

vengine

0.1.0

Generated by Doxygen 1.11.0

1 engine	1
1.1 VEngine - Vulkan Graphics Engine	1
1.1.1 Features	1
1.1.2 Planned Features:	1
1.1.3 Build	2
1.1.3.1 Prerequisites	2
1.1.3.2 Linux	2
1.1.3.3 Windows	2
1.1.4 Key Bindings	2
1.1.5 Documentation	3
1.1.6 External Libraries	3
1.1.7 Commit Norms	3
1.1.8 License	3
1.1.9 Acknowledgements	4
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	16
6.1.2.3 LogLevel	16
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	18
6.1.4.5 DEFAULT_CLEAR_DEPTH	18
6.1.4.6 DEFAULT_FAR	18

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	19
6.1.4.12 DEFAULT_LIGHT_RADIUS	19
6.1.4.13 DEFAULT_LOOK_SPEED	19
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	20
6.1.4.18 DEFAULT_ROTATION	20
6.1.4.19 DEFAULT_SHININESS	20
6.1.4.20 DEFAULT_TITLE	20
6.1.4.21 DEFAULT_TRANSFORM	20
6.1.4.22 DEFAULT_WIDTH	20
6.1.4.23 DESCRIPTOR_COUNT	20
6.1.4.24 EPSILON	21
6.1.4.25 MAX_FRAMES_IN_FLIGHT	21
6.1.4.26 MAX_LIGHTS	21
6.1.4.27 MAX_OBJECTS	21
6.1.4.28 SHADERS_BIN_PATH	21
7 Class Documentation	23
7.1 ven::ARenderSystemBase Class Reference	23
7.1.1 Detailed Description	25
7.1.2 Constructor & Destructor Documentation	25
7.1.2.1 ARenderSystemBase()	25
7.1.2.2 ~ARenderSystemBase()	25
7.1.3 Member Function Documentation	26
7.1.3.1 createPipeline()	26
7.1.3.2 createPipelineLayout()	26
7.1.3.3 getDevice()	27
7.1.3.4 getPipelineLayout()	27
7.1.3.5 getShaders()	28
7.1.3.6 render()	28
7.1.4 Member Data Documentation	28
7.1.4.1 m_device	28
7.1.4.2 m_pipelineLayout	29
7.1.4.3 m_shaders	29
7.1.4.4 renderSystemLayout	29
7.2 ven::Buffer Class Reference	29

7.2.1 Detailed Description	32
7.2.2 Constructor & Destructor Documentation	32
7.2.2.1 Buffer() [1/2]	32
7.2.2.2 ~Buffer()	32
7.2.2.3 Buffer() [2/2]	32
7.2.3 Member Function Documentation	32
7.2.3.1 descriptorInfo()	32
7.2.3.2 descriptorInfoForIndex()	33
7.2.3.3 flush()	34
7.2.3.4 flushIndex()	34
7.2.3.5 getAlignment()	35
7.2.3.6 getAlignmentSize()	35
7.2.3.7 getBuffer()	36
7.2.3.8 getBufferSize()	36
7.2.3.9 getInstanceCount()	36
7.2.3.10 getInstanceSize()	36
7.2.3.11 getMappedMemory()	36
7.2.3.12 getMemoryPropertyFlags()	36
7.2.3.13 getUsageFlags()	37
7.2.3.14 invalidate()	37
7.2.3.15 invalidateIndex()	37
7.2.3.16 map()	38
7.2.3.17 operator=()	39
7.2.3.18 unmap()	39
7.2.3.19 writeToBuffer()	39
7.2.3.20 writeToIndex()	39
7.2.4 Member Data Documentation	40
7.2.4.1 m_alignmentSize	40
7.2.4.2 m_buffer	40
7.2.4.3 m_bufferSize	40
7.2.4.4 m_device	41
7.2.4.5 m_instanceCount	41
7.2.4.6 m_instanceSize	41
7.2.4.7 m_mapped	41
7.2.4.8 m_memory	41
7.2.4.9 m_memoryPropertyFlags	41
7.2.4.10 m_usageFlags	42
7.3 ven::DescriptorPool::Builder Class Reference	42
7.3.1 Detailed Description	44
7.3.2 Constructor & Destructor Documentation	44
7.3.2.1 Builder()	44
7.3.3 Member Function Documentation	44

7.3.3.1 addPoolSize()	44
7.3.3.2 build()	45
7.3.3.3 setMaxSets()	45
7.3.3.4 setPoolFlags()	46
7.3.4 Member Data Documentation	46
7.3.4.1 m_device	46
7.3.4.2 m_maxSets	46
7.3.4.3 m_poolFlags	46
7.3.4.4 m_poolSizes	47
7.4 ven::DescriptorSetLayout::Builder Class Reference	47
7.4.1 Detailed Description	49
7.4.2 Constructor & Destructor Documentation	49
7.4.2.1 Builder()	49
7.4.3 Member Function Documentation	49
7.4.3.1 addBinding()	49
7.4.3.2 build()	50
7.4.4 Member Data Documentation	50
7.4.4.1 m_bindings	50
7.4.4.2 m_device	50
7.5 ven::Model::Builder Struct Reference	51
7.5.1 Detailed Description	52
7.5.2 Member Function Documentation	52
7.5.2.1 loadModel()	52
7.5.2.2 processMesh()	52
7.5.2.3 processNode()	52
7.5.3 Member Data Documentation	53
7.5.3.1 indices	53
7.5.3.2 vertices	53
7.6 ven::Camera Class Reference	53
7.6.1 Detailed Description	55
7.6.2 Constructor & Destructor Documentation	55
7.6.2.1 Camera() [1/2]	55
7.6.2.2 ~Camera()	55
7.6.2.3 Camera() [2/2]	56
7.6.3 Member Function Documentation	56
7.6.3.1 getFar()	56
7.6.3.2 getFov()	56
7.6.3.3 getInverseView()	57
7.6.3.4 getLookSpeed()	57
7.6.3.5 getMoveSpeed()	57
7.6.3.6 getNear()	58
7.6.3.7 getProjection()	58

7.6.3.8 <code>getView()</code>	58
7.6.3.9 <code>operator=()</code>	58
7.6.3.10 <code>setFar()</code>	59
7.6.3.11 <code>setFov()</code>	59
7.6.3.12 <code>setLookSpeed()</code>	60
7.6.3.13 <code>setMoveSpeed()</code>	60
7.6.3.14 <code>setNear()</code>	61
7.6.3.15 <code>setOrthographicProjection()</code>	61
7.6.3.16 <code>setPerspectiveProjection()</code>	61
7.6.3.17 <code>setViewDirection()</code>	61
7.6.3.18 <code>setViewTarget()</code>	62
7.6.3.19 <code>setViewXYZ()</code>	62
7.6.4 Member Data Documentation	62
7.6.4.1 <code>m_far</code>	62
7.6.4.2 <code>m_fov</code>	62
7.6.4.3 <code>m_inverseViewMatrix</code>	62
7.6.4.4 <code>m_lookSpeed</code>	62
7.6.4.5 <code>m_moveSpeed</code>	63
7.6.4.6 <code>m_near</code>	63
7.6.4.7 <code>m_projectionMatrix</code>	63
7.6.4.8 <code>m_viewMatrix</code>	63
7.6.4.9 <code>transform</code>	63
7.7 <code>ven::Clock</code> Class Reference	64
7.7.1 Detailed Description	65
7.7.2 Constructor & Destructor Documentation	65
7.7.2.1 <code>Clock() [1/2]</code>	65
7.7.2.2 <code>~Clock()</code>	65
7.7.2.3 <code>Clock() [2/2]</code>	65
7.7.3 Member Function Documentation	66
7.7.3.1 <code>getDeltaTime()</code>	66
7.7.3.2 <code>getDeltaTimeMS()</code>	66
7.7.3.3 <code>getFPS()</code>	66
7.7.3.4 <code>now()</code>	67
7.7.3.5 <code>operator=()</code>	67
7.7.3.6 <code>resume()</code>	67
7.7.3.7 <code>start()</code>	67
7.7.3.8 <code>stop()</code>	67
7.7.3.9 <code>update()</code>	68
7.7.4 Member Data Documentation	68
7.7.4.1 <code>m_deltaTime</code>	68
7.7.4.2 <code>m_isStopped</code>	68
7.7.4.3 <code>mStartTime</code>	68

