

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	External Libraries	2
1.1.4	Usage	2
1.1.4.1	Build	2
1.1.4.2	Run	2
1.1.5	Key Bindings	2
1.1.6	Documentation	3
1.1.7	Commit Norms	3
1.1.8	License	3
1.1.9	Acknowledgements	3
2	Namespace Index	5
2.1	Namespace List	5
3	Hierarchical Index	7
3.1	Class Hierarchy	7
4	Class Index	9
4.1	Class List	9
5	File Index	11
5.1	File List	11
6	Namespace Documentation	13
6.1	ven Namespace Reference	13
6.1.1	Typedef Documentation	15
6.1.1.1	TimePoint	15
6.1.2	Enumeration Type Documentation	15
6.1.2.1	ENGINE_STATE	15
6.1.2.2	GUI_STATE	15
6.1.3	Function Documentation	16
6.1.3.1	hashCombine()	16
6.1.4	Variable Documentation	17
6.1.4.1	COLOR_MAX	17
6.1.4.2	DEFAULT_AMBIENT_LIGHT_COLOR	17
6.1.4.3	DEFAULT_AMBIENT_LIGHT_INTENSITY	17
6.1.4.4	DEFAULT_CLEAR_COLOR	17
6.1.4.5	DEFAULT_CLEAR_DEPTH	17
6.1.4.6	DEFAULT_FAR	17
6.1.4.7	DEFAULT_FOV	18

6.1.4.8	DEFAULT_HEIGHT	18
6.1.4.9	DEFAULT_KEY_MAPPINGS	18
6.1.4.10	DEFAULT_LIGHT_COLOR	18
6.1.4.11	DEFAULT_LIGHT_INTENSITY	18
6.1.4.12	DEFAULT_LIGHT_RADIUS	18
6.1.4.13	DEFAULT_LOOK_SPEED	18
6.1.4.14	DEFAULT_MAX_SETS	19
6.1.4.15	DEFAULT_MOVE_SPEED	19
6.1.4.16	DEFAULT_NEAR	19
6.1.4.17	DEFAULT_POSITION	19
6.1.4.18	DEFAULT_ROTATION	19
6.1.4.19	DEFAULT_SHININESS	19
6.1.4.20	DEFAULT_TITLE	20
6.1.4.21	DEFAULT_WIDTH	20
6.1.4.22	DESCRIPTOR_COUNT	20
6.1.4.23	EPSILON	20
6.1.4.24	MAX_FRAMES_IN_FLIGHT	20
6.1.4.25	MAX_LIGHTS	20
6.1.4.26	MAX_OBJECTS	20
6.1.4.27	SHADERS_BIN_PATH	20
7	Class Documentation	21
7.1	ven::ARenderSystemBase Class Reference	21
7.1.1	Detailed Description	23
7.1.2	Constructor & Destructor Documentation	23
7.1.2.1	ARenderSystemBase()	23
7.1.2.2	~ARenderSystemBase()	23
7.1.3	Member Function Documentation	24
7.1.3.1	createPipeline()	24
7.1.3.2	createPipelineLayout()	24
7.1.3.3	getDevice()	25
7.1.3.4	getPipelineLayout()	25
7.1.3.5	getShaders()	26
7.1.3.6	render()	26
7.1.4	Member Data Documentation	26
7.1.4.1	m_device	26
7.1.4.2	m_pipelineLayout	27
7.1.4.3	m_shaders	27
7.1.4.4	renderSystemLayout	27
7.2	ven::Buffer Class Reference	27
7.2.1	Detailed Description	30
7.2.2	Constructor & Destructor Documentation	30

7.2.2.1 Buffer() [1/2]	30
7.2.2.2 ~Buffer()	30
7.2.2.3 Buffer() [2/2]	30
7.2.3 Member Function Documentation	30
7.2.3.1 descriptorInfo()	30
7.2.3.2 descriptorInfoForIndex()	31
7.2.3.3 flush()	32
7.2.3.4 flushIndex()	32
7.2.3.5 getAlignment()	33
7.2.3.6 getAlignmentSize()	33
7.2.3.7 getBuffer()	34
7.2.3.8 getBufferSize()	34
7.2.3.9 getInstanceCount()	34
7.2.3.10 getInstanceSize()	34
7.2.3.11 getMappedMemory()	34
7.2.3.12 getMemoryPropertyFlags()	34
7.2.3.13 getUsageFlags()	35
7.2.3.14 invalidate()	35
7.2.3.15 invalidateIndex()	35
7.2.3.16 map()	36
7.2.3.17 operator=()	37
7.2.3.18 unmap()	37
7.2.3.19 writeToBuffer()	37
7.2.3.20 writeToIndex()	37
7.2.4 Member Data Documentation	38
7.2.4.1 m_alignmentSize	38
7.2.4.2 m_buffer	38
7.2.4.3 m_bufferSize	38
7.2.4.4 m_device	39
7.2.4.5 m_instanceCount	39
7.2.4.6 m_instanceSize	39
7.2.4.7 m_mapped	39
7.2.4.8 m_memory	39
7.2.4.9 m_memoryPropertyFlags	39
7.2.4.10 m_usageFlags	40
7.3 ven::DescriptorPool::Builder Class Reference	40
7.3.1 Detailed Description	42
7.3.2 Constructor & Destructor Documentation	42
7.3.2.1 Builder()	42
7.3.3 Member Function Documentation	42
7.3.3.1 addPoolSize()	42
7.3.3.2 build()	43

7.3.3.3 setMaxSets()	43
7.3.3.4 setPoolFlags()	44
7.3.4 Member Data Documentation	44
7.3.4.1 m_device	44
7.3.4.2 m_maxSets	44
7.3.4.3 m_poolFlags	44
7.3.4.4 m_poolSizes	45
7.4 ven::DescriptorSetLayout::Builder Class Reference	45
7.4.1 Detailed Description	47
7.4.2 Constructor & Destructor Documentation	47
7.4.2.1 Builder()	47
7.4.3 Member Function Documentation	47
7.4.3.1 addBinding()	47
7.4.3.2 build()	48
7.4.4 Member Data Documentation	48
7.4.4.1 m_bindings	48
7.4.4.2 m_device	48
7.5 ven::Model::Builder Struct Reference	49
7.5.1 Detailed Description	50
7.5.2 Member Function Documentation	50
7.5.2.1 loadModel()	50
7.5.2.2 processMesh()	50
7.5.2.3 processNode()	50
7.5.3 Member Data Documentation	50
7.5.3.1 indices	50
7.5.3.2 vertices	51
7.6 ven::Camera Class Reference	51
7.6.1 Detailed Description	53
7.6.2 Constructor & Destructor Documentation	53
7.6.2.1 Camera() [1/2]	53
7.6.2.2 ~Camera()	53
7.6.2.3 Camera() [2/2]	54
7.6.3 Member Function Documentation	54
7.6.3.1 getFar()	54
7.6.3.2 getFov()	54
7.6.3.3 getInverseView()	55
7.6.3.4 getLookSpeed()	55
7.6.3.5 getMoveSpeed()	55
7.6.3.6 getNear()	56
7.6.3.7 getProjection()	56
7.6.3.8 getView()	56
7.6.3.9 operator=()	56

7.6.3.10 setFar()	57
7.6.3.11 setFov()	57
7.6.3.12 setLookSpeed()	58
7.6.3.13 setMoveSpeed()	58
7.6.3.14 setNear()	59
7.6.3.15 setOrthographicProjection()	59
7.6.3.16 setPerspectiveProjection()	59
7.6.3.17 setViewDirection()	59
7.6.3.18 setViewTarget()	60
7.6.3.19 setViewXYZ()	60
7.6.4 Member Data Documentation	60
7.6.4.1 m_far	60
7.6.4.2 m_fov	60
7.6.4.3 m_inverseViewMatrix	60
7.6.4.4 m_lookSpeed	60
7.6.4.5 m_moveSpeed	61
7.6.4.6 m_near	61
7.6.4.7 m_projectionMatrix	61
7.6.4.8 m_viewMatrix	61
7.6.4.9 transform	61
7.7 ven::Clock Class Reference	62
7.7.1 Detailed Description	63
7.7.2 Constructor & Destructor Documentation	63
7.7.2.1 Clock() [1/2]	63
7.7.2.2 ~Clock()	63
7.7.2.3 Clock() [2/2]	63
7.7.3 Member Function Documentation	63
7.7.3.1 getDeltaTime()	63
7.7.3.2 operator=()	64
7.7.3.3 resume()	64
7.7.3.4 start()	64
7.7.3.5 stop()	64
7.7.3.6 update()	64
7.7.4 Member Data Documentation	65
7.7.4.1 m_deltaTime	65
7.7.4.2 m_isStopped	65
7.7.4.3 m_startTime	65
7.7.4.4 m_stopTime	65
7.8 ven::Colors Class Reference	66
7.8.1 Detailed Description	68
7.8.2 Member Data Documentation	68
7.8.2.1 AQUA_3	68

7.8.2.2 AQUA_4	68
7.8.2.3 AQUA_V	68
7.8.2.4 BLACK_3	68
7.8.2.5 BLACK_4	68
7.8.2.6 BLACK_V	68
7.8.2.7 BLUE_3	69
7.8.2.8 BLUE_4	69
7.8.2.9 BLUE_V	69
7.8.2.10 COLOR_PRESETS_3	69
7.8.2.11 COLOR_PRESETS_4	70
7.8.2.12 COLOR_PRESETS_VK	70
7.8.2.13 CYAN_3	70
7.8.2.14 CYAN_4	71
7.8.2.15 CYAN_V	71
7.8.2.16 FUCHSIA_3	71
7.8.2.17 FUCHSIA_4	71
7.8.2.18 FUCHSIA_V	71
7.8.2.19 GRAY_3	71
7.8.2.20 GRAY_4	71
7.8.2.21 GRAY_V	72
7.8.2.22 GREEN_3	72
7.8.2.23 GREEN_4	72
7.8.2.24 GREEN_V	72
7.8.2.25 LIME_3	72
7.8.2.26 LIME_4	72
7.8.2.27 LIME_V	72
7.8.2.28 MAGENTA_3	73
7.8.2.29 MAGENTA_4	73
7.8.2.30 MAGENTA_V	73
7.8.2.31 MAROON_3	73
7.8.2.32 MAROON_4	73
7.8.2.33 MAROON_V	73
7.8.2.34 NAVY_3	73
7.8.2.35 NAVY_4	74
7.8.2.36 NAVY_V	74
7.8.2.37 NIGHT_BLUE_3	74
7.8.2.38 NIGHT_BLUE_4	74
7.8.2.39 NIGHT_BLUE_V	74
7.8.2.40 OLIVE_3	74
7.8.2.41 OLIVE_4	74
7.8.2.42 OLIVE_V	75
7.8.2.43 RED_3	75

7.8.2.44 RED_4	75
7.8.2.45 RED_V	75
7.8.2.46 SILVER_3	75
7.8.2.47 SILVER_4	75
7.8.2.48 SILVER_V	75
7.8.2.49 SKY_BLUE_3	76
7.8.2.50 SKY_BLUE_4	76
7.8.2.51 SKY_BLUE_V	76
7.8.2.52 SUNSET_3	76
7.8.2.53 SUNSET_4	76
7.8.2.54 SUNSET_V	76
7.8.2.55 TEAL_3	76
7.8.2.56 TEAL_4	77
7.8.2.57 TEAL_V	77
7.8.2.58 WHITE_3	77
7.8.2.59 WHITE_4	77
7.8.2.60 WHITE_V	77
7.8.2.61 YELLOW_3	77
7.8.2.62 YELLOW_4	77
7.8.2.63 YELLOW_V	78
7.9 ven::DescriptorPool Class Reference	78
7.9.1 Detailed Description	80
7.9.2 Constructor & Destructor Documentation	80
7.9.2.1 DescriptorPool() [1/2]	80
7.9.2.2 ~DescriptorPool()	81
7.9.2.3 DescriptorPool() [2/2]	81
7.9.3 Member Function Documentation	81
7.9.3.1 allocateDescriptor()	81
7.9.3.2 freeDescriptors()	81
7.9.3.3 getDescriptorPool()	82
7.9.3.4 operator=()	82
7.9.3.5 resetPool()	82
7.9.4 Friends And Related Symbol Documentation	82
7.9.4.1 DescriptorWriter	82
7.9.5 Member Data Documentation	82
7.9.5.1 m_descriptorPool	82
7.9.5.2 m_device	83
7.10 ven::DescriptorSetLayout Class Reference	83
7.10.1 Detailed Description	85
7.10.2 Constructor & Destructor Documentation	85
7.10.2.1 DescriptorSetLayout() [1/2]	85
7.10.2.2 ~DescriptorSetLayout()	86

7.10.2.3 DescriptorSetLayout() [2/2]	86
7.10.3 Member Function Documentation	86
7.10.3.1 getDescriptorSetLayout()	86
7.10.3.2 operator=()	86
7.10.4 Friends And Related Symbol Documentation	86
7.10.4.1 DescriptorWriter	86
7.10.5 Member Data Documentation	87
7.10.5.1 m_bindings	87
7.10.5.2 m_descriptorSetLayout	87
7.10.5.3 m_device	87
7.11 ven::DescriptorWriter Class Reference	87
7.11.1 Detailed Description	89
7.11.2 Constructor & Destructor Documentation	89
7.11.2.1 DescriptorWriter() [1/2]	89
7.11.2.2 ~DescriptorWriter()	89
7.11.2.3 DescriptorWriter() [2/2]	89
7.11.3 Member Function Documentation	89
7.11.3.1 build()	89
7.11.3.2 operator=()	90
7.11.3.3 overwrite()	90
7.11.3.4 writeBuffer()	90
7.11.3.5 writImage()	90
7.11.4 Member Data Documentation	91
7.11.4.1 m_pool	91
7.11.4.2 m_setLayout	91
7.11.4.3 m_writes	91
7.12 ven::Device Class Reference	91
7.12.1 Detailed Description	94
7.12.2 Constructor & Destructor Documentation	94
7.12.2.1 Device() [1/2]	94
7.12.2.2 ~Device()	95
7.12.2.3 Device() [2/2]	95
7.12.3 Member Function Documentation	95
7.12.3.1 beginSingleTimeCommands()	95
7.12.3.2 checkDeviceExtensionSupport()	95
7.12.3.3 checkValidationLayerSupport()	95
7.12.3.4 copyBuffer()	96
7.12.3.5 copyBufferToImage()	96
7.12.3.6 createBuffer()	96
7.12.3.7 createCommandPool()	97
7.12.3.8 createImageWithInfo()	97
7.12.3.9 createInstance()	98

7.12.3.10 createLogicalDevice()	98
7.12.3.11 createSurface()	98
7.12.3.12 device()	99
7.12.3.13 endSingleTimeCommands()	100
7.12.3.14 findMemoryType()	100
7.12.3.15 findPhysicalQueueFamilies()	101
7.12.3.16 findQueueFamilies()	101
7.12.3.17 findSupportedFormat()	102
7.12.3.18 getCommandPool()	102
7.12.3.19 getGraphicsQueue()	102
7.12.3.20 getPhysicalDevice()	102
7.12.3.21 getProperties()	103
7.12.3.22 getRequiredExtensions()	103
7.12.3.23 getSwapChainSupport()	103
7.12.3.24 graphicsQueue()	104
7.12.3.25 hasGlfwRequiredInstanceExtensions()	104
7.12.3.26 isDeviceSuitable()	104
7.12.3.27 operator=()	104
7.12.3.28 pickPhysicalDevice()	105
7.12.3.29 populateDebugMessengerCreateInfo()	105
7.12.3.30 presentQueue()	105
7.12.3.31 querySwapChainSupport()	106
7.12.3.32 setupDebugMessenger()	106
7.12.3.33 surface()	107
7.12.3.34 transitionImageLayout()	107
7.12.4 Member Data Documentation	107
7.12.4.1 enableValidationLayers	107
7.12.4.2 m_commandPool	107
7.12.4.3 m_debugMessenger	107
7.12.4.4 m_device	107
7.12.4.5 m_deviceExtensions	108
7.12.4.6 m_graphicsQueue	108
7.12.4.7 m_instance	108
7.12.4.8 m_physicalDevice	108
7.12.4.9 m_presentQueue	108
7.12.4.10 m_properties	108
7.12.4.11 m_surface	109
7.12.4.12 m_validationLayers	109
7.12.4.13 m_window	109
7.13 ven::Engine Class Reference	109
7.13.1 Detailed Description	111
7.13.2 Constructor & Destructor Documentation	111

7.13.2.1 Engine() [1/2]	111
7.13.2.2 ~Engine()	112
7.13.2.3 Engine() [2/2]	112
7.13.3 Member Function Documentation	112
7.13.3.1 createInstance()	112
7.13.3.2 createSurface()	113
7.13.3.3 loadObjects()	113
7.13.3.4 mainLoop()	114
7.13.3.5 operator=()	115
7.13.4 Member Data Documentation	115
7.13.4.1 framePools	115
7.13.4.2 m_device	115
7.13.4.3 m_globalPool	115
7.13.4.4 m_gui	115
7.13.4.5 m_instance	115
7.13.4.6 m_renderer	116
7.13.4.7 m_sceneManager	116
7.13.4.8 m_state	116
7.13.4.9 m_surface	116
7.13.4.10 m_window	116
7.14 ven::EventManager Class Reference	117
7.14.1 Detailed Description	118
7.14.2 Constructor & Destructor Documentation	118
7.14.2.1 EventManager() [1/2]	118
7.14.2.2 ~EventManager()	118
7.14.2.3 EventManager() [2/2]	118
7.14.3 Member Function Documentation	118
7.14.3.1 handleEvents()	118
7.14.3.2 isKeyJustPressed()	119
7.14.3.3 moveCamera()	119
7.14.3.4 operator=()	119
7.14.3.5 processKeyActions()	119
7.14.3.6 updateEngineState()	120
7.14.4 Member Data Documentation	120
7.14.4.1 m_keyState	120
7.15 ven::FrameInfo Struct Reference	120
7.15.1 Detailed Description	122
7.15.2 Member Data Documentation	122
7.15.2.1 camera	122
7.15.2.2 commandBuffer	122
7.15.2.3 frameDescriptorPool	122
7.15.2.4 frameIndex	122

7.15.2.5 frameTime	122
7.15.2.6 globalDescriptorSet	123
7.15.2.7 lights	123
7.15.2.8 objects	123
7.16 ven::Gui::funcs Struct Reference	123
7.16.1 Detailed Description	124
7.16.2 Member Function Documentation	124
7.16.2.1 IsLegacyNativeDupe()	124
7.17 ven::GlobalUbo Struct Reference	125
7.17.1 Detailed Description	125
7.17.2 Member Data Documentation	126
7.17.2.1 ambientLightColor	126
7.17.2.2 inverseView	126
7.17.2.3 numLights	126
7.17.2.4 pointLights	126
7.17.2.5 projection	126
7.17.2.6 view	126
7.18 ven::Gui Class Reference	127
7.18.1 Detailed Description	128
7.18.2 Constructor & Destructor Documentation	128
7.18.2.1 Gui() [1/2]	128
7.18.2.2 ~Gui()	128
7.18.2.3 Gui() [2/2]	129
7.18.3 Member Function Documentation	129
7.18.3.1 cameraSection()	129
7.18.3.2 cleanup()	130
7.18.3.3 devicePropertiesSection()	130
7.18.3.4 getState()	130
7.18.3.5 init()	131
7.18.3.6 initStyle()	132
7.18.3.7 inputsSection()	132
7.18.3.8 lightsSection()	132
7.18.3.9 objectsSection()	132
7.18.3.10 operator=()	132
7.18.3.11 render()	133
7.18.3.12 rendererSection()	133
7.18.3.13 renderFrameWindow()	134
7.18.3.14 setState()	134
7.18.4 Member Data Documentation	135
7.18.4.1 m_intensity	135
7.18.4.2 m_io	135
7.18.4.3 m_shininess	135

7.18.4.4 m_state	135
7.19 std::hash< ven::Model::Vertex > Struct Reference	135
7.19.1 Detailed Description	136
7.19.2 Member Function Documentation	136
7.19.2.1 operator()	136
7.20 ven::KeyAction Struct Reference	136
7.20.1 Detailed Description	137
7.20.2 Member Data Documentation	137
7.20.2.1 dir	137
7.20.2.2 key	137
7.20.2.3 value	137
7.21 ven::KeyMappings Struct Reference	138
7.21.1 Detailed Description	138
7.21.2 Member Data Documentation	139
7.21.2.1 lookDown	139
7.21.2.2 lookLeft	139
7.21.2.3 lookRight	139
7.21.2.4 lookUp	139
7.21.2.5 moveBackward	139
7.21.2.6 moveDown	139
7.21.2.7 moveForward	140
7.21.2.8 moveLeft	140
7.21.2.9 moveRight	140
7.21.2.10 moveUp	140
7.21.2.11 toggleGui	140
7.22 ven::Light Class Reference	141
7.22.1 Detailed Description	142
7.22.2 Member Typedef Documentation	142
7.22.2.1 Map	142
7.22.3 Constructor & Destructor Documentation	142
7.22.3.1 Light() [1/3]	142
7.22.3.2 ~Light()	143
7.22.3.3 Light() [2/3]	143
7.22.3.4 Light() [3/3]	143
7.22.4 Member Function Documentation	143
7.22.4.1 getId()	143
7.22.4.2 getName()	143
7.22.4.3 getShininess()	144
7.22.4.4 operator=() [1/2]	144
7.22.4.5 operator=() [2/2]	144
7.22.4.6 setName()	144
7.22.4.7 setShininess()	144

7.22.5 Member Data Documentation	144
7.22.5.1 color	144
7.22.5.2 m_lightId	145
7.22.5.3 m_name	145
7.22.5.4 m_shininess	145
7.22.5.5 transform	145
7.23 ven::LightPushConstantData Struct Reference	145
7.23.1 Detailed Description	146
7.23.2 Member Data Documentation	146
7.23.2.1 color	146
7.23.2.2 position	146
7.23.2.3 radius	146
7.24 ven::Model Class Reference	147
7.24.1 Detailed Description	148
7.24.2 Constructor & Destructor Documentation	148
7.24.2.1 Model() [1/2]	148
7.24.2.2 ~Model()	149
7.24.2.3 Model() [2/2]	149
7.24.3 Member Function Documentation	149
7.24.3.1 bind()	149
7.24.3.2 createIndexBuffer()	149
7.24.3.3 createModelFromFile()	150
7.24.3.4 createVertexBuffer()	150
7.24.3.5 draw()	151
7.24.3.6 operator=()	151
7.24.4 Member Data Documentation	151
7.24.4.1 m_device	151
7.24.4.2 m_hasIndexBuffer	151
7.24.4.3 m_indexBuffer	151
7.24.4.4 m_indexCount	152
7.24.4.5 m_vertexBuffer	152
7.24.4.6 m_vertexCount	152
7.25 ven::Object Class Reference	152
7.25.1 Detailed Description	154
7.25.2 Member Typedef Documentation	154
7.25.2.1 Map	154
7.25.3 Constructor & Destructor Documentation	154
7.25.3.1 Object() [1/3]	154
7.25.3.2 Object() [2/3]	155
7.25.3.3 Object() [3/3]	155
7.25.4 Member Function Documentation	155
7.25.4.1 getBufferInfo()	155

7.25.4.2 getDiffuseMap()	155
7.25.4.3 getId()	155
7.25.4.4 getModel()	155
7.25.4.5 getName()	156
7.25.4.6 operator=() [1/2]	156
7.25.4.7 operator=() [2/2]	156
7.25.4.8 setBufferInfo()	156
7.25.4.9 setDiffuseMap()	156
7.25.4.10 setModel()	156
7.25.4.11 setName()	157
7.25.5 Member Data Documentation	157
7.25.5.1 m_bufferInfo	157
7.25.5.2 m_diffuseMap	157
7.25.5.3 m_model	157
7.25.5.4 m_name	157
7.25.5.5 m_objId	157
7.25.5.6 transform	158
7.26 ven::ObjectBufferData Struct Reference	158
7.26.1 Detailed Description	158
7.26.2 Member Data Documentation	158
7.26.2.1 modelMatrix	158
7.26.2.2 normalMatrix	159
7.27 ven::ObjectPushConstantData Struct Reference	159
7.27.1 Detailed Description	159
7.27.2 Member Data Documentation	159
7.27.2.1 modelMatrix	159
7.27.2.2 normalMatrix	160
7.28 ven::ObjectRenderSystem Class Reference	160
7.28.1 Detailed Description	162
7.28.2 Constructor & Destructor Documentation	162
7.28.2.1 ObjectRenderSystem() [1/2]	162
7.28.2.2 ObjectRenderSystem() [2/2]	163
7.28.3 Member Function Documentation	163
7.28.3.1 operator=()	163
7.28.3.2 render()	163
7.29 ven::PipelineConfigInfo Struct Reference	164
7.29.1 Detailed Description	165
7.29.2 Constructor & Destructor Documentation	165
7.29.2.1 PipelineConfigInfo() [1/2]	165
7.29.2.2 PipelineConfigInfo() [2/2]	165
7.29.3 Member Function Documentation	165
7.29.3.1 operator=()	165

7.29.4 Member Data Documentation	165
7.29.4.1 attributeDescriptions	165
7.29.4.2 bindingDescriptions	166
7.29.4.3 colorBlendAttachment	166
7.29.4.4 colorBlendInfo	166
7.29.4.5 depthStencilInfo	166
7.29.4.6 dynamicStateEnables	166
7.29.4.7 dynamicStateInfo	166
7.29.4.8 inputAssemblyInfo	167
7.29.4.9 multisampleInfo	167
7.29.4.10 pipelineLayout	167
7.29.4.11 rasterizationInfo	167
7.29.4.12 renderPass	167
7.29.4.13 subpass	167
7.30 ven::PointLightData Struct Reference	168
7.30.1 Detailed Description	168
7.30.2 Member Data Documentation	168
7.30.2.1 color	168
7.30.2.2 padding	168
7.30.2.3 position	169
7.30.2.4 shininess	169
7.31 ven::PointLightRenderSystem Class Reference	169
7.31.1 Detailed Description	171
7.31.2 Constructor & Destructor Documentation	171
7.31.2.1 PointLightRenderSystem() [1/2]	171
7.31.2.2 PointLightRenderSystem() [2/2]	172
7.31.3 Member Function Documentation	172
7.31.3.1 operator=()	172
7.31.3.2 render()	172
7.32 ven::QueueFamilyIndices Struct Reference	173
7.32.1 Detailed Description	173
7.32.2 Member Function Documentation	173
7.32.2.1 isComplete()	173
7.32.3 Member Data Documentation	174
7.32.3.1 graphicsFamily	174
7.32.3.2 graphicsFamilyHasValue	174
7.32.3.3 presentFamily	174
7.32.3.4 presentFamilyHasValue	174
7.33 ven::Renderer Class Reference	175
7.33.1 Detailed Description	176
7.33.2 Constructor & Destructor Documentation	177
7.33.2.1 Renderer() [1/2]	177

7.33.2.2 ~Renderer()	177
7.33.2.3 Renderer() [2/2]	177
7.33.3 Member Function Documentation	178
7.33.3.1 beginFrame()	178
7.33.3.2 beginSwapChainRenderPass()	178
7.33.3.3 createCommandBuffers()	178
7.33.3.4 endFrame()	179
7.33.3.5 endSwapChainRenderPass()	179
7.33.3.6 freeCommandBuffers()	179
7.33.3.7 getAspectRatio()	179
7.33.3.8 getClearColor()	180
7.33.3.9 getCurrentCommandBuffer()	180
7.33.3.10 getFrameIndex()	181
7.33.3.11 getSwapChainRenderPass()	181
7.33.3.12 getWindow()	181
7.33.3.13 isFrameInProgress()	182
7.33.3.14 operator=()	182
7.33.3.15 recreateSwapChain()	182
7.33.3.16 setClearColor()	183
7.33.4 Member Data Documentation	183
7.33.4.1 m_clearValues	183
7.33.4.2 m_commandBuffers	183
7.33.4.3 m_currentFrameIndex	183
7.33.4.4 m_currentImageIndex	184
7.33.4.5 m_device	184
7.33.4.6 m_isFrameStarted	184
7.33.4.7 m_swapChain	184
7.33.4.8 m_window	184
7.34 ven::SceneManager Class Reference	185
7.34.1 Detailed Description	186
7.34.2 Constructor & Destructor Documentation	186
7.34.2.1 SceneManager() [1/3]	186
7.34.2.2 SceneManager() [2/3]	187
7.34.2.3 SceneManager() [3/3]	187
7.34.3 Member Function Documentation	187
7.34.3.1 createLight()	187
7.34.3.2 createObject()	187
7.34.3.3 getBufferInfoForObject()	187
7.34.3.4 getLights()	188
7.34.3.5 getObjects()	188
7.34.3.6 getUboBuffers()	188
7.34.3.7 operator=() [1/2]	188

7.34.3.8 operator=() [2/2]	188
7.34.3.9 updateBuffer()	188
7.34.4 Member Data Documentation	189
7.34.4.1 m_currentLightId	189
7.34.4.2 m_currentObjId	189
7.34.4.3 m_lights	189
7.34.4.4 m_objects	189
7.34.4.5 m_textureDefault	189
7.34.4.6 m_uboBuffers	189
7.35 ven::Shaders Class Reference	190
7.35.1 Detailed Description	191
7.35.2 Constructor & Destructor Documentation	192
7.35.2.1 Shaders() [1/2]	192
7.35.2.2 ~Shaders()	192
7.35.2.3 Shaders() [2/2]	192
7.35.3 Member Function Documentation	193
7.35.3.1 bind()	193
7.35.3.2 createGraphicsPipeline()	193
7.35.3.3 createShaderModule()	193
7.35.3.4 defaultPipelineConfigInfo()	194
7.35.3.5 operator=()	194
7.35.3.6 readFile()	194
7.35.4 Member Data Documentation	195
7.35.4.1 m_device	195
7.35.4.2 m_fragShaderModule	195
7.35.4.3 m_graphicsPipeline	195
7.35.4.4 m_vertShaderModule	195
7.36 ven::SwapChain Class Reference	196
7.36.1 Detailed Description	198
7.36.2 Constructor & Destructor Documentation	198
7.36.2.1 SwapChain() [1/3]	198
7.36.2.2 SwapChain() [2/3]	198
7.36.2.3 ~SwapChain()	199
7.36.2.4 SwapChain() [3/3]	199
7.36.3 Member Function Documentation	199
7.36.3.1 acquireNextImage()	199
7.36.3.2 chooseSwapExtent()	199
7.36.3.3 chooseSwapPresentMode()	199
7.36.3.4 chooseSwapSurfaceFormat()	200
7.36.3.5 compareSwapFormats()	200
7.36.3.6 createDepthResources()	200
7.36.3.7 createFrameBuffers()	200

7.36.3.8 createImageViews()	200
7.36.3.9 createRenderPass()	200
7.36.3.10 createSwapChain()	200
7.36.3.11 createSyncObjects()	201
7.36.3.12 extentAspectRatio()	201
7.36.3.13 findDepthFormat()	201
7.36.3.14 getFrameBuffer()	201
7.36.3.15 getImageView()	201
7.36.3.16 getRenderPass()	201
7.36.3.17 getSwapChainExtent()	202
7.36.3.18 getSwapChainImageFormat()	202
7.36.3.19 height()	202
7.36.3.20 imageCount()	202
7.36.3.21 init()	202
7.36.3.22 operator=()	203
7.36.3.23 submitCommandBuffers()	203
7.36.3.24 width()	203
7.36.4 Member Data Documentation	203
7.36.4.1 m_currentFrame	203
7.36.4.2 m_depthImageMemory	203
7.36.4.3 m_depthImages	203
7.36.4.4 m_depthImageViews	204
7.36.4.5 m_device	204
7.36.4.6 m_imageAvailableSemaphores	204
7.36.4.7 m_imagesInFlight	204
7.36.4.8 m_inFlightFences	204
7.36.4.9 m_oldSwapChain	204
7.36.4.10 m_renderFinishedSemaphores	205
7.36.4.11 m_renderPass	205
7.36.4.12 m_swapChain	205
7.36.4.13 m_swapChainDepthFormat	205
7.36.4.14 m_swapChainExtent	205
7.36.4.15 m_swapChainFrameBuffers	205
7.36.4.16 m_swapChainImageFormat	206
7.36.4.17 m_swapChainImages	206
7.36.4.18 m_swapChainImageViews	206
7.36.4.19 m_windowExtent	206
7.37 ven::SwapChainSupportDetails Struct Reference	206
7.37.1 Detailed Description	207
7.37.2 Member Data Documentation	207
7.37.2.1 capabilities	207
7.37.2.2 formats	207

7.37.2.3 presentModes	207
7.38 ven::Texture Class Reference	208
7.38.1 Detailed Description	209
7.38.2 Constructor & Destructor Documentation	210
7.38.2.1 Texture() [1/3]	210
7.38.2.2 Texture() [2/3]	210
7.38.2.3 ~Texture()	211
7.38.2.4 Texture() [3/3]	211
7.38.3 Member Function Documentation	211
7.38.3.1 createTextureFromFile()	211
7.38.3.2 createTextureImage()	212
7.38.3.3 createTextureImageView()	212
7.38.3.4 createTextureSampler()	212
7.38.3.5 getExtent()	213
7.38.3.6 getFormat()	213
7.38.3.7 getImage()	213
7.38.3.8 getImageInfo()	213
7.38.3.9 getImageLayout()	213
7.38.3.10 getImageView()	214
7.38.3.11 imageView()	214
7.38.3.12 operator=()	214
7.38.3.13 sampler()	214
7.38.3.14 transitionLayout()	214
7.38.3.15 updateDescriptor()	214
7.38.4 Member Data Documentation	215
7.38.4.1 m_descriptor	215
7.38.4.2 m_device	215
7.38.4.3 m_extent	215
7.38.4.4 m_format	215
7.38.4.5 m_layerCount	215
7.38.4.6 m_mipLevels	215
7.38.4.7 m_textureImage	216
7.38.4.8 m_textureImageMemory	216
7.38.4.9 m_textureImageView	216
7.38.4.10 m_textureLayout	216
7.38.4.11 m_textureSampler	216
7.39 ven::Transform3D Class Reference	217
7.39.1 Detailed Description	217
7.39.2 Member Function Documentation	218
7.39.2.1 normalMatrix()	218
7.39.2.2 transformMatrix()	218
7.39.3 Member Data Documentation	218

7.39.3.1 rotation	218
7.39.3.2 scale	219
7.39.3.3 translation	219
7.40 ven::Model::Vertex Struct Reference	219
7.40.1 Detailed Description	220
7.40.2 Member Function Documentation	220
7.40.2.1 getAttributeDescriptions()	220
7.40.2.2 getBindingDescriptions()	220
7.40.2.3 operator==()	221
7.40.3 Member Data Documentation	221
7.40.3.1 color	221
7.40.3.2 normal	221
7.40.3.3 position	221
7.40.3.4 uv	221
7.41 ven::Window Class Reference	222
7.41.1 Detailed Description	223
7.41.2 Constructor & Destructor Documentation	223
7.41.2.1 Window() [1/2]	223
7.41.2.2 ~Window()	223
7.41.2.3 Window() [2/2]	223
7.41.3 Member Function Documentation	224
7.41.3.1 createWindow()	224
7.41.3.2 createWindowSurface()	224
7.41.3.3 framebufferResizeCallback()	225
7.41.3.4 getExtent()	225
7.41.3.5 getGLFWWindow()	226
7.41.3.6 operator=()	226
7.41.3.7 resetWindowResizedFlag()	226
7.41.3.8 setFullscreen()	226
7.41.3.9 wasWindowResized()	227
7.41.4 Member Data Documentation	227
7.41.4.1 m_framebufferResized	227
7.41.4.2 m_height	227
7.41.4.3 m_width	227
7.41.4.4 m_window	227
8 File Documentation	229
8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag File Reference	229
8.2 fragment_point_light.frag	229
8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag File Reference	229
8.4 fragment_shader.frag	229
8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert File Reference	230

8.6 vertex_point_light.vert	230
8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference	231
8.8 vertex_shader.vert	231
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference	232
8.9.1 Detailed Description	233
8.10 Buffer.hpp	233
8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference	235
8.11.1 Detailed Description	237
8.12 Camera.hpp	237
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp File Reference	238
8.13.1 Detailed Description	239
8.14 DescriptorPool.hpp	239
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference	240
8.15.1 Detailed Description	241
8.16 DescriptorSetLayout.hpp	241
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference	242
8.17.1 Detailed Description	243
8.18 DescriptorWriter.hpp	243
8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference	244
8.19.1 Detailed Description	245
8.20 Device.hpp	245
8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference	246
8.21.1 Detailed Description	248
8.22 Engine.hpp	248
8.23 /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp File Reference	249
8.23.1 Detailed Description	250
8.24 EventManager.hpp	250
8.25 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference	251
8.25.1 Detailed Description	252
8.26 FrameInfo.hpp	252
8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp File Reference	252
8.27.1 Detailed Description	254
8.28 Gui.hpp	254
8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp File Reference	255
8.29.1 Detailed Description	256
8.30 Light.hpp	256
8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference	257
8.31.1 Detailed Description	258
8.32 Model.hpp	258
8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference	259
8.33.1 Detailed Description	260

8.34 Object.hpp	260
8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference	261
8.35.1 Detailed Description	262
8.36 Renderer.hpp	263
8.37 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp File Reference	264
8.37.1 Detailed Description	265
8.38 ARenderSystemBase.hpp	265
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp File Reference	265
8.39.1 Detailed Description	266
8.40 ObjectRenderSystem.hpp	267
8.41 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp File Reference	267
8.41.1 Detailed Description	268
8.42 PointLightRenderSystem.hpp	268
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/SceneManager.hpp File Reference	269
8.43.1 Detailed Description	270
8.44 SceneManager.hpp	270
8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference	271
8.45.1 Detailed Description	273
8.46 Shaders.hpp	273
8.47 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference	274
8.47.1 Detailed Description	275
8.48 SwapChain.hpp	275
8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Texture.hpp File Reference	277
8.49.1 Detailed Description	278
8.50 Texture.hpp	278
8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3D.hpp File Reference	279
8.51.1 Detailed Description	279
8.52 Transform3D.hpp	280
8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp File Reference	280
8.53.1 Detailed Description	281
8.54 Clock.hpp	281
8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp File Reference	282
8.56 Colors.hpp	283
8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp File Reference	285
8.58 HashCombine.hpp	286
8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference	286
8.59.1 Detailed Description	288
8.59.2 Macro Definition Documentation	288
8.59.2.1 GLFW_INCLUDE_VULKAN	288
8.60 Window.hpp	288

8.61 /home/runner/work/VEngine/VEngine/README.md File Reference	289
8.62 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference	289
8.63 buffer.cpp	289
8.64 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference	290
8.65 camera.cpp	291
8.66 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference	293
8.67 descriptorPool.cpp	293
8.68 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference	294
8.69 descriptorSetLayout.cpp	294
8.70 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference	295
8.71 descriptorWriter.cpp	295
8.72 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference	296
8.72.1 Function Documentation	297
8.72.1.1 CreateDebugUtilsMessengerEXT()	297
8.72.1.2 debugCallback()	298
8.72.1.3 DestroyDebugUtilsMessengerEXT()	298
8.73 device.cpp	299
8.74 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference	306
8.75 engine.cpp	306
8.76 /home/runner/work/VEngine/VEngine/src/eventManager.cpp File Reference	309
8.76.1 Macro Definition Documentation	309
8.76.1.1 GLM_ENABLE_EXPERIMENTAL	309
8.77 eventManager.cpp	309
8.78 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference	310
8.79 init.cpp	311
8.80 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference	312
8.81 render.cpp	312
8.82 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference	317
8.82.1 Function Documentation	317
8.82.1.1 main()	317
8.83 main.cpp	318
8.84 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference	318
8.84.1 Macro Definition Documentation	319
8.84.1.1 GLM_ENABLE_EXPERIMENTAL	319
8.85 model.cpp	319
8.86 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference	322
8.87 renderer.cpp	322
8.88 /home/runner/work/VEngine/VEngine/src/sceneManager.cpp File Reference	324
8.89 sceneManager.cpp	324
8.90 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference	325
8.91 shaders.cpp	326
8.92 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference	328

8.93 swapChain.cpp	329
8.94 /home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp File Reference	334
8.95 objectRenderSystem.cpp	334
8.96 /home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp File Reference	335
8.97 pointLightRenderSystem.cpp	335
8.98 /home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp File Reference	336
8.99 renderSystemBase.cpp	336
8.100 /home/runner/work/VEngine/VEngine/src/texture.cpp File Reference	337
8.100.1 Macro Definition Documentation	337
8.100.1.1 STB_IMAGE_IMPLEMENTATION	337
8.101 texture.cpp	338
8.102 /home/runner/work/VEngine/VEngine/src/utlis/clock.cpp File Reference	342
8.103 clock.cpp	342
8.104 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference	342
8.105 window.cpp	343
Index	345

Chapter 1

vengine

1.1 VEngine - Vulkan Graphics Engine

WORK IN PROGRESS!

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

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

1.1.1 Features

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

1.1.1.1 Planned Features:

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

1.1.2 Prerequisites

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

- **CMake 3.27**
- **C++20**
- **Vulkan SDK**
- **X11**

if you are using a Debian-based distribution, you can install the required packages using the following command:

```
$> ./tools/install-dependencies.sh build
```

1.1.3 External Libraries

- **Assimp**: Open Asset Import Library to load various 3D model formats into the engine.
- **Doxygen Awesome CSS**: A custom CSS theme for Doxygen documentation.
- **GLFW**: For creating windows, receiving input, and managing OpenGL and Vulkan contexts.
- **GLM**: A header-only C++ mathematics library for 3D transformations, vectors, and matrices, compatible with OpenGL and Vulkan.
- **ImGui**: Immediate Mode Graphical User Interface for real-time debugging and tool development.
- **stb**: A set of single-file public domain libraries for graphics, image loading, and more.

These libraries are included directly into the project to simplify dependency management. Be sure to initialize and update the submodules when cloning the repository:

```
$> git submodule update --init --recursive
```

1.1.4 Usage

1.1.4.1 Build

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

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

1.1.4.2 Run

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

1.1.5 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

1.1.6 Documentation

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

1.1.7 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

1.1.8 License

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

1.1.9 Acknowledgements

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

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ven::ARenderSystemBase	21
ven::ObjectRenderSystem	160
ven::PointLightRenderSystem	169
ven::Buffer	27
ven::DescriptorPool::Builder	40
ven::DescriptorSetLayout::Builder	45
ven::Model::Builder	49
ven::Camera	51
ven::Clock	62
ven::Colors	66
ven::DescriptorPool	78
ven::DescriptorSetLayout	83
ven::DescriptorWriter	87
ven::Device	91
ven::Engine	109
ven::EventManager	117
ven::FrameInfo	120
ven::Gui::funcs	123
ven::GlobalUbo	125
ven::Gui	127
std::hash< ven::Model::Vertex >	135
ven::KeyAction	136
ven::KeyMappings	138
ven::Light	141
ven::LightPushConstantData	145
ven::Model	147
ven::Object	152
ven::ObjectBufferData	158
ven::ObjectPushConstantData	159
ven::PipelineConfigInfo	164
ven::PointLightData	168
ven::QueueFamilyIndices	173
ven::Renderer	175
ven::SceneManager	185
ven::Shaders	190

ven::SwapChain	196
ven::SwapChainSupportDetails	206
ven::Texture	208
ven::Transform3D	217
ven::Model::Vertex	219
ven::Window	222

Chapter 4

Class Index

4.1 Class List

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

ven::ARenderSystemBase	
Abstract class for render system base	21
ven::Buffer	
Class for buffer	27
ven::DescriptorPool::Builder	40
ven::DescriptorSetLayout::Builder	45
ven::Model::Builder	49
ven::Camera	
Class for camera	51
ven::Clock	
Class for clock	62
ven::Colors	
Class for colors	66
ven::DescriptorPool	
Class for descriptor pool	78
ven::DescriptorSetLayout	
Class for descriptor set layout	83
ven::DescriptorWriter	
Class for descriptor writer	87
ven::Device	
Class for device	91
ven::Engine	
Class for engine	109
ven::EventManager	
Class for event manager	117
ven::FrameInfo	120
ven::Gui::funcs	123
ven::GlobalUbo	125
ven::Gui	
Class for Gui	127
std::hash< ven::Model::Vertex >	135
ven::KeyAction	136
ven::KeyMappings	138
ven::Light	
Class for light	141

ven::LightPushConstantData	145
ven::Model	
Class for model	147
ven::Object	
Class for object	152
ven::ObjectBufferData	158
ven::ObjectPushConstantData	159
ven::ObjectRenderSystem	
Class for object render system	160
ven::PipelineConfigInfo	164
ven::PointLightData	168
ven::PointLightRenderSystem	
Class for point light system	169
ven::QueueFamilyIndices	173
ven::Renderer	
Class for renderer	175
ven::SceneManager	
Class for object manager	185
ven::Shaders	
Class for shaders	190
ven::SwapChain	
Class for swap chain	196
ven::SwapChainSupportDetails	206
ven::Texture	
Class for texture	208
ven::Transform3D	
Class for 3D transformation	217
ven::Model::Vertex	219
ven::Window	
Class for window	222

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

/home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag	229
/home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag	229
/home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert	230
/home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert	231
/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp	
This file contains the Buffer class	232
/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp	
This file contains the Camera class	235
/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp	
This file contains the Device class	244
/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp	
This file contains the Engine class	246
/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp	
This file contains the EventManager class	249
/home/runner/work/VEngine/VEngine/include/VEngine/FramelInfo.hpp	
This file contains the FramelInfo class	251
/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp	
This file contains the ImGuiWindowManager class	252
/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp	
This file contains the Light class	255
/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp	
This file contains the Model class	257
/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp	
This file contains the Object class	259
/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp	
This file contains the Renderer class	261
/home/runner/work/VEngine/VEngine/include/VEngine/SceneManager.hpp	
This file contains the SceneManager class	269
/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp	
This file contains the Shader class	271
/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp	
This file contains the Shader class	274
/home/runner/work/VEngine/VEngine/include/VEngine/Texture.hpp	
This file contains the Texture class	277
/home/runner/work/VEngine/VEngine/include/VEngine/Transform3D.hpp	
This file contains the Transform3D class	279

/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp	
This file contains the Window class	286
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp	
This file contains the DescriptorPool class	238
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp	
This file contains the DescriptorSetLayout class	240
/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp	
This file contains the DescriptorsWriter class	242
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp	
This file contains the ARenderSystemBase class	264
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp	
This file contains the ObjectRenderSystem class	265
/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp	
This file contains the PointLightRenderSystem class	267
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp	
This file contains the Clock class	280
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp	282
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp	285
/home/runner/work/VEngine/VEngine/src/buffer.cpp	289
/home/runner/work/VEngine/VEngine/src/camera.cpp	290
/home/runner/work/VEngine/VEngine/src/device.cpp	296
/home/runner/work/VEngine/VEngine/src/engine.cpp	306
/home/runner/work/VEngine/VEngine/src/eventManager.cpp	309
/home/runner/work/VEngine/VEngine/src/main.cpp	317
/home/runner/work/VEngine/VEngine/src/model.cpp	318
/home/runner/work/VEngine/VEngine/src/renderer.cpp	322
/home/runner/work/VEngine/VEngine/src/sceneManager.cpp	324
/home/runner/work/VEngine/VEngine/src/shaders.cpp	325
/home/runner/work/VEngine/VEngine/src/swapChain.cpp	328
/home/runner/work/VEngine/VEngine/src/texture.cpp	337
/home/runner/work/VEngine/VEngine/src/window.cpp	342
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp	293
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp	294
/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp	295
/home/runner/work/VEngine/VEngine/src/gui/init.cpp	310
/home/runner/work/VEngine/VEngine/src/gui/render.cpp	312
/home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp	334
/home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp	335
/home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp	336
/home/runner/work/VEngine/VEngine/src/utils/clock.cpp	342

Chapter 6

Namespace Documentation

6.1 ven Namespace Reference

Classes

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

- class [Model](#)
Class for model.
- class [Object](#)
Class for object.
- struct [ObjectBufferData](#)
- struct [ObjectPushConstantData](#)
- class [ObjectRenderSystem](#)
Class for object render system.
- struct [PipelineConfigInfo](#)
- struct [PointLightData](#)
- class [PointLightRenderSystem](#)
Class for point light system.
- struct [QueueFamilyIndices](#)
- class [Renderer](#)
Class for renderer.
- class [SceneManager](#)
Class for object manager.
- class [Shaders](#)
Class for shaders.
- class [SwapChain](#)
Class for swap chain.
- struct [SwapChainSupportDetails](#)
- class [Texture](#)
Class for texture.
- class [Transform3D](#)
Class for 3D transformation.
- class [Window](#)
Class for window.

Typedefs

- using [TimePoint](#) = std::chrono::time_point<std::chrono::high_resolution_clock>

Enumerations

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

Functions

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

Variables

- static constexpr glm::vec3 [DEFAULT_POSITION](#) {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 [DEFAULT_ROTATION](#) {0.F, 0.F, 0.F}
- static constexpr float [DEFAULT_FOV](#) = glm::radians(50.0F)
- static constexpr float [DEFAULT_NEAR](#) = 0.1F
- static constexpr float [DEFAULT_FAR](#) = 100.F
- static constexpr float [DEFAULT_MOVE_SPEED](#) = 3.F
- static constexpr float [DEFAULT_LOOK_SPEED](#) = 1.5F
- static constexpr uint32_t [DEFAULT_MAX_SETS](#) = 1000
- static constexpr float [EPSILON](#) = std::numeric_limits<float>::epsilon()
- static constexpr [KeyMappings](#) [DEFAULT_KEY_MAPPINGS](#) {}
- 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 uint16_t [DESCRIPTOR_COUNT](#) = 1000
- static constexpr float [DEFAULT_LIGHT_INTENSITY](#) = .2F
- static constexpr float [DEFAULT_LIGHT_RADIUS](#) = 0.1F
- static constexpr float [DEFAULT_SHININESS](#) = 32.F
- static constexpr glm::vec4 [DEFAULT_LIGHT_COLOR](#) = {glm::vec3(1.F), [DEFAULT_LIGHT_INTENSITY](#)}
- static constexpr uint8_t [MAX_LIGHTS](#) = 10
- static constexpr uint16_t [MAX_OBJECTS](#) = 1000
- static constexpr VkClearColorValue [DEFAULT_CLEAR_COLOR](#) = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue [DEFAULT_CLEAR_DEPTH](#) = {1.0F, 0}
- static constexpr std::string_view [SHADERS_BIN_PATH](#) = "build/shaders/"
- static constexpr int [MAX_FRAMES_IN_FLIGHT](#) = 2
- static constexpr float [COLOR_MAX](#) = 255.0F
- static constexpr uint32_t [DEFAULT_WIDTH](#) = 1920
- static constexpr uint32_t [DEFAULT_HEIGHT](#) = 1080
- static constexpr std::string_view [DEFAULT_TITLE](#) = "VEngine"

6.1.1 Typedef Documentation

6.1.1.1 TimePoint

using [ven::TimePoint](#) = std::chrono::time_point<std::chrono::high_resolution_clock>

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

6.1.2 Enumeration Type Documentation

6.1.2.1 ENGINE_STATE

enum [ven::ENGINE_STATE](#) : uint8_t

Enumerator

EDITOR	
GAME	
PAUSED	
EXIT	

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

6.1.2.2 GUI_STATE

enum [ven::GUI_STATE](#) : uint8_t

Enumerator

VISIBLE	
HIDDEN	

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

6.1.3 Function Documentation

6.1.3.1 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 [HashCombine.hpp](#).

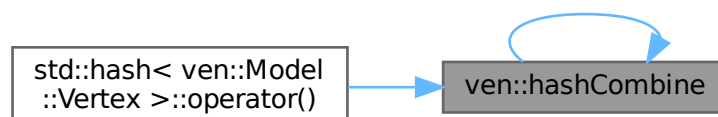
References [hashCombine\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.4 Variable Documentation

6.1.4.1 COLOR_MAX

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

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

6.1.4.2 DEFAULT_AMBIENT_LIGHT_COLOR

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

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

6.1.4.3 DEFAULT_AMBIENT_LIGHT_INTENSITY

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

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

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

6.1.4.4 DEFAULT_CLEAR_COLOR

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

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

6.1.4.5 DEFAULT_CLEAR_DEPTH

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

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

6.1.4.6 DEFAULT_FAR

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

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

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

6.1.4.7 DEFAULT_FOV

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

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

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

6.1.4.8 DEFAULT_HEIGHT

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

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

6.1.4.9 DEFAULT_KEY_MAPPINGS

```
KeyMappings ven::DEFAULT_KEY_MAPPINGS {} [static], [constexpr]
```

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

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

6.1.4.10 DEFAULT_LIGHT_COLOR

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

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

6.1.4.11 DEFAULT_LIGHT_INTENSITY

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

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

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

6.1.4.12 DEFAULT_LIGHT_RADIUS

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

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

6.1.4.13 DEFAULT_LOOK_SPEED

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

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

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

6.1.4.14 DEFAULT_MAX_SETS

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

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

6.1.4.15 DEFAULT_MOVE_SPEED

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

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

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

6.1.4.16 DEFAULT_NEAR

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

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

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

6.1.4.17 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::Gui::cameraSection\(\)](#).

6.1.4.18 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::Gui::cameraSection\(\)](#).

6.1.4.19 DEFAULT_SHININESS

```
float ven::DEFAULT_SHININESS = 32.F [static], [constexpr]
```

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

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

6.1.4.20 DEFAULT_TITLE

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

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

6.1.4.21 DEFAULT_WIDTH

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

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

6.1.4.22 DESCRIPTOR_COUNT

```
uint16_t ven::DESCRIPTOR_COUNT = 1000 [static], [constexpr]
```

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

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

6.1.4.23 EPSILON

```
float ven::EPSILON = std::numeric_limits<float>::epsilon() [static], [constexpr]
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

6.1.4.24 MAX_FRAMES_IN_FLIGHT

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

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

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

6.1.4.25 MAX_LIGHTS

```
uint8_t ven::MAX_LIGHTS = 10 [static], [constexpr]
```

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

Referenced by [ven::SceneManager::createLight\(\)](#).

6.1.4.26 MAX_OBJECTS

```
uint16_t ven::MAX_OBJECTS = 1000 [static], [constexpr]
```

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

Referenced by [ven::SceneManager::createObject\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

6.1.4.27 SHADERS_BIN_PATH

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

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

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

Chapter 7

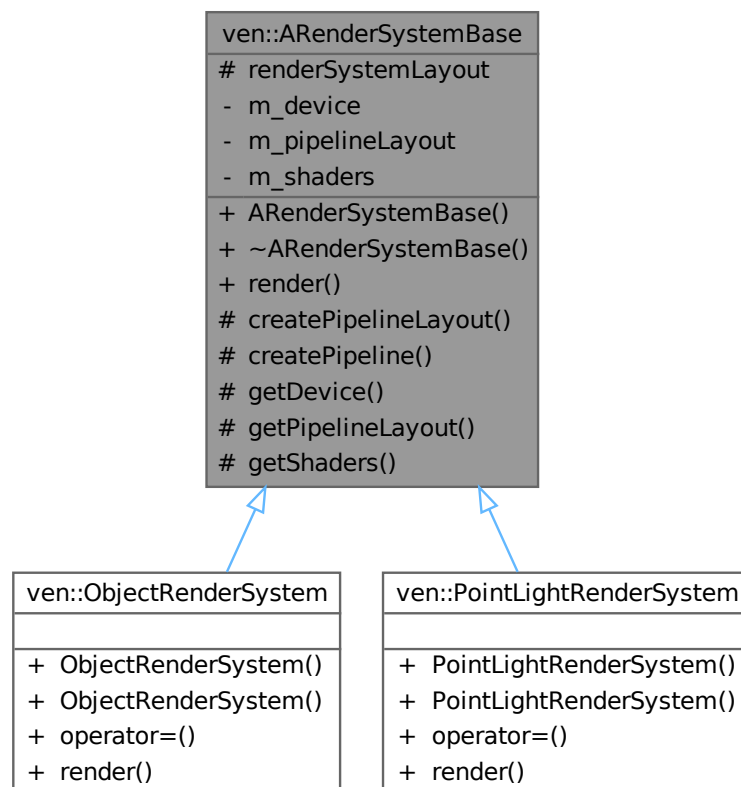
Class Documentation

7.1 ven::ARenderSystemBase Class Reference

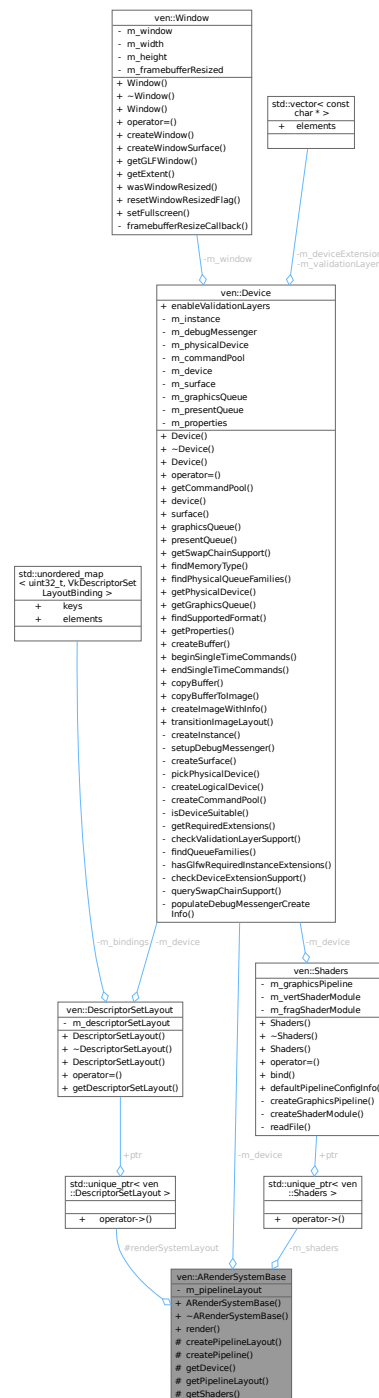
Abstract class for render system base.

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

Inheritance diagram for ven::ARenderSystemBase:



Collaboration diagram for `ven::ARenderSystemBase`:



Public Member Functions

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

Protected Member Functions

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

Protected Attributes

- std::unique_ptr< [DescriptorSetLayout](#) > [renderSystemLayout](#)

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 23 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 27 of file [ARenderSystemBase.hpp](#).

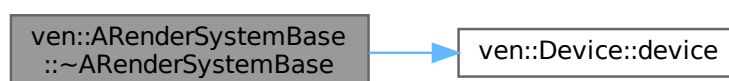
7.1.2.2 ~ARenderSystemBase()

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

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 38 of file [renderSystemBase.cpp](#).

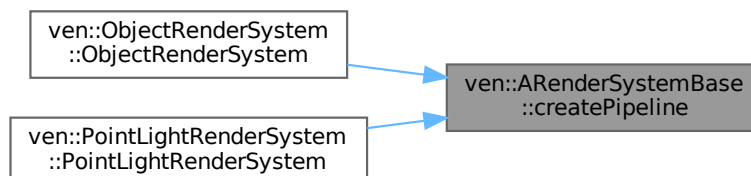
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.2 createPipelineLayout()

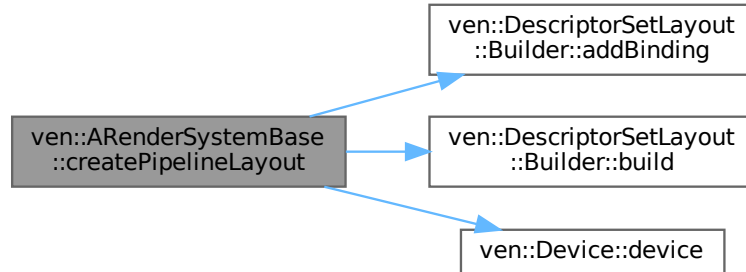
```
void ven::ARenderSystemBase::createPipelineLayout (
    VkDescriptorSetLayout globalSetLayout,
    uint32_t pushConstantSize) [protected]
```

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

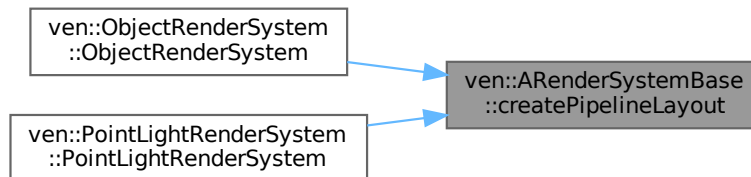
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::Device::device\(\)](#), [m_device](#), [m_pipelineLayout](#), and [renderSystemLayout](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.3.3 getDevice()

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

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

References [m_device](#).

7.1.3.4 getPipelineLayout()

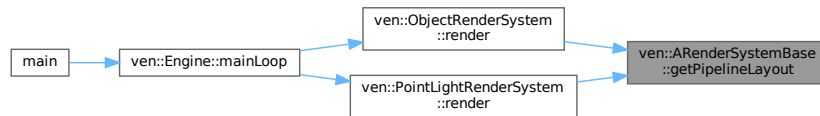
```
VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]
```

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

References [m_pipelineLayout](#).

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

Here is the caller graph for this function:



7.1.3.5 getShaders()

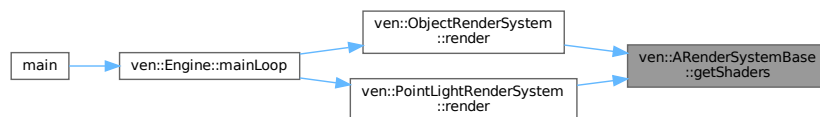
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],
[nodiscard], [protected]
```

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

References [m_shaders](#).

