::Object::operator= (
    const Object & ) [delete]
```

7.24.4.6 operator=() [2/2]

```
Object & ven::Object::operator= (
    Object && ) [default]
```

7.24.4.7 setModel()

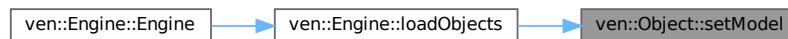
```
void ven::Object::setModel (
    const std::shared_ptr< Model > & model) [inline]
```

Definition at line 44 of file [Object.hpp](#).

References [m_model](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.4.8 setName()

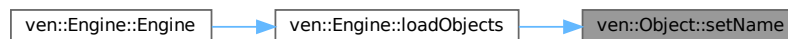
```
void ven::Object::setName (
    const std::string & name) [inline]
```

Definition at line 43 of file [Object.hpp](#).

References [m_name](#).

Referenced by [ven::Engine::loadObjects\(\)](#).

Here is the caller graph for this function:



7.24.5 Member Data Documentation

7.24.5.1 color

```
glm::vec3 ven::Object::color {}
```

Definition at line 46 of file [Object.hpp](#).

7.24.5.2 m_model

```
std::shared_ptr<Model> ven::Object::m_model {} [private]
```

Definition at line 55 of file [Object.hpp](#).

Referenced by [getModel\(\)](#), and [setModel\(\)](#).

7.24.5.3 m_name

```
std::string ven::Object::m_name {} [private]
```

Definition at line 54 of file [Object.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.24.5.4 m_objId

```
unsigned int ven::Object::m_objId [private]
```

Definition at line 53 of file [Object.hpp](#).

Referenced by [getId\(\)](#).

7.24.5.5 transform3D

```
Transform3DComponent ven::Object::transform3D {}
```

Definition at line 47 of file [Object.hpp](#).

Referenced by [ven::ImGuiWindowManager::cameraSection\(\)](#), [ven::Engine::loadObjects\(\)](#), and [ven::Engine::mainLoop\(\)](#).

The documentation for this class was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#)

7.25 ven::ObjectPushConstantData Struct Reference

```
#include <RenderSystem.hpp>
```

Collaboration diagram for ven::ObjectPushConstantData:

ven::ObjectPushConstantData	
+	modelMatrix
+	normalMatrix

Public Attributes

- glm::mat4 [modelMatrix](#) {}
- glm::mat4 [normalMatrix](#) {}

7.25.1 Detailed Description

Definition at line 14 of file [RenderSystem.hpp](#).

7.25.2 Member Data Documentation

7.25.2.1 modelMatrix

```
glm::mat4 ven::ObjectPushConstantData::modelMatrix {}
```

Definition at line 15 of file [RenderSystem.hpp](#).

Referenced by [ven::RenderSystem::renderObjects\(\)](#).

7.25.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

Definition at line 16 of file [RenderSystem.hpp](#).

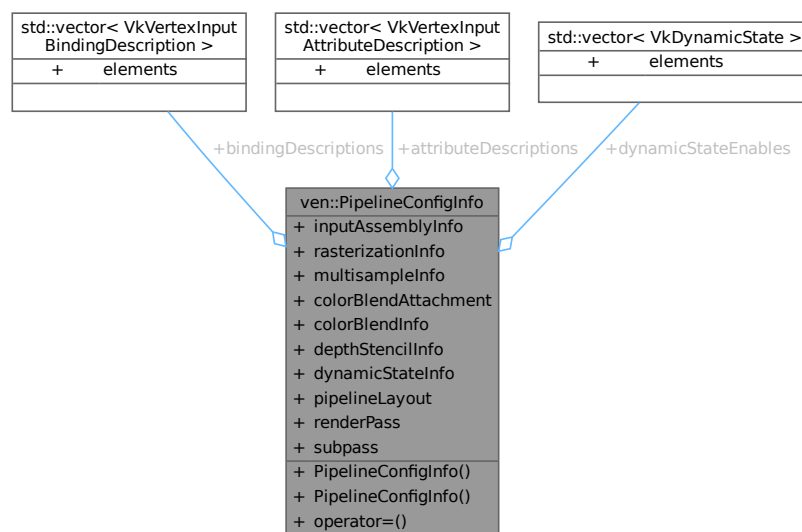
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/System/[RenderSystem.hpp](#)

7.26 ven::PipelineConfigInfo Struct Reference

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::PipelineConfigInfo:



Public Member Functions

- [PipelineConfigInfo](#) ()=default
- [PipelineConfigInfo](#) (const [PipelineConfigInfo](#) &)=delete
- [PipelineConfigInfo](#) & operator= (const [PipelineConfigInfo](#) &)=delete

Public Attributes

- std::vector< [VkVertexInputBindingDescription](#) > [bindingDescriptions](#)
- std::vector< [VkVertexInputAttributeDescription](#) > [attributeDescriptions](#)
- [VkPipelineInputAssemblyStateCreateInfo](#) [inputAssemblyInfo](#) {}
- [VkPipelineRasterizationStateCreateInfo](#) [rasterizationInfo](#) {}
- [VkPipelineMultisampleStateCreateInfo](#) [multisampleInfo](#) {}
- [VkPipelineColorBlendAttachmentState](#) [colorBlendAttachment](#) {}
- [VkPipelineColorBlendStateCreateInfo](#) [colorBlendInfo](#) {}
- [VkPipelineDepthStencilStateCreateInfo](#) [depthStencilInfo](#) {}
- std::vector< [VkDynamicState](#) > [dynamicStateEnables](#)
- [VkPipelineDynamicStateCreateInfo](#) [dynamicStateInfo](#) {}
- [VkPipelineLayout](#) [pipelineLayout](#) = nullptr
- [VkRenderPass](#) [renderPass](#) = nullptr
- uint32_t [subpass](#) = 0

7.26.1 Detailed Description

Definition at line 21 of file [Shaders.hpp](#).

7.26.2 Constructor & Destructor Documentation

7.26.2.1 PipelineConfigInfo() [1/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo () [default]
```

7.26.2.2 PipelineConfigInfo() [2/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo (
    const PipelineConfigInfo & ) [delete]
```

7.26.3 Member Function Documentation

7.26.3.1 operator=()

```
PipelineConfigInfo & ven::PipelineConfigInfo::operator= (
    const PipelineConfigInfo & ) [delete]
```

7.26.4 Member Data Documentation

7.26.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

Definition at line 27 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

Definition at line 26 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

Definition at line 31 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

Definition at line 32 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

Definition at line 33 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

Definition at line 34 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

Definition at line 35 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

Definition at line 28 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

Definition at line 30 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

Definition at line 36 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.26.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

Definition at line 29 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.26.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

Definition at line 37 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.26.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 38 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

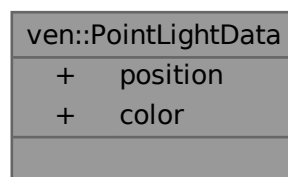
The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)

7.27 ven::PointLightData Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for `ven::PointLightData`:



Public Attributes

- `glm::vec4` [position](#) {}
- `glm::vec4` [color](#) {}

7.27.1 Detailed Description

Definition at line 22 of file [FrameInfo.hpp](#).

7.27.2 Member Data Documentation

7.27.2.1 color

```
glm::vec4 ven::PointLightData::color {}
```

Definition at line 25 of file [FrameInfo.hpp](#).

7.27.2.2 position

```
glm::vec4 ven::PointLightData::position {}
```

Definition at line 24 of file [FrameInfo.hpp](#).

The documentation for this struct was generated from the following file:

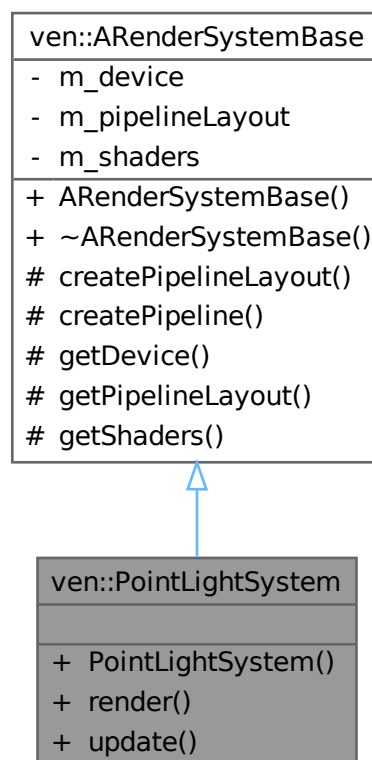
- [/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp](#)

7.28 ven::PointLightSystem Class Reference

Class for point light system.

```
#include <PointLightSystem.hpp>
```

Inheritance diagram for ven::PointLightSystem:



Collaboration diagram for `ven::PointLightSystem`:



Public Member Functions

- `PointLightSystem` (`Device` &device, const `VkRenderPass` renderPass, const `VkDescriptorSetLayout` globalSetLayout)
- void `render` (const `FramelInfo` &frameInfo) const

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- [~ARenderSystemBase](#) ()

Static Public Member Functions

- static void [update](#) (const [FrameInfo](#) &frameInfo, [GlobalUbo](#) &ubo)

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

7.28.1 Detailed Description

Class for point light system.

Definition at line 25 of file [PointLightSystem.hpp](#).

7.28.2 Constructor & Destructor Documentation

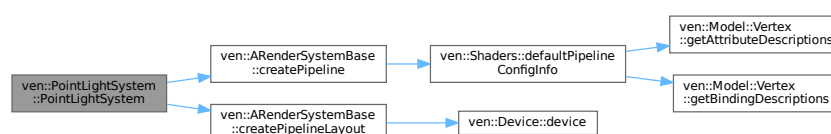
7.28.2.1 PointLightSystem()

```
ven::PointLightSystem::PointLightSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 29 of file [PointLightSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.28.3 Member Function Documentation

7.28.3.1 render()

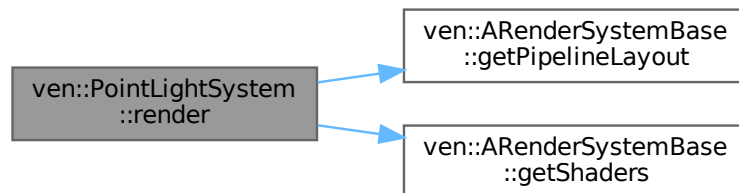
```
void ven::PointLightSystem::render (
    const FrameInfo & frameInfo) const
```

Definition at line 5 of file [pointLightSystem.cpp](#).

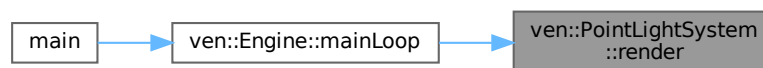
References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::FrameInfo::lights](#), and [ven::LightPushConstantData::position](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.28.3.2 update()

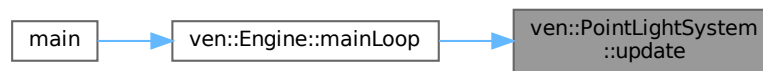
```
void ven::PointLightSystem::update (
    const FrameInfo & frameInfo,
    GlobalUbo & ubo) [static]
```

Definition at line 21 of file [pointLightSystem.cpp](#).

References [ven::FrameInfo::frameTime](#), [ven::FrameInfo::lights](#), [ven::MAX_LIGHTS](#), [ven::GlobalUbo::numLights](#), and [ven::GlobalUbo::pointLights](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



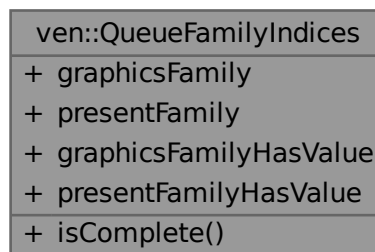
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp](#)

7.29 ven::QueueFamilyIndices Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::QueueFamilyIndices:



Public Member Functions

- bool [isComplete](#) () const

Public Attributes

- uint32_t [graphicsFamily](#) {}
- uint32_t [presentFamily](#) {}
- bool [graphicsFamilyHasValue](#) = false
- bool [presentFamilyHasValue](#) = false

7.29.1 Detailed Description

Definition at line 21 of file [Device.hpp](#).

7.29.2 Member Function Documentation

7.29.2.1 isComplete()

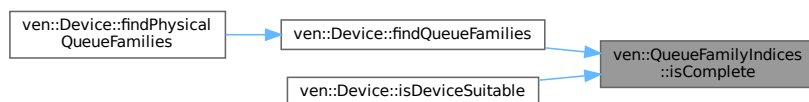
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

Definition at line 26 of file [Device.hpp](#).

References [graphicsFamilyHasValue](#), and [presentFamilyHasValue](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [ven::Device::isDeviceSuitable\(\)](#).

Here is the caller graph for this function:



7.29.3 Member Data Documentation

7.29.3.1 graphicsFamily

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

Definition at line 22 of file [Device.hpp](#).

Referenced by [ven::Device::createCommandPool\(\)](#), and [ven::Device::findQueueFamilies\(\)](#).

7.29.3.2 graphicsFamilyHasValue

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 24 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

7.29.3.3 presentFamily

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

Definition at line 23 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.29.3.4 presentFamilyHasValue

```
bool ven::QueueFamilyIndices::presentFamilyHasValue = false
```

Definition at line 25 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

The documentation for this struct was generated from the following file:

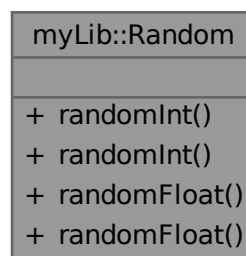
- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

7.30 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](#) ()

7.30.1 Detailed Description

Class for random number generation.

Definition at line 21 of file [Random.hpp](#).

7.30.2 Member Function Documentation

7.30.2.1 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:



7.30.2.2 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](#).

7.30.2.3 `randomInt()` [1/2]

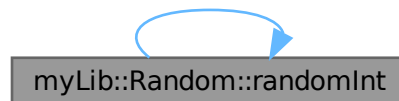
```
static int myLib::Random::randomInt () [inline], [static]
```

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:

**7.30.2.4** `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](#)

```
#include <Renderer.hpp>
```

Collaboration diagram for `ven::Renderer`:



- Generated by Doxygen

- [~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
- [Window](#) & [getWindow](#) ()
- void [setClearValue](#) ([VkClearColorValue](#) clearColorValue=[DEFAULT_CLEAR_COLOR](#), [VkClearDepthStencilValue](#) clearDepthValue=[DEFAULT_CLEAR_DEPTH](#))
- [VkCommandBuffer](#) [beginFrame](#) ()
- void [endFrame](#) ()
- void [beginSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer) const
- void [endSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer) const

Private Member Functions

- void [createCommandBuffers](#) ()
- void [freeCommandBuffers](#) ()
- void [recreateSwapChain](#) ()

Private Attributes

- [Window](#) & [m_window](#)
- [Device](#) & [m_device](#)
- std::unique_ptr< [SwapChain](#) > [m_swapChain](#)
- std::vector< [VkCommandBuffer](#) > [m_commandBuffers](#)
- std::array< [VkClearValue](#), 2 > [m_clearValues](#)
- uint32_t [m_currentImageIndex](#) {0}
- int [m_currentFrameIndex](#) {0}
- bool [m_isFrameStarted](#) {false}

7.31.1 Detailed Description

Definition at line 23 of file [Renderer.hpp](#).

7.31.2 Constructor & Destructor Documentation

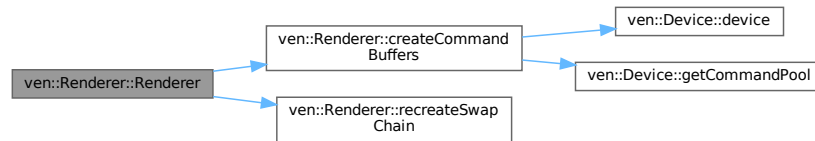
7.31.2.1 [Renderer\(\)](#) [1/2]

```
ven::Renderer::Renderer (  
    Window & window,  
    Device & device) [inline]
```

Definition at line 27 of file [Renderer.hpp](#).

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



7.31.2.2 ~Renderer()

```
ven::Renderer::~~Renderer () [inline]
```

Definition at line 28 of file [Renderer.hpp](#).

References [createCommandBuffers\(\)](#).

Here is the call graph for this function:



7.31.2.3 Renderer() [2/2]

```
ven::Renderer::Renderer (
    const Renderer & ) [delete]
```

7.31.3 Member Function Documentation

7.31.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

Definition at line 43 of file [renderer.cpp](#).

7.31.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

Definition at line 89 of file [renderer.cpp](#).

7.31.3.3 createCommandBuffers()

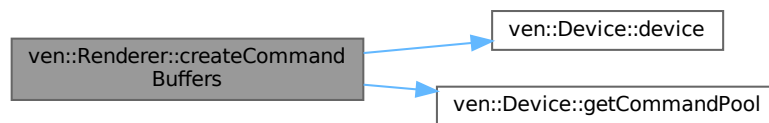
```
void ven::Renderer::createCommandBuffers () [private]
```

Definition at line 3 of file [renderer.cpp](#).

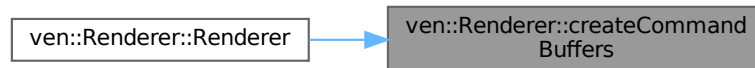
References [ven::Device::device\(\)](#), [ven::Device::getCommandPool\(\)](#), [m_commandBuffers](#), [m_device](#), and [ven::SwapChain::MAX_FRAMES_IN_FLIGHT](#).

Referenced by [Renderer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.31.3.4 endFrame()

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::SwapChain::MAX_FRAMES_IN_FLIGHT](#).

7.31.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (  
    VkCommandBuffer commandBuffer) const
```

Definition at line 119 of file [renderer.cpp](#).

7.31.3.6 freeCommandBuffers()

```
void ven::Renderer::freeCommandBuffers () [private]
```

Definition at line 17 of file [renderer.cpp](#).

Referenced by [~Renderer\(\)](#).

Here is the caller graph for this function:



7.31.3.7 getAspectRatio()

```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

Definition at line 34 of file [Renderer.hpp](#).

References [m_swapChain](#).

Referenced by [ven::ImGuiWindowManager::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.3.8 getClearColor()

```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

Definition at line 39 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::ImGuiWindowManager::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.3.9 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

Definition at line 36 of file [Renderer.hpp](#).

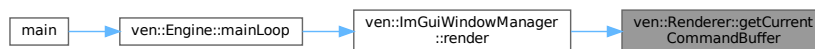
References [isFrameInProgress\(\)](#), [m_commandBuffers](#), and [m_currentFrameIndex](#).

Referenced by [ven::ImGuiWindowManager::render\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



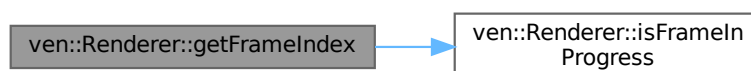
7.31.3.10 getFrameIndex()

```
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:



7.31.3.11 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:



7.31.3.12 getWindow()

`Window & ven::Renderer::getWindow () [inline], [nodiscard]`

Definition at line 46 of file [Renderer.hpp](#).

References [m_window](#).

Referenced by [ven::ImGuiWindowManager::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.3.13 isFrameInProgress()

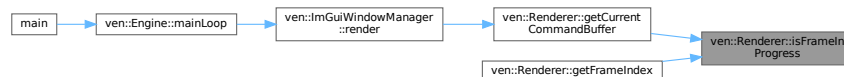
`bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]`

Definition at line 35 of file [Renderer.hpp](#).

References [m_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



7.31.3.14 operator=()

```
Renderer & ven::Renderer::operator= (
    const Renderer & ) [delete]
```

7.31.3.15 recreateSwapChain()

```
void ven::Renderer::recreateSwapChain () [private]
```

Definition at line 23 of file [renderer.cpp](#).

Referenced by [Renderer\(\)](#).

Here is the caller graph for this function:



7.31.3.16 setClearValue()

```
void ven::Renderer::setClearValue (
    VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

Definition at line 48 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::ImGuiWindowManager::rendererSection\(\)](#).

Here is the caller graph for this function:



7.31.4 Member Data Documentation

7.31.4.1 m_clearValues

```
std::array<VkClearColor, 2> ven::Renderer::m_clearValues [private]
```

Definition at line 64 of file [Renderer.hpp](#).

Referenced by [getClearColor\(\)](#), and [setClearColor\(\)](#).

7.31.4.2 m_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

Definition at line 63 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

7.31.4.3 m_currentFrameIndex

```
int ven::Renderer::m_currentFrameIndex {0} [private]
```

Definition at line 67 of file [Renderer.hpp](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

7.31.4.4 m_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

Definition at line 66 of file [Renderer.hpp](#).

7.31.4.5 m_device

```
Device& ven::Renderer::m_device [private]
```

Definition at line 61 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#).

7.31.4.6 m_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

Definition at line 68 of file [Renderer.hpp](#).

Referenced by [isFrameInProgress\(\)](#).

7.31.4.7 m_swapChain

`std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]`

Definition at line 62 of file [Renderer.hpp](#).

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

7.31.4.8 m_window

`Window& ven::Renderer::m_window [private]`

Definition at line 60 of file [Renderer.hpp](#).

Referenced by [getWindow\(\)](#).

The documentation for this class was generated from the following files:

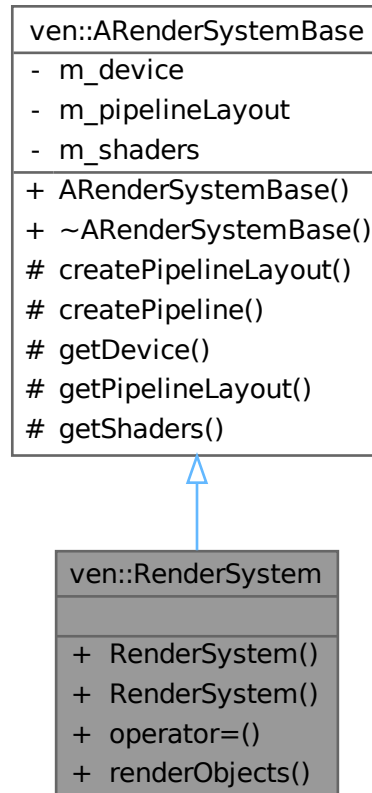
- [/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/renderer.cpp](#)

7.32 ven::RenderSystem Class Reference

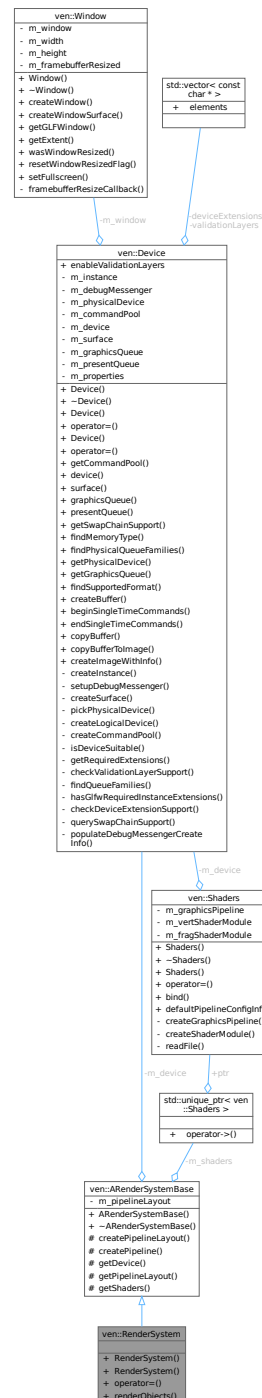
Class for render system.

`#include <RenderSystem.hpp>`

Inheritance diagram for `ven::RenderSystem`:



Collaboration diagram for `ven::RenderSystem`:



Public Member Functions

- `RenderSystem` (`Device` &device, const `VkRenderPass` renderPass, const `VkDescriptorSetLayout` globalSetLayout)
- `RenderSystem` (const `RenderSystem` &)=delete
- `RenderSystem` & operator= (const `RenderSystem` &)=delete
- void `renderObjects` (const `FrameInfo` &frameInfo) const

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- [~ARenderSystemBase](#) ()

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

7.32.1 Detailed Description

Class for render system.

Definition at line 24 of file [RenderSystem.hpp](#).

7.32.2 Constructor & Destructor Documentation

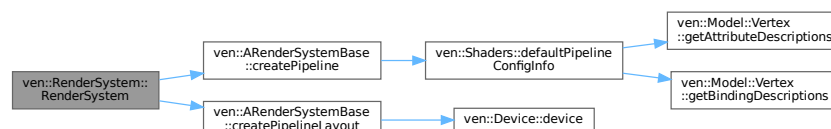
7.32.2.1 RenderSystem() [1/2]

```
ven::RenderSystem::RenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 28 of file [RenderSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.32.2.2 RenderSystem() [2/2]

```
ven::RenderSystem::RenderSystem (
    const RenderSystem & ) [delete]
```

7.32.3 Member Function Documentation

7.32.3.1 operator=()

```
RenderSystem & ven::RenderSystem::operator= (
    const RenderSystem & ) [delete]
```

7.32.3.2 renderObjects()

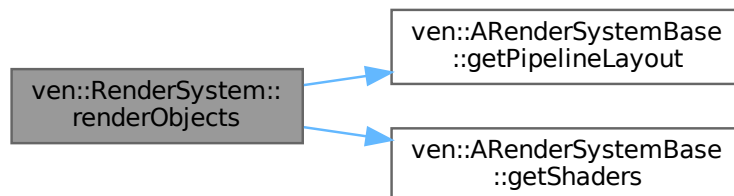
```
void ven::RenderSystem::renderObjects (
    const FrameInfo & frameInfo) const
```

Definition at line 5 of file [renderSystem.cpp](#).

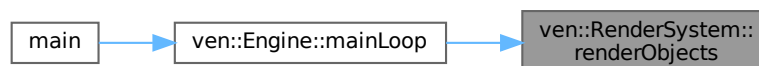
References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::ObjectPushConstantData::modelMatrix](#), and [ven::FrameInfo::objects](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

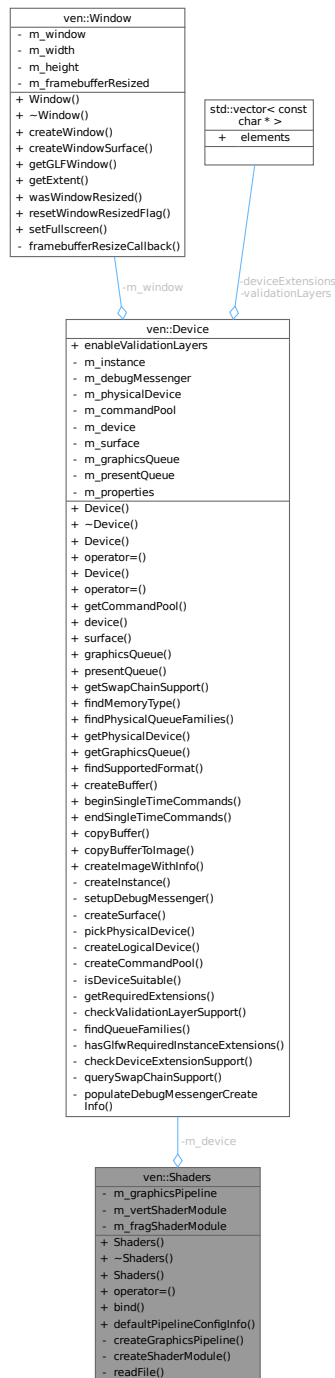
- [/home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp](#)

7.33 ven::Shaders Class Reference

Class for shaders.

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::Shaders:



Public Member Functions

- [Shaders](#) ([Device](#) &device, const std::string &vertFilepath, const std::string &fragFilepath, const [PipelineConfigInfo](#) &configInfo)
- [~Shaders](#) ()
- [Shaders](#) (const [Shaders](#) &)=delete
- [Shaders](#) & [operator=](#) (const [Shaders](#) &)=delete
- void [bind](#) (const VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void [defaultPipelineConfigInfo](#) ([PipelineConfigInfo](#) &configInfo)

Private Member Functions

- void [createGraphicsPipeline](#) (const std::string &vertFilepath, const std::string &fragFilepath, const [PipelineConfigInfo](#) &configInfo)
- void [createShaderModule](#) (const std::vector< char > &code, VkShaderModule *shaderModule) const

Static Private Member Functions

- static std::vector< char > [readFile](#) (const std::string &filename)

Private Attributes

- [Device](#) & [m_device](#)
- VkPipeline [m_graphicsPipeline](#) {nullptr}
- VkShaderModule [m_vertShaderModule](#) {nullptr}
- VkShaderModule [m_fragShaderModule](#) {nullptr}

7.33.1 Detailed Description

Class for shaders.

Definition at line 46 of file [Shaders.hpp](#).

7.33.2 Constructor & Destructor Documentation

7.33.2.1 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:



7.33.2.2 ~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:



7.33.2.3 Shaders() [2/2]

```
ven::Shaders::Shaders (
    const Shaders & ) [delete]
```

7.33.3 Member Function Documentation

7.33.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

Definition at line 57 of file [Shaders.hpp](#).

References [m_graphicsPipeline](#).

7.33.3.2 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:



7.33.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

Definition at line 100 of file [shaders.cpp](#).

7.33.3.4 defaultPipelineConfigInfo()

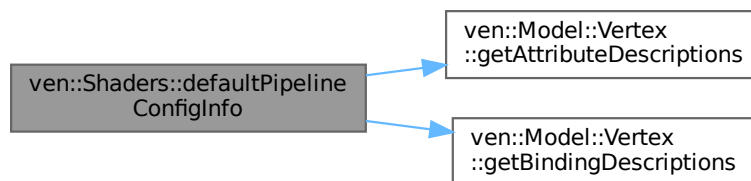
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 112 of file [shaders.cpp](#).

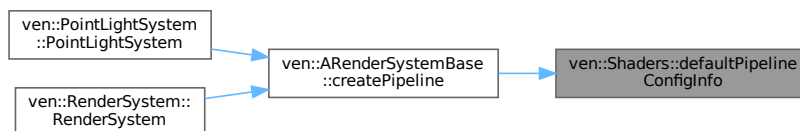
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::ARenderSystemBase::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.3.5 operator=()

```
Shaders & ven::Shaders::operator= (
    const Shaders & ) [delete]
```

7.33.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

Definition at line 13 of file [shaders.cpp](#).

7.33.4 Member Data Documentation

7.33.4.1 m_device

`Device& ven::Shaders::m_device [private]`

Definition at line 65 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.33.4.2 m_fragShaderModule

`VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]`

Definition at line 68 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.33.4.3 m_graphicsPipeline

`VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]`

Definition at line 66 of file [Shaders.hpp](#).

Referenced by [bind\(\)](#), and [~Shaders\(\)](#).

7.33.4.4 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:

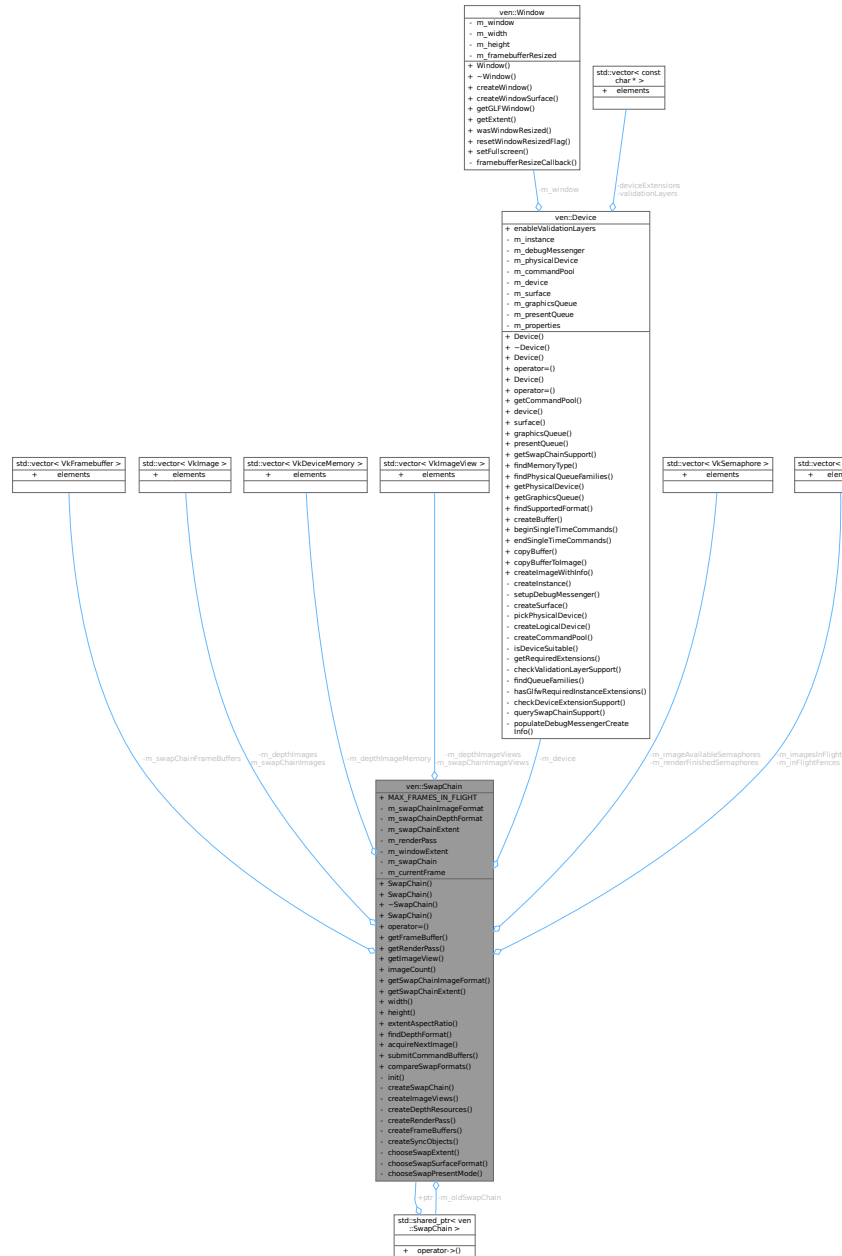
- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/shaders.cpp](#)

7.34 ven::SwapChain Class Reference

Class for swap chain.

```
#include <SwapChain.hpp>
```

Collaboration diagram for ven::SwapChain:



Public Member Functions

- [SwapChain](#) ([Device](#) &deviceRef, const VkExtent2D windowExtentRef)
- [SwapChain](#) ([Device](#) &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr< [SwapChain](#) > previous)

- [~SwapChain](#) ()
- [SwapChain](#) (const [SwapChain](#) &)=delete
- [SwapChain](#) & [operator=](#) (const [SwapChain](#) &)=delete
- [VkFramebuffer](#) [getFramebuffer](#) (const unsigned long index) const
- [VkRenderPass](#) [getRenderPass](#) () const
- [VkImageView](#) [getImageView](#) (const int index) const
- [size_t](#) [imageCount](#) () const
- [VkFormat](#) [getSwapChainImageFormat](#) () const
- [VkExtent2D](#) [getSwapChainExtent](#) () const
- [uint32_t](#) [width](#) () const
- [uint32_t](#) [height](#) () const
- [float](#) [extentAspectRatio](#) () const
- [VkFormat](#) [findDepthFormat](#) () const
- [VkResult](#) [acquireNextImage](#) (uint32_t *imageIndex) const
- [VkResult](#) [submitCommandBuffers](#) (const [VkCommandBuffer](#) *buffers, const uint32_t *imageIndex)
- [bool](#) [compareSwapFormats](#) (const [SwapChain](#) &swapChain) const

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](#) [m_swapChainImageFormat](#) {}
- [VkFormat](#) [m_swapChainDepthFormat](#) {}
- [VkExtent2D](#) [m_swapChainExtent](#) {}
- [std::vector](#)< [VkFramebuffer](#) > [m_swapChainFrameBuffers](#)
- [VkRenderPass](#) [m_renderPass](#) {}
- [std::vector](#)< [VkImage](#) > [m_depthImages](#)
- [std::vector](#)< [VkDeviceMemory](#) > [m_depthImageMemory](#)
- [std::vector](#)< [VkImageView](#) > [m_depthImageViews](#)
- [std::vector](#)< [VkImage](#) > [m_swapChainImages](#)
- [std::vector](#)< [VkImageView](#) > [m_swapChainImageViews](#)

- [Device](#) & [m_device](#)
- [VkExtent2D](#) [m_windowExtent](#)
- [VkSwapchainKHR](#) [m_swapChain](#) {}
- [std::shared_ptr< SwapChain >](#) [m_oldSwapChain](#)
- [std::vector< VkSemaphore >](#) [m_imageAvailableSemaphores](#)
- [std::vector< VkSemaphore >](#) [m_renderFinishedSemaphores](#)
- [std::vector< VkFence >](#) [m_inFlightFences](#)
- [std::vector< VkFence >](#) [m_imagesInFlight](#)
- [size_t](#) [m_currentFrame](#) {0}

7.34.1 Detailed Description

Class for swap chain.