7.7.4.4 m_stopTime	69
7.8 ven::Gui::ClockData Struct Reference	69
7.8.1 Detailed Description	69
7.8.2 Member Data Documentation	69
7.8.2.1 deltaTimeMS	69
7.8.2.2 fps	70
7.9 ven::Colors Class Reference	70
7.9.1 Detailed Description	72
7.9.2 Member Data Documentation	72
7.9.2.1 AQUA_3	72
7.9.2.2 AQUA_4	72
7.9.2.3 AQUA_V	72
7.9.2.4 BLACK_3	72
7.9.2.5 BLACK_4	73
7.9.2.6 BLACK_V	73
7.9.2.7 BLUE_3	73
7.9.2.8 BLUE_4	73
7.9.2.9 BLUE_V	73
7.9.2.10 COLOR_PRESETS_3	73
7.9.2.11 COLOR_PRESETS_4	74
7.9.2.12 COLOR_PRESETS_VK	74
7.9.2.13 CYAN_3	74
7.9.2.14 CYAN_4	75
7.9.2.15 CYAN_V	75
7.9.2.16 FUCHSIA_3	75
7.9.2.17 FUCHSIA_4	75
7.9.2.18 FUCHSIA_V	75
7.9.2.19 GRAY_3	75
7.9.2.20 GRAY_4	75
7.9.2.21 GRAY_V	76
7.9.2.22 GREEN_3	76
7.9.2.23 GREEN_4	76
7.9.2.24 GREEN_V	76
7.9.2.25 LIME_3	76
7.9.2.26 LIME_4	76
7.9.2.27 LIME_V	76
7.9.2.28 MAGENTA_3	77
7.9.2.29 MAGENTA_4	77
7.9.2.30 MAGENTA_V	77
7.9.2.31 MAROON_3	77
7.9.2.32 MAROON_4	77
7.9.2.33 MAROON_V	77

7.9.2.34 NAVY_3	77
7.9.2.35 NAVY_4	78
7.9.2.36 NAVY_V	78
7.9.2.37 NIGHT_BLUE_3	78
7.9.2.38 NIGHT_BLUE_4	78
7.9.2.39 NIGHT_BLUE_V	78
7.9.2.40 OLIVE_3	78
7.9.2.41 OLIVE_4	78
7.9.2.42 OLIVE_V	79
7.9.2.43 RED_3	79
7.9.2.44 RED_4	79
7.9.2.45 RED_V	79
7.9.2.46 SILVER_3	79
7.9.2.47 SILVER_4	79
7.9.2.48 SILVER_V	79
7.9.2.49 SKY_BLUE_3	80
7.9.2.50 SKY_BLUE_4	80
7.9.2.51 SKY_BLUE_V	80
7.9.2.52 SUNSET_3	80
7.9.2.53 SUNSET_4	80
7.9.2.54 SUNSET_V	80
7.9.2.55 TEAL_3	80
7.9.2.56 TEAL_4	81
7.9.2.57 TEAL_V	81
7.9.2.58 WHITE_3	81
7.9.2.59 WHITE_4	81
7.9.2.60 WHITE_V	81
7.9.2.61 YELLOW_3	81
7.9.2.62 YELLOW_4	81
7.9.2.63 YELLOW_V	82
7.10 ven::DescriptorPool Class Reference	82
7.10.1 Detailed Description	84
7.10.2 Constructor & Destructor Documentation	84
7.10.2.1 DescriptorPool() [1/2]	84
7.10.2.2 ~DescriptorPool()	85
7.10.2.3 DescriptorPool() [2/2]	85
7.10.3 Member Function Documentation	85
7.10.3.1 allocateDescriptor()	85
7.10.3.2 freeDescriptors()	85
7.10.3.3 getDescriptorPool()	86
7.10.3.4 operator=()	86
7.10.3.5 resetPool()	86

7.10.4 Friends And Related Symbol Documentation	86
7.10.4.1 DescriptorWriter	86
7.10.5 Member Data Documentation	86
7.10.5.1 m_descriptorPool	86
7.10.5.2 m_device	87
7.11 ven::DescriptorsetLayout Class Reference	87
7.11.1 Detailed Description	89
7.11.2 Constructor & Destructor Documentation	89
7.11.2.1 DescriptorsetLayout() [1/2]	89
7.11.2.2 ~DescriptorsetLayout()	90
7.11.2.3 DescriptorsetLayout() [2/2]	90
7.11.3 Member Function Documentation	90
7.11.3.1 getDescriptorsetLayout()	90
7.11.3.2 operator=()	90
7.11.4 Friends And Related Symbol Documentation	90
7.11.4.1 DescriptorWriter	90
7.11.5 Member Data Documentation	91
7.11.5.1 m_bindings	91
7.11.5.2 m_descriptorsetLayout	91
7.11.5.3 m_device	91
7.12 ven::DescriptorWriter Class Reference	91
7.12.1 Detailed Description	93
7.12.2 Constructor & Destructor Documentation	93
7.12.2.1 DescriptorWriter() [1/2]	93
7.12.2.2 ~DescriptorWriter()	93
7.12.2.3 DescriptorWriter() [2/2]	93
7.12.3 Member Function Documentation	93
7.12.3.1 build()	93
7.12.3.2 operator=()	94
7.12.3.3 overwrite()	94
7.12.3.4 writeBuffer()	94
7.12.3.5 writelImage()	94
7.12.4 Member Data Documentation	95
7.12.4.1 m_pool	95
7.12.4.2 m_setLayout	95
7.12.4.3 m_writes	95
7.13 ven::Device Class Reference	95
7.13.1 Detailed Description	98
7.13.2 Constructor & Destructor Documentation	98
7.13.2.1 Device() [1/2]	98
7.13.2.2 ~Device()	99
7.13.2.3 Device() [2/2]	99

7.13.3 Member Function Documentation	99
7.13.3.1 beginSingleTimeCommands()	99
7.13.3.2 checkDeviceExtensionSupport()	99
7.13.3.3 checkValidationLayerSupport()	99
7.13.3.4 copyBuffer()	100
7.13.3.5 copyBufferToImage()	100
7.13.3.6 createBuffer()	100
7.13.3.7 createCommandPool()	101
7.13.3.8 createImageWithInfo()	101
7.13.3.9 createInstance()	102
7.13.3.10 createLogicalDevice()	102
7.13.3.11 createSurface()	102
7.13.3.12 device()	103
7.13.3.13 endSingleTimeCommands()	104
7.13.3.14 findMemoryType()	104
7.13.3.15 findPhysicalQueueFamilies()	105
7.13.3.16 findQueueFamilies()	105
7.13.3.17 findSupportedFormat()	106
7.13.3.18 getCommandPool()	106
7.13.3.19 getGraphicsQueue()	106
7.13.3.20 getInstance()	106
7.13.3.21 getPhysicalDevice()	107
7.13.3.22 getProperties()	107
7.13.3.23 getRequiredExtensions()	107
7.13.3.24 getSwapChainSupport()	108
7.13.3.25 graphicsQueue()	108
7.13.3.26 hasGIfwRequiredInstanceExtensions()	108
7.13.3.27 isDeviceSuitable()	109
7.13.3.28 operator=(())	109
7.13.3.29 pickPhysicalDevice()	109
7.13.3.30 populateDebugMessengerCreateInfo()	110
7.13.3.31 presentQueue()	110
7.13.3.32 querySwapChainSupport()	110
7.13.3.33 setupDebugMessenger()	111
7.13.3.34 surface()	111
7.13.3.35 transitionImageLayout()	111
7.13.4 Member Data Documentation	112
7.13.4.1 enableValidationLayers	112
7.13.4.2 m_commandPool	112
7.13.4.3 m_debugMessenger	112
7.13.4.4 m_device	112
7.13.4.5 m_deviceExtensions	112