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

Here is the caller graph for this function:



7.1.3.6 render()

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

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

7.1.4 Member Data Documentation

7.1.4.1 m_device

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

Definition at line 45 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 46 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 47 of file [ARenderSystemBase.hpp](#).

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

7.1.4.4 renderSystemLayout

```
std::unique_ptr<DescriptorSetLayout> ven::ARenderSystemBase::renderSystemLayout [protected]
```

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

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

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

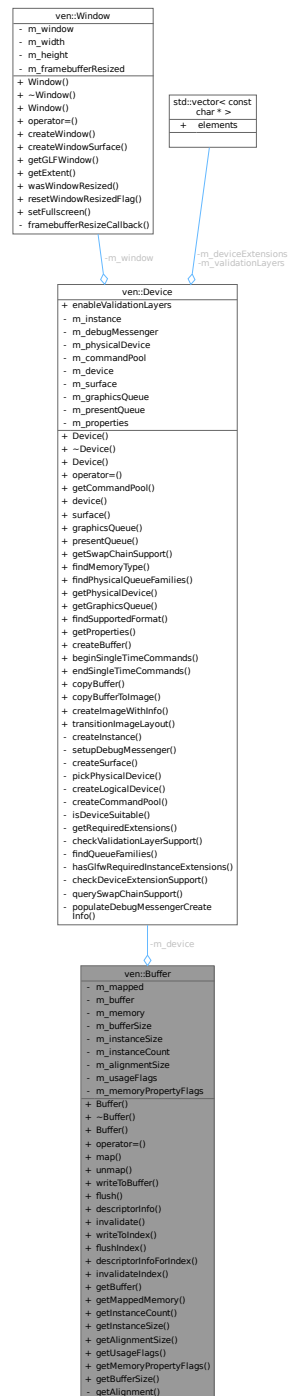
- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp](#)

7.2 ven::Buffer Class Reference

Class for buffer.

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

Collaboration diagram for ven::Buffer:



Public Member Functions

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

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

Static Private Member Functions

- `static VkDeviceSize` `getAlignment` (`const VkDeviceSize` instanceSize, `const VkDeviceSize` minOffset↵
Alignment)
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 6 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 12 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

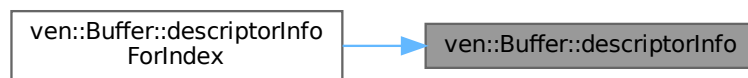
VkDescriptorBufferInfo of specified offset and range

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
```

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 46 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.

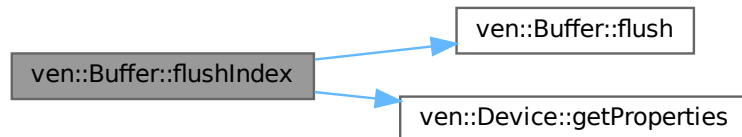
Parameters

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

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

References [flush\(\)](#), [ven::Device::getProperties\(\)](#), [m_alignmentSize](#), and [m_device](#).

Here is the call graph for this function:



7.2.3.5 getAlignment()

```
static VkDeviceSize ven::Buffer::getAlignment (
    const VkDeviceSize instanceSize,
    const VkDeviceSize minOffsetAlignment) [inline], [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 145 of file [Buffer.hpp](#).

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 56 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: $\text{index} * \text{alignmentSize}$
--------------	---

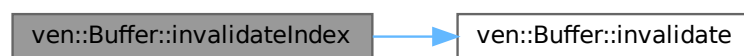
Returns

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

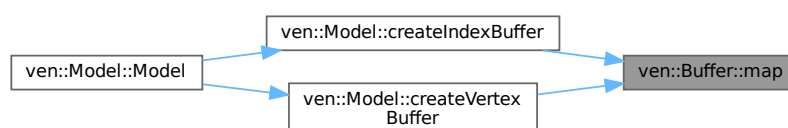
Returns

VkResult of the buffer mapping call

Definition at line 19 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 25 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 33 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](#).

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

7.2.4.5 m_instanceCount

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

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

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

7.2.4.6 m_instanceSize

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

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

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

7.2.4.7 m_mapped

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

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

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

7.2.4.8 m_memory

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

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

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

7.2.4.9 m_memoryPropertyFlags

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

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

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

7.2.4.10 m_usageFlags

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

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

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

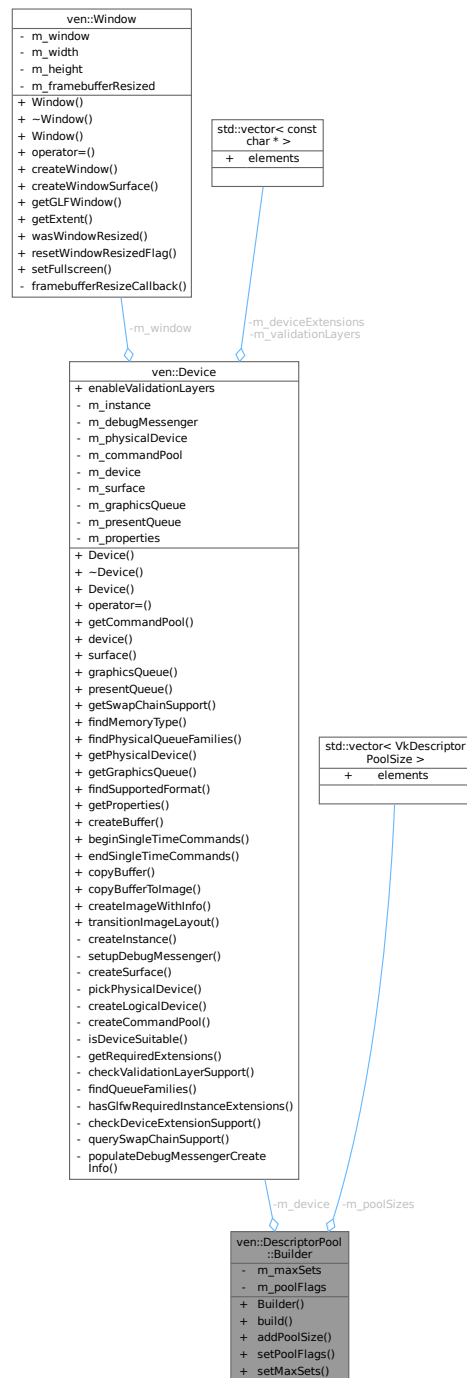
The documentation for this class was generated from the following files:

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

7.3 ven::DescriptorPool::Builder Class Reference

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

Collaboration diagram for ven::DescriptorPool::Builder:



Public Member Functions

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

Private Attributes

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

7.3.1 Detailed Description

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 Builder()

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

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

7.3.3 Member Function Documentation

7.3.3.1 addPoolSize()

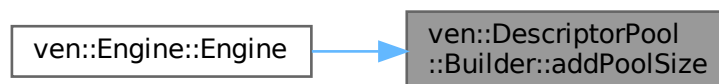
```
Builder & ven::DescriptorPool::Builder::addPoolSize (
    const VkDescriptorType descriptorType,
    const uint32_t count) [inline]
```

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

References [m_poolSizes](#).

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

Here is the caller graph for this function:



7.3.3.2 build()

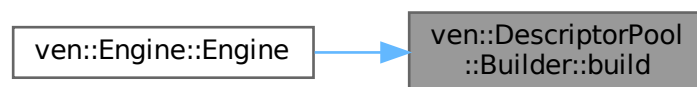
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],  
[nodiscard]
```

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

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

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

Here is the caller graph for this function:



7.3.3.3 setMaxSets()

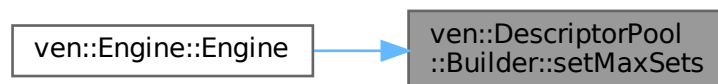
```
Builder & ven::DescriptorPool::Builder::setMaxSets (  
    const uint32_t count) [inline]
```

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

References [m_maxSets](#).

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

Here is the caller graph for this function:



7.3.3.4 setPoolFlags()

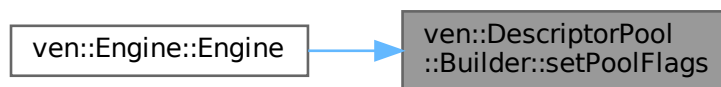
```
Builder & ven::DescriptorPool::Builder::setPoolFlags (
    const VkDescriptorPoolCreateFlags flags) [inline]
```

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

References [m_poolFlags](#).

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

Here is the caller graph for this function:



7.3.4 Member Data Documentation

7.3.4.1 m_device

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

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

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

7.3.4.2 m_maxSets

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

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

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

7.3.4.3 m_poolFlags

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

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

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

7.3.4.4 m_poolSizes

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

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

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

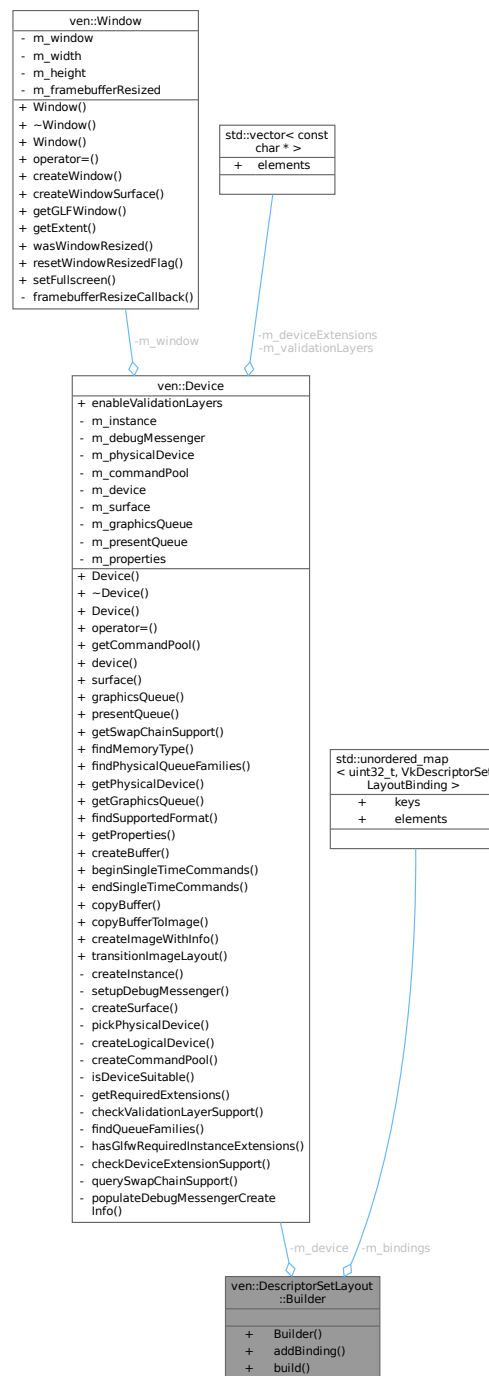
The documentation for this class was generated from the following file:

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

7.4 ven::DescriptorSetLayout::Builder Class Reference

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

Collaboration diagram for `ven::DescriptorSetLayout::Builder`:



Public Member Functions

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

Private Attributes

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

7.4.1 Detailed Description

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

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Builder()

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

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

7.4.3 Member Function Documentation

7.4.3.1 addBinding()

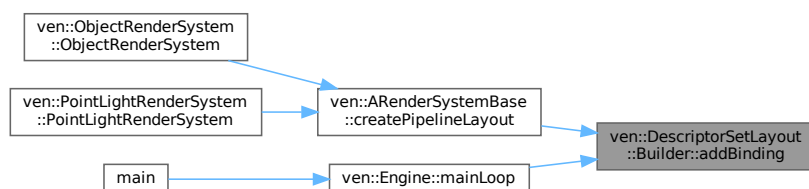
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

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

References [m_bindings](#).

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.4.3.2 build()

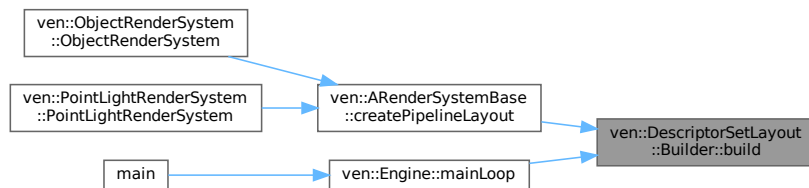
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

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

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

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::Engine::mainLoop\(\)](#).

Here is the caller graph for this function:



7.4.4 Member Data Documentation

7.4.4.1 m_bindings

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

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

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

7.4.4.2 m_device

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

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

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

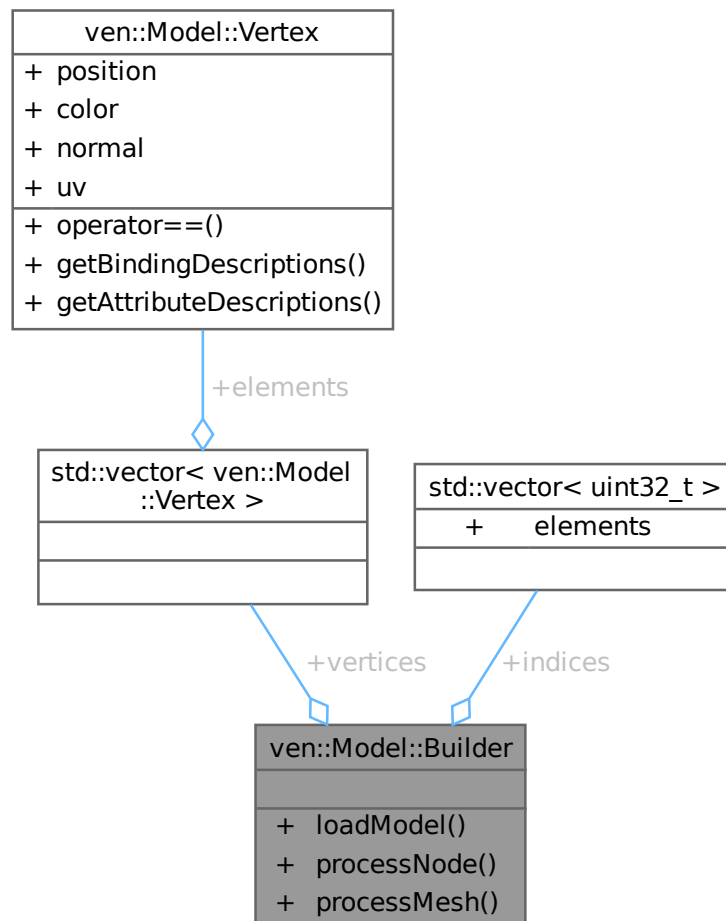
The documentation for this class was generated from the following files:

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

7.5 ven::Model::Builder Struct Reference

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

Collaboration diagram for ven::Model::Builder:



Public Member Functions

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

Public Attributes

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

7.5.1 Detailed Description

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

7.5.2 Member Function Documentation

7.5.2.1 loadModel()

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

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

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

Here is the caller graph for this function:



7.5.2.2 processMesh()

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

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

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

7.5.2.3 processNode()

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

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

7.5.3 Member Data Documentation

7.5.3.1 indices

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

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

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

7.5.3.2 vertices

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

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

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

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

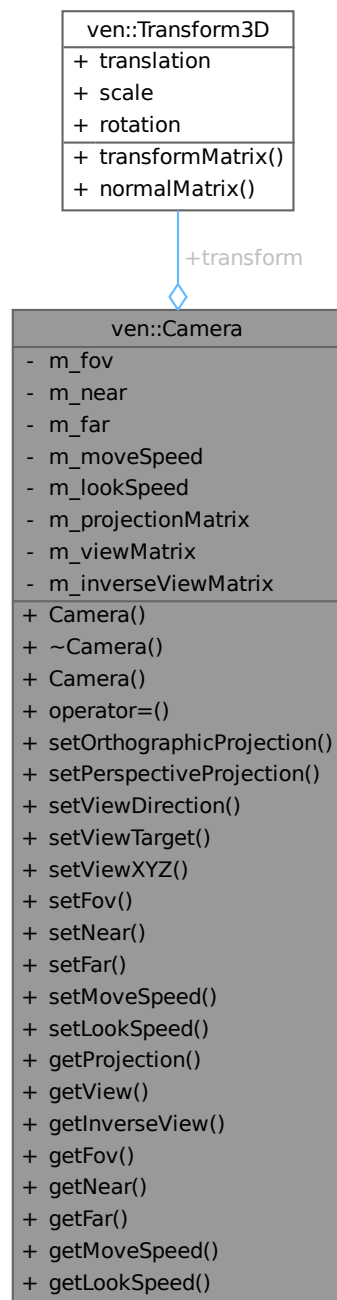
- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.6 ven::Camera Class Reference

Class for camera.

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

Collaboration diagram for ven::Camera:



Public Member Functions

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

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

Public Attributes

- [Transform3D transform](#) {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}

Private Attributes

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

7.6.1 Detailed Description

Class for camera.

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

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Camera() [1/2]

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

7.6.2.2 ~Camera()

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

7.6.2.3 Camera() [2/2]

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

7.6.3 Member Function Documentation

7.6.3.1 getFar()

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

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

References [m_far](#).

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

Here is the caller graph for this function:



7.6.3.2 getFov()

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

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

References [m_fov](#).

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

Here is the caller graph for this function:



7.6.3.3 getInverseView()

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

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

References [m_inverseViewMatrix](#).

7.6.3.4 getLookSpeed()

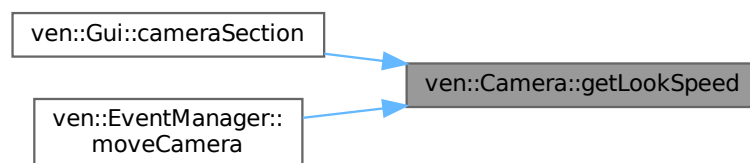
```
float ven::Camera::getLookSpeed () const [inline], [nodiscard]
```

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

References [m_lookSpeed](#).

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

Here is the caller graph for this function:



7.6.3.5 getMoveSpeed()

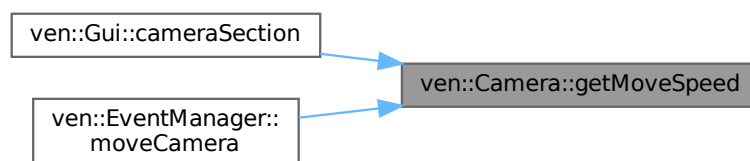
```
float ven::Camera::getMoveSpeed () const [inline], [nodiscard]
```

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

References [m_moveSpeed](#).

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

Here is the caller graph for this function:



7.6.3.6 getNear()

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

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

References [m_near](#).

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

Here is the caller graph for this function:



7.6.3.7 getProjection()

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

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

References [m_projectionMatrix](#).

7.6.3.8 getView()

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

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

References [m_viewMatrix](#).

7.6.3.9 operator=()

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

7.6.3.10 setFar()

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

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

References [m_far](#).

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

Here is the caller graph for this function:



7.6.3.11 setFov()

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

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

References [m_fov](#).

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

Here is the caller graph for this function:



7.6.3.12 setLookSpeed()

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

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

References [m_lookSpeed](#).

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

Here is the caller graph for this function:



7.6.3.13 setMoveSpeed()

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

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

References [m_moveSpeed](#).

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

Here is the caller graph for this function:



7.6.3.14 setNear()

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

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

References [m_near](#).

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

Here is the caller graph for this function:



7.6.3.15 setOrthographicProjection()

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

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

References [m_projectionMatrix](#).

7.6.3.16 setPerspectiveProjection()

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

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

7.6.3.17 setViewDirection()

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

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

7.6.3.18 setViewTarget()

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

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

7.6.3.19 setViewXYZ()

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

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

7.6.4 Member Data Documentation

7.6.4.1 m_far

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

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

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

7.6.4.2 m_fov

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

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

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

7.6.4.3 m_inverseViewMatrix

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

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

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

7.6.4.4 m_lookSpeed

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

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

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

7.6.4.5 m_moveSpeed

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

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

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

7.6.4.6 m_near

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

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

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

7.6.4.7 m_projectionMatrix

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

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

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

7.6.4.8 m_viewMatrix

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

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

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

7.6.4.9 transform

```
Transform3D ven::Camera::transform {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}
```

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

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

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

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

7.7 ven::Clock Class Reference

Class for clock.

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

Collaboration diagram for ven::Clock:

ven::Clock
<ul style="list-style-type: none"> - m_startTime - m_stopTime - m_deltaTime - m_isStopped
<ul style="list-style-type: none"> + Clock() + ~Clock() + Clock() + operator=() + start() + stop() + resume() + update() + getDeltaTime()

Public Member Functions

- [Clock](#) ()
- [~Clock](#) ()=default
- [Clock](#) (const [Clock](#) &)=delete
- [Clock](#) & [operator=](#) (const [Clock](#) &)=delete
- void [start](#) ()
- void [stop](#) ()
- void [resume](#) ()
- void [update](#) ()
- float [getDeltaTime](#) () const

Private Attributes

- [TimePoint](#) m_startTime
- [TimePoint](#) m_stopTime
- std::chrono::duration< float > [m_deltaTime](#) {0.F}
- bool [m_isStopped](#) {false}

7.7.1 Detailed Description

Class for clock.

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Clock() [1/2]

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

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

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

Here is the call graph for this function:



7.7.2.2 ~Clock()

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

7.7.2.3 Clock() [2/2]

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

7.7.3 Member Function Documentation

7.7.3.1 getDeltaTime()

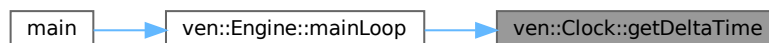
```
float ven::Clock::getDeltaTime () const [inline], [nodiscard]
```

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

References [m_deltaTime](#).

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

Here is the caller graph for this function:



7.7.3.2 operator=()

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

7.7.3.3 resume()

```
void ven::Clock::resume ()
```

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

7.7.3.4 start()

```
void ven::Clock::start () [inline]
```

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

References [m_startTime](#).

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

Here is the caller graph for this function:



7.7.3.5 stop()

```
void ven::Clock::stop ()
```

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

7.7.3.6 update()

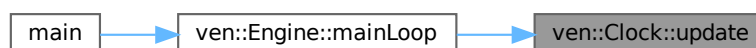
```
void ven::Clock::update ()
```

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

References [m_deltaTime](#), and [m_startTime](#).

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

Here is the caller graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 m_deltaTime

```
std::chrono::duration<float> ven::Clock::m_deltaTime {0.F} [private]
```

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

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

7.7.4.2 m_isStopped

```
bool ven::Clock::m_isStopped {false} [private]
```

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

7.7.4.3 m_startTime

```
TimePoint ven::Clock::m_startTime [private]
```

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

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

7.7.4.4 m_stopTime

```
TimePoint ven::Clock::m_stopTime [private]
```

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

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

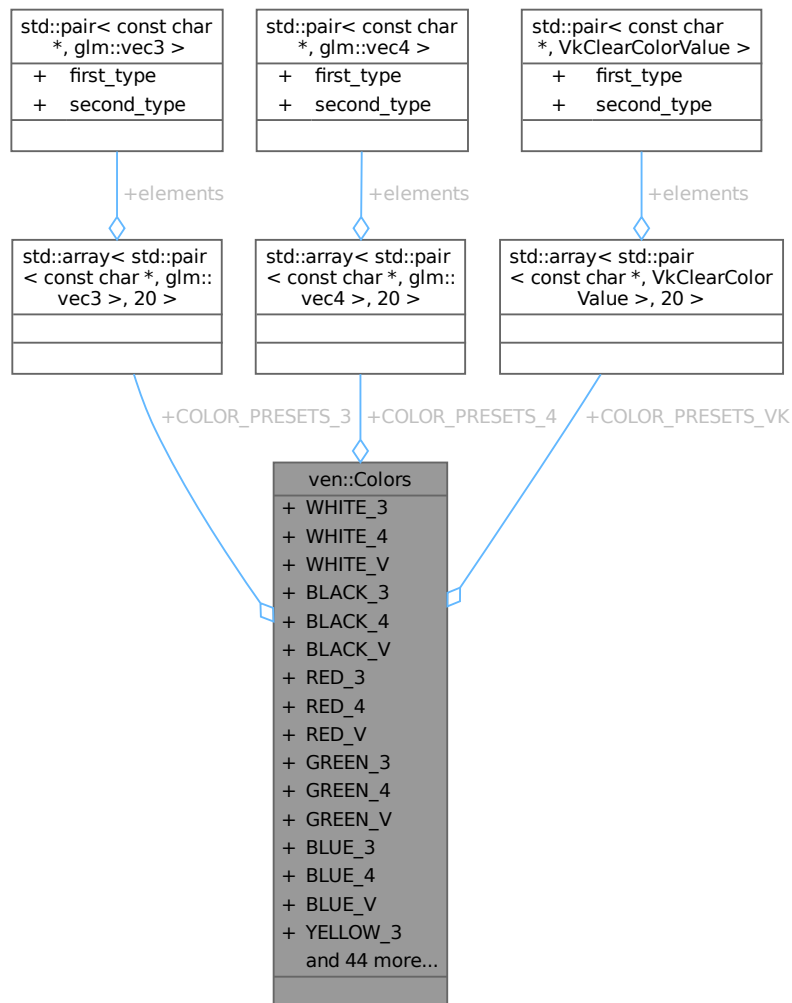
- [/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/Utils/clock.cpp](#)

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_3](#) = glm::vec3([COLOR_MAX](#)) / [COLOR_MAX](#)
- static constexpr glm::vec4 [WHITE_4](#) = { 1.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue [WHITE_V](#) = { { 1.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 [BLACK_3](#) = glm::vec3(0.0F)
- static constexpr glm::vec4 [BLACK_4](#) = { 0.0F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue [BLACK_V](#) = { { 0.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 [RED_3](#) = glm::vec3([COLOR_MAX](#), 0.0F, 0.0F) / [COLOR_MAX](#)
- static constexpr glm::vec4 [RED_4](#) = { 1.0F, 0.0F, 0.0F, 1.0F }

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

7.8.1 Detailed Description

Class for colors.

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

7.8.2 Member Data Documentation

7.8.2.1 AQUA_3

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

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

7.8.2.2 AQUA_4

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

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

7.8.2.3 AQUA_V

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

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

7.8.2.4 BLACK_3

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

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

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

7.8.2.5 BLACK_4

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

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

7.8.2.6 BLACK_V

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

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

7.8.2.7 BLUE_3

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

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

7.8.2.8 BLUE_4

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

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

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

7.8.2.9 BLUE_V

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

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

7.8.2.10 COLOR_PRESETS_3

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

Initial value:

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

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

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

7.8.2.11 COLOR_PRESETS_4

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

Initial value:

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

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

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

7.8.2.12 COLOR_PRESETS_VK

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

Initial value:

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

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

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

7.8.2.13 CYAN_3

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

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

7.8.2.14 CYAN_4

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

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

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

7.8.2.15 CYAN_V

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

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

7.8.2.16 FUCHSIA_3

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

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

7.8.2.17 FUCHSIA_4

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

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

7.8.2.18 FUCHSIA_V

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

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

7.8.2.19 GRAY_3

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

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

7.8.2.20 GRAY_4

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

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

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

7.8.2.21 GRAY_V

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

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

7.8.2.22 GREEN_3

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

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

7.8.2.23 GREEN_4

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

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

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

7.8.2.24 GREEN_V

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

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

7.8.2.25 LIME_3

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

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

7.8.2.26 LIME_4

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

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

7.8.2.27 LIME_V

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

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

7.8.2.28 MAGENTA_3

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

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

7.8.2.29 MAGENTA_4

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

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

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

7.8.2.30 MAGENTA_V

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

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

7.8.2.31 MAROON_3

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

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

7.8.2.32 MAROON_4

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

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

7.8.2.33 MAROON_V

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

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

7.8.2.34 NAVY_3

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

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

7.8.2.35 NAVY_4

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

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

7.8.2.36 NAVY_V

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

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

7.8.2.37 NIGHT_BLUE_3

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

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

7.8.2.38 NIGHT_BLUE_4

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

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

7.8.2.39 NIGHT_BLUE_V

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

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

7.8.2.40 OLIVE_3

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

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

7.8.2.41 OLIVE_4

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

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

7.8.2.42 OLIVE_V

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

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

7.8.2.43 RED_3

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

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

7.8.2.44 RED_4

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

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

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

7.8.2.45 RED_V

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

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

7.8.2.46 SILVER_3

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

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

7.8.2.47 SILVER_4

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

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

7.8.2.48 SILVER_V

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

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

7.8.2.49 SKY_BLUE_3

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

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

7.8.2.50 SKY_BLUE_4

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

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

7.8.2.51 SKY_BLUE_V

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

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

7.8.2.52 SUNSET_3

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

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

7.8.2.53 SUNSET_4

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

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

7.8.2.54 SUNSET_V

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

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

7.8.2.55 TEAL_3

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

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

7.8.2.56 TEAL_4

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

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

7.8.2.57 TEAL_V

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

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

7.8.2.58 WHITE_3

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

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

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

7.8.2.59 WHITE_4

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

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

7.8.2.60 WHITE_V

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

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

7.8.2.61 YELLOW_3

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

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

7.8.2.62 YELLOW_4

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

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

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

7.8.2.63 YELLOW_V

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

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

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

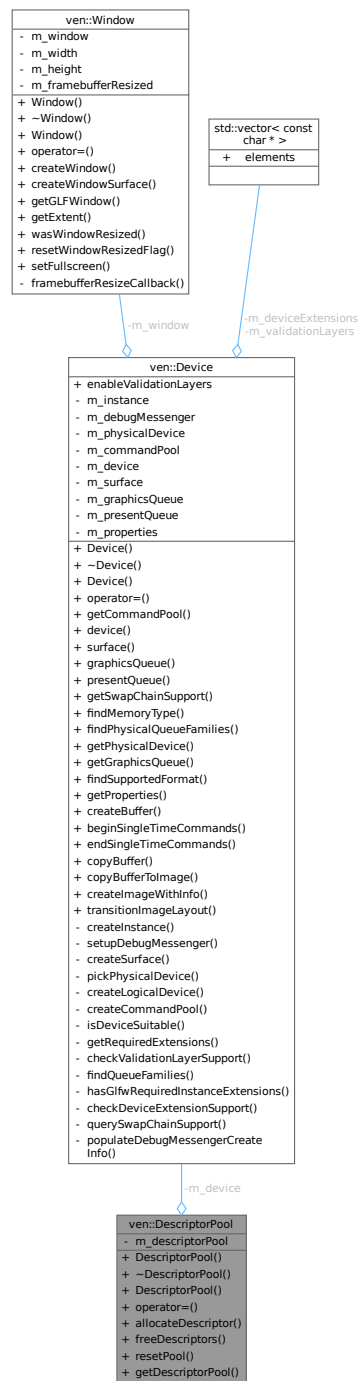
- [/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp](#)

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 22 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 3 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 48 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 18 of file [descriptorPool.cpp](#).

7.9.3.2 freeDescriptors()

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

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

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

Here is the call graph for this function:



7.9.3.3 getDescriptorPool()

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

Definition at line 57 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 55 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 63 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 62 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 61 of file [DescriptorPool.hpp](#).

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

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

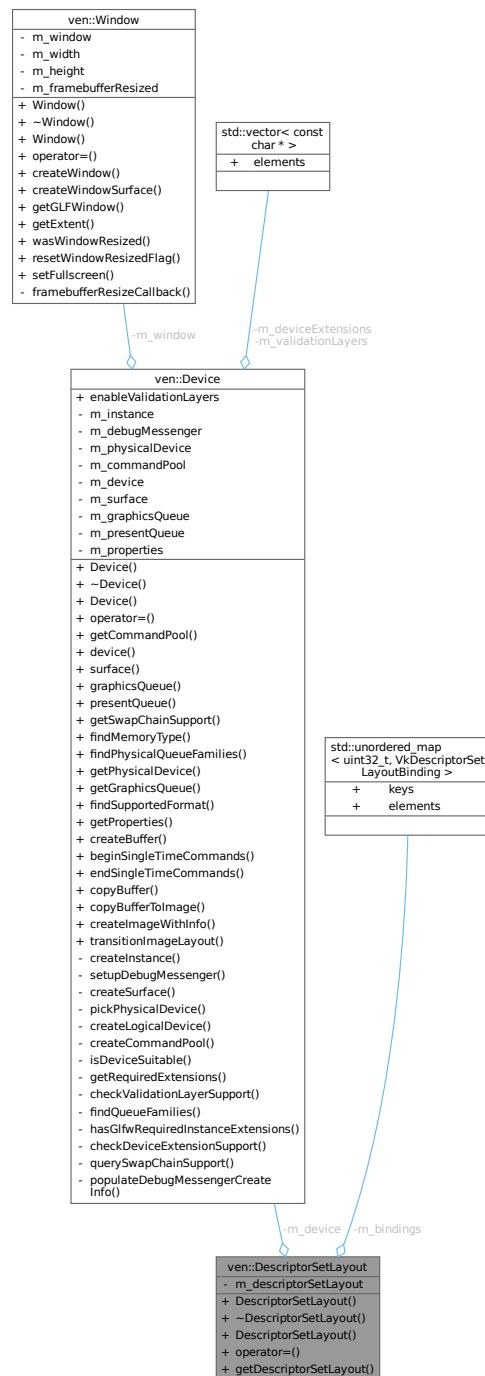
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp](#)

7.10 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

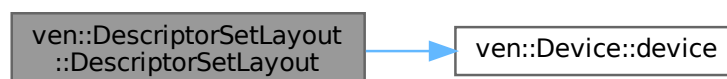
7.10.2.1 DescriptorSetLayout() [1/2]

```
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:



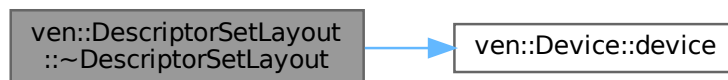
7.10.2.2 ~DescriptorSetLayout()

```
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 47 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 55 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 53 of file [DescriptorSetLayout.hpp](#).

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

7.10.5.2 m_descriptorSetLayout

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

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

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

7.10.5.3 m_device

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

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

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

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

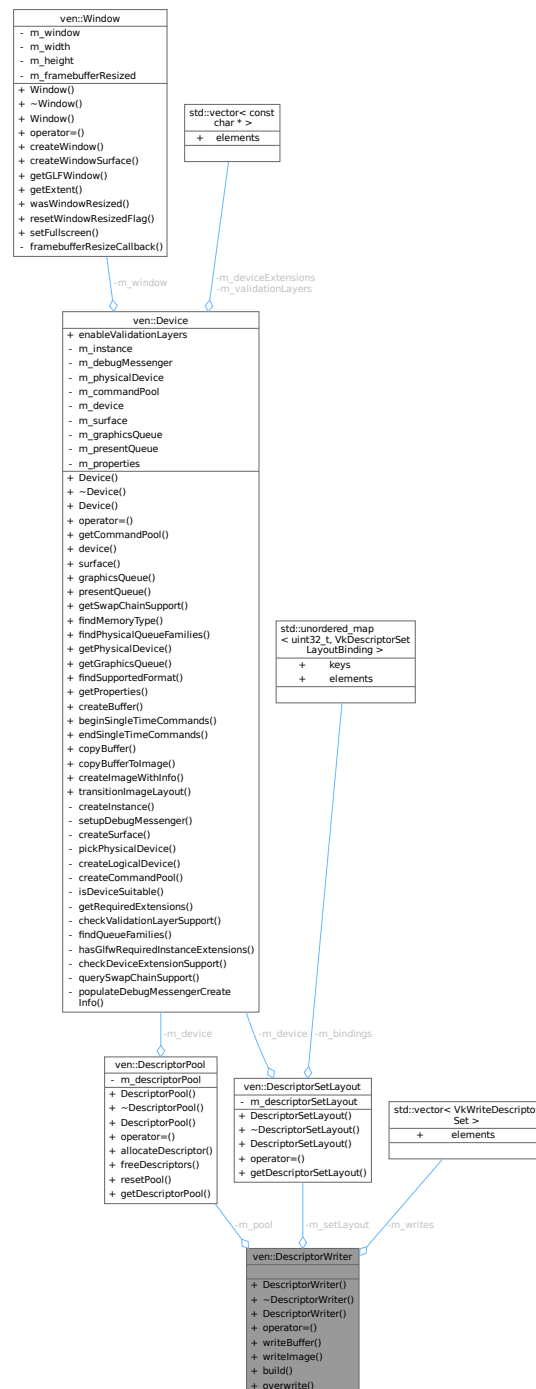
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp](#)

7.11 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` ()=default
- `DescriptorWriter` (const `DescriptorWriter` &)=delete
- `DescriptorWriter` & operator= (const `DescriptorWriter` &)=delete
- `DescriptorWriter` & writeBuffer (uint32_t binding, const `VkDescriptorBufferInfo` *bufferInfo)

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

Private Attributes

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

7.11.1 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() [1/2]

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

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

7.11.2.2 ~DescriptorWriter()

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

7.11.2.3 DescriptorWriter() [2/2]

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

7.11.3 Member Function Documentation

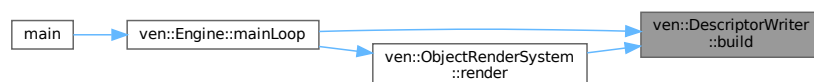
7.11.3.1 build()

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

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

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

Here is the caller graph for this function:



7.11.3.2 operator=()

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

7.11.3.3 overwrite()

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

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

7.11.3.4 writeBuffer()

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

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

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

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

Here is the caller graph for this function:



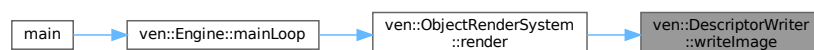
7.11.3.5 writelImage()

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

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

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

Here is the caller graph for this function:



7.11.4 Member Data Documentation

7.11.4.1 m_pool

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

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

7.11.4.2 m_setLayout

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

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

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

7.11.4.3 m_writes

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

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

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

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

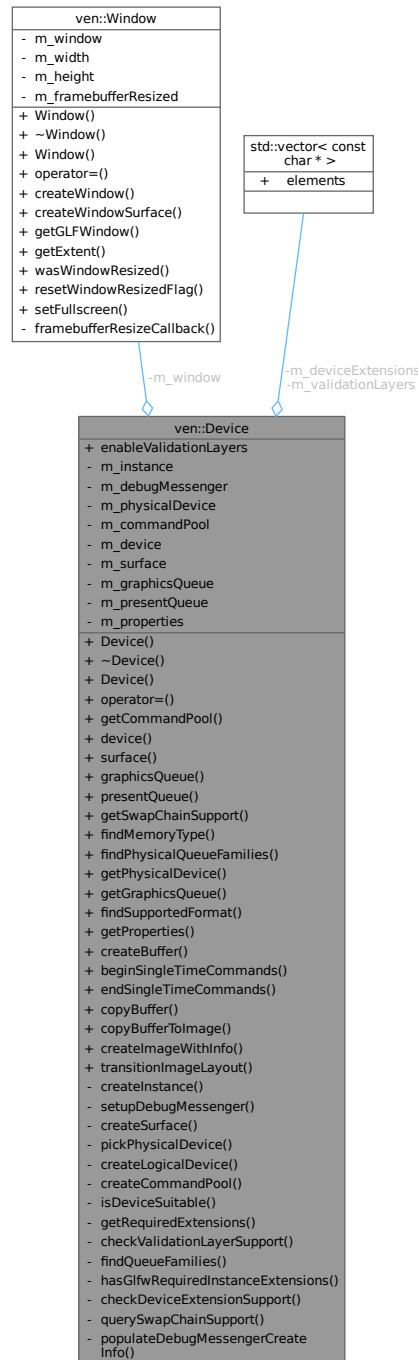
- [/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp](#)

7.12 ven::Device Class Reference

Class for device.

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

Collaboration diagram for ven::Device:



Public Member Functions

- `Device (Window &>window)`
- `~Device ()`
- `Device (const Device &)=delete`
- `Device & operator= (const Device &)=delete`
- `VkCommandPool getCommandPool () const`

- `VkDevice` [device](#) () const
- `VkSurfaceKHR` [surface](#) () const
- `VkQueue` [graphicsQueue](#) () const
- `VkQueue` [presentQueue](#) () const
- [SwapChainSupportDetails](#) [getSwapChainSupport](#) () const
- `uint32_t` [findMemoryType](#) (`uint32_t` typeFilter, `VkMemoryPropertyFlags` properties) const
- [QueueFamilyIndices](#) [findPhysicalQueueFamilies](#) () const
- `VkPhysicalDevice` [getPhysicalDevice](#) () const
- `VkQueue` [getGraphicsQueue](#) () const
- `VkFormat` [findSupportedFormat](#) (const `std::vector< VkFormat >` &candidates, `VkImageTiling` tiling, `VkFormatFeatureFlags` features) const
- `VkPhysicalDeviceProperties` [getProperties](#) () 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
- void [transitionImageLayout](#) (`VkImage` image, `VkFormat` format, `VkImageLayout` oldLayout, `VkImageLayout` newLayout, `uint32_t` mipLevels=1, `uint32_t` layerCount=1) const

Public Attributes

- const bool [enableValidationLayers](#) = true

Private Member Functions

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

Static Private Member Functions

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

Private Attributes

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

7.12.1 Detailed Description

Class for device.

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

7.12.2 Constructor & Destructor Documentation

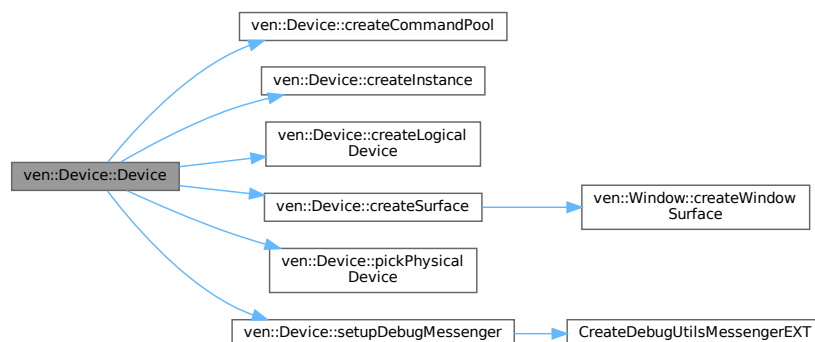
7.12.2.1 Device() [1/2]

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

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

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

Here is the call graph for this function:



7.12.2.2 ~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() [2/2]

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

7.12.3 Member Function Documentation

7.12.3.1 beginSingleTimeCommands()

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

Definition at line 413 of file [device.cpp](#).

7.12.3.2 checkDeviceExtensionSupport()

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

Definition at line 290 of file [device.cpp](#).

7.12.3.3 checkValidationLayerSupport()

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

Definition at line 227 of file [device.cpp](#).

7.12.3.4 copyBuffer()

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

Definition at line 447 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 460 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 384 of file [device.cpp](#).

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

Here is the caller graph for this function:



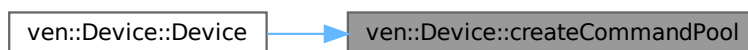
7.12.3.7 createCommandPool()

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

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

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 481 of file [device.cpp](#).

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

Here is the caller graph for this function:



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 124 of file [device.cpp](#).

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

Here is the caller graph for this function:



7.12.3.11 createSurface()

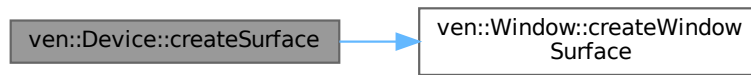
```
void ven::Device::createSurface () [inline], [private]
```

Definition at line 78 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()

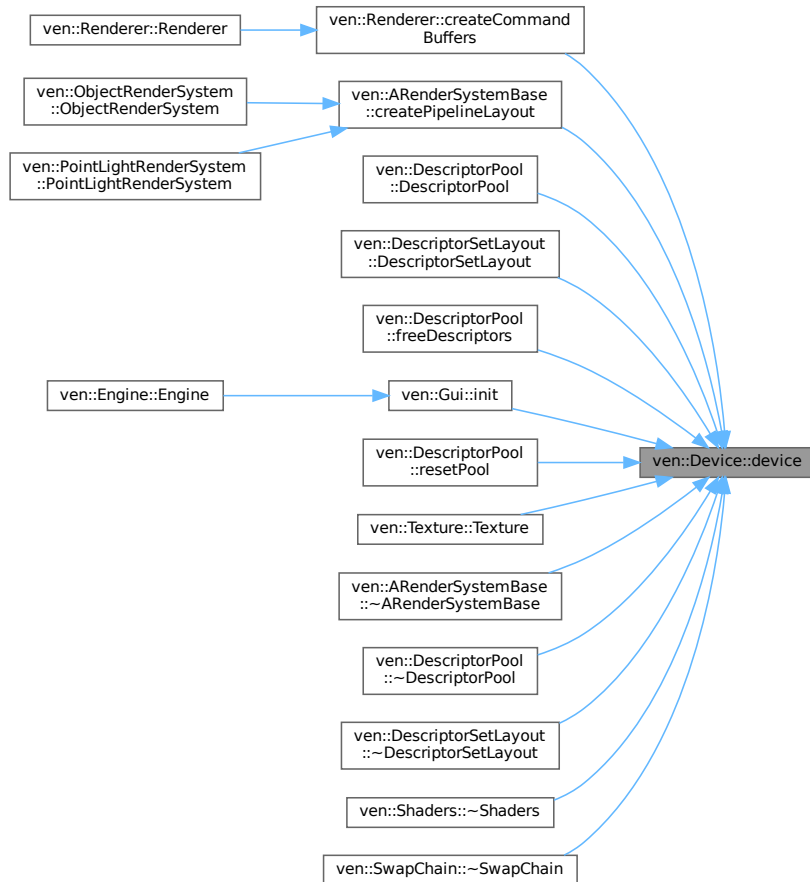
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

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

References [m_device](#).

Referenced by [ven::Renderer::createCommandBuffers\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), [ven::DescriptorPool::DescriptorPool\(\)](#), [ven::DescriptorSetLayout::DescriptorSetLayout\(\)](#), [ven::DescriptorPool::freeDescriptors\(\)](#), [ven::Gui::init\(\)](#), [ven::DescriptorPool::resetPool\(\)](#), [ven::Texture::Texture\(\)](#), [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 432 of file [device.cpp](#).

7.12.3.14 findMemoryType()

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

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

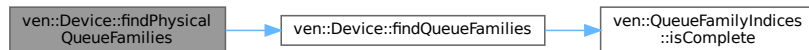
7.12.3.15 findPhysicalQueueFamilies()

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

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

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

Here is the call graph for this function:



7.12.3.16 findQueueFamilies()

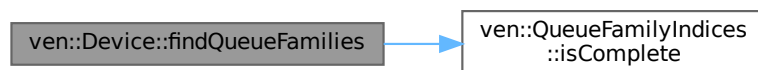
`ven::QueueFamilyIndices` `ven::Device::findQueueFamilies (VkPhysicalDevice device) const [private]`

Definition at line 306 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 355 of file [device.cpp](#).

7.12.3.18 getCommandPool()

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

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

References [m_commandPool](#).

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

Here is the caller graph for this function:



7.12.3.19 getGraphicsQueue()

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

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

References [m_graphicsQueue](#).

7.12.3.20 getPhysicalDevice()

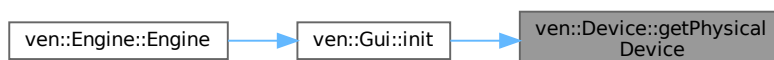
```
VkPhysicalDevice ven::Device::getPhysicalDevice () const [inline], [nodiscard]
```

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

References [m_physicalDevice](#).

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

Here is the caller graph for this function:



7.12.3.21 `getProperties()`

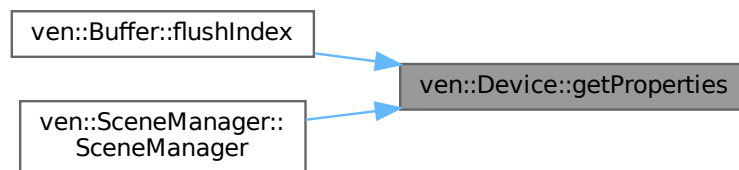
```
VkPhysicalDeviceProperties ven::Device::getProperties () const [inline], [nodiscard]
```

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

References [m_properties](#).

Referenced by [ven::Buffer::flushIndex\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

Here is the caller graph for this function:



7.12.3.22 `getRequiredExtensions()`

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

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

7.12.3.23 `getSwapChainSupport()`

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

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

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

Here is the call graph for this function:



7.12.3.24 graphicsQueue()

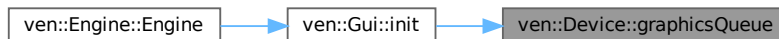
```
VkQueue ven::Device::graphicsQueue () const [inline], [nodiscard]
```

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

References [m_graphicsQueue](#).

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

Here is the caller graph for this function:



7.12.3.25 hasGlfwRequiredInstanceExtensions()

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

Definition at line 267 of file [device.cpp](#).

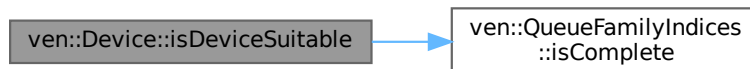
7.12.3.26 isDeviceSuitable()

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

Definition at line 187 of file [device.cpp](#).

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

Here is the call graph for this function:



7.12.3.27 operator=()

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

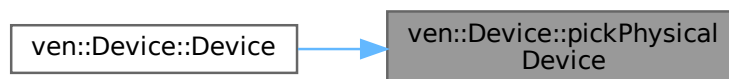
7.12.3.28 pickPhysicalDevice()

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

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

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

Here is the caller graph for this function:



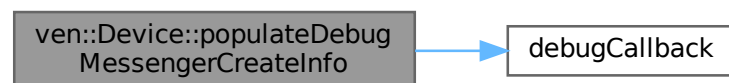
7.12.3.29 populateDebugMessengerCreateInfo()

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

Definition at line 204 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 54 of file [Device.hpp](#).

References [m_presentQueue](#).

7.12.3.31 querySwapChainSupport()

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

Definition at line 334 of file [device.cpp](#).

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

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

Here is the caller graph for this function:



7.12.3.32 setupDebugMessenger()

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

Definition at line 217 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 52 of file [Device.hpp](#).

References [m_surface](#).

7.12.3.34 transitionImageLayout()

```
void ven::Device::transitionImageLayout (  
    VkImage image,  
    VkFormat format,  
    VkImageLayout oldLayout,  
    VkImageLayout newLayout,  
    uint32_t mipLevels = 1,  
    uint32_t layerCount = 1) const
```

Definition at line 504 of file [device.cpp](#).

7.12.4 Member Data Documentation

7.12.4.1 enableValidationLayers

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

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

7.12.4.2 m_commandPool

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

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

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

7.12.4.3 m_debugMessenger

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

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

7.12.4.4 m_device

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

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

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

7.12.4.5 m_deviceExtensions

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

Definition at line 106 of file [Device.hpp](#).

7.12.4.6 m_graphicsQueue

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

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

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

7.12.4.7 m_instance

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

Definition at line 93 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 95 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 102 of file [Device.hpp](#).

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

7.12.4.10 m_properties

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

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

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

7.12.4.11 m_surface

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

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

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

7.12.4.12 m_validationLayers

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

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

7.12.4.13 m_window

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

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

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

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

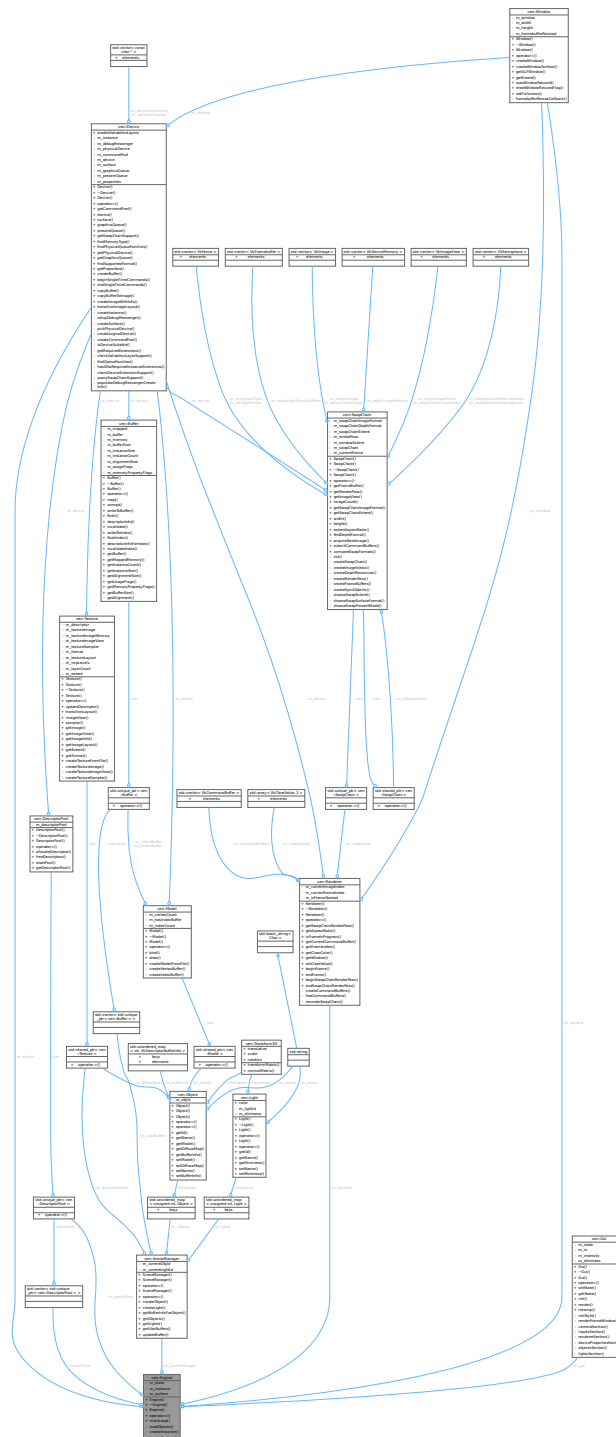
- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/device.cpp](#)

7.13 ven::Engine Class Reference

Class for engine.

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

Collaboration diagram for `ven::Engine`:



Public Member Functions

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

Private Member Functions

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

Private Attributes

- [ENGINE_STATE](#) [m_state](#) {EXIT}
- [Window](#) [m_window](#)
- [Device](#) [m_device](#) {[m_window](#)}
- [Renderer](#) [m_renderer](#) {[m_window](#), [m_device](#)}
- [Gui](#) [m_gui](#)
- [std::unique_ptr< DescriptorPool >](#) [m_globalPool](#)
- [std::vector< std::unique_ptr< DescriptorPool > >](#) [framePools](#)
- [SceneManager](#) [m_sceneManager](#) {[m_device](#)}
- [VkInstance](#) [m_instance](#) {nullptr}
- [VkSurfaceKHR](#) [m_surface](#) {nullptr}

7.13.1 Detailed Description

Class for engine.

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

7.13.2 Constructor & Destructor Documentation

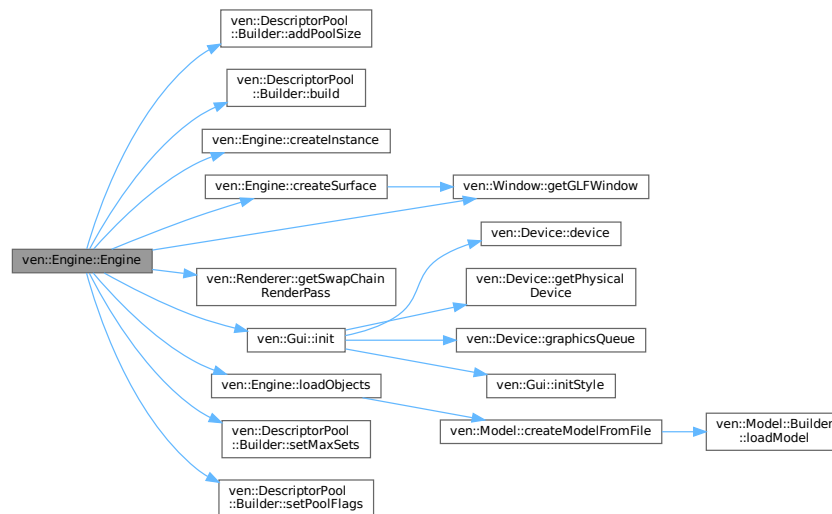
7.13.2.1 Engine() [1/2]

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

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

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

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 35 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 62 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()

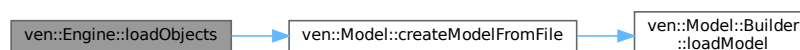
```
void ven::Engine::loadObjects () [private]
```

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

References [ven::Colors::BLUE_4](#), [ven::Model::createModelFromFile\(\)](#), [ven::Colors::CYAN_4](#), [ven::Colors::GREEN_4](#), [ven::Colors::MAGENTA_4](#), [ven::Colors::RED_4](#), and [ven::Colors::YELLOW_4](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.4 mainLoop()

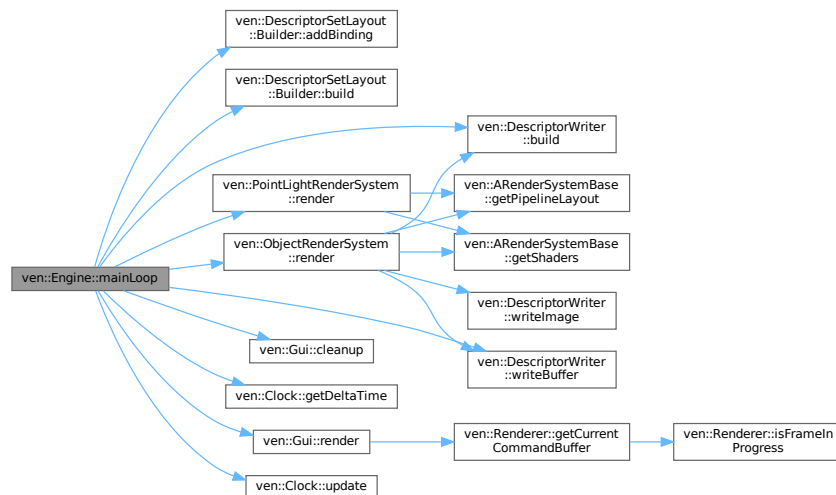
```
void ven::Engine::mainLoop ()
```

Definition at line 103 of file [engine.cpp](#).

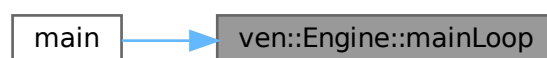
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::DescriptorWriter::build\(\)](#), [ven::Gui::cleanup\(\)](#), [ven::EXIT](#), [ven::FrameInfo::frameIndex](#), [ven::Clock::getDeltaTime\(\)](#), [ven::MAX_FRAMES_IN_FLIGHT](#), [ven::Gui::render\(\)](#), [ven::ObjectRenderSystem::render\(\)](#), [ven::PointLightRenderSystem::render\(\)](#), [ven::Clock::update\(\)](#), [ven::VISIBLE](#), and [ven::DescriptorWriter::writeBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.5 operator=()

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

7.13.4 Member Data Documentation

7.13.4.1 framePools

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

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

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

7.13.4.2 m_device

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

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

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

7.13.4.3 m_globalPool

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

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

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

7.13.4.4 m_gui

```
Gui ven::Engine::m_gui [private]
```

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

7.13.4.5 m_instance

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

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

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

7.13.4.6 m_renderer

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

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

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

7.13.4.7 m_sceneManager

```
SceneManager ven::Engine::m_sceneManager {m_device} [private]
```

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

7.13.4.8 m_state

```
ENGINE_STATE ven::Engine::m_state {EXIT} [private]
```

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

7.13.4.9 m_surface

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

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

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

7.13.4.10 m_window

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

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

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

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

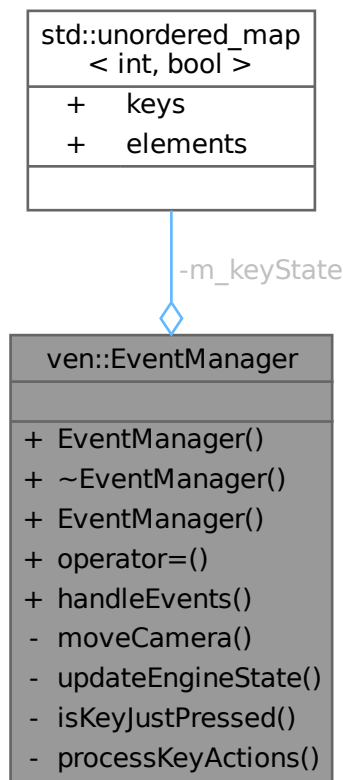
- [/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/engine.cpp](#)

7.14 ven::EventManager Class Reference

Class for event manager.

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::EventManager:



Public Member Functions

- [EventManager](#) ()=default
- [~EventManager](#) ()=default
- [EventManager](#) (const [EventManager](#) &)=delete
- [EventManager](#) & [operator=](#) (const [EventManager](#) &)=delete
- void [handleEvents](#) (GLFWwindow *window, [ENGINE_STATE](#) *engineState, [Camera](#) &camera, [Gui](#) &gui, float dt) const

Static Private Member Functions

- static void [moveCamera](#) (GLFWwindow *window, [Camera](#) &camera, float dt)
- static void [updateEngineState](#) ([ENGINE_STATE](#) *engineState, const [ENGINE_STATE](#) newState)
- static bool [isKeyJustPressed](#) (GLFWwindow *window, int key, std::unordered_map< int, bool > &keyStates)
- template<typename Iterator >
static void [processKeyActions](#) (GLFWwindow *window, Iterator begin, Iterator end)

Private Attributes

- `std::unordered_map< int, bool > m_keyState`

7.14.1 Detailed Description

Class for event manager.

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

7.14.2 Constructor & Destructor Documentation

7.14.2.1 EventManager() [1/2]

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

7.14.2.2 ~EventManager()

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

7.14.2.3 EventManager() [2/2]

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

7.14.3 Member Function Documentation

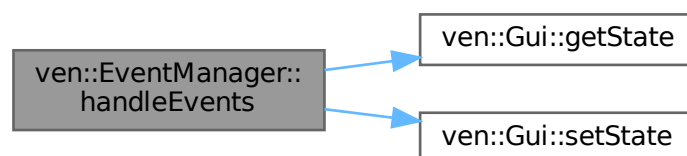
7.14.3.1 handleEvents()

```
void ven::EventManager::handleEvents (
    GLFWwindow * window,
    ENGINE_STATE * engineState,
    Camera & camera,
    Gui & gui,
    float dt) const
```

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

References [ven::DEFAULT_KEY_MAPPINGS](#), [ven::EXIT](#), [ven::Gui::getState\(\)](#), [ven::HIDDEN](#), [ven::Gui::setState\(\)](#), [ven::KeyMappings::toggleGui](#), and [ven::VISIBLE](#).

Here is the call graph for this function:



7.14.3.2 isKeyJustPressed()

```
bool ven::EventManager::isKeyJustPressed (
    GLFWwindow * window,
    int key,
    std::unordered_map< int, bool > & keyStates) [static], [private]
```

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

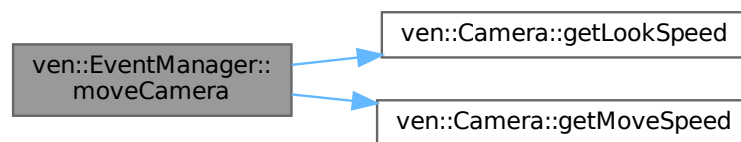
7.14.3.3 moveCamera()

```
void ven::EventManager::moveCamera (
    GLFWwindow * window,
    Camera & camera,
    float dt) [static], [private]
```

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

References [ven::DEFAULT_KEY_MAPPINGS](#), [ven::EPSILON](#), [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::KeyMappings::lookDown](#), [ven::KeyMappings::lookLeft](#), [ven::KeyMappings::lookRight](#), [ven::KeyMappings::lookUp](#), [ven::KeyMappings::moveBackward](#), [ven::KeyMappings::moveDown](#), [ven::KeyMappings::moveForward](#), [ven::KeyMappings::moveLeft](#), [ven::KeyMappings::moveRight](#), [ven::KeyMappings::moveUp](#), [ven::Transform3D::rotation](#), [ven::Camera::transform](#), and [ven::Transform3D::translation](#).

Here is the call graph for this function:



7.14.3.4 operator=()

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

7.14.3.5 processKeyActions()

```
template<typename Iterator >
void ven::EventManager::processKeyActions (
    GLFWwindow * window,
    Iterator begin,
    Iterator end) [static], [private]
```

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

7.14.3.6 updateEngineState()

```
static void ven::EventManager::updateEngineState (  
    ENGINE_STATE * engineState,  
    const ENGINE_STATE newState) [inline], [static], [private]
```

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

7.14.4 Member Data Documentation

7.14.4.1 m_keyState

```
std::unordered_map<int, bool> ven::EventManager::m_keyState [mutable], [private]
```

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

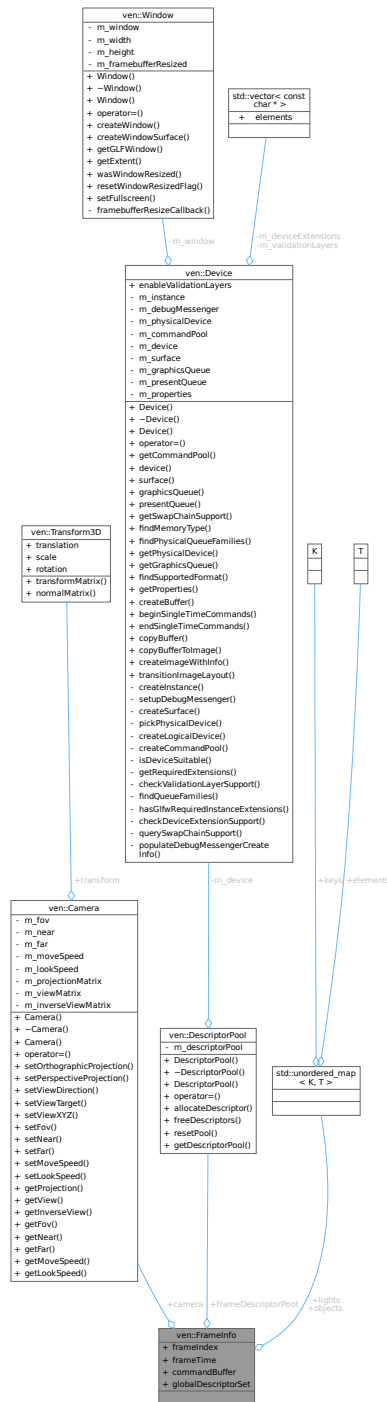
The documentation for this class was generated from the following files:

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

7.15 ven::FrameInfo Struct Reference

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

Collaboration diagram for ven::FrameInfo:



Public Attributes

- unsigned long [frameIndex](#)
- float [frameTime](#)
- `VkCommandBuffer` [commandBuffer](#)
- [Camera](#) & [camera](#)
- `VkDescriptorSet` [globalDescriptorSet](#)

- [DescriptorPool](#) & [frameDescriptorPool](#)
- [Object::Map](#) & [objects](#)
- [Light::Map](#) & [lights](#)

7.15.1 Detailed Description

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

7.15.2 Member Data Documentation

7.15.2.1 camera

[Camera](#)& ven::FrameInfo::camera

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

7.15.2.2 commandBuffer

VkCommandBuffer ven::FrameInfo::commandBuffer

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

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

7.15.2.3 frameDescriptorPool

[DescriptorPool](#)& ven::FrameInfo::frameDescriptorPool

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

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

7.15.2.4 frameIndex

unsigned long ven::FrameInfo::frameIndex

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

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

7.15.2.5 frameTime

float ven::FrameInfo::frameTime

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

7.15.2.6 globalDescriptorSet

VkDescriptorSet ven::FrameInfo::globalDescriptorSet

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

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

7.15.2.7 lights

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

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

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

7.15.2.8 objects

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

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

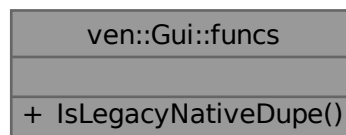
Referenced by [ven::ObjectRenderSystem::render\(\)](#).

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

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

7.16 ven::Gui::funcs Struct Reference

Collaboration diagram for ven::Gui::funcs:



Static Public Member Functions

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

7.16.1 Detailed Description

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

7.16.2 Member Function Documentation

7.16.2.1 IsLegacyNativeDupe()

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



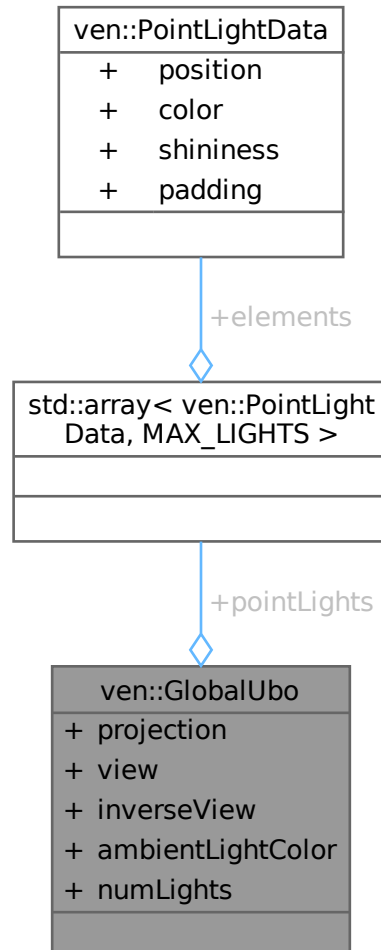
The documentation for this struct was generated from the following file:

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

7.17 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](#)
- uint8_t [numLights](#)

7.17.1 Detailed Description

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

7.17.2 Member Data Documentation

7.17.2.1 ambientLightColor

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

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

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

7.17.2.2 inverseView

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

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

7.17.2.3 numLights

```
uint8_t ven::GlobalUbo::numLights
```

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

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

7.17.2.4 pointLights

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

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

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

7.17.2.5 projection

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

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

7.17.2.6 view

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

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

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

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

7.18 ven::Gui Class Reference

Class for [Gui](#).