Definition at line 21 of file [SwapChain.hpp](#).

7.34.2 Constructor & Destructor Documentation

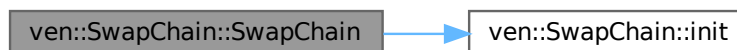
7.34.2.1 SwapChain() [1/3]

```
ven::SwapChain::SwapChain (  
    Device & deviceRef,  
    const VkExtent2D windowExtentRef) [inline]
```

Definition at line 27 of file [SwapChain.hpp](#).

References [init\(\)](#).

Here is the call graph for this function:



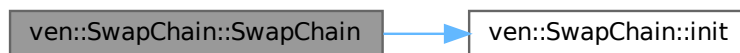
7.34.2.2 SwapChain() [2/3]

```
ven::SwapChain::SwapChain (
    Device & deviceRef,
    const VkExtent2D windowExtentRef,
    std::shared_ptr< SwapChain > previous) [inline]
```

Definition at line 28 of file [SwapChain.hpp](#).

References [init\(\)](#), and [m_oldSwapChain](#).

Here is the call graph for this function:



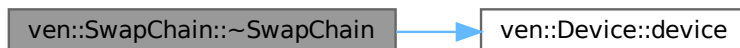
7.34.2.3 ~SwapChain()

```
ven::SwapChain::~~SwapChain ()
```

Definition at line 7 of file [swapChain.cpp](#).

References [ven::Device::device\(\)](#), [m_depthImageMemory](#), [m_depthImages](#), [m_depthImageViews](#), [m_device](#), [m_imageAvailableSemaphores](#), [m_inFlightFences](#), [m_renderFinishedSemaphores](#), [m_renderPass](#), [m_swapChain](#), [m_swapChainFrameBuffers](#), [m_swapChainImageViews](#), and [MAX_FRAMES_IN_FLIGHT](#).

Here is the call graph for this function:



7.34.2.4 SwapChain() [3/3]

```
ven::SwapChain::SwapChain (
    const SwapChain & ) [delete]
```

7.34.3 Member Function Documentation

7.34.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

7.34.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

7.34.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

7.34.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

Definition at line 331 of file [swapChain.cpp](#).

7.34.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (
    const SwapChain & swapChain) const [inline], [nodiscard]
```

Definition at line 49 of file [SwapChain.hpp](#).

References [m_swapChainDepthFormat](#), and [m_swapChainImageFormat](#).

7.34.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

7.34.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

7.34.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

7.34.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

7.34.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

Definition at line 103 of file [swapChain.cpp](#).

7.34.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

7.34.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

Definition at line 43 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.34.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

7.34.3.14 getFramebuffer()

```
VkFramebuffer ven::SwapChain::getFramebuffer (
    const unsigned long index) const [inline], [nodiscard]
```

Definition at line 34 of file [SwapChain.hpp](#).

References [m_swapChainFrameBuffers](#).

7.34.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (
    const int index) const [inline], [nodiscard]
```

Definition at line 36 of file [SwapChain.hpp](#).

References [m_swapChainImageViews](#).

7.34.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

Definition at line 35 of file [SwapChain.hpp](#).

References [m_renderPass](#).

7.34.3.17 getSwapChainExtent()

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

Definition at line 39 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.34.3.18 getSwapChainImageFormat()

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

Definition at line 38 of file [SwapChain.hpp](#).

References [m_swapChainImageFormat](#).

7.34.3.19 height()

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

Definition at line 41 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.34.3.20 imageCount()

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

Definition at line 37 of file [SwapChain.hpp](#).

References [m_swapChainImages](#).

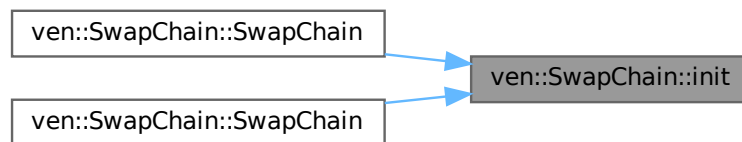
7.34.3.21 init()

```
void ven::SwapChain::init () [private]
```

Definition at line 39 of file [swapChain.cpp](#).

Referenced by [SwapChain\(\)](#), and [SwapChain\(\)](#).

Here is the caller graph for this function:



7.34.3.22 operator=()

```
SwapChain & ven::SwapChain::operator= (
    const SwapChain & ) [delete]
```

7.34.3.23 submitCommandBuffers()

```
VkResult ven::SwapChain::submitCommandBuffers (
    const VkCommandBuffer * buffers,
    const uint32_t * imageIndex)
```

Definition at line 56 of file [swapChain.cpp](#).

7.34.3.24 width()

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

Definition at line 40 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.34.4 Member Data Documentation

7.34.4.1 m_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0} [private]
```

Definition at line 88 of file [SwapChain.hpp](#).

7.34.4.2 m_depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::m_depthImageMemory [private]
```

Definition at line 73 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.3 m_depthImages

```
std::vector<VkImage> ven::SwapChain::m_depthImages [private]
```

Definition at line 72 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.4 m_depthImageViews

```
std::vector<VkImageView> ven::SwapChain::m_depthImageViews [private]
```

Definition at line 74 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.5 m_device

```
Device& ven::SwapChain::m_device [private]
```

Definition at line 78 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.6 m_imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_imageAvailableSemaphores [private]
```

Definition at line 84 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.7 m_imagesInFlight

```
std::vector<VkFence> ven::SwapChain::m_imagesInFlight [private]
```

Definition at line 87 of file [SwapChain.hpp](#).

7.34.4.8 m_inFlightFences

```
std::vector<VkFence> ven::SwapChain::m_inFlightFences [private]
```

Definition at line 86 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.9 m_oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::m_oldSwapChain [private]
```

Definition at line 82 of file [SwapChain.hpp](#).

Referenced by [SwapChain\(\)](#).

7.34.4.10 m_renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_renderFinishedSemaphores [private]
```

Definition at line 85 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.11 m_renderPass

```
VkRenderPass ven::SwapChain::m_renderPass {} [private]
```

Definition at line 70 of file [SwapChain.hpp](#).

Referenced by [getRenderPass\(\)](#), and [~SwapChain\(\)](#).

7.34.4.12 m_swapChain

```
VkSwapchainKHR ven::SwapChain::m_swapChain {} [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.34.4.13 m_swapChainDepthFormat

```
VkFormat ven::SwapChain::m_swapChainDepthFormat {} [private]
```

Definition at line 66 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#).

7.34.4.14 m_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

Definition at line 67 of file [SwapChain.hpp](#).

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

7.34.4.15 m_swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::m_swapChainFrameBuffers [private]
```

Definition at line 69 of file [SwapChain.hpp](#).

Referenced by [getFramebuffer\(\)](#), and [~SwapChain\(\)](#).

7.34.4.16 m_swapChainImageFormat

```
VkFormat ven::SwapChain::m_swapChainImageFormat {} [private]
```

Definition at line 65 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

7.34.4.17 m_swapChainImages

```
std::vector<VkImage> ven::SwapChain::m_swapChainImages [private]
```

Definition at line 75 of file [SwapChain.hpp](#).

Referenced by [imageCount\(\)](#).

7.34.4.18 m_swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::m_swapChainImageViews [private]
```

Definition at line 76 of file [SwapChain.hpp](#).

Referenced by [getImageView\(\)](#), and [~SwapChain\(\)](#).

7.34.4.19 m_windowExtent

```
VkExtent2D ven::SwapChain::m_windowExtent [private]
```

Definition at line 79 of file [SwapChain.hpp](#).

7.34.4.20 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\(\)](#).

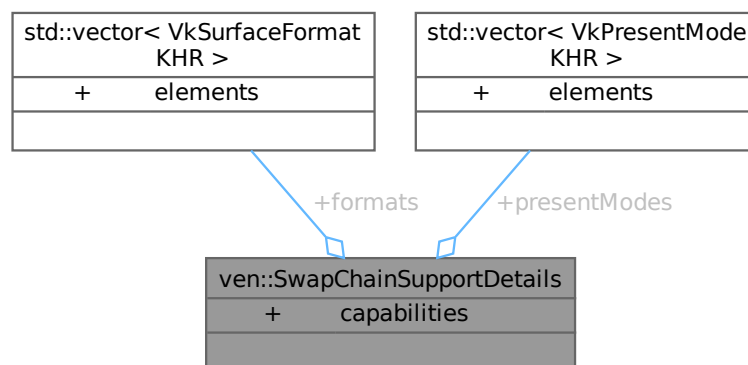
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/swapChain.cpp](#)

7.35 ven::SwapChainSupportDetails Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::SwapChainSupportDetails:



Public Attributes

- [VkSurfaceCapabilitiesKHR](#) [capabilities](#)
- [std::vector< VkSurfaceFormatKHR >](#) [formats](#)
- [std::vector< VkPresentModeKHR >](#) [presentModes](#)

7.35.1 Detailed Description

Definition at line 15 of file [Device.hpp](#).

7.35.2 Member Data Documentation

7.35.2.1 capabilities

```
VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities
```

Definition at line 16 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.35.2.2 formats

```
std::vector<VkSurfaceFormatKHR> ven::SwapChainSupportDetails::formats
```

Definition at line 17 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.35.2.3 presentModes

```
std::vector<VkPresentModeKHR> ven::SwapChainSupportDetails::presentModes
```

Definition at line 18 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

7.36 myLib::Time Class Reference

Class used for time management.

```
#include <Time.hpp>
```

Collaboration diagram for myLib::Time:



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}

7.36.1 Detailed Description

Class used for time management.

Definition at line 18 of file [Time.hpp](#).

7.36.2 Constructor & Destructor Documentation

7.36.2.1 Time()

```
myLib::Time::Time (  
    const double seconds) [inline], [explicit]
```

Construct a new [Time](#) object.

Definition at line 25 of file [Time.hpp](#).

7.36.3 Member Function Documentation

7.36.3.1 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](#).

7.36.3.2 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](#).

7.36.3.3 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](#).

7.36.4 Member Data Documentation

7.36.4.1 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](#)

7.37 ven::Transform3DComponent Class Reference

```
#include <Transform3DComponent.hpp>
```

Collaboration diagram for ven::Transform3DComponent:

ven::Transform3DComponent	
+	translation
+	scale
+	rotation
+	mat4()
+	normalMatrix()

Public Member Functions

- glm::mat4 [mat4](#) () const
- glm::mat3 [normalMatrix](#) () const

Public Attributes

- glm::vec3 [translation](#) {}
- glm::vec3 [scale](#) {1.F, 1.F, 1.F}
- glm::vec3 [rotation](#) {}

7.37.1 Detailed Description

Definition at line [13](#) of file [Transform3DComponent.hpp](#).

7.37.2 Member Function Documentation

7.37.2.1 mat4()

```
glm::mat4 ven::Transform3DComponent::mat4 () const [nodiscard]
```

Definition at line [3](#) of file [transform3DComponent.cpp](#).

References [rotation](#), [scale](#), and [translation](#).

7.37.2.2 normalMatrix()

```
glm::mat3 ven::Transform3DComponent::normalMatrix () const [nodiscard]
```

Definition at line 38 of file [transform3DComponent.cpp](#).

7.37.3 Member Data Documentation

7.37.3.1 rotation

```
glm::vec3 ven::Transform3DComponent::rotation {}
```

Definition at line 19 of file [Transform3DComponent.hpp](#).

Referenced by [ven::ImGuiWindowManager::cameraSection\(\)](#), [ven::Engine::mainLoop\(\)](#), [mat4\(\)](#), and [ven::KeyboardController::moveInPlaneXZ\(\)](#).

7.37.3.2 scale

```
glm::vec3 ven::Transform3DComponent::scale {1.F, 1.F, 1.F}
```

Definition at line 18 of file [Transform3DComponent.hpp](#).

Referenced by [ven::Light::createLight\(\)](#), [ven::Engine::loadObjects\(\)](#), and [mat4\(\)](#).

7.37.3.3 translation

```
glm::vec3 ven::Transform3DComponent::translation {}
```

Definition at line 17 of file [Transform3DComponent.hpp](#).

Referenced by [ven::ImGuiWindowManager::cameraSection\(\)](#), [ven::Engine::loadObjects\(\)](#), [ven::Engine::mainLoop\(\)](#), [mat4\(\)](#), and [ven::KeyboardController::moveInPlaneXZ\(\)](#).

The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp](#)

7.38 ven::Model::Vertex Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model::Vertex:

ven::Model::Vertex
+ position
+ color
+ normal
+ uv
+ operator==()
+ getBindingDescriptions()
+ getAttributeDescriptions()

Public Member Functions

- bool [operator==](#) (const [Vertex](#) &other) const

Static Public Member Functions

- static std::vector< [VkVertexInputBindingDescription](#) > [getBindingDescriptions](#) ()
- static std::vector< [VkVertexInputAttributeDescription](#) > [getAttributeDescriptions](#) ()

Public Attributes

- glm::vec3 [position](#) {}
- glm::vec3 [color](#) {}
- glm::vec3 [normal](#) {}
- glm::vec2 [uv](#) {}

7.38.1 Detailed Description

Definition at line 25 of file [Model.hpp](#).

7.38.2 Member Function Documentation

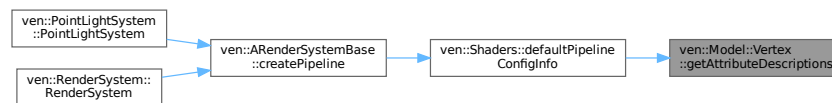
7.38.2.1 getAttributeDescriptions()

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions
() [static]
```

Definition at line 105 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



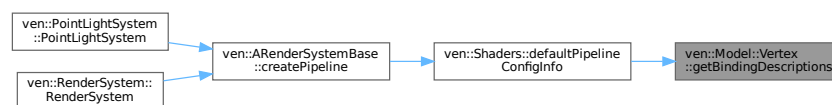
7.38.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()
[static]
```

Definition at line 96 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



7.38.2.3 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](#).

7.38.3 Member Data Documentation

7.38.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 27 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.38.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 28 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.38.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 26 of file [Model.hpp](#).

Referenced by [ven::Model::Builder::loadModel\(\)](#), and [operator==\(\)](#).

7.38.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 29 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

The documentation for this struct was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.39 ven::Window Class Reference

Class for window.

```
#include <Window.hpp>
```

Collaboration diagram for ven::Window:

ven::Window
<ul style="list-style-type: none"> - m_window - m_width - m_height - m_framebufferResized
<ul style="list-style-type: none"> + Window() + ~Window() + createWindow() + createWindowSurface() + getGLFWWindow() + getExtent() + wasWindowResized() + resetWindowResizedFlag() + setFullscreen() - framebufferResizeCallback()

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](#) ()
- void [setFullscreen](#) (bool fullscreen, uint32_t width, uint32_t height)

Static Private Member Functions

- static void [framebufferResizeCallback](#) (GLFWwindow *window, int width, int height)

Private Attributes

- GLFWwindow * [m_window](#) {nullptr}
- uint32_t [m_width](#)
- uint32_t [m_height](#)
- bool [m_framebufferResized](#) = false

7.39.1 Detailed Description

Class for window.

Definition at line 26 of file [Window.hpp](#).

7.39.2 Constructor & Destructor Documentation

7.39.2.1 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](#).

7.39.2.2 ~Window()

```
ven::Window::~~Window () [inline]
```

Definition at line 31 of file [Window.hpp](#).

References [m_window](#).

7.39.3 Member Function Documentation

7.39.3.1 createWindow()

```
GLFWwindow * ven::Window::createWindow (
    uint32_t width,
    uint32_t height,
    const std::string & title) [nodiscard]
```

Definition at line 5 of file [window.cpp](#).

References [framebufferResizeCallback\(\)](#).

Here is the call graph for this function:



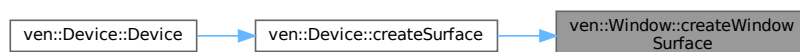
7.39.3.2 createWindowSurface()

```
void ven::Window::createWindowSurface (
    VkInstance instance,
    VkSurfaceKHR * surface) const
```

Definition at line 24 of file [window.cpp](#).

Referenced by [ven::Device::createSurface\(\)](#).

Here is the caller graph for this function:



7.39.3.3 framebufferResizeCallback()

```
void ven::Window::framebufferResizeCallback (
    GLFWwindow * window,
    int width,
    int height) [static], [private]
```

Definition at line 31 of file [window.cpp](#).

References [m_framebufferResized](#).

Referenced by [createWindow\(\)](#).

Here is the caller graph for this function:



7.39.3.4 `getExtent()`

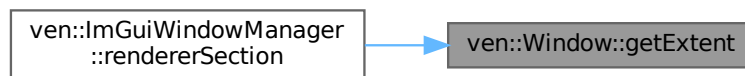
```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

Definition at line 38 of file [Window.hpp](#).

References [m_height](#), and [m_width](#).

Referenced by [ven::ImGuiWindowManager::renderSection\(\)](#).

Here is the caller graph for this function:



7.39.3.5 `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:



7.39.3.6 `resetWindowResizedFlag()`

```
void ven::Window::resetWindowResizedFlag () [inline]
```

Definition at line 40 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.39.3.7 setFullscreen()

```
void ven::Window::setFullscreen (
    bool fullscreen,
    uint32_t width,
    uint32_t height)
```

Definition at line 39 of file [window.cpp](#).

Referenced by [ven::ImGuiWindowManager::renderSection\(\)](#).

Here is the caller graph for this function:



7.39.3.8 wasWindowResized()

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

Definition at line 39 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.39.4 Member Data Documentation

7.39.4.1 m_framebufferResized

```
bool ven::Window::m_framebufferResized = false [private]
```

Definition at line 52 of file [Window.hpp](#).

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).

7.39.4.2 m_height

```
uint32_t ven::Window::m_height [private]
```

Definition at line 50 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.39.4.3 m_width

```
uint32_t ven::Window::m_width [private]
```

Definition at line 49 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.39.4.4 m_window

```
GLFWwindow* ven::Window::m_window {nullptr} [private]
```

Definition at line 48 of file [Window.hpp](#).

Referenced by [getGLFWWindow\(\)](#), and [~Window\(\)](#).

The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/window.cpp](#)

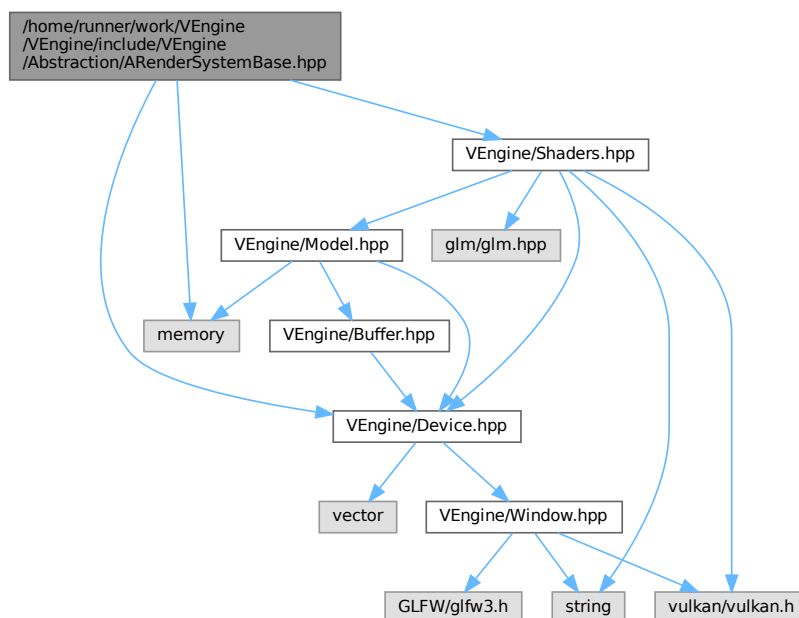
Chapter 8

File Documentation

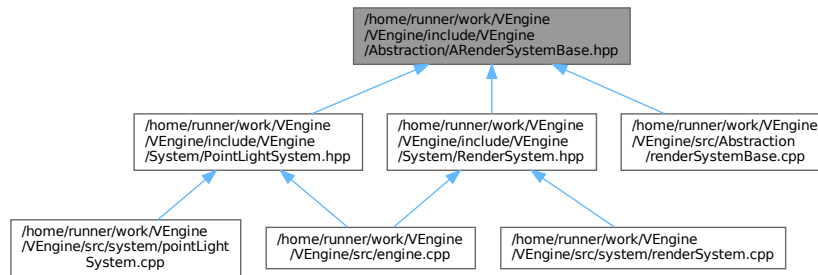
8.1 /home/runner/work/VEngine/VEngine/include/VEngine/Abstraction/ARenderSystemBase.hpp File Reference

This file contains the ARenderSystemBase class.

```
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
Include dependency graph for ARenderSystemBase.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::ARenderSystemBase](#)
Abstract class for render system base.

Namespaces

- namespace [ven](#)

8.1.1 Detailed Description

This file contains the ARenderSystemBase class.

Definition in file [ARenderSystemBase.hpp](#).

8.2 ARenderSystemBase.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file ARenderSystemBase.hpp
00003 /// @brief This file contains the ARenderSystemBase class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Shader.hpp"
00013
00014
00015 namespace ven {
00016
00017     ///
00018     /// @class ARenderSystemBase
00019     /// @brief Abstract class for render system base
00020     /// @namespace ven
00021     ///
00022     class ARenderSystemBase {
00023
00024     public:
00025         explicit ARenderSystemBase(Device& device)
00026             : m_device{device} {}
00027
00028     private:
00029         Device m_device;
00030     };
00031
00032 }

```

```

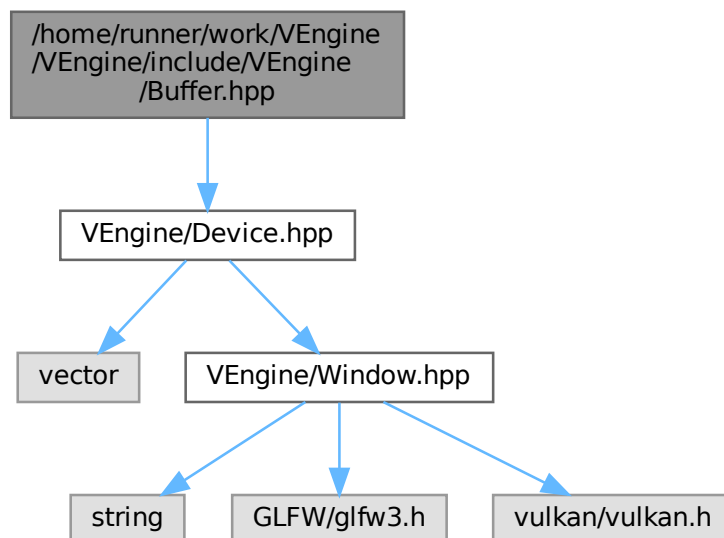
00027
00028     ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout,
00029 nullptr); }
00029
00030     protected:
00031
00032         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout, uint32_t
00033 pushConstantSize);
00033         void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
00034 std::string &shadersFragPath, bool isLight);
00035
00036         [[nodiscard]] Device& getDevice() const { return m_device; }
00037         [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
00038         [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }
00039
00040     private:
00041         Device &m_device;
00042         VkPipelineLayout m_pipelineLayout{nullptr};
00043         std::unique_ptr<Shaders> m_shaders;
00044
00045 }; // class RenderSystemBase
00046
00047 } // namespace ven

```

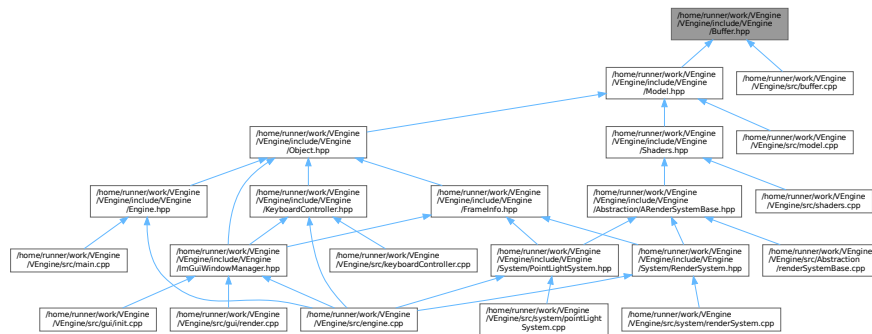
8.3 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

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](#)

8.3.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

8.4 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_alignmentSize; }
00131         [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00132         [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
m_memoryPropertyFlags; }
00133         [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }
00134
00135     private:
00136         ///
00137         /// Returns the minimum instance size required to be compatible with devices
minOffsetAlignment
00138         ///
00139         /// @param instanceSize The size of an instance
00140         /// @param minOffsetAlignment The minimum required alignment, in bytes, for the offset
member (eg
00141         /// minUniformBufferOffsetAlignment)
00142         ///
00143         /// @return VkResult of the buffer mapping call
00144         ///
00145         static VkDeviceSize getAlignment(VkDeviceSize instanceSize, VkDeviceSize
minOffsetAlignment);
00146
00147         Device& m_device;
00148         void* m_mapped = nullptr;
00149         VkBuffer m_buffer = VK_NULL_HANDLE;
00150         VkDeviceMemory m_memory = VK_NULL_HANDLE;
00151
00152         VkDeviceSize m_bufferSize;
00153         VkDeviceSize m_instanceSize;
00154         uint32_t m_instanceCount;
00155         VkDeviceSize m_alignmentSize;
00156         VkBufferUsageFlags m_usageFlags;
00157         VkMemoryPropertyFlags m_memoryPropertyFlags;
00158
00159     }; // class Buffer
00160
00161 } // namespace ven

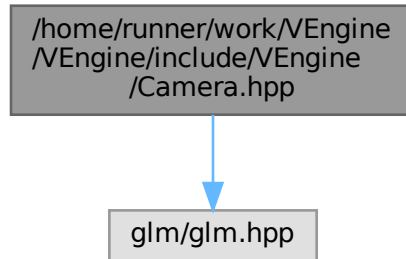
```

8.5 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

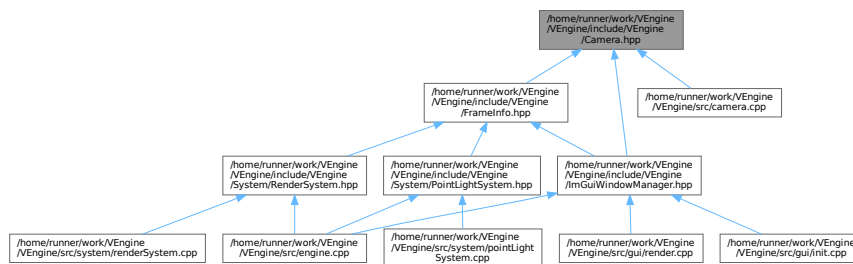
This file contains the Camera class.

```
#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

8.5.1 Detailed Description

This file contains the Camera class.

This file contains the KeyboardController class.

Definition in file [Camera.hpp](#).

8.6 Camera.hpp

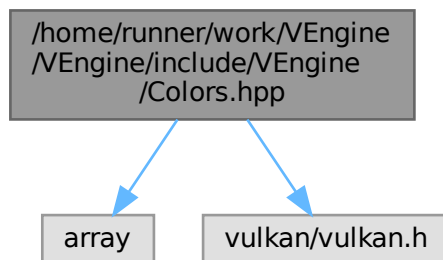
[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Camera.hpp
00003 /// @brief This file contains the Camera class
00004 /// @namespace ven
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     /// @class Camera
00022     /// @brief Class for camera
00023     /// @namespace ven
00024     ///
00025     class Camera {
00026
00027     public:
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(const glm::vec3 position, const glm::vec3 target, const 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(const float fov) { m_fov = fov; }
00035         void setNear(const float near) { m_near = near; }
00036         void setFar(const 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
00045     private:
00046
00047         float m_fov{DEFAULT_FOV};
00048         float m_near{DEFAULT_NEAR};
00049         float m_far{DEFAULT_FAR};
00050         glm::mat4 m_projectionMatrix{1.F};
00051         glm::mat4 m_viewMatrix{1.F};
00052         glm::mat4 m_inverseViewMatrix{1.F};
00053
00054     }; // class Camera
00055
00056 } // namespace ven
```

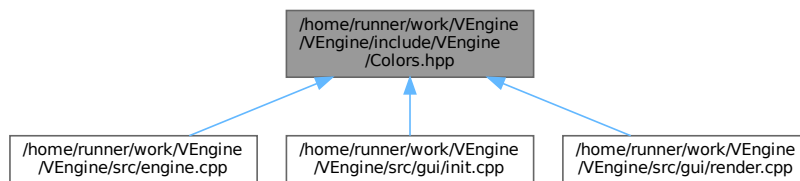

8.7 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference

```
#include <array>
#include <vulkan/vulkan.h>
```

Include dependency graph for Colors.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Colors](#)
Class for colors.

Namespaces

- namespace [ven](#)

8.8 Colors.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Colors.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <array>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 namespace ven {
00014
00015     ///
00016     /// @class Colors
00017     /// @brief Class for colors
00018     /// @namespace ven
00019     ///
00020     class Colors {
00021
00022     public:
00023
00024         static constexpr float COLOR_MAX = 255.0F;
00025
00026         static constexpr glm::vec3 WHITE = glm::vec3(COLOR_MAX, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00027         static constexpr glm::vec3 BLACK = glm::vec3(0.0F);
00028         static constexpr glm::vec3 RED = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
00029         static constexpr glm::vec3 GREEN = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00030         static constexpr glm::vec3 BLUE = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
00031         static constexpr glm::vec3 YELLOW = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
00032         static constexpr glm::vec3 CYAN = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00033         static constexpr glm::vec3 MAGENTA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00034         static constexpr glm::vec3 SILVER = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
00035         static constexpr glm::vec3 GRAY = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
00036         static constexpr glm::vec3 MAROON = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
00037         static constexpr glm::vec3 OLIVE = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
00038         static constexpr glm::vec3 LIME = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00039         static constexpr glm::vec3 AQUA = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00040         static constexpr glm::vec3 TEAL = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
00041         static constexpr glm::vec3 NAVY = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
00042         static constexpr glm::vec3 FUCHSIA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00043         static constexpr glm::vec3 NIGHT_BLUE = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
00044         static constexpr glm::vec3 SKY_BLUE = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
00045         static constexpr glm::vec3 SUNSET = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
00046
00047         static constexpr VkClearColorValue WHITE_V = {{1.0F, 1.0F, 1.0F, 1.0F}};
00048         static constexpr VkClearColorValue BLACK_V = {{0.0F, 0.0F, 0.0F, 1.0F}};
00049         static constexpr VkClearColorValue RED_V = {{1.0F, 0.0F, 0.0F, 1.0F}};
00050         static constexpr VkClearColorValue GREEN_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
00051         static constexpr VkClearColorValue BLUE_V = {{0.0F, 0.0F, 1.0F, 1.0F}};
00052         static constexpr VkClearColorValue YELLOW_V = {{1.0F, 1.0F, 0.0F, 1.0F}};
00053         static constexpr VkClearColorValue CYAN_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
00054         static constexpr VkClearColorValue MAGENTA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
00055         static constexpr VkClearColorValue SILVER_V = {{0.75F, 0.75F, 0.75F, 1.0F}};
00056         static constexpr VkClearColorValue GRAY_V = {{0.5F, 0.5F, 0.5F, 1.0F}};
00057         static constexpr VkClearColorValue MAROON_V = {{0.5F, 0.0F, 0.0F, 1.0F}};
00058         static constexpr VkClearColorValue OLIVE_V = {{0.5F, 0.5F, 0.0F, 1.0F}};
00059         static constexpr VkClearColorValue LIME_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
00060         static constexpr VkClearColorValue AQUA_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
00061         static constexpr VkClearColorValue TEAL_V = {{0.0F, 0.5F, 0.5F, 1.0F}};
00062         static constexpr VkClearColorValue NAVY_V = {{0.0F, 0.0F, 0.5F, 1.0F}};
00063         static constexpr VkClearColorValue FUCHSIA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
00064         static constexpr VkClearColorValue NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}};
00065         static constexpr VkClearColorValue SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}};
00066         static constexpr VkClearColorValue SUNSET_V = {{1.0F, 0.5F, 0.0F, 1.0F}};
00067         static constexpr VkClearColorValue NIGHT_MODE_V = {{0.0F, 0.0F, 0.0F, 1.0F}};
00068
00069         static constexpr std::array<std::pair<const char*, glm::vec3>, 20> COLORS = {{
00070             {"White", Colors::WHITE},
00071             {"Black", Colors::BLACK},
00072             {"Red", Colors::RED},
00073             {"Green", Colors::GREEN},
00074             {"Blue", Colors::BLUE},
00075             {"Yellow", Colors::YELLOW},
00076             {"Cyan", Colors::CYAN},
00077             {"Magenta", Colors::MAGENTA},
00078             {"Silver", Colors::SILVER},
00079             {"Gray", Colors::GRAY},
00080             {"Maroon", Colors::MAROON},
00081             {"Olive", Colors::OLIVE},
00082             {"Lime", Colors::LIME},
```

```

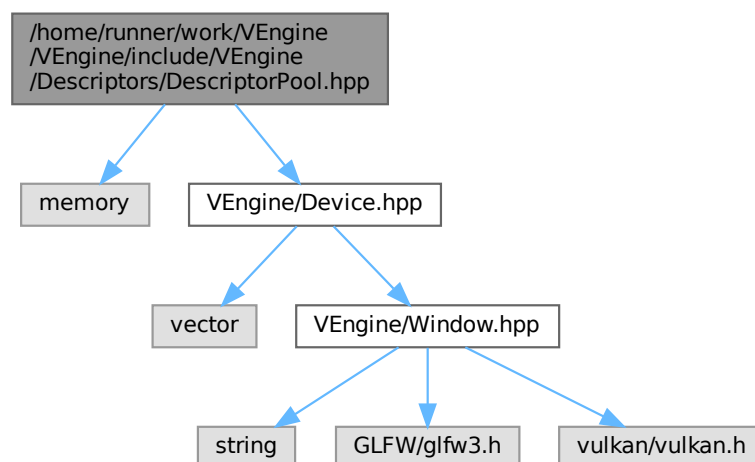
00083         {"Aqua", Colors::AQUA},
00084         {"Teal", Colors::TEAL},
00085         {"Navy", Colors::NAVY},
00086         {"Fuchsia", Colors::FUCHSIA},
00087         {"Night Blue", ven::Colors::NIGHT_BLUE},
00088         {"Sky Blue", Colors::SKY_BLUE},
00089         {"Sunset", Colors::SUNSET}
00090     };
00091
00092     static constexpr std::array<std::pair<const char*, VkClearColorValue>, 20> CLEAR_COLORS = {{
00093         {"White", Colors::WHITE_V},
00094         {"Black", Colors::BLACK_V},
00095         {"Red", Colors::RED_V},
00096         {"Green", Colors::GREEN_V},
00097         {"Blue", Colors::BLUE_V},
00098         {"Yellow", Colors::YELLOW_V},
00099         {"Cyan", Colors::CYAN_V},
00100         {"Magenta", Colors::MAGENTA_V},
00101         {"Silver", Colors::SILVER_V},
00102         {"Gray", Colors::GRAY_V},
00103         {"Maroon", Colors::MAROON_V},
00104         {"Olive", Colors::OLIVE_V},
00105         {"Lime", Colors::LIME_V},
00106         {"Aqua", Colors::AQUA_V},
00107         {"Teal", Colors::TEAL_V},
00108         {"Navy", Colors::NAVY_V},
00109         {"Fuchsia", Colors::FUCHSIA_V},
00110         {"Night Blue", Colors::NIGHT_BLUE_V},
00111         {"Sky Blue", Colors::SKY_BLUE_V},
00112         {"Sunset", Colors::SUNSET_V}
00113     }};
00114
00115 }; // class Colors
00116
00117 } // namespace ven

```

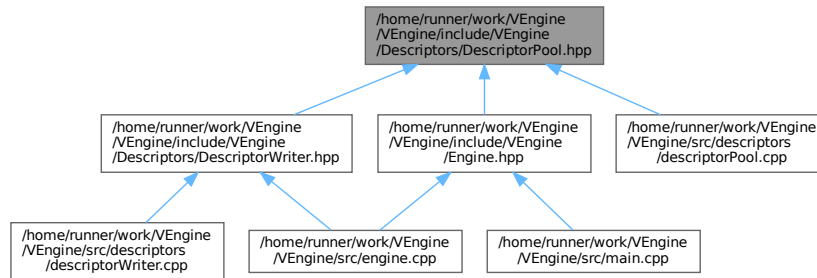
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference ↩

This file contains the DescriptorPool class.