7.13.4.6 m_graphicsQueue	112
7.13.4.7 m_instance	113
7.13.4.8 m_physicalDevice	113
7.13.4.9 m_presentQueue	113
7.13.4.10 m_properties	113
7.13.4.11 m_surface	113
7.13.4.12 m_validationLayers	113
7.13.4.13 m_window	114
7.14 ven::Engine Class Reference	114
7.14.1 Detailed Description	116
7.14.2 Constructor & Destructor Documentation	116
7.14.2.1 Engine() [1/2]	116
7.14.2.2 ~Engine()	117
7.14.2.3 Engine() [2/2]	117
7.14.3 Member Function Documentation	117
7.14.3.1 cleanup()	117
7.14.3.2 loadObjects()	117
7.14.3.3 mainLoop()	118
7.14.3.4 operator=()	119
7.14.4 Member Data Documentation	119
7.14.4.1 m_device	119
7.14.4.2 m_framePools	119
7.14.4.3 m_globalPool	119
7.14.4.4 m_gui	119
7.14.4.5 m_renderer	120
7.14.4.6 m_sceneManager	120
7.14.4.7 m_state	120
7.14.4.8 m_window	120
7.15 ven::EventManager Class Reference	120
7.15.1 Detailed Description	122
7.15.2 Constructor & Destructor Documentation	122
7.15.2.1 EventManager() [1/2]	122
7.15.2.2 ~EventManager()	122
7.15.2.3 EventManager() [2/2]	122
7.15.3 Member Function Documentation	122
7.15.3.1 handleEvents()	122
7.15.3.2 isKeyJustPressed()	123
7.15.3.3 moveCamera()	123
7.15.3.4 operator=()	123
7.15.3.5 processKeyActions()	123
7.15.3.6 updateEngineState()	124
7.15.4 Member Data Documentation	124

7.15.4.1 <code>m_keyState</code>	124
7.16 <code>ven::FrameInfo</code> Struct Reference	124
7.16.1 Detailed Description	126
7.16.2 Member Data Documentation	126
7.16.2.1 <code>camera</code>	126
7.16.2.2 <code>commandBuffer</code>	126
7.16.2.3 <code>frameDescriptorPool</code>	126
7.16.2.4 <code>frameIndex</code>	126
7.16.2.5 <code>frameTime</code>	126
7.16.2.6 <code>globalDescriptorSet</code>	127
7.16.2.7 <code>lights</code>	127
7.16.2.8 <code>objects</code>	127
7.17 <code>ven::Gui::funcs</code> Struct Reference	127
7.17.1 Detailed Description	128
7.17.2 Member Function Documentation	128
7.17.2.1 <code>IsLegacyNativeDupe()</code>	128
7.18 <code>ven::GlobalUbo</code> Struct Reference	129
7.18.1 Detailed Description	129
7.18.2 Member Data Documentation	130
7.18.2.1 <code>ambientLightColor</code>	130
7.18.2.2 <code>inverseView</code>	130
7.18.2.3 <code>numLights</code>	130
7.18.2.4 <code>pointLights</code>	130
7.18.2.5 <code>projection</code>	130
7.18.2.6 <code>view</code>	130
7.19 <code>ven::Gui</code> Class Reference	131
7.19.1 Detailed Description	132
7.19.2 Constructor & Destructor Documentation	133
7.19.2.1 <code>Gui() [1/2]</code>	133
7.19.2.2 <code>~Gui()</code>	133
7.19.2.3 <code>Gui() [2/2]</code>	133
7.19.3 Member Function Documentation	133
7.19.3.1 <code>cameraSection()</code>	133
7.19.3.2 <code>cleanup()</code>	134
7.19.3.3 <code>devicePropertiesSection()</code>	135
7.19.3.4 <code>getLightsToRemove()</code>	135
7.19.3.5 <code>getObjectsToRemove()</code>	135
7.19.3.6 <code>getState()</code>	135
7.19.3.7 <code>init()</code>	136
7.19.3.8 <code>initStyle()</code>	137
7.19.3.9 <code>inputsSection()</code>	137
7.19.3.10 <code>lightsSection()</code>	137

7.19.3.11 objectsSection()	138
7.19.3.12 operator=()	138
7.19.3.13 render()	138
7.19.3.14 rendererSection()	139
7.19.3.15 renderFrameWindow()	139
7.19.3.16 setState()	139
7.19.4 Member Data Documentation	140
7.19.4.1 m_intensity	140
7.19.4.2 m_io	140
7.19.4.3 m_lightsToRemove	140
7.19.4.4 m_objectsToRemove	140
7.19.4.5 m_shininess	141
7.19.4.6 m_state	141
7.20 std::hash< ven::Model::Vertex > Struct Reference	141
7.20.1 Detailed Description	141
7.20.2 Member Function Documentation	142
7.20.2.1 operator()()	142
7.21 ven::KeyAction Struct Reference	142
7.21.1 Detailed Description	143
7.21.2 Member Data Documentation	143
7.21.2.1 dir	143
7.21.2.2 key	143
7.21.2.3 value	143
7.22 ven::KeyMappings Struct Reference	144
7.22.1 Detailed Description	144
7.22.2 Member Data Documentation	145
7.22.2.1 lookDown	145
7.22.2.2 lookLeft	145
7.22.2.3 lookRight	145
7.22.2.4 lookUp	145
7.22.2.5 moveBackward	145
7.22.2.6 moveDown	145
7.22.2.7 moveForward	146
7.22.2.8 moveLeft	146
7.22.2.9 moveRight	146
7.22.2.10 moveUp	146
7.22.2.11 toggleGui	146
7.23 ven::Light Class Reference	147
7.23.1 Detailed Description	148
7.23.2 Member Typedef Documentation	148
7.23.2.1 Map	148
7.23.3 Constructor & Destructor Documentation	148

7.23.3.1 Light() [1/3]	148
7.23.3.2 ~Light()	149
7.23.3.3 Light() [2/3]	149
7.23.3.4 Light() [3/3]	149
7.23.4 Member Function Documentation	149
7.23.4.1 getId()	149
7.23.4.2 getName()	149
7.23.4.3 getShininess()	149
7.23.4.4 operator=() [1/2]	149
7.23.4.5 operator=() [2/2]	150
7.23.4.6 setName()	150
7.23.4.7 setShininess()	150
7.23.5 Member Data Documentation	150
7.23.5.1 color	150
7.23.5.2 m_lightId	150
7.23.5.3 m_name	150
7.23.5.4 m_shininess	151
7.23.5.5 transform	151
7.24 ven::LightFactory Class Reference	151
7.24.1 Detailed Description	152
7.24.2 Member Function Documentation	152
7.24.2.1 create()	152
7.24.2.2 duplicate()	152
7.24.3 Member Data Documentation	153
7.24.3.1 m_currentLightId	153
7.25 ven::LightPushConstantData Struct Reference	153
7.25.1 Detailed Description	154
7.25.2 Member Data Documentation	154
7.25.2.1 color	154
7.25.2.2 position	154
7.25.2.3 radius	154
7.26 ven::Logger Class Reference	155
7.26.1 Detailed Description	156
7.26.2 Constructor & Destructor Documentation	156
7.26.2.1 Logger()	156
7.26.3 Member Function Documentation	156
7.26.3.1 formatLogMessage()	156
7.26.3.2 getColorForDuration()	156
7.26.3.3 getInstance()	157
7.26.3.4 logExecutionTime()	157
7.26.3.5 logWarning()	158
7.26.4 Member Data Documentation	158