```
#include <Gui.hpp>
```

Collaboration diagram for ven::Gui:

ven::Gui
<ul style="list-style-type: none"> - m_state - m_io - m_intensity - m_shininess
<ul style="list-style-type: none"> + Gui() + ~Gui() + Gui() + operator=() + setState() + getState() + init() + render() + cleanup() - initStyle() - renderFrameWindow() - cameraSection() - inputsSection() - rendererSection() - devicePropertiesSection() - objectsSection() - lightsSection()

Classes

- struct [funcs](#)

Public Member Functions

- [Gui](#) ()=default
- [~Gui](#) ()=default
- [Gui](#) (const [Gui](#) &)=delete
- [Gui](#) & [operator=](#) (const [Gui](#) &)=delete
- void [setState](#) (const [GUI_STATE](#) state)
- [GUI_STATE](#) [getState](#) () const

Static Public Member Functions

- static void `init` (GLFWwindow *window, VkInstance instance, const `Device` *device, VkRenderPass render↵ Pass)
- static void `render` (`Renderer` *renderer, std::unordered_map< unsigned int, `Object` > &objects, std↵ ::unordered_map< unsigned int, `Light` > &lights, `Camera` &camera, VkPhysicalDevice physicalDevice, `GlobalUbo` &ubo)
- static void `cleanup` ()

Static Private Member Functions

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

Private Attributes

- `GUI_STATE m_state` {VISIBLE}

Static Private Attributes

- static ImGuiIO * `m_io` = nullptr
- static float `m_intensity` = 1.0F
- static float `m_shininess` = DEFAULT_SHININESS

7.18.1 Detailed Description

Class for `Gui`.

Definition at line 30 of file `Gui.hpp`.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 `Gui()` [1/2]

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

7.18.2.2 `~Gui()`

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

7.18.2.3 Gui() [2/2]

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

7.18.3 Member Function Documentation

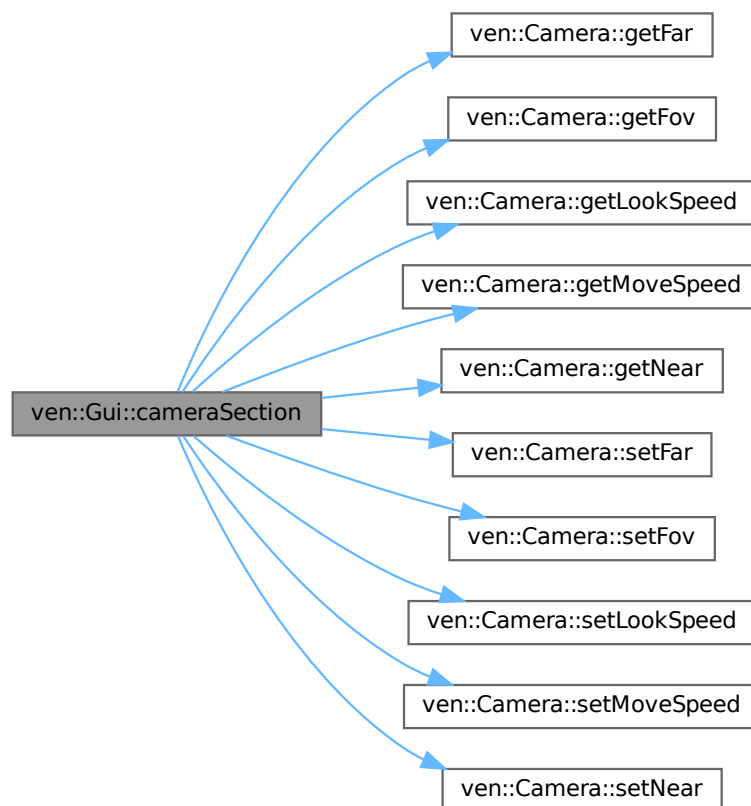
7.18.3.1 cameraSection()

```
void ven::Gui::cameraSection (  
    Camera & camera) [static], [private]
```

Definition at line 109 of file [render.cpp](#).

References [ven::DEFAULT_FAR](#), [ven::DEFAULT_FOV](#), [ven::DEFAULT_LOOK_SPEED](#), [ven::DEFAULT_MOVE_SPEED](#), [ven::DEFAULT_NEAR](#), [ven::DEFAULT_POSITION](#), [ven::DEFAULT_ROTATION](#), [ven::Camera::getFar\(\)](#), [ven::Camera::getFov\(\)](#), [ven::Camera::getLookSpeed\(\)](#), [ven::Camera::getMoveSpeed\(\)](#), [ven::Camera::getNear\(\)](#), [ven::Transform3D::rotation](#), [ven::Camera::setFar\(\)](#), [ven::Camera::setFov\(\)](#), [ven::Camera::setLookSpeed\(\)](#), [ven::Camera::setMoveSpeed\(\)](#), [ven::Camera::setNear\(\)](#), [ven::Camera::transform](#), and [ven::Transform3D::translation](#).

Here is the call graph for this function:



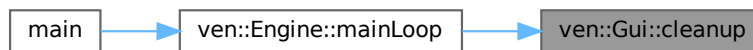
7.18.3.2 cleanup()

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

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

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

Here is the caller graph for this function:



7.18.3.3 devicePropertiesSection()

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

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

7.18.3.4 getState()

```
GUI\_STATE ven::Gui::getState () const [inline], [nodiscard]
```

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

References [m_state](#).

Referenced by [ven::EventManager::handleEvents\(\)](#).

Here is the caller graph for this function:



7.18.3.5 init()

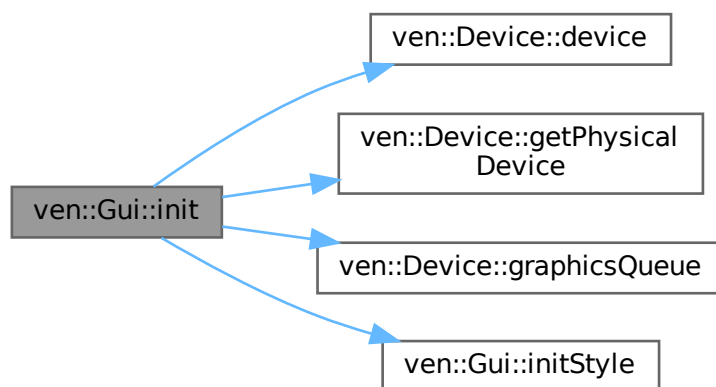
```
void ven::Gui::init (
    GLFWwindow * window,
    VkInstance instance,
    const Device * device,
    VkRenderPass renderPass) [static]
```

Definition at line 11 of file [init.cpp](#).

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



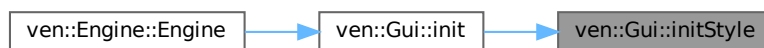
7.18.3.6 initStyle()

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

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

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

Here is the caller graph for this function:



7.18.3.7 inputsSection()

```
void ven::Gui::inputsSection (
    const ImGuiIO * io) [static], [private]
```

Definition at line 267 of file [render.cpp](#).

7.18.3.8 lightsSection()

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

Definition at line 177 of file [render.cpp](#).

References [ven::Colors::COLOR_PRESETS_3](#), [ven::DEFAULT_LIGHT_INTENSITY](#), and [ven::DEFAULT_SHININESS](#).

7.18.3.9 objectsSection()

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

Definition at line 158 of file [render.cpp](#).

References [ven::Colors::GRAY_4](#).

7.18.3.10 operator=()

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

7.18.3.11 render()

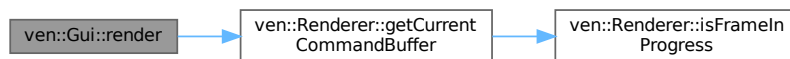
```
void ven::Gui::render (
    Renderer * renderer,
    std::unordered_map< unsigned int, Object > & objects,
    std::unordered_map< unsigned int, Light > & lights,
    Camera & camera,
    VkPhysicalDevice physicalDevice,
    GlobalUbo & ubo) [static]
```

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

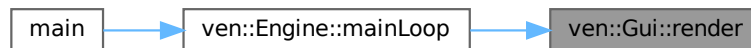
References [ven::Renderer::getCurrentCommandBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



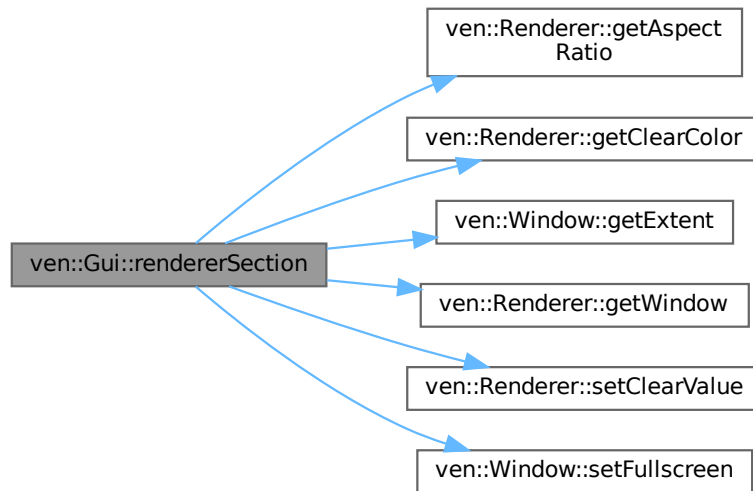
7.18.3.12 rendererSection()

```
void ven::Gui::rendererSection (
    Renderer * renderer,
    GlobalUbo & ubo) [static], [private]
```

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

References [ven::GlobalUbo::ambientLightColor](#), [ven::Colors::COLOR_PRESETS_4](#), [ven::Colors::COLOR_PRESETS_VK](#), [ven::DEFAULT_AMBIENT_LIGHT_INTENSITY](#), [ven::Renderer::getAspectRatio\(\)](#), [ven::Renderer::getClearColor\(\)](#), [ven::Window::getExtent\(\)](#), [ven::Renderer::getWindow\(\)](#), [ven::Renderer::setClearColor\(\)](#), and [ven::Window::setFullscreen\(\)](#).

Here is the call graph for this function:



7.18.3.13 `renderFrameWindow()`

```
void ven::Gui::renderFrameWindow () [static], [private]
```

Definition at line 38 of file [render.cpp](#).

7.18.3.14 `setState()`

```
void ven::Gui::setState (
    const GUI_STATE state) [inline]
```

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

References [m_state](#).

Referenced by [ven::EventManager::handleEvents\(\)](#).

Here is the caller graph for this function:



7.18.4 Member Data Documentation

7.18.4.1 m_intensity

```
float ven::Gui::m_intensity = 1.0F [static], [private]
```

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

7.18.4.2 m_io

```
ImGuiIO * ven::Gui::m_io = nullptr [static], [private]
```

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

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

7.18.4.3 m_shininess

```
float ven::Gui::m_shininess = DEFAULT_SHININESS [static], [private]
```

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

7.18.4.4 m_state

```
GUI_STATE ven::Gui::m_state {VISIBLE} [private]
```

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

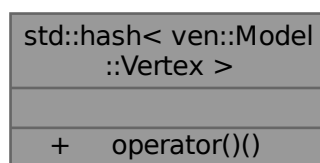
Referenced by [getState\(\)](#), and [setState\(\)](#).

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

- [/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/init.cpp](#)
- [/home/runner/work/VEngine/VEngine/src/gui/render.cpp](#)

7.19 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`

7.19.1 Detailed Description

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

7.19.2 Member Function Documentation

7.19.2.1 operator()()

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

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

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

Here is the call graph for this function:



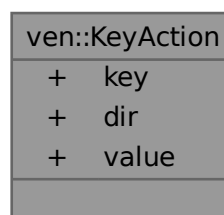
The documentation for this struct was generated from the following file:

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

7.20 ven::KeyAction Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for `ven::KeyAction`:



Public Attributes

- uint16_t [key](#)
- glm::vec3 * [dir](#)
- glm::vec3 [value](#)

7.20.1 Detailed Description

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

7.20.2 Member Data Documentation

7.20.2.1 dir

```
glm::vec3* ven::KeyAction::dir
```

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

7.20.2.2 key

```
uint16_t ven::KeyAction::key
```

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

7.20.2.3 value

```
glm::vec3 ven::KeyAction::value
```

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

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

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

7.21 ven::KeyMappings Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyMappings:

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

Public Attributes

- uint16_t [moveLeft](#) = GLFW_KEY_A
- uint16_t [moveRight](#) = GLFW_KEY_D
- uint16_t [moveForward](#) = GLFW_KEY_W
- uint16_t [moveBackward](#) = GLFW_KEY_S
- uint16_t [moveUp](#) = GLFW_KEY_SPACE
- uint16_t [moveDown](#) = GLFW_KEY_LEFT_SHIFT
- uint16_t [lookLeft](#) = GLFW_KEY_LEFT
- uint16_t [lookRight](#) = GLFW_KEY_RIGHT
- uint16_t [lookUp](#) = GLFW_KEY_UP
- uint16_t [lookDown](#) = GLFW_KEY_DOWN
- uint16_t [toggleGui](#) = GLFW_KEY_F1

7.21.1 Detailed Description

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

7.21.2 Member Data Documentation

7.21.2.1 lookDown

```
uint16_t ven::KeyMappings::lookDown = GLFW_KEY_DOWN
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.2 lookLeft

```
uint16_t ven::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.3 lookRight

```
uint16_t ven::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.4 lookUp

```
uint16_t ven::KeyMappings::lookUp = GLFW_KEY_UP
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.5 moveBackward

```
uint16_t ven::KeyMappings::moveBackward = GLFW_KEY_S
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.6 moveDown

```
uint16_t ven::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.7 moveForward

```
uint16_t ven::KeyMappings::moveForward = GLFW_KEY_W
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.8 moveLeft

```
uint16_t ven::KeyMappings::moveLeft = GLFW_KEY_A
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.9 moveRight

```
uint16_t ven::KeyMappings::moveRight = GLFW_KEY_D
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.10 moveUp

```
uint16_t ven::KeyMappings::moveUp = GLFW_KEY_SPACE
```

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

Referenced by [ven::EventManager::moveCamera\(\)](#).

7.21.2.11 toggleGui

```
uint16_t ven::KeyMappings::toggleGui = GLFW_KEY_F1
```

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

Referenced by [ven::EventManager::handleEvents\(\)](#).

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

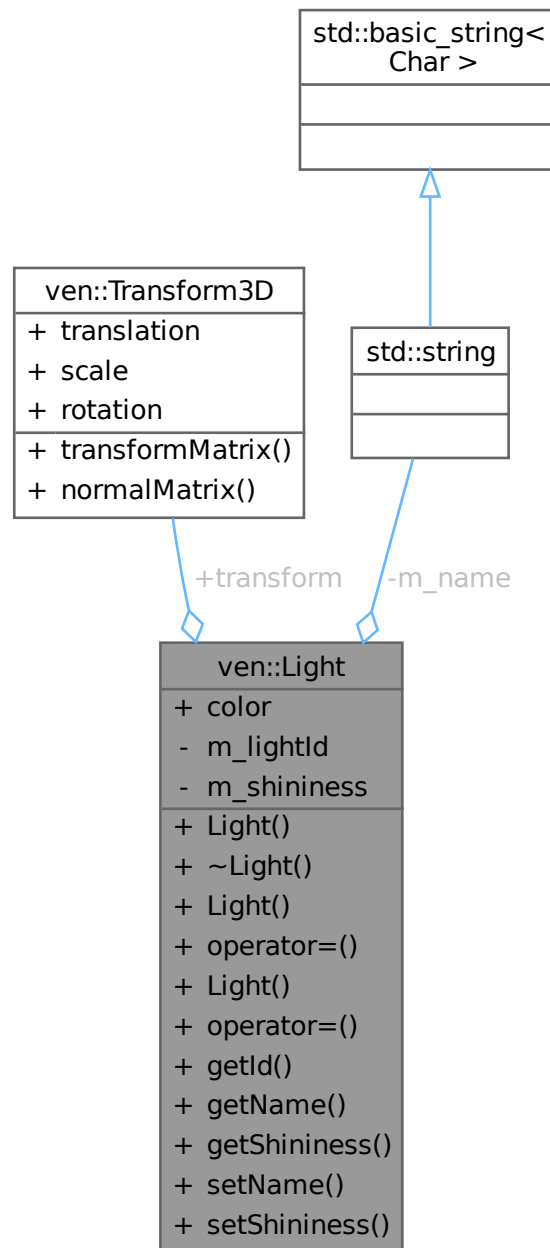
- [/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp](#)

7.22 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](#) (const unsigned int objId)
- [~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
- float [getShininess](#) () const
- void [setName](#) (const std::string &name)
- void [setShininess](#) (const float shininess)

Public Attributes

- glm::vec4 [color](#) {DEFAULT_LIGHT_COLOR}
- [Transform3D](#) [transform](#) {}

Private Attributes

- unsigned int [m_lightId](#)
- std::string [m_name](#) {"point light"}
- float [m_shininess](#) {DEFAULT_SHININESS}

7.22.1 Detailed Description

Class for light.

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

7.22.2 Member Typedef Documentation

7.22.2.1 Map

```
using ven::Light::Map = std::unordered_map<unsigned int, Light>
```

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

7.22.3 Constructor & Destructor Documentation

7.22.3.1 [Light](#)() [1/3]

```
ven::Light::Light (
    const unsigned int objId) [inline], [explicit]
```

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

7.22.3.2 ~Light()

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

7.22.3.3 Light() [2/3]

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

7.22.3.4 Light() [3/3]

```
ven::Light::Light (  
    Light && ) [default]
```

7.22.4 Member Function Documentation

7.22.4.1 getId()

```
unsigned int ven::Light::getId () const [inline], [nodiscard]
```

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

References [m_lightId](#).

Referenced by [ven::SceneManager::createLight\(\)](#).

Here is the caller graph for this function:



7.22.4.2 getName()

```
std::string ven::Light::getName () const [inline], [nodiscard]
```

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

References [m_name](#).

7.22.4.3 getShininess()

```
float ven::Light::getShininess () const [inline], [nodiscard]
```

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

References [m_shininess](#).

7.22.4.4 operator=() [1/2]

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

7.22.4.5 operator=() [2/2]

```
Light & ven::Light::operator= (  
    Light && ) [default]
```

7.22.4.6 setName()

```
void ven::Light::setName (  
    const std::string & name) [inline]
```

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

References [m_name](#).

7.22.4.7 setShininess()

```
void ven::Light::setShininess (  
    const float shininess) [inline]
```

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

References [m_shininess](#).

7.22.5 Member Data Documentation

7.22.5.1 color

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

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

Referenced by [ven::SceneManager::createLight\(\)](#).

7.22.5.2 m_lightId

unsigned int ven::Light::m_lightId [private]

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

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

7.22.5.3 m_name

std::string ven::Light::m_name {"point light"} [private]

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

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.22.5.4 m_shininess

float ven::Light::m_shininess {DEFAULT_SHININESS} [private]

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

Referenced by [getShininess\(\)](#), and [setShininess\(\)](#).

7.22.5.5 transform

[Transform3D](#) ven::Light::transform {}

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

Referenced by [ven::SceneManager::createLight\(\)](#).

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

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

7.23 ven::LightPushConstantData Struct Reference

```
#include <PointLightRenderSystem.hpp>
```

Collaboration diagram for ven::LightPushConstantData:

ven::LightPushConstantData	
+	position
+	color
+	radius

Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [radius](#)

7.23.1 Detailed Description

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

7.23.2 Member Data Documentation

7.23.2.1 color

```
glm::vec4 ven::LightPushConstantData::color {}
```

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

7.23.2.2 position

```
glm::vec4 ven::LightPushConstantData::position {}
```

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

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

7.23.2.3 radius

```
float ven::LightPushConstantData::radius
```

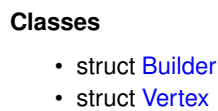
Definition at line 17 of file [PointLightRenderSystem.hpp](#).

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

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

Class for model.

Collaboration diagram for ven::Model:



Public Member Functions

- [Model](#) ([Device](#) &device, const [Builder](#) &builder)
- [~Model](#) ()=default
- [Model](#) (const [Model](#) &)=delete
- void [operator=](#) (const [Model](#) &)=delete
- void [bind](#) (VkCommandBuffer commandBuffer) const
- void [draw](#) (VkCommandBuffer commandBuffer) const

Static Public Member Functions

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

Private Member Functions

- void [createVertexBuffer](#) (const std::vector< [Vertex](#) > &vertices)
- void [createIndexBuffer](#) (const std::vector< uint32_t > &indices)

Private Attributes

- [Device](#) & [m_device](#)
- std::unique_ptr< [Buffer](#) > [m_vertexBuffer](#)
- uint32_t [m_vertexCount](#)
- bool [m_hasIndexBuffer](#) {false}
- std::unique_ptr< [Buffer](#) > [m_indexBuffer](#)
- uint32_t [m_indexCount](#)

7.24.1 Detailed Description

Class for model.

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

7.24.2 Constructor & Destructor Documentation

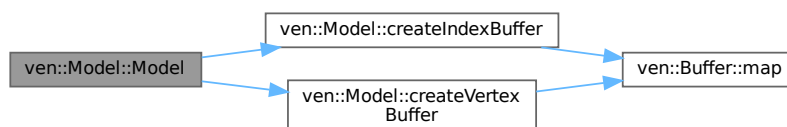
7.24.2.1 Model() [1/2]

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

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

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

Here is the call graph for this function:



7.24.2.2 ~Model()

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

7.24.2.3 Model() [2/2]

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

7.24.3 Member Function Documentation

7.24.3.1 bind()

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

Definition at line 78 of file [model.cpp](#).

7.24.3.2 createIndexBuffer()

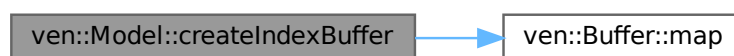
```
void ven::Model::createIndexBuffer (  
    const std::vector< uint32_t > & indices) [private]
```

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:



7.24.3.3 createModelFromFile()

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

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

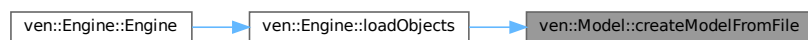
References [ven::Model::Builder::loadModel\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.3.4 createVertexBuffer()

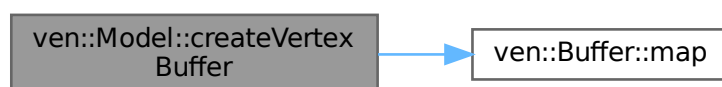
```
void ven::Model::createVertexBuffer (
    const std::vector< Vertex > & vertices) [private]
```

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:



7.24.3.5 draw()

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

Definition at line 69 of file [model.cpp](#).

7.24.3.6 operator=()

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

7.24.4 Member Data Documentation

7.24.4.1 m_device

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

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

7.24.4.2 m_hasIndexBuffer

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

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

7.24.4.3 m_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

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

7.24.4.4 m_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

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

7.24.4.5 m_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

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

7.24.4.6 m_vertexCount

```
uint32_t ven::Model::m_vertexCount [private]
```

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

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

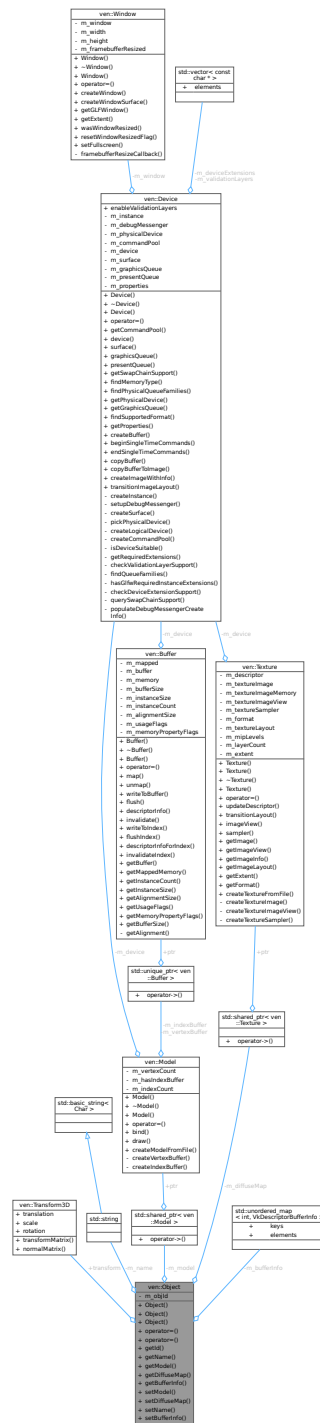
- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.25 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** (const unsigned int objId)

- [Object](#) (const [Object](#) &)=delete
- [Object](#) ([Object](#) &&)=default
- [Object](#) & [operator=](#) (const [Object](#) &)=delete
- [Object](#) & [operator=](#) ([Object](#) &&)=delete
- unsigned int [getId](#) () const
- std::string [getName](#) () const
- std::shared_ptr< [Model](#) > [getModel](#) () const
- std::shared_ptr< [Texture](#) > [getDiffuseMap](#) () const
- VkDescriptorBufferInfo [getBufferInfo](#) (const int frameIndex) const
- void [setModel](#) (const std::shared_ptr< [Model](#) > &model)
- void [setDiffuseMap](#) (const std::shared_ptr< [Texture](#) > &diffuseMap)
- void [setName](#) (const std::string &name)
- void [setBufferInfo](#) (const int frameIndex, const VkDescriptorBufferInfo &info)

Public Attributes

- [Transform3D](#) transform {}

Private Attributes

- unsigned int [m_objId](#)
- std::string [m_name](#)
- std::shared_ptr< [Model](#) > [m_model](#) = nullptr
- std::shared_ptr< [Texture](#) > [m_diffuseMap](#) = nullptr
- std::unordered_map< int, VkDescriptorBufferInfo > [m_bufferInfo](#)

7.25.1 Detailed Description

Class for object.

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

7.25.2 Member Typedef Documentation

7.25.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

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

7.25.3 Constructor & Destructor Documentation

7.25.3.1 Object() [1/3]

```
ven::Object::Object (  
    const unsigned int objId) [inline], [explicit]
```

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

7.25.3.2 Object() [2/3]

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

7.25.3.3 Object() [3/3]

```
ven::Object::Object (
    Object && ) [default]
```

7.25.4 Member Function Documentation

7.25.4.1 getBufferInfo()

```
VkDescriptorBufferInfo ven::Object::getBufferInfo (
    const int frameIndex) const [inline], [nodiscard]
```

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

References [m_bufferInfo](#).

7.25.4.2 getDiffuseMap()

```
std::shared_ptr< Texture > ven::Object::getDiffuseMap () const [inline], [nodiscard]
```

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

References [m_diffuseMap](#).

7.25.4.3 getId()

```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

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

References [m_objId](#).

7.25.4.4 getModel()

```
std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]
```

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

References [m_model](#).

7.25.4.5 getName()

```
std::string ven::Object::getName () const [inline], [nodiscard]
```

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

References [m_name](#).

7.25.4.6 operator=() [1/2]

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

7.25.4.7 operator=() [2/2]

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

7.25.4.8 setBufferInfo()

```
void ven::Object::setBufferInfo (  
    const int frameIndex,  
    const VkDescriptorBufferInfo & info) [inline]
```

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

References [m_bufferInfo](#).

7.25.4.9 setDiffuseMap()

```
void ven::Object::setDiffuseMap (  
    const std::shared_ptr< Texture > & diffuseMap) [inline]
```

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

References [m_diffuseMap](#).

7.25.4.10 setModel()

```
void ven::Object::setModel (  
    const std::shared_ptr< Model > & model) [inline]
```

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

References [m_model](#).

7.25.4.11 setName()

```
void ven::Object::setName (  
    const std::string & name) [inline]
```

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

References [m_name](#).

7.25.5 Member Data Documentation

7.25.5.1 m_bufferInfo

```
std::unordered_map<int, VkDescriptorBufferInfo> ven::Object::m_bufferInfo [private]
```

Definition at line 65 of file [Object.hpp](#).

Referenced by [getBufferInfo\(\)](#), and [setBufferInfo\(\)](#).

7.25.5.2 m_diffuseMap

```
std::shared_ptr<Texture> ven::Object::m_diffuseMap = nullptr [private]
```

Definition at line 64 of file [Object.hpp](#).

Referenced by [getDiffuseMap\(\)](#), and [setDiffuseMap\(\)](#).

7.25.5.3 m_model

```
std::shared_ptr<Model> ven::Object::m_model = nullptr [private]
```

Definition at line 63 of file [Object.hpp](#).

Referenced by [getModel\(\)](#), and [setModel\(\)](#).

7.25.5.4 m_name

```
std::string ven::Object::m_name [private]
```

Definition at line 62 of file [Object.hpp](#).

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.25.5.5 m_objId

```
unsigned int ven::Object::m_objId [private]
```

Definition at line 61 of file [Object.hpp](#).

Referenced by [getId\(\)](#).

7.25.5.6 transform

```
Transform3D ven::Object::transform {}
```

Definition at line 57 of file [Object.hpp](#).

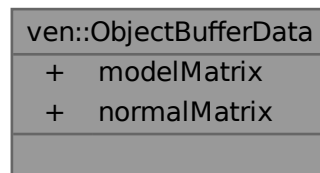
The documentation for this class was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#)

7.26 ven::ObjectBufferData Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::ObjectBufferData:



Public Attributes

- `glm::mat4` [modelMatrix](#) {1.F}
- `glm::mat4` [normalMatrix](#) {1.F}

7.26.1 Detailed Description

Definition at line 23 of file [Object.hpp](#).

7.26.2 Member Data Documentation

7.26.2.1 modelMatrix

```
glm::mat4 ven::ObjectBufferData::modelMatrix {1.F}
```

Definition at line 24 of file [Object.hpp](#).

Referenced by [ven::SceneManager::updateBuffer\(\)](#).

7.26.2.2 normalMatrix

`glm::mat4 ven::ObjectBufferData::normalMatrix {1.F}`

Definition at line 25 of file [Object.hpp](#).

The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp](#)

7.27 ven::ObjectPushConstantData Struct Reference

`#include <ObjectRenderSystem.hpp>`

Collaboration diagram for `ven::ObjectPushConstantData`:

ven::ObjectPushConstantData	
+	modelMatrix
+	normalMatrix

Public Attributes

- `glm::mat4` [modelMatrix](#) {}
- `glm::mat4` [normalMatrix](#) {}

7.27.1 Detailed Description

Definition at line 14 of file [ObjectRenderSystem.hpp](#).

7.27.2 Member Data Documentation

7.27.2.1 modelMatrix

`glm::mat4 ven::ObjectPushConstantData::modelMatrix {}`

Definition at line 15 of file [ObjectRenderSystem.hpp](#).

Referenced by [ven::ObjectRenderSystem::render\(\)](#).

7.27.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

Definition at line 16 of file [ObjectRenderSystem.hpp](#).

The documentation for this struct was generated from the following file:

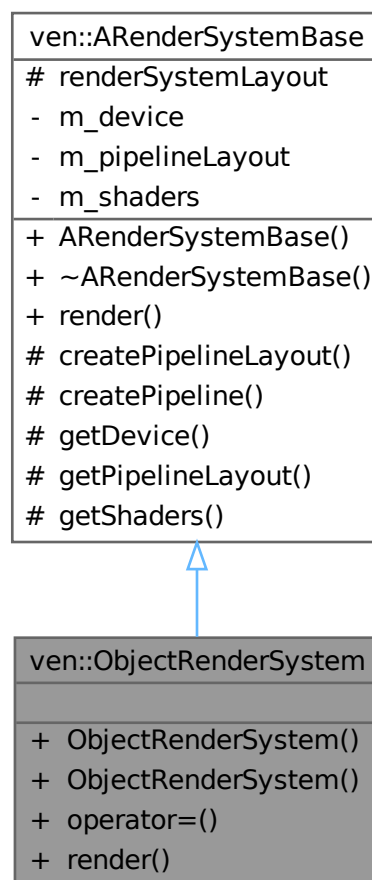
- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp](#)

7.28 ven::ObjectRenderSystem Class Reference

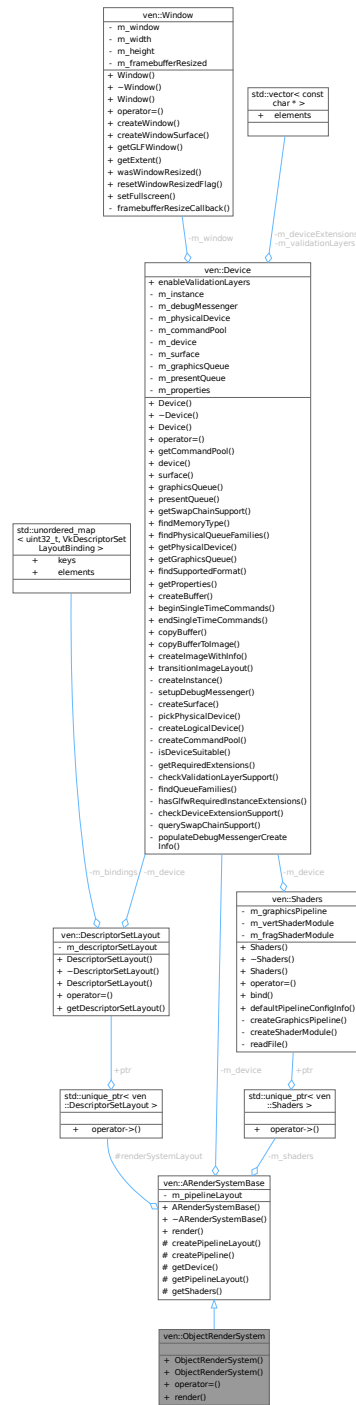
Class for object render system.

```
#include <ObjectRenderSystem.hpp>
```

Inheritance diagram for ven::ObjectRenderSystem:



Collaboration diagram for ven::ObjectRenderSystem:



Public Member Functions

- [ObjectRenderSystem](#) ([Device](#) &device, const [VkRenderPass](#) renderPass, const [VkDescriptorSetLayout](#) globalSetLayout)
- [ObjectRenderSystem](#) (const [ObjectRenderSystem](#) &)=delete
- [ObjectRenderSystem](#) & operator= (const [ObjectRenderSystem](#) &)=delete
- void [render](#) (const [FrameInfo](#) &frameInfo) const override

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- virtual [~ARenderSystemBase](#) ()

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

Protected Attributes inherited from [ven::ARenderSystemBase](#)

- std::unique_ptr< [DescriptorSetLayout](#) > [renderSystemLayout](#)

7.28.1 Detailed Description

Class for object render system.

Definition at line 24 of file [ObjectRenderSystem.hpp](#).

7.28.2 Constructor & Destructor Documentation

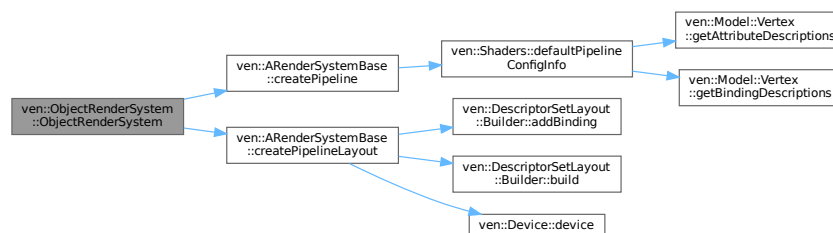
7.28.2.1 ObjectRenderSystem() [1/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 28 of file [ObjectRenderSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.28.2.2 ObjectRenderSystem() [2/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    const ObjectRenderSystem & ) [delete]
```

7.28.3 Member Function Documentation

7.28.3.1 operator=()

```
ObjectRenderSystem & ven::ObjectRenderSystem::operator= (
    const ObjectRenderSystem & ) [delete]
```

7.28.3.2 render()

```
void ven::ObjectRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

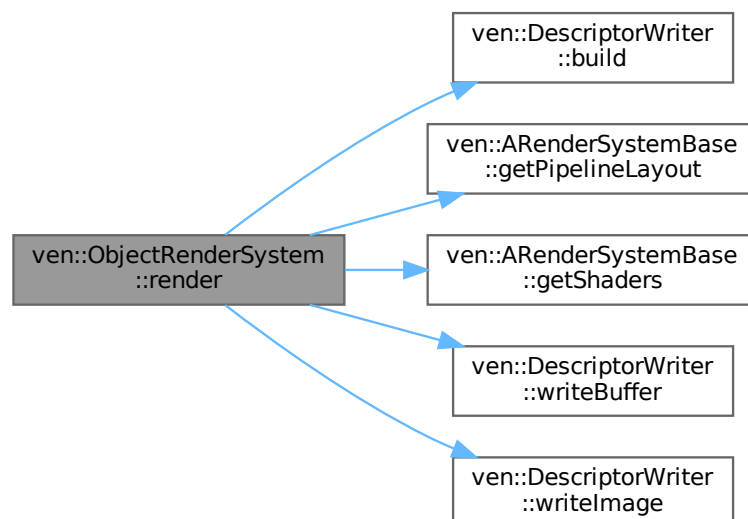
Implements [ven::ARenderSystemBase](#).

Definition at line 6 of file [objectRenderSystem.cpp](#).

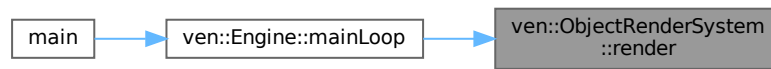
References [ven::DescriptorWriter::build\(\)](#), [ven::FrameInfo::commandBuffer](#), [ven::FrameInfo::frameDescriptorPool](#), [ven::FrameInfo::frameIndex](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders\(\)](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::ObjectPushConstantData::modelMatrix](#), [ven::FrameInfo::objects](#), [ven::ARenderSystemBase::renderSystemLayout](#), [ven::DescriptorWriter::writeBuffer\(\)](#), and [ven::DescriptorWriter::writeImage\(\)](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



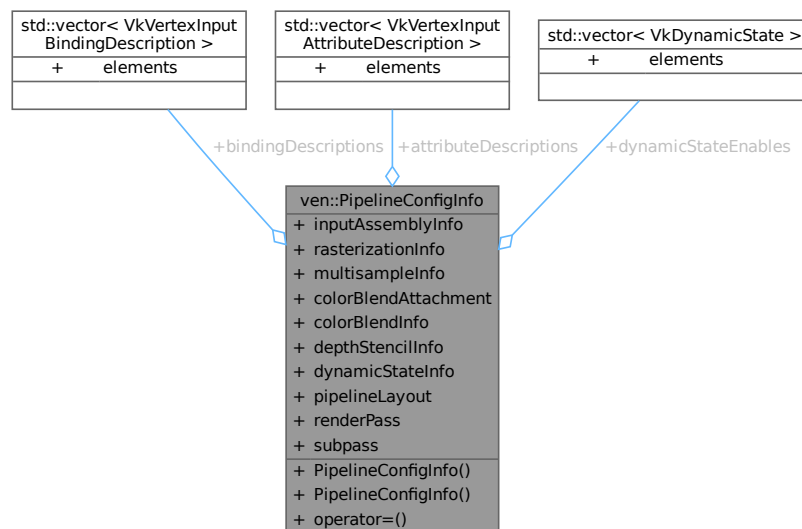
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp](#)

7.29 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

7.29.1 Detailed Description

Definition at line 19 of file [Shaders.hpp](#).

7.29.2 Constructor & Destructor Documentation

7.29.2.1 PipelineConfigInfo() [1/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo () [default]
```

7.29.2.2 PipelineConfigInfo() [2/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo (
    const PipelineConfigInfo & ) [delete]
```

7.29.3 Member Function Documentation

7.29.3.1 operator=()

```
PipelineConfigInfo & ven::PipelineConfigInfo::operator= (
    const PipelineConfigInfo & ) [delete]
```

7.29.4 Member Data Documentation

7.29.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

Definition at line 25 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

Definition at line 24 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

Definition at line 29 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

Definition at line 30 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

Definition at line 31 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

Definition at line 32 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

Definition at line 33 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

Definition at line 26 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

Definition at line 28 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

Definition at line 34 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.29.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

Definition at line 27 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.29.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

Definition at line 35 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.29.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 36 of file [Shaders.hpp](#).

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

The documentation for this struct was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)

7.30 ven::PointLightData Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::PointLightData:

ven::PointLightData
<ul style="list-style-type: none">+ position+ color+ shininess+ padding

Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [shininess](#) {32.F}
- float [padding](#) [3]

7.30.1 Detailed Description

Definition at line 21 of file [FrameInfo.hpp](#).

7.30.2 Member Data Documentation

7.30.2.1 color

```
glm::vec4 ven::PointLightData::color {}
```

Definition at line 24 of file [FrameInfo.hpp](#).

7.30.2.2 padding

```
float ven::PointLightData::padding[3]
```

Definition at line 26 of file [FrameInfo.hpp](#).

7.30.2.3 position

```
glm::vec4 ven::PointLightData::position {}
```

Definition at line 23 of file [FrameInfo.hpp](#).

7.30.2.4 shininess

```
float ven::PointLightData::shininess {32.F}
```

Definition at line 25 of file [FrameInfo.hpp](#).

The documentation for this struct was generated from the following file:

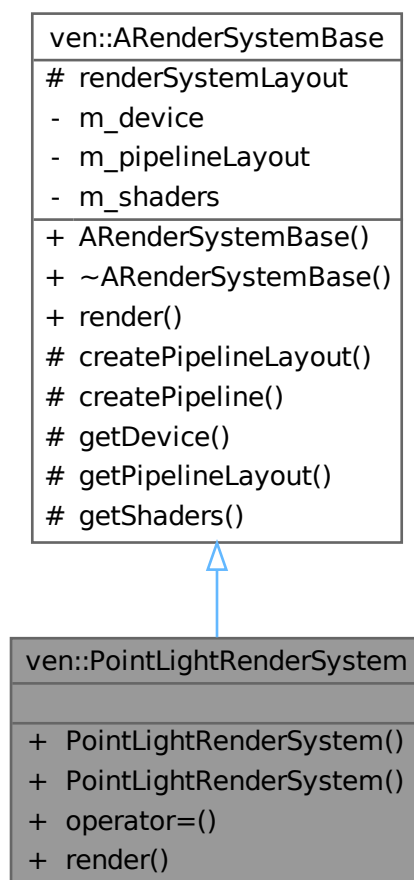
- [/home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp](#)

7.31 ven::PointLightRenderSystem Class Reference

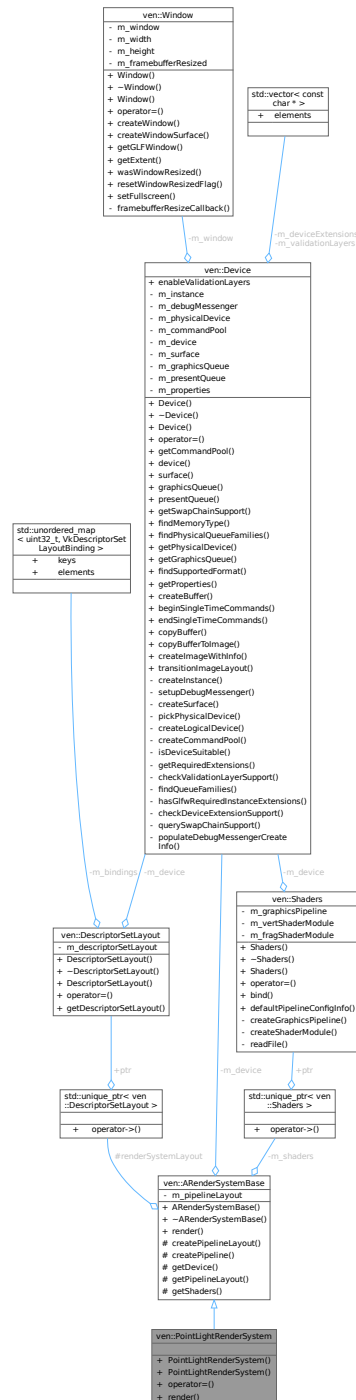
Class for point light system.

```
#include <PointLightRenderSystem.hpp>
```

Inheritance diagram for ven::PointLightRenderSystem:



Collaboration diagram for `ven::PointLightRenderSystem`:



Public Member Functions

- `PointLightRenderSystem` (`Device` &device, const `VkRenderPass` renderPass, const `VkDescriptorSetLayout` globalSetLayout)
- `PointLightRenderSystem` (const `PointLightRenderSystem` &)=delete
- `PointLightRenderSystem` & operator= (const `PointLightRenderSystem` &)=delete
- void `render` (const `FramelInfo` &frameInfo) const override

Public Member Functions inherited from [ven::ARenderSystemBase](#)

- [ARenderSystemBase](#) ([Device](#) &device)
- virtual [~ARenderSystemBase](#) ()

Additional Inherited Members

Protected Member Functions inherited from [ven::ARenderSystemBase](#)

- void [createPipelineLayout](#) (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void [createPipeline](#) (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- [Device](#) & [getDevice](#) () const
- VkPipelineLayout [getPipelineLayout](#) () const
- const std::unique_ptr< [Shaders](#) > & [getShaders](#) () const

Protected Attributes inherited from [ven::ARenderSystemBase](#)

- std::unique_ptr< [DescriptorSetLayout](#) > [renderSystemLayout](#)

7.31.1 Detailed Description

Class for point light system.

Definition at line 25 of file [PointLightRenderSystem.hpp](#).

7.31.2 Constructor & Destructor Documentation

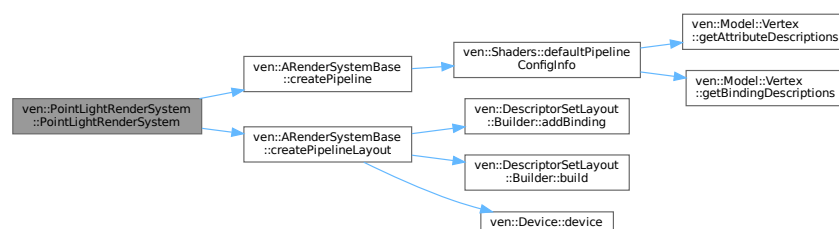
7.31.2.1 PointLightRenderSystem() [1/2]

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalSetLayout) [inline], [explicit]
```

Definition at line 29 of file [PointLightRenderSystem.hpp](#).

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.31.2.2 PointLightRenderSystem() [2/2]

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    const PointLightRenderSystem & ) [delete]
```

7.31.3 Member Function Documentation

7.31.3.1 operator=()

```
PointLightRenderSystem & ven::PointLightRenderSystem::operator= (
    const PointLightRenderSystem & ) [delete]
```

7.31.3.2 render()

```
void ven::PointLightRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

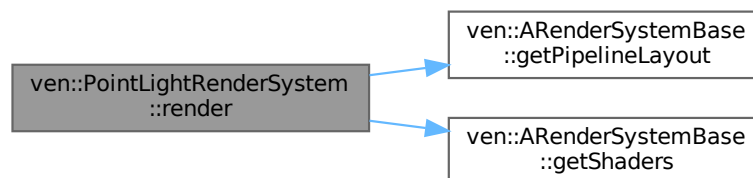
Implements [ven::ARenderSystemBase](#).

Definition at line 5 of file [pointLightRenderSystem.cpp](#).

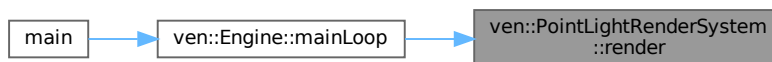
References [ven::FrameInfo::commandBuffer](#), [ven::ARenderSystemBase::getPipelineLayout\(\)](#), [ven::ARenderSystemBase::getShaders](#), [ven::FrameInfo::globalDescriptorSet](#), [ven::FrameInfo::lights](#), and [ven::LightPushConstantData::position](#).

Referenced by [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



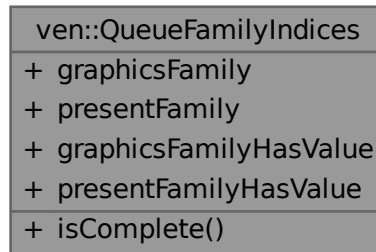
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp](#)

7.32 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

7.32.1 Detailed Description

Definition at line 21 of file [Device.hpp](#).

7.32.2 Member Function Documentation

7.32.2.1 isComplete()

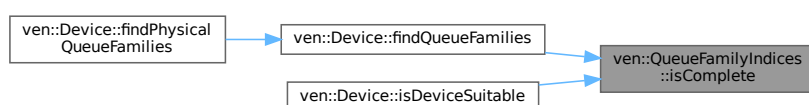
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

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:



7.32.3 Member Data Documentation

7.32.3.1 graphicsFamily

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

Definition at line 22 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.32.3.2 graphicsFamilyHasValue

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 24 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

7.32.3.3 presentFamily

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

Definition at line 23 of file [Device.hpp](#).

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.32.3.4 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:

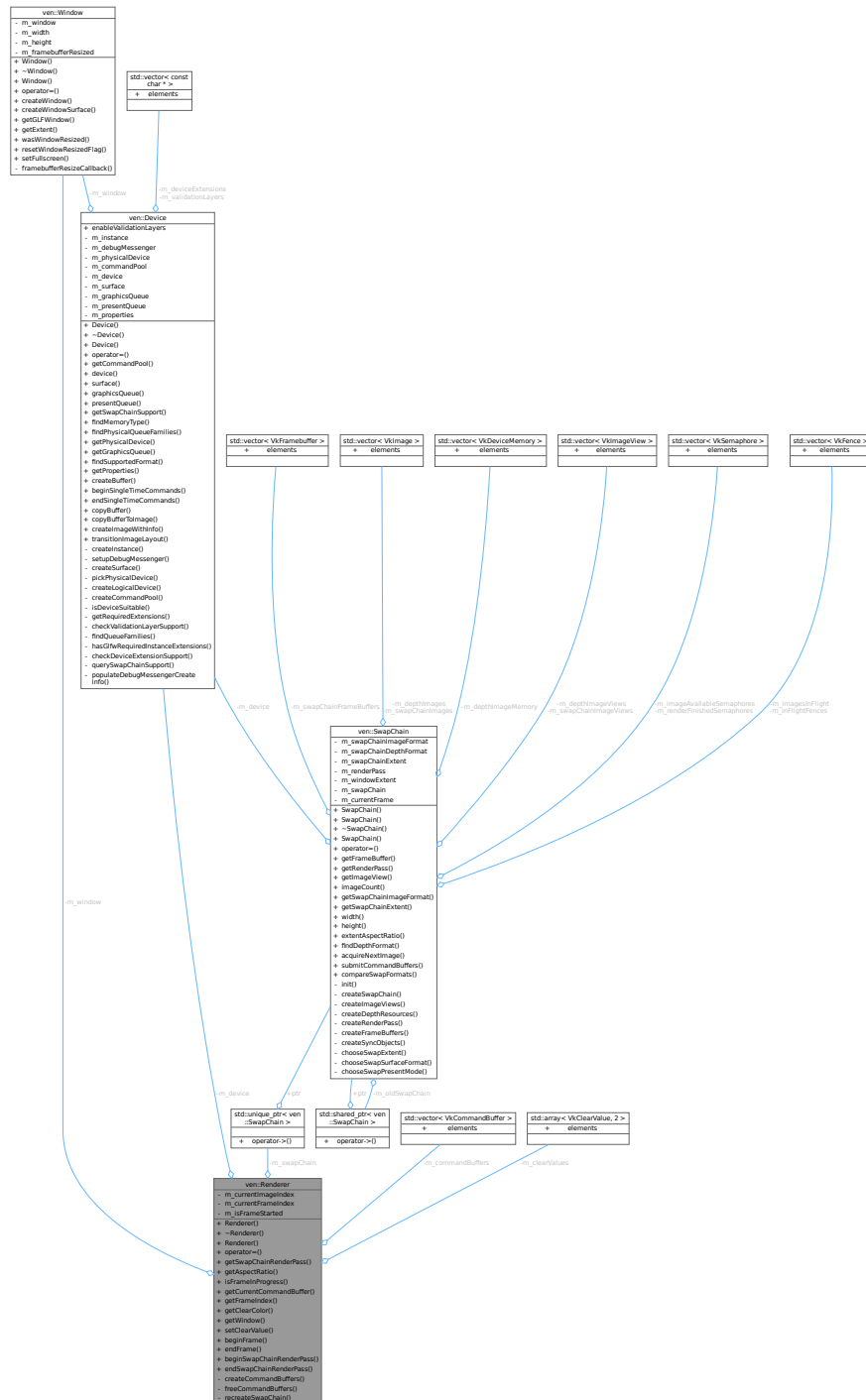
- [/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp](#)

7.33 ven::Renderer Class Reference

Class for renderer.

```
#include <Renderer.hpp>
```

Collaboration diagram for ven::Renderer:



Public Member Functions

- [Renderer](#) ([Window](#) &window, [Device](#) &device)
- [~Renderer](#) ()
- [Renderer](#) (const [Renderer](#) &)=delete
- [Renderer](#) & [operator=](#) (const [Renderer](#) &)=delete
- [VkRenderPass](#) [getSwapChainRenderPass](#) () const
- float [getAspectRatio](#) () const
- bool [isFrameInProgress](#) () const
- [VkCommandBuffer](#) [getCurrentCommandBuffer](#) () const
- unsigned long [getFrameIndex](#) () const
- std::array< float, 4 > [getClearColor](#) () const
- [Window](#) & [getWindow](#) () const
- void [setClearValue](#) (const [VkClearColorValue](#) clearColorValue=[DEFAULT_CLEAR_COLOR](#), const [VkClearDepthStencilValue](#) clearDepthValue=[DEFAULT_CLEAR_DEPTH](#))
- [VkCommandBuffer](#) [beginFrame](#) ()
- void [endFrame](#) ()
- void [beginSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer) const
- void [endSwapChainRenderPass](#) ([VkCommandBuffer](#) commandBuffer) const

Private Member Functions

- void [createCommandBuffers](#) ()
- void [freeCommandBuffers](#) ()
- void [recreateSwapChain](#) ()

Private Attributes

- [Window](#) & [m_window](#)
- [Device](#) & [m_device](#)
- std::unique_ptr< [SwapChain](#) > [m_swapChain](#)
- std::vector< [VkCommandBuffer](#) > [m_commandBuffers](#)
- std::array< [VkClearColorValue](#), 2 > [m_clearValues](#) {[DEFAULT_CLEAR_COLOR](#), 1.0F, 0.F}
- uint32_t [m_currentImageIndex](#) {0}
- unsigned long [m_currentFrameIndex](#) {0}
- bool [m_isFrameStarted](#) {false}

7.33.1 Detailed Description

Class for renderer.