```
#include <memory>
#include "VEngine/Device.hpp"
Include dependency graph for DescriptorPool.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorPool](#)
Class for descriptor pool.
- class [ven::DescriptorPool::Builder](#)

Namespaces

- namespace [ven](#)

8.9.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file [DescriptorPool.hpp](#).

8.10 DescriptorPool.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file DescriptorPool.hpp
00003 /// @brief This file contains the DescriptorPool class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012
00013 namespace ven {
00014
00015     ///
00016     /// @class DescriptorPool
00017     /// @brief Class for descriptor pool
00018     /// @namespace ven
00019     ///
00020     class DescriptorPool {
00021
00022     public:
00023
00024         class Builder {

```

```

00025
00026         public:
00027
00028             explicit Builder(Device &device) : m_device{device} {}
00029
00030             Builder &addPoolSize(VkDescriptorType descriptorType, uint32_t count);
00031             Builder &setPoolFlags(VkDescriptorPoolCreateFlags flags);
00032             Builder &setMaxSets(uint32_t count);
00033             [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00034
00035         private:
00036
00037             Device &m_device;
00038             std::vector<VkDescriptorPoolSize> m_poolSizes;
00039             uint32_t m_maxSets = 1000;
00040             VkDescriptorPoolCreateFlags m_poolFlags = 0;
00041
00042     }; // class Builder
00043
00044     DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
const std::vector<VkDescriptorPoolSize> &poolSizes);
00045     ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
}
00046     DescriptorPool(const DescriptorPool &) = delete;
00047     DescriptorPool &operator=(const DescriptorPool &) = delete;
00048
00049     bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
&descriptor) const;
00050     void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
descriptors.data()); }
00051     void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00052
00053     [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }
00054
00055     private:
00056
00057         Device &m_device;
00058         VkDescriptorPool m_descriptorPool;
00059         friend class DescriptorWriter;
00060
00061     }; // class DescriptorPool
00062
00063 } // namespace ven

```

8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference ↩

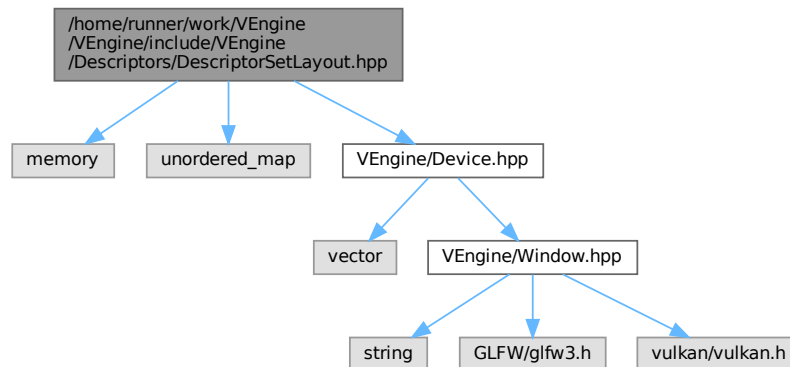
This file contains the DescriptorSetLayout class.

```

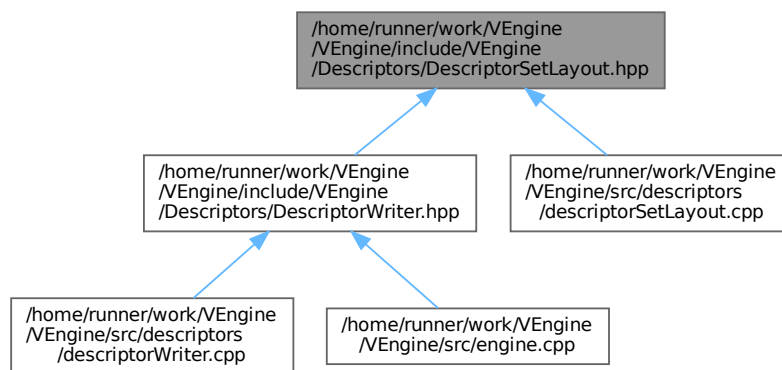
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"

```

Include dependency graph for DescriptorSetLayout.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorSetLayout](#)
Class for descriptor set layout.
- class [ven::DescriptorSetLayout::Builder](#)

Namespaces

- namespace [ven](#)

8.11.1 Detailed Description

This file contains the `DescriptorSetLayout` class.

Definition in file [DescriptorSetLayout.hpp](#).

8.12 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

```

8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference

This file contains the DescriptorsWriter class.

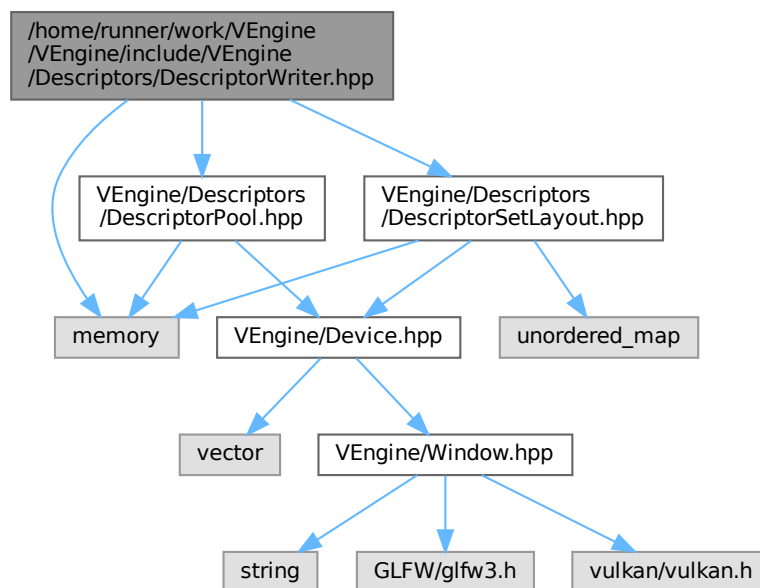
```

#include <memory>
#include "VEngine/Descriptors/DescriptorPool.hpp"

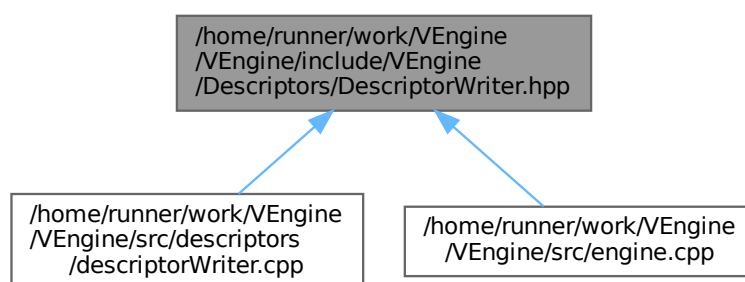
```

```
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"
```

Include dependency graph for DescriptorWriter.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorWriter](#)
Class for descriptor writer.

Namespaces

- namespace [ven](#)

8.13.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file [DescriptorWriter.hpp](#).

8.14 DescriptorWriter.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file DescriptorWriter.hpp
00003 /// @brief This file contains the DescriptorsWriter class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Descriptors/DescriptorPool.hpp"
00012 #include "VEngine/Descriptors/DescriptorSetLayout.hpp"
00013
00014 namespace ven {
00015
00016     ///
00017     /// @class DescriptorWriter
00018     /// @brief Class for descriptor writer
00019     /// @namespace ven
00020     ///
00021     class DescriptorWriter {
00022
00023     public:
00024
00025         DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :
00026             m_setLayout(setLayout), m_pool{pool} {}
00027
00028         DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00029         DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00030
00031         bool build(VkDescriptorSet &set);
00032         void overwrite(const VkDescriptorSet &set);
00033
00034     private:
00035         DescriptorSetLayout &m_setLayout;
00036         DescriptorPool &m_pool;
00037         std::vector<VkWriteDescriptorSet> m_writes;
00038
00039     }; // class DescriptorWriter
00040
00041 } // namespace ven

```

8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

This file contains the Device class.

```

#include <vector>
#include "VEngine/Window.hpp"

```


8.16 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]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00056         [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00057         [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
VkImageTiling tiling, VkFormatFeatureFlags features) const;
00058
00059         // Buffer Helper Functions
00060         void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00061         [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00062         void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00063         void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00064         void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
uint32_t layerCount) const;
00065
00066         void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00067
00068     private:
00069
00070         void createInstance();
00071         void setupDebugMessenger();
00072         void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00073
00074

```

```

00075         void pickPhysicalDevice();
00076         void createLogicalDevice();
00077         void createCommandPool();
00078
00079         // helper functions
00080         bool isDeviceSuitable(VkPhysicalDevice device) const;
00081         [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00082         [[nodiscard]] bool checkValidationLayerSupport() const;
00083         QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00084         static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
&createInfo);
00085         void hasGlfwRequiredInstanceExtensions() const;
00086         bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00087         SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00088
00089         VkInstance m_instance;
00090         VkDebugUtilsMessengerEXT m_debugMessenger;
00091         VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00092         Window &m_window;
00093         VkCommandPool m_commandPool;
00094
00095         VkDevice m_device;
00096         VkSurfaceKHR m_surface;
00097         VkQueue m_graphicsQueue;
00098         VkQueue m_presentQueue;
00099         VkPhysicalDeviceProperties m_properties;
00100
00101         const std::vector<const char *> validationLayers = {"VK_LAYER_KHRONOS_validation"};
00102         const std::vector<const char *> deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00103
00104     }; // class Device
00105
00106 } // namespace ven

```

8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

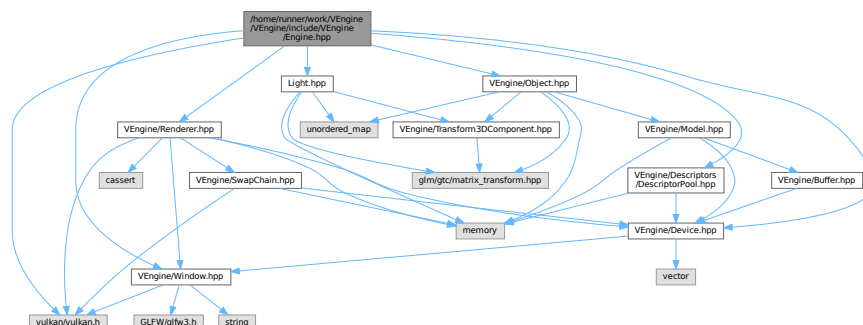
This file contains the Engine class.

```

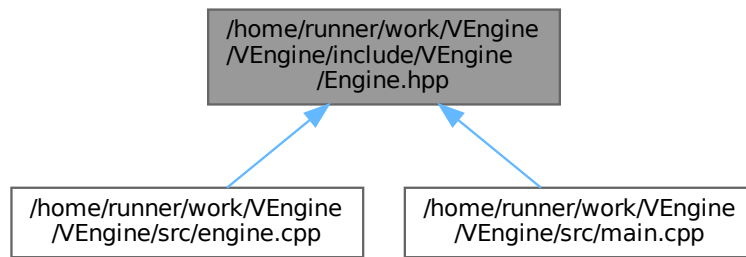
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "Light.hpp"

```

Include dependency graph for Engine.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Engine](#)

Namespaces

- namespace [ven](#)

8.17.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

8.18 Engine.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Engine.hpp
00003 /// @brief This file contains the Engine class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Window.hpp"
00012 #include "VEngine/Device.hpp"
00013 #include "VEngine/Object.hpp"
00014 #include "VEngine/Renderer.hpp"
00015 #include "VEngine/Descriptors/DescriptorPool.hpp"
00016 #include "Light.hpp"
00017
00018 namespace ven {
00019
00020     class Engine {
00021     public:
00022
00023         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string &title
00024             = DEFAULT_TITLE.data());
  
```

```

00025     ~Engine() = default;
00026
00027     Engine(const Engine &) = delete;
00028     Engine operator=(const Engine &) = delete;
00029
00030     void mainLoop();
00031
00032     private:
00033
00034     void loadObjects();
00035
00036     Window m_window;
00037     Device m_device{m_window};
00038     Renderer m_renderer{m_window, m_device};
00039
00040     std::unique_ptr<DescriptorPool> m_globalPool;
00041     Object::Map m_objects;
00042     Light::Map m_lights;
00043
00044     VkInstance m_instance{nullptr};
00045     VkSurfaceKHR m_surface{nullptr};
00046
00047     void createInstance();
00048     void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWWindow(),
nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00049
00050 }; // class Engine
00051
00052 } // namespace ven

```

8.19 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference

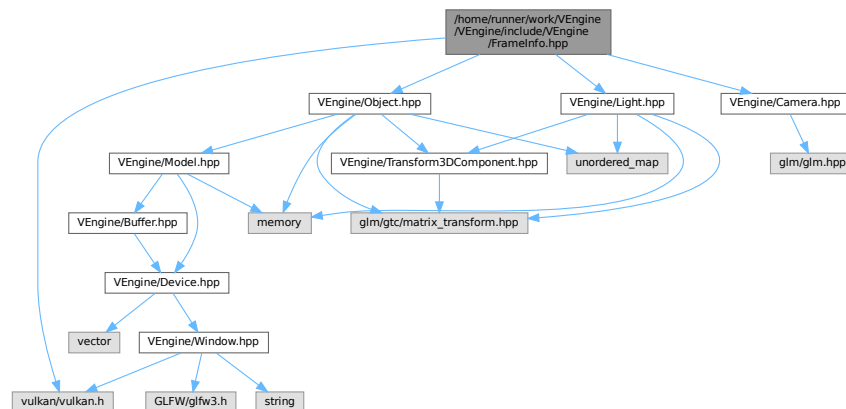
This file contains the FrameInfo class.

```

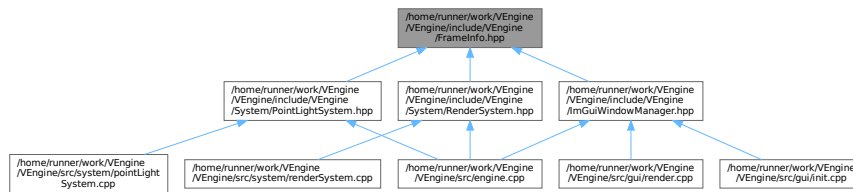
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Light.hpp"

```

Include dependency graph for FrameInfo.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::PointLightData](#)
- struct [ven::GlobalUbo](#)
- struct [ven::FrameInfo](#)

Namespaces

- namespace [ven](#)

Variables

- static constexpr std::size_t [ven::MAX_LIGHTS](#) = 10
- static constexpr float [ven::DEFAULT_AMBIENT_LIGHT_INTENSITY](#) = .2F
- static constexpr glm::vec4 [ven::DEFAULT_AMBIENT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_AMBIENT_LIGHT_INTENSITY](#)}

8.19.1 Detailed Description

This file contains the FrameInfo class.

Definition in file [FrameInfo.hpp](#).

8.20 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 #include "VEngine/Light.hpp"
00014
00015 namespace ven {
00016
00017 static constexpr std::size_t MAX_LIGHTS = 10;
00018
00019 static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F;

```

```

00020 static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F),
    DEFAULT_AMBIENT_LIGHT_INTENSITY};
00021
00022 struct PointLightData
00023 {
00024     glm::vec4 position{};
00025     glm::vec4 color{};
00026 };
00027
00028 struct GlobalUbo
00029 {
00030     glm::mat4 projection{1.F};
00031     glm::mat4 view{1.F};
00032     glm::mat4 inverseView{1.F};
00033     glm::vec4 ambientLightColor{DEFAULT_AMBIENT_LIGHT_COLOR};
00034     std::array<PointLightData, MAX_LIGHTS> pointLights;
00035     int numLights;
00036 };
00037
00038 struct FrameInfo
00039 {
00040     int frameIndex;
00041     float frameTime;
00042     VkCommandBuffer commandBuffer;
00043     Camera &camera;
00044     VkDescriptorSet globalDescriptorSet;
00045     Object::Map &objects;
00046     Light::Map &lights;
00047 };
00048
00049 } // namespace ven

```

8.21 /home/runner/work/VEngine/VEngine/include/VEngine/ImGui/WindowManager.hpp File Reference

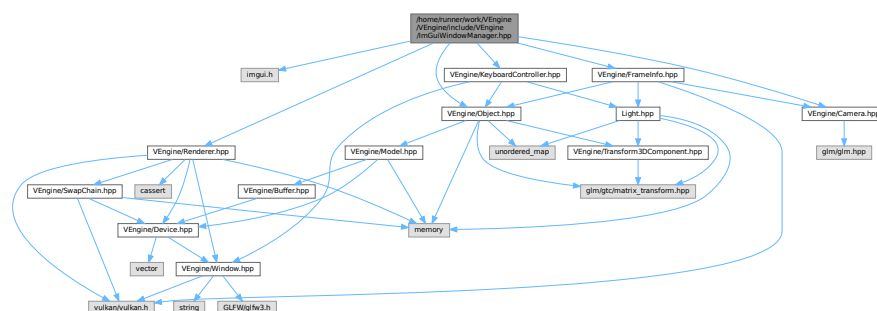
This file contains the ImGuiWindowManager class.

```

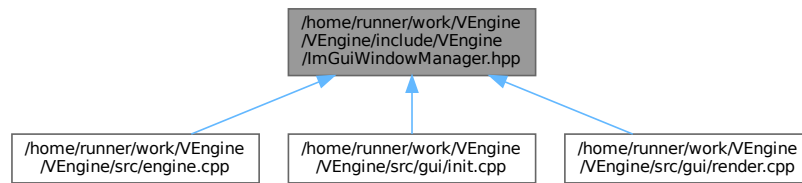
#include <imgui.h>
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/KeyboardController.hpp"
#include "VEngine/FrameInfo.hpp"

```

Include dependency graph for ImGuiWindowManager.hpp:



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](#)

8.21.1 Detailed Description

This file contains the ImGuiWindowManager class.

Definition in file [ImGuiWindowManager.hpp](#).

8.22 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.h>
00010
00011 #include "VEngine/Object.hpp"
00012 #include "VEngine/Renderer.hpp"
00013 #include "VEngine/Camera.hpp"
00014 #include "VEngine/KeyboardController.hpp"
00015 #include "VEngine/FrameInfo.hpp"
00016
00017 namespace ven {
00018
00019     ///
00020     /// @class ImGuiWindowManager
00021     /// @brief Class for ImGui window manager
00022     /// @namespace ven
00023     ///
00024     class ImGuiWindowManager {
00025
00026     public:
00027
00028         ImGuiWindowManager() = default;

```

```

00029         ~ImGuiWindowManager() = default;
00030
00031         ImGuiWindowManager(const ImGuiWindowManager&) = delete;
00032         ImGuiWindowManager& operator=(const ImGuiWindowManager&) = delete;
00033
00034         static void init(GLFWwindow* window, VkInstance instance, const Device* device,
VkRenderPass renderPass);
00035         static void render(Renderer *renderer, std::unordered_map<unsigned int, Object>& objects,
std::unordered_map<unsigned int, Light>& lights, const ImGuiIO& io, Object& cameraObj, Camera& camera,
KeyboardController& cameraController, VkPhysicalDevice physicalDevice, GlobalUbo& ubo);
00036         static void cleanup();
00037
00038     private:
00039
00040         static void initStyle();
00041         static void renderFrameWindow(const ImGuiIO& io);
00042         static void cameraSection(Object& cameraObj, Camera& camera, KeyboardController&
cameraController);
00043         static void inputsSection(const ImGuiIO& io);
00044         static void rendererSection(Renderer *renderer, GlobalUbo& ubo);
00045         static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
00046         static void objectsSection(std::unordered_map<unsigned int, Object>& objects);
00047         static void lightsSection(std::unordered_map<unsigned int, Light>& lights);
00048
00049         struct funcs { static bool IsLegacyNativeDupe(const ImGuiKey key) { return key >= 0 && key
< 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist
00050
00051     }; // class ImGuiWindowManager
00052
00053 } // namespace ven

```

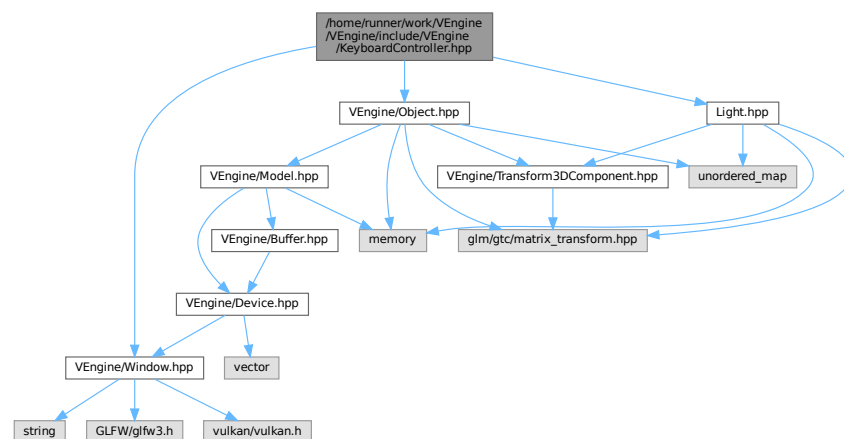
8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Keyboard← Controller.hpp File Reference

```

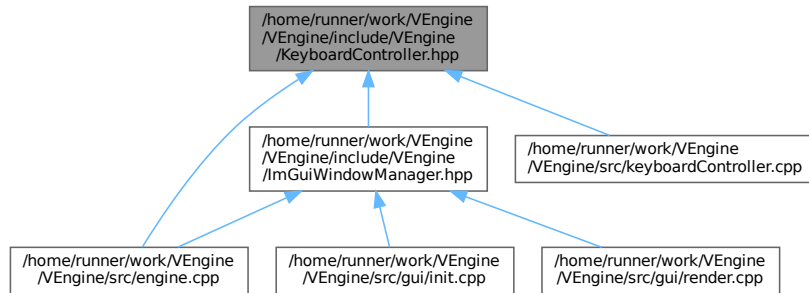
#include "VEngine/Window.hpp"
#include "VEngine/Object.hpp"
#include "Light.hpp"

```

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

8.24 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 #include "Light.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr float DEFAULT_MOVE_SPEED = 3.F;
00016     static constexpr float DEFAULT_LOOK_SPEED = 1.5F;
00017
00018     ///
00019     /// @class KeyboardController
00020     /// @brief Class for keyboard controller
00021     /// @namespace ven
00022     ///
00023     class KeyboardController {
00024     public:
00025 
```

```

00026
00027     struct KeyMappings {
00028         int moveLeft = GLFW_KEY_A;
00029         int moveRight = GLFW_KEY_D;
00030         int moveForward = GLFW_KEY_W;
00031         int moveBackward = GLFW_KEY_S;
00032         int moveUp = GLFW_KEY_SPACE;
00033         int moveDown = GLFW_KEY_LEFT_SHIFT;
00034         int lookLeft = GLFW_KEY_LEFT;
00035         int lookRight = GLFW_KEY_RIGHT;
00036         int lookUp = GLFW_KEY_UP;
00037         int lookDown = GLFW_KEY_DOWN;
00038     };
00039
00040     void moveInPlaneXZ(GLFWwindow* window, float dt, Object& object, bool* showDebugWindow)
00041     const;
00042
00043     void moveInPlaneXZ(GLFWwindow* window, float dt, Light& light, bool* showDebugWindow)
00044     const;
00045
00046     KeyMappings m_keys{};
00047     float m_moveSpeed{DEFAULT_MOVE_SPEED};
00048     float m_lookSpeed{DEFAULT_LOOK_SPEED};
00049 }; // class KeyboardController
00050
00051 } // namespace ven

```

8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp File Reference

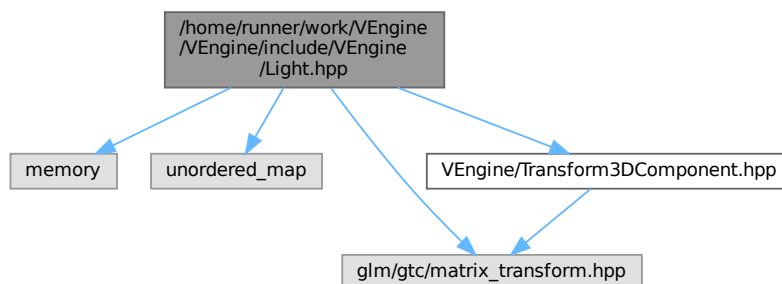
This file contains the Light class.

```

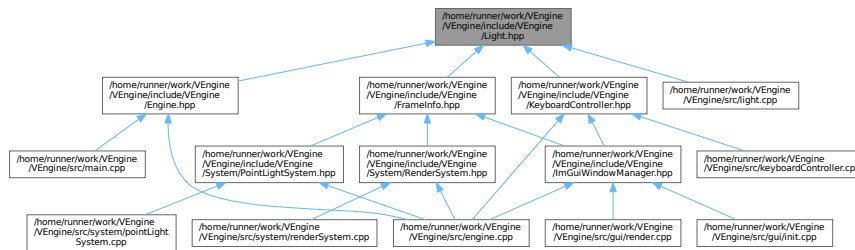
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Transform3DComponent.hpp"

```

Include dependency graph for Light.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Light](#)
Class for light.

Namespaces

- namespace [ven](#)

Variables

- static constexpr float [ven::DEFAULT_LIGHT_INTENSITY](#) = .2F
- static constexpr float [ven::DEFAULT_LIGHT_RADIUS](#) = 0.1F
- static constexpr glm::vec4 [ven::DEFAULT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_LIGHT_INTENSITY](#)}

8.25.1 Detailed Description

This file contains the Light class.

Definition in file [Light.hpp](#).

8.26 Light.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Light.hpp
00003 /// @brief This file contains the Light class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include <glm/gtc/matrix_transform.hpp>
00013
00014 #include "VEngine/Transform3DComponent.hpp"
00015
00016 namespace ven {
00017
00018     static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
```

```

00019     static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
00020     static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY};
00021
00022     ///
00023     /// @class Light
00024     /// @brief Class for light
00025     /// @namespace ven
00026     ///
00027     class Light {
00028
00029     public:
00030
00031         using Map = std::unordered_map<unsigned int, Light>;
00032
00033         ~Light() = default;
00034
00035         Light(const Light&) = delete;
00036         Light& operator=(const Light&) = delete;
00037         Light(Light&&) = default;
00038         Light& operator=(Light&&) = default;
00039
00040         static Light createLight(float radius = DEFAULT_LIGHT_RADIUS, glm::vec4 color =
DEFAULT_LIGHT_COLOR);
00041
00042         glm::vec4 color{DEFAULT_LIGHT_COLOR};
00043         Transform3DComponent transform3D{};
00044
00045         [[nodiscard]] unsigned int getId() const { return m_lightId; }
00046         [[nodiscard]] std::string getName() const { return m_name; }
00047
00048         void setName(const std::string &name) { m_name = name; }
00049
00050     private:
00051
00052         explicit Light(const unsigned int lightId) : m_lightId(lightId) {}
00053
00054         unsigned int m_lightId;
00055         std::string m_name{"point light"};
00056
00057     }; // class Light
00058
00059 } // namespace ven

```

8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference

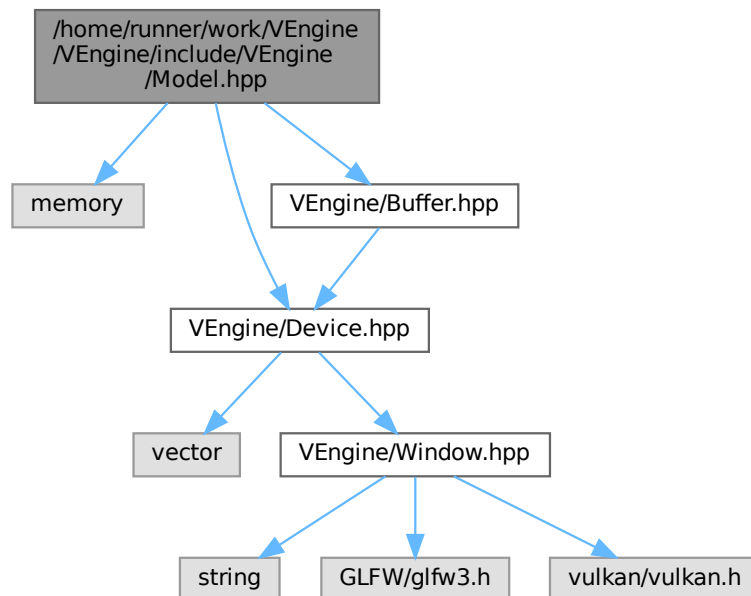
This file contains the Model class.

```

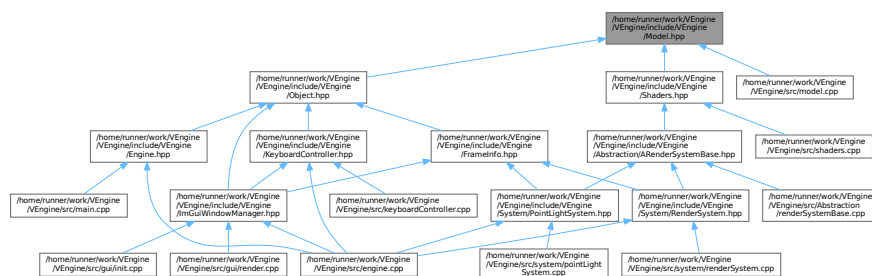
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"

```

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**

8.27.1 Detailed Description

This file contains the Model class.

Definition in file [Model.hpp](#).

8.28 Model.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Buffer.hpp"
00013
00014 namespace ven {
00015
00016     ///
00017     /// @class Model
00018     /// @brief Class for model
00019     /// @namespace ven
00020     ///
00021     class Model {
00022
00023     public:
00024
00025         struct Vertex {
00026             glm::vec3 position{};
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

```

8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

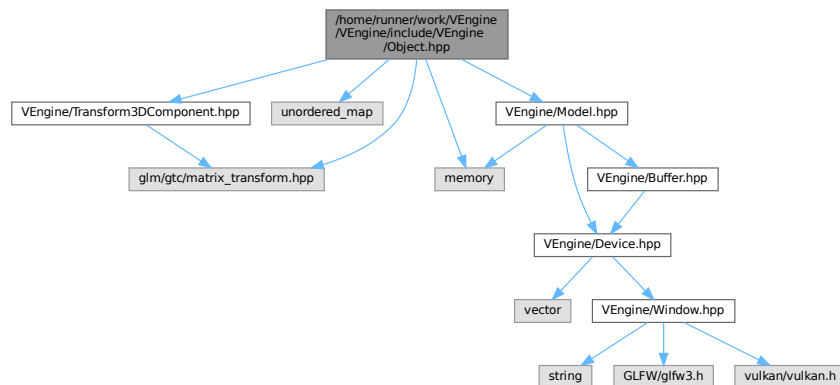
This file contains the Object class.

```

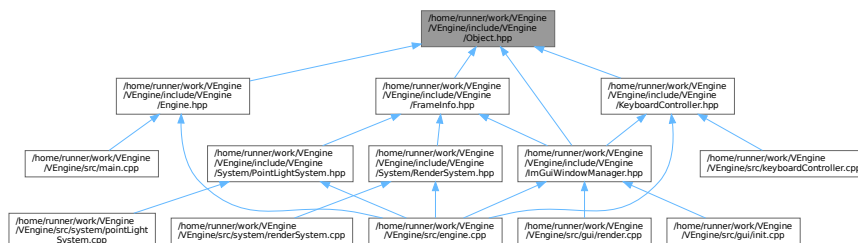
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"
#include "VEngine/Transform3DComponent.hpp"

```

Include dependency graph for Object.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Object`
Class for object.