7.26.4.1 LOG_LEVEL_COLOR	158
7.26.4.2 LOG_LEVEL_STRING	159
7.27 ven::Model Class Reference	159
7.27.1 Detailed Description	161
7.27.2 Constructor & Destructor Documentation	161
7.27.2.1 Model() [1/2]	161
7.27.2.2 ~Model()	162
7.27.2.3 Model() [2/2]	162
7.27.3 Member Function Documentation	162
7.27.3.1 bind()	162
7.27.3.2 createIndexBuffer()	162
7.27.3.3 createVertexBuffer()	163
7.27.3.4 draw()	163
7.27.3.5 operator=()	163
7.27.4 Member Data Documentation	164
7.27.4.1 m_device	164
7.27.4.2 m_hasIndexBuffer	164
7.27.4.3 m_indexBuffer	164
7.27.4.4 m_indexCount	164
7.27.4.5 m_vertexBuffer	164
7.27.4.6 m_vertexCount	164
7.28 ven::ModelFactory Class Reference	165
7.28.1 Detailed Description	165
7.28.2 Member Function Documentation	165
7.28.2.1 create()	165
7.28.2.2 loadAll()	166
7.29 ven::Object Class Reference	166
7.29.1 Detailed Description	168
7.29.2 Member Typedef Documentation	168
7.29.2.1 Map	168
7.29.3 Constructor & Destructor Documentation	168
7.29.3.1 Object() [1/3]	168
7.29.3.2 ~Object()	169
7.29.3.3 Object() [2/3]	169
7.29.3.4 Object() [3/3]	169
7.29.4 Member Function Documentation	169
7.29.4.1 getBufferInfo()	169
7.29.4.2 getDiffuseMap()	169
7.29.4.3 getId()	170
7.29.4.4 getModel()	170
7.29.4.5 getName()	170
7.29.4.6 operator=() [1/2]	170

7.29.4.7 operator=() [2/2]	171
7.29.4.8 setBufferInfo()	171
7.29.4.9 setDiffuseMap()	171
7.29.4.10 setModel()	171
7.29.4.11 setName()	171
7.29.5 Member Data Documentation	171
7.29.5.1 m_bufferInfo	171
7.29.5.2 m_diffuseMap	172
7.29.5.3 m_model	172
7.29.5.4 m_name	172
7.29.5.5 m_objId	172
7.29.5.6 transform	172
7.30 ven::ObjectBufferData Struct Reference	173
7.30.1 Detailed Description	173
7.30.2 Member Data Documentation	173
7.30.2.1 modelMatrix	173
7.30.2.2 normalMatrix	173
7.31 ven::ObjectFactory Class Reference	174
7.31.1 Detailed Description	174
7.31.2 Member Function Documentation	174
7.31.2.1 create()	174
7.31.2.2 duplicate()	175
7.31.3 Member Data Documentation	176
7.31.3.1 m_currentObjId	176
7.32 ven::ObjectPushConstantData Struct Reference	176
7.32.1 Detailed Description	176
7.32.2 Member Data Documentation	176
7.32.2.1 modelMatrix	176
7.32.2.2 normalMatrix	177
7.33 ven::ObjectRenderSystem Class Reference	177
7.33.1 Detailed Description	179
7.33.2 Constructor & Destructor Documentation	179
7.33.2.1 ObjectRenderSystem() [1/2]	179
7.33.2.2 ObjectRenderSystem() [2/2]	180
7.33.3 Member Function Documentation	180
7.33.3.1 operator=()	180
7.33.3.2 render()	180
7.34 ven::PipelineConfigInfo Struct Reference	181
7.34.1 Detailed Description	182
7.34.2 Constructor & Destructor Documentation	182
7.34.2.1 PipelineConfigInfo() [1/2]	182
7.34.2.2 PipelineConfigInfo() [2/2]	182

7.34.3 Member Function Documentation	182
7.34.3.1 operator=()	182
7.34.4 Member Data Documentation	182
7.34.4.1 attributeDescriptions	182
7.34.4.2 bindingDescriptions	183
7.34.4.3 colorBlendAttachment	183
7.34.4.4 colorBlendInfo	183
7.34.4.5 depthStencilInfo	183
7.34.4.6 dynamicStateEnables	183
7.34.4.7 dynamicStateInfo	183
7.34.4.8 inputAssemblyInfo	184
7.34.4.9 multisampleInfo	184
7.34.4.10 pipelineLayout	184
7.34.4.11 rasterizationInfo	184
7.34.4.12 renderPass	184
7.34.4.13 subpass	184
7.35 ven::PointLightData Struct Reference	185
7.35.1 Detailed Description	185
7.35.2 Member Data Documentation	185
7.35.2.1 color	185
7.35.2.2 padding	185
7.35.2.3 position	186
7.35.2.4 shininess	186
7.36 ven::PointLightRenderSystem Class Reference	186
7.36.1 Detailed Description	188
7.36.2 Constructor & Destructor Documentation	188
7.36.2.1 PointLightRenderSystem() [1/2]	188
7.36.2.2 PointLightRenderSystem() [2/2]	189
7.36.3 Member Function Documentation	189
7.36.3.1 operator=()	189
7.36.3.2 render()	189
7.37 ven::QueueFamilyIndices Struct Reference	190
7.37.1 Detailed Description	190
7.37.2 Member Function Documentation	190
7.37.2.1 isComplete()	190
7.37.3 Member Data Documentation	191
7.37.3.1 graphicsFamily	191
7.37.3.2 graphicsFamilyHasValue	191
7.37.3.3 presentFamily	191
7.37.3.4 presentFamilyHasValue	191
7.38 ven::Renderer Class Reference	192
7.38.1 Detailed Description	193

7.38.2 Constructor & Destructor Documentation	194
7.38.2.1 Renderer() [1/2]	194
7.38.2.2 ~Renderer()	194
7.38.2.3 Renderer() [2/2]	194
7.38.3 Member Function Documentation	195
7.38.3.1 beginFrame()	195
7.38.3.2 beginSwapChainRenderPass()	195
7.38.3.3 createCommandBuffers()	195
7.38.3.4 endFrame()	196
7.38.3.5 endSwapChainRenderPass()	196
7.38.3.6 freeCommandBuffers()	196
7.38.3.7 getAspectRatio()	196
7.38.3.8 getClearColor()	197
7.38.3.9 getCurrentCommandBuffer()	197
7.38.3.10 getFrameIndex()	198
7.38.3.11 getSwapChainRenderPass()	198
7.38.3.12 getWindow()	198
7.38.3.13 isFrameInProgress()	199
7.38.3.14 operator=(())	199
7.38.3.15 recreateSwapChain()	199
7.38.3.16 setClearColor()	200
7.38.4 Member Data Documentation	200
7.38.4.1 m_clearValues	200
7.38.4.2 m_commandBuffers	200
7.38.4.3 m_currentFrameIndex	200
7.38.4.4 m_currentImageIndex	201
7.38.4.5 m_device	201
7.38.4.6 m_isFrameStarted	201
7.38.4.7 m_swapChain	201
7.38.4.8 m_window	201
7.39 ven::SceneManager Class Reference	202
7.39.1 Detailed Description	203
7.39.2 Constructor & Destructor Documentation	203
7.39.2.1 SceneManager() [1/3]	203
7.39.2.2 SceneManager() [2/3]	204
7.39.2.3 SceneManager() [3/3]	204
7.39.3 Member Function Documentation	204
7.39.3.1 addLight()	204
7.39.3.2 addObject()	204
7.39.3.3 destroyEntity()	205
7.39.3.4 destroyLight()	205
7.39.3.5 destroyObject()	205