Definition at line 28 of file [Renderer.hpp](#).

7.33.2 Constructor & Destructor Documentation

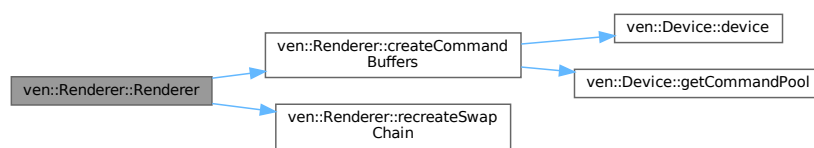
7.33.2.1 `Renderer()` [1/2]

```
ven::Renderer::Renderer (
    Window & window,
    Device & device) [inline]
```

Definition at line 32 of file [Renderer.hpp](#).

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



7.33.2.2 `~Renderer()`

```
ven::Renderer::~~Renderer () [inline]
```

Definition at line 33 of file [Renderer.hpp](#).

References [freeCommandBuffers\(\)](#).

Here is the call graph for this function:



7.33.2.3 `Renderer()` [2/2]

```
ven::Renderer::Renderer (
    const Renderer & ) [delete]
```

7.33.3 Member Function Documentation

7.33.3.1 beginFrame()

VkCommandBuffer ven::Renderer::beginFrame ()

Definition at line 43 of file [renderer.cpp](#).

7.33.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (  
    VkCommandBuffer commandBuffer) const
```

Definition at line 89 of file [renderer.cpp](#).

7.33.3.3 createCommandBuffers()

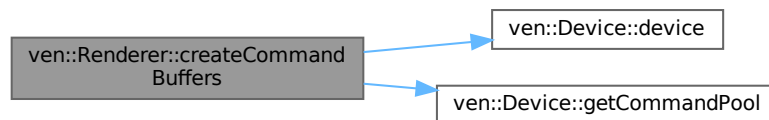
```
void ven::Renderer::createCommandBuffers () [private]
```

Definition at line 3 of file [renderer.cpp](#).

References [ven::Device::device\(\)](#), [ven::Device::getCommandPool\(\)](#), [m_commandBuffers](#), [m_device](#), and [ven::MAX_FRAMES_IN_FLIGHT](#).

Referenced by [Renderer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.3.4 endFrame()

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.33.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

Definition at line 119 of file [renderer.cpp](#).

7.33.3.6 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:



7.33.3.7 getAspectRatio()

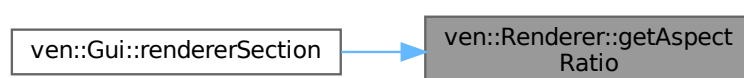
```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

Definition at line 39 of file [Renderer.hpp](#).

References [m_swapChain](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.33.3.8 getClearColor()

```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

Definition at line 44 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.33.3.9 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

Definition at line 41 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), [m_commandBuffers](#), and [m_currentFrameIndex](#).

Referenced by [ven::Gui::render\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



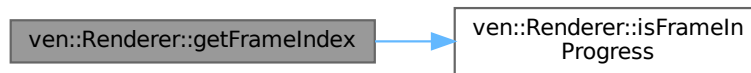
7.33.3.10 getFrameIndex()

```
unsigned long ven::Renderer::getFrameIndex () const [inline], [nodiscard]
```

Definition at line 43 of file [Renderer.hpp](#).

References [isFrameInProgress\(\)](#), and [m_currentFrameIndex](#).

Here is the call graph for this function:

**7.33.3.11 getSwapChainRenderPass()**

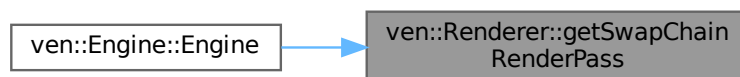
```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

Definition at line 38 of file [Renderer.hpp](#).

References [m_swapChain](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:

**7.33.3.12 getWindow()**

```
Window & ven::Renderer::getWindow () const [inline], [nodiscard]
```

Definition at line 51 of file [Renderer.hpp](#).

References [m_window](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.33.3.13 isFrameInProgress()

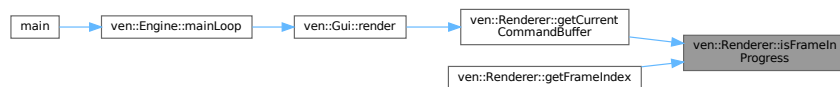
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

Definition at line 40 of file [Renderer.hpp](#).

References [m_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



7.33.3.14 operator=()

```
Renderer & ven::Renderer::operator= (
    const Renderer & ) [delete]
```

7.33.3.15 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:



7.33.3.16 setClearValue()

```
void ven::Renderer::setClearValue (
    const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

Definition at line 53 of file [Renderer.hpp](#).

References [m_clearValues](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.33.4 Member Data Documentation

7.33.4.1 m_clearValues

```
std::array<VkClearColorValue, 2> ven::Renderer::m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F}
[private]
```

Definition at line 69 of file [Renderer.hpp](#).

Referenced by [getClearColor\(\)](#), and [setClearValue\(\)](#).

7.33.4.2 m_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

Definition at line 68 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

7.33.4.3 m_currentFrameIndex

```
unsigned long ven::Renderer::m_currentFrameIndex {0} [private]
```

Definition at line 72 of file [Renderer.hpp](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

7.33.4.4 m_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

Definition at line 71 of file [Renderer.hpp](#).

7.33.4.5 m_device

```
Device& ven::Renderer::m_device [private]
```

Definition at line 66 of file [Renderer.hpp](#).

Referenced by [createCommandBuffers\(\)](#).

7.33.4.6 m_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

Definition at line 73 of file [Renderer.hpp](#).

Referenced by [isFrameInProgress\(\)](#).

7.33.4.7 m_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]
```

Definition at line 67 of file [Renderer.hpp](#).

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

7.33.4.8 m_window

```
Window& ven::Renderer::m_window [private]
```

Definition at line 65 of file [Renderer.hpp](#).

Referenced by [getWindow\(\)](#).

The documentation for this class was generated from the following files:

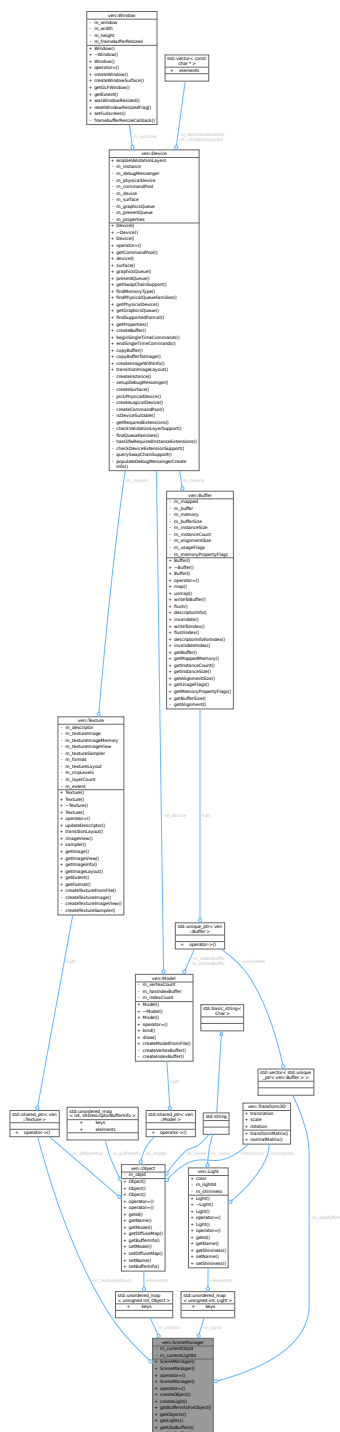
- [/home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/renderer.cpp](#)

7.34 ven::SceneManager Class Reference

Class for object manager.

```
#include <SceneManager.hpp>
```

Collaboration diagram for `ven::SceneManager`:



Public Member Functions

- [SceneManager](#) ([Device](#) &device)
- [SceneManager](#) (const [SceneManager](#) &)=delete
- [SceneManager](#) & operator= (const [SceneManager](#) &)=delete
- [SceneManager](#) ([SceneManager](#) &&)=delete
- [SceneManager](#) & operator= ([SceneManager](#) &&)=delete
- [Object](#) & createObject ()
- [Light](#) & createLight (float radius=[DEFAULT_LIGHT_RADIUS](#), glm::vec4 color=[DEFAULT_LIGHT_COLOR](#))
- [VkDescriptorBufferInfo](#) getBufferInfoForObject (const int frameIndex, const unsigned int objectId) const
- [Object::Map](#) & getObjects ()
- [Light::Map](#) & getLights ()
- std::vector< std::unique_ptr< [Buffer](#) > > & getUboBuffers ()
- void updateBuffer ([GlobalUbo](#) &ubo, unsigned long frameIndex, float frameTime)

Private Attributes

- unsigned int m_currentObjId {0}
- unsigned int m_currentLightId {0}
- std::shared_ptr< [Texture](#) > m_textureDefault
- [Object::Map](#) m_objects
- [Light::Map](#) m_lights
- std::vector< std::unique_ptr< [Buffer](#) > > m_uboBuffers {[MAX_FRAMES_IN_FLIGHT](#)}

7.34.1 Detailed Description

Class for object manager.

Definition at line 20 of file [SceneManager.hpp](#).

7.34.2 Constructor & Destructor Documentation

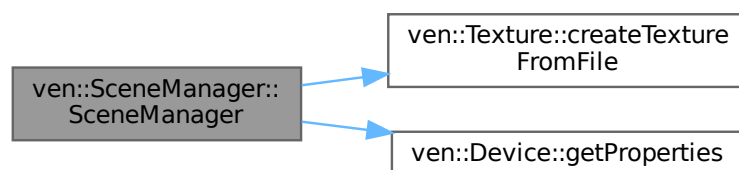
7.34.2.1 SceneManager() [1/3]

```
ven::SceneManager::SceneManager (
    Device & device) [explicit]
```

Definition at line 7 of file [sceneManager.cpp](#).

References [ven::Texture::createTextureFromFile\(\)](#), [ven::Device::getProperties\(\)](#), [m_textureDefault](#), [m_uboBuffers](#), and [ven::MAX_OBJECTS](#).

Here is the call graph for this function:



7.34.2.2 SceneManager() [2/3]

```
ven::SceneManager::SceneManager (
    const SceneManager & ) [delete]
```

7.34.2.3 SceneManager() [3/3]

```
ven::SceneManager::SceneManager (
    SceneManager && ) [delete]
```

7.34.3 Member Function Documentation

7.34.3.1 createLight()

```
ven::Light & ven::SceneManager::createLight (
    float radius = DEFAULT_LIGHT_RADIUS,
    glm::vec4 color = DEFAULT_LIGHT_COLOR)
```

Definition at line 37 of file [sceneManager.cpp](#).

References [ven::Light::color](#), [ven::Light::getId\(\)](#), [ven::MAX_LIGHTS](#), [ven::Transform3D::scale](#), and [ven::Light::transform](#).

Here is the call graph for this function:



7.34.3.2 createObject()

```
ven::Object & ven::SceneManager::createObject ()
```

Definition at line 27 of file [sceneManager.cpp](#).

References [ven::MAX_OBJECTS](#).

7.34.3.3 getBufferInfoForObject()

```
VkDescriptorBufferInfo ven::SceneManager::getBufferInfoForObject (
    const int frameIndex,
    const unsigned int objectId) const [inline]
```

Definition at line 34 of file [SceneManager.hpp](#).

References [m_uboBuffers](#).

7.34.3.4 getLights()

```
Light::Map & ven::SceneManager::getLights () [inline]
```

Definition at line 36 of file [SceneManager.hpp](#).

References [m_lights](#).

7.34.3.5 getObjects()

```
Object::Map & ven::SceneManager::getObjects () [inline]
```

Definition at line 35 of file [SceneManager.hpp](#).

References [m_objects](#).

7.34.3.6 getUboBuffers()

```
std::vector< std::unique_ptr< Buffer > > & ven::SceneManager::getUboBuffers () [inline]
```

Definition at line 37 of file [SceneManager.hpp](#).

References [m_uboBuffers](#).

7.34.3.7 operator=() [1/2]

```
SceneManager & ven::SceneManager::operator= (  
    const SceneManager & ) [delete]
```

7.34.3.8 operator=() [2/2]

```
SceneManager & ven::SceneManager::operator= (  
    SceneManager && ) [delete]
```

7.34.3.9 updateBuffer()

```
void ven::SceneManager::updateBuffer (  
    GlobalUbo & ubo,  
    unsigned long frameIndex,  
    float frameTime)
```

Definition at line 48 of file [sceneManager.cpp](#).

References [ven::ObjectBufferData::modelMatrix](#), [ven::GlobalUbo::numLights](#), and [ven::GlobalUbo::pointLights](#).

7.34.4 Member Data Documentation

7.34.4.1 m_currentLightId

```
unsigned int ven::SceneManager::m_currentLightId {0} [private]
```

Definition at line 44 of file [SceneManager.hpp](#).

7.34.4.2 m_currentObjId

```
unsigned int ven::SceneManager::m_currentObjId {0} [private]
```

Definition at line 43 of file [SceneManager.hpp](#).

7.34.4.3 m_lights

```
Light::Map ven::SceneManager::m_lights [private]
```

Definition at line 47 of file [SceneManager.hpp](#).

Referenced by [getLights\(\)](#).

7.34.4.4 m_objects

```
Object::Map ven::SceneManager::m_objects [private]
```

Definition at line 46 of file [SceneManager.hpp](#).

Referenced by [getObjects\(\)](#).

7.34.4.5 m_textureDefault

```
std::shared_ptr<Texture> ven::SceneManager::m_textureDefault [private]
```

Definition at line 45 of file [SceneManager.hpp](#).

Referenced by [SceneManager\(\)](#).

7.34.4.6 m_uboBuffers

```
std::vector<std::unique_ptr<Buffer> > ven::SceneManager::m_uboBuffers {MAX_FRAMES_IN_FLIGHT} [private]
```

Definition at line 48 of file [SceneManager.hpp](#).

Referenced by [getBufferInfoForObject\(\)](#), [getUboBuffers\(\)](#), and [SceneManager\(\)](#).

The documentation for this class was generated from the following files:

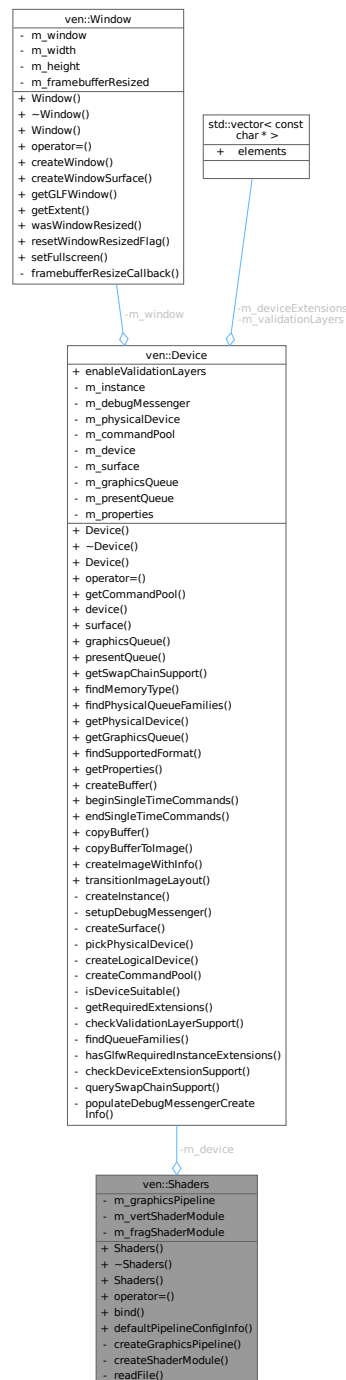
- [/home/runner/work/VEngine/VEngine/include/VEngine/SceneManager.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/sceneManager.cpp](#)

7.35 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}

7.35.1 Detailed Description

Class for shaders.

Definition at line 44 of file [Shaders.hpp](#).

7.35.2 Constructor & Destructor Documentation

7.35.2.1 Shaders() [1/2]

```
ven::Shaders::Shaders (
    Device & device,
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [inline]
```

Definition at line 48 of file [Shaders.hpp](#).

References [createGraphicsPipeline\(\)](#).

Here is the call graph for this function:



7.35.2.2 ~Shaders()

```
ven::Shaders::~~Shaders ()
```

Definition at line 8 of file [shaders.cpp](#).

References [ven::Device::device\(\)](#), [m_device](#), [m_fragShaderModule](#), [m_graphicsPipeline](#), and [m_vertShaderModule](#).

Here is the call graph for this function:



7.35.2.3 Shaders() [2/2]

```
ven::Shaders::Shaders (
    const Shaders & ) [delete]
```

7.35.3 Member Function Documentation

7.35.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

Definition at line 55 of file [Shaders.hpp](#).

References [m_graphicsPipeline](#).

7.35.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [private]
```

Definition at line 28 of file [shaders.cpp](#).

References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::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:



7.35.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

Definition at line 97 of file [shaders.cpp](#).

7.35.3.4 defaultPipelineConfigInfo()

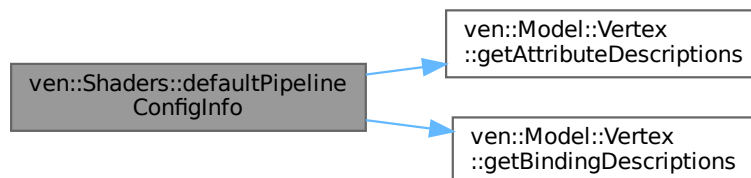
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 109 of file [shaders.cpp](#).

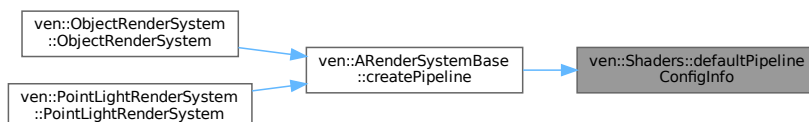
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::ARenderSystemBase::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.3.5 operator=()

```
Shaders & ven::Shaders::operator= (
    const Shaders & ) [delete]
```

7.35.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

Definition at line 15 of file [shaders.cpp](#).

7.35.4 Member Data Documentation

7.35.4.1 m_device

`Device& ven::Shaders::m_device [private]`

Definition at line 63 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.35.4.2 m_fragShaderModule

`VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]`

Definition at line 66 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

7.35.4.3 m_graphicsPipeline

`VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]`

Definition at line 64 of file [Shaders.hpp](#).

Referenced by [bind\(\)](#), and [~Shaders\(\)](#).

7.35.4.4 m_vertShaderModule

`VkShaderModule ven::Shaders::m_vertShaderModule {nullptr} [private]`

Definition at line 65 of file [Shaders.hpp](#).

Referenced by [~Shaders\(\)](#).

The documentation for this class was generated from the following files:

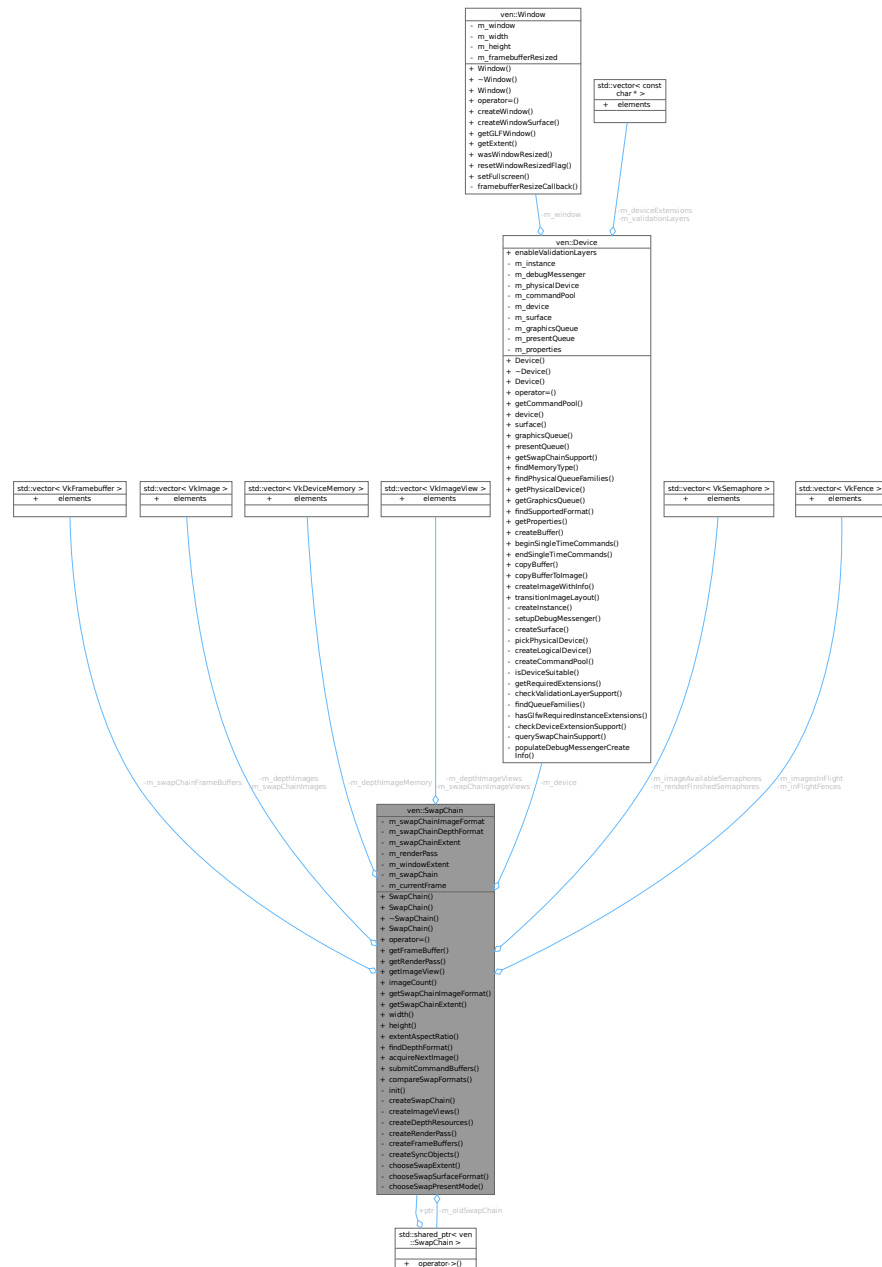
- [/home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/shaders.cpp](#)

7.36 ven::SwapChain Class Reference

Class for swap chain.

```
#include <SwapChain.hpp>
```

Collaboration diagram for ven::SwapChain:



Public Member Functions

- [SwapChain](#) ([Device](#) &deviceRef, const [VkExtent2D](#) windowExtentRef)
- [SwapChain](#) ([Device](#) &deviceRef, const [VkExtent2D](#) windowExtentRef, std::shared_ptr< [SwapChain](#) > previous)

- [~SwapChain](#) ()
- [SwapChain](#) (const [SwapChain](#) &)=delete
- [SwapChain](#) & [operator=](#) (const [SwapChain](#) &)=delete
- [VkFramebuffer](#) [getFramebuffer](#) (const unsigned long index) const
- [VkRenderPass](#) [getRenderPass](#) () const
- [VkImageView](#) [getImageView](#) (const int index) const
- [size_t](#) [imageCount](#) () const
- [VkFormat](#) [getSwapChainImageFormat](#) () const
- [VkExtent2D](#) [getSwapChainExtent](#) () const
- [uint32_t](#) [width](#) () const
- [uint32_t](#) [height](#) () const
- [float](#) [extentAspectRatio](#) () const
- [VkFormat](#) [findDepthFormat](#) () const
- [VkResult](#) [acquireNextImage](#) (uint32_t *imageIndex) const
- [VkResult](#) [submitCommandBuffers](#) (const [VkCommandBuffer](#) *buffers, const uint32_t *imageIndex)
- [bool](#) [compareSwapFormats](#) (const [SwapChain](#) &swapChain) const

Private Member Functions

- [void](#) [init](#) ()
- [void](#) [createSwapChain](#) ()
- [void](#) [createImageViews](#) ()
- [void](#) [createDepthResources](#) ()
- [void](#) [createRenderPass](#) ()
- [void](#) [createFrameBuffers](#) ()
- [void](#) [createSyncObjects](#) ()
- [VkExtent2D](#) [chooseSwapExtent](#) (const [VkSurfaceCapabilitiesKHR](#) &capabilities) const

Static Private Member Functions

- [static](#) [VkSurfaceFormatKHR](#) [chooseSwapSurfaceFormat](#) (const [std::vector](#)< [VkSurfaceFormatKHR](#) > &availableFormats)
- [static](#) [VkPresentModeKHR](#) [chooseSwapPresentMode](#) (const [std::vector](#)< [VkPresentModeKHR](#) > &availablePresentModes)

Private Attributes

- [VkFormat](#) [m_swapChainImageFormat](#) {}
- [VkFormat](#) [m_swapChainDepthFormat](#) {}
- [VkExtent2D](#) [m_swapChainExtent](#) {}
- [std::vector](#)< [VkFramebuffer](#) > [m_swapChainFrameBuffers](#)
- [VkRenderPass](#) [m_renderPass](#) {}
- [std::vector](#)< [VkImage](#) > [m_depthImages](#)
- [std::vector](#)< [VkDeviceMemory](#) > [m_depthImageMemory](#)
- [std::vector](#)< [VkImageView](#) > [m_depthImageViews](#)
- [std::vector](#)< [VkImage](#) > [m_swapChainImages](#)
- [std::vector](#)< [VkImageView](#) > [m_swapChainImageViews](#)
- [Device](#) & [m_device](#)
- [VkExtent2D](#) [m_windowExtent](#)
- [VkSwapchainKHR](#) [m_swapChain](#) {}
- [std::shared_ptr](#)< [SwapChain](#) > [m_oldSwapChain](#)
- [std::vector](#)< [VkSemaphore](#) > [m_imageAvailableSemaphores](#)
- [std::vector](#)< [VkSemaphore](#) > [m_renderFinishedSemaphores](#)
- [std::vector](#)< [VkFence](#) > [m_inFlightFences](#)
- [std::vector](#)< [VkFence](#) > [m_imagesInFlight](#)
- [size_t](#) [m_currentFrame](#) {0}

7.36.1 Detailed Description

Class for swap chain.

Definition at line 23 of file [SwapChain.hpp](#).

7.36.2 Constructor & Destructor Documentation

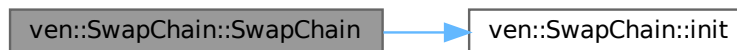
7.36.2.1 SwapChain() [1/3]

```
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:



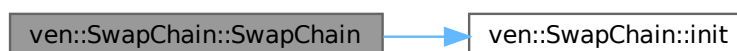
7.36.2.2 SwapChain() [2/3]

```
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:



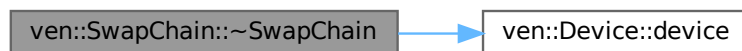
7.36.2.3 ~SwapChain()

```
ven::SwapChain::~~SwapChain ()
```

Definition at line 7 of file [swapChain.cpp](#).

References [ven::Device::device\(\)](#), [m_depthImageMemory](#), [m_depthImages](#), [m_depthImageViews](#), [m_device](#), [m_imageAvailableSemaphores](#), [m_inFlightFences](#), [m_renderFinishedSemaphores](#), [m_renderPass](#), [m_swapChain](#), [m_swapChainFrameBuffers](#), [m_swapChainImageViews](#), and [ven::MAX_FRAMES_IN_FLIGHT](#).

Here is the call graph for this function:



7.36.2.4 SwapChain() [3/3]

```
ven::SwapChain::SwapChain (
    const SwapChain & ) [delete]
```

7.36.3 Member Function Documentation

7.36.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

7.36.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

7.36.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

7.36.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (  
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

Definition at line 331 of file [swapChain.cpp](#).

7.36.3.5 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](#).

7.36.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

7.36.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

7.36.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

7.36.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

7.36.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

Definition at line 103 of file [swapChain.cpp](#).

7.36.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.36.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

Definition at line 43 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.36.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

7.36.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (  
    const unsigned long index) const [inline], [nodiscard]
```

Definition at line 34 of file [SwapChain.hpp](#).

References [m_swapChainFrameBuffers](#).

7.36.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (  
    const int index) const [inline], [nodiscard]
```

Definition at line 36 of file [SwapChain.hpp](#).

References [m_swapChainImageViews](#).

7.36.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

Definition at line 35 of file [SwapChain.hpp](#).

References [m_renderPass](#).

7.36.3.17 getSwapChainExtent()

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

Definition at line 39 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.36.3.18 getSwapChainImageFormat()

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

Definition at line 38 of file [SwapChain.hpp](#).

References [m_swapChainImageFormat](#).

7.36.3.19 height()

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

Definition at line 41 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.36.3.20 imageCount()

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

Definition at line 37 of file [SwapChain.hpp](#).

References [m_swapChainImages](#).

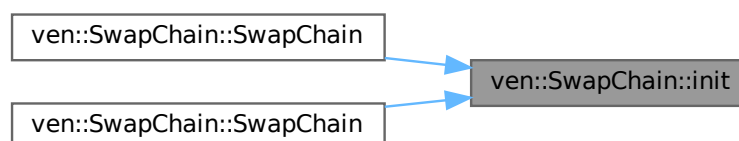
7.36.3.21 init()

```
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:



7.36.3.22 operator=()

```
SwapChain & ven::SwapChain::operator= (
    const SwapChain & ) [delete]
```

7.36.3.23 submitCommandBuffers()

```
VkResult ven::SwapChain::submitCommandBuffers (
    const VkCommandBuffer * buffers,
    const uint32_t * imageIndex)
```

Definition at line 56 of file [swapChain.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.36.3.24 width()

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

Definition at line 40 of file [SwapChain.hpp](#).

References [m_swapChainExtent](#).

7.36.4 Member Data Documentation

7.36.4.1 m_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0} [private]
```

Definition at line 88 of file [SwapChain.hpp](#).

7.36.4.2 m_depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::m_depthImageMemory [private]
```

Definition at line 73 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.3 m_depthImages

```
std::vector<VkImage> ven::SwapChain::m_depthImages [private]
```

Definition at line 72 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.4 m_depthImageViews

```
std::vector<VkImageView> ven::SwapChain::m_depthImageViews [private]
```

Definition at line 74 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.5 m_device

```
Device& ven::SwapChain::m_device [private]
```

Definition at line 78 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.6 m_imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_imageAvailableSemaphores [private]
```

Definition at line 84 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.7 m_imagesInFlight

```
std::vector<VkFence> ven::SwapChain::m_imagesInFlight [private]
```

Definition at line 87 of file [SwapChain.hpp](#).

7.36.4.8 m_inFlightFences

```
std::vector<VkFence> ven::SwapChain::m_inFlightFences [private]
```

Definition at line 86 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.9 m_oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::m_oldSwapChain [private]
```

Definition at line 82 of file [SwapChain.hpp](#).

Referenced by [SwapChain\(\)](#).

7.36.4.10 m_renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_renderFinishedSemaphores [private]
```

Definition at line 85 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.11 m_renderPass

```
VkRenderPass ven::SwapChain::m_renderPass {} [private]
```

Definition at line 70 of file [SwapChain.hpp](#).

Referenced by [getRenderPass\(\)](#), and [~SwapChain\(\)](#).

7.36.4.12 m_swapChain

```
VkSwapchainKHR ven::SwapChain::m_swapChain {} [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.36.4.13 m_swapChainDepthFormat

```
VkFormat ven::SwapChain::m_swapChainDepthFormat {} [private]
```

Definition at line 66 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#).

7.36.4.14 m_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

Definition at line 67 of file [SwapChain.hpp](#).

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

7.36.4.15 m_swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::m_swapChainFrameBuffers [private]
```

Definition at line 69 of file [SwapChain.hpp](#).

Referenced by [getFramebuffer\(\)](#), and [~SwapChain\(\)](#).

7.36.4.16 m_swapChainImageFormat

VkFormat ven::SwapChain::m_swapChainImageFormat {} [private]

Definition at line 65 of file [SwapChain.hpp](#).

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

7.36.4.17 m_swapChainImages

std::vector<VkImage> ven::SwapChain::m_swapChainImages [private]

Definition at line 75 of file [SwapChain.hpp](#).

Referenced by [imageCount\(\)](#).

7.36.4.18 m_swapChainImageViews

std::vector<VkImageView> ven::SwapChain::m_swapChainImageViews [private]

Definition at line 76 of file [SwapChain.hpp](#).

Referenced by [getImageView\(\)](#), and [~SwapChain\(\)](#).

7.36.4.19 m_windowExtent

VkExtent2D ven::SwapChain::m_windowExtent [private]

Definition at line 79 of file [SwapChain.hpp](#).

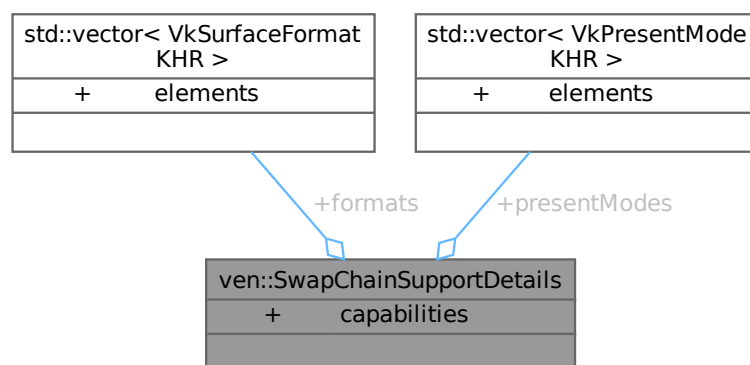
The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/swapChain.cpp](#)

7.37 ven::SwapChainSupportDetails Struct Reference

#include <Device.hpp>

Collaboration diagram for ven::SwapChainSupportDetails:



Public Attributes

- VkSurfaceCapabilitiesKHR [capabilities](#)
- std::vector< VkSurfaceFormatKHR > [formats](#)
- std::vector< VkPresentModeKHR > [presentModes](#)

7.37.1 Detailed Description

Definition at line 15 of file [Device.hpp](#).

7.37.2 Member Data Documentation

7.37.2.1 capabilities

```
VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities
```

Definition at line 16 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.37.2.2 formats

```
std::vector<VkSurfaceFormatKHR> ven::SwapChainSupportDetails::formats
```

Definition at line 17 of file [Device.hpp](#).

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.37.2.3 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](#)

7.38 ven::Texture Class Reference

Class for texture.

```
#include <Texture.hpp>
```

Collaboration diagram for ven::Texture:



Public Member Functions

- `Texture` (`Device` &device, const std::string &textureFilepath)
- `Texture` (`Device` &device, `VkFormat` format, `VkExtent3D` extent, `VkImageUsageFlags` usage, `VkSampleCountFlagBits` sampleCount)
- `~Texture` ()
- `Texture` (const `Texture` &)=delete
- `Texture` & `operator=` (const `Texture` &)=delete
- void `updateDescriptor` ()
- void `transitionLayout` (`VkCommandBuffer` commandBuffer, `VkImageLayout` oldLayout, `VkImageLayout` newLayout) const
- `VkImageView` `imageView` () const
- `VkSampler` `sampler` () const
- `VkImage` `getImage` () const
- `VkImageView` `getImageView` () const
- `VkDescriptorImageInfo` `getImageInfo` () const
- `VkImageLayout` `getImageLayout` () const
- `VkExtent3D` `getExtent` () const
- `VkFormat` `getFormat` () const

Static Public Member Functions

- static std::unique_ptr< `Texture` > `createTextureFromFile` (`Device` &device, const std::string &filepath)

Private Member Functions

- void `createTextureImage` (const std::string &filepath)
- void `createTextureImageView` (`VkImageViewType` viewType)
- void `createTextureSampler` ()

Private Attributes

- `VkDescriptorImageInfo` `m_descriptor` {}
- `Device` & `m_device`
- `VkImage` `m_textureImage` = nullptr
- `VkDeviceMemory` `m_textureImageMemory` = nullptr
- `VkImageView` `m_textureImageView` = nullptr
- `VkSampler` `m_textureSampler` = nullptr
- `VkFormat` `m_format`
- `VkImageLayout` `m_textureLayout` {}
- `uint32_t` `m_mipLevels` {1}
- `uint32_t` `m_layerCount` {1}
- `VkExtent3D` `m_extent` {}

7.38.1 Detailed Description

Class for texture.

Definition at line 23 of file `Texture.hpp`.

7.38.2 Constructor & Destructor Documentation

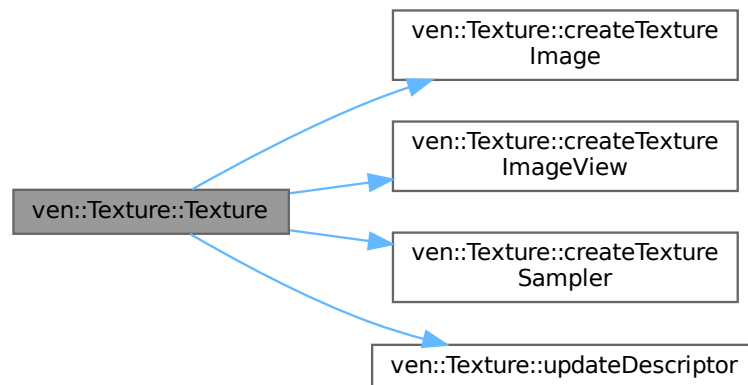
7.38.2.1 Texture() [1/3]

```
ven::Texture::Texture (
    Device & device,
    const std::string & textureFilepath)
```

Definition at line 8 of file [texture.cpp](#).

References [createTextureImage\(\)](#), [createTextureImageView\(\)](#), [createTextureSampler\(\)](#), and [updateDescriptor\(\)](#).

Here is the call graph for this function:



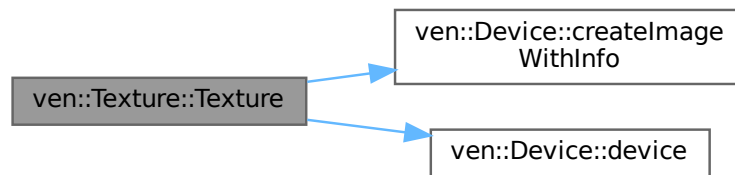
7.38.2.2 Texture() [2/3]

```
ven::Texture::Texture (
    Device & device,
    VkFormat format,
    VkExtent3D extent,
    VkImageUsageFlags usage,
    VkSampleCountFlagBits sampleCount)
```

Definition at line 16 of file [texture.cpp](#).

References [ven::Device::createImageWithInfo\(\)](#), [ven::Device::device\(\)](#), [m_descriptor](#), [m_textureImage](#), [m_textureImageMemory](#), [m_textureImageView](#), and [m_textureSampler](#).

Here is the call graph for this function:



7.38.2.3 ~Texture()

```
ven::Texture::~~Texture ()
```

Definition at line 90 of file [texture.cpp](#).

7.38.2.4 Texture() [3/3]

```
ven::Texture::Texture (
    const Texture & ) [delete]
```

7.38.3 Member Function Documentation

7.38.3.1 createTextureFromFile()

```
static std::unique_ptr< Texture > ven::Texture::createTextureFromFile (
    Device & device,
    const std::string & filepath) [inline], [static]
```

Definition at line 34 of file [Texture.hpp](#).

Referenced by [ven::SceneManager::SceneManager\(\)](#).

Here is the caller graph for this function:



7.38.3.2 createTextureImage()

```
void ven::Texture::createTextureImage (  
    const std::string & filepath) [private]
```

Definition at line 105 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



7.38.3.3 createTextureImageView()

```
void ven::Texture::createTextureImageView (  
    VkImageViewType viewType) [private]
```

Definition at line 192 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



7.38.3.4 createTextureSampler()

```
void ven::Texture::createTextureSampler () [private]
```

Definition at line 210 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



7.38.3.5 getExtent()

```
VkExtent3D ven::Texture::getExtent () const [inline], [nodiscard]
```

Definition at line 45 of file [Texture.hpp](#).

References [m_extent](#).

7.38.3.6 getFormat()

```
VkFormat ven::Texture::getFormat () const [inline], [nodiscard]
```

Definition at line 46 of file [Texture.hpp](#).

References [m_format](#).

7.38.3.7 getImage()

```
VkImage ven::Texture::getImage () const [inline], [nodiscard]
```

Definition at line 41 of file [Texture.hpp](#).

References [m_textureImage](#).

7.38.3.8 getImageInfo()

```
VkDescriptorImageInfo ven::Texture::getImageInfo () const [inline], [nodiscard]
```

Definition at line 43 of file [Texture.hpp](#).

References [m_descriptor](#).

7.38.3.9 getImageLayout()

```
VkImageLayout ven::Texture::getImageLayout () const [inline], [nodiscard]
```

Definition at line 44 of file [Texture.hpp](#).

References [m_textureLayout](#).

7.38.3.10 getImageView()

```
VkImageView ven::Texture::getImageView () const [inline], [nodiscard]
```

Definition at line 42 of file [Texture.hpp](#).

References [m_textureImageView](#).

7.38.3.11 imageView()

```
VkImageView ven::Texture::imageView () const [inline], [nodiscard]
```

Definition at line 39 of file [Texture.hpp](#).

References [m_textureImageView](#).

7.38.3.12 operator=()

```
Texture & ven::Texture::operator= (
    const Texture & ) [delete]
```

7.38.3.13 sampler()

```
VkSampler ven::Texture::sampler () const [inline], [nodiscard]
```

Definition at line 40 of file [Texture.hpp](#).

References [m_textureSampler](#).

7.38.3.14 transitionLayout()

```
void ven::Texture::transitionLayout (
    VkCommandBuffer commandBuffer,
    VkImageLayout oldLayout,
    VkImageLayout newLayout) const
```

Definition at line 240 of file [texture.cpp](#).

7.38.3.15 updateDescriptor()

```
void ven::Texture::updateDescriptor ()
```

Definition at line 98 of file [texture.cpp](#).

Referenced by [Texture\(\)](#).

Here is the caller graph for this function:



7.38.4 Member Data Documentation

7.38.4.1 m_descriptor

```
VkDescriptorImageInfo ven::Texture::m_descriptor {} [private]
```

Definition at line 54 of file [Texture.hpp](#).

Referenced by [getImageInfo\(\)](#), and [Texture\(\)](#).

7.38.4.2 m_device

```
Device& ven::Texture::m_device [private]
```

Definition at line 55 of file [Texture.hpp](#).

7.38.4.3 m_extent

```
VkExtent3D ven::Texture::m_extent {} [private]
```

Definition at line 64 of file [Texture.hpp](#).

Referenced by [getExtent\(\)](#).

7.38.4.4 m_format

```
VkFormat ven::Texture::m_format [private]
```

Definition at line 60 of file [Texture.hpp](#).

Referenced by [getFormat\(\)](#).

7.38.4.5 m_layerCount

```
uint32_t ven::Texture::m_layerCount {1} [private]
```

Definition at line 63 of file [Texture.hpp](#).

7.38.4.6 m_mipLevels

```
uint32_t ven::Texture::m_mipLevels {1} [private]
```

Definition at line 62 of file [Texture.hpp](#).

7.38.4.7 m_textureImage

```
VkImage ven::Texture::m_textureImage = nullptr [private]
```

Definition at line 56 of file [Texture.hpp](#).

Referenced by [getImage\(\)](#), and [Texture\(\)](#).

7.38.4.8 m_textureImageMemory

```
VkDeviceMemory ven::Texture::m_textureImageMemory = nullptr [private]
```

Definition at line 57 of file [Texture.hpp](#).

Referenced by [Texture\(\)](#).

7.38.4.9 m_textureImageView

```
VkImageView ven::Texture::m_textureImageView = nullptr [private]
```

Definition at line 58 of file [Texture.hpp](#).

Referenced by [getImageView\(\)](#), [imageView\(\)](#), and [Texture\(\)](#).

7.38.4.10 m_textureLayout

```
VkImageLayout ven::Texture::m_textureLayout {} [private]
```

Definition at line 61 of file [Texture.hpp](#).

Referenced by [getImageLayout\(\)](#).

7.38.4.11 m_textureSampler

```
VkSampler ven::Texture::m_textureSampler = nullptr [private]
```

Definition at line 59 of file [Texture.hpp](#).

Referenced by [sampler\(\)](#), and [Texture\(\)](#).

The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Texture.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/texture.cpp](#)

7.39 ven::Transform3D Class Reference

Class for 3D transformation.

```
#include <Transform3D.hpp>
```

Collaboration diagram for ven::Transform3D:

ven::Transform3D
+ translation
+ scale
+ rotation
+ transformMatrix()
+ normalMatrix()

Public Member Functions

- glm::mat4 [transformMatrix](#) () const
- glm::mat3 [normalMatrix](#) () const

Public Attributes

- glm::vec3 [translation](#) {}
- glm::vec3 [scale](#) {1.F, 1.F, 1.F}
- glm::vec3 [rotation](#) {}

7.39.1 Detailed Description

Class for 3D transformation.

Definition at line 18 of file [Transform3D.hpp](#).

7.39.2 Member Function Documentation

7.39.2.1 normalMatrix()

```
glm::mat3 ven::Transform3D::normalMatrix () const [inline], [nodiscard]
```

Definition at line 34 of file [Transform3D.hpp](#).

References [transformMatrix\(\)](#).

Here is the call graph for this function:



7.39.2.2 transformMatrix()

```
glm::mat4 ven::Transform3D::transformMatrix () const [inline], [nodiscard]
```

Definition at line 22 of file [Transform3D.hpp](#).

References [rotation](#), [scale](#), and [translation](#).

Referenced by [normalMatrix\(\)](#).

Here is the caller graph for this function:



7.39.3 Member Data Documentation

7.39.3.1 rotation

```
glm::vec3 ven::Transform3D::rotation {}
```

Definition at line 38 of file [Transform3D.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::EventManager::moveCamera\(\)](#), and [transformMatrix\(\)](#).

7.39.3.2 scale

```
glm::vec3 ven::Transform3D::scale {1.F, 1.F, 1.F}
```

Definition at line 37 of file [Transform3D.hpp](#).

Referenced by [ven::SceneManager::createLight\(\)](#), and [transformMatrix\(\)](#).

7.39.3.3 translation

```
glm::vec3 ven::Transform3D::translation {}
```

Definition at line 36 of file [Transform3D.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::EventManager::moveCamera\(\)](#), and [transformMatrix\(\)](#).

The documentation for this class was generated from the following file:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Transform3D.hpp](#)

7.40 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](#) {}

7.40.1 Detailed Description

Definition at line 29 of file [Model.hpp](#).

7.40.2 Member Function Documentation

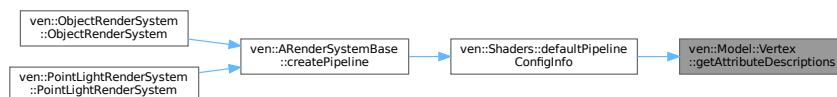
7.40.2.1 [getAttributeDescriptions\(\)](#)

```
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:



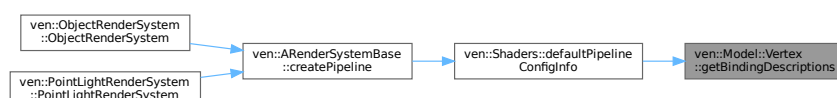
7.40.2.2 [getBindingDescriptions\(\)](#)

```
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:



7.40.2.3 operator==()

```
bool ven::Model::Vertex::operator== (
    const Vertex & other) const [inline]
```

Definition at line 38 of file [Model.hpp](#).

References [color](#), [normal](#), [position](#), and [uv](#).

7.40.3 Member Data Documentation

7.40.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 31 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.40.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 32 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.40.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 30 of file [Model.hpp](#).

Referenced by [operator==\(\)](#), and [ven::Model::Builder::processMesh\(\)](#).

7.40.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 33 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

The documentation for this struct was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/model.cpp](#)

7.41 ven::Window Class Reference

Class for window.

```
#include <Window.hpp>
```

Collaboration diagram for ven::Window:

ven::Window
<ul style="list-style-type: none"> - m_window - m_width - m_height - m_framebufferResized
<ul style="list-style-type: none"> + Window() + ~Window() + Window() + operator=() + createWindow() + createWindowSurface() + getGLFWWindow() + getExtent() + wasWindowResized() + resetWindowResizedFlag() + setFullscreen() - framebufferResizeCallback()

Public Member Functions

- [Window](#) (const uint32_t width=[DEFAULT_WIDTH](#), const uint32_t height=[DEFAULT_HEIGHT](#), const std::string &title=[DEFAULT_TITLE](#).data())
- [~Window](#) ()
- [Window](#) (const [Window](#) &)=delete
- [Window](#) & [operator=](#) (const [Window](#) &)=delete
- GLFWwindow * [createWindow](#) (uint32_t width, uint32_t height, const std::string &title)
- void [createWindowSurface](#) (VkInstance instance, VkSurfaceKHR *surface) const
- GLFWwindow * [getGLFWWindow](#) () const
- VkExtent2D [getExtent](#) () const
- bool [wasWindowResized](#) () const
- void [resetWindowResizedFlag](#) ()
- void [setFullscreen](#) (bool fullscreen, uint32_t width, uint32_t height)

Static Private Member Functions

- static void [framebufferResizeCallback](#) (GLFWwindow *window, int width, int height)

Private Attributes

- GLFWwindow * [m_window](#) {nullptr}
- uint32_t [m_width](#) {DEFAULT_WIDTH}
- uint32_t [m_height](#) {DEFAULT_HEIGHT}
- bool [m_framebufferResized](#) = false

7.41.1 Detailed Description

Class for window.

Definition at line 26 of file [Window.hpp](#).

7.41.2 Constructor & Destructor Documentation

7.41.2.1 Window() [1/2]

```
ven::Window::Window (
    const uint32_t width = DEFAULT\_WIDTH,
    const uint32_t height = DEFAULT\_HEIGHT,
    const std::string & title = DEFAULT\_TITLE.data()) [inline], [explicit]
```

Definition at line 30 of file [Window.hpp](#).

7.41.2.2 ~Window()

```
ven::Window::~Window () [inline]
```

Definition at line 31 of file [Window.hpp](#).

References [m_window](#).

7.41.2.3 Window() [2/2]

```
ven::Window::Window (
    const Window & ) [delete]
```

7.41.3 Member Function Documentation

7.41.3.1 `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:



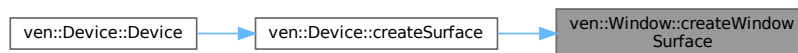
7.41.3.2 `createWindowSurface()`

```
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:



7.41.3.3 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:



7.41.3.4 getExtent()

```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

Definition at line 41 of file [Window.hpp](#).

References [m_height](#), and [m_width](#).

Referenced by [ven::Gui::renderSection\(\)](#).

Here is the caller graph for this function:



7.41.3.5 getGLFWWindow()

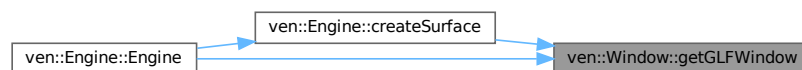
```
GLFWwindow * ven::Window::getGLFWWindow () const [inline], [nodiscard]
```

Definition at line 39 of file [Window.hpp](#).

References [m_window](#).

Referenced by [ven::Engine::createSurface\(\)](#), and [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.41.3.6 operator=()

```
Window & ven::Window::operator= (
    const Window & ) [delete]
```

7.41.3.7 resetWindowResizedFlag()

```
void ven::Window::resetWindowResizedFlag () [inline]
```

Definition at line 43 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.41.3.8 setFullscreen()

```
void ven::Window::setFullscreen (
    bool fullscreen,
    uint32_t width,
    uint32_t height)
```

Definition at line 39 of file [window.cpp](#).

Referenced by [ven::Gui::rendererSection\(\)](#).

Here is the caller graph for this function:



7.41.3.9 wasWindowResized()

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

Definition at line 42 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.41.4 Member Data Documentation

7.41.4.1 m_framebufferResized

```
bool ven::Window::m_framebufferResized = false [private]
```

Definition at line 55 of file [Window.hpp](#).

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).

7.41.4.2 m_height

```
uint32_t ven::Window::m_height {DEFAULT_HEIGHT} [private]
```

Definition at line 53 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.41.4.3 m_width

```
uint32_t ven::Window::m_width {DEFAULT_WIDTH} [private]
```

Definition at line 52 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.41.4.4 m_window

```
GLFWwindow* ven::Window::m_window {nullptr} [private]
```

Definition at line 51 of file [Window.hpp](#).

Referenced by [getGLFWWindow\(\)](#), and [~Window\(\)](#).

The documentation for this class was generated from the following files:

- [/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp](#)
- [/home/runner/work/VEngine/VEngine/src/window.cpp](#)

Chapter 8

File Documentation

8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_↵ point_light.frag File Reference

8.2 fragment_point_light.frag

[Go to the documentation of this file.](#)

```
00001 #version 450
00002
00003 layout(location = 0) in vec2 fragOffset;
00004 layout(location = 0) out vec4 outColor;
00005
00006 struct PointLight {
00007     vec4 position; // ignore w
00008     vec4 color; // w is intensity
00009     float shininess;
00010 };
00011
00012 layout(set = 0, binding = 0) uniform GlobalUbo {
00013     mat4 projection;
00014     mat4 view;
00015     mat4 invView;
00016     vec4 ambientLightColor; // w is intensity
00017     PointLight pointLights[10];
00018     int numLights;
00019 } ubo;
00020
00021 layout(push_constant) uniform Push {
00022     vec4 position;
00023     vec4 color;
00024     float radius;
00025 } push;
00026
00027 const float M_PI = 3.1415926538;
00028
00029 void main() {
00030     float dis = length(fragOffset);
00031     if (dis >= 1.0) {
00032         discard;
00033     }
00034
00035     float cosDis = 0.5 * (cos(dis * M_PI) + 1.0);
00036     outColor = vec4(push.color.rgb + 0.5 * cosDis, cosDis);
00037 }
```

8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_↵ shader.frag File Reference

8.4 fragment_shader.frag

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 fragColor;
00004 layout(location = 1) in vec3 fragPosWorld;
00005 layout(location = 2) in vec3 fragNormalWorld;
00006 layout(location = 3) in vec2 fragUv;
00007
00008 layout(location = 0) out vec4 outColor;
00009
00010 struct PointLight {
00011     vec4 position; // ignore w
00012     vec4 color; // w is intensity
00013     float shininess;
00014 };
00015
00016 layout(set = 0, binding = 0) uniform GlobalUbo {
00017     mat4 projection;
00018     mat4 view;
00019     mat4 invView;
00020     vec4 ambientLightColor; // w is intensity
00021     PointLight pointLights[10];
00022     int numLights;
00023 } ubo;
00024
00025 layout(set = 1, binding = 1) uniform sampler2D diffuseMap;
00026
00027 layout(push_constant) uniform Push {
00028     mat4 modelMatrix;
00029     mat4 normalMatrix;
00030 } push;
00031
00032 void main() {
00033     vec3 specularLight = vec3(0.0);
00034     vec3 surfaceNormal = normalize(gl_FrontFacing ? fragNormalWorld : -fragNormalWorld);
00035     vec3 diffuseLight = ubo.ambientLightColor.rgb * ubo.ambientLightColor.a;
00036
00037     vec3 cameraPosWorld = ubo.invView[3].xyz;
00038     vec3 viewDirection = normalize(cameraPosWorld - fragPosWorld);
00039
00040     for (int i = 0; i < ubo.numLights; i++) {
00041         PointLight light = ubo.pointLights[i];
00042         vec3 directionToLight = light.position.xyz - fragPosWorld;
00043         float distanceSquared = dot(directionToLight, directionToLight);
00044         float attenuation = distanceSquared > 0.001 ? (light.position.w + 1.0) / distanceSquared : 0.0;
00045         directionToLight = normalize(directionToLight);
00046
00047         float cosAngIncidence = max(dot(surfaceNormal, directionToLight), 0);
00048         vec3 intensity = light.color.rgb * light.color.a * attenuation;
00049
00050         if (cosAngIncidence > 0) {
00051             vec3 halfVector = normalize(directionToLight + viewDirection);
00052             float cosAngHalf = max(dot(surfaceNormal, halfVector), 0);
00053
00054             float specular = pow(cosAngHalf, light.shininess);
00055
00056             diffuseLight += intensity * cosAngIncidence;
00057             specularLight += intensity * specular;
00058         }
00059     }
00060
00061     vec3 color = texture(diffuseMap, fragUv).xyz;
00062     outColor = vec4(diffuseLight * color + specularLight, 1.0);
00063 }

```

8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_↵ light.vert File Reference

8.6 vertex_point_light.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 const vec2 OFFSETS[6] = vec2[](
00004     vec2(-1.0, -1.0),
00005     vec2(-1.0, 1.0),
00006     vec2(1.0, -1.0),
00007     vec2(1.0, 1.0),

```



```

00008 vec2(-1.0, 1.0),
00009 vec2(1.0, 1.0)
00010 );
00011
00012 layout(location = 0) out vec2 fragOffset;
00013
00014 struct PointLight {
00015     vec4 position; // ignore w
00016     vec4 color; // w is intensity
00017     float shininess;
00018 };
00019
00020 layout(set = 0, binding = 0) uniform GlobalUbo {
00021     mat4 projection;
00022     mat4 view;
00023     mat4 invView;
00024     vec4 ambientLightColor; // w is intensity
00025     PointLight pointLights[10];
00026     int numLights;
00027 } ubo;
00028
00029 layout(push_constant) uniform Push {
00030     vec4 position;
00031     vec4 color;
00032     float radius;
00033 } push;
00034
00035 void main() {
00036     fragOffset = OFFSETS[gl_VertexIndex];
00037     vec3 cameraRightWorld = vec3(ubo.view[0][0], ubo.view[1][0], ubo.view[2][0]);
00038     vec3 cameraUpWorld = vec3(ubo.view[0][1], ubo.view[1][1], ubo.view[2][1]);
00039
00040     vec3 positionWorld = push.position.xyz
00041     + push.radius * fragOffset.x * cameraRightWorld
00042     + push.radius * fragOffset.y * cameraUpWorld;
00043
00044     gl_Position = ubo.projection * ubo.view * vec4(positionWorld, 1.0);
00045 }

```

8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference

8.8 vertex_shader.vert

[Go to the documentation of this file.](#)

```

00001 #version 450
00002
00003 layout(location = 0) in vec3 position;
00004 layout(location = 1) in vec3 color;
00005 layout(location = 2) in vec3 normal;
00006 layout(location = 3) in vec2 uv;
00007
00008 layout(location = 0) out vec3 fragColor;
00009 layout(location = 1) out vec3 fragPosWorld;
00010 layout(location = 2) out vec3 fragNormalWorld;
00011 layout(location = 3) out vec2 fragUv;
00012
00013 struct PointLight {
00014     vec4 position; // ignore w
00015     vec4 color; // w is intensity
00016     float shininess;
00017 };
00018
00019 layout(set = 0, binding = 0) uniform GlobalUbo {
00020     mat4 projection;
00021     mat4 view;
00022     mat4 invView;
00023     vec4 ambientLightColor; // w is intensity
00024     PointLight pointLights[10];
00025     int numLights;
00026 } ubo;
00027
00028 layout(set = 1, binding = 0) uniform ObjectBufferData {
00029     mat4 modelMatrix;
00030     mat4 normalMatrix;
00031 } object;
00032

```

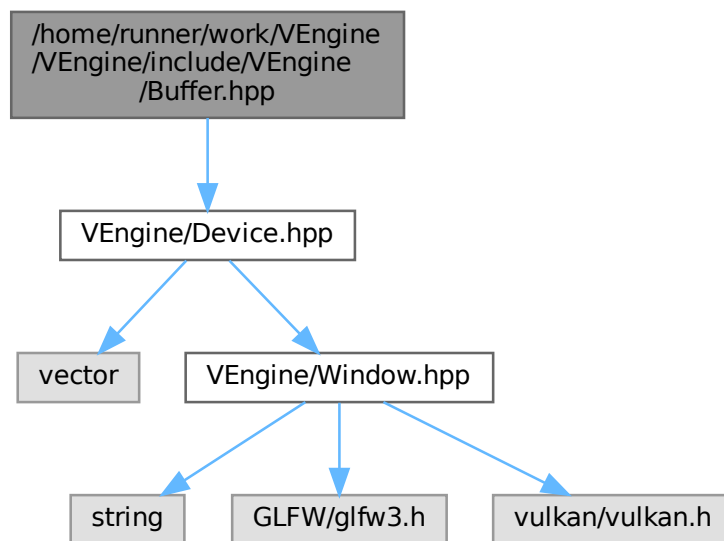
```
00033 layout(push_constant) uniform Push {
00034     mat4 modelMatrix;
00035     mat4 normalMatrix;
00036 } push;
00037
00038 void main() {
00039     vec4 positionWorld = object.modelMatrix * vec4(position, 1.0);
00040     gl_Position = ubo.projection * ubo.view * positionWorld;
00041     fragNormalWorld = normalize(mat3(object.normalMatrix) * normal);
00042     fragPosWorld = positionWorld.xyz;
00043     fragColor = color;
00044     fragUv = uv;
00045 }
```

8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

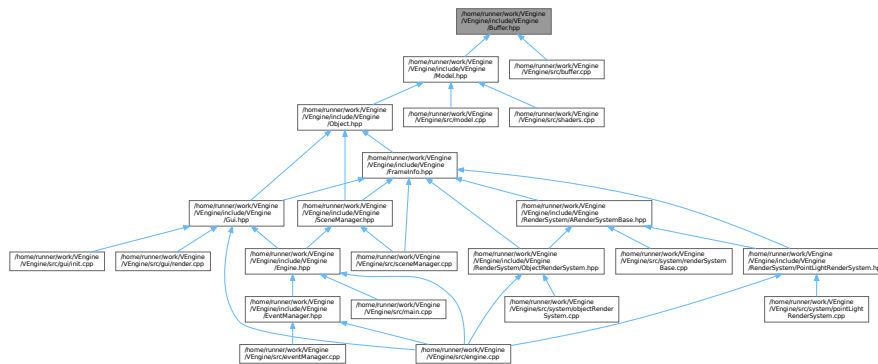
This file contains the Buffer class.