Namespaces

- namespace [ven](#)

8.29.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

8.30 Object.hpp

[Go to the documentation of this file.](#)

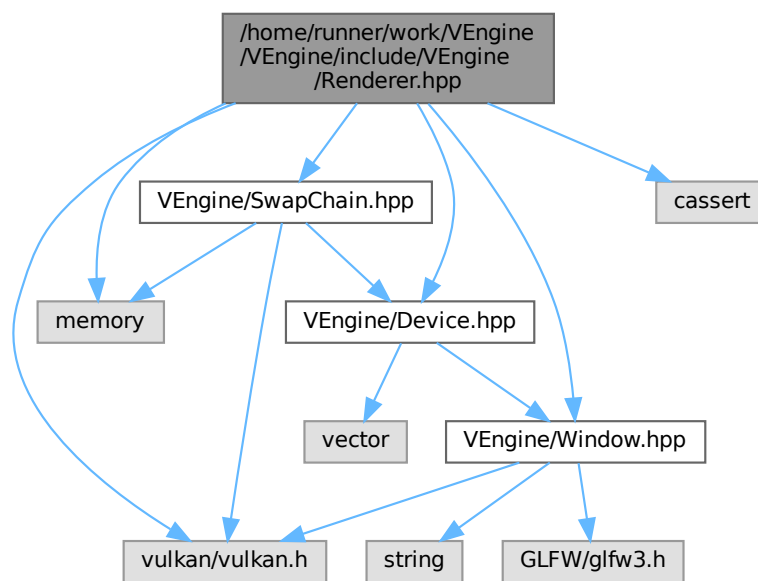
```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <memory>  
00010 #include <unordered_map>  
00011 ///  
00012 #include <glm/gtc/matrix_transform.hpp>  
00013 ///  
00014 #include "VEngine/Model.hpp"  
00015 #include "VEngine/Transform3DComponent.hpp"  
00016 ///  
00017 namespace ven {  
00018 ///  
00019 ///  
00020 ///  
00021 ///  
00022 ///  
00023 ///  
00024 class Object {  
00025     public:  
00026         using Map = std::unordered_map<unsigned int, Object>;  
00027         ~Object() = default;  
00028         Object(const Object&) = delete;  
00029         Object& operator=(const Object&) = delete;  
00030         Object(Object&&) = default;  
00031         Object& operator=(Object&&) = default;  
00032         static Object createObject() { static unsigned int objId = 0; return Object(objId++); }  
00033         [[nodiscard]] unsigned int getId() const { return m_objId; }  
00034         [[nodiscard]] std::string getName() const { return m_name; }  
00035         [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }  
00036         void setName(const std::string &name) { m_name = name; }  
00037         void setModel(const std::shared_ptr<Model> &model) { m_model = model; }  
00038         glm::vec3 color{};  
00039         Transform3DComponent transform3D{};  
00040     private:  
00041         explicit Object(const unsigned int objId) : m_objId(objId) {}  
00042         unsigned int m_objId;  
00043         std::string m_name{};  
00044         std::shared_ptr<Model> m_model{};  
00045     }; // class Object  
00046 } // namespace ven
```

8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

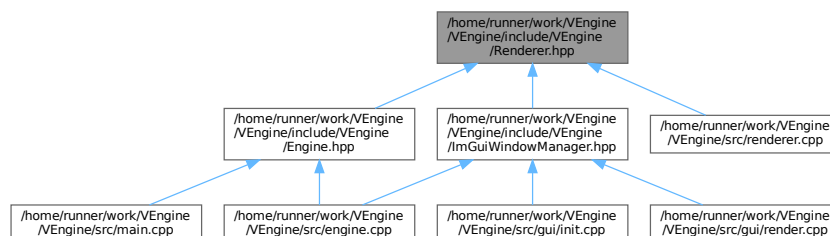
This file contains the `Renderer` class.

```
#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}

8.31.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

8.32 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         [[nodiscard]] Window& getWindow() { return m_window; }
```

```

00048         void setClearColor(VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00049         VkCommandBuffer beginFrame();
00050         void endFrame();
00051         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00052         void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00053
00054     private:
00055
00056         void createCommandBuffers();
00057         void freeCommandBuffers();
00058         void recreateSwapChain();
00059
00060         Window &m_window;
00061         Device &m_device;
00062         std::unique_ptr<SwapChain> m_swapChain;
00063         std::vector<VkCommandBuffer> m_commandBuffers;
00064         std::array<VkClearColorValue, 2> m_clearValues;
00065
00066         uint32_t m_currentImageIndex{0};
00067         int m_currentFrameIndex{0};
00068         bool m_isFrameStarted{false};
00069
00070     }; // class Renderer
00071
00072 } // namespace ven

```

8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Shader.h File Reference

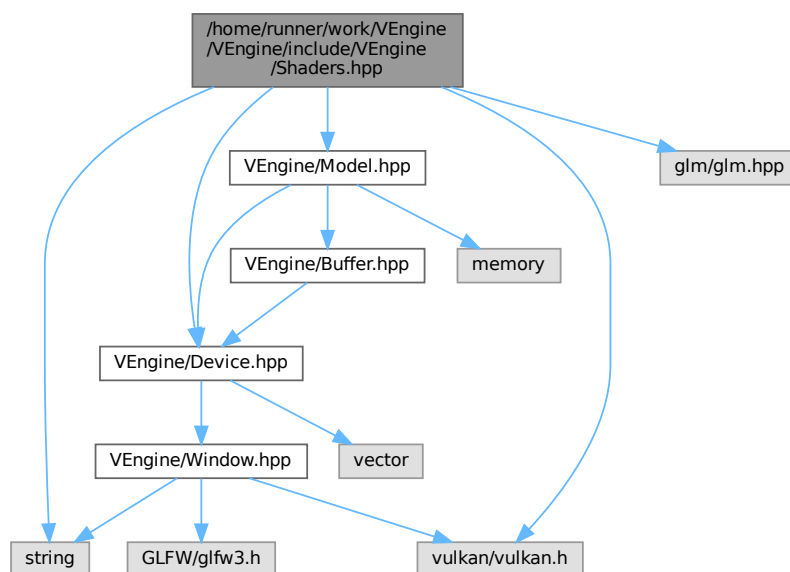
This file contains the Shader class.

```

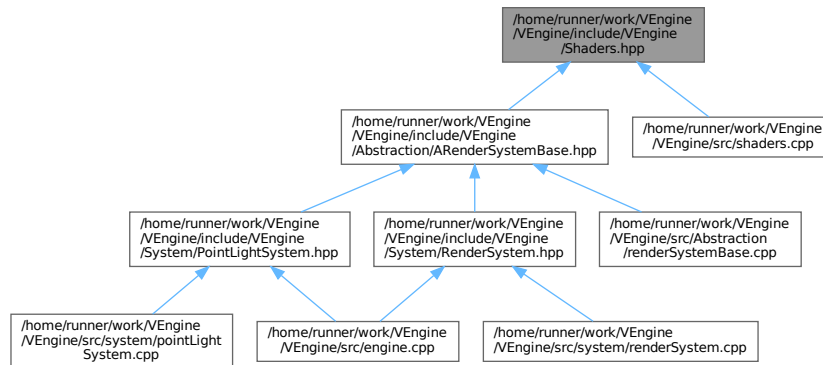
#include <string>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
#include "VEngine/Device.hpp"
#include "VEngine/Model.hpp"

```

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/"

8.33.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

8.34 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
00075     }; // class Shaders
00076
00077 } // namespace ven

```

8.35 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

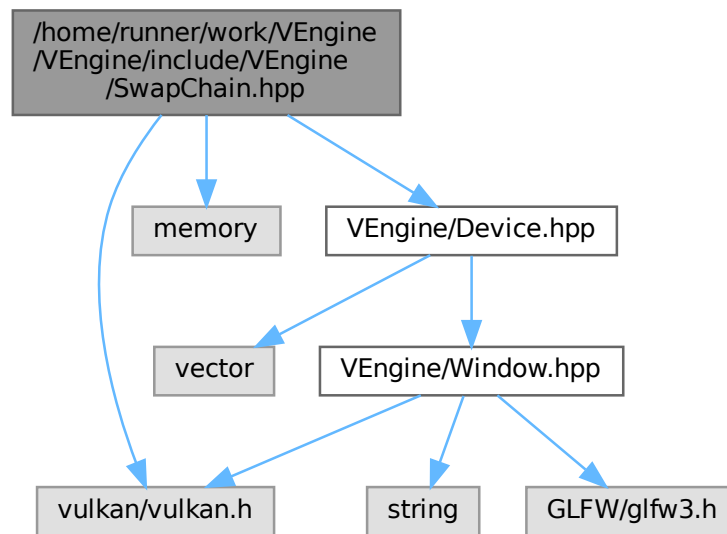
This file contains the Shader class.

```

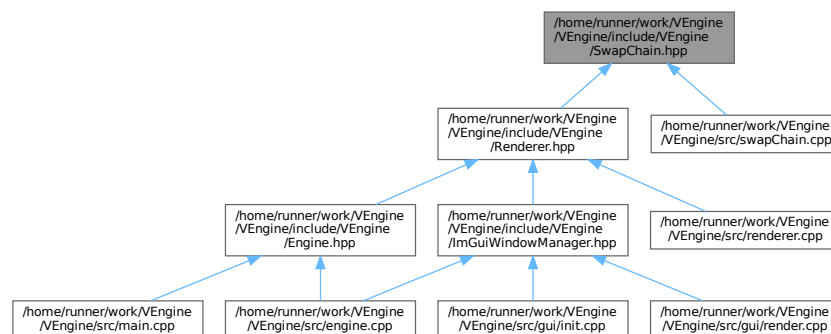
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"

```

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](#)

8.35.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

8.36 SwapChain.hpp

[Go to the documentation of this file.](#)

```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
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     ///  
00018     ///  
00019     ///  
00020     ///  
00021     class SwapChain {  
00022     public:  
00023         ///  
00024         static constexpr int MAX_FRAMES_IN_FLIGHT = 2;  
00025         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : m_device{deviceRef},  
00026 m_windowExtent{windowExtentRef} { init(); }  
00027         SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>  
00028 previous) : m_device{deviceRef}, m_windowExtent{windowExtentRef}, m_oldSwapChain{std::move(previous)}  
00029 { init(); m_oldSwapChain = nullptr; }  
00030         ~SwapChain();  
00031         SwapChain(const SwapChain &) = delete;  
00032         SwapChain& operator=(const SwapChain &) = delete;  
00033         [[nodiscard]] VkFramebuffer getFramebuffer(const unsigned long index) const { return  
00034 m_swapChainFrameBuffers[index]; }  
00035         [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }  
00036         [[nodiscard]] VkImageView getImageView(const int index) const { return  
00037 m_swapChainImageViews[static_cast<unsigned long>(index)]; }  
00038         [[nodiscard]] size_t imageCount() const { return m_swapChainImages.size(); }  
00039         [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_swapChainImageFormat; }  
00040         [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }  
00041         [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }  
00042         [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }  
00043         [[nodiscard]] float extentAspectRatio() const { return  
00044 static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }  
00045         [[nodiscard]] VkFormat findDepthFormat() const;  
00046         VkResult acquireNextImage(uint32_t *imageIndex) const;  
00047         VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);  
00048         [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return  
00049 m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat ==  
00050 swapChain.m_swapChainDepthFormat; }  
00051     private:  
00052         void init();  
00053         void createSwapChain();  
00054         void createImageViews();  
00055         void createDepthResources();  
00056         void createRenderPass();  
00057         void createFrameBuffers();  
00058         void createSyncObjects();  
00059         static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>  
00060 &availableFormats);
```

```

00062         static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
00063             &availablePresentModes);
00063     [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
00064     const;
00064
00065     VkFormat m_swapChainImageFormat{};
00066     VkFormat m_swapChainDepthFormat{};
00067     VkExtent2D m_swapChainExtent{};
00068
00069     std::vector<VkFramebuffer> m_swapChainFrameBuffers;
00070     VkRenderPass m_renderPass{};
00071
00072     std::vector<VkImage> m_depthImages;
00073     std::vector<VkDeviceMemory> m_depthImageMemory;
00074     std::vector<VkImageView> m_depthImageViews;
00075     std::vector<VkImage> m_swapChainImages;
00076     std::vector<VkImageView> m_swapChainImageViews;
00077
00078     Device &m_device;
00079     VkExtent2D m_windowExtent;
00080
00081     VkSwapchainKHR m_swapChain{};
00082     std::shared_ptr<SwapChain> m_oldSwapChain;
00083
00084     std::vector<VkSemaphore> m_imageAvailableSemaphores;
00085     std::vector<VkSemaphore> m_renderFinishedSemaphores;
00086     std::vector<VkFence> m_inFlightFences;
00087     std::vector<VkFence> m_imagesInFlight;
00088     size_t m_currentFrame{0};
00089
00090 }; // class SwapChain
00091
00092 } // namespace ven

```

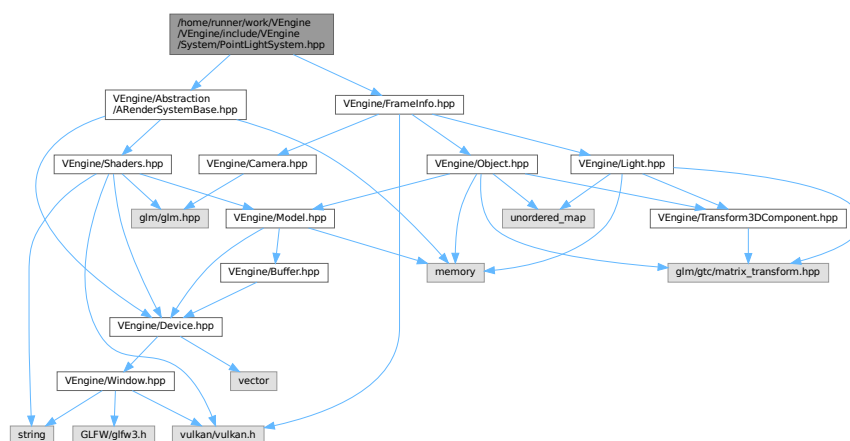
8.37 /home/runner/work/VEngine/VEngine/include/VEngine/System/↵ PointLightSystem.hpp File Reference

This file contains the PointLightSystem class.

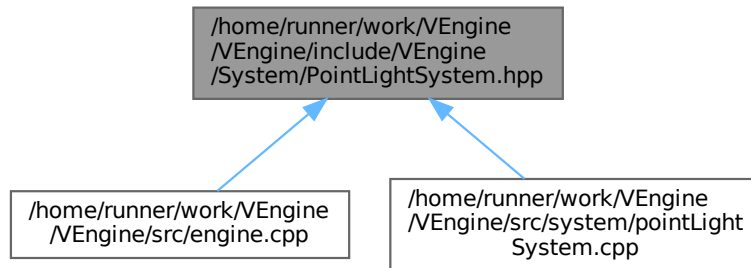
```
#include "VEngine/Abstraction/ARenderSystemBase.hpp"
```

```
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for PointLightSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::LightPushConstantData](#)
- class [ven::PointLightSystem](#)
Class for point light system.

Namespaces

- namespace [ven](#)

8.37.1 Detailed Description

This file contains the PointLightSystem class.

Definition in file [PointLightSystem.hpp](#).

8.38 PointLightSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file PointLightSystem.hpp
00003 /// @brief This file contains the PointLightSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Abstraction/ARenderSystemBase.hpp"
00010 #include "VEngine/FrameInfo.hpp"
00011
00012 namespace ven {
00013
00014     struct LightPushConstantData {
00015         glm::vec4 position{};
00016         glm::vec4 color{};
00017         float radius;
00018     };
00019
00020     ///
00021     /// @class PointLightSystem
00022     /// @brief Class for point light system
  
```

```

00023     /// @namespace ven
00024     ///
00025     class PointLightSystem : public ARenderSystemBase {
00026     public:
00027
00028
00029         explicit PointLightSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00030             createPipelineLayout(globalSetLayout, sizeof(LightPushConstantData));
00031             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "point_light_vert.spv",
std::string(SHADERS_BIN_PATH) + "point_light_frag.spv", true);
00032         }
00033
00034         void render(const FrameInfo &frameInfo) const;
00035         static void update(const FrameInfo &frameInfo, GlobalUbo &ubo);
00036
00037     }; // class PointLightSystem
00038
00039 } // namespace ven

```

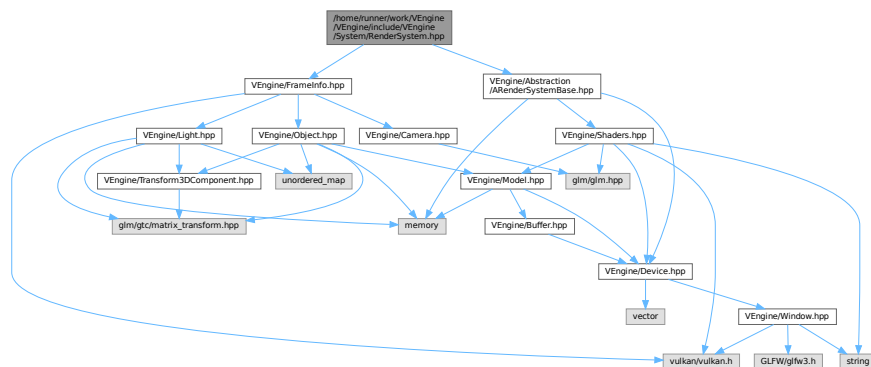
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp File Reference

This file contains the RenderSystem class.

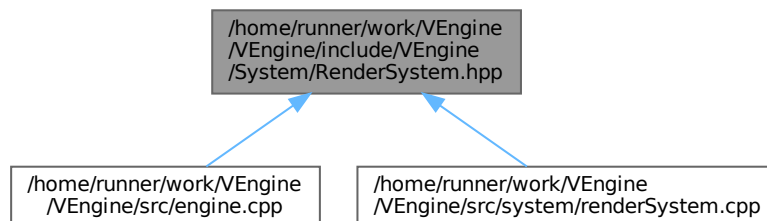
```
#include "VEngine/FrameInfo.hpp"
```

```
#include "VEngine/Abstraction/ARenderSystemBase.hpp"
```

Include dependency graph for RenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::ObjectPushConstantData](#)
- class [ven::RenderSystem](#)
Class for render system.

Namespaces

- namespace [ven](#)

8.39.1 Detailed Description

This file contains the RenderSystem class.

Definition in file [RenderSystem.hpp](#).

8.40 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 "VEngine/FrameInfo.hpp"
00010 #include "VEngine/Abstraction/ARenderSystemBase.hpp"
00011
00012 namespace ven {
00013
00014     struct ObjectPushConstantData {
00015         glm::mat4 modelMatrix{};
00016         glm::mat4 normalMatrix{};
00017     };
00018
00019     ///
00020     /// @class RenderSystem
00021     /// @brief Class for render system
00022     /// @namespace ven
00023     ///
00024     class RenderSystem : public ARenderSystemBase {
00025
00026     public:
00027
00028         explicit RenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00029             createPipelineLayout(globalSetLayout, sizeof(ObjectPushConstantData));
00030             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "shader_vert.spv",
std::string(SHADERS_BIN_PATH) + "shader_frag.spv", false);
00031         }
00032
00033         RenderSystem(const RenderSystem&) = delete;
00034         RenderSystem& operator=(const RenderSystem&) = delete;
00035
00036         void renderObjects(const FrameInfo &frameInfo) const;
00037
00038     }; // class RenderSystem
00039
00040 } // namespace ven

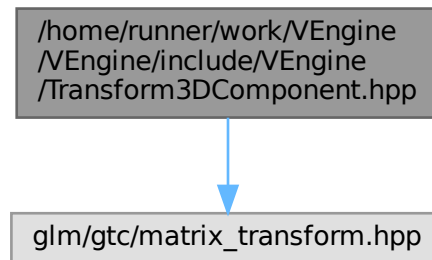
```

8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp File Reference

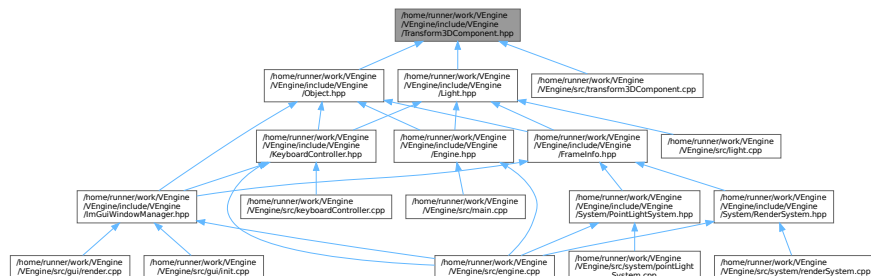
This file contains the Transform3DComponent class.

```
#include <glm/gtc/matrix_transform.hpp>
```

Include dependency graph for Transform3DComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Transform3DComponent](#)

Namespaces

- namespace [ven](#)

8.41.1 Detailed Description

This file contains the Transform3DComponent class.

Definition in file [Transform3DComponent.hpp](#).

8.42 Transform3DComponent.hpp

[Go to the documentation of this file.](#)

```

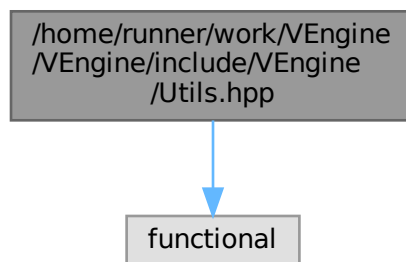
00001 ///
00002 /// @file Transform3DComponent.hpp
00003 /// @brief This file contains the Transform3DComponent class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <glm/gtc/matrix_transform.hpp>
00010
00011 namespace ven {
00012
00013     class Transform3DComponent {
00014
00015     public:
00016
00017         glm::vec3 translation{};
00018         glm::vec3 scale{1.F, 1.F, 1.F};
00019         glm::vec3 rotation{};
00020
00021         [[nodiscard]] glm::mat4 mat4() const;
00022         [[nodiscard]] glm::mat3 normalMatrix() const;
00023
00024     }; // class Transform3DComponent
00025
00026 } // namespace ven

```

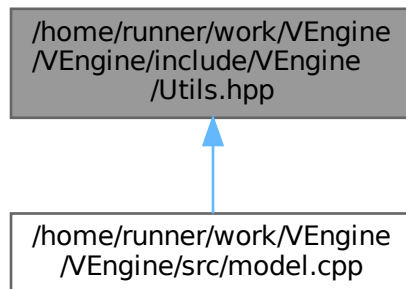
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

```
#include <functional>
```

Include dependency graph for Utils.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [ven](#)

Functions

- `template<typename T, typename... Rest>`
`void ven::hashCombine (std::size_t &seed, const T &v, const Rest &... rest)`

8.44 Utlis.hpp

[Go to the documentation of this file.](#)

```

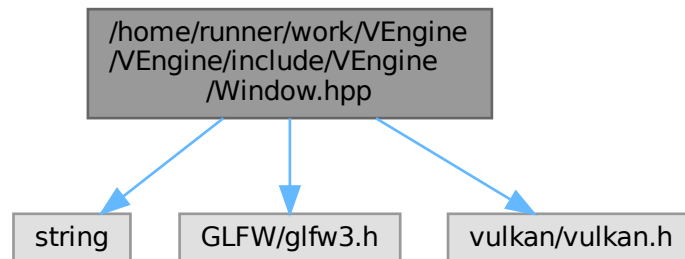
00001 ///
00002 /// @file Utlis.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018
00019 } // namespace ven
  
```

8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp

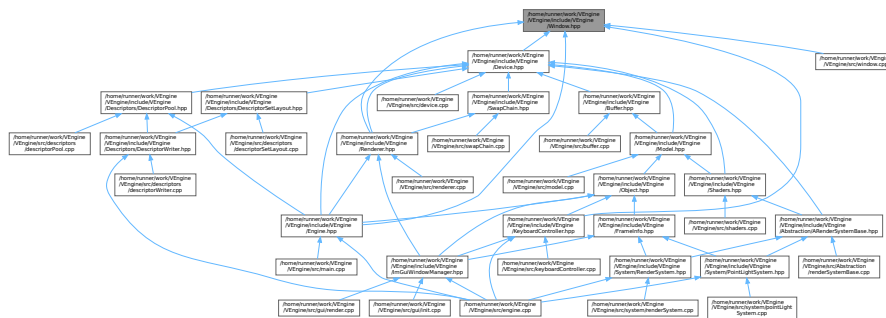
File Reference

This file contains the Window class.


```
#include <string>
#include <GLFW/glfw3.h>
#include <vulkan/vulkan.h>
Include dependency graph for Window.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Window`
Class for window.

Namespaces

- namespace **ven**

Macros

- #define GLFW_INCLUDE_VULKAN

Variables

- static constexpr uint32_t `ven::DEFAULT_WIDTH` = 1920
- static constexpr uint32_t `ven::DEFAULT_HEIGHT` = 1080
- static constexpr std::string_view `ven::DEFAULT_TITLE` = "VEngine"

8.45.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

8.45.2 Macro Definition Documentation

8.45.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

8.46 Window.hpp

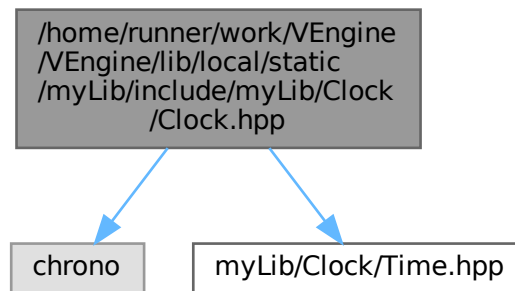
[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00018     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00019     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00020
00021     ///
00022     /// @class Window
00023     /// @brief Class for window
00024     /// @namespace ven
00025     ///
00026     class Window {
00027     public:
00028
00029         Window(const uint32_t width, const uint32_t height, const std::string &title) :
00030             m_window(createWindow(width, height, title)), m_width(width), m_height(height) {};
00031         ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr; };
00032
00033         [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
00034             &title);
00035         void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00036
00037         [[nodiscard]] GLFWwindow* getGLFWWindow() const { return m_window; };
00038
00039         [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; };
00040         [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; };
00041         void resetWindowResizedFlag() { m_framebufferResized = false; };
00042
00043         void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);
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
00056 } // namespace ven
```

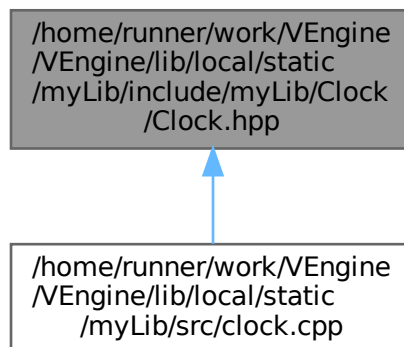
8.47 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference

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.

8.47.1 Detailed Description

Clock class for time management.

Definition in file [Clock.hpp](#).

8.47.2 Typedef Documentation

8.47.2.1 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](#).

8.48 Clock.hpp

[Go to the documentation of this file.](#)

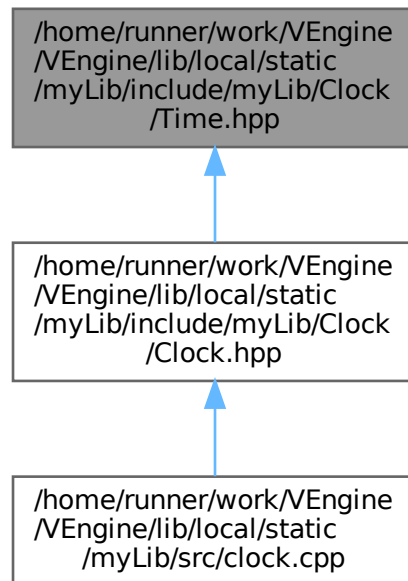
```
00001 ///
00002 /// @file Clock.hpp
00003 /// @brief Clock class for time management
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 #include <chrono>
00010
00011 #include "myLib/Clock/Time.hpp"
00012
00013 ///
00014 /// @brief TimePoint is a type alias for a time point which is a very long and complicated type in the
00015 /// standard library
00016
00017 using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;
00018
00019 namespace myLib {
00020
00021     ///
00022     /// @brief Class for time management
00023     ///
00024     class Clock {
00025     public:
00026
00027         Clock() : m_start(std::chrono::high_resolution_clock::now()) {};
00028
00029         ~Clock() = default;
00030
00031         ///
00032         /// @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         ///  
00052     private:  
00053         ///  
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         ///  
00069     }; // Clock  
00070     ///  
00071 } // namespace myLib
```

8.49 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/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

8.49.1 Detailed Description

Class for time management.

Definition in file [Time.hpp](#).

8.50 Time.hpp

[Go to the documentation of this file.](#)

```

00001 ///  

00002 ///  

00003 ///  

00004 ///  

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     ///  

00016     ///  

00017     ///  

00018     class Time {  

00019     public:  

00020     public:  

00021     public:  

00022         ///  

00023         ///  

00024         ///  

00025         explicit Time(const double seconds) : m_seconds(seconds) {};  

00026         ///  

00027         ///  

00028         ///  

00029         ///  

00030         ///  

00031         [[nodiscard]] int asSeconds() const { return static_cast<int>(m_seconds); };  

00032         ///  

00033         ///  

00034         ///  

00035         ///  

00036         ///  

00037         [[nodiscard]] int asMilliseconds() const { return static_cast<int>(m_seconds *  

MILLISECONDS_PER_SECOND); }  

00038         ///  

00039         ///  

00040         ///  

00041         ///  

00042         ///  

00043         [[nodiscard]] int asMicroseconds() const { return static_cast<int>(m_seconds *  

MICROSECONDS_PER_SECOND); };  

00044     private:  

00045     private:  

00046     private:  

00047         ///  

00048         ///  

00049         ///  

00050         double m_seconds{0.0F};  

00051     }; // Time  

00052     ///  

00053     ///  

00054 } // namespace myLib

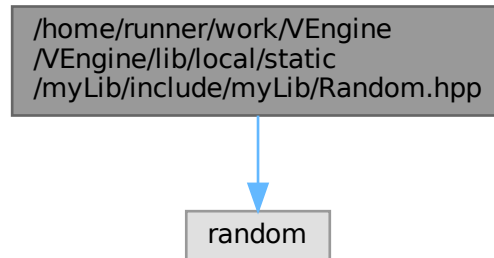
```

8.51 /home/runner/work/VEngine/VEngine/lib/local/static/my↵ Lib/include/myLib/Random.hpp File Reference

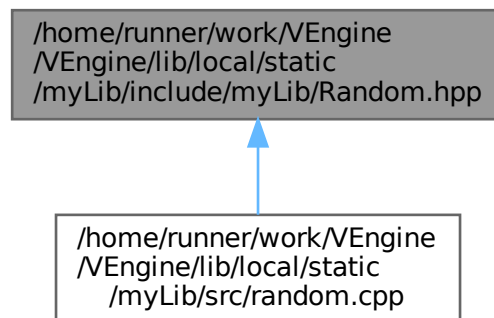
Class for random number generation.

```
#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

8.51.1 Detailed Description

Class for random number generation.

Definition in file [Random.hpp](#).

8.52 Random.hpp

[Go to the documentation of this file.](#)

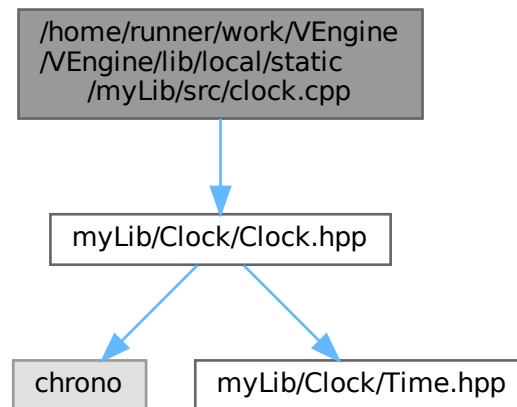
```

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
```

8.53 /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:



8.54 clock.cpp

[Go to the documentation of this file.](#)

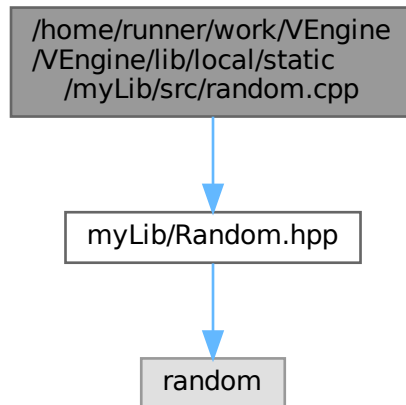
```

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 }

```

8.55 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference

```
#include "myLib/Random.hpp"
Include dependency graph for random.cpp:
```



8.56 random.cpp

[Go to the documentation of this file.](#)

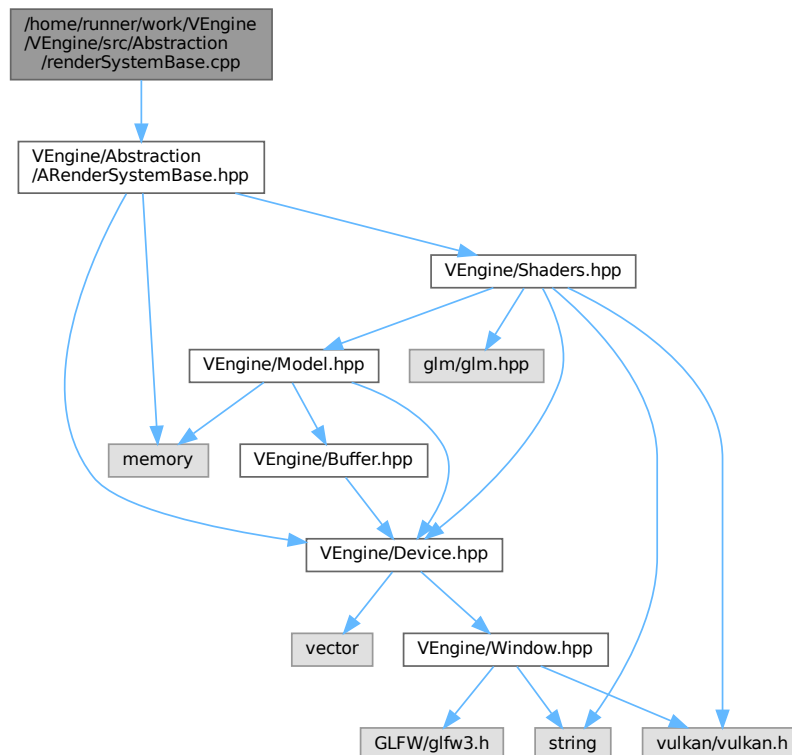
```
00001 #include "myLib/Random.hpp"
00002
00003 int myLib::Random::randomInt(const int min, const int max)
00004 {
00005     std::mt19937 gen(std::random_device{}());
00006     std::uniform_int_distribution<> dis(min, max);
00007     return dis(gen);
00008 }
00009
00010 float myLib::Random::randomFloat(const 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 }
```

8.57 /home/runner/work/VEngine/VEngine/README.md File Reference

8.58 /home/runner/work/VEngine/VEngine/src/Abstraction/renderSystemBase.cpp File Reference

```
#include "VEngine/Abstraction/ARenderSystemBase.hpp"
```

Include dependency graph for renderSystemBase.cpp:



8.59 renderSystemBase.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Abstraction/ARenderSystemBase.hpp"
00002
00003 void ven::ARenderSystemBase::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout, const
uint32_t pushConstantSize)
00004 {
00005     VkPushConstantRange pushConstantRange{};
00006     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00007     pushConstantRange.offset = 0;
00008     pushConstantRange.size = pushConstantSize;
00009
00010     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};
00011
00012     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00013     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00014     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00015     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00016     pipelineLayoutInfo.pushConstantRangeCount = 1;
00017     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;

```

```

00018     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
VK_SUCCESS)
00019     {
00020         throw std::runtime_error("Failed to create pipeline layout");
00021     }
00022 }
00023
00024 void ven::ARenderSystemBase::createPipeline(const VkRenderPass renderPass, const std::string
&shadersVertPath, const std::string &shadersFragPath, const bool isLight)
00025 {
00026     assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
00027     PipelineConfigInfo pipelineConfig{};
00028     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00029     if (isLight) {
00030         pipelineConfig.attributeDescriptions.clear();
00031         pipelineConfig.bindingDescriptions.clear();
00032     }
00033     pipelineConfig.renderPass = renderPass;
00034     pipelineConfig.pipelineLayout = m_pipelineLayout;
00035     m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
00036 }

```

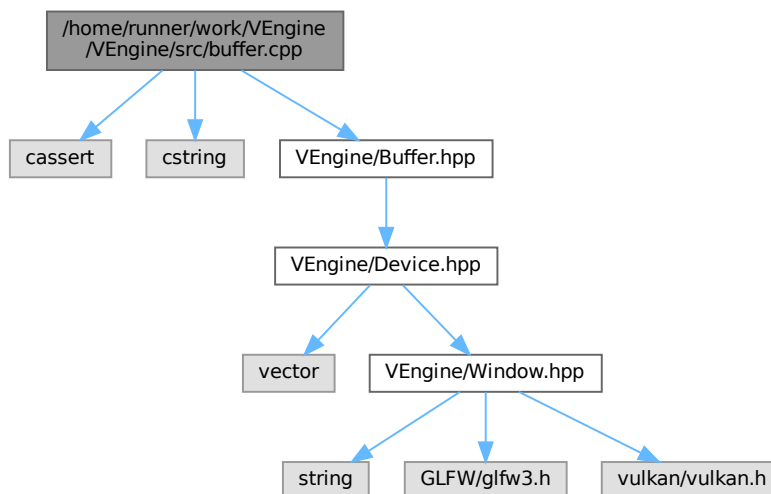
8.60 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```

#include <cassert>
#include <cstring>
#include "VEngine/Buffer.hpp"

```

Include dependency graph for buffer.cpp:



8.61 buffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <cstring>
00003
00004 #include "VEngine/Buffer.hpp"
00005
00006 VkDeviceSize ven::Buffer::getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
minOffsetAlignment) {
00007     if (minOffsetAlignment > 0) {