7.39.3.6 getBufferInfoForObject()	205
7.39.3.7 getDestroyState()	205
7.39.3.8 getLights()	206
7.39.3.9 getObjects()	206
7.39.3.10 getTextureDefault()	206
7.39.3.11 getUboBuffers()	207
7.39.3.12 operator=() [1/2]	207
7.39.3.13 operator=() [2/2]	207
7.39.3.14 setDestroyState()	207
7.39.3.15 updateBuffer()	207
7.39.4 Member Data Documentation	208
7.39.4.1 m_currentLightId	208
7.39.4.2 m_currentObjId	208
7.39.4.3 m_destroyState	208
7.39.4.4 m_lights	208
7.39.4.5 m_objects	208
7.39.4.6 m_textureDefault	208
7.39.4.7 m_uboBuffers	209
7.40 ven::Shaders Class Reference	209
7.40.1 Detailed Description	211
7.40.2 Constructor & Destructor Documentation	211
7.40.2.1 Shaders() [1/2]	211
7.40.2.2 ~Shaders()	212
7.40.2.3 Shaders() [2/2]	212
7.40.3 Member Function Documentation	212
7.40.3.1 bind()	212
7.40.3.2 createGraphicsPipeline()	212
7.40.3.3 createShaderModule()	213
7.40.3.4 defaultPipelineConfigInfo()	213
7.40.3.5 operator=()	214
7.40.3.6 readFile()	214
7.40.4 Member Data Documentation	214
7.40.4.1 m_device	214
7.40.4.2 m_fragShaderModule	214
7.40.4.3 m_graphicsPipeline	214
7.40.4.4 m_vertShaderModule	215
7.41 ven::SwapChain Class Reference	215
7.41.1 Detailed Description	218
7.41.2 Constructor & Destructor Documentation	218
7.41.2.1 SwapChain() [1/3]	218
7.41.2.2 SwapChain() [2/3]	218
7.41.2.3 ~SwapChain()	219

7.41.2.4 <code>SwapChain()</code> [3 / 3]	219
7.41.3 Member Function Documentation	219
7.41.3.1 <code>acquireNextImage()</code>	219
7.41.3.2 <code>chooseSwapExtent()</code>	219
7.41.3.3 <code>chooseSwapPresentMode()</code>	219
7.41.3.4 <code>chooseSwapSurfaceFormat()</code>	220
7.41.3.5 <code>compareSwapFormats()</code>	220
7.41.3.6 <code>createDepthResources()</code>	220
7.41.3.7 <code>createFrameBuffers()</code>	220
7.41.3.8 <code>createImageViews()</code>	220
7.41.3.9 <code>createRenderPass()</code>	220
7.41.3.10 <code>createSwapChain()</code>	220
7.41.3.11 <code>createSyncObjects()</code>	221
7.41.3.12 <code>extentAspectRatio()</code>	221
7.41.3.13 <code>findDepthFormat()</code>	221
7.41.3.14 <code>getFrameBuffer()</code>	221
7.41.3.15 <code>getImageView()</code>	221
7.41.3.16 <code>getRenderPass()</code>	221
7.41.3.17 <code>getSwapChainExtent()</code>	222
7.41.3.18 <code>getSwapChainImageFormat()</code>	222
7.41.3.19 <code>height()</code>	222
7.41.3.20 <code>imageCount()</code>	222
7.41.3.21 <code>init()</code>	222
7.41.3.22 <code>operator=()</code>	223
7.41.3.23 <code>submitCommandBuffers()</code>	223
7.41.3.24 <code>width()</code>	223
7.41.4 Member Data Documentation	223
7.41.4.1 <code>m_currentFrame</code>	223
7.41.4.2 <code>m_depthImageMemory</code>	223
7.41.4.3 <code>m_depthImages</code>	223
7.41.4.4 <code>m_depthImageViews</code>	224
7.41.4.5 <code>m_device</code>	224
7.41.4.6 <code>m_imageAvailableSemaphores</code>	224
7.41.4.7 <code>m_imagesInFlight</code>	224
7.41.4.8 <code>m_inFlightFences</code>	224
7.41.4.9 <code>m_oldSwapChain</code>	224
7.41.4.10 <code>m_renderFinishedSemaphores</code>	225
7.41.4.11 <code>m_renderPass</code>	225
7.41.4.12 <code>m_swapChain</code>	225
7.41.4.13 <code>m_swapChainDepthFormat</code>	225
7.41.4.14 <code>m_swapChainExtent</code>	225
7.41.4.15 <code>m_swapChainFrameBuffers</code>	225

7.41.4.16 m_swapChainImageFormat	226
7.41.4.17 m_swapChainImages	226
7.41.4.18 m_swapChainImageViews	226
7.41.4.19 m_windowExtent	226
7.42 ven::SwapChainSupportDetails Struct Reference	226
7.42.1 Detailed Description	227
7.42.2 Member Data Documentation	227
7.42.2.1 capabilities	227
7.42.2.2 formats	227
7.42.2.3 presentModes	227
7.43 ven::Texture Class Reference	228
7.43.1 Detailed Description	229
7.43.2 Constructor & Destructor Documentation	230
7.43.2.1 Texture() [1/3]	230
7.43.2.2 Texture() [2/3]	230
7.43.2.3 ~Texture()	231
7.43.2.4 Texture() [3/3]	231
7.43.3 Member Function Documentation	231
7.43.3.1 createTextureImage()	231
7.43.3.2 createTextureImageView()	232
7.43.3.3 createTextureSampler()	232
7.43.3.4 getExtent()	232
7.43.3.5 getFormat()	233
7.43.3.6 getImage()	233
7.43.3.7 getImageInfo()	233
7.43.3.8 getImageLayout()	233
7.43.3.9 getImageView()	233
7.43.3.10 imageView()	233
7.43.3.11 operator=(())	234
7.43.3.12 sampler()	234
7.43.3.13 transitionLayout()	234
7.43.3.14 updateDescriptor()	234
7.43.4 Member Data Documentation	234
7.43.4.1 m_descriptor	234
7.43.4.2 m_device	235
7.43.4.3 m_extent	235
7.43.4.4 m_format	235
7.43.4.5 m_layerCount	235
7.43.4.6 m_mipLevels	235
7.43.4.7 m_textureImage	235
7.43.4.8 m_textureImageMemory	235
7.43.4.9 m_textureImageView	236

7.43.4.10 m_textureLayout	236
7.43.4.11 m_textureSampler	236
7.44 ven::TextureFactory Class Reference	236
7.44.1 Detailed Description	237
7.44.2 Member Function Documentation	237
7.44.2.1 create()	237
7.44.2.2 loadAll()	237
7.45 ven::Transform3D Class Reference	238
7.45.1 Detailed Description	239
7.45.2 Member Function Documentation	239
7.45.2.1 normalMatrix()	239
7.45.2.2 transformMatrix()	239
7.45.3 Member Data Documentation	240
7.45.3.1 rotation	240
7.45.3.2 scale	240
7.45.3.3 translation	240
7.46 ven::Model::Vertex Struct Reference	240
7.46.1 Detailed Description	241
7.46.2 Member Function Documentation	241
7.46.2.1 getAttributeDescriptions()	241
7.46.2.2 getBindingDescriptions()	242
7.46.2.3 operator==()	242
7.46.3 Member Data Documentation	242
7.46.3.1 color	242
7.46.3.2 normal	242
7.46.3.3 position	243
7.46.3.4 uv	243
7.47 ven::Window Class Reference	243
7.47.1 Detailed Description	244
7.47.2 Constructor & Destructor Documentation	244
7.47.2.1 Window() [1/2]	244
7.47.2.2 ~Window()	244
7.47.2.3 Window() [2/2]	245
7.47.3 Member Function Documentation	245
7.47.3.1 createWindow()	245
7.47.3.2 createWindowSurface()	245
7.47.3.3 framebufferResizeCallback()	246
7.47.3.4 getExtent()	246
7.47.3.5 getGLFWWindow()	247
7.47.3.6 operator=()	247
7.47.3.7 resetWindowResizedFlag()	247
7.47.3.8 setFullscreen()	247