```
#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**

8.9.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

8.10 Buffer.hpp

[Go to the documentation of this file.](#)

```

00001 ///  

00002 ///  

00003 ///  

00004 ///  

00005 ///  

00006 ///  

00007 #pragma once  

00008 ///  

00009 #include "VEngine/Device.hpp"  

00010 ///  

00011 namespace ven {  

00012     ///  

00013     ///  

00014     ///  

00015     ///  

00016     ///  

00017     ///  

00018     class Buffer {  

00019     public:  

00020         Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,  

00021             VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize  

00022             minOffsetAlignment = 1);  

00023         ~Buffer();

```

```

00024
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     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); }
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 {
assert(m_alignmentSize % m_device.getProperties().limits.nonCoherentAtomSize == 0 && "Cannot use
LveBuffer::flushIndex if alignmentSize isn't a multiple of Device Limits nonCoherentAtomSize"); 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(const VkDeviceSize instanceSize, const VkDeviceSize
minOffsetAlignment) { return (minOffsetAlignment > 0) ? (instanceSize + minOffsetAlignment - 1) &
~(minOffsetAlignment - 1) : instanceSize; }
00146
00147         Device& m_device;
00148         void* m_mapped = nullptr;
00149         VkBuffer m_buffer = VK_NULL_HANDLE;
00150         VkDeviceMemory m_memory = VK_NULL_HANDLE;
00151
00152         VkDeviceSize m_bufferSize;
00153         VkDeviceSize m_instanceSize;
00154         uint32_t m_instanceCount;
00155         VkDeviceSize m_alignmentSize;
00156         VkBufferUsageFlags m_usageFlags;
00157         VkMemoryPropertyFlags m_memoryPropertyFlags;
00158
00159     }; // class Buffer
00160
00161 } // namespace ven

```

8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

This file contains the Camera class.

8.11.1 Detailed Description

This file contains the Camera class.

Definition in file [Camera.hpp](#).

8.12 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 "VEngine/Transform3D.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     static constexpr float DEFAULT_MOVE_SPEED = 3.F;
00021     static constexpr float DEFAULT_LOOK_SPEED = 1.5F;
00022
00023     ///
00024     /// @class Camera
00025     /// @brief Class for camera
00026     /// @namespace ven
00027     ///
00028     class Camera {
00029
00030     public:
00031
00032         Camera() = default;
00033         ~Camera() = default;
00034
00035         Camera(const Camera&) = delete;
00036         Camera& operator=(const Camera&) = delete;
00037
00038         void setOrthographicProjection(float left, float right, float top, float bottom, float
near, float far);
00039         void setPerspectiveProjection(float aspect);
00040         void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up = {0.F, -1.F,
0.F});
00041         void setViewTarget(const glm::vec3 position, const glm::vec3 target, const glm::vec3 up =
{0.F, -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00042         void setViewXYZ(glm::vec3 position, glm::vec3 rotation);
00043         void setFov(const float fov) { m_fov = fov; }
00044         void setNear(const float near) { m_near = near; }
00045         void setFar(const float far) { m_far = far; }
00046         void setMoveSpeed(const float moveSpeed) { m_moveSpeed = moveSpeed; }
00047         void setLookSpeed(const float lookSpeed) { m_lookSpeed = lookSpeed; }
00048
00049         [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00050         [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00051         [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00052         [[nodiscard]] float getFov() const { return m_fov; }
00053         [[nodiscard]] float getNear() const { return m_near; }
00054         [[nodiscard]] float getFar() const { return m_far; }
00055         [[nodiscard]] float getMoveSpeed() const { return m_moveSpeed; }
00056         [[nodiscard]] float getLookSpeed() const { return m_lookSpeed; }
00057
00058         Transform3D transform(DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION);
00059
00060     private:
00061
00062         float m_fov{DEFAULT_FOV};
00063         float m_near{DEFAULT_NEAR};
00064         float m_far{DEFAULT_FAR};
00065         float m_moveSpeed{DEFAULT_MOVE_SPEED};
00066         float m_lookSpeed{DEFAULT_LOOK_SPEED};

```


Namespaces

- namespace `ven`

Variables

- static constexpr uint32_t `ven::DEFAULT_MAX_SETS` = 1000

8.13.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file [DescriptorPool.hpp](#).

8.14 DescriptorPool.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file DescriptorPool.hpp
00003 /// @brief This file contains the DescriptorPool class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr uint32_t DEFAULT_MAX_SETS = 1000;
00016
00017     ///
00018     /// @class DescriptorPool
00019     /// @brief Class for descriptor pool
00020     /// @namespace ven
00021     ///
00022     class DescriptorPool {
00023
00024     public:
00025
00026         class Builder {
00027
00028         public:
00029
00030             explicit Builder(Device &device) : m_device{device} {}
00031
00032             [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00033
00034             Builder &addPoolSize(const VkDescriptorType descriptorType, const uint32_t count)
{ m_poolSizes.push_back({descriptorType, count}); return *this; }
00035             Builder &setPoolFlags(const VkDescriptorPoolCreateFlags flags) { m_poolFlags =
flags; return *this; }
00036             Builder &setMaxSets(const uint32_t count) { m_maxSets = count; return *this; }
00037
00038         private:
00039
00040             Device &m_device;
00041             std::vector<VkDescriptorPoolSize> m_poolSizes;
00042             uint32_t m_maxSets{DEFAULT_MAX_SETS};
00043             VkDescriptorPoolCreateFlags m_poolFlags{0};
00044
00045         }; // class Builder
00046
00047         DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
const std::vector<VkDescriptorPoolSize> &poolSizes);
00048         ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
}
00049     }
```


Classes

- class [ven::DescriptorSetLayout](#)
Class for descriptor set layout.
- class [ven::DescriptorSetLayout::Builder](#)

Namespaces

- namespace [ven](#)

8.15.1 Detailed Description

This file contains the DescriptorSetLayout class.

Definition in file [DescriptorSetLayout.hpp](#).

8.16 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
00038             Device &m_device;
00039             std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00040
00041         }; // class Builder
00042
00043         DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
00044                               VkDescriptorSetLayoutBinding>& bindings);
00045         ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(),
00046                               m_descriptorSetLayout, nullptr); }
00047
00048         DescriptorSetLayout(const DescriptorSetLayout &) = delete;
00049         DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;
00050
00051     };
00052
00053 }

```

```

00047         VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }
00048
00049     private:
00050
00051         Device &m_device;
00052         VkDescriptorSetLayout m_descriptorSetLayout;
00053         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00054
00055         friend class DescriptorWriter;
00056
00057     }; // class DescriptorSetLayout
00058
00059 } // namespace ven

```

8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference

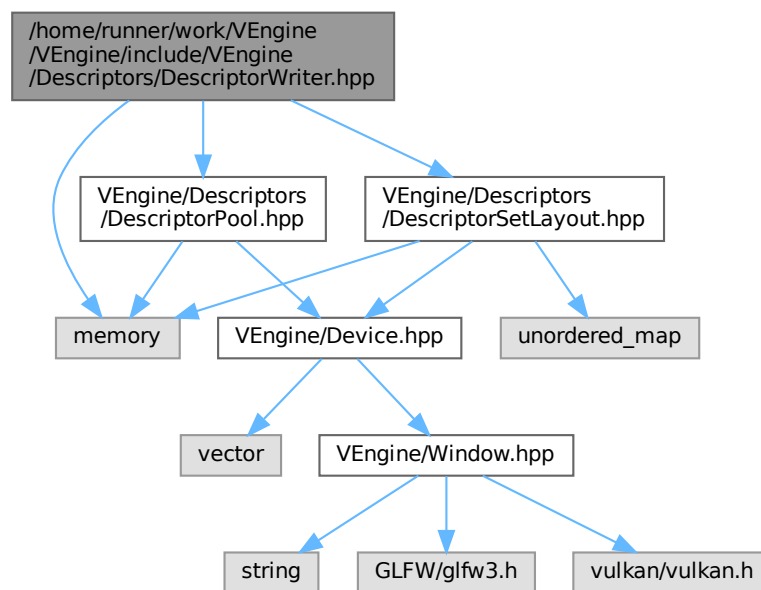
This file contains the DescriptorsWriter class.

```

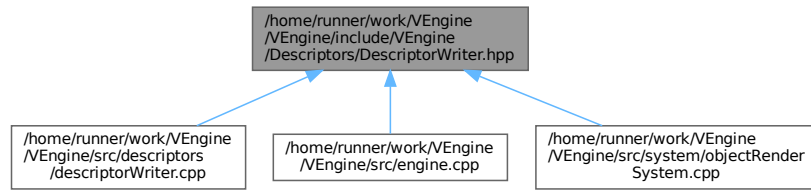
#include <memory>
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

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](#)

8.17.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file [DescriptorWriter.hpp](#).

8.18 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     public:
00023
00024         DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :
00025             m_setLayout{setLayout}, m_pool{pool} {}
00026         ~DescriptorWriter() = default;
00027
00028         DescriptorWriter(const DescriptorWriter &) = delete;
00029         DescriptorWriter &operator=(const DescriptorWriter &) = delete;
  
```

```

00030
00031     DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00032     DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00033
00034     bool build(VkDescriptorSet &set);
00035     void overwrite(const VkDescriptorSet &set);
00036
00037 private:
00038
00039     DescriptorSetLayout &m_setLayout;
00040     DescriptorPool &m_pool;
00041     std::vector<VkWriteDescriptorSet> m_writes;
00042
00043 }; // class DescriptorWriter
00044
00045 } // namespace ven

```

8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

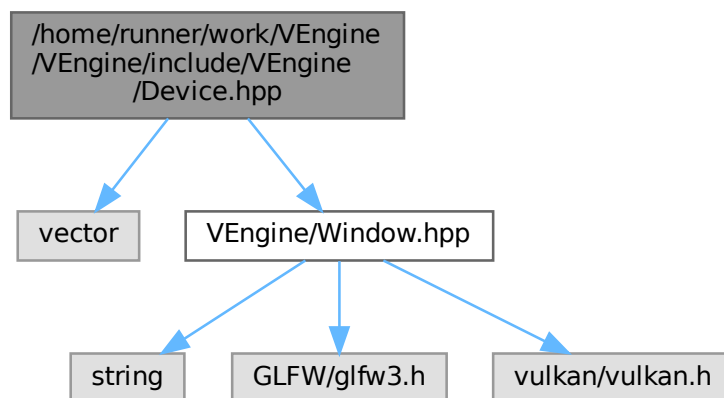
This file contains the Device class.

```

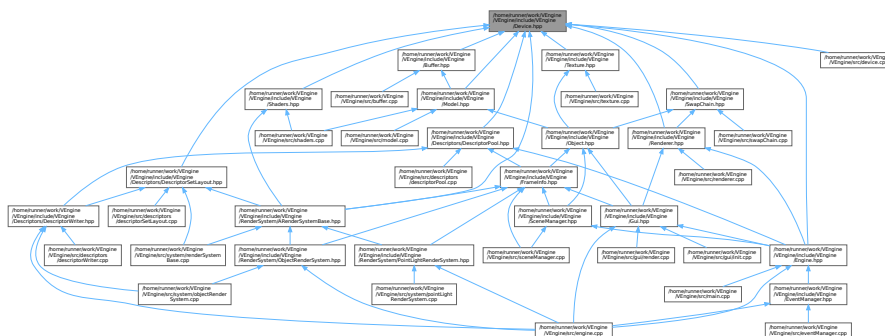
#include <vector>
#include "VEngine/Window.hpp"

```

Include dependency graph for Device.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::SwapChainSupportDetails](#)
- struct [ven::QueueFamilyIndices](#)
- class [ven::Device](#)

Class for device.

Namespaces

- namespace [ven](#)

8.19.1 Detailed Description

This file contains the Device class.

Definition in file [Device.hpp](#).

8.20 Device.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Device.hpp
00003 /// @brief This file contains the Device class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vector>
00010
00011 #include "VEngine/Window.hpp"
00012
00013 namespace ven {
00014
00015     struct SwapChainSupportDetails {
00016         VkSurfaceCapabilitiesKHR capabilities;
00017         std::vector<VkSurfaceFormatKHR> formats;
00018         std::vector<VkPresentModeKHR> presentModes;
00019     };
00020
00021     struct QueueFamilyIndices {
00022         uint32_t graphicsFamily{};
00023         uint32_t presentFamily{};
00024         bool graphicsFamilyHasValue = false;
00025         bool presentFamilyHasValue = false;
00026         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
presentFamilyHasValue; }
00027     };
00028
00029     ///
00030     /// @class Device
00031     /// @brief Class for device
00032     /// @namespace ven
00033     ///
00034     class Device {
00035     public:
00036
00037         #ifndef NDEBUB
00038             const bool enableValidationLayers = false;
00039         #else
00040             const bool enableValidationLayers = true;
00041         #endif
00042
00043         explicit Device(Window &window);
00044         ~Device();
00045
00046         Device(const Device&) = delete;
00047         Device& operator=(const Device&) = delete;

```

```

00049
00050 [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
00051 [[nodiscard]] VkDevice device() const { return m_device; }
00052 [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
00053 [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
00054 [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }
00055
00056 [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return
querySwapChainSupport(m_physicalDevice); }
00057 [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags
properties) const;
00058 [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
findQueueFamilies(m_physicalDevice); }
00059 [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00060 [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00061 [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
VkImageTiling tiling, VkFormatFeatureFlags features) const;
00062 [[nodiscard]] VkPhysicalDeviceProperties getProperties() const { return m_properties; }
00063
00064 // Buffer Helper Functions
00065 void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00066 [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00067 void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00068 void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00069 void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
uint32_t layerCount) const;
00070
00071 void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00072 void transitionImageLayout(VkImage image, VkFormat format, VkImageLayout oldLayout,
VkImageLayout newLayout, uint32_t mipLevels = 1, uint32_t layerCount = 1) const;
00073
00074 private:
00075
00076 void createInstance();
00077 void setupDebugMessenger();
00078 void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00079 void pickPhysicalDevice();
00080 void createLogicalDevice();
00081 void createCommandPool();
00082
00083 // helper functions
00084 bool isDeviceSuitable(VkPhysicalDevice device) const;
00085 [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00086 [[nodiscard]] bool checkValidationLayerSupport() const;
00087 QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00088 static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
&createInfo);
00089 void hasGlfwRequiredInstanceExtensions() const;
00090 bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00091 SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00092
00093 VkInstance m_instance;
00094 VkDebugUtilsMessengerEXT m_debugMessenger;
00095 VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00096 Window &m_window;
00097 VkCommandPool m_commandPool;
00098
00099 VkDevice m_device;
00100 VkSurfaceKHR m_surface;
00101 VkQueue m_graphicsQueue;
00102 VkQueue m_presentQueue;
00103 VkPhysicalDeviceProperties m_properties;
00104
00105 const std::vector<const char *> m_validationLayers = {"VK_LAYER_KHRONOS_validation"};
00106 const std::vector<const char *> m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00107
00108 }; // class Device
00109
00110 } // namespace ven

```

8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

This file contains the Engine class.

```

#include <vulkan/vulkan.h>
#include "VEngine/Gui.hpp"

```


[illegible]

```
graph TD; E_hpp["/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp"]; E_mgr_hpp["/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp"]; main_cpp["/home/runner/work/VEngine/VEngine/src/main.cpp"]; engine_cpp["/home/runner/work/VEngine/VEngine/src/engine.cpp"]; event_mgr_cpp["/home/runner/work/VEngine/VEngine/src/eventManager.cpp"]; E_hpp --> E_mgr_hpp; E_hpp --> main_cpp; E_mgr_hpp --> engine_cpp; E_mgr_hpp --> event_mgr_cpp; engine_cpp --> E_hpp; event_mgr_cpp --> E_hpp;
```

- class `ven::Engine`
Class for engine.

- namespace **ven**

- enum `ven::ENGINE_STATE` : `uint8_t` { `ven::EDITOR` = 0 , `ven::GAME` = 1 , `ven::PAUSED` = 2 , `ven::EXIT` = 3 }

8.21.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

8.22 Engine.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Engine.hpp
00003 /// @brief This file contains the Engine class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Gui.hpp"
00012 #include "VEngine/Window.hpp"
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Renderer.hpp"
00015 #include "VEngine/Descriptors/DescriptorPool.hpp"
00016 #include "VEngine/SceneManager.hpp"
00017
00018 namespace ven {
00019
00020     enum ENGINE_STATE : uint8_t {
00021         EDITOR = 0,
00022         GAME = 1,
00023         PAUSED = 2,
00024         EXIT = 3
00025     };
00026
00027     ///
00028     /// @class Engine
00029     /// @brief Class for engine
00030     /// @namespace ven
00031     ///
00032     class Engine {
00033     public:
00034
00035         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string
&title = DEFAULT_TITLE.data());
00036         ~Engine() = default;
00037
00038         Engine(const Engine&) = delete;
00039         Engine operator=(const Engine&) = delete;
00040
00041         void mainLoop();
00042
00043     private:
00044
00045         void loadObjects();
00046
00047         ENGINE_STATE m_state{EXIT};
00048
00049         Window m_window;
00050         Device m_device{m_window};
00051         Renderer m_renderer{m_window, m_device};
00052         Gui m_gui;
00053         std::unique_ptr<DescriptorPool> m_globalPool;
00054         std::vector<std::unique_ptr<DescriptorPool>> framePools;
00055         SceneManager m_sceneManager{m_device};
00056
00057         VkInstance m_instance{nullptr};
00058         VkSurfaceKHR m_surface{nullptr};
00059
00060         void createInstance();
00061         void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWWindow(),
nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00062
00063     }; // class Engine
00064
00065 } // namespace ven

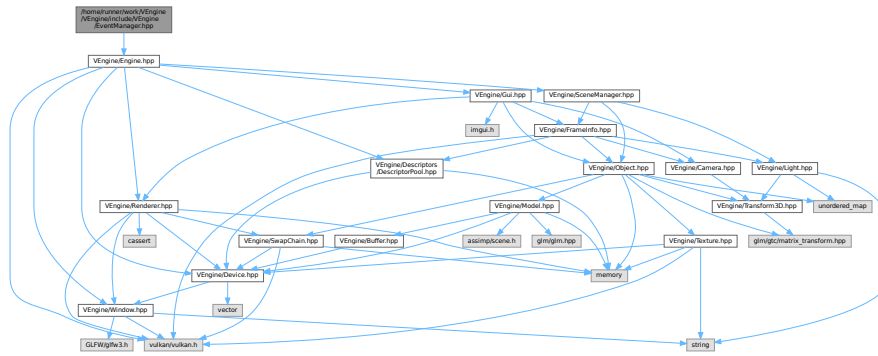
```

8.23 /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp File Reference

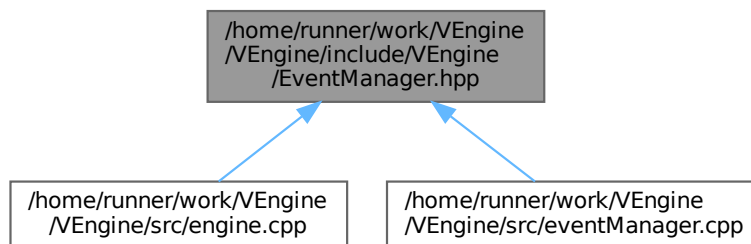
This file contains the EventManager class.

```
#include "VEngine/Engine.hpp"
```

Include dependency graph for EventManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::KeyAction](#)
- struct [ven::KeyMappings](#)
- class [ven::EventManager](#)
Class for event manager.

Namespaces

- namespace [ven](#)

Variables

- static constexpr float [ven::EPSILON](#) = std::numeric_limits<float>::epsilon()
- static constexpr [KeyMappings](#) [ven::DEFAULT_KEY_MAPPINGS](#) {}

8.23.1 Detailed Description

This file contains the EventManager class.

Definition in file [EventManager.hpp](#).

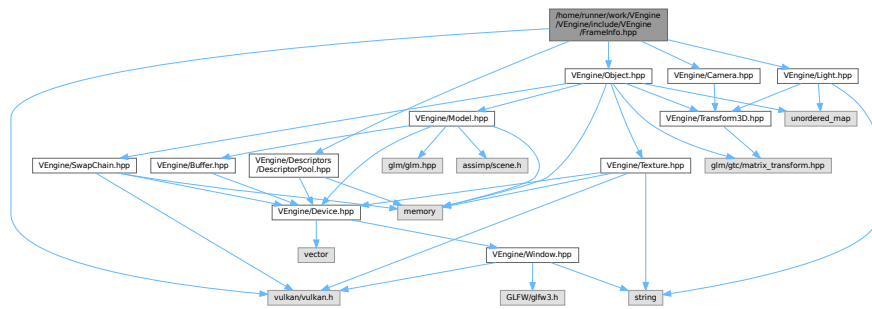
8.24 EventManager.hpp

[Go to the documentation of this file.](#)

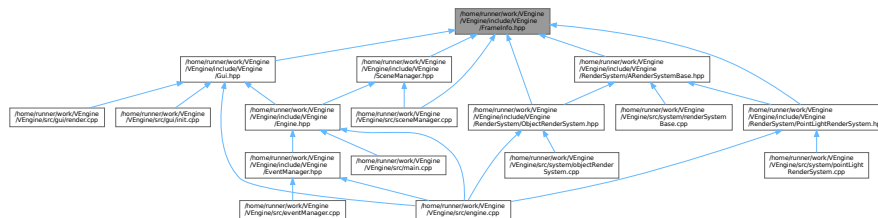
```
00001 ///
00002 /// @file EventManager.hpp
00003 /// @brief This file contains the EventManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Engine.hpp"
00010
00011 namespace ven {
00012
00013     struct KeyAction {
00014         uint16_t key;
00015         glm::vec3* dir;
00016         glm::vec3 value;
00017     };
00018
00019     struct KeyMappings {
00020         uint16_t moveLeft = GLFW_KEY_A;
00021         uint16_t moveRight = GLFW_KEY_D;
00022         uint16_t moveForward = GLFW_KEY_W;
00023         uint16_t moveBackward = GLFW_KEY_S;
00024         uint16_t moveUp = GLFW_KEY_SPACE;
00025         uint16_t moveDown = GLFW_KEY_LEFT_SHIFT;
00026         uint16_t lookLeft = GLFW_KEY_LEFT;
00027         uint16_t lookRight = GLFW_KEY_RIGHT;
00028         uint16_t lookUp = GLFW_KEY_UP;
00029         uint16_t lookDown = GLFW_KEY_DOWN;
00030         uint16_t toggleGui = GLFW_KEY_F1;
00031     };
00032
00033     static constexpr float EPSILON = std::numeric_limits<float>::epsilon();
00034     static constexpr KeyMappings DEFAULT_KEY_MAPPINGS{};
00035
00036     ///
00037     /// @class EventManager
00038     /// @brief Class for event manager
00039     /// @namespace ven
00040     ///
00041     class EventManager {
00042     public:
00043
00044         EventManager() = default;
00045         ~EventManager() = default;
00046
00047         EventManager(const EventManager&) = delete;
00048         EventManager& operator=(const EventManager&) = delete;
00049
00050         void handleEvents(GLFWwindow* window, ENGINE_STATE* engineState, Camera& camera, Gui& gui,
00051             float dt) const;
00052
00053     private:
00054         static void moveCamera(GLFWwindow* window, Camera& camera, float dt);
00055         static void updateEngineState(ENGINE_STATE* engineState, const ENGINE_STATE newState) {
00056             *engineState = newState; }
00056         static bool isKeyJustPressed(GLFWwindow* window, int key, std::unordered_map<int, bool>&
00057             keyStates);
00058
00059         template<typename Iterator>
00060         static void processKeyActions(GLFWwindow* window, Iterator begin, Iterator end);
00061
00062         mutable std::unordered_map<int, bool> m_keyState;
00063     }; // class EventManager
00064
00065 } // namespace ven
```

This file contains the FrameInfo class.

```
#include <vulkan/vulkan.h>
#include "VEngine/Descriptors/DescriptorPool.hpp"
#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:



- struct `ven::PointLightData`
- struct `ven::GlobalUbo`
- struct `ven::FrameInfo`

- namespace **ven**

- 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`}

8.25.1 Detailed Description

This file contains the FrameInfo class.

Definition in file [FrameInfo.hpp](#).

8.26 FrameInfo.hpp

[Go to the documentation of this file.](#)

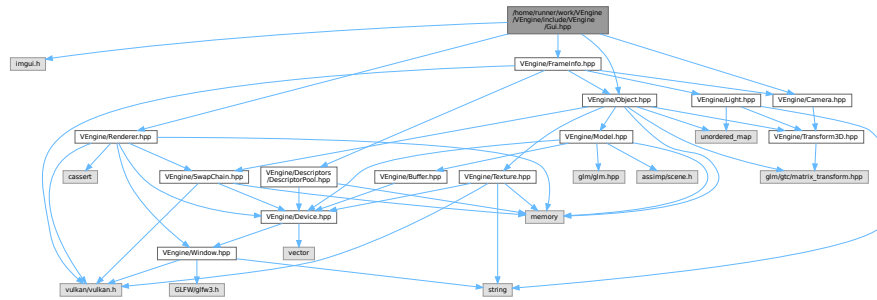
```
00001 ///  
00002 ///  
00003 ///  
00004 ///  
00005 ///  
00006 ///  
00007 #pragma once  
00008 ///  
00009 #include <vulkan/vulkan.h>  
00010 ///  
00011 #include "VEngine/Descriptors/DescriptorPool.hpp"  
00012 #include "VEngine/Camera.hpp"  
00013 #include "VEngine/Object.hpp"  
00014 #include "VEngine/Light.hpp"  
00015 ///  
00016 namespace ven {  
00017 ///  
00018 static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F;  
00019 static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F),  
    DEFAULT_AMBIENT_LIGHT_INTENSITY};  
00020 ///  
00021 struct PointLightData  
00022 {  
00023     glm::vec4 position{};  
00024     glm::vec4 color{};  
00025     float shininess{32.F};  
00026     float padding[3]; // Pad to 32 bytes  
00027 };  
00028 ///  
00029 struct GlobalUbo  
00030 {  
00031     glm::mat4 projection{1.F};  
00032     glm::mat4 view{1.F};  
00033     glm::mat4 inverseView{1.F};  
00034     glm::vec4 ambientLightColor{DEFAULT_AMBIENT_LIGHT_COLOR};  
00035     std::array<PointLightData, MAX_LIGHTS> pointLights;  
00036     uint8_t numLights;  
00037 };  
00038 ///  
00039 struct FrameInfo  
00040 {  
00041     unsigned long frameIndex;  
00042     float frameTime;  
00043     VkCommandBuffer commandBuffer;  
00044     Camera &camera;  
00045     VkDescriptorSet globalDescriptorSet;  
00046     DescriptorPool &frameDescriptorPool;  
00047     Object::Map &objects;  
00048     Light::Map &lights;  
00049 };  
00050 ///  
00051 } // namespace ven
```

8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp File Reference

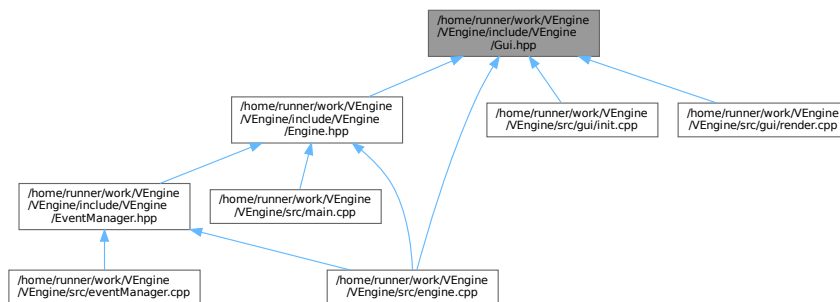
This file contains the ImGuiWindowManager class.

```
#include <imgui.h>  
#include "VEngine/Object.hpp"
```

```
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/FrameInfo.hpp"
Include dependency graph for Gui.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Gui`
Class for `Gui`.
- struct `ven::Gui::funcs`

Namespaces

- namespace **ven**

Enumerations

- enum `ven::GUI_STATE` : `uint8_t` { `ven::VISIBLE` = 0 , `ven::HIDDEN` = 1 }

Variables

- `static constexpr uint16_t ven::DESCRIPTOR_COUNT = 1000`

8.27.1 Detailed Description

This file contains the ImGuiWindowManager class.

Definition in file [Gui.hpp](#).

8.28 Gui.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Gui.hpp
00003 /// @brief This file contains the ImGuiWindowManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <imgui.h>
00010
00011 #include "VEngine/Object.hpp"
00012 #include "VEngine/Renderer.hpp"
00013 #include "VEngine/Camera.hpp"
00014 #include "VEngine/FrameInfo.hpp"
00015
00016 namespace ven {
00017
00018     static constexpr uint16_t DESCRIPTOR_COUNT = 1000;
00019
00020     enum GUI_STATE : uint8_t {
00021         VISIBLE = 0,
00022         HIDDEN = 1
00023     };
00024
00025     ///
00026     /// @class Gui
00027     /// @brief Class for Gui
00028     /// @namespace ven
00029     ///
00030     class Gui {
00031     public:
00032
00033         Gui() = default;
00034         ~Gui() = default;
00035
00036         Gui(const Gui&) = delete;
00037         Gui& operator=(const Gui&) = delete;
00038
00039         static void init(GLFWwindow* window, VkInstance instance, const Device* device,
00040             VkRenderPass renderPass);
00041
00042         static void render(Renderer* renderer, std::unordered_map<unsigned int, Object>& objects,
00043             std::unordered_map<unsigned int, Light>& lights, Camera& camera, VkPhysicalDevice physicalDevice,
00044             GlobalUbo& ubo);
00045         static void cleanup();
00046
00047         void setState(const GUI_STATE state) { m_state = state; }
00048         [[nodiscard]] GUI_STATE getState() const { return m_state; }
00049
00050     private:
00051
00052         static void initStyle();
00053         static void renderFrameWindow();
00054         static void cameraSection(Camera& camera);
00055         static void inputsSection(const ImGuiIO* io);
00056         static void rendererSection(Renderer* renderer, GlobalUbo& ubo);
00057         static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
00058         static void objectsSection(std::unordered_map<unsigned int, Object>& objects);
00059         static void lightsSection(std::unordered_map<unsigned int, Light>& lights);
00060
00061         struct funcs { static bool IsLegacyNativeDupe(const ImGuiKey key) { return key >= 0 && key
00062             < 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist
00063
00064         static ImGuiIO* m_io;
00065         GUI_STATE m_state{VISIBLE};
00066         static float m_intensity;
00067         static float m_shininess;
00068     }; // class Gui
00069 } // namespace ven

```


This file contains the Light class.

```
graph TD; A["/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp"] --> B["unordered_map"]; A --> C["string"]; A --> D["VEngine/Transform3D.hpp"]; D --> E["glm/gtc/matrix_transform.hpp"];
```

```

graph TD
    VulkanHpp[RenderSystem/Vulkan/Vulkan.hpp] --> VulkanBaseHpp[RenderSystem/Vulkan/VulkanBase.hpp]
    VulkanHpp --> VulkanManagerHpp[RenderSystem/Vulkan/VulkanManager.hpp]
    VulkanBaseHpp --> VulkanBaseCpp[RenderSystem/Vulkan/VulkanBase.cpp]
    VulkanBaseHpp --> VulkanRendererHpp[RenderSystem/Vulkan/VulkanRenderer.hpp]
    VulkanBaseHpp --> VulkanRendererCpp[RenderSystem/Vulkan/VulkanRenderer.cpp]
    VulkanBaseHpp --> VulkanRendererBaseHpp[RenderSystem/Vulkan/VulkanRendererBase.hpp]
    VulkanBaseHpp --> VulkanRendererBaseCpp[RenderSystem/Vulkan/VulkanRendererBase.cpp]
    VulkanBaseHpp --> VulkanRendererBaseHpp[RenderSystem/Vulkan/VulkanRendererBase.hpp]
    VulkanBaseHpp --> VulkanRendererBaseCpp[RenderSystem/Vulkan/VulkanRendererBase.cpp]
    VulkanBaseHpp --> VulkanRendererBaseHpp[RenderSystem/Vulkan/VulkanRendererBase.hpp]
    VulkanBaseHpp --> VulkanRendererBaseCpp[RenderSystem/Vulkan/VulkanRendererBase.cpp]
    VulkanManagerHpp --> VulkanManagerHpp[RenderSystem/Vulkan/VulkanManager.hpp]
    VulkanManagerHpp --> VulkanManagerCpp[RenderSystem/Vulkan/VulkanManager.cpp]
    VulkanManagerHpp --> VulkanManagerHpp[RenderSystem/Vulkan/VulkanManager.hpp]
    VulkanManagerHpp --> VulkanManagerCpp[RenderSystem/Vulkan/VulkanManager.cpp]
    VulkanManagerHpp --> VulkanManagerHpp[RenderSystem/Vulkan/VulkanManager.hpp]
    VulkanManagerHpp --> VulkanManagerCpp[RenderSystem/Vulkan/VulkanManager.cpp]
    VulkanManagerHpp --> VulkanManagerHpp[RenderSystem/Vulkan/VulkanManager.hpp]
    VulkanManagerHpp --> VulkanManagerCpp[RenderSystem/Vulkan/VulkanManager.cpp]
  
```

- class `ven::Light`
Class for light.

- namespace **ven**

Variables

- static constexpr float `ven::DEFAULT_LIGHT_INTENSITY` = .2F
- static constexpr float `ven::DEFAULT_LIGHT_RADIUS` = 0.1F
- static constexpr float `ven::DEFAULT_SHININESS` = 32.F
- static constexpr glm::vec4 `ven::DEFAULT_LIGHT_COLOR` = {glm::vec3(1.F), `DEFAULT_LIGHT_INTENSITY`}
- static constexpr uint8_t `ven::MAX_LIGHTS` = 10

8.29.1 Detailed Description

This file contains the Light class.

Definition in file [Light.hpp](#).

8.30 Light.hpp

[Go to the documentation of this file.](#)

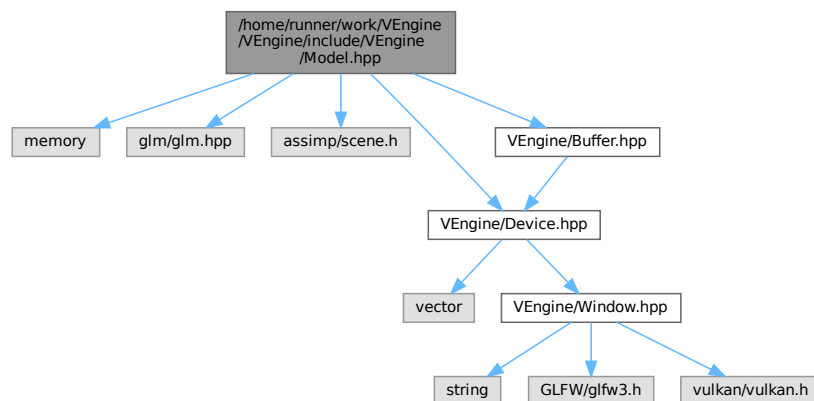
```

00001 ///
00002 /// @file Light.hpp
00003 /// @brief This file contains the Light class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <unordered_map>
00010 #include <string>
00011
00012 #include "VEngine/Transform3D.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
00017     static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
00018     static constexpr float DEFAULT_SHININESS = 32.F;
00019     static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY};
00020
00021     static constexpr uint8_t MAX_LIGHTS = 10;
00022
00023     ///
00024     /// @class Light
00025     /// @brief Class for light
00026     /// @namespace ven
00027     ///
00028     class Light {
00029
00030     public:
00031
00032         using Map = std::unordered_map<unsigned int, Light>;
00033
00034         explicit Light(const unsigned int objId) : m_lightId{objId} {}
00035
00036         ~Light() = default;
00037
00038         Light(const Light&) = delete;
00039         Light& operator=(const Light&) = delete;
00040         Light(Light&&) = default;
00041         Light& operator=(Light&&) = default;
00042
00043         [[nodiscard]] unsigned int getId() const { return m_lightId; }
00044         [[nodiscard]] std::string getName() const { return m_name; }
00045         [[nodiscard]] float getShininess() const { return m_shininess; }
00046
00047         void setName(const std::string &name) { m_name = name; }
00048         void setShininess(const float shininess) { m_shininess = shininess; }
00049
00050         glm::vec4 color{DEFAULT_LIGHT_COLOR};
00051         Transform3D transform{};
00052
00053     private:
00054
00055         unsigned int m_lightId;
00056         std::string m_name{"point light"};
00057         float m_shininess{DEFAULT_SHININESS};
00058
00059     }; // class Light
00060
00061 } // namespace ven

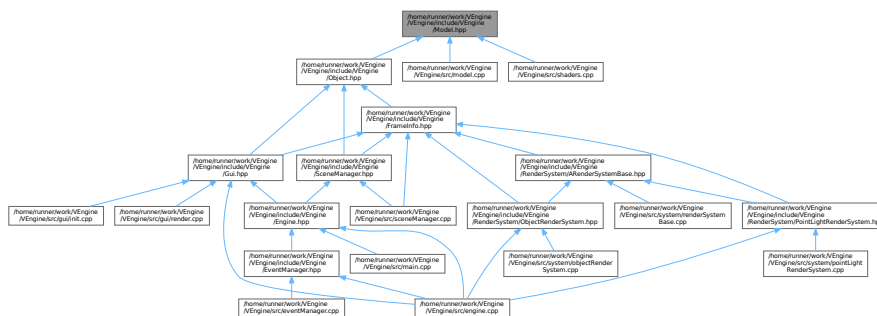
```

This file contains the Model class.

```
#include <memory>
#include <glm/glm.hpp>
#include <assimp/scene.h>
#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"
Include dependency graph for Model.hpp:
```



This graph shows which files directly or indirectly include this file:



- class `ven::Model`
Class for model.
- struct `ven::Model::Vertex`
- struct `ven::Model::Builder`

- namespace **ven**

8.31.1 Detailed Description

This file contains the Model class.

Definition in file [Model.hpp](#).

8.32 Model.hpp

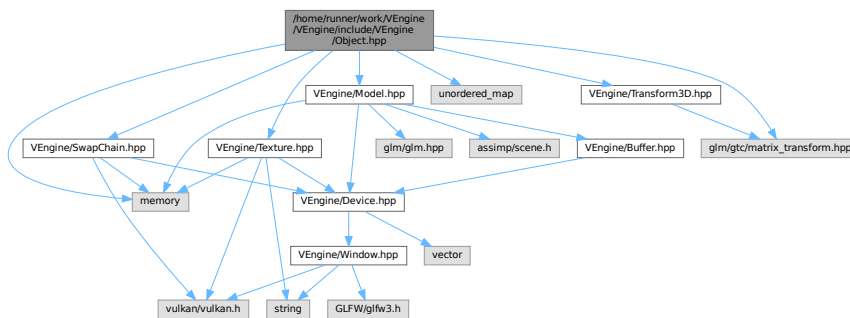
[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include <glm/glm.hpp>
00012
00013 #include <assimp/scene.h>
00014
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/Buffer.hpp"
00017
00018 namespace ven {
00019
00020     ///
00021     /// @class Model
00022     /// @brief Class for model
00023     /// @namespace ven
00024     ///
00025     class Model {
00026
00027     public:
00028
00029         struct Vertex {
00030             glm::vec3 position{};
00031             glm::vec3 color{};
00032             glm::vec3 normal{};
00033             glm::vec2 uv{};
00034
00035             static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00036             static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00037
00038             bool operator==(const Vertex& other) const {
00039                 return position == other.position && color == other.color && normal ==
00040                     other.normal && uv == other.uv;
00041             };
00042
00043             struct Builder {
00044                 std::vector<Vertex> vertices;
00045                 std::vector<uint32_t> indices;
00046
00047                 void loadModel(const std::string &filename);
00048                 void processNode(const aiNode* node, const aiScene* scene);
00049                 void processMesh(const aiMesh* mesh, const aiScene* scene);
00050             };
00051
00052             Model(Device &device, const Builder &builder);
00053             ~Model() = default;
00054
00055             Model(const Model&) = delete;
00056             void operator=(const Model&) = delete;
00057
00058             static std::unique_ptr<Model> createModelFromFile(Device &device, const std::string
00059 &filename);
00060
00061             void bind(VkCommandBuffer commandBuffer) const;
00062             void draw(VkCommandBuffer commandBuffer) const;
00063
00064         private:
00065             void createVertexBuffer(const std::vector<Vertex>& vertices);
00066             void createIndexBuffer(const std::vector<uint32_t>& indices);
00067
00068     };
00069
00070 }
```

8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp

File Reference

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/SwapChain.hpp"
#include "VEngine/Texture.hpp"
#include "VEngine/Model.hpp"
#include "VEngine/Transform3D.hpp"
Include dependency graph for Object.hpp:
```

[illegible]

- struct `ven::ObjectBufferData`
- class `ven::Object`
Class for object.

Namespaces

- namespace [ven](#)

Variables

- static constexpr uint16_t [ven::MAX_OBJECTS](#) = 1000

8.33.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

8.34 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/SwapChain.hpp"  
00015 #include "VEngine/Texture.hpp"  
00016 #include "VEngine/Model.hpp"  
00017 #include "VEngine/Transform3D.hpp"  
00018  
00019 namespace ven {  
00020  
00021     static constexpr uint16_t MAX_OBJECTS = 1000;  
00022  
00023     struct ObjectBufferData {  
00024         glm::mat4 modelMatrix{1.F};  
00025         glm::mat4 normalMatrix{1.F};  
00026     };  
00027  
00028     ///  
00029     ///  
00030     ///  
00031     ///  
00032     ///  
00033     class Object {  
00034  
00035     public:  
00036  
00037         using Map = std::unordered_map<unsigned int, Object>;  
00038  
00039         explicit Object(const unsigned int objId) : m_objId(objId) {}  
00040         Object(const Object &) = delete;  
00041         Object(Object &&) = default;  
00042         Object &operator=(const Object &) = delete;  
00043         Object &operator=(Object &&) = delete;  
00044  
00045         [[nodiscard]] unsigned int getId() const { return m_objId; }  
00046         [[nodiscard]] std::string getName() const { return m_name; }  
00047         [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }  
00048         [[nodiscard]] std::shared_ptr<Texture> getDiffuseMap() const { return m_diffuseMap; }  
00049         [[nodiscard]] VkDescriptorBufferInfo getBufferInfo(const int frameIndex) const { return  
m_bufferInfo.at(frameIndex); }  
00050         void setModel(const std::shared_ptr<Model> &model) { m_model = model; }  
00051         void setDiffuseMap(const std::shared_ptr<Texture> &diffuseMap) { m_diffuseMap =  
diffuseMap; }  
00052         void setName(const std::string &name) { m_name = name; }
```

```

00053         void setBufferInfo(const int frameIndex, const VkDescriptorBufferInfo& info) {
00054             m_bufferInfo[frameIndex] = info;
00055         }
00056
00057         Transform3D transform{};
00058
00059     private:
00060
00061         unsigned int m_objId;
00062         std::string m_name;
00063         std::shared_ptr<Model> m_model = nullptr;
00064         std::shared_ptr<Texture> m_diffuseMap = nullptr;
00065         std::unordered_map<int, VkDescriptorBufferInfo> m_bufferInfo;
00066
00067     }; // class Object
00068
00069 } // namespace ven

```

8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

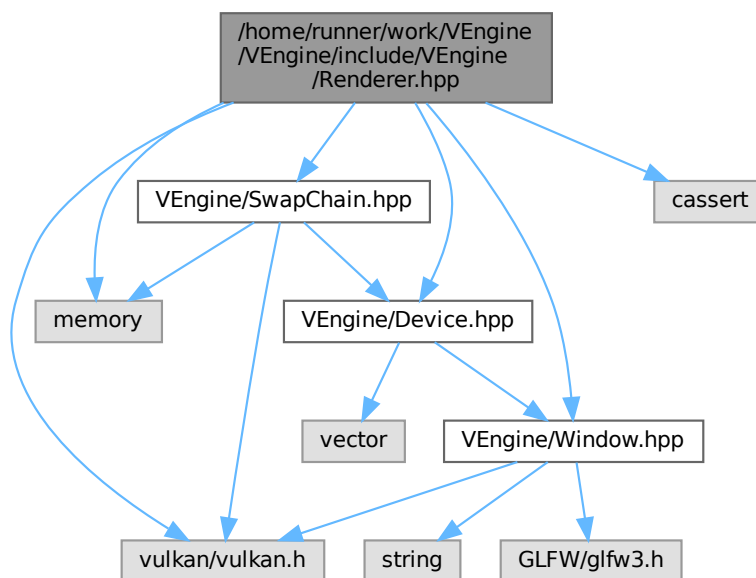
This file contains the Renderer class.

```

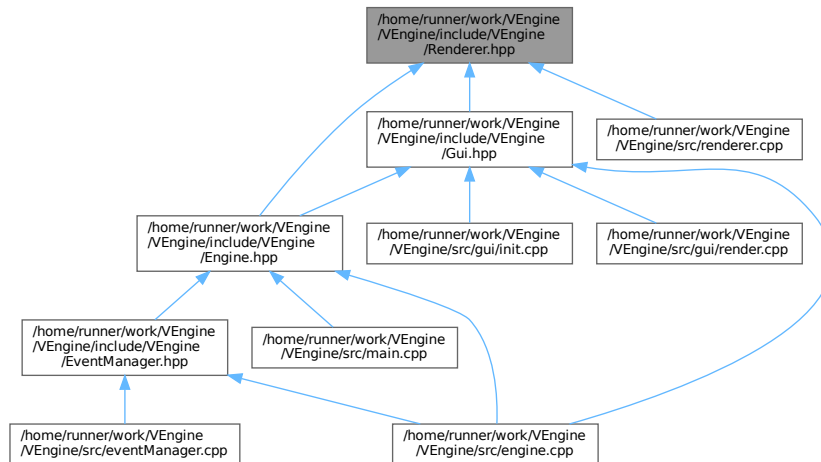
#include <memory>
#include <cassert>
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/SwapChain.hpp"

```

Include dependency graph for Renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Renderer](#)
Class for renderer.

Namespaces

- namespace [ven](#)

Variables

- static constexpr VkClearColorValue [ven::DEFAULT_CLEAR_COLOR](#) = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue [ven::DEFAULT_CLEAR_DEPTH](#) = {1.0F, 0}

8.35.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

8.36 Renderer.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Renderer.hpp
00003 /// @brief This file contains the Renderer class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <cassert>
00011
00012 #include <vulkan/vulkan.h>
00013
00014 #include "VEngine/Window.hpp"
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/SwapChain.hpp"
00017
00018 namespace ven {
00019
00020     static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}};
00021     static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0};
00022
00023     ///
00024     /// @class Renderer
00025     /// @brief Class for renderer
00026     /// @namespace ven
00027     ///
00028     class Renderer {
00029
00030     public:
00031
00032         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
00033             recreateSwapChain(); createCommandBuffers(); }
00034         ~Renderer() { freeCommandBuffers(); }
00035
00036         Renderer(const Renderer &) = delete;
00037         Renderer& operator=(const Renderer &) = delete;
00038
00039         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
00040             m_swapChain->getRenderPass(); }
00041         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00042         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00043         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress())
00044             && "cannot get command m_buffer when frame not in progress"; return
00045             m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00046
00047         [[nodiscard]] unsigned long getFrameIndex() const { assert(isFrameInProgress()) && "cannot
00048             get frame index when frame not in progress"; return m_currentFrameIndex; }
00049         [[nodiscard]] std::array<float, 4> getClearColor() const { return {
00050             m_clearValues[0].color.float32[0],
00051             m_clearValues[0].color.float32[1],
00052             m_clearValues[0].color.float32[2],
00053             m_clearValues[0].color.float32[3]
00054         }; }
00055
00056         [[nodiscard]] Window& getWindow() const { return m_window; }
00057
00058         void setClearColor(const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR, const
00059             VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
00060             clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00061         void beginFrame();
00062         void endFrame();
00063         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00064         void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00065
00066     private:
00067
00068         void createCommandBuffers();
00069         void freeCommandBuffers();
00070         void recreateSwapChain();
00071
00072         Window &m_window;
00073         Device &m_device;
00074         std::unique_ptr<SwapChain> m_swapChain;
00075         std::vector<VkCommandBuffer> m_commandBuffers;
00076         std::array<VkClearValue, 2> m_clearValues{DEFAULT_CLEAR_COLOR, 1.0F, 0.0F};
00077
00078         uint32_t m_currentImageIndex{0};
00079         unsigned long m_currentFrameIndex{0};
00080         bool m_isFrameStarted{false};
00081
00082     }; // class Renderer

```


8.37.1 Detailed Description

This file contains the ARenderSystemBase class.

Definition in file [ARenderSystemBase.hpp](#).

8.38 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/Descriptors/DescriptorSetLayout.hpp"
00012 #include "VEngine/Device.hpp"
00013 #include "VEngine/Shaders.hpp"
00014 #include "VEngine/FrameInfo.hpp"
00015
00016 namespace ven {
00017
00018     ///
00019     /// @class ARenderSystemBase
00020     /// @brief Abstract class for render system base
00021     /// @namespace ven
00022     ///
00023     class ARenderSystemBase {
00024
00025     public:
00026
00027         explicit ARenderSystemBase(Device& device) : m_device(device) {}
00028         virtual ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(),
m_pipelineLayout, nullptr); }
00029
00030         virtual void render(const FrameInfo &frameInfo) const = 0;
00031
00032     protected:
00033
00034         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout, uint32_t
pushConstantSize);
00035         void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
std::string &shadersFragPath, bool isLight);
00036
00037         [[nodiscard]] Device& getDevice() const { return m_device; }
00038         [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
00039         [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }
00040
00041         std::unique_ptr<DescriptorSetLayout> renderSystemLayout;
00042
00043     private:
00044
00045         Device &m_device;
00046         VkPipelineLayout m_pipelineLayout{nullptr};
00047         std::unique_ptr<Shaders> m_shaders;
00048
00049     }; // class ARenderSystemBase
00050
00051 } // namespace ven
00052 }
```

8.39 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp File Reference

This file contains the ObjectRenderSystem class.

8.40 ObjectRenderSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file ObjectRenderSystem.hpp
00003 /// @brief This file contains the ObjectRenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/FrameInfo.hpp"
00010 #include "VEngine/RenderSystem/ARenderSystemBase.hpp"
00011
00012 namespace ven {
00013
00014     struct ObjectPushConstantData {
00015         glm::mat4 modelMatrix{};
00016         glm::mat4 normalMatrix{};
00017     };
00018
00019     ///
00020     /// @class ObjectRenderSystem
00021     /// @brief Class for object render system
00022     /// @namespace ven
00023     ///
00024     class ObjectRenderSystem final : public ARenderSystemBase {
00025
00026     public:
00027
00028         explicit ObjectRenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00029             createPipelineLayout(globalSetLayout, sizeof(ObjectPushConstantData));
00030             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_shader.spv",
std::string(SHADERS_BIN_PATH) + "fragment_shader.spv", false);
00031         }
00032
00033         ObjectRenderSystem(const ObjectRenderSystem&) = delete;
00034         ObjectRenderSystem& operator=(const ObjectRenderSystem&) = delete;
00035
00036         void render(const FrameInfo &frameInfo) const override;
00037
00038     }; // class ObjectRenderSystem
00039
00040 } // namespace ven

```

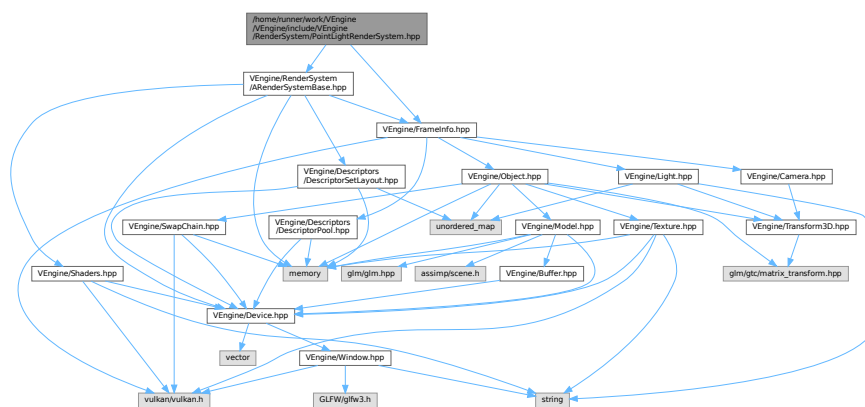
8.41 [/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp](#) File Reference

This file contains the `PointLightRenderSystem` class.

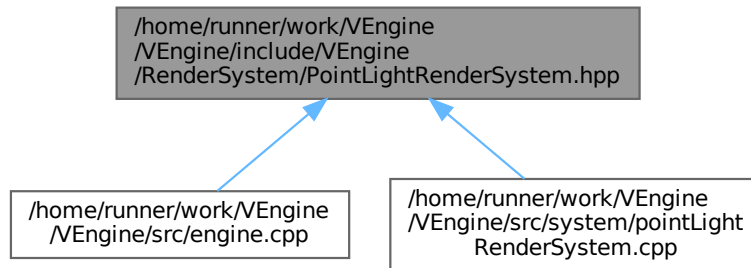
```
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
```

```
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for PointLightRenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::LightPushConstantData](#)
- class [ven::PointLightRenderSystem](#)

Class for point light system.

Namespaces

- namespace [ven](#)

8.41.1 Detailed Description

This file contains the PointLightRenderSystem class.

Definition in file [PointLightRenderSystem.hpp](#).

8.42 PointLightRenderSystem.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file PointLightRenderSystem.hpp
00003 /// @brief This file contains the PointLightRenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/RenderSystem/ARenderSystemBase.hpp"
00010 #include "VEngine/FrameInfo.hpp"
00011
00012 namespace ven {
00013
00014     struct LightPushConstantData {
00015         glm::vec4 position{};
00016         glm::vec4 color{};
00017         float radius;
00018     };
00019
00020     ///
00021     /// @class PointLightRenderSystem
00022     /// @brief Class for point light system
  
```

```

00023     /// @namespace ven
00024     ///
00025     class PointLightRenderSystem final : public ARenderSystemBase {
00026     public:
00027
00028
00029         explicit PointLightRenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
00030             createPipelineLayout(globalSetLayout, sizeof(LightPushConstantData));
00031             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_point_light.spv",
std::string(SHADERS_BIN_PATH) + "fragment_point_light.spv", true);
00032         }
00033
00034         PointLightRenderSystem(const PointLightRenderSystem&) = delete;
00035         PointLightRenderSystem& operator=(const PointLightRenderSystem&) = delete;
00036
00037         void render(const FrameInfo &frameInfo) const override;
00038
00039     }; // class PointLightRenderSystem
00040
00041 } // namespace ven

```

8.43 /home/runner/work/VEngine/VEngine/include/VEngine/SceneManager.hpp File Reference

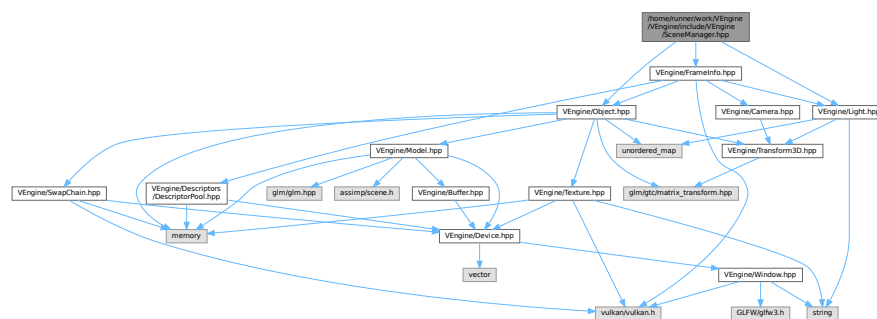
This file contains the SceneManager class.

```

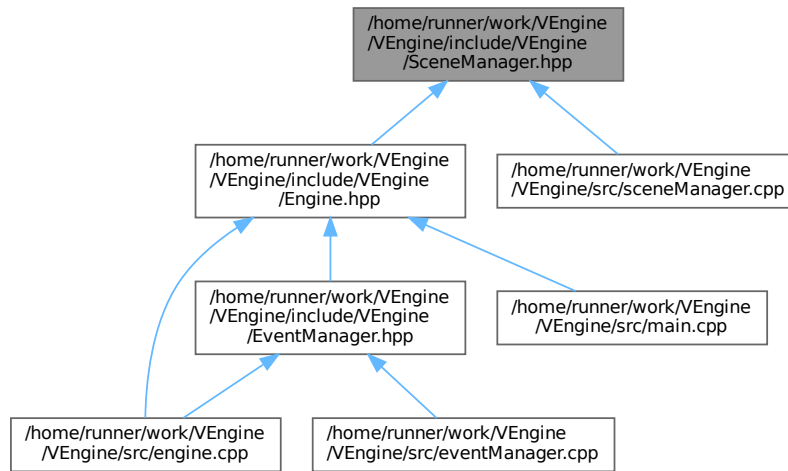
#include "VEngine/FrameInfo.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Light.hpp"

```

Include dependency graph for SceneManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::SceneManager](#)
Class for object manager.

Namespaces

- namespace [ven](#)

8.43.1 Detailed Description

This file contains the SceneManager class.

Definition in file [SceneManager.hpp](#).

8.44 SceneManager.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file SceneManager.hpp
00003 /// @brief This file contains the SceneManager class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/FrameInfo.hpp"
00010 #include "VEngine/Object.hpp"
00011 #include "VEngine/Light.hpp"
00012
00013 namespace ven {
00014
00015     ///
```



```

00016     /// @class SceneManager
00017     /// @brief Class for object manager
00018     /// @namespace ven
00019     ///
00020     class SceneManager {
00021     public:
00022
00023         explicit SceneManager(Device &device);
00024
00025         SceneManager(const SceneManager &) = delete;
00026         SceneManager &operator=(const SceneManager &) = delete;
00027         SceneManager(SceneManager &&) = delete;
00028         SceneManager &operator=(SceneManager &&) = delete;
00029
00030         Object& createObject();
00031         Light& createLight(float radius = DEFAULT_LIGHT_RADIUS, glm::vec4 color =
00032         DEFAULT_LIGHT_COLOR);
00033
00034         VkDescriptorBufferInfo getBufferInfoForObject(const int frameIndex, const unsigned int
00035         objectId) const { return m_uboBuffers.at(static_cast<long unsigned
00036         int>(frameIndex))->descriptorInfoForIndex(objectId); }
00037         Object::Map& getObjects() { return m_objects; }
00038         Light::Map& getLights() { return m_lights; }
00039         std::vector<std::unique_ptr<Buffer>> &getUboBuffers() { return m_uboBuffers; }
00040
00041         void updateBuffer(GlobalUbo &ubo, unsigned long frameIndex, float frameTime);
00042
00043     private:
00044         unsigned int m_currentObjId{0};
00045         unsigned int m_currentLightId{0};
00046         std::shared_ptr<Texture> m_textureDefault;
00047         Object::Map m_objects;
00048         Light::Map m_lights;
00049         std::vector<std::unique_ptr<Buffer>> m_uboBuffers{MAX_FRAMES_IN_FLIGHT};
00050     }; // class SceneManager
00051
00052 } // namespace ven

```

8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Shader.h File Reference

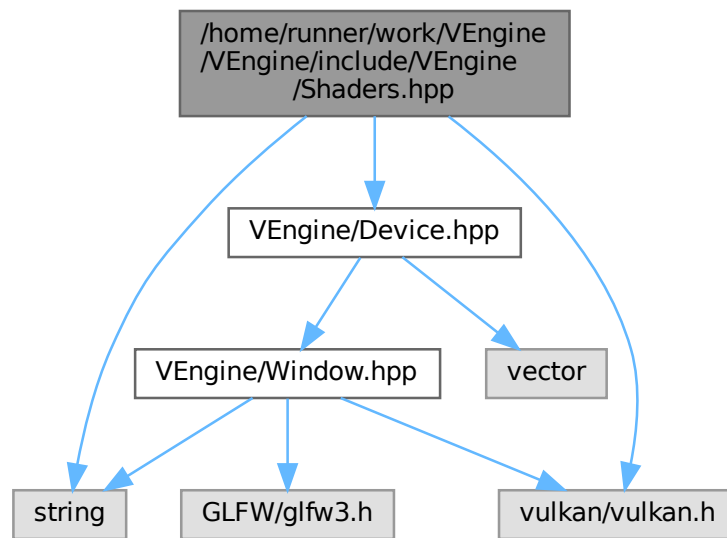
This file contains the Shader class.