```

```

00008         return (instanceSize + minOffsetAlignment - 1) & ~(minOffsetAlignment - 1);
00009     }
00010     return instanceSize;
00011 }
00012
00013 ven::Buffer::Buffer(Device &device, const VkDeviceSize instanceSize, const uint32_t instanceCount,
    const VkBufferUsageFlags usageFlags, const VkMemoryPropertyFlags memoryPropertyFlags, const
    VkDeviceSize minOffsetAlignment) : m_device{device}, m_instanceSize{instanceSize},
    m_instanceCount{instanceCount}, m_alignmentSize{getAlignment(instanceSize, minOffsetAlignment)},
    m_usageFlags{usageFlags}, m_memoryPropertyFlags{memoryPropertyFlags}
00014 {
00015     m_bufferSize = m_alignmentSize * m_instanceCount;
00016     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00017 }
00018
00019 ven::Buffer::~Buffer()
00020 {
00021     unmap();
00022     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00023     vkFreeMemory(m_device.device(), m_memory, nullptr);
00024 }
00025
00026 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00027 {
00028     assert(m_buffer && m_memory && "Called map on m_buffer before create");
00029     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00030 }
00031
00032 void ven::Buffer::unmap()
00033 {
00034     if (m_mapped != nullptr) {
00035         vkUnmapMemory(m_device.device(), m_memory);
00036         m_mapped = nullptr;
00037     }
00038 }
00039
00040 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
    const
00041 {
00042     assert(m_mapped && "Cannot copy to unmapped m_buffer");
00043
00044     if (size == VK_WHOLE_SIZE) {
00045         memcpy(m_mapped, data, m_bufferSize);
00046     } else {
00047         char *memOffset = static_cast<char *>(m_mapped);
00048         memOffset += offset;
00049         memcpy(memOffset, data, size);
00050     }
00051 }
00052
00053 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00054 {
00055     VkMappedMemoryRange mappedRange = {};
00056     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00057     mappedRange.memory = m_memory;
00058     mappedRange.offset = offset;
00059     mappedRange.size = size;
00060     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00061 }
00062
00063 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00064 {
00065     VkMappedMemoryRange mappedRange = {};
00066     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00067     mappedRange.memory = m_memory;
00068     mappedRange.offset = offset;
00069     mappedRange.size = size;
00070     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00071 }

```

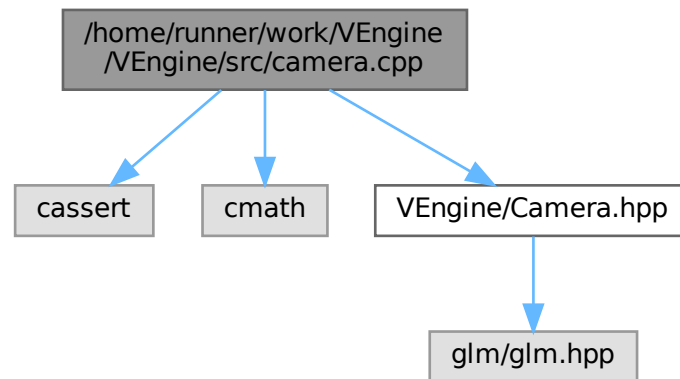
8.62 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```

#include <cassert>
#include <cmath>
#include "VEngine/Camera.hpp"

```

Include dependency graph for camera.cpp:



8.63 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;

```

```

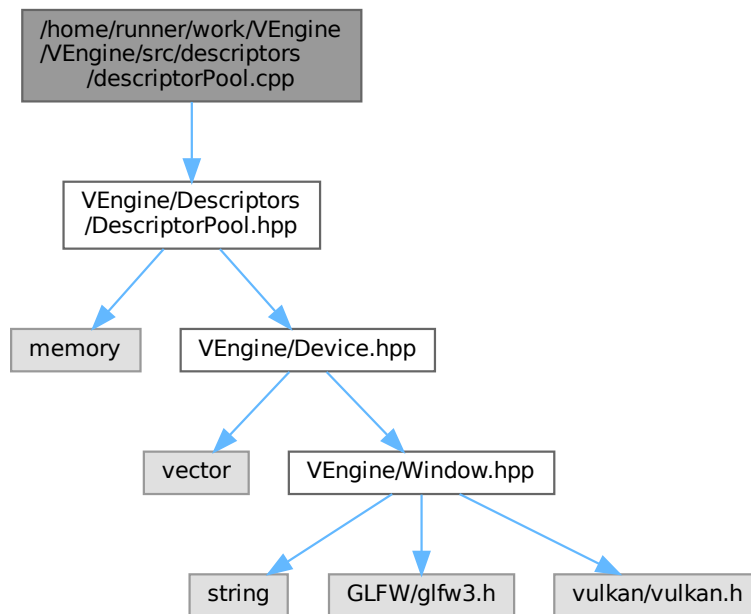
00045     m_viewMatrix[3][0] = -dot(u, position);
00046     m_viewMatrix[3][1] = -dot(v, position);
00047     m_viewMatrix[3][2] = -dot(w, position);
00048
00049     m_inverseViewMatrix = glm::mat4(1.F);
00050     m_inverseViewMatrix[0][0] = u.x;
00051     m_inverseViewMatrix[0][1] = u.y;
00052     m_inverseViewMatrix[0][2] = 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 }

```


8.64 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference

```
#include "VEngine/Descriptors/DescriptorPool.hpp"
```

Include dependency graph for descriptorPool.cpp:



8.65 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 }

```

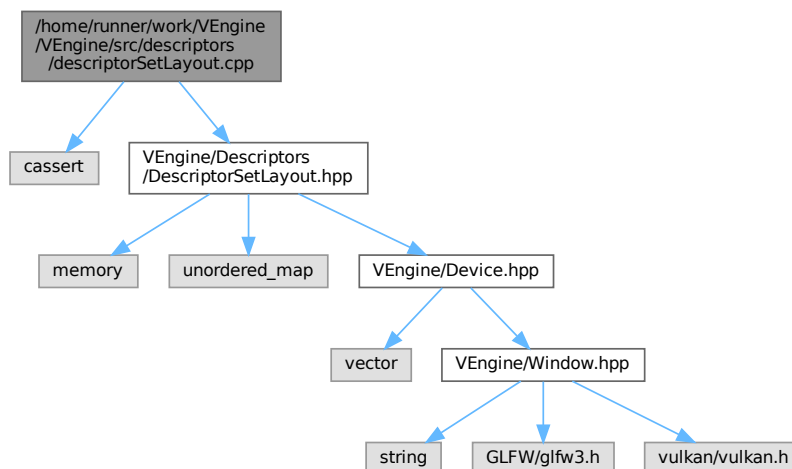
8.66 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

Include dependency graph for descriptorSetLayout.cpp:



8.67 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 }

```

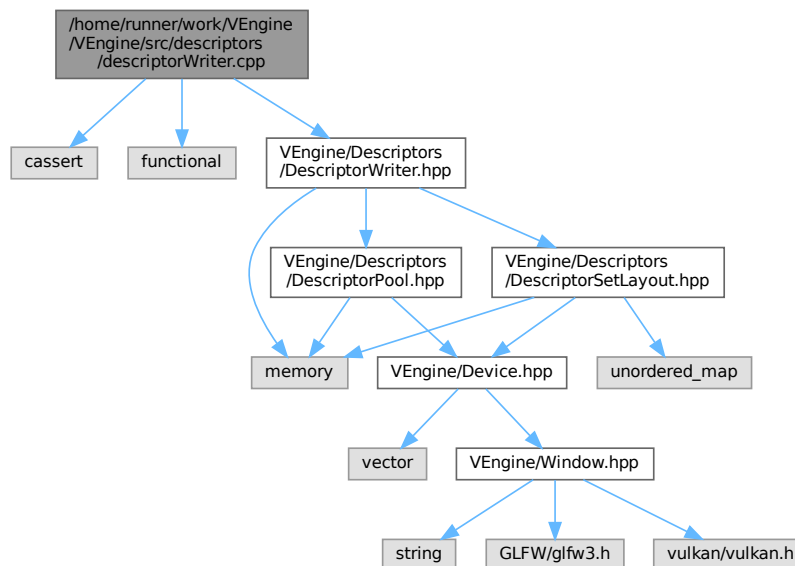
8.68 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference ↩

```

#include <cassert>
#include <functional>
#include "VEngine/Descriptors/DescriptorWriter.hpp"

```

Include dependency graph for descriptorWriter.cpp:



8.69 descriptorWriter.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <functional>
00003
00004 #include "VEngine/Descriptors/DescriptorWriter.hpp"
00005
00006 ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const
VkDescriptorBufferInfo *bufferInfo)
00007 {
00008     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00009
00010     const auto &bindingDescription = m_setLayout.m_bindings.at(binding);
00011
00012     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00013
00014     VkWriteDescriptorSet write{};
00015     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00016     write.descriptorType = bindingDescription.descriptorType;
00017     write.dstBinding = binding;
00018     write.pBufferInfo = bufferInfo;
00019     write.descriptorCount = 1;
00020
00021     m_writes.push_back(write);
00022     return *this;
00023 }
00024
00025 ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const
VkDescriptorImageInfo *imageInfo)
00026 {
00027     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00028
00029     const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings.at(binding);
00030
00031     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00032
00033     VkWriteDescriptorSet write{};
00034     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00035     write.descriptorType = bindingDescription.descriptorType;
00036     write.dstBinding = binding;
00037     write.pImageInfo = imageInfo;
00038     write.descriptorCount = 1;
00039
00040     m_writes.push_back(write);
00041     return *this;
00042 }
00043
00044 bool ven::DescriptorWriter::build(VkDescriptorSet &set)
00045 {
00046     if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
00047         return false;
00048     }
00049     overwrite(set);
00050     return true;
00051 }
00052
00053 void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set) {
00054     for (auto &[sType, pNext, dstSet, dstBinding, dstArrayElement, descriptorCount, descriptorType,
pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
00055         dstSet = set;
00056     }
00057     vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
m_writes.data(), 0, nullptr);
00058 }

```

8.70 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

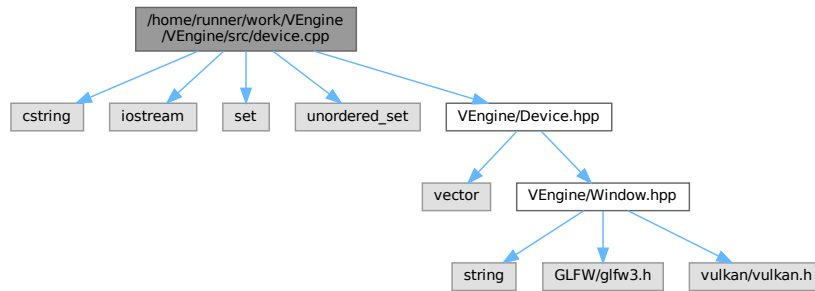
```

#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>

```

```
#include "VEngine/Device.hpp"
```

Include dependency graph for device.cpp:



Functions

- static VKAPI_ATTR VkBool32 VKAPI_CALL [debugCallback](#) (const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
- VkResult [CreateDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator, VkDebugUtilsMessengerEXT *pDebugMessenger)
- void [DestroyDebugUtilsMessengerEXT](#) (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks *pAllocator)

8.70.1 Function Documentation

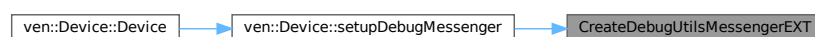
8.70.1.1 CreateDebugUtilsMessengerEXT()

```
VkResult CreateDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerCreateInfoEXT * pCreateInfo,
    const VkAllocationCallbacks * pAllocator,
    VkDebugUtilsMessengerEXT * pDebugMessenger)
```

Definition at line 16 of file [device.cpp](#).

Referenced by [ven::Device::setupDebugMessenger\(\)](#).

Here is the caller graph for this function:



8.70.1.2 debugCallback()

```
static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (
    const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity,
    const VkDebugUtilsMessageTypeFlagsEXT messageType,
    const VkDebugUtilsMessengerCallbackDataEXT * pCallbackData,
    void * pUserData) [static]
```

Definition at line 8 of file [device.cpp](#).

Referenced by [ven::Device::populateDebugMessengerCreateInfo\(\)](#).

Here is the caller graph for this function:



8.70.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerEXT debugMessenger,
    const VkAllocationCallbacks * pAllocator)
```

Definition at line 25 of file [device.cpp](#).

Referenced by [ven::Device::~~Device\(\)](#).

Here is the caller graph for this function:



8.71 device.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cstring>
00002 #include <iostream>
00003 #include <set>
00004 #include <unordered_set>
00005
00006 #include "VEngine/Device.hpp"
00007
00008 static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT
messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const
VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
00009 {
00010     (void) pUserData; (void) messageSeverity; (void) messageType;
00011
00012     std::cerr << "validation layer: " << pCallbackData->pMessage << '\n';
00013     return VK_FALSE;
00014 }
00015
00016 VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const
VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator,
VkDebugUtilsMessengerEXT *pDebugMessenger)
00017 {
00018     if (const auto func =
reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkCreateDebugUtilsMessengerEXT")); func != nullptr) {
00019         return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00020     }
00021
00022     return VK_ERROR_EXTENSION_NOT_PRESENT;
00023 }
00024
00025 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
debugMessenger, const VkAllocationCallbacks *pAllocator)
00026 {
00027     if (const auto func =
reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
"vkDestroyDebugUtilsMessengerEXT")); func != nullptr) {
00028         func(instance, debugMessenger, pAllocator);
00029     }
00030 }
00031
00032 ven::Device::Device(Window &window) : m_window{window}
00033 {
00034     createInstance();
00035     setupDebugMessenger();
00036     createSurface();
00037     pickPhysicalDevice();
00038     createLogicalDevice();
00039     createCommandPool();
00040 }
00041
00042 ven::Device::~Device()
00043 {
00044     vkDestroyCommandPool(m_device, m_commandPool, nullptr);
00045     vkDestroyDevice(m_device, nullptr);
00046
00047     if (enableValidationLayers) {
00048         DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
00049     }
00050
00051     vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
00052     vkDestroyInstance(m_instance, nullptr);
00053 }
00054
00055 void ven::Device::createInstance()
00056 {
00057     if (enableValidationLayers && !checkValidationLayerSupport()) {
00058         throw std::runtime_error("validation layers requested, but not available!");
00059     }
00060
00061     VkApplicationInfo appInfo = {};
00062     appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
00063     appInfo.pApplicationName = "LittleVulkanEngine App";
00064     appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
00065     appInfo.pEngineName = "No Engine";
00066     appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
00067     appInfo.apiVersion = VK_API_VERSION_1_0;
00068
00069     VkInstanceCreateInfo createInfo = {};
00070     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00071     createInfo.pApplicationInfo = &appInfo;
00072
00073     std::vector<const char *> extensions = getRequiredExtensions();

```

```

00074     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00075     createInfo.ppEnabledExtensionNames = extensions.data();
00076
00077     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00078     if (enableValidationLayers) {
00079         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00080         createInfo.ppEnabledLayerNames = validationLayers.data();
00081
00082         populateDebugMessengerCreateInfo(debugCreateInfo);
00083         createInfo.pNext = &debugCreateInfo;
00084     } else {
00085         createInfo.enabledLayerCount = 0;
00086         createInfo.pNext = nullptr;
00087     }
00088
00089     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
00090         throw std::runtime_error("failed to create instance!");
00091     }
00092
00093     hasGlfwRequiredInstanceExtensions();
00094 }
00095
00096 void ven::Device::pickPhysicalDevice()
00097 {
00098     uint32_t deviceCount = 0;
00099     vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
00100     if (deviceCount == 0) {
00101         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00102     }
00103     std::cout << "Device count: " << deviceCount << '\n';
00104     std::vector<VkPhysicalDevice> devices(deviceCount);
00105     vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());
00106
00107     for (const auto &device : devices) {
00108         if (isDeviceSuitable(device)) {
00109             m_physicalDevice = device;
00110             break;
00111         }
00112     }
00113
00114     if (m_physicalDevice == VK_NULL_HANDLE) {
00115         throw std::runtime_error("failed to find a suitable GPU!");
00116     }
00117
00118     vkGetPhysicalDeviceProperties(m_physicalDevice, &m_properties);
00119     std::cout << "physical device: " << m_properties.deviceName << '\n';
00120 }
00121
00122 void ven::Device::createLogicalDevice()
00123 {
00124     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
        findQueueFamilies(m_physicalDevice);
00125
00126     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00127     const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, presentFamily};
00128     float queuePriority = 1.0F;
00129
00130     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00131         VkDeviceQueueCreateInfo queueCreateInfo = {};
00132         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00133         queueCreateInfo.queueFamilyIndex = queueFamily;
00134         queueCreateInfo.queueCount = 1;
00135         queueCreateInfo.pQueuePriorities = &queuePriority;
00136         queueCreateInfos.push_back(queueCreateInfo);
00137     }
00138
00139     VkPhysicalDeviceFeatures deviceFeatures = {};
00140     deviceFeatures.samplerAnisotropy = VK_TRUE;
00141
00142     VkDeviceCreateInfo createInfo = {};
00143     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00144
00145     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00146     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00147
00148     createInfo.pEnabledFeatures = &deviceFeatures;
00149     createInfo.enabledExtensionCount = static_cast<uint32_t>(deviceExtensions.size());
00150     createInfo.ppEnabledExtensionNames = deviceExtensions.data();
00151
00152     // might not really be necessary anymore because device specific validation layers
00153     // have been deprecated
00154     if (enableValidationLayers) {
00155         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00156         createInfo.ppEnabledLayerNames = validationLayers.data();
00157     } else {
00158         createInfo.enabledLayerCount = 0;
00159     }

```



```

00160
00161     if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
00162         throw std::runtime_error("failed to create logical device!");
00163     }
00164
00165     vkGetDeviceQueue(m_device, graphicsFamily, 0, &m_graphicsQueue);
00166     vkGetDeviceQueue(m_device, presentFamily, 0, &m_presentQueue);
00167 }
00168
00169 void ven::Device::createCommandPool()
00170 {
00171     const QueueFamilyIndices queueFamilyIndices = findPhysicalQueueFamilies();
00172
00173     VkCommandPoolCreateInfo poolInfo = {};
00174     poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
00175     poolInfo.queueFamilyIndex = queueFamilyIndices.graphicsFamily;
00176     poolInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
00177
00178     if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
00179         throw std::runtime_error("failed to create command pool!");
00180     }
00181 }
00182
00183 bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00184 {
00185     const QueueFamilyIndices indices = findQueueFamilies(device);
00186     const bool extensionsSupported = checkDeviceExtensionSupport(device);
00187     bool swapChainAdequate = false;
00188
00189     if (extensionsSupported) {
00190         auto [capabilities, formats, presentModes] = querySwapChainSupport(device);
00191         swapChainAdequate = !formats.empty() && !presentModes.empty();
00192     }
00193
00194     VkPhysicalDeviceFeatures supportedFeatures;
00195     vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00196
00197     return indices.isComplete() && extensionsSupported && swapChainAdequate &&
(supportedFeatures.samplerAnisotropy != 0U);
00198 }
00199
00200 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00201 {
00202     createInfo = {};
00203     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00204     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00205     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00206     createInfo.pfnUserCallback = debugCallback;
00207     createInfo.pUserData = nullptr; // Optional
00208 }
00209
00210 void ven::Device::setupDebugMessenger()
00211 {
00212     if (!enableValidationLayers) { return; }
00213     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00214     populateDebugMessengerCreateInfo(createInfo);
00215     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
VK_SUCCESS) {
00216         throw std::runtime_error("failed to set up debug messenger!");
00217     }
00218 }
00219
00220 bool ven::Device::checkValidationLayerSupport() const
00221 {
00222     uint32_t layerCount = 0;
00223     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00224
00225     std::vector<VkLayerProperties> availableLayers(layerCount);
00226     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00227
00228     for (const char *layerName : validationLayers) {
00229         bool layerFound = false;
00230
00231         for (const auto &[layerName, specVersion, implementationVersion, description] :
availableLayers) {
00232             if (strcmp(layerName, layerName) == 0) {
00233                 layerFound = true;
00234                 break;
00235             }
00236         }
00237         if (!layerFound) {
00238             return false;
00239         }
00240     }
00241     return true;
00242 }

```

```

00243     }
00244
00245     return true;
00246 }
00247
00248 std::vector<const char *> ven::Device::getRequiredExtensions() const
00249 {
00250     uint32_t glfwExtensionCount = 0;
00251     const char **glfwExtensions = nullptr;
00252     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00253
00254     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00255
00256     if (enableValidationLayers) {
00257         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00258     }
00259
00260     return extensions;
00261 }
00262
00263 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00264 {
00265     uint32_t extensionCount = 0;
00266     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
00267     std::vector<VkExtensionProperties> extensions(extensionCount);
00268     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00269
00270     std::cout << "available extensions:\n";
00271     std::unordered_set<std::string> available;
00272     for (const auto &[extensionName, specVersion] : extensions) {
00273         std::cout << '\t' << extensionName << '\n';
00274         available.insert(extensionName);
00275     }
00276
00277     std::cout << "required extensions:\n";
00278     const std::vector<const char *> requiredExtensions = getRequiredExtensions();
00279     for (const auto &required : requiredExtensions) {
00280         std::cout << "\t" << required << '\n';
00281         if (!available.contains(required)) {
00282             throw std::runtime_error("Missing required glfw extension");
00283         }
00284     }
00285 }
00286
00287 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00288 {
00289     uint32_t extensionCount = 0;
00290     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00291
00292     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00293     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
    availableExtensions.data());
00294
00295     std::set<std::string> requiredExtensions(deviceExtensions.begin(), deviceExtensions.end());
00296     for (const auto &[extensionName, specVersion] : availableExtensions) {
00297         requiredExtensions.erase(extensionName);
00298     }
00299
00300     return requiredExtensions.empty();
00301 }
00302
00303 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00304 {
00305     QueueFamilyIndices indices;
00306
00307     uint32_t queueFamilyCount = 0;
00308     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00309     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00310     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00311     uint32_t index = 0;
00312
00313     for (const auto &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
    queueFamilies) {
00314         if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00315             indices.graphicsFamily = index;
00316             indices.graphicsFamilyHasValue = true;
00317         }
00318         VkBool32 presentSupport = 0U;
00319         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00320         if (queueCount > 0 && (presentSupport != 0U)) {
00321             indices.presentFamily = index;
00322             indices.presentFamilyHasValue = true;
00323         }
00324         if (indices.isComplete()) {
00325             break;
00326         }
00327         index++;
    }

```

```

00328     }
00329     return indices;
00330 }
00331
00332 ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00333 {
00334     SwapChainSupportDetails details;
00335     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
00336     uint32_t formatCount = 0;
00337
00338     vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
00339     if (formatCount != 0) {
00340         details.formats.resize(formatCount);
00341         vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
00342     }
00343     uint32_t presentModeCount = 0;
00344     vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
00345     if (presentModeCount != 0) {
00346         details.presentModes.resize(presentModeCount);
00347         vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount,
00348         details.presentModes.data());
00349     }
00350     return details;
00351 }
00352
00353 VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
00354 tiling, const VkFormatFeatureFlags features) const
00355 {
00356     for (const VkFormat format : candidates) {
00357         VkFormatProperties props;
00358         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00359         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00360             return format;
00361         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
00362         features) {
00363             return format;
00364         }
00365     }
00366     throw std::runtime_error("failed to find supported format!");
00367 }
00368
00369 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
00370 properties) const
00371 {
00372     VkPhysicalDeviceMemoryProperties memProperties;
00373     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
00374
00375     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00376         if (((typeFilter & (1 << i)) != 0U) &&
00377             (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00378             return i;
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
00385 VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00386 {
00387     VkBufferCreateInfo bufferInfo{};
00388     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00389     bufferInfo.size = size;
00390     bufferInfo.usage = usage;
00391     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00392
00393     if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00394         throw std::runtime_error("failed to create vertex m_buffer!");
00395     }
00396
00397     VkMemoryRequirements memRequirements;
00398     vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);
00399
00400     VkMemoryAllocateInfo allocInfo{};
00401     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00402     allocInfo.allocationSize = memRequirements.size;
00403     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00404
00405     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00406         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00407     }
00408
00409     vkBindBufferMemory(m_device, buffer, bufferMemory, 0);
00410 }
00411
00412 VkCommandBuffer ven::Device::beginSingleTimeCommands() const

```

```

00410 {
00411     VkCommandBufferAllocateInfo allocInfo{};
00412     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00413     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00414     allocInfo.commandPool = m_commandPool;
00415     allocInfo.commandBufferCount = 1;
00416
00417     VkCommandBuffer commandBuffer = nullptr;
00418     vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);
00419
00420     VkCommandBufferBeginInfo beginInfo{};
00421     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00422     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00423
00424     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00425     return commandBuffer;
00426 }
00427
00428 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00429 {
00430     vkEndCommandBuffer(commandBuffer);
00431
00432     VkSubmitInfo submitInfo{};
00433     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00434     submitInfo.commandBufferCount = 1;
00435     submitInfo.pCommandBuffers = &commandBuffer;
00436
00437     vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
00438     vkQueueWaitIdle(m_graphicsQueue);
00439
00440     vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
00441 }
00442
00443 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
size) const
00444 {
00445     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00446
00447     VkBufferCopy copyRegion{};
00448     copyRegion.srcOffset = 0; // Optional
00449     copyRegion.dstOffset = 0; // Optional
00450     copyRegion.size = size;
00451     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00452
00453     endSingleTimeCommands(commandBuffer);
00454 }
00455
00456 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
const uint32_t height, const uint32_t layerCount) const
00457 {
00458     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00459
00460     VkBufferImageCopy region{};
00461     region.bufferOffset = 0;
00462     region.bufferRowLength = 0;
00463     region.bufferImageHeight = 0;
00464
00465     region.imageSubresource.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00466     region.imageSubresource.mipLevel = 0;
00467     region.imageSubresource.baseArrayLayer = 0;
00468     region.imageSubresource.layerCount = layerCount;
00469
00470     region.imageOffset = {.x=0, .y=0, .z=0};
00471     region.imageExtent = {.width=width, .height=height, .depth=1};
00472
00473     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
&region);
00474     endSingleTimeCommands(commandBuffer);
00475 }
00476
00477 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const
00478 {
00479     if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00480         throw std::runtime_error("failed to create image!");
00481     }
00482
00483     VkMemoryRequirements memRequirements;
00484     vkGetImageMemoryRequirements(m_device, image, &memRequirements);
00485
00486     VkMemoryAllocateInfo allocInfo{};
00487     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00488     allocInfo.allocationSize = memRequirements.size;
00489     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00490
00491     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00492         throw std::runtime_error("failed to allocate image memory!");

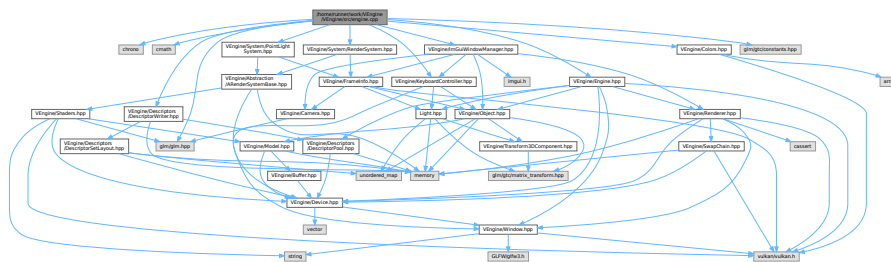
```

```
00493     }
00494
00495     if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
00496         throw std::runtime_error("failed to bind image m_memory!");
00497     }
00498 }
```

8.72 /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:



8.73 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     constexpr VkApplicationInfo appInfo{ .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO, .pNext =
nullptr, .pApplicationName = "VEngine App", .applicationVersion = VK_MAKE_API_VERSION(0, 1, 0, 0),
.pEngineName = "VEngine", .engineVersion = VK_MAKE_API_VERSION(0, 1, 0, 0), .apiVersion =
VK_API_VERSION_1_0 };
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/quad.obj");
00046
00047     Object floor = Object::createObject();
00048     floor.setName("floor");
00049     floor.setModel(model);
00050     floor.transform3D.translation = {0.F, .5F, 0.F};
00051     floor.transform3D.scale = {3.F, 1.F, 3.F};
00052     m_objects.emplace(floor.getId(), std::move(floor));
00053
00054     model = Model::createModelFromFile(m_device, "models/flat_vase.obj");
00055     Object flatVase = Object::createObject();
00056     flatVase.setName("flat vase");
00057     flatVase.setModel(model);
00058     flatVase.transform3D.translation = {-.5F, .5F, 0.F};
00059     flatVase.transform3D.scale = {3.F, 1.5F, 3.F};
00060     m_objects.emplace(flatVase.getId(), std::move(flatVase));
00061
00062     model = Model::createModelFromFile(m_device, "models/smooth_vase.obj");
00063     Object smoothVase = Object::createObject();
00064     smoothVase.setName("smooth vase");
00065     smoothVase.setModel(model);
00066     smoothVase.transform3D.translation = {.5F, .5F, 0.F};
00067     smoothVase.transform3D.scale = {3.F, 1.5F, 3.F};
00068     m_objects.emplace(smoothVase.getId(), std::move(smoothVase));
00069
00070     const std::vector<glm::vec4> lightColors{
00071         {Colors::RED, DEFAULT_LIGHT_INTENSITY},
00072         {Colors::GREEN, DEFAULT_LIGHT_INTENSITY},
00073         {Colors::BLUE, DEFAULT_LIGHT_INTENSITY},
00074         {Colors::YELLOW, DEFAULT_LIGHT_INTENSITY},
00075         {Colors::CYAN, DEFAULT_LIGHT_INTENSITY},
00076         {Colors::MAGENTA, DEFAULT_LIGHT_INTENSITY}
00077     };
00078
00079     for (std::size_t i = 0; i < lightColors.size(); i++)
00080     {
00081         Light pointLight = Light::createLight();
00082         pointLight.color = lightColors[i];
00083         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});
00084         pointLight.transform3D.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F,
1.F));
00085         m_lights.emplace(pointLight.getId(), std::move(pointLight));
00086     }
00087 }
00088
00089 void ven::Engine::mainLoop()
00090 {
00091     GlobalUbo ubo{};
00092     Camera camera{};
00093     KeyboardController cameraController{};
00094     std::chrono::duration<float> deltaTime{};
00095     VkCommandBuffer_T *commandBuffer = nullptr;
00096     bool showDebugWindow = true;
00097     float frameTime = NAN;
00098     int frameIndex = 0;
00099     Object viewerObject = Object::createObject();
00100     std::chrono::time_point<std::chrono::system_clock> newTime;
00101     std::chrono::time_point<std::chrono::system_clock> currentTime =
std::chrono::high_resolution_clock::now();
00102     std::unique_ptr<DescriptorSetLayout> globalSetLayout =
DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
VK_SHADER_STAGE_ALL_GRAPHICS).build();
00103     std::vector<std::unique_ptr<Buffer>> uboBuffers (SwapChain::MAX_FRAMES_IN_FLIGHT);

```

```

00104     std::vector<VkDescriptorSet> globalDescriptorSets(SwapChain::MAX_FRAMES_IN_FLIGHT);
00105     RenderSystem renderSystem(m_device, m_renderer.getSwapChainRenderPass(),
00106                               globalSetLayout->getDescriptorSetLayout());
00106     PointLightSystem pointLightSystem(m_device, m_renderer.getSwapChainRenderPass(),
00107                                       globalSetLayout->getDescriptorSetLayout());
00107     ImGuiIO &io = ImGui::GetIO();
00108     VkDescriptorBufferInfo bufferInfo{};
00109
00110     for (auto& uboBuffer : uboBuffers)
00111     {
00112         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
00113       VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00113         uboBuffer->map();
00114     }
00115     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00116         bufferInfo = uboBuffers[i]->descriptorInfo();
00117         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
00118       &bufferInfo).build(globalDescriptorSets[i]);
00118     }
00119     camera.setViewTarget({-1.F, -2.F, -2.F}, {0.F, 0.F, 2.5F});
00120     viewerObject.transform3D.translation.z = DEFAULT_POSITION[2];
00121
00122     m_renderer.setClearColor();
00123
00124     while (glfwWindowShouldClose(m_window.getGLFWWindow()) == 0)
00125     {
00126         glfwPollEvents();
00127
00128         newTime = std::chrono::high_resolution_clock::now();
00129         deltaTime = newTime - currentTime;
00130         currentTime = newTime;
00131         frameTime = deltaTime.count();
00132         commandBuffer = m_renderer.beginFrame();
00133
00134         cameraController.moveInPlaneXZ(m_window.getGLFWWindow(), frameTime, viewerObject,
00135       &showDebugWindow);
00135         camera.setViewYXZ(viewerObject.transform3D.translation, viewerObject.transform3D.rotation);
00136         camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00137
00138         if (commandBuffer != nullptr) {
00139             frameIndex = m_renderer.getFrameIndex();
00140             FrameInfo frameInfo{.frameIndex=frameIndex, .frameTime=frameTime,
00141           .commandBuffer=commandBuffer, .camera=camera,
00142           .globalDescriptorSet=globalDescriptorSets[static_cast<unsigned long>(frameIndex)], .objects=m_objects,
00143           .lights=m_lights};
00141             ubo.projection = camera.getProjection();
00142             ubo.view = camera.getView();
00143             ubo.inverseView = camera.getInverseView();
00144             PointLightSystem::update(frameInfo, ubo);
00145             uboBuffers[static_cast<unsigned long>(frameIndex)]->writeToBuffer(&ubo);
00146             uboBuffers[static_cast<unsigned long>(frameIndex)]->flush();
00147
00148             m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00149             renderSystem.renderObjects(frameInfo);
00150             pointLightSystem.render(frameInfo);
00151
00152             if (showDebugWindow) { ImGuiWindowManager::render(&m_renderer, m_objects, m_lights, io,
00153           viewerObject, camera, cameraController, m_device.getPhysicalDevice(), ubo); }
00153
00154             m_renderer.endSwapChainRenderPass(commandBuffer);
00155             m_renderer.endFrame();
00156             commandBuffer = nullptr;
00157         }
00158     }
00159     ImGuiWindowManager::cleanup();
00160     vkDeviceWaitIdle(m_device.device());
00161 }

```

8.74 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference

```

#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"

```



```

00027         { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
00028     });
00029     const VkDescriptorPoolCreateInfo pool_info = {
00030         VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00031         nullptr,
00032         VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00033         DESCRIPTOR_COUNT,
00034         std::size(pool_sizes),
00035         pool_sizes.data()
00036     };
00037
00038     if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00039         throw std::runtime_error("Failed to create ImGui descriptor pool");
00040     }
00041     ImGui_ImplVulkan_InitInfo init_info = {
00042         .Instance = instance,
00043         .PhysicalDevice = device->getPhysicalDevice(),
00044         .Device = device->device(),
00045         .Queue = device->graphicsQueue(),
00046         .DescriptorPool = pool,
00047         .MinImageCount = 3,
00048         .ImageCount = 3,
00049         .MSAASamples = VK_SAMPLE_COUNT_1_BIT
00050     };
00051
00052     ImGui_ImplGlfw_InitForVulkan(window, true);
00053     ImGui_ImplVulkan_Init(&init_info, renderPass);
00054     initStyle();
00055 }
00056
00057 void ven::ImGuiWindowManager::initStyle()
00058 {
00059     ImGuiStyle& style = ImGui::GetStyle();
00060     style.Alpha = 1.0;
00061     style.WindowRounding = 3;
00062     style.GrabRounding = 1;
00063     style.GrabMinSize = 20;
00064     style.FrameRounding = 3;
00065
00066     style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00067     style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
00068     style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
00069     style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
00070     style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
00071     style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
00072     style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
00073     style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
00074     style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
00075     style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
00076     style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
00077     style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
00078     style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
00079     style.Colors[ImGuiCol_ScrollbarGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.44F);
00080     style.Colors[ImGuiCol_ScrollbarGrabHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00081     style.Colors[ImGuiCol_ScrollbarGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00082     style.Colors[ImGuiCol_CheckMark] = ImVec4(0.00F, 1.00F, 1.00F, 0.68F);
00083     style.Colors[ImGuiCol_SliderGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.36F);
00084     style.Colors[ImGuiCol_SliderGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.76F);
00085     style.Colors[ImGuiCol_Button] = ImVec4(0.00F, 0.65F, 0.65F, 0.46F);
00086     style.Colors[ImGuiCol_ButtonHovered] = ImVec4(0.01F, 1.00F, 1.00F, 0.43F);
00087     style.Colors[ImGuiCol_ButtonActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.62F);
00088     style.Colors[ImGuiCol_Header] = ImVec4(0.00F, 1.00F, 1.00F, 0.33F);
00089     style.Colors[ImGuiCol_HeaderHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.42F);
00090     style.Colors[ImGuiCol_HeaderActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00091     style.Colors[ImGuiCol_ResizeGrip] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00092     style.Colors[ImGuiCol_ResizeGripHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00093     style.Colors[ImGuiCol_ResizeGripActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00094     style.Colors[ImGuiCol_PlotLines] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00095     style.Colors[ImGuiCol_PlotLinesHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00096     style.Colors[ImGuiCol_PlotHistogram] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00097     style.Colors[ImGuiCol_PlotHistogramHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00098     style.Colors[ImGuiCol_TextSelectedBg] = ImVec4(0.00F, 1.00F, 1.00F, 0.22F);
00099 }