7.47.3.9 wasWindowResized()	248
7.47.4 Member Data Documentation	248
7.47.4.1 m_framebufferResized	248
7.47.4.2 m_height	248
7.47.4.3 m_width	248
7.47.4.4 m_window	248
8 File Documentation	249
8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag File Reference	249
8.2 fragment_point_light.frag	249
8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag File Reference	249
8.4 fragment_shader.frag	249
8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert File Reference	250
8.6 vertex_point_light.vert	250
8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference	251
8.8 vertex_shader.vert	251
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp File Reference	252
8.9.1 Detailed Description	253
8.10 Device.hpp	253
8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Engine.hpp File Reference	254
8.11.1 Detailed Description	255
8.12 Engine.hpp	256
8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp File Reference	256
8.13.1 Detailed Description	257
8.14 EventManager.hpp	258
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp File Reference	258
8.15.1 Detailed Description	259
8.16 FrameInfo.hpp	260
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp File Reference	260
8.17.1 Detailed Description	262
8.18 Gui.hpp	262
8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ABase.hpp File Reference	263
8.19.1 Detailed Description	264
8.20 ABase.hpp	264
8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/PointLight.hpp File Reference	265
8.21.1 Detailed Description	265
8.22 PointLight.hpp	266
8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Window.hpp File Reference	266
8.23.1 Detailed Description	268
8.23.2 Macro Definition Documentation	268
8.23.2.1 GLFW_INCLUDE_VULKAN	268
8.24 Window.hpp	268

8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Light.hpp File Reference	269
8.25.1 Detailed Description	270
8.26 Light.hpp	270
8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp File Reference	271
8.27.1 Detailed Description	272
8.28 Light.hpp	272
8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp File Reference	273
8.29.1 Detailed Description	273
8.30 Model.hpp	274
8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Model.hpp File Reference	274
8.31.1 Detailed Description	275
8.32 Model.hpp	275
8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/Object.hpp File Reference	276
8.33.1 Detailed Description	277
8.34 Object.hpp	278
8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Object.hpp File Reference	278
8.35.1 Detailed Description	279
8.36 Object.hpp	279
8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp File Reference	280
8.37.1 Detailed Description	281
8.38 Object.hpp	281
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Texture.hpp File Reference	282
8.39.1 Detailed Description	283
8.40 Texture.hpp	283
8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp File Reference	283
8.41.1 Detailed Description	284
8.42 Texture.hpp	285
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Buffer.hpp File Reference	285
8.43.1 Detailed Description	286
8.44 Buffer.hpp	287
8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp File Reference	289
8.45.1 Detailed Description	290
8.46 Pool.hpp	290
8.47 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp File Reference	291
8.47.1 Detailed Description	292
8.48 SetLayout.hpp	292
8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp File Reference	293
8.49.1 Detailed Description	294
8.50 Writer.hpp	295
8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Renderer.hpp File Reference	295
8.51.1 Detailed Description	296
8.52 Renderer.hpp	297

8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp File Reference	298
8.53.1 Detailed Description	299
8.54 Shaders.hpp	299
8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp File Reference	300
8.55.1 Detailed Description	301
8.56 SwapChain.hpp	301
8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Camera.hpp File Reference	302
8.57.1 Detailed Description	304
8.58 Camera.hpp	304
8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp File Reference	305
8.59.1 Detailed Description	306
8.60 Manager.hpp	306
8.61 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp File Reference	307
8.61.1 Detailed Description	307
8.62 Transform3D.hpp	308
8.63 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp File Reference	308
8.63.1 Detailed Description	309
8.64 Clock.hpp	309
8.65 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp File Reference	310
8.66 Colors.hpp	311
8.67 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp File Reference	314
8.68 HashCombine.hpp	315
8.69 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp File Reference	315
8.69.1 Detailed Description	316
8.70 Logger.hpp	316
8.71 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp File Reference	317
8.71.1 Detailed Description	318
8.72 Utils.hpp	318
8.73 /home/runner/work/VEngine/VEngine/README.md File Reference	319
8.74 /home/runner/work/VEngine/src/Core/device.cpp File Reference	319
8.74.1 Function Documentation	319
8.74.1.1 CreateDebugUtilsMessengerEXT()	319
8.74.1.2 debugCallback()	320
8.74.1.3 DestroyDebugUtilsMessengerEXT()	320
8.75 device.cpp	321
8.76 /home/runner/work/VEngine/src/Core/engine.cpp File Reference	328
8.77 engine.cpp	328
8.78 /home/runner/work/VEngine/src/Core/eventManager.cpp File Reference	330
8.78.1 Macro Definition Documentation	331
8.78.1.1 GLM_ENABLE_EXPERIMENTAL	331
8.79 eventManager.cpp	331
8.80 /home/runner/work/VEngine/src/Core/GUI/init.cpp File Reference	332

8.81 init.cpp	333
8.82 /home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp File Reference	334
8.83 render.cpp	335
8.84 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp File Reference	339
8.85 base.cpp	339
8.86 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp File Reference	340
8.87 object.cpp	341
8.88 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp File Reference	341
8.89 pointLight.cpp	342
8.90 /home/runner/work/VEngine/VEngine/src/Core/window.cpp File Reference	342
8.91 window.cpp	342
8.92 /home/runner/work/VEngine/VEngine/src/Factories/Light.cpp File Reference	344
8.93 Light.cpp	344
8.94 /home/runner/work/VEngine/VEngine/src/Factories/Model.cpp File Reference	345
8.95 Model.cpp	345
8.96 /home/runner/work/VEngine/VEngine/src/Factories/Object.cpp File Reference	346
8.97 Object.cpp	346
8.98 /home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp File Reference	346
8.99 Texture.cpp	347
8.100 /home/runner/work/VEngine/VEngine/src/Gfx/buffer.cpp File Reference	347
8.101 buffer.cpp	348
8.102 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp File Reference	349
8.103 pool.cpp	349
8.104 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/setLayout.cpp File Reference	350
8.105 setLayout.cpp	350
8.106 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/writer.cpp File Reference	351
8.107 writer.cpp	352
8.108 /home/runner/work/VEngine/VEngine/src/Gfx/model.cpp File Reference	353
8.108.1 Macro Definition Documentation	354
8.108.1.1 GLM_ENABLE_EXPERIMENTAL	354
8.109 model.cpp	354
8.110 /home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp File Reference	356
8.111 renderer.cpp	356
8.112 /home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp File Reference	358
8.113 shaders.cpp	358
8.114 /home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp File Reference	361
8.115 swapChain.cpp	361
8.116 /home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp File Reference	366
8.116.1 Macro Definition Documentation	366
8.116.1.1 STB_IMAGE_IMPLEMENTATION	366
8.117 texture.cpp	367
8.118 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference	370

8.118.1 Function Documentation	371
8.118.1.1 main()	371
8.119 main.cpp	372
8.120 /home/runner/work/VEngine/VEngine/src/Scene/camera.cpp File Reference	372
8.121 camera.cpp	372
8.122 /home/runner/work/VEngine/VEngine/src/Scene/manager.cpp File Reference	374
8.123 manager.cpp	374
8.124 /home/runner/work/VEngine/VEngine/src/Utils/clock.cpp File Reference	376
8.125 clock.cpp	376
8.126 /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp File Reference	377
8.127 logger.cpp	377
Index	379

Chapter 1

vengine

1.1 VEngine - Vulkan Graphics Engine

WORK IN PROGRESS!

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

I Build this project to learn more about Vulkan and graphics programming in general. The goal is to create an efficient engine that can be used for various projects, such as games, simulations, and visualizations.