```

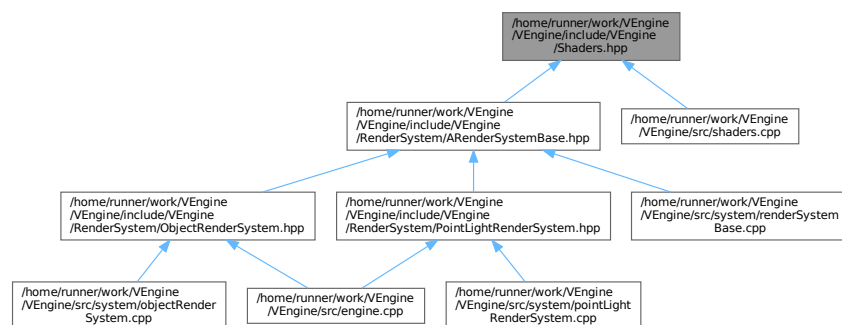
#include <string>
#include <vulkan/vulkan.h>
#include "VEngine/Device.h"

```

Include dependency graph for Shaders.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::PipelineConfigInfo](#)
- class [ven::Shaders](#)

Class for shaders.

Namespaces

- namespace [ven](#)

Variables

- static constexpr std::string_view `ven::SHADERS_BIN_PATH` = "build/shaders/"

8.45.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

8.46 Shaders.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Shaders.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 #include "VEngine/Device.hpp"
00014
00015 namespace ven {
00016
00017     static constexpr std::string_view SHADERS_BIN_PATH = "build/shaders/";
00018
00019     struct PipelineConfigInfo {
00020         PipelineConfigInfo() = default;
00021         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00022         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00023
00024         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00025         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00026         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00027         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00028         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00029         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00030         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00031         VkPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00032         std::vector<VkDynamicState> dynamicStateEnables;
00033         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00034         VkPipelineLayout pipelineLayout = nullptr;
00035         VkRenderPass renderPass = nullptr;
00036         uint32_t subpass = 0;
00037     };
00038
00039     ///
00040     /// @class Shaders
00041     /// @brief Class for shaders
00042     /// @namespace ven
00043     ///
00044     class Shaders {
00045     public:
00046
00047         Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
00048             const PipelineConfigInfo& configInfo) : m_device(device) { createGraphicsPipeline(vertFilepath,
00049                 fragFilepath, configInfo); };
00050         ~Shaders();
00051
00052         Shaders(const Shaders&) = delete;
00053         Shaders& operator=(const Shaders&) = delete;
00054
00055         static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00056         void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
00057             VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00058     private:

```

```

00059         static std::vector<char> readFile(const std::string &filename);
00060         void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo);
00061         void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
const;
00062
00063         Device& m_device;
00064         VkPipeline m_graphicsPipeline{nullptr};
00065         VkShaderModule m_vertShaderModule{nullptr};
00066         VkShaderModule m_fragShaderModule{nullptr};
00067
00068     }; // class Shaders
00069
00070 } // namespace ven

```

8.47 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

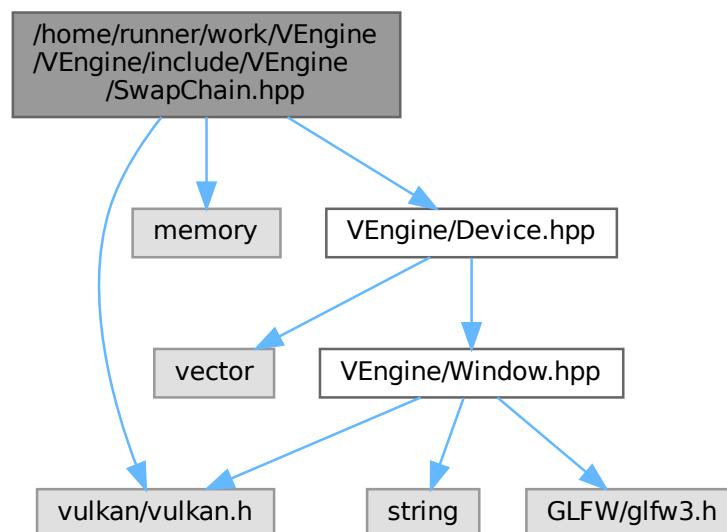
This file contains the Shader class.

```

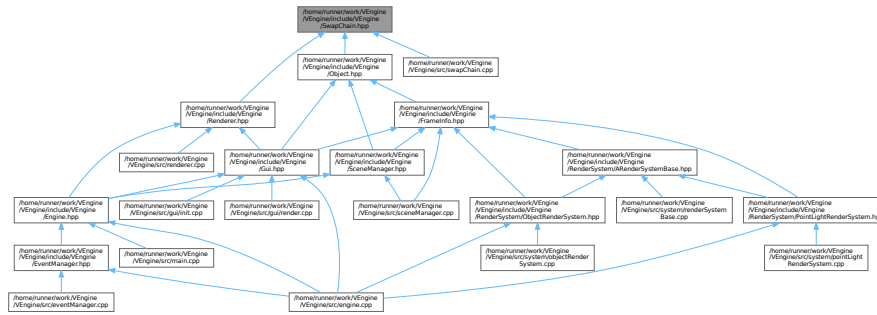
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"

```

Include dependency graph for SwapChain.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::SwapChain`
Class for swap chain.

Namespaces

- namespace **ven**

Variables

- static constexpr int **ven::MAX_FRAMES_IN_FLIGHT** = 2

8.47.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

8.48 SwapChain.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file SwapChain.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010 #include <memory>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00017
00018     ///
00019     /// @class SwapChain
```

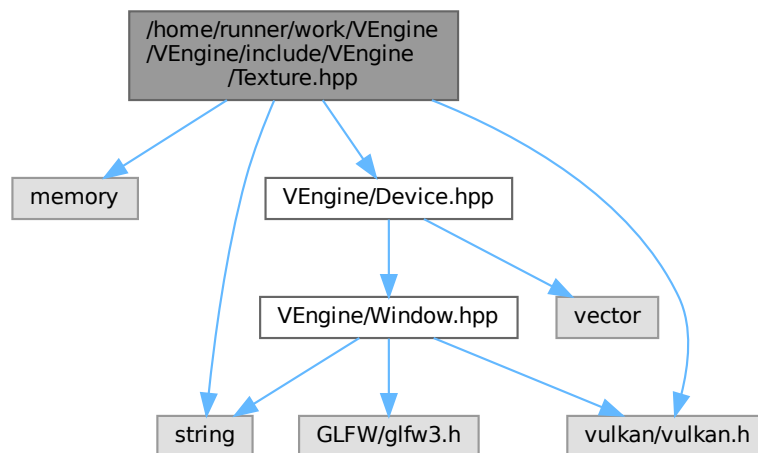
```

00020     /// @brief Class for swap chain
00021     /// @namespace ven
00022     ///
00023     class SwapChain {
00024     public:
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
00034         [[nodiscard]] VkFramebuffer getFramebuffer(const unsigned long index) const { return
00035         m_swapChainFrameBuffers[index]; }
00036         [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }
00037         [[nodiscard]] VkImageView getImageView(const int index) const { return
00038         m_swapChainImageViews[static_cast<unsigned long>(index)]; }
00039         [[nodiscard]] size_t imageCount() const { return m_swapChainImages.size(); }
00040         [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_swapChainImageFormat; }
00041         [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00042         [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
00043         [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00044         [[nodiscard]] float extentAspectRatio() const { return
00045         static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
00046         [[nodiscard]] VkFormat findDepthFormat() const;
00047         VkResult acquireNextImage(uint32_t *imageIndex) const;
00048         VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00049         [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return
00050         m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat ==
00051         swapChain.m_swapChainDepthFormat; }
00052     private:
00053         void init();
00054         void createSwapChain();
00055         void createImageViews();
00056         void createDepthResources();
00057         void createRenderPass();
00058         void createFrameBuffers();
00059         void createSyncObjects();
00060
00061         static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
00062         &availableFormats);
00063         static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
00064         &availablePresentModes);
00065         [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
00066         const;
00067         VkFormat m_swapChainImageFormat{};
00068         VkFormat m_swapChainDepthFormat{};
00069         VkExtent2D m_swapChainExtent{};
00070         std::vector<VkFramebuffer> m_swapChainFrameBuffers;
00071         VkRenderPass m_renderPass{};
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         Device &m_device;
00078         VkExtent2D m_windowExtent;
00079         VkSwapchainKHR m_swapChain{};
00080         std::shared_ptr<SwapChain> m_oldSwapChain;
00081         std::vector<VkSemaphore> m_imageAvailableSemaphores;
00082         std::vector<VkSemaphore> m_renderFinishedSemaphores;
00083         std::vector<VkFence> m_inFlightFences;
00084         std::vector<VkFence> m_imagesInFlight;
00085         size_t m_currentFrame{0};
00086     }; // class SwapChain
00087 } // namespace ven

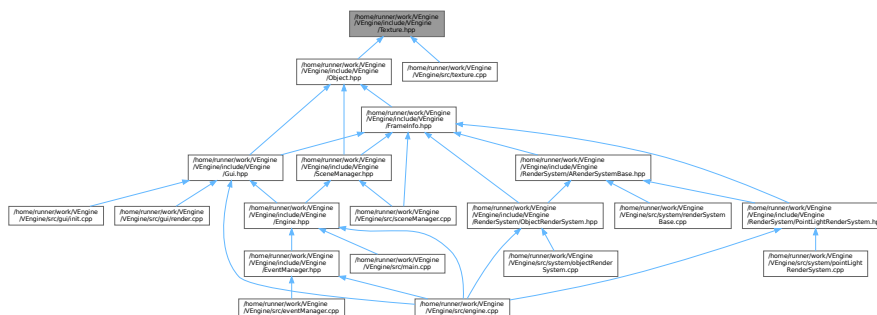
```

This file contains the Texture class.

```
#include <memory>
#include <string>
#include <vulkan/vulkan.h>
#include "VEngine/Device.hpp"
Include dependency graph for Texture.hpp:
```



This graph shows which files directly or indirectly include this file:



- class `ven::Texture`
Class for texture.

- namespace **ven**

8.49.1 Detailed Description

This file contains the Texture class.

Definition in file [Texture.hpp](#).

8.50 Texture.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Texture.hpp
00003 /// @brief This file contains the Texture class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <string>
00011
00012 #include<vulkan/vulkan.h>
00013
00014 #include "VEngine/Device.hpp"
00015
00016 namespace ven {
00017
00018     ///
00019     /// @class Texture
00020     /// @brief Class for texture
00021     /// @namespace ven
00022     ///
00023     class Texture {
00024
00025     public:
00026
00027         Texture(Device &device, const std::string &textureFilepath);
00028         Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00029             VkSampleCountFlagBits sampleCount);
00030         ~Texture();
00031
00032         Texture(const Texture &) = delete;
00033         Texture &operator=(const Texture &) = delete;
00034
00035         static std::unique_ptr<Texture> createTextureFromFile(Device &device, const std::string
00036             &filepath) { return std::make_unique<Texture>(device, filepath); }
00037
00038         void updateDescriptor();
00039         void transitionLayout(VkCommandBuffer commandBuffer, VkImageLayout oldLayout,
00040             VkImageLayout newLayout) const;
00041
00042         [[nodiscard]] VkImageView imageView() const { return m_textureImageView; }
00043         [[nodiscard]] VkSampler sampler() const { return m_textureSampler; }
00044         [[nodiscard]] VkImage getImage() const { return m_textureImage; }
00045         [[nodiscard]] VkImageView getImageView() const { return m_textureImageView; }
00046         [[nodiscard]] VkDescriptorImageInfo getImageInfo() const { return m_descriptor; }
00047         [[nodiscard]] VkImageLayout getImageLayout() const { return m_textureLayout; }
00048         [[nodiscard]] VkExtent3D getExtent() const { return m_extent; }
00049         [[nodiscard]] VkFormat getFormat() const { return m_format; }
00050
00051     private:
00052
00053         void createTextureImage(const std::string &filepath);
00054         void createTextureImageView(VkImageViewType viewType);
00055         void createTextureSampler();
00056
00057         VkDescriptorImageInfo m_descriptor{};
00058         Device &m_device;
00059         VkImage m_textureImage = nullptr;
00060         VkDeviceMemory m_textureImageMemory = nullptr;
00061         VkImageView m_textureImageView = nullptr;
00062         VkSampler m_textureSampler = nullptr;
00063         VkFormat m_format;
00064         VkImageLayout m_textureLayout{};
00065         uint32_t m_mipLevels{1};
00066         uint32_t m_layerCount{1};
00067         VkExtent3D m_extent{};
00068
00069     }; // class Texture
00070
00071 } // namespace ven

```


8.52 Transform3D.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Transform3D.hpp
00003 /// @brief This file contains the Transform3D class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <glm/gtc/matrix_transform.hpp>
00010
00011 namespace ven {
00012
00013     ///
00014     /// @class Transform3D
00015     /// @brief Class for 3D transformation
00016     /// @namespace ven
00017     ///
00018     class Transform3D {
00019     public:
00020
00021         [[nodiscard]] glm::mat4 transformMatrix() const {
00022             auto rotationMatrix = glm::mat4(1.0F);
00023
00024             rotationMatrix = rotate(rotationMatrix, rotation.x, glm::vec3(1.0F, 0.0F, 0.0F));
00025             rotationMatrix = rotate(rotationMatrix, rotation.y, glm::vec3(0.0F, 1.0F, 0.0F));
00026             rotationMatrix = rotate(rotationMatrix, rotation.z, glm::vec3(0.0F, 0.0F, 1.0F));
00027
00028             const glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0F), scale);
00029             const glm::mat4 translationMatrix = translate(glm::mat4(1.0F), translation);
00030
00031             return translationMatrix * rotationMatrix * scaleMatrix;
00032         }
00033
00034         [[nodiscard]] glm::mat3 normalMatrix() const { return
transpose(inverse(glm::mat3(transformMatrix()))); }
00035
00036         glm::vec3 translation{};
00037         glm::vec3 scale{1.F, 1.F, 1.F};
00038         glm::vec3 rotation{};
00039
00040     }; // class Transform3D
00041
00042 } // namespace ven

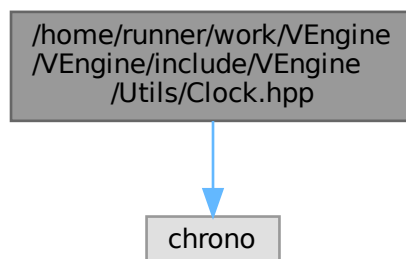
```

8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp File Reference

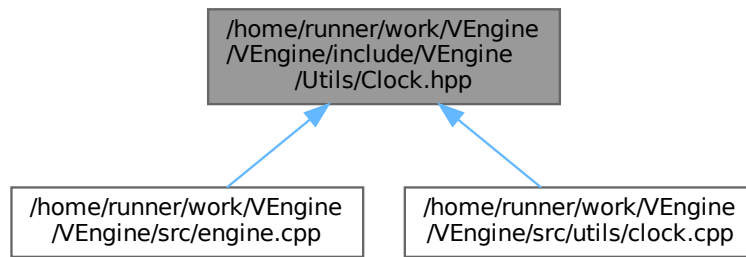
This file contains the Clock class.

```
#include <chrono>
```

Include dependency graph for Clock.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Clock](#)
Class for clock.

Namespaces

- namespace [ven](#)

Typedefs

- using [ven::TimePoint](#) = `std::chrono::time_point<std::chrono::high_resolution_clock>`

8.53.1 Detailed Description

This file contains the Clock class.

Definition in file [Clock.hpp](#).

8.54 Clock.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Clock.hpp
00003 /// @brief This file contains the Clock class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <chrono>
00010
00011 namespace ven {
00012
00013     using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;
00014
00015     ///
00016     /// @class Clock
  
```

```

00017     /// @brief Class for clock
00018     /// @namespace ven
00019     ///
00020     class Clock {
00021     public:
00022         Clock() { start(); }
00023         ~Clock() = default;
00024         Clock(const Clock&) = delete;
00025         Clock& operator=(const Clock&) = delete;
00026
00027         void start() { m_startTime = std::chrono::high_resolution_clock::now(); }
00028         void stop();
00029         void resume();
00030         void update();
00031
00032         [[nodiscard]] float getDeltaTime() const { return m_deltaTime.count(); }
00033
00034     private:
00035         TimePoint m_startTime;
00036         TimePoint m_stopTime;
00037         std::chrono::duration<float> m_deltaTime{0.F};
00038         bool m_isStopped{false};
00039     }; // class Clock
00040
00041 } // namespace ven

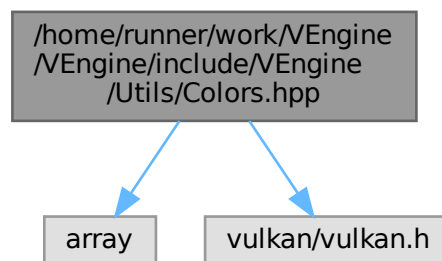
```

8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp File Reference

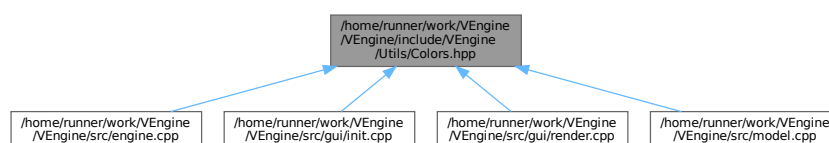
```
#include <array>
```

```
#include <vulkan/vulkan.h>
```

Include dependency graph for Colors.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Colors](#)
Class for colors.

Namespaces

- namespace [ven](#)

Variables

- static constexpr float [ven::COLOR_MAX](#) = 255.0F

8.56 Colors.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file Colors.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <array>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 namespace ven {
00014
00015     static constexpr float COLOR_MAX = 255.0F;
00016
00017     ///
00018     /// @class Colors
00019     /// @brief Class for colors
00020     /// @namespace ven
00021     ///
00022     class Colors {
00023
00024     public:
00025
00026         static constexpr glm::vec3 WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX;
00027         static constexpr glm::vec4 WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F };
00028         static constexpr VkClearColorValue WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } };
00029
00030         static constexpr glm::vec3 BLACK_3 = glm::vec3(0.0F);
00031         static constexpr glm::vec4 BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F };
00032         static constexpr VkClearColorValue BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } };
00033
00034         static constexpr glm::vec3 RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
00035         static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F };
00036         static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } };
00037
00038         static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00039         static constexpr glm::vec4 GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00040         static constexpr VkClearColorValue GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };
00041
00042         static constexpr glm::vec3 BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
00043         static constexpr glm::vec4 BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F };
00044         static constexpr VkClearColorValue BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } };
00045
00046         static constexpr glm::vec3 YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
00047         static constexpr glm::vec4 YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F };
00048         static constexpr VkClearColorValue YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } };
00049
00050         static constexpr glm::vec3 CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00051         static constexpr glm::vec4 CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00052         static constexpr VkClearColorValue CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };
00053
00054         static constexpr glm::vec3 MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00055         static constexpr glm::vec4 MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00056         static constexpr VkClearColorValue MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };

```

```

00057
00058 static constexpr glm::vec3 SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
00059 static constexpr glm::vec4 SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F };
00060 static constexpr VkClearColorValue SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } };
00061
00062 static constexpr glm::vec3 GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
00063 static constexpr glm::vec4 GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F };
00064 static constexpr VkClearColorValue GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } };
00065
00066 static constexpr glm::vec3 MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
00067 static constexpr glm::vec4 MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F };
00068 static constexpr VkClearColorValue MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } };
00069
00070 static constexpr glm::vec3 OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
00071 static constexpr glm::vec4 OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F };
00072 static constexpr VkClearColorValue OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } };
00073
00074 static constexpr glm::vec3 LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00075 static constexpr glm::vec4 LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
00076 static constexpr VkClearColorValue LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };
00077
00078 static constexpr glm::vec3 AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
00079 static constexpr glm::vec4 AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
00080 static constexpr VkClearColorValue AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };
00081
00082 static constexpr glm::vec3 TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
00083 static constexpr glm::vec4 TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F };
00084 static constexpr VkClearColorValue TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } };
00085
00086 static constexpr glm::vec3 NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
00087 static constexpr glm::vec4 NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F };
00088 static constexpr VkClearColorValue NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } };
00089
00090 static constexpr glm::vec3 FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
00091 static constexpr glm::vec4 FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
00092 static constexpr VkClearColorValue FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };
00093
00094 static constexpr glm::vec3 NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
00095 static constexpr glm::vec4 NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F };
00096 static constexpr VkClearColorValue NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } };
00097
00098 static constexpr glm::vec3 SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
00099 static constexpr glm::vec4 SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F };
00100 static constexpr VkClearColorValue SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } };
00101
00102 static constexpr glm::vec3 SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
00103 static constexpr glm::vec4 SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F };
00104 static constexpr VkClearColorValue SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } };
00105
00106
00107 static constexpr std::array<std::pair<const char *, glm::vec3>, 20> COLOR_PRESETS_3 = {{
00108     {"White", WHITE_3},
00109     {"Black", BLACK_3},
00110     {"Red", RED_3},
00111     {"Green", GREEN_3},
00112     {"Blue", BLUE_3},
00113     {"Yellow", YELLOW_3},
00114     {"Cyan", CYAN_3},
00115     {"Magenta", MAGENTA_3},
00116     {"Silver", SILVER_3},
00117     {"Gray", GRAY_3},
00118     {"Maroon", MAROON_3},
00119     {"Olive", OLIVE_3},
00120     {"Lime", LIME_3},
00121     {"Aqua", AQUA_3},
00122     {"Teal", TEAL_3},
00123     {"Navy", NAVY_3},
00124     {"Fuchsia", FUCHSIA_3},
00125     {"Night Blue", NIGHT_BLUE_3},
00126     {"Sky Blue", SKY_BLUE_3},
00127     {"Sunset", SUNSET_3}
00128 }};
00129
00130 static constexpr std::array<std::pair<const char *, glm::vec4>, 20> COLOR_PRESETS_4 = {{
00131     {"White", WHITE_4},
00132     {"Black", BLACK_4},
00133     {"Red", RED_4},
00134     {"Green", GREEN_4},
00135     {"Blue", BLUE_4},
00136     {"Yellow", YELLOW_4},
00137     {"Cyan", CYAN_4},
00138     {"Magenta", MAGENTA_4},
00139     {"Silver", SILVER_4},
00140     {"Gray", GRAY_4},
00141     {"Maroon", MAROON_4},
00142     {"Olive", OLIVE_4},
00143     {"Lime", LIME_4},

```

```

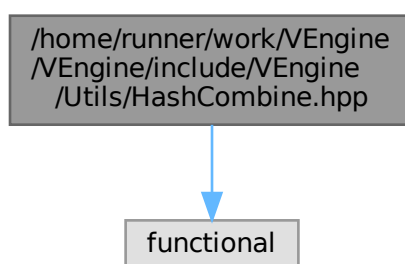
00144         {"Aqua",  AQUA_4},
00145         {"Teal",  TEAL_4},
00146         {"Navy",  NAVY_4},
00147         {"Fuchsia", FUCHSIA_4},
00148         {"Night Blue", NIGHT_BLUE_4},
00149         {"Sky Blue", SKY_BLUE_4},
00150         {"Sunset", SUNSET_4}
00151     };
00152
00153     static constexpr std::array<std::pair<const char *, VkClearColorValue>, 20>
COLOR_PRESETS_VK = {{
00154         {"White",  WHITE_V},
00155         {"Black",  BLACK_V},
00156         {"Red",    RED_V},
00157         {"Green",  GREEN_V},
00158         {"Blue",   BLUE_V},
00159         {"Yellow", YELLOW_V},
00160         {"Cyan",   CYAN_V},
00161         {"Magenta", MAGENTA_V},
00162         {"Silver", SILVER_V},
00163         {"Gray",   GRAY_V},
00164         {"Maroon", MAROON_V},
00165         {"Olive",  OLIVE_V},
00166         {"Lime",   LIME_V},
00167         {"Aqua",   AQUA_V},
00168         {"Teal",   TEAL_V},
00169         {"Navy",   NAVY_V},
00170         {"Fuchsia", FUCHSIA_V},
00171         {"Night Blue", NIGHT_BLUE_V},
00172         {"Sky Blue", SKY_BLUE_V},
00173         {"Sunset", SUNSET_V}
00174     }};
00175
00176     }; // class Colors
00177
00178 } // namespace ven

```

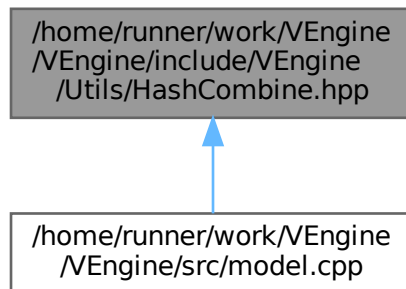
8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp File Reference

#include <functional>

Include dependency graph for HashCombine.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)`

8.58 HashCombine.hpp

[Go to the documentation of this file.](#)

```

00001 ///
00002 /// @file HashCombine.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
  
```

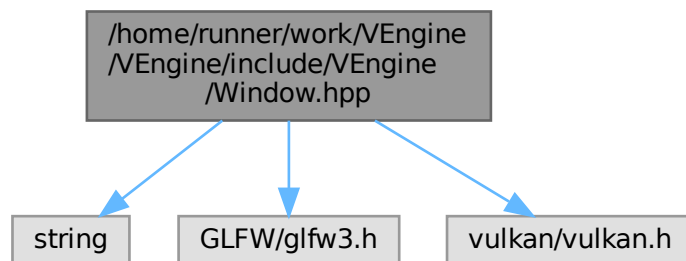
8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp

File Reference

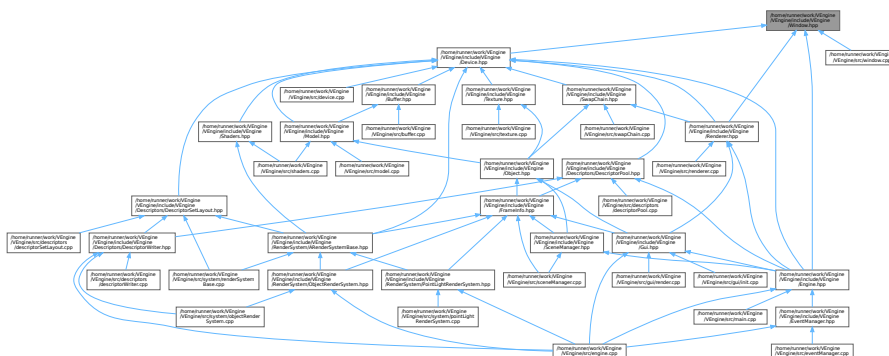
This file contains the Window class.


```
#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"

8.59.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

8.59.2 Macro Definition Documentation

8.59.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

8.60 Window.hpp

[Go to the documentation of this file.](#)

```
00001 ///
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00018     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00019     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00020
00021     ///
00022     /// @class Window
00023     /// @brief Class for window
00024     /// @namespace ven
00025     ///
00026     class Window {
00027     public:
00028
00029         explicit Window(const uint32_t width = DEFAULT_WIDTH, const uint32_t height =
00030             DEFAULT_HEIGHT, const std::string &title = DEFAULT_TITLE.data()) : m_window(createWindow(width,
00031             height, title)), m_width(width), m_height(height) {}
00032
00033         ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr; }
00034
00035         Window(const Window&) = delete;
00036         Window& operator=(const Window&) = delete;
00037
00038         [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
00039             &title);
00040         void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00041
00042         [[nodiscard]] GLFWwindow* getGLFWWindow() const { return m_window; }
```

```

00041     [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; }
00042     [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00043     void resetWindowResizedFlag() { m_framebufferResized = false; }
00044
00045     void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);
00046
00047     private:
00048
00049         static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00050
00051         GLFWwindow* m_window{nullptr};
00052         uint32_t m_width{DEFAULT_WIDTH};
00053         uint32_t m_height{DEFAULT_HEIGHT};
00054
00055         bool m_framebufferResized = false;
00056
00057     }; // class Window
00058
00059 } // namespace ven

```

8.61 /home/runner/work/VEngine/VEngine/README.md File Reference

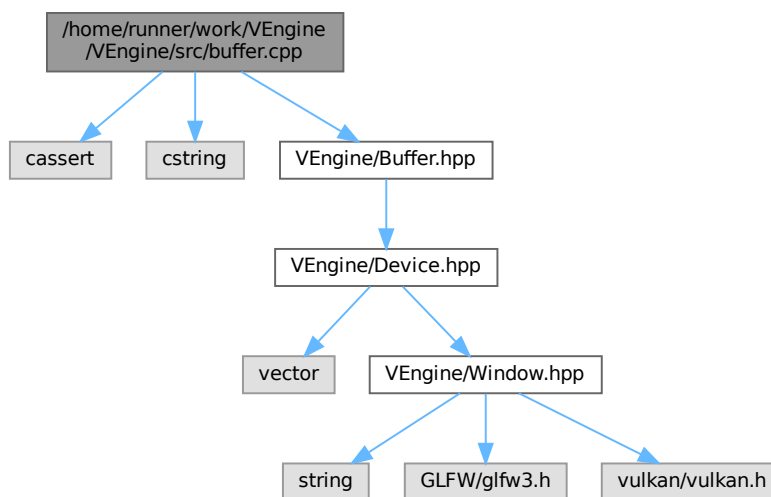
8.62 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```

#include <cassert>
#include <cstring>
#include "VEngine/Buffer.hpp"

```

Include dependency graph for buffer.cpp:



8.63 buffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002 #include <cstring>
00003
00004 #include "VEngine/Buffer.hpp"
00005

```

```

00006 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}
00007 {
00008     m_bufferSize = m_alignmentSize * m_instanceCount;
00009     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00010 }
00011
00012 ven::Buffer::~~Buffer()
00013 {
00014     unmap();
00015     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00016     vkFreeMemory(m_device.device(), m_memory, nullptr);
00017 }
00018
00019 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00020 {
00021     assert(m_buffer && m_memory && "Called map on buffer before create");
00022     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00023 }
00024
00025 void ven::Buffer::unmap()
00026 {
00027     if (m_mapped != nullptr) {
00028         vkUnmapMemory(m_device.device(), m_memory);
00029         m_mapped = nullptr;
00030     }
00031 }
00032
00033 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
const
00034 {
00035     assert(m_mapped && "Cannot copy to unmapped buffer");
00036
00037     if (size == VK_WHOLE_SIZE) {
00038         memcpy(m_mapped, data, m_bufferSize);
00039     } else {
00040         auto memOffset = static_cast<char *>(m_mapped);
00041         memOffset += offset;
00042         memcpy(memOffset, data, size);
00043     }
00044 }
00045
00046 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00047 {
00048     VkMappedMemoryRange mappedRange = {};
00049     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00050     mappedRange.memory = m_memory;
00051     mappedRange.offset = offset;
00052     mappedRange.size = size;
00053     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00054 }
00055
00056 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00057 {
00058     VkMappedMemoryRange mappedRange = {};
00059     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00060     mappedRange.memory = m_memory;
00061     mappedRange.offset = offset;
00062     mappedRange.size = size;
00063     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00064 }

```

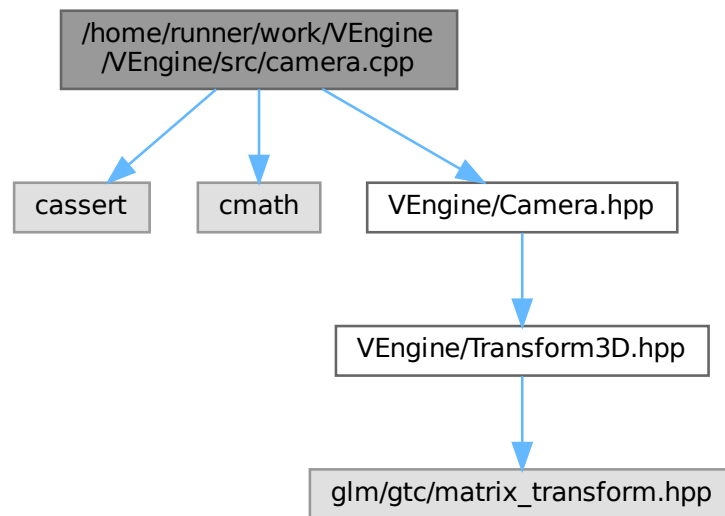
8.64 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```

#include <cassert>
#include <cmath>
#include "VEngine/Camera.hpp"

```

Include dependency graph for camera.cpp:



8.65 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 / (top - bottom);
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) / (top - bottom);
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;

```

```

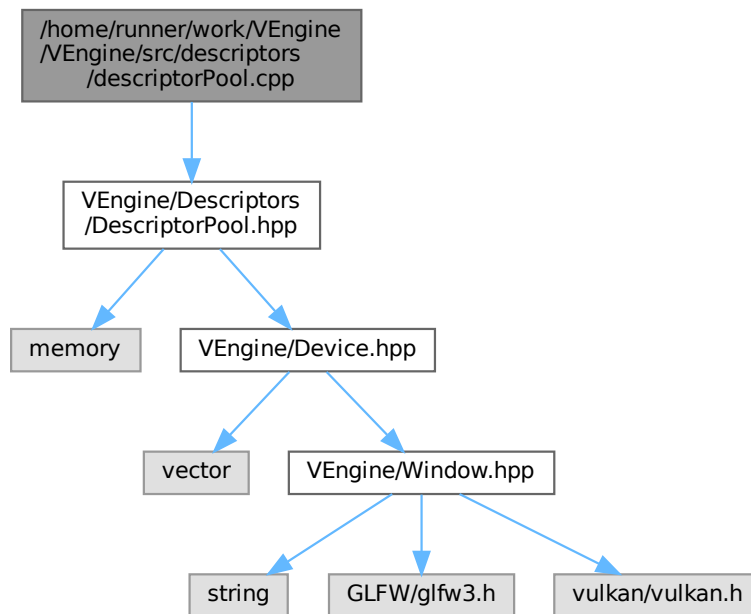
00038     m_viewMatrix[2][0] = u.z;
00039     m_viewMatrix[0][1] = v.x;
00040     m_viewMatrix[1][1] = v.y;
00041     m_viewMatrix[2][1] = v.z;
00042     m_viewMatrix[0][2] = w.x;
00043     m_viewMatrix[1][2] = w.y;
00044     m_viewMatrix[2][2] = w.z;
00045     m_viewMatrix[3][0] = -dot(u, position);
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 }

```

8.66 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp File Reference

```
#include "VEngine/Descriptors/DescriptorPool.hpp"
```

Include dependency graph for descriptorPool.cpp:



8.67 descriptorPool.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Descriptors/DescriptorPool.hpp"
00002
00003 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device(device)
00004 {
00005     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00006     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00007     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00008     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00009     descriptorPoolInfo.maxSets = maxSets;
00010     descriptorPoolInfo.flags = poolFlags;
00011
00012     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) !=
00013         VK_SUCCESS) {
00014         throw std::runtime_error("failed to create descriptor pool!");
00015     }
00016 }
00017
00018 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorSetLayout,
    VkDescriptorSet &descriptor) const
00019 {
00020     VkDescriptorSetAllocateInfo allocInfo{};
00021     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00022     allocInfo.descriptorPool = m_descriptorPool;
00023     allocInfo.pSetLayouts = &descriptorSetLayout;
00024     allocInfo.descriptorSetCount = 1;
00025

```

```

00026 // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00027 // a new pool whenever an old pool fills up. But this is beyond our current scope
00028 return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00029 }

```

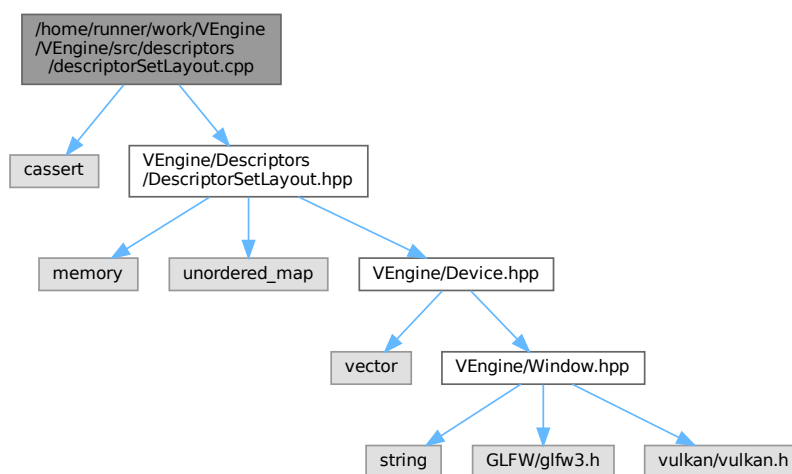
8.68 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

```

Include dependency graph for descriptorSetLayout.cpp:



8.69 descriptorSetLayout.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Descriptors/DescriptorSetLayout.hpp"
00004
00005 ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t
binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t
count)
00006 {
00007     assert(m_bindings.contains(binding) == 0 && "Binding already exists in layout");
00008     VkDescriptorSetLayoutBinding layoutBinding{};
00009     layoutBinding.binding = binding;
00010     layoutBinding.descriptorType = descriptorType;
00011     layoutBinding.descriptorCount = count;
00012     layoutBinding.stageFlags = stageFlags;
00013     m_bindings[binding] = layoutBinding;
00014     return *this;
00015 }
00016
00017 ven::DescriptorSetLayout::DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
00018 {
00019     std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
00020     setLayoutBindings.reserve(bindings.size());
00021     for (auto [fst, snd] : bindings) {
00022         setLayoutBindings.push_back(snd);
00023     }
00024 }

```



```

00025     VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
00026     descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
00027     descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
00028     descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();
00029
00030     if (vkCreateDescriptorSetLayout (
00031         m_device.device(),
00032         &descriptorSetLayoutInfo,
00033         nullptr,
00034         &m_descriptorSetLayout) != VK_SUCCESS) {
00035         throw std::runtime_error("failed to create descriptor set layout!");
00036     }
00037 }

```

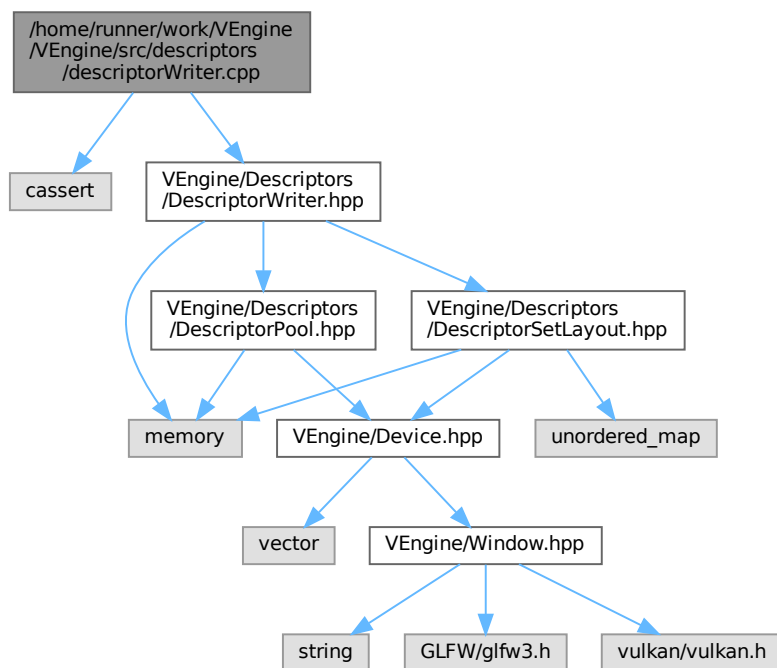
8.70 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference

```

#include <cassert>
#include "VEngine/Descriptors/DescriptorWriter.hpp"

```

Include dependency graph for descriptorWriter.cpp:



8.71 descriptorWriter.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Descriptors/DescriptorWriter.hpp"
00004
00005 ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const
    VkDescriptorBufferInfo *bufferInfo)
00006 {

```

```

00007     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00008
00009     const auto &bindingDescription = m_setLayout.m_bindings.at(binding);
00010
00011     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00012
00013     VkWriteDescriptorSet write{};
00014     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00015     write.descriptorType = bindingDescription.descriptorType;
00016     write.dstBinding = binding;
00017     write.pBufferInfo = bufferInfo;
00018     write.descriptorCount = 1;
00019
00020     m_writes.push_back(write);
00021     return *this;
00022 }
00023
00024 ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const
VkDescriptorImageInfo *imageInfo)
00025 {
00026     assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");
00027
00028     const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings.at(binding);
00029
00030     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
expects multiple");
00031
00032     VkWriteDescriptorSet write{};
00033     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00034     write.descriptorType = bindingDescription.descriptorType;
00035     write.dstBinding = binding;
00036     write.pImageInfo = imageInfo;
00037     write.descriptorCount = 1;
00038
00039     m_writes.push_back(write);
00040     return *this;
00041 }
00042
00043 bool ven::DescriptorWriter::build(VkDescriptorSet &set)
00044 {
00045     if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
00046         return false;
00047     }
00048     overwrite(set);
00049     return true;
00050 }
00051
00052 void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set) {
00053     for (auto &[sType, pNext, dstSet, dstBinding, dstArrayElement, descriptorCount, descriptorType,
pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
00054         dstSet = set;
00055     }
00056     vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
m_writes.data(), 0, nullptr);
00057 }

```

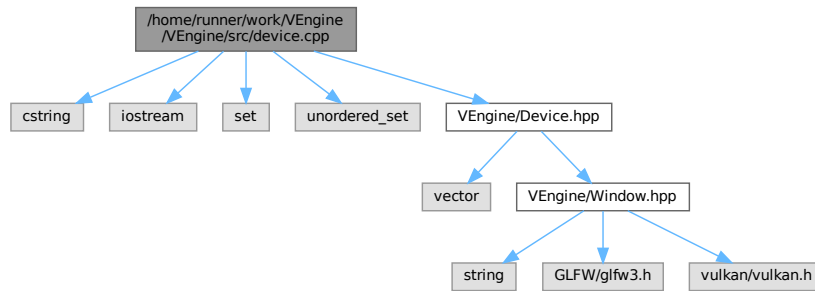
8.72 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

```

#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)

8.72.1 Function Documentation

8.72.1.1 CreateDebugUtilsMessengerEXT()

```

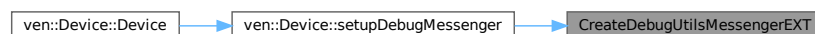
VkResult CreateDebugUtilsMessengerEXT (
    const VkInstance instance,
    const VkDebugUtilsMessengerCreateInfoEXT * pCreateInfo,
    const VkAllocationCallbacks * pAllocator,
    VkDebugUtilsMessengerEXT * pDebugMessenger)

```

Definition at line 16 of file [device.cpp](#).

Referenced by [ven::Device::setupDebugMessenger\(\)](#).

Here is the caller graph for this function:



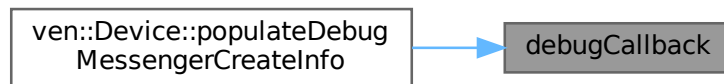
8.72.1.2 debugCallback()

```
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:



8.72.1.3 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:



8.73 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 VkDebugUtilsMessageSeverityFlagsEXT
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     constexpr VkApplicationInfo appInfo = {
00062         .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
00063         .pNext = nullptr,
00064         .pApplicationName = "VEngine App",
00065         .applicationVersion = VK_MAKE_VERSION(1, 0, 0),
00066         .pEngineName = "VEngine",
00067         .engineVersion = VK_MAKE_VERSION(1, 0, 0),
00068         .apiVersion = VK_API_VERSION_1_0
00069     };
00070
00071     VkInstanceCreateInfo createInfo = {};
00072     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00073     createInfo.pApplicationInfo = &appInfo;

```

```

00074
00075     const std::vector<const char *> extensions = getRequiredExtensions();
00076     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00077     createInfo.ppEnabledExtensionNames = extensions.data();
00078
00079     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00080     if (enableValidationLayers) {
00081         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00082         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00083
00084         populateDebugMessengerCreateInfo(debugCreateInfo);
00085         createInfo.pNext = &debugCreateInfo;
00086     } else {
00087         createInfo.enabledLayerCount = 0;
00088         createInfo.pNext = nullptr;
00089     }
00090
00091     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
00092         throw std::runtime_error("failed to create instance!");
00093     }
00094
00095     hasGlfwRequiredInstanceExtensions();
00096 }
00097
00098 void ven::Device::pickPhysicalDevice()
00099 {
00100     uint32_t deviceCount = 0;
00101     vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
00102     if (deviceCount == 0) {
00103         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00104     }
00105     std::cout << "Device count: " << deviceCount << '\n';
00106     std::vector<VkPhysicalDevice> devices(deviceCount);
00107     vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());
00108
00109     for (const auto &device : devices) {
00110         if (isDeviceSuitable(device)) {
00111             m_physicalDevice = device;
00112             break;
00113         }
00114     }
00115
00116     if (m_physicalDevice == VK_NULL_HANDLE) {
00117         throw std::runtime_error("failed to find a suitable GPU!");
00118     }
00119
00120     vkGetPhysicalDeviceProperties(m_physicalDevice, &m_properties);
00121     std::cout << "physical device: " << m_properties.deviceName << '\n';
00122 }
00123
00124 void ven::Device::createLogicalDevice()
00125 {
00126     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
        findQueueFamilies(m_physicalDevice);
00127
00128     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00129     const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, presentFamily};
00130     float queuePriority = 1.0F;
00131
00132     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00133         VkDeviceQueueCreateInfo queueCreateInfo = {};
00134         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00135         queueCreateInfo.queueFamilyIndex = queueFamily;
00136         queueCreateInfo.queueCount = 1;
00137         queueCreateInfo.pQueuePriorities = &queuePriority;
00138         queueCreateInfos.push_back(queueCreateInfo);
00139     }
00140
00141     VkPhysicalDeviceFeatures deviceFeatures = {};
00142     deviceFeatures.samplerAnisotropy = VK_TRUE;
00143
00144     VkDeviceCreateInfo createInfo = {};
00145     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00146
00147     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00148     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00149
00150     createInfo.pEnabledFeatures = &deviceFeatures;
00151     createInfo.enabledExtensionCount = static_cast<uint32_t>(m_deviceExtensions.size());
00152     createInfo.ppEnabledExtensionNames = m_deviceExtensions.data();
00153
00154     // might not really be necessary anymore because device specific validation layers
00155     // have been deprecated
00156     if (enableValidationLayers) {
00157         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00158         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00159     } else {

```

```

00160         createInfo.enabledLayerCount = 0;
00161     }
00162
00163     if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
00164         throw std::runtime_error("failed to create logical device!");
00165     }
00166
00167     vkGetDeviceQueue(m_device, graphicsFamily, 0, &m_graphicsQueue);
00168     vkGetDeviceQueue(m_device, presentFamily, 0, &m_presentQueue);
00169 }
00170
00171 void ven::Device::createCommandPool()
00172 {
00173     const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
00174         findPhysicalQueueFamilies();
00175
00176     const VkCommandPoolCreateInfo poolInfo = {
00177         .sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO,
00178         .pNext = nullptr,
00179         .flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
00180             VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT,
00181         .queueFamilyIndex = graphicsFamily
00182     };
00183
00184     if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
00185         throw std::runtime_error("failed to create command pool!");
00186     }
00187 }
00188 bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00189 {
00190     const QueueFamilyIndices indices = findQueueFamilies(device);
00191     const bool extensionsSupported = checkDeviceExtensionSupport(device);
00192     bool swapChainAdequate = false;
00193
00194     if (extensionsSupported) {
00195         auto [capabilities, formats, presentModes] = querySwapChainSupport(device);
00196         swapChainAdequate = !formats.empty() && !presentModes.empty();
00197     }
00198
00199     VkPhysicalDeviceFeatures supportedFeatures;
00200     vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00201
00202     return indices.isComplete() && extensionsSupported && swapChainAdequate &&
00203         (supportedFeatures.samplerAnisotropy != 0U);
00204 }
00205 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00206 {
00207     createInfo = {};
00208     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00209     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
00210         VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00211     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
00212         VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
00213         VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00214     createInfo.pfnUserCallback = debugCallback;
00215     createInfo.pUserData = nullptr; // Optional
00216 }
00217 void ven::Device::setupDebugMessenger()
00218 {
00219     if (!enableValidationLayers) { return; }
00220     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00221     populateDebugMessengerCreateInfo(createInfo);
00222     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
00223         VK_SUCCESS) {
00224         throw std::runtime_error("failed to set up debug messenger!");
00225     }
00226 }
00227 bool ven::Device::checkValidationLayerSupport() const
00228 {
00229     uint32_t layerCount = 0;
00230     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00231
00232     std::vector<VkLayerProperties> availableLayers(layerCount);
00233     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00234
00235     for (const char *validationLayer : m_validationLayers) {
00236         bool layerFound = false;
00237
00238         for (const auto &[layerName, specVersion, implementationVersion, description] :
00239             availableLayers) {
00240             if (strcmp(layerName, validationLayer) == 0) {
00241                 layerFound = true;
00242                 break;
00243             }
00244         }
00245     }
00246 }

```

```

00242         }
00243     }
00244     if (!layerFound) {
00245         return false;
00246     }
00247 }
00248
00249 return true;
00250 }
00251
00252 std::vector<const char *> ven::Device::getRequiredExtensions() const
00253 {
00254     uint32_t glfwExtensionCount = 0;
00255     const char **glfwExtensions = nullptr;
00256     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00257
00258     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00259
00260     if (enableValidationLayers) {
00261         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00262     }
00263
00264     return extensions;
00265 }
00266
00267 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00268 {
00269     uint32_t extensionCount = 0;
00270     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
00271     std::vector<VkExtensionProperties> extensions(extensionCount);
00272     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00273
00274     std::cout << "available extensions:\n";
00275     std::unordered_set<std::string> available;
00276     for (const auto &[extensionName, specVersion] : extensions) {
00277         std::cout << '\t' << extensionName << '\n';
00278         available.insert(extensionName);
00279     }
00280
00281     std::cout << "required extensions:\n";
00282     for (const std::vector<const char *> requiredExtensions = getRequiredExtensions(); const auto
&required : requiredExtensions) {
00283         std::cout << "\t" << required << '\n';
00284         if (!available.contains(required)) {
00285             throw std::runtime_error("Missing required glfw extension");
00286         }
00287     }
00288 }
00289
00290 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00291 {
00292     uint32_t extensionCount = 0;
00293     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00294
00295     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00296     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
availableExtensions.data());
00297
00298     std::set<std::string> requiredExtensions(m_deviceExtensions.begin(), m_deviceExtensions.end());
00299     for (const auto &[extensionName, specVersion] : availableExtensions) {
00300         requiredExtensions.erase(extensionName);
00301     }
00302
00303     return requiredExtensions.empty();
00304 }
00305
00306 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00307 {
00308     QueueFamilyIndices indices;
00309     uint32_t queueFamilyCount = 0;
00310     uint32_t index = 0;
00311     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00312     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00313     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00314
00315     for (const auto &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
queueFamilies) {
00316         if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00317             indices.graphicsFamily = index;
00318             indices.graphicsFamilyHasValue = true;
00319         }
00320         VkBool32 presentSupport = 0U;
00321         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00322         if (queueCount > 0 && (presentSupport != 0U)) {
00323             indices.presentFamily = index;
00324             indices.presentFamilyHasValue = true;
00325         }
00326     }

```



```

00326         if (indices.isComplete()) {
00327             break;
00328         }
00329         index++;
00330     }
00331     return indices;
00332 }
00333
00334 ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00335 {
00336     uint32_t formatCount = 0;
00337     uint32_t presentModeCount = 0;
00338     SwapChainSupportDetails details;
00339     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
00340
00341     vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
00342     if (formatCount != 0) {
00343         details.formats.resize(formatCount);
00344         vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
00345     }
00346     vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
00347     if (presentModeCount != 0) {
00348         details.presentModes.resize(presentModeCount);
00349         vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount,
00350         details.presentModes.data());
00351     }
00352     return details;
00353 }
00354
00355 VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
00356 tiling, const VkFormatFeatureFlags features) const
00357 {
00358     for (const VkFormat format : candidates) {
00359         VkFormatProperties props;
00360         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00361         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00362             return format;
00363         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
00364 features) {
00365             return format;
00366         }
00367     }
00368     throw std::runtime_error("failed to find supported format!");
00369 }
00370
00371 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
00372 properties) const
00373 {
00374     VkPhysicalDeviceMemoryProperties memProperties;
00375     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
00376
00377     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00378         if (((typeFilter & (1 << i)) != 0U) &&
00379         (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00380             return i;
00381         }
00382     }
00383     throw std::runtime_error("failed to find suitable m_memory type!");
00384 }
00385
00386 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
00387 VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00388 {
00389     VkBufferCreateInfo bufferInfo{};
00390     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00391     bufferInfo.size = size;
00392     bufferInfo.usage = usage;
00393     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00394
00395     if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00396         throw std::runtime_error("failed to create vertex m_buffer!");
00397     }
00398
00399     VkMemoryRequirements memRequirements;
00400     vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);
00401
00402     const VkMemoryAllocateInfo allocInfo{
00403         .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
00404         .pNext = nullptr,
00405         .allocationSize = memRequirements.size,
00406         .memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties)
00407     };
00408
00409     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00410         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00411     }

```

```

00408     }
00409
00410     vkBindBufferMemory(m_device, buffer, bufferMemory, 0);
00411 }
00412
00413 VkCommandBuffer ven::Device::beginSingleTimeCommands() const
00414 {
00415     VkCommandBufferAllocateInfo allocInfo{};
00416     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00417     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00418     allocInfo.commandPool = m_commandPool;
00419     allocInfo.commandBufferCount = 1;
00420
00421     VkCommandBuffer commandBuffer = nullptr;
00422     vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);
00423
00424     VkCommandBufferBeginInfo beginInfo{};
00425     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00426     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00427
00428     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00429     return commandBuffer;
00430 }
00431
00432 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00433 {
00434     vkEndCommandBuffer(commandBuffer);
00435
00436     VkSubmitInfo submitInfo{};
00437     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00438     submitInfo.commandBufferCount = 1;
00439     submitInfo.pCommandBuffers = &commandBuffer;
00440
00441     vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
00442     vkQueueWaitIdle(m_graphicsQueue);
00443
00444     vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
00445 }
00446
00447 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
size) const
00448 {
00449     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00450
00451     VkBufferCopy copyRegion{};
00452     copyRegion.srcOffset = 0; // Optional
00453     copyRegion.dstOffset = 0; // Optional
00454     copyRegion.size = size;
00455     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00456
00457     endSingleTimeCommands(commandBuffer);
00458 }
00459
00460 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
const uint32_t height, const uint32_t layerCount) const
00461 {
00462     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00463     const VkBufferImageCopy region{
00464         .bufferOffset = 0,
00465         .bufferRowLength = 0,
00466         .bufferImageHeight = 0,
00467         .imageSubresource = {
00468             .aspectMask = VK_IMAGE_ASPECT_COLOR_BIT,
00469             .mipLevel = 0,
00470             .baseArrayLayer = 0,
00471             .layerCount = layerCount
00472         },
00473         .imageOffset = {0, 0, 0},
00474         .imageExtent = {width, height, 1}
00475     };
00476
00477     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
&region);
00478     endSingleTimeCommands(commandBuffer);
00479 }
00480
00481 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const
00482 {
00483     if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00484         throw std::runtime_error("failed to create image!");
00485     }
00486
00487     VkMemoryRequirements memRequirements;
00488     vkGetImageMemoryRequirements(m_device, image, &memRequirements);
00489
00490     VkMemoryAllocateInfo allocInfo{};

```

```

00491     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00492     allocInfo.allocationSize = memRequirements.size;
00493     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00494
00495     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00496         throw std::runtime_error("failed to allocate image memory!");
00497     }
00498
00499     if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
00500         throw std::runtime_error("failed to bind image memory!");
00501     }
00502 }
00503
00504 void ven::Device::transitionImageLayout(const VkImage image, const VkFormat format, const
VkImageLayout oldLayout, const VkImageLayout newLayout, const uint32_t mipLevels, const uint32_t
layerCount) const {
00505     // uses an image memory barrier transition image layouts and transfer queue
00506     // family ownership when VK_SHARING_MODE_EXCLUSIVE is used. There is an
00507     // equivalent buffer memory barrier to do this for buffers
00508     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00509
00510     VkImageMemoryBarrier barrier{};
00511     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00512     barrier.oldLayout = oldLayout;
00513     barrier.newLayout = newLayout;
00514
00515     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00516     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00517
00518     barrier.image = image;
00519     barrier.subresourceRange.baseMipLevel = 0;
00520     barrier.subresourceRange.levelCount = mipLevels;
00521     barrier.subresourceRange.baseArrayLayer = 0;
00522     barrier.subresourceRange.layerCount = layerCount;
00523
00524     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00525         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00526         if (format == VK_FORMAT_D32_SFLOAT_S8_UINT || format == VK_FORMAT_D24_UNORM_S8_UINT) {
00527             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00528         }
00529     } else {
00530         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00531     }
00532
00533     VkPipelineStageFlags sourceStage = 0;
00534     VkPipelineStageFlags destinationStage = 0;
00535
00536     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00537         barrier.srcAccessMask = 0;
00538         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00539
00540         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00541         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00542     } else if (
00543         oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00544         barrier.srcAccessMask = 0;
00545         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00546
00547         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00548         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00549     } else if (
00550         oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL &&
00551         newLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00552         barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00553         barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00554
00555         sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00556         destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00557     } else if (
00558         oldLayout == VK_IMAGE_LAYOUT_UNDEFINED &&
00559         newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00560         barrier.srcAccessMask = 0;
00561         barrier.dstAccessMask =
00562             VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00563
00564         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00565         destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00566     } else {
00567         throw std::invalid_argument("unsupported layout transition!");
00568     }
00569     vkCmdPipelineBarrier(
00570         commandBuffer,
00571         sourceStage,
00572         destinationStage,
00573         0,
00574         0,
00575         nullptr,

```



```

00021
00022     framePools.resize(MAX_FRAMES_IN_FLIGHT);
00023     const auto framePoolBuilder = DescriptorPool::Builder(m_device)
00024         .setMaxSets(1000)
00025         .addPoolSize(VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1000)
00026         .addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1000)
00027         .setPoolFlags(VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT);
00028     for (auto & framePool : framePools) {
00029         framePool = framePoolBuilder.build();
00030     }
00031
00032     loadObjects();
00033 }
00034
00035 void ven::Engine::createInstance()
00036 {
00037     uint32_t glfwExtensionCount = 0;
00038     const char** glfwExtensions = nullptr;
00039     VkInstanceCreateInfo createInfo{};
00040     constexpr VkApplicationInfo appInfo{
00041         .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
00042         .pNext = nullptr,
00043         .pApplicationName = "VEngine App",
00044         .applicationVersion = VK_MAKE_API_VERSION(0, 1, 0, 0),
00045         .pEngineName = "VEngine",
00046         .engineVersion = VK_MAKE_API_VERSION(0, 1, 0, 0),
00047         .apiVersion = VK_API_VERSION_1_0
00048     };
00049
00050     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00051     createInfo.pApplicationInfo = &appInfo;
00052     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00053     createInfo.enabledExtensionCount = glfwExtensionCount;
00054     createInfo.ppEnabledExtensionNames = glfwExtensions;
00055
00056     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS)
00057     {
00058         throw std::runtime_error("Failed to create Vulkan instance");
00059     }
00060 }
00061
00062 void ven::Engine::loadObjects()
00063 {
00064     constexpr std::array lightColors{
00065         Colors::RED_4,
00066         Colors::GREEN_4,
00067         Colors::BLUE_4,
00068         Colors::YELLOW_4,
00069         Colors::CYAN_4,
00070         Colors::MAGENTA_4
00071     };
00072     auto& quad = m_sceneManager.createObject();
00073     quad.setName("quad");
00074     quad.setModel(Model::createModelFromFile(m_device, "assets/models/quad.obj"));
00075     quad.transform.translation = {0.F, .5F, 0.F};
00076     quad.transform.scale = {3.F, 1.F, 3.F};
00077
00078     auto& flatVase = m_sceneManager.createObject();
00079     flatVase.setName("flat vase");
00080     flatVase.setModel(Model::createModelFromFile(m_device, "assets/models/flat_vase.obj"));
00081     flatVase.transform.translation = {-0.5F, .5F, 0.F};
00082     flatVase.transform.scale = {3.F, 1.5F, 3.F};
00083
00084     auto& smoothVase = m_sceneManager.createObject();
00085     smoothVase.setName("smooth vase");
00086     smoothVase.setModel(Model::createModelFromFile(m_device, "assets/models/smooth_vase.obj"));
00087     smoothVase.transform.translation = {.5F, .5F, 0.F};
00088     smoothVase.transform.scale = {3.F, 1.5F, 3.F};
00089
00090     for (std::size_t i = 0; i < lightColors.size(); i++)
00091     {
00092         glm::mat4 rotateLight = rotate(
00093             glm::mat4(1.F),
00094             static_cast<float>(i) * glm::two_pi<float>() / static_cast<float>(lightColors.size()),
00095             {0.F, -1.F, 0.F}
00096         );
00097         auto& pointLight = m_sceneManager.createLight();
00098         pointLight.color = lightColors.at(i);
00099         pointLight.transform.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F, 1.F));
00100     }
00101 }
00102
00103 void ven::Engine::mainLoop()
00104 {
00105     Clock clock;
00106     GlobalUbo ubo{};
00107     Camera camera{};