```

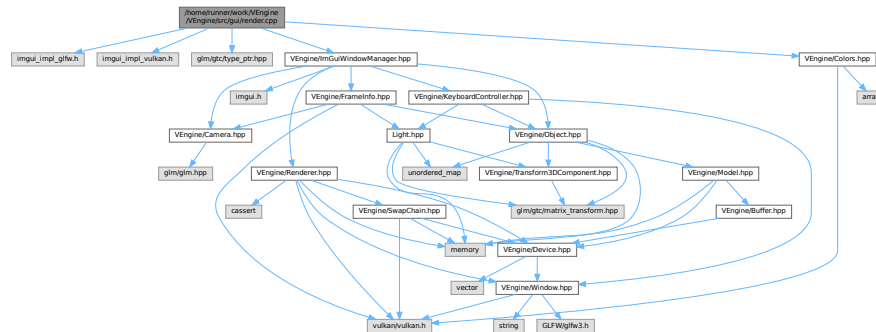
8.76 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference

```

#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>

```

```
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"
Include dependency graph for render.cpp:
```



8.77 render.cpp

[Go to the documentation of this file.](#)

```
00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include <glm/gtc/type_ptr.hpp>
00005
00006 #include "VEngine/ImGuiWindowManager.hpp"
00007 #include "VEngine/Colors.hpp"
00008
00009
00010 void ven::ImGuiWindowManager::cleanup()
00011 {
00012     ImGui_ImplVulkan_Shutdown();
00013     ImGui_ImplGlfw_Shutdown();
00014     ImGui::DestroyContext();
00015 }
00016
00017 void ven::ImGuiWindowManager::render(Renderer* renderer, std::unordered_map<unsigned int, Object>&
objects, std::unordered_map<unsigned int, Light>& lights, const ImGuiIO& io, Object& cameraObj,
Camera& camera, KeyboardController& cameraController, VkPhysicalDevice physicalDevice, GlobalUbo& ubo)
00018 {
00019     VkPhysicalDeviceProperties deviceProperties;
00020     vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
00021
00022     ImGui_ImplVulkan_NewFrame();
00023     ImGui_ImplGlfw_NewFrame();
00024     ImGui::NewFrame();
00025
00026     renderFrameWindow(io);
00027
00028     ImGui::Begin("Debug Window");
00029     rendererSection(renderer, ubo);
00030     cameraSection(cameraObj, camera, cameraController);
00031     objectsSection(objects);
00032     lightsSection(lights);
00033     inputsSection(io);
00034     devicePropertiesSection(deviceProperties);
00035
00036     ImGui::End();
00037     ImGui::Render();
00038     ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
00039 }
00040
00041 void ven::ImGuiWindowManager::renderFrameWindow(const ImGuiIO& io)
00042 {
00043     const float framerate = io.Framerate;
00044
00045     ImGui::SetNextWindowPos(ImVec2(0.0F, 0.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
00046     ImGui::Begin("Application Info", nullptr, ImGuiWindowFlags_NoDecoration | ImGuiWindowFlags_NoMove
| ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoSavedSettings | ImGuiWindowFlags_NoFocusOnAppearing |
ImGuiWindowFlags_NoNav);
00047     ImGui::Text("FPS: %.1f", framerate);
```

```

00048     ImGui::Text("Frame time: %.3fms", 1000.0F / framerate);
00049     ImGui::End();
00050 }
00051
00052 void ven::ImGuiWindowManager::rendererSection(Renderer *renderer, GlobalUbo& ubo)
00053 {
00054     if (ImGui::CollapsingHeader("Renderer")) {
00055         ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());
00056
00057         if (ImGui::BeginTable("ClearColorTable", 2)) {
00058             ImGui::TableNextColumn();
00059             std::array<float, 4> clearColor = renderer->getClearColor();
00060
00061             if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
00062                 VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1], clearColor[2],
clearColor[3]}};
00063                 renderer->setClearColorValue(clearColorValue);
00064             }
00065
00066             ImGui::TableNextColumn();
00067             static int item_current = 0;
00068
00069             if (ImGui::Combo("Color Presets##clearColor",
00070                             &item_current,
00071                             [](void*, int idx, const char** out_text) -> bool {
00072                                 if (idx < 0 || idx >=
static_cast<int>(std::size(Colors::CLEAR_COLORS))) { return false; }
00073                                 *out_text = Colors::CLEAR_COLORS.at(static_cast<unsigned
long>(idx)).first;
00074                                 return true;
00075                             },
00076                             nullptr,
00077                             std::size(Colors::CLEAR_COLORS))) {
00078                 renderer->setClearColorValue(Colors::CLEAR_COLORS.at(static_cast<unsigned
long>(item_current)).second);
00079             }
00080
00081             ImGui::TableNextColumn();
00082             ImGui::ColorEdit4("Ambient Light Color", glm::value_ptr(ubo.ambientLightColor));
00083             ImGui::TableNextColumn();
00084             if (ImGui::Combo("Color Presets##ambientColor",
00085                             &item_current,
00086                             [](void*, int idx, const char** out_text) -> bool {
00087                                 if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLORS))) {
return false; }
00088                                 *out_text = Colors::COLORS.at(static_cast<unsigned long>(idx)).first;
00089                                 return true;
00090                             },
00091                             nullptr,
00092                             std::size(Colors::COLORS))) {
00093                 ubo.ambientLightColor = glm::vec4(Colors::COLORS.at(static_cast<unsigned
long>(item_current)).second.r, Colors::COLORS.at(static_cast<unsigned long>(item_current)).second.g,
Colors::COLORS.at(static_cast<unsigned long>(item_current)).second.b, 1.0F);
00094             }
00095
00096             ImGui::TableNextColumn();
00097             ImGui::SliderFloat(("Intensity##" + std::to_string(0)).c_str(), &ubo.ambientLightColor.a,
00098                                0.0F, 1.0F);
00099             ImGui::TableNextColumn();
00100             if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a =
DEFAULT_AMBIENT_LIGHT_INTENSITY; }
00101
00102             ImGui::EndTable();
00103         }
00104
00105         static bool fullscreen = false;
00106         if (ImGui::Checkbox("Fullscreen", &fullscreen)) {
00107             renderer->getWindow().setFullscreen(fullscreen, renderer->getWindow().getExtent().width,
renderer->getWindow().getExtent().height);
00108         }
00109     }
00110 }
00111
00112 void ven::ImGuiWindowManager::cameraSection(Object &cameraObj, Camera &camera, KeyboardController
&cameraController)
00113 {
00114     if (ImGui::CollapsingHeader("Camera")) {
00115         float fov = camera.getFov();
00116         float near = camera.getNear();
00117         float far = camera.getFar();
00118         if (ImGui::BeginTable("CameraTable", 2)) {
00119             ImGui::TableNextColumn();
00120             ImGui::DragFloat3("Position", glm::value_ptr(cameraObj.transform3D.translation), 0.1F);
00121             ImGui::TableNextColumn();
00122             if (ImGui::Button("Reset##position")) { cameraObj.transform3D.translation =
DEFAULT_POSITION; }

```

```

00123
00124     ImGui::TableNextColumn();
00125     ImGui::DragFloat3("Rotation", glm::value_ptr(cameraObj.transform3D.rotation), 0.1F);
00126     ImGui::TableNextColumn();
00127     if (ImGui::Button("Reset##rotation")) { cameraObj.transform3D.rotation = DEFAULT_ROTATION;
    }
00128
00129     ImGui::TableNextColumn();
00130     if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
camera.setFov(fov); }
00131     ImGui::TableNextColumn();
00132     if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }
00133
00134     ImGui::TableNextColumn();
00135     if (ImGui::SliderFloat("Near", &near, 0.001F, 10.0F)) { camera.setNear(near); }
00136     ImGui::TableNextColumn();
00137     if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }
00138
00139     ImGui::TableNextColumn();
00140     if (ImGui::SliderFloat("Far", &far, 1.F, 1000.0F)) { camera.setFar(far); }
00141     ImGui::TableNextColumn();
00142     if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }
00143
00144     ImGui::TableNextColumn();
00145     ImGui::SliderFloat("Move Speed", &cameraController.m_moveSpeed, 0.1F, 10.0F);
00146     ImGui::TableNextColumn();
00147     if (ImGui::Button("Reset##moveSpeed")) { cameraController.m_moveSpeed =
DEFAULT_MOVE_SPEED; }
00148
00149     ImGui::TableNextColumn();
00150     ImGui::SliderFloat("Look Speed", &cameraController.m_lookSpeed, 0.1F, 10.0F);
00151     ImGui::TableNextColumn();
00152     if (ImGui::Button("Reset##lookSpeed")) { cameraController.m_lookSpeed =
DEFAULT_LOOK_SPEED; }
00153
00154     ImGui::EndTable();
00155 }
00156 }
00157 }
00158
00159 void ven::ImGuiWindowManager::objectsSection(std::unordered_map<unsigned int, Object>& objects)
00160 {
00161     if (ImGui::CollapsingHeader("Objects")) {
00162         ImVec4 color;
00163         bool open = false;
00164
00165         for (auto& [id, object] : objects) {
00166             if (object.color.r == 0.0F && object.color.g == 0.0F && object.color.b == 0.0F) {
00167                 color = { Colors::GRAY.r, Colors::GRAY.g, Colors::GRAY.b, 1.0F };
00168             } else {
00169                 color = { object.color.r, object.color.g, object.color.b, 1.0F };
00170             }
00171             ImGui::PushStyleColor(ImGuiCol_Text, color);
00172             open = ImGui::TreeNode(std::string(object.getName() + " [" +
std::to_string(object.getId()) + "]").c_str());
00173             ImGui::PopStyleColor(1);
00174             if (open) {
00175                 ImGui::DragFloat3(("Position##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.translation), 0.1F);
00176                 ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.rotation), 0.1F);
00177                 ImGui::DragFloat3(("Scale##" + object.getName()).c_str(),
glm::value_ptr(object.transform3D.scale), 0.1F);
00178                 ImGui::Text("Address: %p", &object);
00179                 ImGui::TreePop();
00180             }
00181         }
00182     }
00183 }
00184
00185 void ven::ImGuiWindowManager::lightsSection(std::unordered_map<unsigned int, Light> &lights)
00186 {
00187     if (ImGui::CollapsingHeader("Lights")) {
00188         bool open = false;
00189
00190         for (auto& [id, light] : lights) {
00191             ImVec4 color{light.color.r, light.color.g, light.color.b, 1.0F};
00192             ImGui::PushStyleColor(ImGuiCol_Text, color);
00193             open = ImGui::TreeNode(std::string(light.getName() + " [" + std::to_string(light.getId())
+ "]").c_str());
00194             ImGui::PopStyleColor(1);
00195             if (open) {
00196                 ImGui::Text("Address: %p", &light);
00197                 ImGui::DragFloat3(("Position##" + std::to_string(light.getId()).c_str(),
glm::value_ptr(light.transform3D.translation), 0.1F);
00198                 ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId()).c_str(),
glm::value_ptr(light.transform3D.rotation), 0.1F);

```

```

00199         ImGui::DragFloat3(("Scale###" + std::to_string(light.getId())).c_str(),
glm::value_ptr(light.transform3D.scale), 0.1F);
00200         if (ImGui::BeginTable("ColorTable", 2)) {
00201             ImGui::TableNextColumn(); ImGui::ColorEdit4(("Color###" +
std::to_string(light.getId())).c_str(), glm::value_ptr(light.color));
00202
00203             ImGui::TableNextColumn();
00204             static int item_current = 0;
00205             if (ImGui::Combo("Color Presets",
00206                             &item_current,
00207                             [](void*, const int idx, const char** out_text) -> bool {
00208                                 if (idx < 0 || idx >=
static_cast<int>(std::size(Colors::COLORS))) { return false; }
00209                                 *out_text = Colors::COLORS.at(static_cast<unsigned
long>(idx)).first;
00210                                 return true;
00211                             },
00212                             nullptr,
00213                             std::size(Colors::COLORS))) {
00214                 light.color = {Colors::COLORS.at(static_cast<unsigned
long>(item_current)).second, light.color.a};
00215             }
00216
00217             ImGui::TableNextColumn();
00218             ImGui::SliderFloat(("Intensity###" + std::to_string(light.getId())).c_str(),
&light.color.a, 0.0F, 5.F);
00219             ImGui::TableNextColumn();
00220             if (ImGui::Button(("Reset###" + std::to_string(light.getId())).c_str())) {
light.color.a = DEFAULT_LIGHT_INTENSITY; }
00221
00222             ImGui::EndTable();
00223         }
00224         ImGui::TreePop();
00225     }
00226 }
00227 }
00228 }
00229
00230 void ven::ImGuiWindowManager::inputsSection(const ImGuiIO &io)
00231 {
00232     if (ImGui::CollapsingHeader("Input")) {
00233         ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io.MousePos.x, io.MousePos.y) :
ImGui::Text("Mouse pos: <INVALID>");
00234         ImGui::Text("Mouse delta: (%g, %g)", io.MouseDelta.x, io.MouseDelta.y);
00235         ImGui::Text("Mouse down:");
00236         for (int i = 0; i < static_cast<int>(std::size(io.MouseDown)); i++) {
00237             if (ImGui::IsMouseDown(i)) {
00238                 ImGui::SameLine();
00239                 ImGui::Text("b%d (%.02f secs)", i, io.MouseDownDuration[i]);
00240             }
00241         }
00242         ImGui::Text("Mouse wheel: %.1f", io.MouseWheel);
00243         ImGui::Text("Keys down:");
00244         for (auto key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
static_cast<ImGuiKey>(key + 1)) {
00245             if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
00246             ImGui::SameLine();
00247             ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\\%s\\" : "\\%s\\ " %d",
ImGui::GetKeyName(key), key);
00248         }
00249     }
00250 }
00251
00252 void ven::ImGuiWindowManager::devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties)
00253 {
00254     if (ImGui::CollapsingHeader("Device Properties")) {
00255         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00256
00257             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00258             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion));
00259             ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00260             ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00261             ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00262             ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00263             ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00264             ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantsSize);
00265             ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00266             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00267             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",

```

```

deviceProperties.limits.maxImageDimension2D);
00268     ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00269     ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00270     ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00271     ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00272     ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00273     ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00274     ImGui::EndTable();
00275 }
00276 }
00277 }

```

8.78 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp

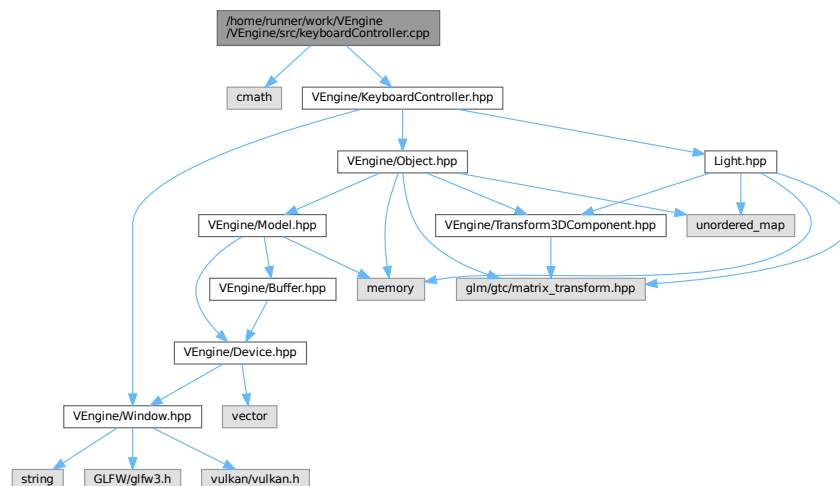
File Reference

```

#include <cmath>
#include "VEngine/KeyboardController.hpp"

```

Include dependency graph for keyboardController.cpp:



8.79 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, const 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);

```

```

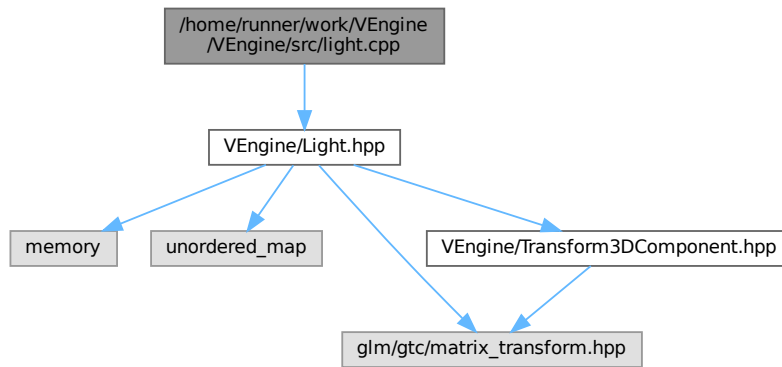
00015     }
00016
00017     object.transform3D.rotation.x = glm::clamp(object.transform3D.rotation.x, -1.5F, 1.5F);
00018     object.transform3D.rotation.y = glm::mod(object.transform3D.rotation.y, glm::two_pi<float>());
00019
00020     float yaw = object.transform3D.rotation.y;
00021     const glm::vec3 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 }
00042
00043 void ven::KeyboardController::moveInPlaneXZ(GLFWwindow *window, const float dt, Light &light, bool
    *showDebugWindow) const
00044 {
00045     glm::vec3 rotate{0};
00046     if (glfwGetKey(window, m_keys.lookLeft) == GLFW_PRESS) { rotate.y -= 1.F; }
00047     if (glfwGetKey(window, m_keys.lookRight) == GLFW_PRESS) { rotate.y += 1.F; }
00048     if (glfwGetKey(window, m_keys.lookUp) == GLFW_PRESS) { rotate.x += 1.F; }
00049     if (glfwGetKey(window, m_keys.lookDown) == GLFW_PRESS) { rotate.x -= 1.F; }
00050
00051     if (dot(rotate, rotate) > std::numeric_limits<float>::epsilon()) {
00052         light.transform3D.rotation += m_lookSpeed * dt * normalize(rotate);
00053     }
00054
00055     light.transform3D.rotation.x = glm::clamp(light.transform3D.rotation.x, -1.5F, 1.5F);
00056     light.transform3D.rotation.y = glm::mod(light.transform3D.rotation.y, glm::two_pi<float>());
00057
00058     float yaw = light.transform3D.rotation.y;
00059     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
00060     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00061     constexpr glm::vec3 upDir{0.F, -1.F, 0.F};
00062
00063     glm::vec3 moveDir{0.F};
00064     if (glfwGetKey(window, m_keys.moveForward) == GLFW_PRESS) {moveDir += forwardDir;}
00065     if (glfwGetKey(window, m_keys.moveBackward) == GLFW_PRESS) {moveDir -= forwardDir;}
00066     if (glfwGetKey(window, m_keys.moveRight) == GLFW_PRESS) {moveDir += rightDir;}
00067     if (glfwGetKey(window, m_keys.moveLeft) == GLFW_PRESS) {moveDir -= rightDir;}
00068     if (glfwGetKey(window, m_keys.moveUp) == GLFW_PRESS) {moveDir += upDir;}
00069     if (glfwGetKey(window, m_keys.moveDown) == GLFW_PRESS) {moveDir -= upDir;}
00070
00071     if (dot(moveDir, moveDir) > std::numeric_limits<float>::epsilon()) {
00072         light.transform3D.translation += m_moveSpeed * dt * normalize(moveDir);
00073     }
00074 }

```

8.80 /home/runner/work/VEngine/VEngine/src/light.cpp File Reference

```
#include "VEngine/Light.hpp"
```

Include dependency graph for light.cpp:



8.81 light.cpp

[Go to the documentation of this file.](#)

```

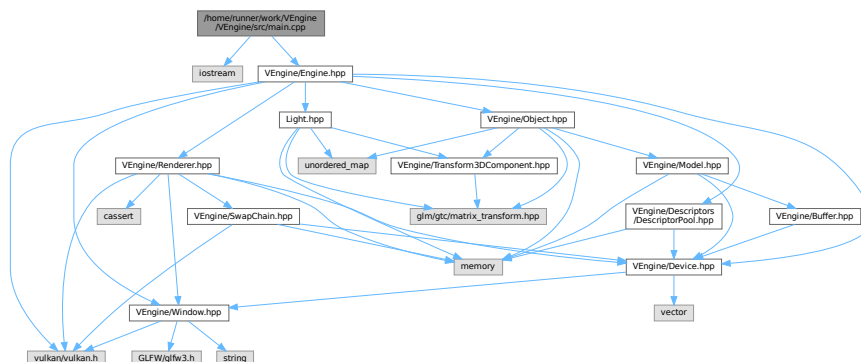
00001 #include "VEngine/Light.hpp"
00002
00003 ven::Light ven::Light::createLight(float radius, glm::vec4 color)
00004 {
00005     static unsigned int objId = 0;
00006     Light light(objId++);
00007
00008     light.color = color;
00009     light.transform3D.scale.x = radius;
00010
00011     return light;
00012 }
  
```

8.82 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```
#include <iostream>
```

```
#include "VEngine/Engine.hpp"
```

Include dependency graph for main.cpp:



Functions

- int [main](#) ()

8.82.1 Function Documentation

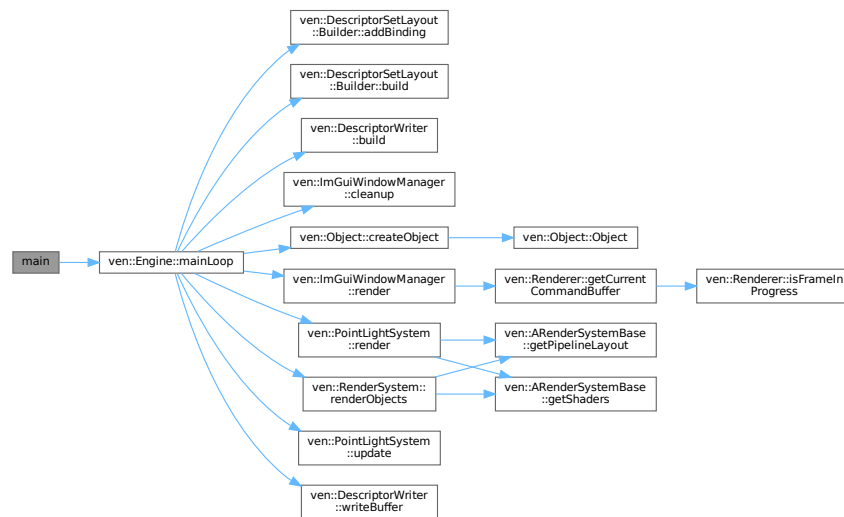
8.82.1.1 main()

```
int main ()
```

Definition at line 7 of file [main.cpp](#).

References [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



8.83 main.cpp

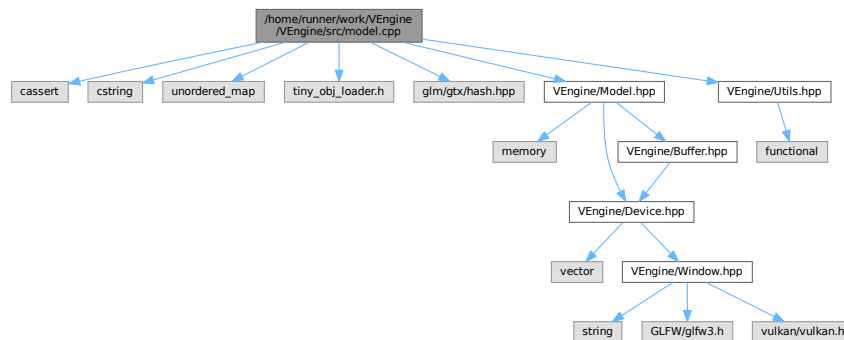
[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002
00003 #include "VEngine/Engine.hpp"
00004
00005 using namespace ven;
00006
00007 int main()
00008 {
00009     try {
00010         Engine engine{};
00011         engine.mainLoop();
00012     } catch (const std::exception &e) {
00013         std::cerr << "std exception: " << e.what() << '\n';
00014         return EXIT_FAILURE;
00015     } catch (...) {
00016         std::cerr << "Unknown error\n";
00017         return EXIT_SUCCESS;
00018     }
00019     return EXIT_SUCCESS;
00020 }
```

8.84 /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 >](#)

Macros

- [#define TINYOBJLOADER_IMPLEMENTATION](#)
- [#define GLM_ENABLE_EXPERIMENTAL](#)

8.84.1 Macro Definition Documentation

8.84.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 8 of file [model.cpp](#).

8.84.1.2 TINYOBJLOADER_IMPLEMENTATION

```
#define TINYOBJLOADER_IMPLEMENTATION
```

Definition at line 5 of file [model.cpp](#).

8.85 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 #define GLM_ENABLE_EXPERIMENTAL
00009 #include <glm/gtx/hash.hpp>
00010
00011 #include "VEngine/Model.hpp"
00012 #include "VEngine/Utils.hpp"
00013
00014 template<>
00015 struct std::hash<ven::Model::Vertex> {
00016     size_t operator() (ven::Model::Vertex const &vertex) const noexcept {
00017         size_t seed = 0;
00018         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00019         return seed;
00020     }
00021 };
00022
00023 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
    m_indexCount(0)
00024 {
00025     createVertexBuffer(builder.vertices);
00026     createIndexBuffer(builder.indices);
00027 }
00028
00029 ven::Model::~Model() = default;
00030
00031 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00032 {
00033     m_vertexCount = static_cast<uint32_t>(vertices.size());
00034     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00035     const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
00036     uint32_t vertexSize = sizeof(vertices[0]);
00037
00038     Buffer stagingBuffer{m_device, vertexSize, m_vertexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00039
00040     stagingBuffer.map();
00041     stagingBuffer.writeToBuffer(vertices.data());
00042
00043     m_vertexBuffer = std::make_unique<Buffer>(m_device, vertexSize, m_vertexCount,
    VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00044
00045     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00046 }
00047
00048 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00049 {
00050     m_indexCount = static_cast<uint32_t>(indices.size());
00051     m_hasIndexBuffer = m_indexCount > 0;
00052
00053     if (!m_hasIndexBuffer) {
00054         return;
00055     }
00056
00057     uint32_t indexSize = sizeof(indices[0]);
00058
00059     Buffer stagingBuffer{m_device, indexSize, m_indexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00060
00061     stagingBuffer.map();
00062     stagingBuffer.writeToBuffer(indices.data());
00063
00064     m_indexBuffer = std::make_unique<Buffer>(m_device, indexSize, m_indexCount,
    VK_BUFFER_USAGE_INDEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00065
00066     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), sizeof(indices[0]) *
    m_indexCount);
00067 }
00068
00069 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00070 {
00071     if (m_hasIndexBuffer) {
00072         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00073     } else {
00074         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00075     }
00076 }

```

```

00075     }
00076 }
00077
00078 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00079 {
00080     const std::array buffers{m_vertexBuffer->getBuffer()};
00081     constexpr VkDeviceSize offsets[] = {0};
00082     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets);
00083
00084     if (m_hasIndexBuffer) {
00085         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00086     }
00087 }
00088
00089 std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
&filename)
00090 {
00091     Builder builder{};
00092     builder.loadModel(filename);
00093     return std::make_unique<Model>(device, builder);
00094 }
00095
00096 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00097 {
00098     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00099     bindingDescriptions[0].binding = 0;
00100     bindingDescriptions[0].stride = sizeof(Vertex);
00101     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00102     return bindingDescriptions;
00103 }
00104
00105 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00106 {
00107     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00108
00109     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00110     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00111     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00112     attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00113
00114     return attributeDescriptions;
00115 }
00116
00117 void ven::Model::Builder::loadModel(const std::string &filename)
00118 {
00119     tinyobj::attrib_t attrib;
00120     std::vector<tinyobj::shape_t> shapes;
00121     std::vector<tinyobj::material_t> materials;
00122     std::string warn;
00123     std::string err;
00124
00125     if (!LoadObj(&attrib, &shapes, &materials, &warn, &err, filename.c_str()))
00126     {
00127         throw std::runtime_error(warn + err);
00128     }
00129
00130     vertices.clear();
00131     indices.clear();
00132
00133     std::unordered_map<Vertex, uint32_t> uniqueVertices{};
00134     for (const auto &[name, mesh, lines, points] : shapes) {
00135         for (const auto &[vertex_index, normal_index, texcoord_index] : mesh.indices) {
00136             Vertex vertex{};
00137             if (vertex_index >= 0) {
00138                 vertex.position = {
00139                     attrib.vertices[3 * static_cast<size_t>(vertex_index) + 0],
00140                     attrib.vertices[3 * static_cast<size_t>(vertex_index) + 1],
00141                     attrib.vertices[3 * static_cast<size_t>(vertex_index) + 2]
00142                 };
00143
00144                 vertex.color = {
00145                     attrib.colors[3 * static_cast<size_t>(vertex_index) + 0],
00146                     attrib.colors[3 * static_cast<size_t>(vertex_index) + 1],
00147                     attrib.colors[3 * static_cast<size_t>(vertex_index) + 2]
00148                 };
00149             }
00150
00151             if (normal_index >= 0) {
00152                 vertex.normal = {
00153                     attrib.normals[3 * static_cast<size_t>(normal_index) + 0],
00154                     attrib.normals[3 * static_cast<size_t>(normal_index) + 1],
00155                     attrib.normals[3 * static_cast<size_t>(normal_index) + 2]
00156                 };
00157             }
00158
00159             if (texcoord_index >= 0) {
00160                 vertex.uv = {

```

```

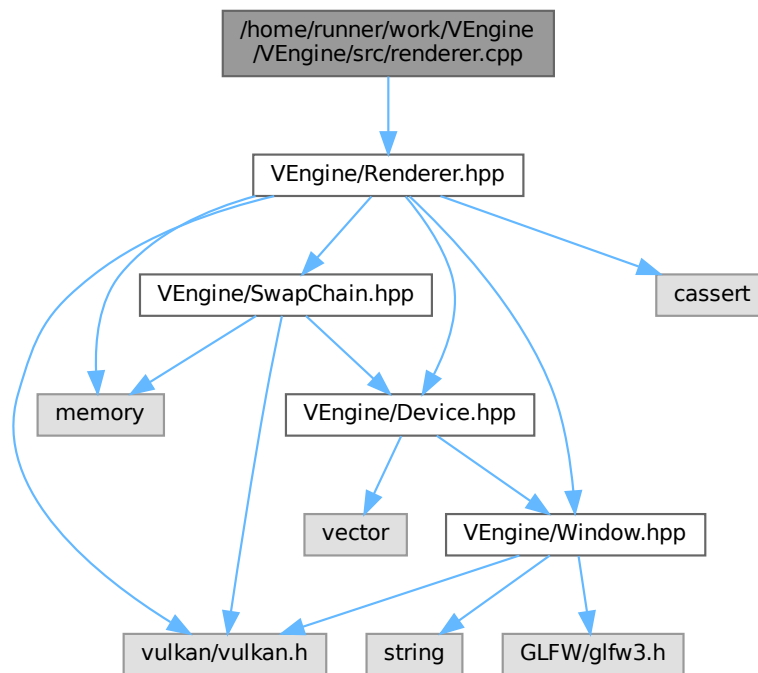
00161             attrib.texcoords[2 * static_cast<size_t>(texcoord_index) + 0],
00162             attrib.texcoords[2 * static_cast<size_t>(texcoord_index) + 1]
00163         };
00164     };
00165
00166     if (!uniqueVertices.contains(vertex)) {
00167         uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00168         vertices.push_back(vertex);
00169     }
00170     indices.push_back(uniqueVertices[vertex]);
00171 }
00172 }
00173 }

```

8.86 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

```
#include "VEngine/Renderer.hpp"
```

Include dependency graph for renderer.cpp:



8.87 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     if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer,
&m_currentImageIndex); result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
m_window.wasWindowResized()) {
00078         m_window.resetWindowResizedFlag();
00079         recreateSwapChain();
00080     }
00081     else if (result != VK_SUCCESS) {
00082         throw std::runtime_error("Failed to submit command buffer");
00083     }
00084
00085     m_isFrameStarted = false;
00086     m_currentFrameIndex = (m_currentFrameIndex + 1) % SwapChain::MAX_FRAMES_IN_FLIGHT;
00087 }
00088
00089 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00090 {
00091     assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");

```

```

00092     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
    from a different frame");
00093
00094     VkRenderPassBeginInfo renderPassInfo{};
00095     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00096     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00097     renderPassInfo.framebuffer = m_swapChain->getFramebuffer(m_currentImageIndex);
00098
00099     renderPassInfo.renderArea.offset = {.x=0, .y=0};
00100     renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();
00101
00102     renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
00103     renderPassInfo.pClearValues = m_clearValues.data();
00104
00105     vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00106
00107     VkViewport viewport{};
00108     viewport.x = 0.0F;
00109     viewport.y = 0.0F;
00110     viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00111     viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00112     viewport.minDepth = 0.0F;
00113     viewport.maxDepth = 1.0F;
00114     const VkRect2D scissor{0, 0, m_swapChain->getSwapChainExtent()};
00115     vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00116     vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00117 }
00118
00119 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00120 {
00121     assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
00122     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
    from a different frame");
00123
00124     vkCmdEndRenderPass(commandBuffer);
00125 }

```

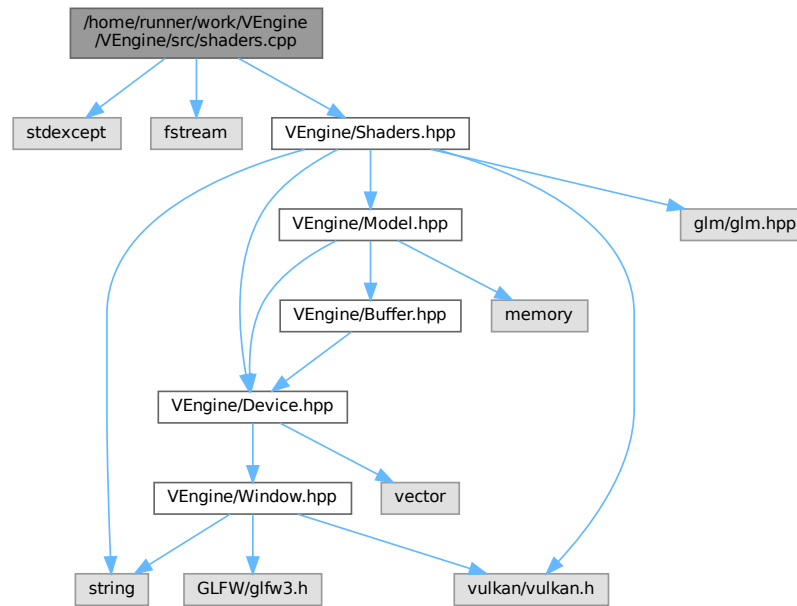
8.88 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```

#include <stdexcept>
#include <fstream>
#include "VEngine/Shaders.hpp"

```

Include dependency graph for shaders.cpp:



8.89 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     std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
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 =
00061         static_cast<uint32_t>(attributeDescriptions.size());
00062     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00063     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00064     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
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.data();
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     void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
00101     const
00102     {
00103         VkShaderModuleCreateInfo createInfo{};
00104         createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00105         createInfo.codeSize = code.size();
00106         createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00107
00108         if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00109             throw std::runtime_error("failed to create shader module");
00110         }
00111     }
00112
00113     void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00114     {
00115         configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00116         configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00117         configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00118
00119         configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00120         configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00121         configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00122         configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00123         configInfo.rasterizationInfo.lineWidth = 1.0F;

```

```

00123     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE;
00124     configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00125     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00126     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00127     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00128     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00129
00130     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00131     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00132     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00133     configInfo.multisampleInfo.minSampleShading = 1.0F;
00134     configInfo.multisampleInfo.pSampleMask = nullptr;
00135     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00136     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00137
00138     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00139     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00140     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00141     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00142     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00143     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00144     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00145     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00146
00147     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00148     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00149     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00150     configInfo.colorBlendInfo.attachmentCount = 1;
00151     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00152     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00153     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00154     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00155     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00156
00157     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00158     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00159     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00160     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00161     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00162     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00163     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00164     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00165     configInfo.depthStencilInfo.front = {};
00166     configInfo.depthStencilInfo.back = {};
00167
00168     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00169     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00170     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00171     configInfo.dynamicStateInfo.dynamicStateCount =
static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00172     configInfo.dynamicStateInfo.flags = 0;
00173     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();
00174     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00175 }

```

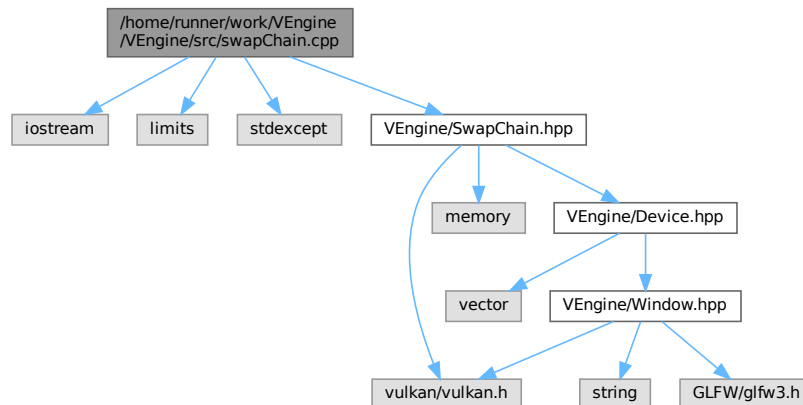
8.90 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference

```

#include <iostream>
#include <limits>
#include <stdexcept>
#include "VEngine/SwapChain.hpp"

```

Include dependency graph for swapChain.cpp:



8.91 swapChain.cpp

[Go to the documentation of this file.](#)

```

00001 #include <iostream>
00002 #include <limits>
00003 #include <stdexcept>
00004
00005 #include "VEngine/SwapChain.hpp"
00006
00007 ven::SwapChain::~SwapChain()
00008 {
00009     for (VkImageView_T *imageView : m_swapChainImageViews) {
00010         vkDestroyImageView(m_device.device(), imageView, nullptr);
00011     }
00012     m_swapChainImageViews.clear();
00013
00014     if (m_swapChain != nullptr) {
00015         vkDestroySwapchainKHR(m_device.device(), m_swapChain, nullptr);
00016         m_swapChain = nullptr;
00017     }
00018
00019     for (size_t i = 0; i < m_depthImages.size(); i++) {
00020         vkDestroyImageView(m_device.device(), m_depthImageViews[i], nullptr);
00021         vkDestroyImage(m_device.device(), m_depthImages[i], nullptr);
00022         vkFreeMemory(m_device.device(), m_depthImageMemory[i], nullptr);
00023     }
00024
00025     for (VkFramebuffer_T *framebuffer : m_swapChainFrameBuffers) {
00026         vkDestroyFramebuffer(m_device.device(), framebuffer, nullptr);
00027     }
00028
00029     vkDestroyRenderPass(m_device.device(), m_renderPass, nullptr);
00030
00031     // cleanup synchronization objects
00032     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00033         vkDestroySemaphore(m_device.device(), m_renderFinishedSemaphores[i], nullptr);
00034         vkDestroySemaphore(m_device.device(), m_imageAvailableSemaphores[i], nullptr);
00035         vkDestroyFence(m_device.device(), m_inFlightFences[i], nullptr);
00036     }
00037 }
00038
00039 void ven::SwapChain::init()
00040 {
00041     createSwapChain();
00042     createImageViews();
00043     createRenderPass();
00044     createDepthResources();
00045     createFrameBuffers();
00046     createSyncObjects();
00047 }
00048

```

```

00049 VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
00050 {
00051     vkWaitForFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame], VK_TRUE,
std::numeric_limits<uint64_t>::max());
00052
00053     return vkAcquireNextImageKHR(m_device.device(), m_swapChain, std::numeric_limits<uint64_t>::max(),
m_imageAvailableSemaphores[m_currentFrame], VK_NULL_HANDLE, imageIndex);
00054 }
00055
00056 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
*imageIndex)
00057 {
00058     if (m_imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00059         vkWaitForFences(m_device.device(), 1, &m_imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00060     }
00061     m_imagesInFlight[*imageIndex] = m_inFlightFences[m_currentFrame];
00062
00063     VkSubmitInfo submitInfo = {};
00064     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00065
00066     const std::array<VkSemaphore, 1> waitSemaphores = {m_imageAvailableSemaphores[m_currentFrame]};
00067     constexpr std::array<VkPipelineStageFlags, 1> waitStages =
{VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00068     submitInfo.waitSemaphoreCount = 1;
00069     submitInfo.pWaitSemaphores = waitSemaphores.data();
00070     submitInfo.pWaitDstStageMask = waitStages.data();
00071
00072     submitInfo.commandBufferCount = 1;
00073     submitInfo.pCommandBuffers = buffers;
00074
00075     const std::array<VkSemaphore, 1> signalSemaphores = {m_renderFinishedSemaphores[m_currentFrame]};
00076     submitInfo.signalSemaphoreCount = 1;
00077     submitInfo.pSignalSemaphores = signalSemaphores.data();
00078
00079     vkResetFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame]);
00080     if (vkQueueSubmit(m_device.graphicsQueue(), 1, &submitInfo, m_inFlightFences[m_currentFrame]) !=
VK_SUCCESS) {
00081         throw std::runtime_error("failed to submit draw command m_buffer!");
00082     }
00083
00084     VkPresentInfoKHR presentInfo = {};
00085     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00086
00087     presentInfo.waitSemaphoreCount = 1;
00088     presentInfo.pWaitSemaphores = signalSemaphores.data();
00089
00090     const std::array<VkSwapchainKHR, 1> swapChains = {m_swapChain};
00091     presentInfo.swapchainCount = 1;
00092     presentInfo.pSwapchains = swapChains.data();
00093
00094     presentInfo.pImageIndices = imageIndex;
00095
00096     const VkResult result = vkQueuePresentKHR(m_device.presentQueue(), &presentInfo);
00097
00098     m_currentFrame = (m_currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00099
00100     return result;
00101 }
00102
00103 void ven::SwapChain::createSwapChain()
00104 {
00105     const auto [capabilities, formats, presentModes] = m_device.getSwapChainSupport();
00106
00107     const auto [format, colorSpace] = chooseSwapSurfaceFormat(formats);
00108     const VkPresentModeKHR presentMode = chooseSwapPresentMode(presentModes);
00109     const VkExtent2D extent = chooseSwapExtent(capabilities);
00110
00111     uint32_t imageCount = capabilities.minImageCount + 1;
00112     if (capabilities.maxImageCount > 0 && imageCount > capabilities.maxImageCount) {
00113         imageCount = capabilities.maxImageCount;
00114     }
00115
00116     VkSwapchainCreateInfoKHR createInfo = {};
00117     createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
00118     createInfo.surface = m_device.surface();
00119
00120     createInfo.minImageCount = imageCount;
00121     createInfo.imageFormat = format;
00122     createInfo.imageColorSpace = colorSpace;
00123     createInfo.imageExtent = extent;
00124     createInfo.imageArrayLayers = 1;
00125     createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
00126
00127     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
m_device.findPhysicalQueueFamilies();
00128     const std::array<uint32_t, 2> queueFamilyIndices = {graphicsFamily, presentFamily};
00129

```

```

00130     if (graphicsFamily != presentFamily) {
00131         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00132         createInfo.queueFamilyIndexCount = 2;
00133         createInfo.pQueueFamilyIndices = queueFamilyIndices.data();
00134     } else {
00135         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00136         createInfo.queueFamilyIndexCount = 0; // Optional
00137         createInfo.pQueueFamilyIndices = nullptr; // Optional
00138     }
00139
00140     createInfo.preTransform = capabilities.currentTransform;
00141     createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
00142
00143     createInfo.presentMode = presentMode;
00144     createInfo.clipped = VK_TRUE;
00145
00146     createInfo.oldSwapchain = m_oldSwapChain == nullptr ? VK_NULL_HANDLE :
m_oldSwapChain->m_swapChain;
00147
00148     if (vkCreateSwapchainKHR(m_device.device(), &createInfo, nullptr, &m_swapChain) != VK_SUCCESS) {
00149         throw std::runtime_error("failed to create swap chain!");
00150     }
00151
00152     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, nullptr);
00153     m_swapChainImages.resize(imageCount);
00154     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, m_swapChainImages.data());
00155
00156     m_swapChainImageFormat = format;
00157     m_swapChainExtent = extent;
00158 }
00159
00160 void ven::SwapChain::createImageViews()
00161 {
00162     m_swapChainImageViews.resize(m_swapChainImages.size());
00163     for (size_t i = 0; i < m_swapChainImages.size(); i++) {
00164         VkImageViewCreateInfo viewInfo{};
00165         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00166         viewInfo.image = m_swapChainImages[i];
00167         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00168         viewInfo.format = m_swapChainImageFormat;
00169         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00170         viewInfo.subresourceRange.baseMipLevel = 0;
00171         viewInfo.subresourceRange.levelCount = 1;
00172         viewInfo.subresourceRange.baseArrayLayer = 0;
00173         viewInfo.subresourceRange.layerCount = 1;
00174
00175         if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_swapChainImageViews[i]) !=
VK_SUCCESS) {
00176             throw std::runtime_error("failed to create texture image view!");
00177         }
00178     }
00179 }
00180
00181 void ven::SwapChain::createRenderPass()
00182 {
00183     VkAttachmentDescription depthAttachment{};
00184     depthAttachment.format = findDepthFormat();
00185     depthAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00186     depthAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00187     depthAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00188     depthAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00189     depthAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00190     depthAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00191     depthAttachment.finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00192
00193     VkAttachmentReference depthAttachmentRef{};
00194     depthAttachmentRef.attachment = 1;
00195     depthAttachmentRef.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00196
00197     VkAttachmentDescription colorAttachment = {};
00198     colorAttachment.format = getSwapChainImageFormat();
00199     colorAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00200     colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00201     colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
00202     colorAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00203     colorAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00204     colorAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00205     colorAttachment.finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
00206
00207     VkAttachmentReference colorAttachmentRef = {};
00208     colorAttachmentRef.attachment = 0;
00209     colorAttachmentRef.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00210
00211     VkSubpassDescription subpass = {};
00212     subpass.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
00213     subpass.colorAttachmentCount = 1;
00214     subpass.pColorAttachments = &colorAttachmentRef;

```

```

00215     subpass.pDepthStencilAttachment = &depthAttachmentRef;
00216
00217     VkSubpassDependency dependency = {};
00218     dependency.srcSubpass = VK_SUBPASS_EXTERNAL;
00219     dependency.srcAccessMask = 0;
00220     dependency.srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00221     dependency.dstSubpass = 0;
00222     dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00223     dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00224
00225     const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
00226     VkRenderPassCreateInfo renderPassInfo = {};
00227     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
00228     renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00229     renderPassInfo.pAttachments = attachments.data();
00230     renderPassInfo.subpassCount = 1;
00231     renderPassInfo.pSubpasses = &subpass;
00232     renderPassInfo.dependencyCount = 1;
00233     renderPassInfo.pDependencies = &dependency;
00234
00235     if (vkCreateRenderPass(m_device.device(), &renderPassInfo, nullptr, &m_renderPass) != VK_SUCCESS)
    {
00236         throw std::runtime_error("failed to create render pass!");
00237     }
00238 }
00239
00240 void ven::SwapChain::createFrameBuffers()
00241 {
00242     m_swapChainFrameBuffers.resize(imageCount());
00243     for (size_t i = 0; i < imageCount(); i++) {
00244         std::array<VkImageView, 2> attachments = {m_swapChainImageViews[i], m_depthImageViews[i]};
00245
00246         const auto [width, height] = getSwapChainExtent();
00247         VkFramebufferCreateInfo framebufferInfo = {};
00248         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00249         framebufferInfo.renderPass = m_renderPass;
00250         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00251         framebufferInfo.pAttachments = attachments.data();
00252         framebufferInfo.width = width;
00253         framebufferInfo.height = height;
00254         framebufferInfo.layers = 1;
00255
00256         if (vkCreateFramebuffer(m_device.device(), &framebufferInfo, nullptr,
&m_swapChainFrameBuffers[i]) != VK_SUCCESS) {
00257             throw std::runtime_error("failed to create framebuffer!");
00258         }
00259     }
00260 }
00261
00262 void ven::SwapChain::createDepthResources()
00263 {
00264     const VkFormat depthFormat = findDepthFormat();
00265     const auto [width, height] = getSwapChainExtent();
00266
00267     m_swapChainDepthFormat = depthFormat;
00268     m_depthImages.resize(imageCount());
00269     m_depthImageMemory.resize(imageCount());
00270     m_depthImageViews.resize(imageCount());
00271
00272     for (size_t i = 0; i < m_depthImages.size(); i++) {
00273         VkImageCreateInfo imageInfo{};
00274         imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00275         imageInfo.imageType = VK_IMAGE_TYPE_2D;
00276         imageInfo.extent.width = width;
00277         imageInfo.extent.height = height;
00278         imageInfo.extent.depth = 1;
00279         imageInfo.mipLevels = 1;
00280         imageInfo.arrayLayers = 1;
00281         imageInfo.format = depthFormat;
00282         imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00283         imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00284         imageInfo.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;
00285         imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00286         imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00287         imageInfo.flags = 0;
00288
00289         m_device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_depthImages[i],
m_depthImageMemory[i]);
00290
00291         VkImageViewCreateInfo viewInfo{};
00292         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00293         viewInfo.image = m_depthImages[i];
00294         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00295         viewInfo.format = depthFormat;

```

```

00296         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00297         viewInfo.subresourceRange.baseMipLevel = 0;
00298         viewInfo.subresourceRange.levelCount = 1;
00299         viewInfo.subresourceRange.baseArrayLayer = 0;
00300         viewInfo.subresourceRange.layerCount = 1;
00301
00302         if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_depthImageViews[i]) !=
VK_SUCCESS) {
00303             throw std::runtime_error("failed to create texture image view!");
00304         }
00305     }
00306 }
00307
00308 void ven::SwapChain::createSyncObjects()
00309 {
00310     m_imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00311     m_renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00312     m_inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
00313     m_imagesInFlight.resize(imageCount(), VK_NULL_HANDLE);
00314
00315     VkSemaphoreCreateInfo semaphoreInfo = {};
00316     semaphoreInfo.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
00317
00318     VkFenceCreateInfo fenceInfo = {};
00319     fenceInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
00320     fenceInfo.flags = VK_FENCE_CREATE_SIGNALED_BIT;
00321
00322     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00323         if (vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
&m_imageAvailableSemaphores[i]) != VK_SUCCESS ||
00324             vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
&m_renderFinishedSemaphores[i]) != VK_SUCCESS ||
00325             vkCreateFence(m_device.device(), &fenceInfo, nullptr, &m_inFlightFences[i]) != VK_SUCCESS)
00326             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
00339     return availableFormats[0];
00340 }
00341
00342 VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes)
00343 {
00344     for (const auto &availablePresentMode : availablePresentModes) {
00345         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00346             std::cout << "Present mode: Mailbox\n";
00347             return availablePresentMode;
00348         }
00349     }
00350
00351     for (const auto &availablePresentMode : availablePresentModes) {
00352         if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00353             std::cout << "Present mode: Immediate" << '\n';
00354             return availablePresentMode;
00355         }
00356     }
00357
00358     std::cout << "Present mode: V-Sync\n";
00359     return VK_PRESENT_MODE_FIFO_KHR;
00360 }
00361
00362 VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00363 {
00364     if (capabilities.currentExtent.width != std::numeric_limits<uint32_t>::max()) {
00365         return capabilities.currentExtent;
00366     }
00367     VkExtent2D actualExtent = m_windowExtent;
00368     actualExtent.width = std::max(capabilities.minImageExtent.width,
std::min(capabilities.maxImageExtent.width, actualExtent.width));
00369     actualExtent.height = std::max(capabilities.minImageExtent.height,
std::min(capabilities.maxImageExtent.height, actualExtent.height));
00370
00371     return actualExtent;
00372 }
00373
00373

```

```

00374 VkFormat ven::SwapChain::findDepthFormat() const
00375 {
00376     return m_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 }

```

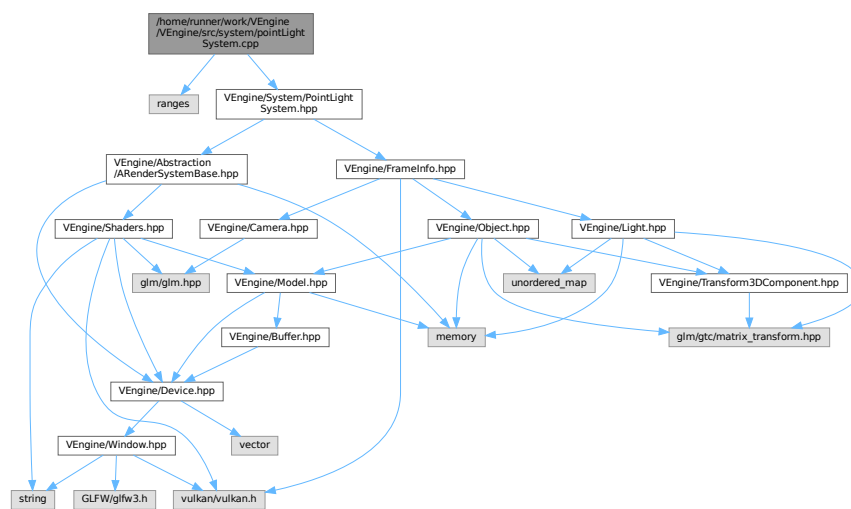
8.92 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference

```

#include <ranges>
#include "VEngine/System/PointLightSystem.hpp"

```

Include dependency graph for pointLightSystem.cpp:



8.93 pointLightSystem.cpp

[Go to the documentation of this file.](#)

```

00001 #include <ranges>
00002
00003 #include "VEngine/System/PointLightSystem.hpp"
00004
00005 void ven::PointLightSystem::render(const FrameInfo &frameInfo) const
00006 {
00007     getShaders()->bind(frameInfo.commandBuffer);
00008     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
00009         getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00010
00011     for (const Light &light : frameInfo.lights | std::views::values) {
00012         const LightPushConstantData push{
00013             .position = glm::vec4(light.transform3D.translation, 1.F),
00014             .color = light.color,
00015             .radius = light.transform3D.scale.x
00016         };
00017         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
00018             VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
00019         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00020     }
00021
00022 void ven::PointLightSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
00023 {
00024     const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});

```



```

00024     int lightIndex = 0;
00025
00026     for (Light &light : frameInfo.lights | std::views::values) {
00027         assert(lightIndex < MAX_LIGHTS && "Too many lights");
00028         light.transform3D.translation = glm::vec3(rotateLight *
glm::vec4(light.transform3D.translation, 1.F));
00029         ubo.pointLights.at(lightIndex).position = glm::vec4(light.transform3D.translation, 1.F);
00030         ubo.pointLights.at(lightIndex).color = light.color;
00031         lightIndex++;
00032     }
00033     ubo.numLights = lightIndex;
00034 }

```

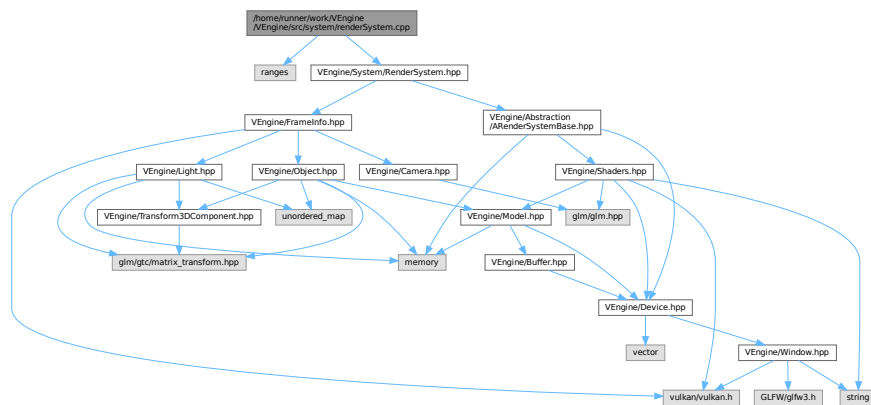
8.94 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference

```

#include <ranges>
#include "VEngine/System/RenderSystem.hpp"

```

Include dependency graph for renderSystem.cpp:



8.95 renderSystem.cpp

[Go to the documentation of this file.](#)

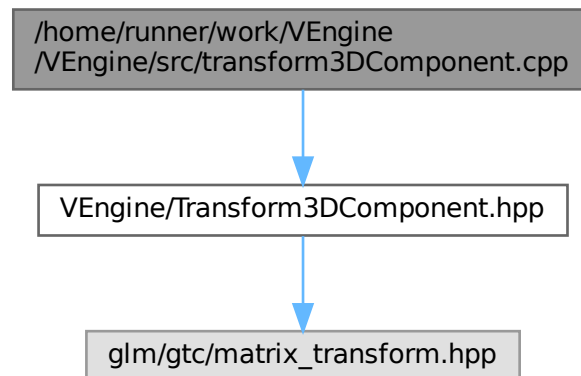
```

00001 #include <ranges>
00002
00003 #include "VEngine/System/RenderSystem.hpp"
00004
00005 void ven::RenderSystem::renderObjects(const FrameInfo &frameInfo) const
00006 {
00007     getShaders()->bind(frameInfo.commandBuffer);
00008
00009     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00010
00011     for (const Object& object : frameInfo.objects | std::views::values) {
00012         if (object.getModel() == nullptr) { continue; }
00013         const ObjectPushConstantData push{
00014             .modelMatrix = object.transform3D.mat4(),
00015             .normalMatrix = object.transform3D.normalMatrix()
00016         };
00017         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(ObjectPushConstantData), &push);
00018         object.getModel()->bind(frameInfo.commandBuffer);
00019         object.getModel()->draw(frameInfo.commandBuffer);
00020     }
00021 }

```

8.96 /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp File Reference

#include "VEngine/Transform3DComponent.hpp"
 Include dependency graph for transform3DComponent.cpp:



8.97 transform3DComponent.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Transform3DComponent.hpp"
00002
00003 glm::mat4 ven::Transform3DComponent::mat4() const {
00004     const float c3 = glm::cos(rotation.z);
00005     const float s3 = glm::sin(rotation.z);
00006     const float c2 = glm::cos(rotation.x);
00007     const float s2 = glm::sin(rotation.x);
00008     const float c1 = glm::cos(rotation.y);
00009     const float s1 = glm::sin(rotation.y);
00010     return glm::mat4{
00011         {
00012             scale.x * (c1 * c3 + s1 * s2 * s3),
00013             scale.x * (c2 * s3),
00014             scale.x * (c1 * s2 * s3 - c3 * s1),
00015             0.0F,
00016         },
00017         {
00018             scale.y * (c3 * s1 * s2 - c1 * s3),
00019             scale.y * (c2 * c3),
00020             scale.y * (c1 * c3 * s2 + s1 * s3),
00021             0.0F,
00022         },
00023         {
00024             scale.z * (c2 * s1),
00025             scale.z * (-s2),
00026             scale.z * (c1 * c2),
00027             0.0F,
00028         },
00029         {
00030             translation.x,
00031             translation.y,
00032             translation.z,
00033             1.0F
00034         }
00035     };
00036 }
00037

```

```

00038 glm::mat3 ven::Transform3DComponent::normalMatrix() const
00039 {
00040     const float c3 = glm::cos(rotation.z);
00041     const float s3 = glm::sin(rotation.z);
00042     const float c2 = glm::cos(rotation.x);
00043     const float s2 = glm::sin(rotation.x);
00044     const float c1 = glm::cos(rotation.y);
00045     const float s1 = glm::sin(rotation.y);
00046     const glm::vec3 invScale = 1.0F / scale;
00047
00048     return glm::mat3{
00049         {
00050             invScale.x * (c1 * c3 + s1 * s2 * s3),
00051             invScale.x * (c2 * s3),
00052             invScale.x * (c1 * s2 * s3 - c3 * s1)
00053         },
00054         {
00055             invScale.y * (c3 * s1 * s2 - c1 * s3),
00056             invScale.y * (c2 * c3),
00057             invScale.y * (c1 * c3 * s2 + s1 * s3)
00058         },
00059         {
00060             invScale.z * (c2 * s1),
00061             invScale.z * (-s2),
00062             invScale.z * (c1 * c2)
00063         }
00064     };
00065 }

```

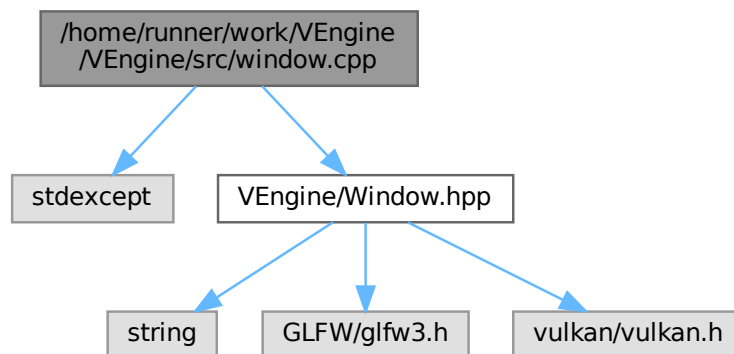
8.98 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

```

#include <stdexcept>
#include "VEngine/Window.hpp"

```

Include dependency graph for window.cpp:



8.99 window.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002
00003 #include "VEngine/Window.hpp"
00004

```