1.1.1 Features

- **Vulkan Rendering Pipeline:** Leveraging Vulkan for high-performance graphics rendering
- **Basic Camera System:** Control camera movement in the 3D space
- **Model Loading:** Import 3D models using `assimp`
- **Real-time debugging:** Use ImGui for real-time debugging and tool development
- **Cross-platform support** (Linux, Windows)
- **Doxygen Documentation:** Automatically generated documentation hosted on [GitHub Pages](#)

1.1.2 Planned Features:

- Ray Tracing
- Physics Integration
- Audio Integration
- Support for more input devices (e.g., mouse, game controller)
- Support for more platforms (e.g., macOS, Android, iOS, PS5 ...)

1.1.3 Build

Before building the project, make sure you update the submodules:

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

1.1.3.1 Prerequisites

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

- CMake 3.27
- C++20
- Vulkan SDK
- X11 (Linux only)
- LLVM

If you are using a Debian-based distribution, you can install the dependencies using the following command:

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

1.1.3.2 Linux

1.1.3.2.1 Build and Run

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

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

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

1.1.3.3 Windows

1.1.3.3.1 Build and Run

I build the project with CLion, also works with Visual Studio. I'm using the Visual studio toolchain with ninja.

You should create your own CMake profile depending on your configuration. Basic configuration should run the following commands:

```
$> cmake -G "Visual Studio 17 2022" .  
$> cmake --build . --config Release
```

1.1.4 Key Bindings

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

Key	Description
Z	Move forward
S	Move backward
Q	Move left
D	Move right
SHIFT	Move down
SPACE	Move up
↑	Look up
↓	Look down
←	Look left
→	Look right
à	Show debug windows

1.1.5 Documentation

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

1.1.6 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:

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	23
ven::ObjectRenderSystem	177
ven::PointLightRenderSystem	186
ven::Buffer	29
ven::DescriptorPool::Builder	42
ven::DescriptorSetLayout::Builder	47
ven::Model::Builder	51
ven::Camera	53
ven::Clock	64
ven::Gui::ClockData	69
ven::Colors	70
ven::DescriptorPool	82
ven::DescriptorSetLayout	87
ven::DescriptorWriter	91
ven::Device	95
ven::Engine	114
ven::EventManager	120
ven::FrameInfo	124
ven::Gui::funcs	127
ven::GlobalUbo	129
ven::Gui	131
std::hash< ven::Model::Vertex >	141
ven::KeyAction	142
ven::KeyMappings	144
ven::Light	147
ven::LightFactory	151
ven::LightPushConstantData	153
ven::Logger	155
ven::Model	159
ven::ModelFactory	165
ven::Object	166
ven::ObjectBufferData	173
ven::ObjectFactory	174
ven::ObjectPushConstantData	176
ven::PipelineConfigInfo	181

ven::PointLightData	185
ven::QueueFamilyIndices	190
ven::Renderer	192
ven::SceneManager	202
ven::Shaders	209
ven::SwapChain	215
ven::SwapChainSupportDetails	226
ven::Texture	228
ven::TextureFactory	236
ven::Transform3D	238
ven::Model::Vertex	240
ven::Window	243

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	23
ven::Buffer	Class for buffer	29
ven::DescriptorPool::Builder		42
ven::DescriptorSetLayout::Builder		47
ven::Model::Builder		51
ven::Camera	Class for camera	53
ven::Clock	Class for clock	64
ven::Gui::ClockData		69
ven::Colors	Class for colors	70
ven::DescriptorPool	Class for descriptor pool	82
ven::DescriptorSetLayout	Class for descriptor set layout	87
ven::DescriptorWriter	Class for descriptor writer	91
ven::Device	Class for device	95
ven::Engine	Class for engine	114
ven::EventManager	Class for event manager	120
ven::FrameInfo		124
ven::Gui::funcs		127
ven::GlobalUbo		129
ven::Gui	Class for Gui	131
std::hash< ven::Model::Vertex >		141
ven::KeyAction		142
ven::KeyMappings		144
ven::Light	Class for light	147

ven::LightFactory	
Class for light factory	151
ven::LightPushConstantData	
.	153
ven::Logger	
.	155
ven::Model	
Class for model	159
ven::ModelFactory	
Class for Model factory	165
ven::Object	
Class for object	166
ven::ObjectBufferData	
.	173
ven::ObjectFactory	
Class for object factory	174
ven::ObjectPushConstantData	
.	176
ven::ObjectRenderSystem	
Class for object render system	177
ven::PipelineConfigInfo	
.	181
ven::PointLightData	
.	185
ven::PointLightRenderSystem	
Class for point light system	186
ven::QueueFamilyIndices	
.	190
ven::Renderer	
Class for renderer	192
ven::SceneManager	
Class for object manager	202
ven::Shaders	
Class for shaders	209
ven::SwapChain	
Class for swap chain	215
ven::SwapChainSupportDetails	
.	226
ven::Texture	
Class for texture	228
ven::TextureFactory	
Class for Texture factory	236
ven::Transform3D	
Class for 3D transformation	238
ven::Model::Vertex	
.	240
ven::Window	
Class for window	243

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	249
/home/runner/work/VEngine/VEngine/assets/shaders/ fragment_shader.frag	249
/home/runner/work/VEngine/VEngine/assets/shaders/ vertex_point_light.vert	250
/home/runner/work/VEngine/VEngine/assets/shaders/ vertex_shader.vert	251
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ Device.hpp	
This file contains the Device class	252
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ Engine.hpp	
This file contains the Engine class	254
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ EventManager.hpp	
This file contains the EventManager class	256
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ FrameInfo.hpp	
This file contains the FrameInfo class	258
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ Gui.hpp	
This file contains the ImGuiWindowManager class	260
/home/runner/work/VEngine/VEngine/include/VEngine/Core/ Window.hpp	
This file contains the Window class	266
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ ABase.hpp	
This file contains the ARenderSystemBase class	263
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ Object.hpp	
This file contains the ObjectRenderSystem class	276
/home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ PointLight.hpp	
This file contains the PointLightRenderSystem class	265
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ Light.hpp	
This file contains the Light Factory class	269
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ Model.hpp	
This file contains the Model Factory class	273
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ Object.hpp	
This file contains the Object Factory class	278
/home/runner/work/VEngine/VEngine/include/VEngine/Factories/ Texture.hpp	
This file contains the Texture Factory class	282
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/ Buffer.hpp	
This file contains the Buffer class	285
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/ Model.hpp	
This file contains the Model class	274
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/ Renderer.hpp	
This file contains the Renderer class	295

/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp	298
This file contains the Shader class	298
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp	300
This file contains the Shader class	300
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp	283
This file contains the Texture class	283
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp	289
This file contains the DescriptorPool class	289
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp	291
This file contains the DescriptorSetLayout class	291
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp	293
This file contains the DescriptorsWriter class	293
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Camera.hpp	302
This file contains the Camera class	302
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp	305
This file contains the SceneManager class	305
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp	307
This file contains the Transform3D class	307
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp	271
This file contains the Light class	271
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp	280
This file contains the Object class	280
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp	308
This file contains the Clock class	308
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp	310
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp	314
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp	315
This file contains the Logger class	315
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp	317
This file contains utils for VEngine	317
/home/runner/work/VEngine/VEngine/src/main.cpp	370
/home/runner/work/VEngine/VEngine/src/Core/device.cpp	319
/home/runner/work/VEngine/VEngine/src/Core/engine.cpp	328
/home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp	330
/home/runner/work/VEngine/VEngine/src/Core/window.cpp	342
/home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp	332
/home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp	334
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp	339
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp	340
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp	341
/home/runner/work/VEngine/VEngine/src/Factories/Light.cpp	344
/home/runner/work/VEngine/VEngine/src/Factories/Model.cpp	345
/home/runner/work/VEngine/VEngine/src/Factories/Object.cpp	346
/home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp	346
/home/runner/work/VEngine/VEngine/src/Gfx/buffer.cpp	347
/home/runner/work/VEngine/VEngine/src/Gfx/model.cpp	353
/home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp	356
/home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp	358
/home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp	361
/home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp	366
/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp	349
/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/setLayout.cpp	350
/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/writer.cpp	351
/home/runner/work/VEngine/VEngine/src/Scene/camera.cpp	372
/home/runner/work/VEngine/VEngine/src/Scene/manager.cpp	374
/home/runner/work/VEngine/VEngine/src/Utils/clock.cpp	376
/home/runner/work/VEngine/VEngine/src/Utils/logger.cpp	377