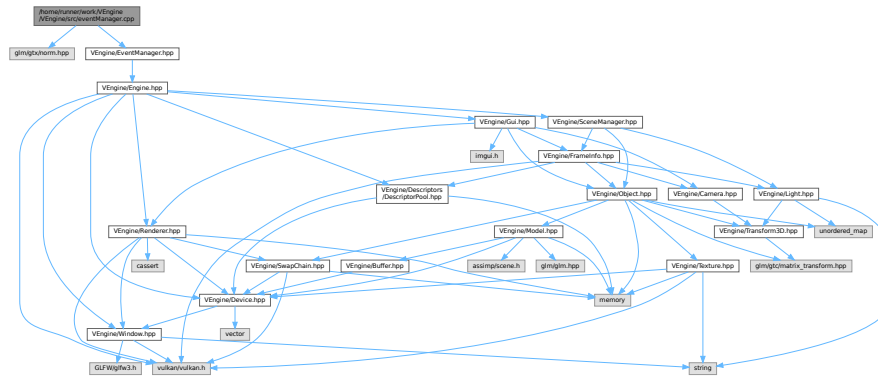
```

```

00108     EventManager eventManager{};
00109     VkCommandBuffer_T *commandBuffer = nullptr;
00110     float frameTime = 0.0F;
00111     unsigned long frameIndex = 0;
00112     std::unique_ptr<DescriptorSetLayout> globalSetLayout =
00113         DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
VK_SHADER_STAGE_ALL_GRAPHICS).build();
00114     std::vector<std::unique_ptr<Buffer>> uboBuffers(MAX_FRAMES_IN_FLIGHT);
00115     std::vector<VkDescriptorSet> globalDescriptorSets(MAX_FRAMES_IN_FLIGHT);
00116     ObjectRenderSystem objectRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorSetLayout());
00117     PointLightRenderSystem pointLightRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
globalSetLayout->getDescriptorSetLayout());
00118     VkDescriptorBufferInfo bufferInfo{};
00119
00120     for (auto& uboBuffer : uboBuffers)
00121     {
00122         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00123         uboBuffer->map();
00124     }
00125     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00126         bufferInfo = uboBuffers[i]->descriptorInfo();
00127         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
&bufferInfo).build(globalDescriptorSets[i]);
00128     }
00129
00130     while (m_state != EXIT)
00131     {
00132         clock.update();
00133         frameTime = clock.getDeltaTime();
00134         eventManager.handleEvents(m_window.getGLFWWindow(), &m_state, camera, m_gui, frameTime);
00135         commandBuffer = m_renderer.beginFrame();
00136
00137         camera.setViewXYZ(camera.transform.translation, camera.transform.rotation);
00138         camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00139
00140         if (commandBuffer != nullptr) {
00141             frameIndex = m_renderer.getFrameIndex();
00142             framePools[frameIndex]->resetPool();
00143             FrameInfo frameInfo{.frameIndex=frameIndex,
00144                             .frameTime=frameTime,
00145                             .commandBuffer=commandBuffer,
00146                             .camera=camera,
00147                             .globalDescriptorSet=globalDescriptorSets[frameIndex],
00148                             .frameDescriptorPool=*framePools[frameIndex],
00149                             .objects=m_sceneManager.getObjects(),
00150                             .lights=m_sceneManager.getLights()
00151             };
00152             ubo.projection = camera.getProjection();
00153             ubo.view = camera.getView();
00154             ubo.inverseView = camera.getInverseView();
00155             m_sceneManager.updateBuffer(ubo, frameIndex, frameTime);
00156             uboBuffers.at(frameIndex)->writeToBuffer(&ubo);
00157             uboBuffers.at(frameIndex)->flush();
00158
00159             m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00160             objectRenderSystem.render(frameInfo);
00161             pointLightRenderSystem.render(frameInfo);
00162
00163             if (m_gui.getState() == VISIBLE) {
00164                 Gui::render(
00165                     &m_renderer,
00166                     m_sceneManager.getObjects(),
00167                     m_sceneManager.getLights(),
00168                     camera,
00169                     m_device.getPhysicalDevice(),
00170                     ubo
00171                 );
00172             }
00173
00174             m_renderer.endSwapChainRenderPass(commandBuffer);
00175             m_renderer.endFrame();
00176             commandBuffer = nullptr;
00177         }
00178     }
00179     Gui::cleanup();
00180     vkDeviceWaitIdle(m_device.device());
00181 }

```

```
#include <glm/gtx/norm.hpp>
#include "VEngine/EventManager.hpp"
Include dependency graph for eventManager.cpp:
```



- #define GLM_ENABLE_EXPERIMENTAL

8.76.1.1 GLM_ENABLE_EXPERIMENTAL

Definition at line 1 of file [EventManager.cpp](#).

[Go to the documentation of this file.](#)

```
00001 #define GLM_ENABLE_EXPERIMENTAL
00002 #include <glm/gtx/norm.hpp>
00003
00004 #include "VEngine/EventManager.hpp"
00005
00006 bool ven::EventManager::isKeyJustPressed(GLFWwindow* window, const int key, std::unordered_map<int,
00007 bool>& keyStates)
00008 {
00009     const bool isPressed = glfwGetKey(window, key) == GLFW_PRESS;
00010     const bool wasPressed = keyStates[key];
00011
00012     keyStates[key] = isPressed;
00013
00014     return isPressed && !wasPressed;
00015 }
00016
00017 template<typename Iterator>
00018 void ven::EventManager::processKeyActions(GLFWwindow* window, Iterator begin, Iterator end)
00019 {
00020     for (auto it = begin; it != end; ++it) {
```

```

00020         if (glfwGetKey(window, it->key) == GLFW_PRESS) {
00021             *it->dir += it->value;
00022         }
00023     }
00024 }
00025
00026 void ven::EventManager::moveCamera(GLFWwindow* window, Camera& camera, const float dt)
00027 {
00028     glm::vec3 rotate{0};
00029     glm::vec3 moveDir{0.F};
00030     static constexpr glm::vec3 upDir{0.F, -1.F, 0.F};
00031     const float yaw = camera.transform.rotation.y;
00032     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
00033     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00034     const std::array<KeyAction, 10> moveActions = {{
00035         {.key=DEFAULT_KEY_MAPPINGS.lookLeft, .dir=&rotate, .value={0.F, -1.F, 0.F}},
00036         {.key=DEFAULT_KEY_MAPPINGS.lookRight, .dir=&rotate, .value={0.F, 1.F, 0.F}},
00037         {.key=DEFAULT_KEY_MAPPINGS.lookUp, .dir=&rotate, .value={1.F, 0.F, 0.F}},
00038         {.key=DEFAULT_KEY_MAPPINGS.lookDown, .dir=&rotate, .value={-1.F, 0.F, 0.F}},
00039         {.key=DEFAULT_KEY_MAPPINGS.moveForward, .dir=&moveDir, .value=forwardDir},
00040         {.key=DEFAULT_KEY_MAPPINGS.moveBackward, .dir=&moveDir, .value=-forwardDir},
00041         {.key=DEFAULT_KEY_MAPPINGS.moveRight, .dir=&moveDir, .value=rightDir},
00042         {.key=DEFAULT_KEY_MAPPINGS.moveLeft, .dir=&moveDir, .value=-rightDir},
00043         {.key=DEFAULT_KEY_MAPPINGS.moveUp, .dir=&moveDir, .value=upDir},
00044         {.key=DEFAULT_KEY_MAPPINGS.moveDown, .dir=&moveDir, .value=-upDir}
00045     }};
00046
00047     processKeyActions(window, moveActions.begin(), moveActions.end());
00048
00049     if (length2(rotate) > EPSILON) {
00050         camera.transform.rotation += camera.getLookSpeed() * dt * normalize(rotate);
00051     }
00052     if (length2(moveDir) > EPSILON) {
00053         camera.transform.translation += camera.getMoveSpeed() * dt * normalize(moveDir);
00054     }
00055
00056     camera.transform.rotation.x = glm::clamp(camera.transform.rotation.x, -1.5F, 1.5F);
00057     camera.transform.rotation.y = glm::mod(camera.transform.rotation.y, glm::two_pi<float>());
00058 }
00059
00060 void ven::EventManager::handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera,
Gui& gui, const float dt) const
00061 {
00062     glfwPollEvents();
00063     if (glfwWindowShouldClose(window) == GLFW_TRUE) {
00064         updateEngineState(engineState, EXIT);
00065     }
00066     if (isKeyJustPressed(window, DEFAULT_KEY_MAPPINGS.toggleGui, m_keyState)) {
00067         gui.getState() == HIDDEN ? gui.setState(VISIBLE) : gui.setState(HIDDEN);
00068     }
00069     moveCamera(window, camera, dt);
00070 }

```

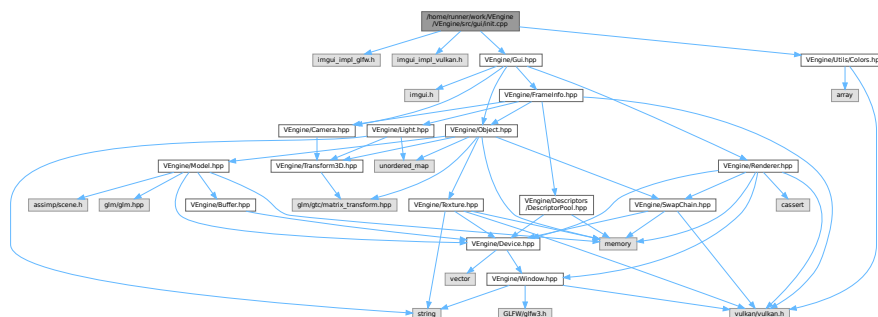
8.78 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference

```

#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include "VEngine/Gui.hpp"
#include "VEngine/Utils/Colors.hpp"

```

Include dependency graph for init.cpp:



8.79 init.cpp

[Go to the documentation of this file.](#)

```
00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include "VEngine/Gui.hpp"
00005 #include "VEngine/Utils/Colors.hpp"
00006
00007 ImGuiIO *ven::Gui::m_io = nullptr;
00008 float ven::Gui::m_intensity = 1.0F;
00009 float ven::Gui::m_shininess = DEFAULT_SHININESS;
00010
00011 void ven::Gui::init(GLFWwindow* window, const VkInstance instance, const Device* device, const
VkRenderPass renderPass)
00012 {
00013     VkDescriptorPool pool = nullptr;
00014     ImGui_ImplVulkan_InitInfo init_info{};
00015     ImGui::CreateContext();
00016     m_io = &ImGui::GetIO();
00017     m_io->IniFilename = "assets/imgui-config.txt";
00018     // ImGui::StyleColorsDark();
00019
00020     std::array<VkDescriptorPoolSize, 11> pool_sizes = {{
00021         { .type=VK_DESCRIPTOR_TYPE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00022         { .type=VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00023         { .type=VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00024         { .type=VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00025         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00026         { .type=VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00027         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00028         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00029         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00030         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00031         { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
00032     }};
00033     const VkDescriptorPoolCreateInfo pool_info = {
00034         VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00035         nullptr,
00036         VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00037         DESCRIPTOR_COUNT,
00038         std::size(pool_sizes),
00039         pool_sizes.data()
00040     };
00041
00042     if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00043         throw std::runtime_error("Failed to create ImGui descriptor pool");
00044     }
00045
00046     init_info.Instance = instance;
00047     init_info.PhysicalDevice = device->getPhysicalDevice();
00048     init_info.Device = device->device();
00049     init_info.Queue = device->graphicsQueue();
00050     init_info.DescriptorPool = pool;
00051     init_info.MinImageCount = 3;
00052     init_info.ImageCount = 3;
00053     init_info.MSAASamples = VK_SAMPLE_COUNT_1_BIT;
00054
00055     ImGui_ImplGlfw_InitForVulkan(window, true);
00056     ImGui_ImplVulkan_Init(&init_info, renderPass);
00057     initStyle();
00058 }
00059
00060 void ven::Gui::initStyle()
00061 {
00062     ImGuiStyle& style = ImGui::GetStyle();
00063     style.Alpha = 1.0;
00064     style.WindowRounding = 3;
00065     style.GrabRounding = 1;
00066     style.GrabMinSize = 20;
00067     style.FrameRounding = 3;
00068
00069     style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00070     style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
00071     style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
00072     style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
00073     style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
00074     style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
00075     style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
00076     style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
00077     style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
00078     style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
00079     style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
00080     style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
00081     style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
```



```

00016 void ven::Gui::render(Renderer* renderer, std::unordered_map<unsigned int, Object>& objects,
std::unordered_map<unsigned int, Light>& lights, Camera& camera, const VkPhysicalDevice
physicalDevice, GlobalUbo& ubo)
00017 {
00018     VkPhysicalDeviceProperties deviceProperties;
00019     vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
00020     ImGui_ImplVulkan_NewFrame();
00021     ImGui_ImplGlfw_NewFrame();
00022     ImGui::NewFrame();
00023     renderFrameWindow();
00024
00025     ImGui::Begin("Debug Window");
00026     rendererSection(renderer, ubo);
00027     cameraSection(camera);
00028     objectsSection(objects);
00029     lightsSection(lights);
00030     inputsSection(m_io);
00031     devicePropertiesSection(deviceProperties);
00032
00033     ImGui::End();
00034     ImGui::Render();
00035     ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
00036 }
00037
00038 void ven::Gui::renderFrameWindow()
00039 {
00040     const float framerate = m_io->Framerate;
00041     const float frametime = 1000.0F / framerate;
00042
00043     ImGui::SetNextWindowPos(ImVec2(0.0F, 0.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
00044     ImGui::Begin("Application Info", nullptr, ImGuiWindowFlags_NoDecoration | ImGuiWindowFlags_NoMove
| ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoSavedSettings | ImGuiWindowFlags_NoFocusOnAppearing |
ImGuiWindowFlags_NoNav);
00045     ImGui::Text("FPS: %.1f", framerate);
00046     ImGui::Text("Frame time: %.3fms", frametime);
00047     ImGui::End();
00048 }
00049
00050 void ven::Gui::rendererSection(Renderer *renderer, GlobalUbo& ubo)
00051 {
00052     if (ImGui::CollapsingHeader("Renderer")) {
00053         ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());
00054
00055         if (ImGui::BeginTable("ClearColorTable", 2)) {
00056             ImGui::TableNextColumn();
00057             std::array<float, 4> clearColor = renderer->getClearColor();
00058
00059             if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
00060                 const VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1],
clearColor[2], clearColor[3]}};
00061                 renderer->setClearColorValue(clearColorValue);
00062             }
00063
00064             ImGui::TableNextColumn();
00065             static int item_current = 0;
00066
00067             if (ImGui::Combo("Color Presets##clearColor",
&item_current,
00068 [(void*, const int idx, const char** out_text) -> bool {
00069     if (idx < 0 || idx >=
00070 static_cast<int>(std::size(Colors::COLOR_PRESETS_VK)) { return false; }
00071     *out_text = Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
long>(idx)).first;
00072     return true;
00073 },
00074 nullptr,
00075 std::size(Colors::COLOR_PRESETS_VK)) {
00076     renderer->setClearColorValue(Colors::COLOR_PRESETS_VK.at(static_cast<unsigned
long>(item_current)).second);
00077 }
00078
00079     ImGui::TableNextColumn();
00080     ImGui::ColorEdit4("Ambient Light Color", glm::value_ptr(ubo.ambientLightColor));
00081     ImGui::TableNextColumn();
00082     if (ImGui::Combo("Color Presets##ambientColor",
&item_current,
00083 [(void*, const int idx, const char** out_text) -> bool {
00084     if (idx < 0 || idx >=
00085 static_cast<int>(std::size(Colors::COLOR_PRESETS_4)) { return false; }
00086     *out_text = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
long>(idx)).first;
00087     return true;
00088 },
00089 nullptr,
00090 std::size(Colors::COLOR_PRESETS_4)) {
00091     ubo.ambientLightColor = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
long>(item_current)).second;

```

```

00092         }
00093
00094         ImGui::TableNextColumn();
00095         ImGui::SliderFloat(("Intensity##" + std::to_string(0)).c_str(), &ubo.ambientLightColor.a,
00096             0.0F, 1.0F);
00097         ImGui::TableNextColumn();
00098         if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a =
00099             DEFAULT_AMBIENT_LIGHT_INTENSITY; }
00100
00101         ImGui::EndTable();
00102     }
00103
00104     static bool fullscreen = false;
00105     if (ImGui::Checkbox("Fullscreen", &fullscreen)) {
00106         renderer->getWindow().setFullscreen(fullscreen, renderer->getWindow().getExtent().width,
00107             renderer->getWindow().getExtent().height);
00108     }
00109 }
00110
00111 void ven::Gui::cameraSection(Camera &camera)
00112 {
00113     if (ImGui::CollapsingHeader("Camera")) {
00114         float fov = camera.getFov();
00115         float near = camera.getNear();
00116         float far = camera.getFar();
00117         if (ImGui::BeginTable("CameraTable", 2)) {
00118             ImGui::TableNextColumn();
00119             ImGui::DragFloat3("Position", glm::value_ptr(camera.transform.translation), 0.1F);
00120             ImGui::TableNextColumn();
00121             if (ImGui::Button("Reset##position")) { camera.transform.translation = DEFAULT_POSITION; }
00122
00123             ImGui::TableNextColumn();
00124             ImGui::DragFloat3("Rotation", glm::value_ptr(camera.transform.rotation), 0.1F);
00125             ImGui::TableNextColumn();
00126             if (ImGui::Button("Reset##rotation")) { camera.transform.rotation = DEFAULT_ROTATION; }
00127
00128             ImGui::TableNextColumn();
00129             if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
00130                 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             float moveSpeed = camera.getMoveSpeed();
00146             if (ImGui::SliderFloat("Move speed", &moveSpeed, 0.1F, 10.0F)) {
00147                 camera.setMoveSpeed(moveSpeed); }
00148             ImGui::TableNextColumn();
00149             if (ImGui::Button("Reset##moveSpeed")) { camera.setMoveSpeed(DEFAULT_MOVE_SPEED); }
00150
00151             ImGui::TableNextColumn();
00152             float lookSpeed = camera.getLookSpeed();
00153             if (ImGui::SliderFloat("Look speed", &lookSpeed, 0.1F, 10.0F)) {
00154                 camera.setLookSpeed(lookSpeed); }
00155             ImGui::TableNextColumn();
00156             if (ImGui::Button("Reset##lookSpeed")) { camera.setLookSpeed(DEFAULT_LOOK_SPEED); }
00157
00158             ImGui::EndTable();
00159         }
00160     }
00161 }
00162
00163 void ven::Gui::objectsSection(std::unordered_map<unsigned int, Object>& objects)
00164 {
00165     if (ImGui::CollapsingHeader("Objects")) {
00166         bool open = false;
00167         for (auto& [id, object] : objects) {
00168             ImGui::PushStyleColor(ImGuiCol_Text, { Colors::GRAY_4.r, Colors::GRAY_4.g,
00169                 Colors::GRAY_4.b, 1.0F });
00170             open = ImGui::TreeNode(std::string(object.getName() + " [" +
00171                 std::to_string(object.getId()) + "]").c_str());
00172             ImGui::PopStyleColor(1);
00173             if (open) {
00174                 ImGui::DragFloat3(("Position##" + object.getName()).c_str(),
00175                     glm::value_ptr(object.transform.translation), 0.1F);
00176                 ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(),
00177                     glm::value_ptr(object.transform.rotation), 0.1F);
00178             }
00179         }
00180     }
00181 }

```

```

00169         ImGui::DragFloat3(("Scale##" + object.getName()).c_str(),
00170 glm::value_ptr(object.transform.scale), 0.1F);
00171         ImGui::Text("Address: %p", &object);
00172         ImGui::TreePop();
00173     }
00174 }
00175 }
00176
00177 void ven::Gui::lightsSection(std::unordered_map<unsigned int, Light> &lights)
00178 {
00179     if (ImGui::CollapsingHeader("Lights")) {
00180         bool open = false;
00181
00182         float tempIntensity = m_intensity;
00183         float tempShininess = m_shininess;
00184
00185         if (ImGui::BeginTable("LightTable", 2)) {
00186             ImGui::TableNextColumn();
00187             if (ImGui::SliderFloat("Global Intensity", &tempIntensity, 0.0F, 5.F)) {
00188                 m_intensity = tempIntensity;
00189                 for (auto&[fst, snd] : lights) {
00190                     snd.color.a = m_intensity;
00191                 }
00192             }
00193             ImGui::TableNextColumn();
00194             if (ImGui::Button("Reset")) {
00195                 m_intensity = DEFAULT_LIGHT_INTENSITY;
00196                 tempIntensity = m_intensity;
00197                 for (auto&[fst, snd] : lights) {
00198                     snd.color.a = m_intensity;
00199                 }
00200             }
00201
00202             ImGui::TableNextColumn();
00203             if (ImGui::SliderFloat("Global Shininess", &tempShininess, 0.0F, 512.F)) {
00204                 m_shininess = tempShininess;
00205                 for (auto&[fst, snd] : lights) {
00206                     snd.setShininess(m_shininess);
00207                 }
00208             }
00209
00210             ImGui::TableNextColumn();
00211             if (ImGui::Button("Reset")) {
00212                 m_shininess = DEFAULT_SHININESS;
00213                 tempShininess = m_shininess;
00214                 for (auto&[fst, snd] : lights) {
00215                     snd.setShininess(m_shininess);
00216                 }
00217             }
00218             ImGui::EndTable();
00219         }
00220
00221         for (auto& [id, light] : lights) {
00222             ImGui::PushStyleColor(ImGuiCol_Text, {light.color.r, light.color.g, light.color.b, 1.0F});
00223             open = ImGui::TreeNode(std::string(light.getName() + " [" + std::to_string(light.getId())
00224 + "]").c_str());
00225             ImGui::PopStyleColor(1);
00226             if (open) {
00227                 ImGui::Text("Address: %p", &light);
00228                 ImGui::DragFloat3(("Position##" + std::to_string(light.getId()).c_str(),
00229 glm::value_ptr(light.transform.translation), 0.1F);
00230                 ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId()).c_str(),
00231 glm::value_ptr(light.transform.rotation), 0.1F);
00232                 ImGui::DragFloat3(("Scale##" + std::to_string(light.getId()).c_str(),
00233 glm::value_ptr(light.transform.scale), 0.1F);
00234                 if (ImGui::BeginTable("ColorTable", 2)) {
00235                     ImGui::TableNextColumn();
00236                     ImGui::ColorEdit4(("Color##" +
00237 std::to_string(light.getId()).c_str(), glm::value_ptr(light.color)));
00238
00239                     ImGui::TableNextColumn();
00240                     static int item_current = 0;
00241                     if (ImGui::Combo("Color Presets",
00242 &item_current,
00243 [](void*, const int idx, const char** out_text) -> bool {
00244 if (idx < 0 || idx >=
00245 static_cast<int>(std::size(Colors::COLOR_PRESETS_3))) { return false; }
00246 *out_text = Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00247 long>(idx)).first;
00248 return true;
00249 },
00250 nullptr,
00251 std::size(Colors::COLOR_PRESETS_3))) {
00252 light.color = {Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00253 long>(item_current)).second, light.color.a};
00254 }
00255 }
00256 }

```

```

00247         ImGui::TableNextColumn();
00248         ImGui::SliderFloat(("Intensity###" + std::to_string(light.getId())).c_str(),
&light.color.a, 0.0F, 5.F);
00249         ImGui::TableNextColumn();
00250         if (ImGui::Button(("Reset###" + std::to_string(light.getId())).c_str())) {
light.color.a = DEFAULT_LIGHT_INTENSITY; }
00251         ImGui::TableNextColumn();
00252         float shininess = light.getShininess();
00253         if (ImGui::SliderFloat("Shininess", &shininess, 0.0F, 512.F)) {
light.setShininess(shininess);
00255         }
00256         ImGui::TableNextColumn();
00257         if (ImGui::Button("Reset##shininess")) { light.setShininess(DEFAULT_SHININESS); }
00258
00259         ImGui::EndTable();
00260     }
00261     ImGui::TreePop();
00262 }
00263 }
00264 }
00265 }
00266
00267 void ven::Gui::inputsSection(const ImGuiIO* io)
00268 {
00269     if (ImGui::CollapsingHeader("Input")) {
00270         ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io->MousePos.x, io->MousePos.y)
: ImGui::Text("Mouse pos: <INVALID>");
00271         ImGui::Text("Mouse delta: (%g, %g)", io->MouseDelta.x, io->MouseDelta.y);
00272         ImGui::Text("Mouse down:");
00273         for (int i = 0; i < static_cast<int>(std::size(io->MouseDown)); i++) {
00274             if (ImGui::IsMouseDown(i)) {
00275                 ImGui::SameLine();
00276                 ImGui::Text("b%d (%.02f secs)", i, io->MouseDownDuration[i]);
00277             }
00278         }
00279         ImGui::Text("Mouse wheel: %.1f", io->MouseWheel);
00280         ImGui::Text("Keys down:");
00281         for (auto key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
static_cast<ImGuiKey>(key + 1)) {
00282             if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
00283             ImGui::SameLine();
00284             ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\\\"%s\\\" : \\\"%s\\\" %d",
ImGui::GetKeyName(key), key);
00285         }
00286     }
00287 }
00288
00289 void ven::Gui::devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties)
00290 {
00291     if (ImGui::CollapsingHeader("Device Properties")) {
00292         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00293
00294             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00295             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion));
00296             ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00297             ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00298             ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00299             ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00300             ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00301             ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantsSize);
00302             ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00303             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00304             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",
deviceProperties.limits.maxImageDimension2D);
00305             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00306             ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00307             ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00308             ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00309             ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00310             ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00311             ImGui::EndTable();
00312         }
00313     }

```


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 }
```

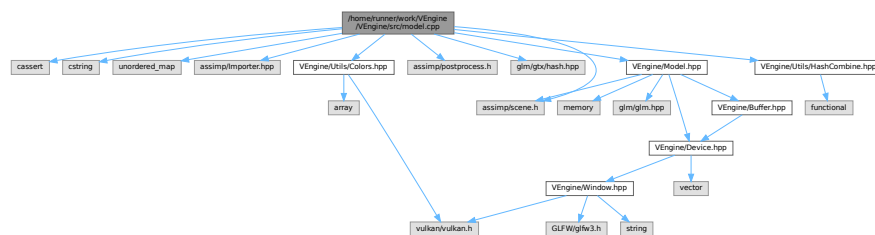
8.84 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference

```

#include <cassert>
#include <cstring>
#include <unordered_map>
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Model.hpp"
#include "VEngine/Utils/HashCombine.hpp"

```

Include dependency graph for model.cpp:



Classes

- struct `std::hash< ven::Model::Vertex >`

Macros

- `#define GLM_ENABLE_EXPERIMENTAL`

8.84.1 Macro Definition Documentation

8.84.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 9 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 #include <assimp/Importer.hpp>
00006 #include <assimp/scene.h>
00007 #include <assimp/postprocess.h>
00008
00009 #define GLM_ENABLE_EXPERIMENTAL
00010 #include <glm/gtx/hash.hpp>
00011
00012 #include "VEngine/Utils/Colors.hpp"
00013 #include "VEngine/Model.hpp"
00014 #include "VEngine/Utils/HashCombine.hpp"
00015
00016 template<>
00017 struct std::hash<ven::Model::Vertex> {
00018     size_t operator()(ven::Model::Vertex const &vertex) const noexcept {
00019         size_t seed = 0;
00020         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00021         return seed;
00022     }
00023 };
00024
00025 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
    m_indexCount(0)
00026 {
00027     createVertexBuffer(builder.vertices);
00028     createIndexBuffer(builder.indices);
00029 }
00030
00031 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00032 {
00033     m_vertexCount = static_cast<uint32_t>(vertices.size());
00034     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00035     const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
00036     uint32_t vertexSize = sizeof(vertices[0]);
00037
00038     Buffer stagingBuffer{m_device, vertexSize, m_vertexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00039
00040     stagingBuffer.map();
00041     stagingBuffer.writeToBuffer(vertices.data());
00042
00043     m_vertexBuffer = std::make_unique<Buffer>(m_device, vertexSize, m_vertexCount,
    VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00044
00045     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00046 }
00047
00048 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00049 {
00050     m_indexCount = static_cast<uint32_t>(indices.size());
00051     m_hasIndexBuffer = m_indexCount > 0;
00052
00053     if (!m_hasIndexBuffer) {
00054         return;
00055     }
00056
00057     uint32_t indexSize = sizeof(indices[0]);
00058
00059     Buffer stagingBuffer{m_device, indexSize, m_indexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00060
00061     stagingBuffer.map();
```

```

00062     stagingBuffer.writeToBuffer(indices.data());
00063
00064     m_indexBuffer = std::make_unique<Buffer>(m_device, indexSize, m_indexCount,
VK_BUFFER_USAGE_INDEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00065
00066     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), sizeof(indices[0]) *
m_indexCount);
00067 }
00068
00069 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00070 {
00071     if (m_hasIndexBuffer) {
00072         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00073     } else {
00074         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00075     }
00076 }
00077
00078 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00079 {
00080     const std::array buffers{m_vertexBuffer->getBuffer()};
00081     constexpr std::array<VkDeviceSize, 1> offsets{0};
00082     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets.data());
00083
00084     if (m_hasIndexBuffer) {
00085         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00086     }
00087 }
00088
00089 std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
&filename)
00090 {
00091     Builder builder{};
00092     builder.loadModel(filename);
00093     return std::make_unique<Model>(device, builder);
00094 }
00095
00096 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00097 {
00098     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00099     bindingDescriptions[0].binding = 0;
00100     bindingDescriptions[0].stride = sizeof(Vertex);
00101     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00102     return bindingDescriptions;
00103 }
00104
00105 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00106 {
00107     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00108
00109     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00110     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00111     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00112     attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00113
00114     return attributeDescriptions;
00115 }
00116
00117 void ven::Model::Builder::loadModel(const std::string &filename) {
00118     Assimp::Importer importer;
00119
00120     const aiScene* scene = importer.ReadFile(filename, aiProcess_Triangulate | aiProcess_FlipUVs |
aiProcess_CalcTangentSpace | aiProcess_GenNormals);
00121
00122     if ((scene == nullptr) || ((scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE) != 0U) ||
!scene->mRootNode) {
00123         throw std::runtime_error("Failed to load model with Assimp: " +
std::string(importer.GetErrorString()));
00124     }
00125
00126     vertices.clear();
00127     indices.clear();
00128
00129     processNode(scene->mRootNode, scene);
00130 }
00131
00132 void ven::Model::Builder::processNode(const aiNode* node, const aiScene* scene) {
00133     for (unsigned int i = 0; i < node->mNumMeshes; i++) {
00134         const aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
00135         processMesh(mesh, scene);
00136     }
00137
00138     for (unsigned int i = 0; i < node->mNumChildren; i++) {
00139         processNode(node->mChildren[i], scene);
00140     }
00141 }

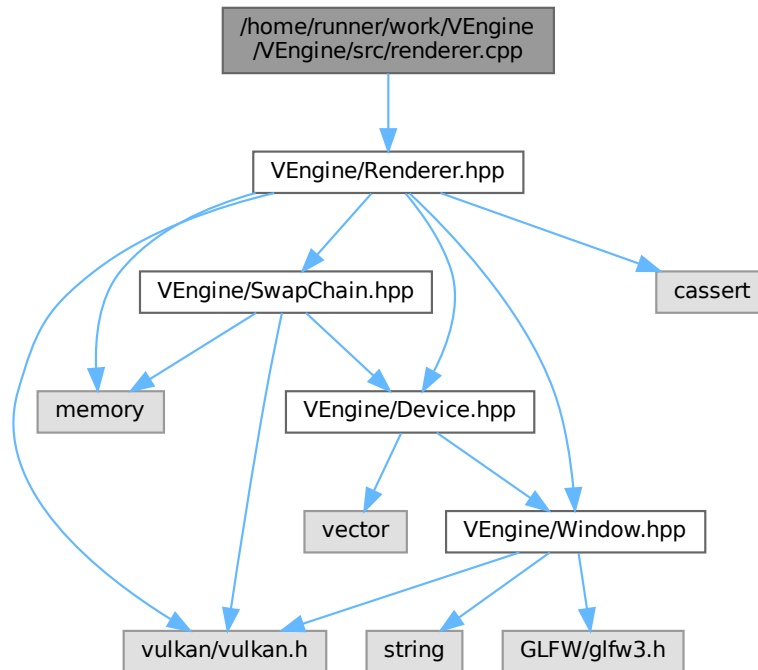
```

```
00142
00143 void ven::Model::Builder::processMesh(const aiMesh* mesh, const aiScene* scene) {
00144     std::unordered_map<Vertex, uint32_t> uniqueVertices;
00145
00146     for (unsigned int i = 0; i < mesh->mNumVertices; i++) {
00147         Vertex vertex{};
00148
00149         vertex.position = glm::vec3(
00150             mesh->mVertices[i].x,
00151             mesh->mVertices[i].y,
00152             mesh->mVertices[i].z
00153         );
00154
00155         if (mesh->HasNormals()) {
00156             vertex.normal = glm::vec3(
00157                 mesh->mNormals[i].x,
00158                 mesh->mNormals[i].y,
00159                 mesh->mNormals[i].z
00160             );
00161         }
00162
00163         if (mesh->mTextureCoords[0] != nullptr) {
00164             vertex.uv = glm::vec2(
00165                 mesh->mTextureCoords[0][i].x,
00166                 mesh->mTextureCoords[0][i].y
00167             );
00168         } else {
00169             vertex.uv = glm::vec2(0.0F, 0.0F);
00170         }
00171
00172         if (vertex.color == Colors::BLACK_3) {
00173             vertex.color = Colors::WHITE_3;
00174         }
00175
00176         if (!uniqueVertices.contains(vertex)) {
00177             uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00178             vertices.push_back(vertex);
00179         }
00180
00181         indices.push_back(uniqueVertices[vertex]);
00182     }
00183 }
00184
```

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(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()) !=
00013         VK_SUCCESS) {
00014         throw std::runtime_error("Failed to allocate command buffers");
00015     }
00016 }
00017 void ven::Renderer::freeCommandBuffers()
00018 {
00019     vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(),
00020         static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
00021     m_commandBuffers.clear();
00022 }
00023 void ven::Renderer::recreateSwapChain()
00024 {

```

```

00025     VkExtent2D extent = m_window.getExtent();
00026     while (extent.width == 0 || extent.height == 0) {
00027         extent = m_window.getExtent();
00028         glfwWaitEvents();
00029     }
00030     vkDeviceWaitIdle(m_device.device());
00031     if (m_swapChain == nullptr) {
00032         m_swapChain = std::make_unique<SwapChain>(m_device, extent);
00033     } else {
00034         std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
00035         m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
00036         if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
00037             throw std::runtime_error("Swap chain image/depth format changed");
00038         }
00039     }
00040     // well be back
00041 }
00042
00043 VkCommandBuffer ven::Renderer::beginFrame()
00044 {
00045     assert(!m_isFrameStarted && "Can't start new frame while previous one is still in progress");
00046
00047     const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
00048     if (result == VK_ERROR_OUT_OF_DATE_KHR) {
00049         recreateSwapChain();
00050         return nullptr;
00051     }
00052
00053     if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
00054         throw std::runtime_error("Failed to acquire swap chain image");
00055     }
00056
00057     m_isFrameStarted = true;
00058
00059     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00060     VkCommandBufferBeginInfo beginInfo{};
00061     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00062
00063     if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
00064         throw std::runtime_error("Failed to begin recording command m_buffer");
00065     }
00066     return commandBuffer;
00067 }
00068
00069 void ven::Renderer::endFrame()
00070 {
00071     assert(m_isFrameStarted && "Can't end frame that hasn't been started");
00072
00073     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00074     if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
00075         throw std::runtime_error("Failed to record command buffer");
00076     }
00077     if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer,
&m_currentImageIndex); result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
m_window.wasWindowResized()) {
00078         m_window.resetWindowResizedFlag();
00079         recreateSwapChain();
00080     }
00081     else if (result != VK_SUCCESS) {
00082         throw std::runtime_error("Failed to submit command buffer");
00083     }
00084
00085     m_isFrameStarted = false;
00086     m_currentFrameIndex = (m_currentFrameIndex + 1) % MAX_FRAMES_IN_FLIGHT;
00087 }
00088
00089 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00090 {
00091     assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");
00092     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
from a different frame");
00093
00094     VkRenderPassBeginInfo renderPassInfo{};
00095     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00096     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00097     renderPassInfo.framebuffer = m_swapChain->getFrameBuffer(m_currentImageIndex);
00098
00099     renderPassInfo.renderArea.offset = {.x=0, .y=0};
00100     renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();
00101
00102     renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
00103     renderPassInfo.pClearValues = m_clearValues.data();
00104
00105     vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00106
00107     VkViewport viewport{};
00108     viewport.x = 0.0F;

```

```

00109     viewport.y = 0.0F;
00110     viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00111     viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00112     viewport.minDepth = 0.0F;
00113     viewport.maxDepth = 1.0F;
00114     const VkRect2D scissor{{0, 0}, m_swapChain->getSwapChainExtent()};
00115     vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00116     vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00117 }
00118
00119 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00120 {
00121     assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
00122     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
from a different frame");
00123
00124     vkCmdEndRenderPass(commandBuffer);
00125 }

```

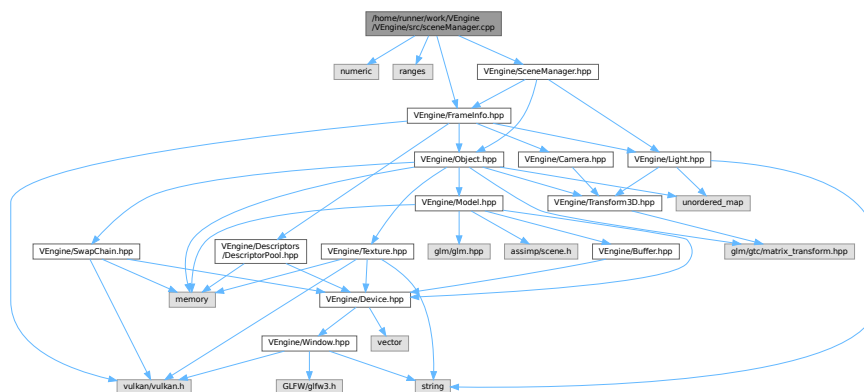
8.88 /home/runner/work/VEngine/VEngine/src/sceneManager.cpp File Reference

```

#include <numeric>
#include <ranges>
#include "VEngine/SceneManager.hpp"
#include "VEngine/FrameInfo.hpp"

```

Include dependency graph for sceneManager.cpp:



8.89 sceneManager.cpp

[Go to the documentation of this file.](#)

```

00001 #include <numeric>
00002 #include <ranges>
00003
00004 #include "VEngine/SceneManager.hpp"
00005 #include "VEngine/FrameInfo.hpp"
00006
00007 ven::SceneManager::SceneManager(Device& device)
00008 {
00009     // including nonCoherentAtomSize allows us to flush a specific index at once
00010     unsigned long alignment = std::lcm(
00011         device.getProperties().limits.nonCoherentAtomSize,
00012         device.getProperties().limits.minUniformBufferOffsetAlignment
00013     );
00014     for (auto & uboBuffer : m_uboBuffers) {
00015         uboBuffer = std::make_unique<Buffer>(
00016             device,
00017             sizeof(ObjectBufferData),

```

```

00018         MAX_OBJECTS,
00019         VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT,
00020         VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT,
00021         alignment);
00022     uboBuffer->map();
00023 }
00024     m_textureDefault = Texture::createTextureFromFile(device, "assets/textures/default.png");
00025 }
00026
00027 ven::Object& ven::SceneManager::createObject()
00028 {
00029     assert(m_currentObjId < MAX_OBJECTS && "Max object count exceeded!");
00030     Object object(m_currentObjId++);
00031     const unsigned int objId = object.getId();
00032     object.setDiffuseMap(m_textureDefault);
00033     m_objects.emplace(objId, std::move(object));
00034     return m_objects.at(objId);
00035 }
00036
00037 ven::Light& ven::SceneManager::createLight(const float radius, const glm::vec4 color)
00038 {
00039     assert(m_currentLightId < MAX_LIGHTS && "Max light count exceeded!");
00040     Light light(m_currentLightId++);
00041     const unsigned int lightId = light.getId();
00042     light.color = color;
00043     light.transform.scale.x = radius;
00044     m_lights.emplace(lightId, std::move(light));
00045     return m_lights.at(lightId);
00046 }
00047
00048 void ven::SceneManager::updateBuffer(GlobalUbo &ubo, const unsigned long frameIndex, const float
    frameTime)
00049 {
00050     uint8_t lightIndex = 0;
00051     const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameTime, {0.F, -1.F, 0.F});
00052
00053     for (auto& [id, object] : m_objects) {
00054         const ObjectBufferData data{
00055             .modelMatrix = object.transform.transformMatrix(),
00056             .normalMatrix = object.transform.normalMatrix()
00057         };
00058         m_uboBuffers.at(frameIndex)->writeToIndex(&data, id);
00059         object.setBufferInfo(static_cast<int>(frameIndex),
    m_uboBuffers.at(frameIndex)->descriptorInfoForIndex(id));
00060     }
00061
00062     for (Light &light : m_lights | std::views::values) {
00063         auto&[position, color, shininess, padding] = ubo.pointLights.at(lightIndex);
00064         light.transform.translation = glm::vec3(rotateLight * glm::vec4(light.transform.translation,
    light.transform.scale.x));
00065         position = glm::vec4(light.transform.translation, light.transform.scale.x);
00066         color = light.color;
00067         shininess = light.getShininess();
00068         lightIndex++;
00069     }
00070     ubo.numLights = lightIndex;
00071 }

```

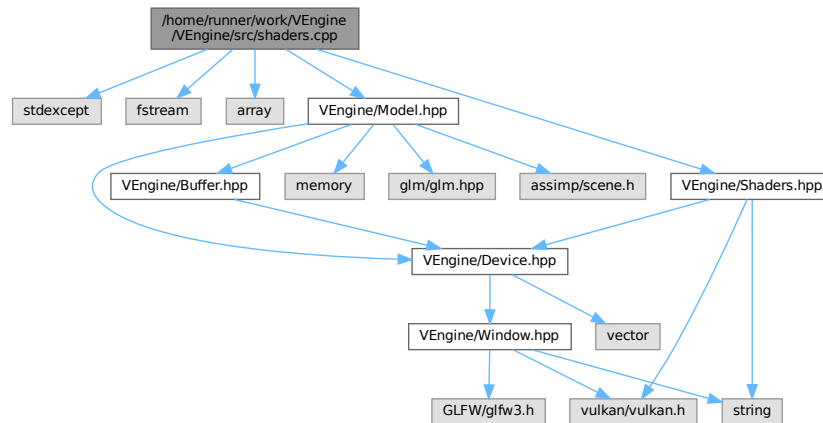
8.90 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```

#include <stdexcept>
#include <fstream>
#include <array>
#include "VEngine/Model.hpp"
#include "VEngine/Shaders.hpp"

```

Include dependency graph for shaders.cpp:



8.91 shaders.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002 #include <fstream>
00003 #include <array>
00004
00005 #include "VEngine/Model.hpp"
00006 #include "VEngine/Shaders.hpp"
00007
00008 ven::Shaders::~Shaders()
00009 {
00010     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00011     vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00012     vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
00013 }
00014
00015 std::vector<char> ven::Shaders::readFile(const std::string &filename) {
00016     std::ifstream file(filename, std::ios::binary | std::ios::ate);
00017     if (!file.is_open()) {
00018         throw std::runtime_error("failed to open file!");
00019     }
00020
00021     const long int fileSize = file.tellg();
00022     std::vector<char> buffer(static_cast<long unsigned int>(fileSize));
00023     file.seekg(0);
00024     file.read(buffer.data(), fileSize);
00025     return buffer;
00026 }
00027
00028 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo)
00029 {
00030     const std::vector<char> vertCode = readFile(vertFilepath);
00031     const std::vector<char> fragCode = readFile(fragFilepath);
00032
00033     createShaderModule(vertCode, &m_vertShaderModule);
00034     createShaderModule(fragCode, &m_fragShaderModule);
00035
00036     std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
00037     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00038     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00039     shaderStages[0].module = m_vertShaderModule;
00040     shaderStages[0].pName = "main";
00041     shaderStages[0].flags = 0;
00042     shaderStages[0].pNext = nullptr;
00043     shaderStages[0].pSpecializationInfo = nullptr;
00044
00045     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00046     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00047     shaderStages[1].module = m_fragShaderModule;

```



```

00048     shaderStages[1].pName = "main";
00049     shaderStages[1].flags = 0;
00050     shaderStages[1].pNext = nullptr;
00051     shaderStages[1].pSpecializationInfo = nullptr;
00052
00053     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00054     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00055     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00056     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00057     vertexInputInfo.vertexAttributeDescriptionCount =
static_cast<uint32_t>(attributeDescriptions.size());
00058     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00059     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00060     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00061
00062
00063     VkPipelineViewportStateCreateInfo viewportInfo{};
00064     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00065     viewportInfo.viewportCount = 1;
00066     viewportInfo.pViewports = nullptr;
00067     viewportInfo.scissorCount = 1;
00068     viewportInfo.pScissors = nullptr;
00069
00070
00071     VkGraphicsPipelineCreateInfo pipelineInfo{};
00072     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00073     pipelineInfo.stageCount = 2;
00074     pipelineInfo.pStages = shaderStages.data();
00075     pipelineInfo.pVertexInputState = &vertexInputInfo;
00076     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00077     pipelineInfo.pViewportState = &viewportInfo;
00078     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00079     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00080
00081     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00082     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00083     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00084
00085     pipelineInfo.layout = configInfo.pipelineLayout;
00086     pipelineInfo.renderPass = configInfo.renderPass;
00087     pipelineInfo.subpass = configInfo.subpass;
00088
00089     pipelineInfo.basePipelineIndex = -1;
00090     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00091
00092     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
&m_graphicsPipeline) != VK_SUCCESS) {
00093         throw std::runtime_error("failed to create graphics pipeline");
00094     }
00095 }
00096
00097 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
const
00098 {
00099     VkShaderModuleCreateInfo createInfo{};
00100     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00101     createInfo.codeSize = code.size();
00102     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00103
00104     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00105         throw std::runtime_error("failed to create shader module");
00106     }
00107 }
00108
00109 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00110 {
00111     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00112     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00113     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00114
00115     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00116     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00117     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00118     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00119     configInfo.rasterizationInfo.lineWidth = 1.0F;
00120     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE; // to enable later
(VK_CULL_MODE_BACK_BIT) back-face culling
00121     configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00122     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00123     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00124     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00125     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00126
00127     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00128     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00129     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00130     configInfo.multisampleInfo.minSampleShading = 1.0F;

```

```

00131     configInfo.multisampleInfo.pSampleMask = nullptr;
00132     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00133     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00134
00135     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00136     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00137     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00138     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00139     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00140     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00141     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00142     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00143
00144     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00145     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00146     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00147     configInfo.colorBlendInfo.attachmentCount = 1;
00148     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00149     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00150     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00151     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00152     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00153
00154     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00155     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00156     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00157     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00158     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00159     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00160     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00161     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00162     configInfo.depthStencilInfo.front = {};
00163     configInfo.depthStencilInfo.back = {};
00164
00165     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00166     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00167     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00168     configInfo.dynamicStateInfo.dynamicStateCount =
static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00169     configInfo.dynamicStateInfo.flags = 0;
00170     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();
00171     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00172 }

```

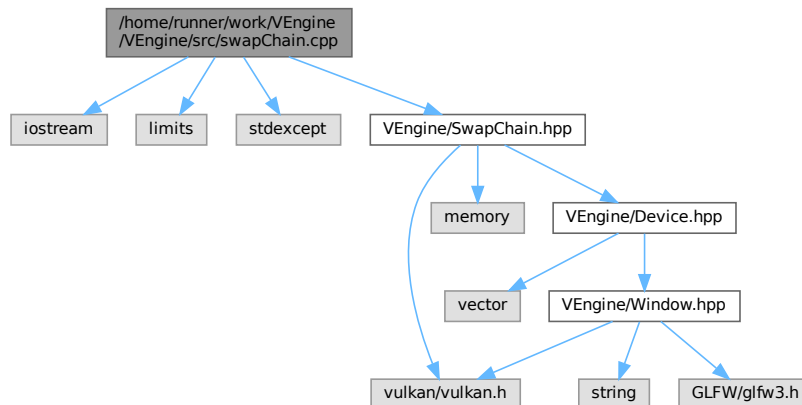
8.92 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference

```

#include <iostream>
#include <limits>
#include <stdexcept>
#include "VEngine/SwapChain.hpp"

```

Include dependency graph for swapChain.cpp:



8.93 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 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         {
00327             throw std::runtime_error("failed to create synchronization objects for a frame!");
00328         }
00329     }
00330 }
00331
00332 VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats)
00333 {
00334     for (const auto &availableFormat : availableFormats) {
00335         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00336             return availableFormat;
00337         }
00338     }
00339     return availableFormats[0];
00340 }
00341
00342 VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes)
00343 {
00344     for (const auto &availablePresentMode : availablePresentModes) {
00345         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00346             std::cout << "Present mode: Mailbox\n";
00347             return availablePresentMode;
00348         }
00349     }
00350
00351     for (const auto &availablePresentMode : availablePresentModes) {
00352         if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00353             std::cout << "Present mode: Immediate" << '\n';
00354             return availablePresentMode;
00355         }
00356     }
00357
00358     std::cout << "Present mode: V-Sync\n";
00359     return VK_PRESENT_MODE_FIFO_KHR;
00360 }
00361
00362 VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00363 {
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
00374
00375

```



```

00028         &objectDescriptorSet,
00029         0,
00030         nullptr);
00031
00032     const ObjectPushConstantData push{
00033         .modelMatrix = object.transform.transformMatrix(),
00034         .normalMatrix = object.transform.normalMatrix()
00035     };
00036     vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(ObjectPushConstantData), &push);
00037     object.getModel()->bind(frameInfo.commandBuffer);
00038     object.getModel()->draw(frameInfo.commandBuffer);
00039 }
00040 }

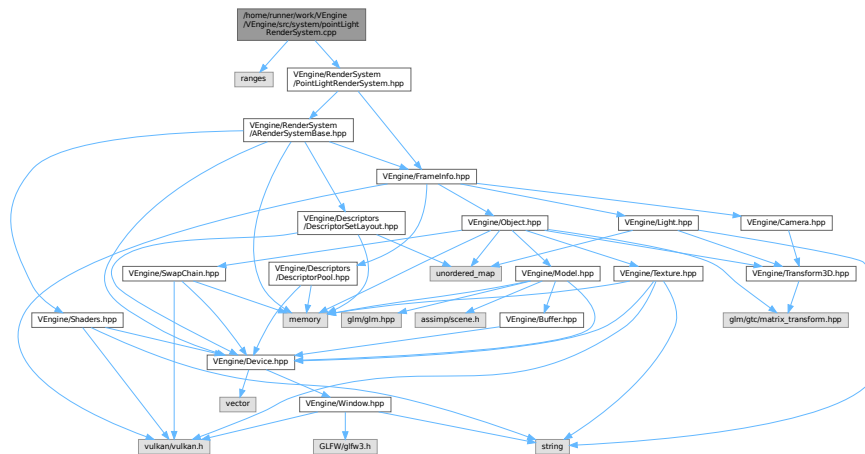
```

8.96 /home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp File Reference

```
#include <ranges>
```

```
#include "VEngine/RenderSystem/PointLightRenderSystem.hpp"
```

Include dependency graph for pointLightRenderSystem.cpp:



8.97 pointLightRenderSystem.cpp

[Go to the documentation of this file.](#)

```

00001 #include <ranges>
00002
00003 #include "VEngine/RenderSystem/PointLightRenderSystem.hpp"
00004
00005 void ven::PointLightRenderSystem::render(const FrameInfo &frameInfo) const
00006 {
00007     getShaders()->bind(frameInfo.commandBuffer);
00008     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00009
00010     for (const Light &light : frameInfo.lights | std::views::values) {
00011         const LightPushConstantData push{
00012             .position = glm::vec4(light.transform.translation, 1.F),
00013             .color = light.color,
00014             .radius = light.transform.scale.x
00015         };
00016         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
00017         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00018     }
00019 }

```



```

00039 {
00040     assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
00041     PipelineConfigInfo pipelineConfig{};
00042     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00043     if (isLight) {
00044         pipelineConfig.attributeDescriptions.clear();
00045         pipelineConfig.bindingDescriptions.clear();
00046     }
00047     pipelineConfig.renderPass = renderPass;
00048     pipelineConfig.pipelineLayout = m_pipelineLayout;
00049     m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
00050 }

```

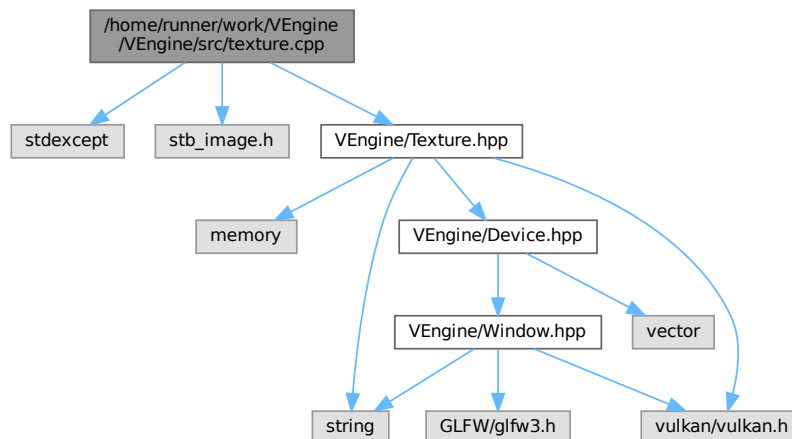
8.100 /home/runner/work/VEngine/VEngine/src/texture.cpp File Reference

```

#include <stdexcept>
#include <stb_image.h>
#include "VEngine/Texture.hpp"

```

Include dependency graph for texture.cpp:



Macros

- `#define STB_IMAGE_IMPLEMENTATION`

8.100.1 Macro Definition Documentation

8.100.1.1 STB_IMAGE_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

Definition at line 3 of file [texture.cpp](#).

8.101 texture.cpp

[Go to the documentation of this file.](#)

```
00001 #include <stdexcept>
00002
00003 #define STB_IMAGE_IMPLEMENTATION
00004 #include <stb_image.h>
00005
00006 #include "VEngine/Texture.hpp"
00007
00008 ven::Texture::Texture(Device &device, const std::string &textureFilepath) : m_device{device}
00009 {
00010     createTextureImage(textureFilepath);
00011     createTextureImageView(VK_IMAGE_VIEW_TYPE_2D);
00012     createTextureSampler();
00013     updateDescriptor();
00014 }
00015
00016 ven::Texture::Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00017     VkSampleCountFlagBits sampleCount)
00018 : m_device{device}, m_format(format), m_extent(extent)
00019 {
00020     VkImageAspectFlags aspectMask = 0;
00021     VkImageLayout imageLayout;
00022
00023     if ((usage & VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT) != 0u) {
00024         aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00025         imageLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00026     }
00027     if ((usage & VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT) != 0u) {
00028         aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00029         imageLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00030     }
00031
00032     // Don't like this, should I be using an image array instead of multiple images?
00033     VkImageCreateInfo imageInfo{};
00034     imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00035     imageInfo.imageType = VK_IMAGE_TYPE_2D;
00036     imageInfo.format = format;
00037     imageInfo.extent = extent;
00038     imageInfo.mipLevels = 1;
00039     imageInfo.arrayLayers = 1;
00040     imageInfo.samples = sampleCount;
00041     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00042     imageInfo.usage = usage;
00043     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00044     device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_textureImage,
00045         m_textureImageMemory);
00046
00047     VkImageViewCreateInfo viewInfo{};
00048     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00049     viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00050     viewInfo.format = format;
00051     viewInfo.subresourceRange = {};
00052     viewInfo.subresourceRange.aspectMask = aspectMask;
00053     viewInfo.subresourceRange.baseMipLevel = 0;
00054     viewInfo.subresourceRange.levelCount = 1;
00055     viewInfo.subresourceRange.baseArrayLayer = 0;
00056     viewInfo.subresourceRange.layerCount = 1;
00057     viewInfo.image = m_textureImage;
00058     if (vkCreateImageView(device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00059         throw std::runtime_error("failed to create texture image view!");
00060     }
00061
00062     // Sampler should be seperated out
00063     if ((usage & VK_IMAGE_USAGE_SAMPLED_BIT) != 0U) {
00064         // Create sampler from the attachment in the fragment shader
00065         VkSamplerCreateInfo samplerInfo{};
00066         samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00067         samplerInfo.magFilter = VK_FILTER_LINEAR;
00068         samplerInfo.minFilter = VK_FILTER_LINEAR;
00069         samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00070         samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER;
00071         samplerInfo.addressModeV = samplerInfo.addressModeU;
00072         samplerInfo.addressModeW = samplerInfo.addressModeU;
00073         samplerInfo.mipLodBias = 0.0F;
00074         samplerInfo.maxAnisotropy = 1.0F;
00075         samplerInfo.minLod = 0.0F;
00076         samplerInfo.maxLod = 1.0F;
00077         samplerInfo.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
00078
00079         if (vkCreateSampler(device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS)
00080         {
00081             throw std::runtime_error("failed to create sampler!");
00082         }
00083     }
00084 }
```

```

00080
00081     VkImageLayout samplerImageLayout = imageLayout == VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
00082         ? VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
00083         : VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL;
00084     m_descriptor.sampler = m_textureSampler;
00085     m_descriptor.imageView = m_textureImageView;
00086     m_descriptor.imageLayout = samplerImageLayout;
00087 }
00088 }
00089
00090 ven::Texture::~Texture()
00091 {
00092     vkDestroySampler(m_device.device(), m_textureSampler, nullptr);
00093     vkDestroyImageView(m_device.device(), m_textureImageView, nullptr);
00094     vkDestroyImage(m_device.device(), m_textureImage, nullptr);
00095     vkFreeMemory(m_device.device(), m_textureImageMemory, nullptr);
00096 }
00097
00098 void ven::Texture::updateDescriptor()
00099 {
00100     m_descriptor.sampler = m_textureSampler;
00101     m_descriptor.imageView = m_textureImageView;
00102     m_descriptor.imageLayout = m_textureLayout;
00103 }
00104
00105 void ven::Texture::createTextureImage(const std::string &filepath)
00106 {
00107     int texWidth = 0;
00108     int texHeight = 0;
00109     int texChannels = 0;
00110     void *data = nullptr;
00111     stbi_uc *pixels = nullptr;
00112
00113     stbi_set_flip_vertically_on_load(1);
00114     pixels = stbi_load(filepath.c_str(), &texWidth, &texHeight, &texChannels, STBI_rgb_alpha);
00115     const auto imageSize = static_cast<VkDeviceSize>(texWidth * texHeight * 4);
00116
00117     if (pixels == nullptr) {
00118         throw std::runtime_error("failed to load texture image!");
00119     }
00120
00121     // mMipLevels = static_cast<uint32_t>(std::floor(std::log2(std::max(texWidth, texHeight)))) + 1;
00122     m_mipLevels = 1;
00123
00124     VkBuffer stagingBuffer = nullptr;
00125     VkDeviceMemory stagingBufferMemory = nullptr;
00126
00127     m_device.createBuffer(
00128         imageSize,
00129         VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
00130         VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT,
00131         stagingBuffer,
00132         stagingBufferMemory);
00133
00134     vkMapMemory(m_device.device(), stagingBufferMemory, 0, imageSize, 0, &data);
00135     memcpy(data, pixels, imageSize);
00136     vkUnmapMemory(m_device.device(), stagingBufferMemory);
00137
00138     stbi_image_free(pixels);
00139
00140     m_format = VK_FORMAT_R8G8B8A8_SRGB;
00141     m_extent = {.width=static_cast<uint32_t>(texWidth), .height=static_cast<uint32_t>(texHeight),
00142         .depth=1};
00143
00144     VkImageCreateInfo imageInfo{};
00145     imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00146     imageInfo.imageType = VK_IMAGE_TYPE_2D;
00147     imageInfo.extent = m_extent;
00148     imageInfo.mipLevels = m_mipLevels;
00149     imageInfo.arrayLayers = m_layerCount;
00150     imageInfo.format = m_format;
00151     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00152     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00153     imageInfo.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT |
00154         VK_IMAGE_USAGE_SAMPLED_BIT;
00155     imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00156     imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00157
00158     m_device.createImageWithInfo(
00159         imageInfo,
00160         VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
00161         m_textureImage,
00162         m_textureImageMemory);
00163     m_device.transitionImageLayout(
00164         m_textureImage,
00165         VK_FORMAT_R8G8B8A8_SRGB,
00166         VK_IMAGE_LAYOUT_UNDEFINED,

```