```

00005 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string
    &title)
00006 {
00007     if (glfwInit() == GLFW_FALSE) {
00008         throw std::runtime_error("Failed to initialize GLFW");
00009     }
00010
00011     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
00012     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00013
00014     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
    title.c_str(), nullptr, nullptr);
00015     if (window == nullptr) {
00016         glfwTerminate();
00017         throw std::runtime_error("Failed to create window");
00018     }
00019     glfwSetWindowUserPointer(window, this);
00020     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
00021     return window;
00022 }
00023
00024 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
00025 {
00026     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
00027         throw std::runtime_error("Failed to create window surface");
00028     }
00029 }
00030
00031 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
00032 {
00033     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
00034     app->m_framebufferResized = true;
00035     app->m_width = static_cast<uint32_t>(width);
00036     app->m_height = static_cast<uint32_t>(height);
00037 }
00038
00039 void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
00040 {
00041     GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
00042     const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);
00043
00044     if (fullscreen) {
00045         glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
    mode->refreshRate);
00046     } else {
00047         // To restore a window that was originally windowed to its original size and position,
00048         // save these before making it full screen and then pass them in as above
00049         glfwSetWindowMonitor(m_window, nullptr, 0, 0, width, height, mode->refreshRate);
00050     }
00051 }
00052
00053
00054     m_width = width;
00055     m_height = height;
00056 }

```

Index

/home/runner/work/VEngine/VEngine/README.md, 260
/home/runner/work/VEngine/VEngine/include/VEngine/Abstraction/RenderSystemBase.hpp, 201, 202
/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp, 203, 204
/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp, 206, 208
/home/runner/work/VEngine/VEngine/include/VEngine/Color.hpp, 209, 210
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp, 211, 212
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp, 213, 215
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp, 215, 217
/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp, 217, 219
/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp, 220, 221
/home/runner/work/VEngine/VEngine/include/VEngine/Framerate.hpp, 222, 223
/home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp, 224, 225
/home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp, 226, 227
/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp, 228, 229
/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp, 230, 232
/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp, 233, 234
/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp, 235, 236
/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp, 237, 238
/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp, 239, 241
/home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp, 242, 243
/home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp, 244, 245
/home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp, 246, 247
/home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp, 247, 248
/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp, 248, 250
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock, 251, 252
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Color, 253, 255
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Framebuffer, 255, 257
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp, 258
/home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp, 259
/home/runner/work/VEngine/VEngine/src/Abstraction/renderSystemBase.cpp, 260
/home/runner/work/VEngine/VEngine/src/buffer.cpp, 261
/home/runner/work/VEngine/VEngine/src/camera.cpp, 262, 263
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp, 265
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp, 266
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp, 267, 268
/home/runner/work/VEngine/VEngine/src/device.cpp, 268, 271
/home/runner/work/VEngine/VEngine/src/engine.cpp, 277
/home/runner/work/VEngine/VEngine/src/gui/init.cpp, 279, 280
/home/runner/work/VEngine/VEngine/src/gui/render.cpp, 281, 282
/home/runner/work/VEngine/VEngine/src/keyboardController.cpp, 286
/home/runner/work/VEngine/VEngine/src/light.cpp, 288
/home/runner/work/VEngine/VEngine/src/main.cpp, 288, 289
/home/runner/work/VEngine/VEngine/src/model.cpp, 290, 291
/home/runner/work/VEngine/VEngine/src/renderer.cpp, 295, 296
/home/runner/work/VEngine/VEngine/src/shaders.cpp, 298, 299
/home/runner/work/VEngine/VEngine/src/swapChain.cpp, 299, 300
/home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp, 304
/home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp, 305
/home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp, 306

/home/runner/work/VEngine/VEngine/src/window.cpp,
 307
 ~ARenderSystemBase
 ven::ARenderSystemBase, 23
 ~Buffer
 ven::Buffer, 29
 ~Clock
 myLib::Clock, 56
 ~DescriptorPool
 ven::DescriptorPool, 69
 ~DescriptorSetLayout
 ven::DescriptorSetLayout, 74
 ~Device
 ven::Device, 83
 ~Engine
 ven::Engine, 99
 ~ImGuiWindowManager
 ven::ImGuiWindowManager, 113
 ~Light
 ven::Light, 125
 ~Model
 ven::Model, 131
 ~Object
 ven::Object, 137
 ~Renderer
 ven::Renderer, 158
 ~Shaders
 ven::Shaders, 171
 ~SwapChain
 ven::SwapChain, 178
 ~Window
 ven::Window, 196

 acquireNextImage
 ven::SwapChain, 179
 addBinding
 ven::DescriptorSetLayout::Builder, 45
 addPoolSize
 ven::DescriptorPool::Builder, 41
 allocateDescriptor
 ven::DescriptorPool, 70
 ambientLightColor
 ven::GlobalUbo, 110
 AQUA
 ven::Colors, 61
 AQUA_V
 ven::Colors, 61
 ARenderSystemBase
 ven::ARenderSystemBase, 23
 asMicroseconds
 myLib::Time, 188
 asMilliseconds
 myLib::Time, 188
 asSeconds
 myLib::Time, 189
 attributeDescriptions
 ven::PipelineConfigInfo, 144
 beginFrame
 ven::Renderer, 158
 beginSingleTimeCommands
 ven::Device, 84
 beginSwapChainRenderPass
 ven::Renderer, 158
 bind
 ven::Model, 132
 ven::Shaders, 172
 bindingDescriptions
 ven::PipelineConfigInfo, 144
 BLACK
 ven::Colors, 61
 BLACK_V
 ven::Colors, 61
 BLUE
 ven::Colors, 61
 BLUE_V
 ven::Colors, 61
 Buffer
 ven::Buffer, 29
 build
 ven::DescriptorPool::Builder, 41
 ven::DescriptorSetLayout::Builder, 45
 ven::DescriptorWriter, 78
 Builder
 ven::DescriptorPool::Builder, 41
 ven::DescriptorSetLayout::Builder, 45

 camera
 ven::FrameInfo, 106
 cameraSection
 ven::ImGuiWindowManager, 113
 capabilities
 ven::SwapChainSupportDetails, 187
 checkDeviceExtensionSupport
 ven::Device, 84
 checkValidationLayerSupport
 ven::Device, 84
 chooseSwapExtent
 ven::SwapChain, 179
 chooseSwapPresentMode
 ven::SwapChain, 179
 chooseSwapSurfaceFormat
 ven::SwapChain, 179
 cleanup
 ven::ImGuiWindowManager, 114
 CLEAR_COLORS
 ven::Colors, 61
 Clock
 myLib::Clock, 56
 Clock.hpp
 TimePoint, 252
 color
 ven::Light, 127
 ven::LightPushConstantData, 129
 ven::Model::Vertex, 194
 ven::Object, 140
 ven::PointLightData, 146
 COLOR_MAX

- ven::Colors, 62
- colorBlendAttachment
 - ven::PipelineConfigInfo, 144
- colorBlendInfo
 - ven::PipelineConfigInfo, 144
- COLORS
 - ven::Colors, 62
- commandBuffer
 - ven::FrameInfo, 106
- compareSwapFormats
 - ven::SwapChain, 179
- copyBuffer
 - ven::Device, 84
- copyBufferToImage
 - ven::Device, 85
- createBuffer
 - ven::Device, 85
- createCommandBuffers
 - ven::Renderer, 158
- createCommandPool
 - ven::Device, 85
- CreateDebugUtilsMessengerEXT
 - device.cpp, 269
- createDepthResources
 - ven::SwapChain, 179
- createFrameBuffers
 - ven::SwapChain, 179
- createGraphicsPipeline
 - ven::Shaders, 172
- createImageViews
 - ven::SwapChain, 180
- createImageWithInfo
 - ven::Device, 86
- createIndexBuffer
 - ven::Model, 132
- createInstance
 - ven::Device, 86
 - ven::Engine, 99
- createLight
 - ven::Light, 126
- createLogicalDevice
 - ven::Device, 86
- createModelFromFile
 - ven::Model, 132
- createObject
 - ven::Object, 138
- createPipeline
 - ven::ARenderSystemBase, 24
- createPipelineLayout
 - ven::ARenderSystemBase, 24
- createRenderPass
 - ven::SwapChain, 180
- createShaderModule
 - ven::Shaders, 172
- createSurface
 - ven::Device, 87
 - ven::Engine, 100
- createSwapChain
 - ven::SwapChain, 180
- createSyncObjects
 - ven::SwapChain, 180
- createVertexBuffer
 - ven::Model, 133
- createWindow
 - ven::Window, 196
- createWindowSurface
 - ven::Window, 196
- CYAN
 - ven::Colors, 62
- CYAN_V
 - ven::Colors, 63
- debugCallback
 - device.cpp, 269
- DEFAULT_AMBIENT_LIGHT_COLOR
 - ven, 16
- DEFAULT_AMBIENT_LIGHT_INTENSITY
 - ven, 16
- DEFAULT_CLEAR_COLOR
 - ven, 17
- DEFAULT_CLEAR_DEPTH
 - ven, 17
- 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, 18
- DEFAULT_LOOK_SPEED
 - ven, 18
- DEFAULT_MOVE_SPEED
 - ven, 18
- DEFAULT_NEAR
 - ven, 18
- DEFAULT_POSITION
 - ven, 18
- DEFAULT_ROTATION
 - ven, 18
- DEFAULT_TITLE
 - ven, 19
- DEFAULT_WIDTH
 - ven, 19
- defaultPipelineConfigInfo
 - ven::Shaders, 172
- depthStencilInfo
 - ven::PipelineConfigInfo, 144
- DESCRIPTOR_COUNT
 - init.cpp, 280
- descriptorInfo
 - ven::Buffer, 29
- descriptorInfoForIndex

- ven::Buffer, 30
- DescriptorPool
 - ven::DescriptorPool, 69, 70
- DescriptorSetLayout
 - ven::DescriptorSetLayout, 74, 75
- DescriptorWriter
 - ven::DescriptorPool, 71
 - ven::DescriptorSetLayout, 75
 - ven::DescriptorWriter, 78
- DestroyDebugUtilsMessengerEXT
 - device.cpp, 270
- Device
 - ven::Device, 83, 84
- device
 - ven::Device, 87
- device.cpp
 - CreateDebugUtilsMessengerEXT, 269
 - debugCallback, 269
 - DestroyDebugUtilsMessengerEXT, 270
- deviceExtensions
 - ven::Device, 94
- devicePropertiesSection
 - ven::ImGuiWindowManager, 114
- draw
 - ven::Model, 134
- dynamicStateEnables
 - ven::PipelineConfigInfo, 144
- dynamicStateInfo
 - ven::PipelineConfigInfo, 144
- enableValidationLayers
 - ven::Device, 94
- endFrame
 - ven::Renderer, 159
- endSingleTimeCommands
 - ven::Device, 88
- endSwapChainRenderPass
 - ven::Renderer, 159
- Engine
 - ven::Engine, 98, 99
- extentAspectRatio
 - ven::SwapChain, 180
- findDepthFormat
 - ven::SwapChain, 180
- findMemoryType
 - ven::Device, 88
- findPhysicalQueueFamilies
 - ven::Device, 89
- findQueueFamilies
 - ven::Device, 89
- findSupportedFormat
 - ven::Device, 89
- flush
 - ven::Buffer, 30
- flushIndex
 - ven::Buffer, 31
- formats
 - ven::SwapChainSupportDetails, 187
- framebufferResizeCallback
 - ven::Window, 197
- frameIndex
 - ven::FrameInfo, 106
- frameTime
 - ven::FrameInfo, 106
- freeCommandBuffers
 - ven::Renderer, 159
- freeDescriptors
 - ven::DescriptorPool, 70
- FUCHSIA
 - ven::Colors, 63
- FUCHSIA_V
 - ven::Colors, 63
- getAlignment
 - ven::Buffer, 32
- getAlignmentSize
 - ven::Buffer, 32
- getAspectRatio
 - ven::Renderer, 160
- getAttributeDescriptions
 - ven::Model::Vertex, 193
- getBindingDescriptions
 - ven::Model::Vertex, 193
- getBuffer
 - ven::Buffer, 32
- getBufferSize
 - ven::Buffer, 33
- getClearColor
 - ven::Renderer, 160
- getCommandPool
 - ven::Device, 90
- getCurrentCommandBuffer
 - ven::Renderer, 160
- getDescriptorPool
 - ven::DescriptorPool, 70
- getDescriptorSetLayout
 - ven::DescriptorSetLayout, 75
- getDevice
 - ven::ARenderSystemBase, 25
- getElapsedTime
 - myLib::Clock, 56
- getExtent
 - ven::Window, 197
- getFar
 - ven::Camera, 50
- getFov
 - ven::Camera, 50
- getFrameBuffer
 - ven::SwapChain, 180
- getFrameIndex
 - ven::Renderer, 161
- getGLFWWindow
 - ven::Window, 198
- getGraphicsQueue
 - ven::Device, 90
- getId
 - ven::Light, 126

- ven::Object, [139](#)
- getImageView
 - ven::SwapChain, [181](#)
- getInstanceCount
 - ven::Buffer, [33](#)
- getInstanceSize
 - ven::Buffer, [33](#)
- getInverseView
 - ven::Camera, [51](#)
- getMappedMemory
 - ven::Buffer, [33](#)
- getMemoryPropertyFlags
 - ven::Buffer, [33](#)
- getModel
 - ven::Object, [139](#)
- getName
 - ven::Light, [126](#)
 - ven::Object, [139](#)
- getNear
 - ven::Camera, [51](#)
- getPhysicalDevice
 - ven::Device, [90](#)
- getPipelineLayout
 - ven::ARenderSystemBase, [25](#)
- getProjection
 - ven::Camera, [51](#)
- getRenderPass
 - ven::SwapChain, [181](#)
- getRequiredExtensions
 - ven::Device, [90](#)
- getShaders
 - ven::ARenderSystemBase, [25](#)
- getSwapChainExtent
 - ven::SwapChain, [181](#)
- getSwapChainImageFormat
 - ven::SwapChain, [181](#)
- getSwapChainRenderPass
 - ven::Renderer, [161](#)
- getSwapChainSupport
 - ven::Device, [91](#)
- getUsageFlags
 - ven::Buffer, [33](#)
- getView
 - ven::Camera, [52](#)
- getWindow
 - ven::Renderer, [162](#)
- GLFW_INCLUDE_VULKAN
 - Window.hpp, [250](#)
- GLM_ENABLE_EXPERIMENTAL
 - model.cpp, [290](#)
- globalDescriptorSet
 - ven::FrameInfo, [106](#)
- graphicsFamily
 - ven::QueueFamilyIndices, [152](#)
- graphicsFamilyHasValue
 - ven::QueueFamilyIndices, [152](#)
- graphicsQueue
 - ven::Device, [91](#)
- GRAY
 - ven::Colors, [63](#)
- GRAY_V
 - ven::Colors, [63](#)
- GREEN
 - ven::Colors, [63](#)
- GREEN_V
 - ven::Colors, [63](#)
- hasGlfwRequiredInstanceExtensions
 - ven::Device, [91](#)
- hashCombine
 - ven, [16](#)
- height
 - ven::SwapChain, [181](#)
- imageCount
 - ven::SwapChain, [181](#)
- ImGuiWindowManager
 - ven::ImGuiWindowManager, [113](#)
- indices
 - ven::Model::Builder, [48](#)
- init
 - ven::ImGuiWindowManager, [114](#)
 - ven::SwapChain, [182](#)
- init.cpp
 - DESCRIPTOR_COUNT, [280](#)
- initStyle
 - ven::ImGuiWindowManager, [115](#)
- inputAssemblyInfo
 - ven::PipelineConfigInfo, [145](#)
- inputsSection
 - ven::ImGuiWindowManager, [116](#)
- invalidate
 - ven::Buffer, [34](#)
- invalidateIndex
 - ven::Buffer, [34](#)
- inverseView
 - ven::GlobalUbo, [110](#)
- isComplete
 - ven::QueueFamilyIndices, [152](#)
- isDeviceSuitable
 - ven::Device, [91](#)
- isFrameInProgress
 - ven::Renderer, [162](#)
- IsLegacyNativeDupe
 - ven::ImGuiWindowManager::funcs, [108](#)
- Light
 - ven::Light, [125](#), [126](#)
- lights
 - ven::FrameInfo, [106](#)
- lightsSection
 - ven::ImGuiWindowManager, [116](#)
- LIME
 - ven::Colors, [64](#)
- LIME_V
 - ven::Colors, [64](#)
- loadModel

- ven::Model::Builder, 48
- loadObjects
 - ven::Engine, 100
- lookDown
 - ven::KeyboardController::KeyMappings, 122
- lookLeft
 - ven::KeyboardController::KeyMappings, 122
- lookRight
 - ven::KeyboardController::KeyMappings, 122
- lookUp
 - ven::KeyboardController::KeyMappings, 122
- m_alignmentSize
 - ven::Buffer, 37
- m_bindings
 - ven::DescriptorSetLayout, 76
 - ven::DescriptorSetLayout::Builder, 46
- m_buffer
 - ven::Buffer, 37
- m_bufferSize
 - ven::Buffer, 37
- m_clearValues
 - ven::Renderer, 164
- m_commandBuffers
 - ven::Renderer, 164
- m_commandPool
 - ven::Device, 95
- m_currentFrame
 - ven::SwapChain, 183
- m_currentFrameIndex
 - ven::Renderer, 164
- m_currentImageIndex
 - ven::Renderer, 164
- m_debugMessenger
 - ven::Device, 95
- m_depthImageMemory
 - ven::SwapChain, 183
- m_depthImages
 - ven::SwapChain, 183
- m_depthImageViews
 - ven::SwapChain, 183
- m_descriptorPool
 - ven::DescriptorPool, 71
- m_descriptorSetLayout
 - ven::DescriptorSetLayout, 76
- m_device
 - ven::ARenderSystemBase, 26
 - ven::Buffer, 37
 - ven::DescriptorPool, 71
 - ven::DescriptorPool::Builder, 43
 - ven::DescriptorSetLayout, 76
 - ven::DescriptorSetLayout::Builder, 46
 - ven::Device, 95
 - ven::Engine, 102
 - ven::Model, 134
 - ven::Renderer, 164
 - ven::Shaders, 174
 - ven::SwapChain, 183
- m_far
 - ven::Camera, 54
- m_fov
 - ven::Camera, 54
- m_fragShaderModule
 - ven::Shaders, 174
- m_framebufferResized
 - ven::Window, 199
- m_globalPool
 - ven::Engine, 102
- m_graphicsPipeline
 - ven::Shaders, 174
- m_graphicsQueue
 - ven::Device, 95
- m_hasIndexBuffer
 - ven::Model, 134
- m_height
 - ven::Window, 199
- m_imageAvailableSemaphores
 - ven::SwapChain, 183
- m_imagesInFlight
 - ven::SwapChain, 183
- m_indexBuffer
 - ven::Model, 134
- m_indexCount
 - ven::Model, 134
- m_inFlightFences
 - ven::SwapChain, 184
- m_instance
 - ven::Device, 95
 - ven::Engine, 103
- m_instanceCount
 - ven::Buffer, 38
- m_instanceSize
 - ven::Buffer, 38
- m_inverseViewMatrix
 - ven::Camera, 54
- m_isFrameStarted
 - ven::Renderer, 164
- m_keys
 - ven::KeyboardController, 120
- m_lightId
 - ven::Light, 127
- m_lights
 - ven::Engine, 103
- m_lookSpeed
 - ven::KeyboardController, 120
- m_mapped
 - ven::Buffer, 38
- m_maxSets
 - ven::DescriptorPool::Builder, 43
- m_memory
 - ven::Buffer, 38
- m_memoryPropertyFlags
 - ven::Buffer, 38
- m_model
 - ven::Object, 140
- m_moveSpeed
 - ven::KeyboardController, 120

- m_name
 - ven::Light, 127
 - ven::Object, 141
- m_near
 - ven::Camera, 54
- m_objects
 - ven::Engine, 103
- m_objId
 - ven::Object, 141
- m_oldSwapChain
 - ven::SwapChain, 184
- m_pause
 - myLib::Clock, 57
- m_paused
 - myLib::Clock, 57
- m_physicalDevice
 - ven::Device, 95
- m_pipelineLayout
 - ven::ARenderSystemBase, 26
- m_pool
 - ven::DescriptorWriter, 79
- m_poolFlags
 - ven::DescriptorPool::Builder, 43
- m_poolSizes
 - ven::DescriptorPool::Builder, 43
- m_presentQueue
 - ven::Device, 96
- m_projectionMatrix
 - ven::Camera, 54
- m_properties
 - ven::Device, 96
- m_renderer
 - ven::Engine, 103
- m_renderFinishedSemaphores
 - ven::SwapChain, 184
- m_renderPass
 - ven::SwapChain, 184
- m_seconds
 - myLib::Time, 189
- m_setLayout
 - ven::DescriptorWriter, 79
- m_shaders
 - ven::ARenderSystemBase, 26
- m_start
 - myLib::Clock, 57
- m_surface
 - ven::Device, 96
 - ven::Engine, 103
- m_swapChain
 - ven::Renderer, 164
 - ven::SwapChain, 184
- m_swapChainDepthFormat
 - ven::SwapChain, 184
- m_swapChainExtent
 - ven::SwapChain, 185
- m_swapChainFrameBuffers
 - ven::SwapChain, 185
- m_swapChainImageFormat
 - ven::SwapChain, 185
- m_swapChainImages
 - ven::SwapChain, 185
- m_swapChainImageViews
 - ven::SwapChain, 185
- m_usageFlags
 - ven::Buffer, 38
- m_vertexBuffer
 - ven::Model, 135
- m_vertexCount
 - ven::Model, 135
- m_vertShaderModule
 - ven::Shaders, 174
- m_viewMatrix
 - ven::Camera, 55
- m_width
 - ven::Window, 199
- m_window
 - ven::Device, 96
 - ven::Engine, 103
 - ven::Renderer, 165
 - ven::Window, 200
- m_windowExtent
 - ven::SwapChain, 185
- m_writes
 - ven::DescriptorWriter, 79
- MAGENTA
 - ven::Colors, 64
- MAGENTA_V
 - ven::Colors, 64
- main
 - main.cpp, 289
- main.cpp
 - main, 289
- mainLoop
 - ven::Engine, 101
- Map
 - ven::Light, 125
 - ven::Object, 137
- map
 - ven::Buffer, 35
- MAROON
 - ven::Colors, 64
- MAROON_V
 - ven::Colors, 64
- mat4
 - ven::Transform3DComponent, 190
- MAX_FRAMES_IN_FLIGHT
 - ven::SwapChain, 186
- MAX_LIGHTS
 - ven, 19
- MICROSECONDS_PER_SECOND
 - myLib, 13
- MILLISECONDS_PER_SECOND
 - myLib, 13
- Model
 - ven::Model, 131, 132
- model.cpp

- GLM_ENABLE_EXPERIMENTAL, 290
- TINYOBJLOADER_IMPLEMENTATION, 290
- modelMatrix
 - ven::ObjectPushConstantData, 142
- moveBackward
 - ven::KeyboardController::KeyMappings, 122
- moveDown
 - ven::KeyboardController::KeyMappings, 122
- moveForward
 - ven::KeyboardController::KeyMappings, 123
- moveInPlaneXZ
 - ven::KeyboardController, 120
- moveLeft
 - ven::KeyboardController::KeyMappings, 123
- moveRight
 - ven::KeyboardController::KeyMappings, 123
- moveUp
 - ven::KeyboardController::KeyMappings, 123
- multisampleInfo
 - ven::PipelineConfigInfo, 145
- myLib, 13
 - MICROSECONDS_PER_SECOND, 13
 - MILLISECONDS_PER_SECOND, 13
 - RANDOM_FLOAT_MAX, 13
 - RANDOM_INT_MAX, 14
 - RANDOM_INT_MIN, 14
- myLib::Clock, 55
 - ~Clock, 56
 - Clock, 56
 - getElapsedTime, 56
 - m_pause, 57
 - m_paused, 57
 - m_start, 57
 - pause, 56
 - restart, 57
 - resume, 57
- myLib::Random, 153
 - randomFloat, 154
 - randomInt, 154, 155
- myLib::Time, 187
 - asMicroseconds, 188
 - asMilliseconds, 188
 - asSeconds, 189
 - m_seconds, 189
 - Time, 188
- NAVY
 - ven::Colors, 64
- NAVY_V
 - ven::Colors, 65
- NIGHT_BLUE
 - ven::Colors, 65
- NIGHT_BLUE_V
 - ven::Colors, 65
- NIGHT_MODE_V
 - ven::Colors, 65
- normal
 - ven::Model::Vertex, 194
- normalMatrix
 - ven::ObjectPushConstantData, 142
 - ven::Transform3DComponent, 190
- numLights
 - ven::GlobalUbo, 110
- Object
 - ven::Object, 137, 138
- objects
 - ven::FrameInfo, 107
- objectsSection
 - ven::ImGuiWindowManager, 116
- OLIVE
 - ven::Colors, 65
- OLIVE_V
 - ven::Colors, 65
- operator()
 - std::hash< ven::Model::Vertex >, 111
- operator=
 - ven::Buffer, 35
 - ven::DescriptorPool, 71
 - ven::DescriptorSetLayout, 75
 - ven::Device, 92
 - ven::Engine, 102
 - ven::ImGuiWindowManager, 116
 - ven::Light, 127
 - ven::Model, 134
 - ven::Object, 139
 - ven::PipelineConfigInfo, 143
 - ven::Renderer, 162
 - ven::RenderSystem, 168
 - ven::Shaders, 173
 - ven::SwapChain, 182
- operator==
 - ven::Model::Vertex, 193
- overwrite
 - ven::DescriptorWriter, 78
- pause
 - myLib::Clock, 56
- pickPhysicalDevice
 - ven::Device, 92
- PipelineConfigInfo
 - ven::PipelineConfigInfo, 143
- pipelineLayout
 - ven::PipelineConfigInfo, 145
- pointLights
 - ven::GlobalUbo, 110
- PointLightSystem
 - ven::PointLightSystem, 149
- populateDebugMessengerCreateInfo
 - ven::Device, 92
- position
 - ven::LightPushConstantData, 129
 - ven::Model::Vertex, 194
 - ven::PointLightData, 146
- presentFamily
 - ven::QueueFamilyIndices, 152
- presentFamilyHasValue
 - ven::QueueFamilyIndices, 152

- presentModes
 - ven::SwapChainSupportDetails, 187
- presentQueue
 - ven::Device, 93
- projection
 - ven::GlobalUbo, 110
- querySwapChainSupport
 - ven::Device, 93
- radius
 - ven::LightPushConstantData, 129
- RANDOM_FLOAT_MAX
 - myLib, 13
- RANDOM_INT_MAX
 - myLib, 14
- RANDOM_INT_MIN
 - myLib, 14
- randomFloat
 - myLib::Random, 154
- randomInt
 - myLib::Random, 154, 155
- rasterizationInfo
 - ven::PipelineConfigInfo, 145
- readFile
 - ven::Shaders, 173
- recreateSwapChain
 - ven::Renderer, 163
- RED
 - ven::Colors, 65
- RED_V
 - ven::Colors, 66
- render
 - ven::ImGuiWindowManager, 116
 - ven::PointLightSystem, 150
- Renderer
 - ven::Renderer, 157, 158
- rendererSection
 - ven::ImGuiWindowManager, 117
- renderFrameWindow
 - ven::ImGuiWindowManager, 118
- renderObjects
 - ven::RenderSystem, 168
- renderPass
 - ven::PipelineConfigInfo, 145
- RenderSystem
 - ven::RenderSystem, 167
- resetPool
 - ven::DescriptorPool, 71
- resetWindowResizedFlag
 - ven::Window, 198
- restart
 - myLib::Clock, 57
- resume
 - myLib::Clock, 57
- rotation
 - ven::Transform3DComponent, 191
- scale
 - ven::Transform3DComponent, 191
- setClearValue
 - ven::Renderer, 163
- setFar
 - ven::Camera, 52
- setFov
 - ven::Camera, 52
- setFullscreen
 - ven::Window, 198
- setMaxSets
 - ven::DescriptorPool::Builder, 42
- setModel
 - ven::Object, 140
- setName
 - ven::Light, 127
 - ven::Object, 140
- setNear
 - ven::Camera, 52
- setOrthographicProjection
 - ven::Camera, 53
- setPerspectiveProjection
 - ven::Camera, 53
- setPoolFlags
 - ven::DescriptorPool::Builder, 42
- setupDebugMessenger
 - ven::Device, 93
- setViewDirection
 - ven::Camera, 53
- setViewTarget
 - ven::Camera, 53
- setViewYXZ
 - ven::Camera, 54
- Shaders
 - ven::Shaders, 171
- SHADERS_BIN_PATH
 - ven, 19
- SILVER
 - ven::Colors, 66
- SILVER_V
 - ven::Colors, 66
- SKY_BLUE
 - ven::Colors, 66
- SKY_BLUE_V
 - ven::Colors, 66
- std::hash< ven::Model::Vertex >, 111
 - operator(), 111
- submitCommandBuffers
 - ven::SwapChain, 182
- subpass
 - ven::PipelineConfigInfo, 145
- SUNSET
 - ven::Colors, 66
- SUNSET_V
 - ven::Colors, 66
- surface
 - ven::Device, 94
- SwapChain
 - ven::SwapChain, 177, 178

- TEAL
 - ven::Colors, 67
- TEAL_V
 - ven::Colors, 67
- Time
 - myLib::Time, 188
- TimePoint
 - Clock.hpp, 252
- TINYOBJLOADER_IMPLEMENTATION
 - model.cpp, 290
- transform3D
 - ven::Light, 128
 - ven::Object, 141
- translation
 - ven::Transform3DComponent, 191
- unmap
 - ven::Buffer, 36
- update
 - ven::PointLightSystem, 150
- uv
 - ven::Model::Vertex, 194
- validationLayers
 - ven::Device, 96
- ven, 14
 - DEFAULT_AMBIENT_LIGHT_COLOR, 16
 - DEFAULT_AMBIENT_LIGHT_INTENSITY, 16
 - DEFAULT_CLEAR_COLOR, 17
 - DEFAULT_CLEAR_DEPTH, 17
 - DEFAULT_FAR, 17
 - DEFAULT_FOV, 17
 - DEFAULT_HEIGHT, 17
 - DEFAULT_LIGHT_COLOR, 17
 - DEFAULT_LIGHT_INTENSITY, 17
 - DEFAULT_LIGHT_RADIUS, 18
 - DEFAULT_LOOK_SPEED, 18
 - DEFAULT_MOVE_SPEED, 18
 - DEFAULT_NEAR, 18
 - DEFAULT_POSITION, 18
 - DEFAULT_ROTATION, 18
 - DEFAULT_TITLE, 19
 - DEFAULT_WIDTH, 19
 - hashCombine, 16
 - MAX_LIGHTS, 19
 - SHADERS_BIN_PATH, 19
- ven::ARenderSystemBase, 21
 - ~ARenderSystemBase, 23
 - ARenderSystemBase, 23
 - createPipeline, 24
 - createPipelineLayout, 24
 - getDevice, 25
 - getPipelineLayout, 25
 - getShaders, 25
 - m_device, 26
 - m_pipelineLayout, 26
 - m_shaders, 26
- ven::Buffer, 27
 - ~Buffer, 29
- Buffer, 29
 - descriptorInfo, 29
 - descriptorInfoForIndex, 30
 - flush, 30
 - flushIndex, 31
 - getAlignment, 32
 - getAlignmentSize, 32
 - getBuffer, 32
 - getBufferSize, 33
 - getInstanceCount, 33
 - getInstanceSize, 33
 - getMappedMemory, 33
 - getMemoryPropertyFlags, 33
 - getUsageFlags, 33
 - invalidate, 34
 - invalidateIndex, 34
 - m_alignmentSize, 37
 - m_buffer, 37
 - m_bufferSize, 37
 - m_device, 37
 - m_instanceCount, 38
 - m_instanceSize, 38
 - m_mapped, 38
 - m_memory, 38
 - m_memoryPropertyFlags, 38
 - m_usageFlags, 38
 - map, 35
 - operator=, 35
 - unmap, 36
 - writeToBuffer, 36
 - writeToIndex, 36
- ven::Camera, 49
 - getFar, 50
 - getFov, 50
 - getInverseView, 51
 - getNear, 51
 - getProjection, 51
 - getView, 52
 - m_far, 54
 - m_fov, 54
 - m_inverseViewMatrix, 54
 - m_near, 54
 - m_projectionMatrix, 54
 - m_viewMatrix, 55
 - setFar, 52
 - setFov, 52
 - setNear, 52
 - setOrthographicProjection, 53
 - setPerspectiveProjection, 53
 - setViewDirection, 53
 - setViewTarget, 53
 - setViewYXZ, 54
- ven::Colors, 58
 - AQUA, 61
 - AQUA_V, 61
 - BLACK, 61
 - BLACK_V, 61
 - BLUE, 61

- BLUE_V, 61
- CLEAR_COLORS, 61
- COLOR_MAX, 62
- COLORS, 62
- CYAN, 62
- CYAN_V, 63
- FUCHSIA, 63
- FUCHSIA_V, 63
- GRAY, 63
- GRAY_V, 63
- GREEN, 63
- GREEN_V, 63
- LIME, 64
- LIME_V, 64
- MAGENTA, 64
- MAGENTA_V, 64
- MAROON, 64
- MAROON_V, 64
- NAVY, 64
- NAVY_V, 65
- NIGHT_BLUE, 65
- NIGHT_BLUE_V, 65
- NIGHT_MODE_V, 65
- OLIVE, 65
- OLIVE_V, 65
- RED, 65
- RED_V, 66
- SILVER, 66
- SILVER_V, 66
- SKY_BLUE, 66
- SKY_BLUE_V, 66
- SUNSET, 66
- SUNSET_V, 66
- TEAL, 67
- TEAL_V, 67
- WHITE, 67
- WHITE_V, 67
- YELLOW, 67
- YELLOW_V, 67
- ven::DescriptorPool, 68
 - ~DescriptorPool, 69
 - allocateDescriptor, 70
 - DescriptorPool, 69, 70
 - DescriptorWriter, 71
 - freeDescriptors, 70
 - getDescriptorPool, 70
 - m_descriptorPool, 71
 - m_device, 71
 - operator=, 71
 - resetPool, 71
- ven::DescriptorPool::Builder, 39
 - addPoolSize, 41
 - build, 41
 - Builder, 41
 - m_device, 43
 - m_maxSets, 43
 - m_poolFlags, 43
 - m_poolSizes, 43
 - setMaxSets, 42
 - setPoolFlags, 42
- ven::DescriptorSetLayout, 72
 - ~DescriptorSetLayout, 74
 - DescriptorSetLayout, 74, 75
 - DescriptorWriter, 75
 - getDescriptorSetLayout, 75
 - m_bindings, 76
 - m_descriptorSetLayout, 76
 - m_device, 76
 - operator=, 75
- ven::DescriptorSetLayout::Builder, 44
 - addBinding, 45
 - build, 45
 - Builder, 45
 - m_bindings, 46
 - m_device, 46
- ven::DescriptorWriter, 76
 - build, 78
 - DescriptorWriter, 78
 - m_pool, 79
 - m_setLayout, 79
 - m_writes, 79
 - overwrite, 78
 - writeBuffer, 78
 - writImage, 79
- ven::Device, 80
 - ~Device, 83
 - beginSingleTimeCommands, 84
 - checkDeviceExtensionSupport, 84
 - checkValidationLayerSupport, 84
 - copyBuffer, 84
 - copyBufferToImage, 85
 - createBuffer, 85
 - createCommandPool, 85
 - createImageWithInfo, 86
 - createInstance, 86
 - createLogicalDevice, 86
 - createSurface, 87
 - Device, 83, 84
 - device, 87
 - deviceExtensions, 94
 - enableValidationLayers, 94
 - endSingleTimeCommands, 88
 - findMemoryType, 88
 - findPhysicalQueueFamilies, 89
 - findQueueFamilies, 89
 - findSupportedFormat, 89
 - getCommandPool, 90
 - getGraphicsQueue, 90
 - getPhysicalDevice, 90
 - getRequiredExtensions, 90
 - getSwapChainSupport, 91
 - graphicsQueue, 91
 - hasGlfwRequiredInstanceExtensions, 91
 - isDeviceSuitable, 91
 - m_commandPool, 95
 - m_debugMessenger, 95

- m_device, 95
- m_graphicsQueue, 95
- m_instance, 95
- m_physicalDevice, 95
- m_presentQueue, 96
- m_properties, 96
- m_surface, 96
- m_window, 96
- operator=, 92
- pickPhysicalDevice, 92
- populateDebugMessengerCreateInfo, 92
- presentQueue, 93
- querySwapChainSupport, 93
- setupDebugMessenger, 93
- surface, 94
- validationLayers, 96
- ven::Engine, 97
 - ~Engine, 99
 - createInstance, 99
 - createSurface, 100
 - Engine, 98, 99
 - loadObjects, 100
 - m_device, 102
 - m_globalPool, 102
 - m_instance, 103
 - m_lights, 103
 - m_objects, 103
 - m_renderer, 103
 - m_surface, 103
 - m_window, 103
 - mainLoop, 101
 - operator=, 102
- ven::FrameInfo, 104
 - camera, 106
 - commandBuffer, 106
 - frameIndex, 106
 - frameTime, 106
 - globalDescriptorSet, 106
 - lights, 106
 - objects, 107
- ven::GlobalUbo, 109
 - ambientLightColor, 110
 - inverseView, 110
 - numLights, 110
 - pointLights, 110
 - projection, 110
 - view, 110
- ven::ImGuiWindowManager, 112
 - ~ImGuiWindowManager, 113
 - cameraSection, 113
 - cleanup, 114
 - devicePropertiesSection, 114
 - ImGuiWindowManager, 113
 - init, 114
 - initStyle, 115
 - inputsSection, 116
 - lightsSection, 116
 - objectsSection, 116
 - operator=, 116
 - render, 116
 - rendererSection, 117
 - renderFrameWindow, 118
- ven::ImGuiWindowManager::funcs, 107
 - IsLegacyNativeDupe, 108
- ven::KeyboardController, 118
 - m_keys, 120
 - m_lookSpeed, 120
 - m_moveSpeed, 120
 - moveInPlaneXZ, 120
- ven::KeyboardController::KeyMappings, 121
 - lookDown, 122
 - lookLeft, 122
 - lookRight, 122
 - lookUp, 122
 - moveBackward, 122
 - moveDown, 122
 - moveForward, 123
 - moveLeft, 123
 - moveRight, 123
 - moveUp, 123
- ven::Light, 124
 - ~Light, 125
 - color, 127
 - createLight, 126
 - getId, 126
 - getName, 126
 - Light, 125, 126
 - m_lightId, 127
 - m_name, 127
 - Map, 125
 - operator=, 127
 - setName, 127
 - transform3D, 128
- ven::LightPushConstantData, 128
 - color, 129
 - position, 129
 - radius, 129
- ven::Model, 129
 - ~Model, 131
 - bind, 132
 - createIndexBuffer, 132
 - createModelFromFile, 132
 - createVertexBuffer, 133
 - draw, 134
 - m_device, 134
 - m_hasIndexBuffer, 134
 - m_indexBuffer, 134
 - m_indexCount, 134
 - m_vertexBuffer, 135
 - m_vertexCount, 135
 - Model, 131, 132
 - operator=, 134
- ven::Model::Builder, 47
 - indices, 48
 - loadModel, 48
 - vertices, 48

- ven::Model::Vertex, 192
 - color, 194
 - getAttributeDescriptions, 193
 - getBindingDescriptions, 193
 - normal, 194
 - operator==, 193
 - position, 194
 - uv, 194
- ven::Object, 135
 - ~Object, 137
 - color, 140
 - createObject, 138
 - getId, 139
 - getModel, 139
 - getName, 139
 - m_model, 140
 - m_name, 141
 - m_objId, 141
 - Map, 137
 - Object, 137, 138
 - operator=, 139
 - setModel, 140
 - setName, 140
 - transform3D, 141
- ven::ObjectPushConstantData, 141
 - modelMatrix, 142
 - normalMatrix, 142
- ven::PipelineConfigInfo, 142
 - attributeDescriptions, 144
 - bindingDescriptions, 144
 - colorBlendAttachment, 144
 - colorBlendInfo, 144
 - depthStencilInfo, 144
 - dynamicStateEnables, 144
 - dynamicStateInfo, 144
 - inputAssemblyInfo, 145
 - multisampleInfo, 145
 - operator=, 143
 - PipelineConfigInfo, 143
 - pipelineLayout, 145
 - rasterizationInfo, 145
 - renderPass, 145
 - subpass, 145
- ven::PointLightData, 146
 - color, 146
 - position, 146
- ven::PointLightSystem, 147
 - PointLightSystem, 149
 - render, 150
 - update, 150
- ven::QueueFamilyIndices, 151
 - graphicsFamily, 152
 - graphicsFamilyHasValue, 152
 - isComplete, 152
 - presentFamily, 152
 - presentFamilyHasValue, 152
- ven::Renderer, 156
 - ~Renderer, 158
 - beginFrame, 158
 - beginSwapChainRenderPass, 158
 - createCommandBuffers, 158
 - endFrame, 159
 - endSwapChainRenderPass, 159
 - freeCommandBuffers, 159
 - getAspectRatio, 160
 - getClearColor, 160
 - getCurrentCommandBuffer, 160
 - getFrameIndex, 161
 - getSwapChainRenderPass, 161
 - getWindow, 162
 - isFrameInProgress, 162
 - m_clearValues, 164
 - m_commandBuffers, 164
 - m_currentFrameIndex, 164
 - m_currentImageIndex, 164
 - m_device, 164
 - m_isFrameStarted, 164
 - m_swapChain, 164
 - m_window, 165
 - operator=, 162
 - recreateSwapChain, 163
 - Renderer, 157, 158
 - setClearValue, 163
- ven::RenderSystem, 165
 - operator=, 168
 - renderObjects, 168
 - RenderSystem, 167
- ven::Shaders, 169
 - ~Shaders, 171
 - bind, 172
 - createGraphicsPipeline, 172
 - createShaderModule, 172
 - defaultPipelineConfigInfo, 172
 - m_device, 174
 - m_fragShaderModule, 174
 - m_graphicsPipeline, 174
 - m_vertShaderModule, 174
 - operator=, 173
 - readFile, 173
 - Shaders, 171
- ven::SwapChain, 175
 - ~SwapChain, 178
 - acquireNextImage, 179
 - chooseSwapExtent, 179
 - chooseSwapPresentMode, 179
 - chooseSwapSurfaceFormat, 179
 - compareSwapFormats, 179
 - createDepthResources, 179
 - createFrameBuffers, 179
 - createImageViews, 180
 - createRenderPass, 180
 - createSwapChain, 180
 - createSyncObjects, 180
 - extentAspectRatio, 180
 - findDepthFormat, 180
 - getFramebuffer, 180

- getImageView, 181
- getRenderPass, 181
- getSwapChainExtent, 181
- getSwapChainImageFormat, 181
- height, 181
- imageCount, 181
- init, 182
- m_currentFrame, 183
- m_depthImageMemory, 183
- m_depthImages, 183
- m_depthImageViews, 183
- m_device, 183
- m_imageAvailableSemaphores, 183
- m_imagesInFlight, 183
- m_inFlightFences, 184
- m_oldSwapChain, 184
- m_renderFinishedSemaphores, 184
- m_renderPass, 184
- m_swapChain, 184
- m_swapChainDepthFormat, 184
- m_swapChainExtent, 185
- m_swapChainFrameBuffers, 185
- m_swapChainImageFormat, 185
- m_swapChainImages, 185
- m_swapChainImageViews, 185
- m_windowExtent, 185
- MAX_FRAMES_IN_FLIGHT, 186
- operator=, 182
- submitCommandBuffers, 182
- SwapChain, 177, 178
- width, 182
- ven::SwapChainSupportDetails, 186
 - capabilities, 187
 - formats, 187
 - presentModes, 187
- ven::Transform3DComponent, 190
 - mat4, 190
 - normalMatrix, 190
 - rotation, 191
 - scale, 191
 - translation, 191
- ven::Window, 195
 - ~Window, 196
 - createWindow, 196
 - createWindowSurface, 196
 - framebufferResizeCallback, 197
 - getExtent, 197
 - getGLFWWindow, 198
 - m_framebufferResized, 199
 - m_height, 199
 - m_width, 199
 - m_window, 200
 - resetWindowResizedFlag, 198
 - setFullscreen, 198
 - wasWindowResized, 199
 - Window, 196
- vengine, 1
- vertices
 - ven::Model::Builder, 48
- view
 - ven::GlobalUbo, 110
- wasWindowResized
 - ven::Window, 199
- WHITE
 - ven::Colors, 67
- WHITE_V
 - ven::Colors, 67
- width
 - ven::SwapChain, 182
- Window
 - ven::Window, 196
- Window.hpp
 - GLFW_INCLUDE_VULKAN, 250
- writeBuffer
 - ven::DescriptorWriter, 78
- writelImage
 - ven::DescriptorWriter, 79
- writeToBuffer
 - ven::Buffer, 36
- writeToIndex
 - ven::Buffer, 36
- YELLOW
 - ven::Colors, 67
- YELLOW_V
 - ven::Colors, 67