```

00165         VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00166         m_mipLevels,
00167         m_layerCount);
00168     m_device.copyBufferToImage(
00169         stagingBuffer,
00170         m_textureImage,
00171         static_cast<uint32_t>(texWidth),
00172         static_cast<uint32_t>(texHeight),
00173         m_layerCount);
00174
00175     // comment this out if using mips
00176     m_device.transitionImageLayout(
00177         m_textureImage,
00178         VK_FORMAT_R8G8B8A8_SRGB,
00179         VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00180         VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL,
00181         m_mipLevels,
00182         m_layerCount);
00183
00184     // If we generate mip maps then the final image will already be READ_ONLY_OPTIMAL
00185     // mDevice.generateMipmaps(mTextureImage, mFormat, texWidth, texHeight, mMipLevels);
00186     m_textureLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
00187
00188     vkDestroyBuffer(m_device.device(), stagingBuffer, nullptr);
00189     vkFreeMemory(m_device.device(), stagingBufferMemory, nullptr);
00190 }
00191
00192 void ven::Texture::createTextureImageView(const VkImageViewType viewType)
00193 {
00194     VkImageViewCreateInfo viewInfo{};
00195     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00196     viewInfo.image = m_textureImage;
00197     viewInfo.viewType = viewType;
00198     viewInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
00199     viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00200     viewInfo.subresourceRange.baseMipLevel = 0;
00201     viewInfo.subresourceRange.levelCount = m_mipLevels;
00202     viewInfo.subresourceRange.baseArrayLayer = 0;
00203     viewInfo.subresourceRange.layerCount = m_layerCount;
00204
00205     if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00206         throw std::runtime_error("failed to create texture image view!");
00207     }
00208 }
00209
00210 void ven::Texture::createTextureSampler()
00211 {
00212     VkSamplerCreateInfo samplerInfo{};
00213     samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00214     samplerInfo.magFilter = VK_FILTER_LINEAR;
00215     samplerInfo.minFilter = VK_FILTER_LINEAR;
00216
00217     samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00218     samplerInfo.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00219     samplerInfo.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00220
00221     samplerInfo.anisotropyEnable = VK_TRUE;
00222     samplerInfo.maxAnisotropy = 16.0F;
00223     samplerInfo.borderColor = VK_BORDER_COLOR_INT_OPAQUE_BLACK;
00224     samplerInfo.unnormalizedCoordinates = VK_FALSE;
00225
00226     // these fields useful for percentage close filtering for shadow maps
00227     samplerInfo.compareEnable = VK_FALSE;
00228     samplerInfo.compareOp = VK_COMPARE_OP_ALWAYS;
00229
00230     samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00231     samplerInfo.mipLodBias = 0.0F;
00232     samplerInfo.minLod = 0.0F;
00233     samplerInfo.maxLod = static_cast<float>(m_mipLevels);
00234
00235     if (vkCreateSampler(m_device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS) {
00236         throw std::runtime_error("failed to create texture sampler!");
00237     }
00238 }
00239
00240 void ven::Texture::transitionLayout(const VkCommandBuffer commandBuffer, const VkImageLayout
oldLayout, const VkImageLayout newLayout) const
00241 {
00242     VkPipelineStageFlags sourceStage = 0;
00243     VkPipelineStageFlags destinationStage = 0;
00244     VkImageMemoryBarrier barrier{};
00245
00246     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00247     barrier.oldLayout = oldLayout;
00248     barrier.newLayout = newLayout;
00249
00250     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;

```

```

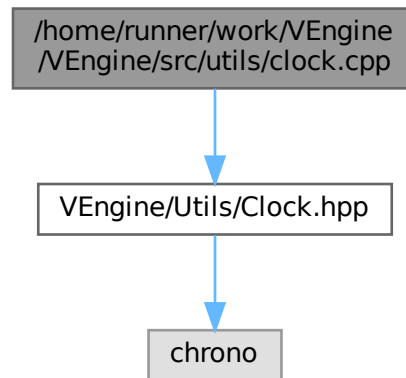
00251     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00252
00253     barrier.image = m_textureImage;
00254     barrier.subresourceRange.baseMipLevel = 0;
00255     barrier.subresourceRange.levelCount = m_mipLevels;
00256     barrier.subresourceRange.baseArrayLayer = 0;
00257     barrier.subresourceRange.layerCount = m_layerCount;
00258
00259     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00260         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00261         if (m_format == VK_FORMAT_D32_SFLOAT_S8_UINT || m_format == VK_FORMAT_D24_UNORM_S8_UINT) {
00262             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00263         }
00264     } else {
00265         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00266     }
00267     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00268         barrier.srcAccessMask = 0;
00269         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00270         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00271         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00272     } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00273         barrier.srcAccessMask = 0;
00274         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00275         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00276         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00277     } else if (oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL && newLayout ==
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00278         barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00279         barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00280
00281         sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00282         destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00283     } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00284         barrier.srcAccessMask = 0;
00285         barrier.dstAccessMask = VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT |
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00286         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00287         destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00288     } else if (oldLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL && newLayout ==
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL) {
00289         // This says that any cmd that acts in color output or after (dstStage)
00290         // that needs read or write access to a resource
00291         // must wait until all previous read accesses in fragment shader
00292         barrier.srcAccessMask = VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_SHADER_WRITE_BIT;
00293         barrier.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
00294         sourceStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00295         destinationStage = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
00296     } else {
00297         throw std::invalid_argument("unsupported layout transition!");
00298     }
00299     vkCmdPipelineBarrier(commandBuffer, sourceStage, destinationStage, 0, 0, nullptr, 0, nullptr, 1,
&barrier);
00300 }

```

8.102 /home/runner/work/VEngine/VEngine/src/utils/clock.cpp File Reference

```
#include "VEngine/Utils/Clock.hpp"
```

Include dependency graph for clock.cpp:



8.103 clock.cpp

[Go to the documentation of this file.](#)

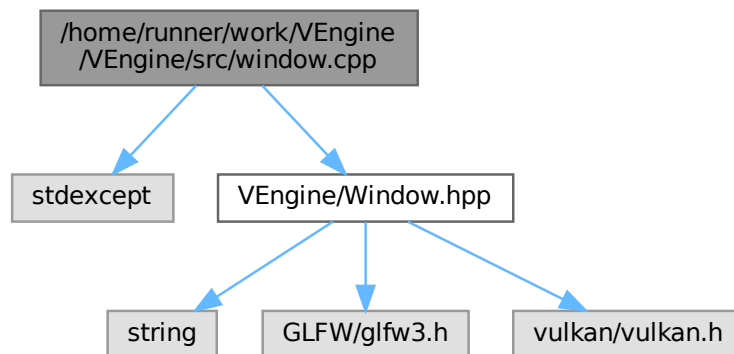
```

00001 #include "VEngine/Utils/Clock.hpp"
00002
00003 void ven::Clock::update()
00004 {
00005     auto newTime = std::chrono::high_resolution_clock::now();
00006     m_deltaTime = newTime - m_startTime;
00007     m_startTime = newTime;
00008 }
00009
00010 void ven::Clock::stop()
00011 {
00012     if (m_isStopped) {
00013         return;
00014     }
00015
00016     m_stopTime = std::chrono::high_resolution_clock::now();
00017     m_isStopped = true;
00018 }
00019
00020 void ven::Clock::resume()
00021 {
00022     if (!m_isStopped) {
00023         return;
00024     }
00025
00026     m_startTime += std::chrono::high_resolution_clock::now() - m_stopTime;
00027     m_isStopped = false;
00028 }
  
```

8.104 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

```
#include <stdexcept>
#include "VEngine/Window.hpp"
```


Include dependency graph for window.cpp:



8.105 window.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002
00003 #include "VEngine/Window.hpp"
00004
00005 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string
&title)
00006 {
00007     if (glfwInit() == GLFW_FALSE) {
00008         throw std::runtime_error("Failed to initialize GLFW");
00009     }
00010
00011     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
00012     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00013
00014     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
title.c_str(), nullptr, nullptr);
00015     if (window == nullptr) {
00016         glfwTerminate();
00017         throw std::runtime_error("Failed to create window");
00018     }
00019     glfwSetWindowUserPointer(window, this);
00020     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
00021     return window;
00022 }
00023
00024 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
00025 {
00026     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
00027         throw std::runtime_error("Failed to create window surface");
00028     }
00029 }
00030
00031 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
00032 {
00033     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
00034     app->m_framebufferResized = true;
00035     app->m_width = static_cast<uint32_t>(width);
00036     app->m_height = static_cast<uint32_t>(height);
00037 }
00038
00039 void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
00040 {
00041     GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
00042     const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);
00043
00044     /*
00045     if (fullscreen) {

```

```
00046         glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
mode->refreshRate);
00047     } else {
00048         // To restore a window that was originally windowed to its original size and position,
00049         // save these before making it full screen and then pass them in as above
00050         glfwSetWindowMonitor(m_window, nullptr, 0, 0, static_cast<int>(width),
static_cast<int>(height), mode->refreshRate);
00051
00052     }
00053
00054     m_width = width;
00055     m_height = height;
00056     */
00057 }
```

Index

[/home/runner/work/VEngine/VEngine/README.md,](#)
[289](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/fragment/pointLight.frag,](#)
[229](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/fragment/shader.frag,](#)
[229](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/vertex/pointLight.vert,](#)
[230](#)

[/home/runner/work/VEngine/VEngine/assets/shaders/vertex/shader.vert,](#)
[231](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp,](#)
[232, 233](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp,](#)
[235, 237](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp,](#)
[238, 239](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp,](#)
[240, 241](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp,](#)
[242, 243](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp,](#)
[244, 245](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp,](#)
[246, 248](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp,](#)
[249, 250](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Framerate.hpp,](#)
[251, 252](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp,](#)
[252, 254](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp,](#)
[255, 256](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp,](#)
[257, 258](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp,](#)
[259, 260](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/RenderSystemBase.hpp,](#)
[264, 265](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp,](#)
[265, 267](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp,](#)
[267, 268](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ShaderRenderSystem.hpp,](#)
[261, 263](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/SceneManager.hpp,](#)
[269, 270](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Shader.hpp,](#)
[271, 273](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp,](#)
[274, 275](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Texture.hpp,](#)
[277, 278](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Transform3D.hpp,](#)
[279, 280](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp,](#)
[280, 281](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp,](#)
[282, 283](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp,](#)
[285, 286](#)

[/home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp,](#)
[286, 288](#)

[/home/runner/work/VEngine/VEngine/src/buffer.cpp,](#)
[289](#)

[/home/runner/work/VEngine/VEngine/src/camera.cpp,](#)
[290, 291](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp,](#)
[293](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp,](#)
[294](#)

[/home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp,](#)
[295](#)

[/home/runner/work/VEngine/VEngine/src/device.cpp,](#)
[296, 299](#)

[/home/runner/work/VEngine/VEngine/src/engine.cpp,](#)
[306](#)

[/home/runner/work/VEngine/VEngine/src/eventManager.cpp,](#)
[309](#)

[/home/runner/work/VEngine/VEngine/src/gui/init.cpp,](#)
[310, 311](#)

[/home/runner/work/VEngine/VEngine/src/gui/render.cpp,](#)
[312](#)

[/home/runner/work/VEngine/VEngine/src/main.cpp,](#)
[317, 318](#)

[/home/runner/work/VEngine/VEngine/src/model.cpp,](#)
[318, 319](#)

[/home/runner/work/VEngine/VEngine/src/renderer.cpp,](#)
[322](#)

[/home/runner/work/VEngine/VEngine/src/sceneManager.cpp,](#)
[324](#)

[/home/runner/work/VEngine/VEngine/src/shaders.cpp,](#)
[325, 326](#)

[/home/runner/work/VEngine/VEngine/src/swapChain.cpp,](#)
[328, 329](#)

[/home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp,](#)
[334](#)

- /home/runner/work/VEngine/VEngine/src/system/pointLight.cpp, 335
- /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp, 336
- /home/runner/work/VEngine/VEngine/src/texture.cpp, 337, 338
- /home/runner/work/VEngine/VEngine/src/utils/clock.cpp, 342
- /home/runner/work/VEngine/VEngine/src/window.cpp, 342, 343
- ~ARenderSystemBase
 - ven::ARenderSystemBase, 23
- ~Buffer
 - ven::Buffer, 30
- ~Camera
 - ven::Camera, 53
- ~Clock
 - ven::Clock, 63
- ~DescriptorPool
 - ven::DescriptorPool, 80
- ~DescriptorSetLayout
 - ven::DescriptorSetLayout, 85
- ~DescriptorWriter
 - ven::DescriptorWriter, 89
- ~Device
 - ven::Device, 94
- ~Engine
 - ven::Engine, 112
- ~EventManager
 - ven::EventManager, 118
- ~Gui
 - ven::Gui, 128
- ~Light
 - ven::Light, 142
- ~Model
 - ven::Model, 148
- ~Renderer
 - ven::Renderer, 177
- ~Shaders
 - ven::Shaders, 192
- ~SwapChain
 - ven::SwapChain, 198
- ~Texture
 - ven::Texture, 211
- ~Window
 - ven::Window, 223
- acquireNextImage
 - ven::SwapChain, 199
- addBinding
 - ven::DescriptorSetLayout::Builder, 47
- addPoolSize
 - ven::DescriptorPool::Builder, 42
- allocateDescriptor
 - ven::DescriptorPool, 81
- ambientLightColor
 - ven::GlobalUbo, 126
- AQUA_3
 - ven::Colors, 68
- ARenderSystem
 - ven::Colors, 68
- ARenderSystemBase
 - ven::ARenderSystemBase, 23
- attributeDescriptions
 - ven::PipelineConfigInfo, 165
- beginFrame
 - ven::Renderer, 178
- beginSingleTimeCommands
 - ven::Device, 95
- beginSwapChainRenderPass
 - ven::Renderer, 178
- bind
 - ven::Model, 149
 - ven::Shaders, 193
- bindingDescriptions
 - ven::PipelineConfigInfo, 165
- BLACK_3
 - ven::Colors, 68
- BLACK_4
 - ven::Colors, 68
- BLACK_V
 - ven::Colors, 68
- BLUE_3
 - ven::Colors, 68
- BLUE_4
 - ven::Colors, 69
- BLUE_V
 - ven::Colors, 69
- Buffer
 - ven::Buffer, 30
- build
 - ven::DescriptorPool::Builder, 42
 - ven::DescriptorSetLayout::Builder, 47
 - ven::DescriptorWriter, 89
- Builder
 - ven::DescriptorPool::Builder, 42
 - ven::DescriptorSetLayout::Builder, 47
- Camera
 - ven::Camera, 53
- camera
 - ven::FrameInfo, 122
- cameraSection
 - ven::Gui, 129
- capabilities
 - ven::SwapChainSupportDetails, 207
- checkDeviceExtensionSupport
 - ven::Device, 95
- checkValidationLayerSupport
 - ven::Device, 95
- chooseSwapExtent
 - ven::SwapChain, 199
- chooseSwapPresentMode
 - ven::SwapChain, 199
- chooseSwapSurfaceFormat

- ven::SwapChain, 199
- cleanup
 - ven::Gui, 129
- Clock
 - ven::Clock, 63
- color
 - ven::Light, 144
 - ven::LightPushConstantData, 146
 - ven::Model::Vertex, 221
 - ven::PointLightData, 168
- COLOR_MAX
 - ven, 17
- COLOR_PRESETS_3
 - ven::Colors, 69
- COLOR_PRESETS_4
 - ven::Colors, 69
- COLOR_PRESETS_VK
 - ven::Colors, 70
- colorBlendAttachment
 - ven::PipelineConfigInfo, 166
- colorBlendInfo
 - ven::PipelineConfigInfo, 166
- commandBuffer
 - ven::FrameInfo, 122
- compareSwapFormats
 - ven::SwapChain, 200
- copyBuffer
 - ven::Device, 95
- copyBufferToImage
 - ven::Device, 96
- createBuffer
 - ven::Device, 96
- createCommandBuffers
 - ven::Renderer, 178
- createCommandPool
 - ven::Device, 96
- CreateDebugUtilsMessengerEXT
 - device.cpp, 297
- createDepthResources
 - ven::SwapChain, 200
- createFrameBuffers
 - ven::SwapChain, 200
- createGraphicsPipeline
 - ven::Shaders, 193
- createImageViews
 - ven::SwapChain, 200
- createImageWithInfo
 - ven::Device, 97
- createIndexBuffer
 - ven::Model, 149
- createInstance
 - ven::Device, 97
 - ven::Engine, 112
- createLight
 - ven::SceneManager, 187
- createLogicalDevice
 - ven::Device, 98
- createModelFromFile
 - ven::Model, 149
- createObject
 - ven::SceneManager, 187
- createPipeline
 - ven::ARenderSystemBase, 24
- createPipelineLayout
 - ven::ARenderSystemBase, 24
- createRenderPass
 - ven::SwapChain, 200
- createShaderModule
 - ven::Shaders, 193
- createSurface
 - ven::Device, 98
 - ven::Engine, 112
- createSwapChain
 - ven::SwapChain, 200
- createSyncObjects
 - ven::SwapChain, 200
- createTextureFromFile
 - ven::Texture, 211
- createTextureImage
 - ven::Texture, 211
- createTextureImageView
 - ven::Texture, 212
- createTextureSampler
 - ven::Texture, 212
- createVertexBuffer
 - ven::Model, 150
- createWindow
 - ven::Window, 224
- createWindowSurface
 - ven::Window, 224
- CYAN_3
 - ven::Colors, 70
- CYAN_4
 - ven::Colors, 70
- CYAN_V
 - ven::Colors, 71
- debugCallback
 - device.cpp, 297
- DEFAULT_AMBIENT_LIGHT_COLOR
 - ven, 17
- DEFAULT_AMBIENT_LIGHT_INTENSITY
 - ven, 17
- DEFAULT_CLEAR_COLOR
 - ven, 17
- DEFAULT_CLEAR_DEPTH
 - ven, 17
- DEFAULT_FAR
 - ven, 17
- DEFAULT_FOV
 - ven, 17
- DEFAULT_HEIGHT
 - ven, 18
- DEFAULT_KEY_MAPPINGS
 - ven, 18
- DEFAULT_LIGHT_COLOR
 - ven, 18

DEFAULT_LIGHT_INTENSITY
 ven, [18](#)
 DEFAULT_LIGHT_RADIUS
 ven, [18](#)
 DEFAULT_LOOK_SPEED
 ven, [18](#)
 DEFAULT_MAX_SETS
 ven, [18](#)
 DEFAULT_MOVE_SPEED
 ven, [19](#)
 DEFAULT_NEAR
 ven, [19](#)
 DEFAULT_POSITION
 ven, [19](#)
 DEFAULT_ROTATION
 ven, [19](#)
 DEFAULT_SHININESS
 ven, [19](#)
 DEFAULT_TITLE
 ven, [19](#)
 DEFAULT_WIDTH
 ven, [20](#)
 defaultPipelineConfigInfo
 ven::Shaders, [193](#)
 depthStencilInfo
 ven::PipelineConfigInfo, [166](#)
 DESCRIPTOR_COUNT
 ven, [20](#)
 descriptorInfo
 ven::Buffer, [30](#)
 descriptorInfoForIndex
 ven::Buffer, [31](#)
 DescriptorPool
 ven::DescriptorPool, [80](#), [81](#)
 DescriptorSetLayout
 ven::DescriptorSetLayout, [85](#), [86](#)
 DescriptorWriter
 ven::DescriptorPool, [82](#)
 ven::DescriptorSetLayout, [86](#)
 ven::DescriptorWriter, [89](#)
 DestroyDebugUtilsMessengerEXT
 device.cpp, [298](#)
 Device
 ven::Device, [94](#), [95](#)
 device
 ven::Device, [99](#)
 device.cpp
 CreateDebugUtilsMessengerEXT, [297](#)
 debugCallback, [297](#)
 DestroyDebugUtilsMessengerEXT, [298](#)
 devicePropertiesSection
 ven::Gui, [130](#)
 dir
 ven::KeyAction, [137](#)
 draw
 ven::Model, [151](#)
 dynamicStateEnables
 ven::PipelineConfigInfo, [166](#)
 dynamicStateInfo
 ven::PipelineConfigInfo, [166](#)
 EDITOR
 ven, [15](#)
 enableValidationLayers
 ven::Device, [107](#)
 endFrame
 ven::Renderer, [178](#)
 endSingleTimeCommands
 ven::Device, [100](#)
 endSwapChainRenderPass
 ven::Renderer, [179](#)
 Engine
 ven::Engine, [111](#), [112](#)
 ENGINE_STATE
 ven, [15](#)
 EPSILON
 ven, [20](#)
 EventManager
 ven::EventManager, [118](#)
 eventManager.cpp
 GLM_ENABLE_EXPERIMENTAL, [309](#)
 EXIT
 ven, [15](#)
 extentAspectRatio
 ven::SwapChain, [201](#)
 findDepthFormat
 ven::SwapChain, [201](#)
 findMemoryType
 ven::Device, [100](#)
 findPhysicalQueueFamilies
 ven::Device, [100](#)
 findQueueFamilies
 ven::Device, [101](#)
 findSupportedFormat
 ven::Device, [101](#)
 flush
 ven::Buffer, [31](#)
 flushIndex
 ven::Buffer, [32](#)
 formats
 ven::SwapChainSupportDetails, [207](#)
 framebufferResizeCallback
 ven::Window, [224](#)
 frameDescriptorPool
 ven::FrameInfo, [122](#)
 frameIndex
 ven::FrameInfo, [122](#)
 framePools
 ven::Engine, [115](#)
 frameTime
 ven::FrameInfo, [122](#)
 freeCommandBuffers
 ven::Renderer, [179](#)
 freeDescriptors
 ven::DescriptorPool, [81](#)
 FUCHSIA_3

- ven::Colors, 71
- FUCHSIA_4
 - ven::Colors, 71
- FUCHSIA_V
 - ven::Colors, 71
- GAME
 - ven, 15
- getAlignment
 - ven::Buffer, 33
- getAlignmentSize
 - ven::Buffer, 33
- getAspectRatio
 - ven::Renderer, 179
- getAttributeDescriptions
 - ven::Model::Vertex, 220
- getBindingDescriptions
 - ven::Model::Vertex, 220
- getBuffer
 - ven::Buffer, 33
- getBufferInfo
 - ven::Object, 155
- getBufferInfoForObject
 - ven::SceneManager, 187
- getBufferSize
 - ven::Buffer, 34
- getClearColor
 - ven::Renderer, 179
- getCommandPool
 - ven::Device, 102
- getCurrentCommandBuffer
 - ven::Renderer, 180
- getDeltaTime
 - ven::Clock, 63
- getDescriptorPool
 - ven::DescriptorPool, 81
- getDescriptorSetLayout
 - ven::DescriptorSetLayout, 86
- getDevice
 - ven::ARenderSystemBase, 25
- getDiffuseMap
 - ven::Object, 155
- getExtent
 - ven::Texture, 213
 - ven::Window, 225
- getFar
 - ven::Camera, 54
- getFormat
 - ven::Texture, 213
- getFov
 - ven::Camera, 54
- getFrameBuffer
 - ven::SwapChain, 201
- getFrameIndex
 - ven::Renderer, 180
- getGLFWWindow
 - ven::Window, 225
- getGraphicsQueue
 - ven::Device, 102
- getId
 - ven::Light, 143
 - ven::Object, 155
- getImage
 - ven::Texture, 213
- getImageInfo
 - ven::Texture, 213
- getImageLayout
 - ven::Texture, 213
- getImageView
 - ven::SwapChain, 201
 - ven::Texture, 213
- getInstanceCount
 - ven::Buffer, 34
- getInstanceSize
 - ven::Buffer, 34
- getInverseView
 - ven::Camera, 54
- getLights
 - ven::SceneManager, 187
- getLookSpeed
 - ven::Camera, 55
- getMappedMemory
 - ven::Buffer, 34
- getMemoryPropertyFlags
 - ven::Buffer, 34
- getModel
 - ven::Object, 155
- getMoveSpeed
 - ven::Camera, 55
- getName
 - ven::Light, 143
 - ven::Object, 155
- getNear
 - ven::Camera, 55
- getObjects
 - ven::SceneManager, 188
- getPhysicalDevice
 - ven::Device, 102
- getPipelineLayout
 - ven::ARenderSystemBase, 25
- getProjection
 - ven::Camera, 56
- getProperties
 - ven::Device, 102
- getRenderPass
 - ven::SwapChain, 201
- getRequiredExtensions
 - ven::Device, 103
- getShaders
 - ven::ARenderSystemBase, 26
- getShininess
 - ven::Light, 143
- getState
 - ven::Gui, 130
- getSwapChainExtent
 - ven::SwapChain, 201
- getSwapChainImageFormat

- ven::SwapChain, 202
- getSwapChainRenderPass
 - ven::Renderer, 181
- getSwapChainSupport
 - ven::Device, 103
- getUboBuffers
 - ven::SceneManager, 188
- getUsageFlags
 - ven::Buffer, 34
- getView
 - ven::Camera, 56
- getWindow
 - ven::Renderer, 181
- GLFW_INCLUDE_VULKAN
 - Window.hpp, 288
- GLM_ENABLE_EXPERIMENTAL
 - eventManager.cpp, 309
 - model.cpp, 319
- globalDescriptorSet
 - ven::FrameInfo, 122
- graphicsFamily
 - ven::QueueFamilyIndices, 174
- graphicsFamilyHasValue
 - ven::QueueFamilyIndices, 174
- graphicsQueue
 - ven::Device, 103
- GRAY_3
 - ven::Colors, 71
- GRAY_4
 - ven::Colors, 71
- GRAY_V
 - ven::Colors, 71
- GREEN_3
 - ven::Colors, 72
- GREEN_4
 - ven::Colors, 72
- GREEN_V
 - ven::Colors, 72
- Gui
 - ven::Gui, 128
- GUI_STATE
 - ven, 15
- handleEvents
 - ven::EventManager, 118
- hasGlfwRequiredInstanceExtensions
 - ven::Device, 104
- hashCombine
 - ven, 16
- height
 - ven::SwapChain, 202
- HIDDEN
 - ven, 16
- imageCount
 - ven::SwapChain, 202
- imageView
 - ven::Texture, 214
- indices
 - ven::Model::Builder, 50
- init
 - ven::Gui, 130
 - ven::SwapChain, 202
- initStyle
 - ven::Gui, 131
- inputAssemblyInfo
 - ven::PipelineConfigInfo, 166
- inputsSection
 - ven::Gui, 132
- invalidate
 - ven::Buffer, 35
- invalidateIndex
 - ven::Buffer, 35
- inverseView
 - ven::GlobalUbo, 126
- isComplete
 - ven::QueueFamilyIndices, 173
- isDeviceSuitable
 - ven::Device, 104
- isFrameInProgress
 - ven::Renderer, 181
- isKeyJustPressed
 - ven::EventManager, 118
- IsLegacyNativeDupe
 - ven::Gui::funcs, 124
- key
 - ven::KeyAction, 137
- Light
 - ven::Light, 142, 143
- lights
 - ven::FrameInfo, 123
- lightsSection
 - ven::Gui, 132
- LIME_3
 - ven::Colors, 72
- LIME_4
 - ven::Colors, 72
- LIME_V
 - ven::Colors, 72
- loadModel
 - ven::Model::Builder, 50
- loadObjects
 - ven::Engine, 113
- lookDown
 - ven::KeyMappings, 139
- lookLeft
 - ven::KeyMappings, 139
- lookRight
 - ven::KeyMappings, 139
- lookUp
 - ven::KeyMappings, 139
- m_alignmentSize
 - ven::Buffer, 38
- m_bindings
 - ven::DescriptorSetLayout, 87

- ven::DescriptorSetLayout::Builder, 48
- m_buffer
 - ven::Buffer, 38
- m_bufferInfo
 - ven::Object, 157
- m_bufferSize
 - ven::Buffer, 38
- m_clearValues
 - ven::Renderer, 183
- m_commandBuffers
 - ven::Renderer, 183
- m_commandPool
 - ven::Device, 107
- m_currentFrame
 - ven::SwapChain, 203
- m_currentFrameIndex
 - ven::Renderer, 183
- m_currentImageIndex
 - ven::Renderer, 183
- m_currentLightId
 - ven::SceneManager, 189
- m_currentObjId
 - ven::SceneManager, 189
- m_debugMessenger
 - ven::Device, 107
- m_deltaTime
 - ven::Clock, 65
- m_depthImageMemory
 - ven::SwapChain, 203
- m_depthImages
 - ven::SwapChain, 203
- m_depthImageViews
 - ven::SwapChain, 203
- m_descriptor
 - ven::Texture, 215
- m_descriptorPool
 - ven::DescriptorPool, 82
- m_descriptorSetLayout
 - ven::DescriptorSetLayout, 87
- m_device
 - ven::ARenderSystemBase, 26
 - ven::Buffer, 38
 - ven::DescriptorPool, 82
 - ven::DescriptorPool::Builder, 44
 - ven::DescriptorSetLayout, 87
 - ven::DescriptorSetLayout::Builder, 48
 - ven::Device, 107
 - ven::Engine, 115
 - ven::Model, 151
 - ven::Renderer, 184
 - ven::Shaders, 195
 - ven::SwapChain, 204
 - ven::Texture, 215
- m_deviceExtensions
 - ven::Device, 107
- m_diffuseMap
 - ven::Object, 157
- m_extent
 - ven::Texture, 215
- m_far
 - ven::Camera, 60
- m_format
 - ven::Texture, 215
- m_fov
 - ven::Camera, 60
- m_fragShaderModule
 - ven::Shaders, 195
- m_framebufferResized
 - ven::Window, 227
- m_globalPool
 - ven::Engine, 115
- m_graphicsPipeline
 - ven::Shaders, 195
- m_graphicsQueue
 - ven::Device, 108
- m_gui
 - ven::Engine, 115
- m_hasIndexBuffer
 - ven::Model, 151
- m_height
 - ven::Window, 227
- m_imageAvailableSemaphores
 - ven::SwapChain, 204
- m_imagesInFlight
 - ven::SwapChain, 204
- m_indexBuffer
 - ven::Model, 151
- m_indexCount
 - ven::Model, 151
- m_inFlightFences
 - ven::SwapChain, 204
- m_instance
 - ven::Device, 108
 - ven::Engine, 115
- m_instanceCount
 - ven::Buffer, 39
- m_instanceSize
 - ven::Buffer, 39
- m_intensity
 - ven::Gui, 135
- m_inverseViewMatrix
 - ven::Camera, 60
- m_io
 - ven::Gui, 135
- m_isFrameStarted
 - ven::Renderer, 184
- m_isStopped
 - ven::Clock, 65
- m_keyState
 - ven::EventManager, 120
- m_layerCount
 - ven::Texture, 215
- m_lightId
 - ven::Light, 144
- m_lights
 - ven::SceneManager, 189

- m_lookSpeed
 - ven::Camera, 60
- m_mapped
 - ven::Buffer, 39
- m_maxSets
 - ven::DescriptorPool::Builder, 44
- m_memory
 - ven::Buffer, 39
- m_memoryPropertyFlags
 - ven::Buffer, 39
- m_mipLevels
 - ven::Texture, 215
- m_model
 - ven::Object, 157
- m_moveSpeed
 - ven::Camera, 60
- m_name
 - ven::Light, 145
 - ven::Object, 157
- m_near
 - ven::Camera, 61
- m_objects
 - ven::SceneManager, 189
- m_objId
 - ven::Object, 157
- m_oldSwapChain
 - ven::SwapChain, 204
- m_physicalDevice
 - ven::Device, 108
- m_pipelineLayout
 - ven::ARenderSystemBase, 26
- m_pool
 - ven::DescriptorWriter, 91
- m_poolFlags
 - ven::DescriptorPool::Builder, 44
- m_poolSizes
 - ven::DescriptorPool::Builder, 44
- m_presentQueue
 - ven::Device, 108
- m_projectionMatrix
 - ven::Camera, 61
- m_properties
 - ven::Device, 108
- m_renderer
 - ven::Engine, 115
- m_renderFinishedSemaphores
 - ven::SwapChain, 204
- m_renderPass
 - ven::SwapChain, 205
- m_sceneManager
 - ven::Engine, 116
- m_setLayout
 - ven::DescriptorWriter, 91
- m_shaders
 - ven::ARenderSystemBase, 27
- m_shininess
 - ven::Gui, 135
 - ven::Light, 145
- m_startTime
 - ven::Clock, 65
- m_state
 - ven::Engine, 116
 - ven::Gui, 135
- m_stopTime
 - ven::Clock, 65
- m_surface
 - ven::Device, 108
 - ven::Engine, 116
- m_swapChain
 - ven::Renderer, 184
 - ven::SwapChain, 205
- m_swapChainDepthFormat
 - ven::SwapChain, 205
- m_swapChainExtent
 - ven::SwapChain, 205
- m_swapChainFrameBuffers
 - ven::SwapChain, 205
- m_swapChainImageFormat
 - ven::SwapChain, 205
- m_swapChainImages
 - ven::SwapChain, 206
- m_swapChainImageViews
 - ven::SwapChain, 206
- m_textureDefault
 - ven::SceneManager, 189
- m_textureImage
 - ven::Texture, 215
- m_textureImageMemory
 - ven::Texture, 216
- m_textureImageView
 - ven::Texture, 216
- m_textureLayout
 - ven::Texture, 216
- m_textureSampler
 - ven::Texture, 216
- m_uboBuffers
 - ven::SceneManager, 189
- m_usageFlags
 - ven::Buffer, 39
- m_validationLayers
 - ven::Device, 109
- m_vertexBuffer
 - ven::Model, 152
- m_vertexCount
 - ven::Model, 152
- m_vertShaderModule
 - ven::Shaders, 195
- m_viewMatrix
 - ven::Camera, 61
- m_width
 - ven::Window, 227
- m_window
 - ven::Device, 109
 - ven::Engine, 116
 - ven::Renderer, 184
 - ven::Window, 227

- m_windowExtent
 - ven::SwapChain, 206
- m_writes
 - ven::DescriptorWriter, 91
- MAGENTA_3
 - ven::Colors, 72
- MAGENTA_4
 - ven::Colors, 73
- MAGENTA_V
 - ven::Colors, 73
- main
 - main.cpp, 317
- main.cpp
 - main, 317
- mainLoop
 - ven::Engine, 114
- Map
 - ven::Light, 142
 - ven::Object, 154
- map
 - ven::Buffer, 36
- MAROON_3
 - ven::Colors, 73
- MAROON_4
 - ven::Colors, 73
- MAROON_V
 - ven::Colors, 73
- MAX_FRAMES_IN_FLIGHT
 - ven, 20
- MAX_LIGHTS
 - ven, 20
- MAX_OBJECTS
 - ven, 20
- Model
 - ven::Model, 148, 149
- model.cpp
 - GLM_ENABLE_EXPERIMENTAL, 319
- modelMatrix
 - ven::ObjectBufferData, 158
 - ven::ObjectPushConstantData, 159
- moveBackward
 - ven::KeyMappings, 139
- moveCamera
 - ven::EventManager, 119
- moveDown
 - ven::KeyMappings, 139
- moveForward
 - ven::KeyMappings, 139
- moveLeft
 - ven::KeyMappings, 140
- moveRight
 - ven::KeyMappings, 140
- moveUp
 - ven::KeyMappings, 140
- multisampleInfo
 - ven::PipelineConfigInfo, 167
- NAVY_3
 - ven::Colors, 73
- NAVY_4
 - ven::Colors, 73
- NAVY_V
 - ven::Colors, 74
- NIGHT_BLUE_3
 - ven::Colors, 74
- NIGHT_BLUE_4
 - ven::Colors, 74
- NIGHT_BLUE_V
 - ven::Colors, 74
- normal
 - ven::Model::Vertex, 221
- normalMatrix
 - ven::ObjectBufferData, 158
 - ven::ObjectPushConstantData, 159
 - ven::Transform3D, 218
- numLights
 - ven::GlobalUbo, 126
- Object
 - ven::Object, 154, 155
- ObjectRenderSystem
 - ven::ObjectRenderSystem, 162
- objects
 - ven::FrameInfo, 123
- objectsSection
 - ven::Gui, 132
- OLIVE_3
 - ven::Colors, 74
- OLIVE_4
 - ven::Colors, 74
- OLIVE_V
 - ven::Colors, 74
- operator()
 - std::hash< ven::Model::Vertex >, 136
- operator=
 - ven::Buffer, 36
 - ven::Camera, 56
 - ven::Clock, 63
 - ven::DescriptorPool, 82
 - ven::DescriptorSetLayout, 86
 - ven::DescriptorWriter, 89
 - ven::Device, 104
 - ven::Engine, 114
 - ven::EventManager, 119
 - ven::Gui, 132
 - ven::Light, 144
 - ven::Model, 151
 - ven::Object, 156
 - ven::ObjectRenderSystem, 163
 - ven::PipelineConfigInfo, 165
 - ven::PointLightRenderSystem, 172
 - ven::Renderer, 182
 - ven::SceneManager, 188
 - ven::Shaders, 194
 - ven::SwapChain, 202
 - ven::Texture, 214
 - ven::Window, 226
- operator==

- ven::Model::Vertex, 220
- overwrite
 - ven::DescriptorWriter, 90
- padding
 - ven::PointLightData, 168
- PAUSED
 - ven, 15
- pickPhysicalDevice
 - ven::Device, 104
- PipelineConfigInfo
 - ven::PipelineConfigInfo, 165
- pipelineLayout
 - ven::PipelineConfigInfo, 167
- PointLightRenderSystem
 - ven::PointLightRenderSystem, 171
- pointLights
 - ven::GlobalUbo, 126
- populateDebugMessengerCreateInfo
 - ven::Device, 105
- position
 - ven::LightPushConstantData, 146
 - ven::Model::Vertex, 221
 - ven::PointLightData, 168
- presentFamily
 - ven::QueueFamilyIndices, 174
- presentFamilyHasValue
 - ven::QueueFamilyIndices, 174
- presentModes
 - ven::SwapChainSupportDetails, 207
- presentQueue
 - ven::Device, 105
- processKeyActions
 - ven::EventManager, 119
- processMesh
 - ven::Model::Builder, 50
- processNode
 - ven::Model::Builder, 50
- projection
 - ven::GlobalUbo, 126
- querySwapChainSupport
 - ven::Device, 105
- radius
 - ven::LightPushConstantData, 146
- rasterizationInfo
 - ven::PipelineConfigInfo, 167
- readFile
 - ven::Shaders, 194
- recreateSwapChain
 - ven::Renderer, 182
- RED_3
 - ven::Colors, 75
- RED_4
 - ven::Colors, 75
- RED_V
 - ven::Colors, 75
- render
 - ven::ARenderSystemBase, 26
 - ven::Gui, 132
 - ven::ObjectRenderSystem, 163
 - ven::PointLightRenderSystem, 172
- Renderer
 - ven::Renderer, 177
- rendererSection
 - ven::Gui, 133
- renderFrameWindow
 - ven::Gui, 134
- renderPass
 - ven::PipelineConfigInfo, 167
- renderSystemLayout
 - ven::ARenderSystemBase, 27
- resetPool
 - ven::DescriptorPool, 82
- resetWindowResizedFlag
 - ven::Window, 226
- resume
 - ven::Clock, 64
- rotation
 - ven::Transform3D, 218
- sampler
 - ven::Texture, 214
- scale
 - ven::Transform3D, 218
- SceneManager
 - ven::SceneManager, 186, 187
- setBufferInfo
 - ven::Object, 156
- setClearColor
 - ven::Renderer, 182
- setDiffuseMap
 - ven::Object, 156
- setFar
 - ven::Camera, 56
- setFov
 - ven::Camera, 57
- setFullscreen
 - ven::Window, 226
- setLookSpeed
 - ven::Camera, 57
- setMaxSets
 - ven::DescriptorPool::Builder, 43
- setModel
 - ven::Object, 156
- setMoveSpeed
 - ven::Camera, 58
- setName
 - ven::Light, 144
 - ven::Object, 156
- setNear
 - ven::Camera, 58
- setOrthographicProjection
 - ven::Camera, 59
- setPerspectiveProjection
 - ven::Camera, 59
- setPoolFlags

- ven::DescriptorPool::Builder, 43
- setShininess
 - ven::Light, 144
- setState
 - ven::Gui, 134
- setupDebugMessenger
 - ven::Device, 106
- setViewDirection
 - ven::Camera, 59
- setViewTarget
 - ven::Camera, 59
- setViewXYZ
 - ven::Camera, 60
- Shaders
 - ven::Shaders, 192
- SHADERS_BIN_PATH
 - ven, 20
- shininess
 - ven::PointLightData, 169
- SILVER_3
 - ven::Colors, 75
- SILVER_4
 - ven::Colors, 75
- SILVER_V
 - ven::Colors, 75
- SKY_BLUE_3
 - ven::Colors, 75
- SKY_BLUE_4
 - ven::Colors, 76
- SKY_BLUE_V
 - ven::Colors, 76
- start
 - ven::Clock, 64
- STB_IMAGE_IMPLEMENTATION
 - texture.cpp, 337
- std::hash< ven::Model::Vertex >, 135
 - operator(), 136
- stop
 - ven::Clock, 64
- submitCommandBuffers
 - ven::SwapChain, 203
- subpass
 - ven::PipelineConfigInfo, 167
- SUNSET_3
 - ven::Colors, 76
- SUNSET_4
 - ven::Colors, 76
- SUNSET_V
 - ven::Colors, 76
- surface
 - ven::Device, 106
- SwapChain
 - ven::SwapChain, 198, 199
- TEAL_3
 - ven::Colors, 76
- TEAL_4
 - ven::Colors, 76
- TEAL_V
 - ven::Colors, 77
- Texture
 - ven::Texture, 210, 211
- texture.cpp
 - STB_IMAGE_IMPLEMENTATION, 337
- TimePoint
 - ven, 15
- toggleGui
 - ven::KeyMappings, 140
- transform
 - ven::Camera, 61
 - ven::Light, 145
 - ven::Object, 157
- transformMatrix
 - ven::Transform3D, 218
- transitionImageLayout
 - ven::Device, 107
- transitionLayout
 - ven::Texture, 214
- translation
 - ven::Transform3D, 219
- unmap
 - ven::Buffer, 37
- update
 - ven::Clock, 64
- updateBuffer
 - ven::SceneManager, 188
- updateDescriptor
 - ven::Texture, 214
- updateEngineState
 - ven::EventManager, 119
- uv
 - ven::Model::Vertex, 221
- value
 - ven::KeyAction, 137
- ven, 13
 - COLOR_MAX, 17
 - DEFAULT_AMBIENT_LIGHT_COLOR, 17
 - DEFAULT_AMBIENT_LIGHT_INTENSITY, 17
 - DEFAULT_CLEAR_COLOR, 17
 - DEFAULT_CLEAR_DEPTH, 17
 - DEFAULT_FAR, 17
 - DEFAULT_FOV, 17
 - DEFAULT_HEIGHT, 18
 - DEFAULT_KEY_MAPPINGS, 18
 - DEFAULT_LIGHT_COLOR, 18
 - DEFAULT_LIGHT_INTENSITY, 18
 - DEFAULT_LIGHT_RADIUS, 18
 - DEFAULT_LOOK_SPEED, 18
 - DEFAULT_MAX_SETS, 18
 - DEFAULT_MOVE_SPEED, 19
 - DEFAULT_NEAR, 19
 - DEFAULT_POSITION, 19
 - DEFAULT_ROTATION, 19
 - DEFAULT_SHININESS, 19
 - DEFAULT_TITLE, 19
 - DEFAULT_WIDTH, 20

- DESCRIPTOR_COUNT, 20
- EDITOR, 15
- ENGINE_STATE, 15
- EPSILON, 20
- EXIT, 15
- GAME, 15
- GUI_STATE, 15
- hashCombine, 16
- HIDDEN, 16
- MAX_FRAMES_IN_FLIGHT, 20
- MAX_LIGHTS, 20
- MAX_OBJECTS, 20
- PAUSED, 15
- SHADERS_BIN_PATH, 20
- TimePoint, 15
- VISIBLE, 16
- ven::ARenderSystemBase, 21
 - ~ARenderSystemBase, 23
 - ARenderSystemBase, 23
 - createPipeline, 24
 - createPipelineLayout, 24
 - getDevice, 25
 - getPipelineLayout, 25
 - getShaders, 26
 - m_device, 26
 - m_pipelineLayout, 26
 - m_shaders, 27
 - render, 26
 - renderSystemLayout, 27
- ven::Buffer, 27
 - ~Buffer, 30
 - Buffer, 30
 - descriptorInfo, 30
 - descriptorInfoForIndex, 31
 - flush, 31
 - flushIndex, 32
 - getAlignment, 33
 - getAlignmentSize, 33
 - getBuffer, 33
 - getBufferSize, 34
 - getInstanceCount, 34
 - getInstanceSize, 34
 - getMappedMemory, 34
 - getMemoryPropertyFlags, 34
 - getUsageFlags, 34
 - invalidate, 35
 - invalidateIndex, 35
 - m_alignmentSize, 35
 - m_buffer, 38
 - m_bufferSize, 38
 - m_device, 38
 - m_instanceCount, 39
 - m_instanceSize, 39
 - m_mapped, 39
 - m_memory, 39
 - m_memoryPropertyFlags, 39
 - m_usageFlags, 39
 - map, 36
 - operator=, 36
 - unmap, 37
 - writeToBuffer, 37
 - writeToIndex, 37
- ven::Camera, 51
 - ~Camera, 53
 - Camera, 53
 - getFar, 54
 - getFov, 54
 - getInverseView, 54
 - getLookSpeed, 55
 - getMoveSpeed, 55
 - getNear, 55
 - getProjection, 56
 - getView, 56
 - m_far, 60
 - m_fov, 60
 - m_inverseViewMatrix, 60
 - m_lookSpeed, 60
 - m_moveSpeed, 60
 - m_near, 61
 - m_projectionMatrix, 61
 - m_viewMatrix, 61
 - operator=, 56
 - setFar, 56
 - setFov, 57
 - setLookSpeed, 57
 - setMoveSpeed, 58
 - setNear, 58
 - setOrthographicProjection, 59
 - setPerspectiveProjection, 59
 - setViewDirection, 59
 - setViewTarget, 59
 - setViewXYZ, 60
 - transform, 61
- ven::Clock, 62
 - ~Clock, 63
 - Clock, 63
 - getDeltaTime, 63
 - m_deltaTime, 65
 - m_isStopped, 65
 - m_startTime, 65
 - m_stopTime, 65
 - operator=, 63
 - resume, 64
 - start, 64
 - stop, 64
 - update, 64
- ven::Colors, 66
 - AQUA_3, 68
 - AQUA_4, 68
 - AQUA_V, 68
 - BLACK_3, 68
 - BLACK_4, 68
 - BLACK_V, 68
 - BLUE_3, 68
 - BLUE_4, 69
 - BLUE_V, 69

- COLOR_PRESETS_3, 69
- COLOR_PRESETS_4, 69
- COLOR_PRESETS_VK, 70
- CYAN_3, 70
- CYAN_4, 70
- CYAN_V, 71
- FUCHSIA_3, 71
- FUCHSIA_4, 71
- FUCHSIA_V, 71
- GRAY_3, 71
- GRAY_4, 71
- GRAY_V, 71
- GREEN_3, 72
- GREEN_4, 72
- GREEN_V, 72
- LIME_3, 72
- LIME_4, 72
- LIME_V, 72
- MAGENTA_3, 72
- MAGENTA_4, 73
- MAGENTA_V, 73
- MAROON_3, 73
- MAROON_4, 73
- MAROON_V, 73
- NAVY_3, 73
- NAVY_4, 73
- NAVY_V, 74
- NIGHT_BLUE_3, 74
- NIGHT_BLUE_4, 74
- NIGHT_BLUE_V, 74
- OLIVE_3, 74
- OLIVE_4, 74
- OLIVE_V, 74
- RED_3, 75
- RED_4, 75
- RED_V, 75
- SILVER_3, 75
- SILVER_4, 75
- SILVER_V, 75
- SKY_BLUE_3, 75
- SKY_BLUE_4, 76
- SKY_BLUE_V, 76
- SUNSET_3, 76
- SUNSET_4, 76
- SUNSET_V, 76
- TEAL_3, 76
- TEAL_4, 76
- TEAL_V, 77
- WHITE_3, 77
- WHITE_4, 77
- WHITE_V, 77
- YELLOW_3, 77
- YELLOW_4, 77
- YELLOW_V, 77
- ven::DescriptorPool, 78
 - ~DescriptorPool, 80
 - allocateDescriptor, 81
 - DescriptorPool, 80, 81
 - DescriptorWriter, 82
 - freeDescriptors, 81
 - getDescriptorPool, 81
 - m_descriptorPool, 82
 - m_device, 82
 - operator=, 82
 - resetPool, 82
- ven::DescriptorPool::Builder, 40
 - addPoolSize, 42
 - build, 42
 - Builder, 42
 - m_device, 44
 - m_maxSets, 44
 - m_poolFlags, 44
 - m_poolSizes, 44
 - setMaxSets, 43
 - setPoolFlags, 43
- ven::DescriptorSetLayout, 83
 - ~DescriptorSetLayout, 85
 - DescriptorSetLayout, 85, 86
 - DescriptorWriter, 86
 - getDescriptorSetLayout, 86
 - m_bindings, 87
 - m_descriptorSetLayout, 87
 - m_device, 87
 - operator=, 86
- ven::DescriptorSetLayout::Builder, 45
 - addBinding, 47
 - build, 47
 - Builder, 47
 - m_bindings, 48
 - m_device, 48
- ven::DescriptorWriter, 87
 - ~DescriptorWriter, 89
 - build, 89
 - DescriptorWriter, 89
 - m_pool, 91
 - m_setLayout, 91
 - m_writes, 91
 - operator=, 89
 - overwrite, 90
 - writeBuffer, 90
 - writImage, 90
- ven::Device, 91
 - ~Device, 94
 - beginSingleTimeCommands, 95
 - checkDeviceExtensionSupport, 95
 - checkValidationLayerSupport, 95
 - copyBuffer, 95
 - copyBufferToImage, 96
 - createBuffer, 96
 - createCommandPool, 96
 - createImageWithInfo, 97
 - createInstance, 97
 - createLogicalDevice, 98
 - createSurface, 98
 - Device, 94, 95
 - device, 99

- enableValidationLayers, 107
- endSingleTimeCommands, 100
- findMemoryType, 100
- findPhysicalQueueFamilies, 100
- findQueueFamilies, 101
- findSupportedFormat, 101
- getCommandPool, 102
- getGraphicsQueue, 102
- getPhysicalDevice, 102
- getProperties, 102
- getRequiredExtensions, 103
- getSwapChainSupport, 103
- graphicsQueue, 103
- hasGlfwRequiredInstanceExtensions, 104
- isDeviceSuitable, 104
- m_commandPool, 107
- m_debugMessenger, 107
- m_device, 107
- m_deviceExtensions, 107
- m_graphicsQueue, 108
- m_instance, 108
- m_physicalDevice, 108
- m_presentQueue, 108
- m_properties, 108
- m_surface, 108
- m_validationLayers, 109
- m_window, 109
- operator=, 104
- pickPhysicalDevice, 104
- populateDebugMessengerCreateInfo, 105
- presentQueue, 105
- querySwapChainSupport, 105
- setupDebugMessenger, 106
- surface, 106
- transitionImageLayout, 107
- ven::Engine, 109
 - ~Engine, 112
 - createInstance, 112
 - createSurface, 112
 - Engine, 111, 112
 - framePools, 115
 - loadObjects, 113
 - m_device, 115
 - m_globalPool, 115
 - m_gui, 115
 - m_instance, 115
 - m_renderer, 115
 - m_sceneManager, 116
 - m_state, 116
 - m_surface, 116
 - m_window, 116
 - mainLoop, 114
 - operator=, 114
- ven::EventManager, 117
 - ~EventManager, 118
 - EventManager, 118
 - handleEvents, 118
 - isKeyJustPressed, 118
- m_keyState, 120
- moveCamera, 119
- operator=, 119
- processKeyActions, 119
- updateEngineState, 119
- ven::FrameInfo, 120
 - camera, 122
 - commandBuffer, 122
 - frameDescriptorPool, 122
 - frameIndex, 122
 - frameTime, 122
 - globalDescriptorSet, 122
 - lights, 123
 - objects, 123
- ven::GlobalUbo, 125
 - ambientLightColor, 126
 - inverseView, 126
 - numLights, 126
 - pointLights, 126
 - projection, 126
 - view, 126
- ven::Gui, 127
 - ~Gui, 128
 - cameraSection, 129
 - cleanup, 129
 - devicePropertiesSection, 130
 - getState, 130
 - Gui, 128
 - init, 130
 - initStyle, 131
 - inputsSection, 132
 - lightsSection, 132
 - m_intensity, 135
 - m_io, 135
 - m_shininess, 135
 - m_state, 135
 - objectsSection, 132
 - operator=, 132
 - render, 132
 - rendererSection, 133
 - renderFrameWindow, 134
 - setState, 134
- ven::Gui::funcs, 123
 - IsLegacyNativeDupe, 124
- ven::KeyAction, 136
 - dir, 137
 - key, 137
 - value, 137
- ven::KeyMappings, 138
 - lookDown, 139
 - lookLeft, 139
 - lookRight, 139
 - lookUp, 139
 - moveBackward, 139
 - moveDown, 139
 - moveForward, 139
 - moveLeft, 140
 - moveRight, 140

- moveUp, 140
- toggleGui, 140
- ven::Light, 141
 - ~Light, 142
 - color, 144
 - getId, 143
 - getName, 143
 - getShininess, 143
 - Light, 142, 143
 - m_lightId, 144
 - m_name, 145
 - m_shininess, 145
 - Map, 142
 - operator=, 144
 - setName, 144
 - setShininess, 144
 - transform, 145
- ven::LightPushConstantData, 145
 - color, 146
 - position, 146
 - radius, 146
- ven::Model, 147
 - ~Model, 148
 - bind, 149
 - createIndexBuffer, 149
 - createModelFromFile, 149
 - createVertexBuffer, 150
 - draw, 151
 - m_device, 151
 - m_hasIndexBuffer, 151
 - m_indexBuffer, 151
 - m_indexCount, 151
 - m_vertexBuffer, 152
 - m_vertexCount, 152
 - Model, 148, 149
 - operator=, 151
- ven::Model::Builder, 49
 - indices, 50
 - loadModel, 50
 - processMesh, 50
 - processNode, 50
 - vertices, 50
- ven::Model::Vertex, 219
 - color, 221
 - getAttributeDescriptions, 220
 - getBindingDescriptions, 220
 - normal, 221
 - operator==, 220
 - position, 221
 - uv, 221
- ven::Object, 152
 - getBufferInfo, 155
 - getDiffuseMap, 155
 - getId, 155
 - getModel, 155
 - getName, 155
 - m_bufferInfo, 157
 - m_diffuseMap, 157
 - m_model, 157
 - m_name, 157
 - m_objId, 157
 - Map, 154
 - Object, 154, 155
 - operator=, 156
 - setBufferInfo, 156
 - setDiffuseMap, 156
 - setModel, 156
 - setName, 156
 - transform, 157
- ven::ObjectBufferData, 158
 - modelMatrix, 158
 - normalMatrix, 158
- ven::ObjectPushConstantData, 159
 - modelMatrix, 159
 - normalMatrix, 159
- ven::ObjectRenderSystem, 160
 - ObjectRenderSystem, 162
 - operator=, 163
 - render, 163
- ven::PipelineConfigInfo, 164
 - attributeDescriptions, 165
 - bindingDescriptions, 165
 - colorBlendAttachment, 166
 - colorBlendInfo, 166
 - depthStencilInfo, 166
 - dynamicStateEnables, 166
 - dynamicStateInfo, 166
 - inputAssemblyInfo, 166
 - multisampleInfo, 167
 - operator=, 165
 - PipelineConfigInfo, 165
 - pipelineLayout, 167
 - rasterizationInfo, 167
 - renderPass, 167
 - subpass, 167
- ven::PointLightData, 168
 - color, 168
 - padding, 168
 - position, 168
 - shininess, 169
- ven::PointLightRenderSystem, 169
 - operator=, 172
 - PointLightRenderSystem, 171
 - render, 172
- ven::QueueFamilyIndices, 173
 - graphicsFamily, 174
 - graphicsFamilyHasValue, 174
 - isComplete, 173
 - presentFamily, 174
 - presentFamilyHasValue, 174
- ven::Renderer, 175
 - ~Renderer, 177
 - beginFrame, 178
 - beginSwapChainRenderPass, 178
 - createCommandBuffers, 178
 - endFrame, 178

- endSwapChainRenderPass, 179
- freeCommandBuffers, 179
- getAspectRatio, 179
- getClearColor, 179
- getCurrentCommandBuffer, 180
- getFrameIndex, 180
- getSwapChainRenderPass, 181
- getWindow, 181
- isFrameInProgress, 181
- m_clearValues, 183
- m_commandBuffers, 183
- m_currentFrameIndex, 183
- m_currentImageIndex, 183
- m_device, 184
- m_isFrameStarted, 184
- m_swapChain, 184
- m_window, 184
- operator=, 182
- recreateSwapChain, 182
- Renderer, 177
- setClearValue, 182
- ven::SceneManager, 185
 - createLight, 187
 - createObject, 187
 - getBufferInfoForObject, 187
 - getLights, 187
 - getObjects, 188
 - getUboBuffers, 188
 - m_currentLightId, 189
 - m_currentObjId, 189
 - m_lights, 189
 - m_objects, 189
 - m_textureDefault, 189
 - m_uboBuffers, 189
 - operator=, 188
 - SceneManager, 186, 187
 - updateBuffer, 188
- ven::Shaders, 190
 - ~Shaders, 192
 - bind, 193
 - createGraphicsPipeline, 193
 - createShaderModule, 193
 - defaultPipelineConfigInfo, 193
 - m_device, 195
 - m_fragShaderModule, 195
 - m_graphicsPipeline, 195
 - m_vertShaderModule, 195
 - operator=, 194
 - readFile, 194
 - Shaders, 192
- ven::SwapChain, 196
 - ~SwapChain, 198
 - acquireNextImage, 199
 - chooseSwapExtent, 199
 - chooseSwapPresentMode, 199
 - chooseSwapSurfaceFormat, 199
 - compareSwapFormats, 200
 - createDepthResources, 200
 - createFrameBuffers, 200
 - createImageViews, 200
 - createRenderPass, 200
 - createSwapChain, 200
 - createSyncObjects, 200
 - extentAspectRatio, 201
 - findDepthFormat, 201
 - getFrameBuffer, 201
 - getImageView, 201
 - getRenderPass, 201
 - getSwapChainExtent, 201
 - getSwapChainImageFormat, 202
 - height, 202
 - imageCount, 202
 - init, 202
 - m_currentFrame, 203
 - m_depthImageMemory, 203
 - m_depthImages, 203
 - m_depthImageViews, 203
 - m_device, 204
 - m_imageAvailableSemaphores, 204
 - m_imagesInFlight, 204
 - m_inFlightFences, 204
 - m_oldSwapChain, 204
 - m_renderFinishedSemaphores, 204
 - m_renderPass, 205
 - m_swapChain, 205
 - m_swapChainDepthFormat, 205
 - m_swapChainExtent, 205
 - m_swapChainFrameBuffers, 205
 - m_swapChainImageFormat, 205
 - m_swapChainImages, 206
 - m_swapChainImageViews, 206
 - m_windowExtent, 206
 - operator=, 202
 - submitCommandBuffers, 203
 - SwapChain, 198, 199
 - width, 203
- ven::SwapChainSupportDetails, 206
 - capabilities, 207
 - formats, 207
 - presentModes, 207
- ven::Texture, 208
 - ~Texture, 211
 - createTextureFromFile, 211
 - createTextureImage, 211
 - createTextureImageView, 212
 - createTextureSampler, 212
 - getExtent, 213
 - getFormat, 213
 - getImage, 213
 - getImageInfo, 213
 - getImageLayout, 213
 - getImageView, 213
 - imageView, 214
 - m_descriptor, 215
 - m_device, 215
 - m_extent, 215

- m_format, [215](#)
 - m_layerCount, [215](#)
 - m_mipLevels, [215](#)
 - m_textureImage, [215](#)
 - m_textureImageMemory, [216](#)
 - m_textureImageView, [216](#)
 - m_textureLayout, [216](#)
 - m_textureSampler, [216](#)
 - operator=, [214](#)
 - sampler, [214](#)
 - Texture, [210](#), [211](#)
 - transitionLayout, [214](#)
 - updateDescriptor, [214](#)
- ven::Transform3D, [217](#)
 - normalMatrix, [218](#)
 - rotation, [218](#)
 - scale, [218](#)
 - transformMatrix, [218](#)
 - translation, [219](#)
- ven::Window, [222](#)
 - ~Window, [223](#)
 - createWindow, [224](#)
 - createWindowSurface, [224](#)
 - framebufferResizeCallback, [224](#)
 - getExtent, [225](#)
 - getGLFWWindow, [225](#)
 - m_framebufferResized, [227](#)
 - m_height, [227](#)
 - m_width, [227](#)
 - m_window, [227](#)
 - operator=, [226](#)
 - resetWindowResizedFlag, [226](#)
 - setFullscreen, [226](#)
 - wasWindowResized, [226](#)
 - Window, [223](#)
- vengine, [1](#)
- vertices
 - ven::Model::Builder, [50](#)
- view
 - ven::GlobalUbo, [126](#)
- VISIBLE
 - ven, [16](#)
- wasWindowResized
 - ven::Window, [226](#)
- WHITE_3
 - ven::Colors, [77](#)
- WHITE_4
 - ven::Colors, [77](#)
- WHITE_V
 - ven::Colors, [77](#)
- width
 - ven::SwapChain, [203](#)
- Window
 - ven::Window, [223](#)
- Window.hpp
 - GLFW_INCLUDE_VULKAN, [288](#)
- writeBuffer
 - ven::DescriptorWriter, [90](#)
- writeImage
 - ven::DescriptorWriter, [90](#)
- writeToBuffer
 - ven::Buffer, [37](#)
- writeToIndex
 - ven::Buffer, [37](#)
- YELLOW_3
 - ven::Colors, [77](#)
- YELLOW_4
 - ven::Colors, [77](#)
- YELLOW_V
 - ven::Colors, [77](#)