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.
- class [LightFactory](#)

- Class for light factory.*
- struct [LightPushConstantData](#)
- class [Logger](#)
- class [Model](#)
 - Class for model.*
 - class [ModelFactory](#)
 - Class for Model factory.*
 - class [Object](#)
 - Class for object.*
 - struct [ObjectBufferData](#)
 - class [ObjectFactory](#)
 - Class for object factory.*
 - 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 [TextureFactory](#)
 - Class for Texture factory.*
 - 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 [GUI_STATE](#) : uint8_t { **SHOW_EDITOR** = 0 , **SHOW_PLAYER** = 1 , **HIDDEN** = 2 }
- enum class [LogLevel](#) : uint8_t { **INFO** , **WARNING** }
- enum [ENGINE_STATE](#) : uint8_t { **EDITOR** = 0 , **PLAYER** = 1 , **PAUSED** = 2 , **EXIT** = 3 }

Functions

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

Variables

- 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 uint32_t [DEFAULT_WIDTH](#) = 1920
- static constexpr uint32_t [DEFAULT_HEIGHT](#) = 1080
- static constexpr std::string_view [DEFAULT_TITLE](#) = "VEngine"
- static constexpr [Transform3D](#) [DEFAULT_TRANSFORM](#) = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F, 0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}}
- static constexpr uint32_t [DEFAULT_MAX_SETS](#) = 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 glm::vec3 [DEFAULT_POSITION](#) {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 [DEFAULT_ROTATION](#) {0.F, 0.F, 0.F}
- static constexpr float [DEFAULT_FOV](#) = glm::radians(50.0F)
- static constexpr float [DEFAULT_NEAR](#) = 0.1F
- static constexpr float [DEFAULT_FAR](#) = 100.F
- static constexpr float [DEFAULT_MOVE_SPEED](#) = 3.F
- static constexpr float [DEFAULT_LOOK_SPEED](#) = 1.5F
- static constexpr float [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 float [COLOR_MAX](#) = 255.0F

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 | |
| PLAYER | |
| PAUSED | |
| EXIT | |

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

6.1.2.2 GUI_STATE

```
enum ven::GUI\_STATE : uint8_t
```

Enumerator

| | |
|-------------|--|
| SHOW_EDITOR | |
| SHOW_PLAYER | |
| HIDDEN | |

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

6.1.2.3 LogLevel

```
enum class ven::LogLevel : uint8_t [strong]
```

Enumerator

| | |
|---------|--|
| INFO | |
| WARNING | |

Definition at line 23 of file [Logger.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 16 of file [Renderer.hpp](#).

6.1.4.5 DEFAULT_CLEAR_DEPTH

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

Definition at line 17 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 35 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 [Pool.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_TRANSFORM

```
Transform3D ven::DEFAULT_TRANSFORM = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F, 0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}} [static], [constexpr]
```

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

6.1.4.22 DEFAULT_WIDTH

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

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

6.1.4.23 DESCRIPTOR_COUNT

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

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

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

6.1.4.24 EPSILON

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

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

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

6.1.4.25 MAX_FRAMES_IN_FLIGHT

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

Definition at line 15 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\(\)](#).

6.1.4.26 MAX_LIGHTS

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

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

Referenced by [ven::LightFactory::create\(\)](#).

6.1.4.27 MAX_OBJECTS

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

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

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

6.1.4.28 SHADERS_BIN_PATH

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

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

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

Chapter 7

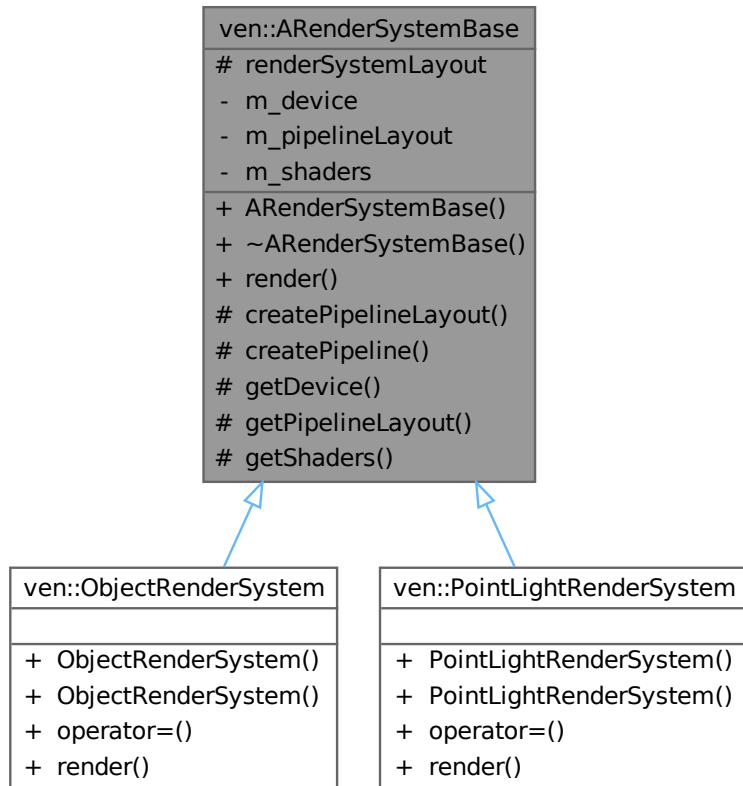
Class Documentation

7.1 ven::ARenderSystemBase Class Reference

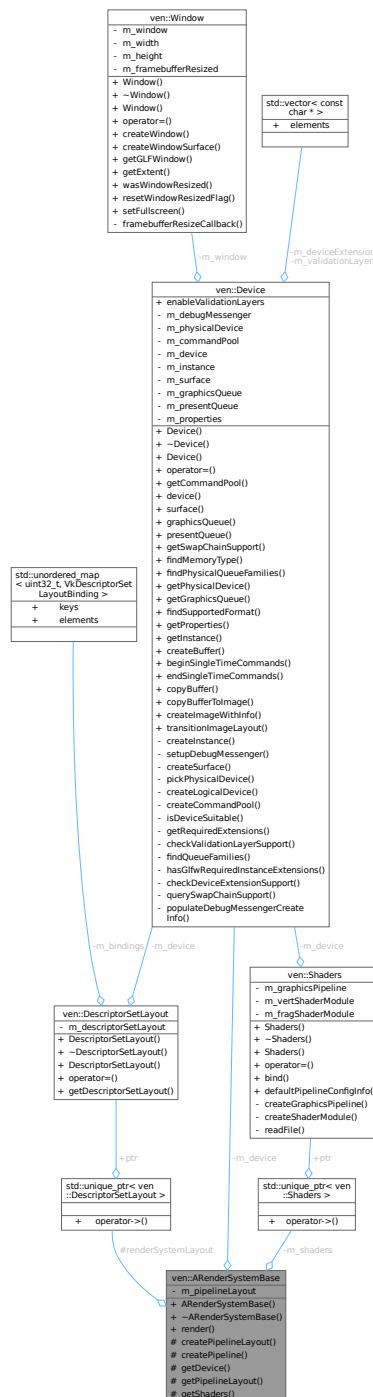
Abstract class for render system base.

```
#include <ABase.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 [20](#) of file [ABase.hpp](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ARenderSystemBase()

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

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

7.1.2.2 ~ARenderSystemBase()

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

Definition at line [25](#) of file [ABase.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 35 of file [base.cpp](#).

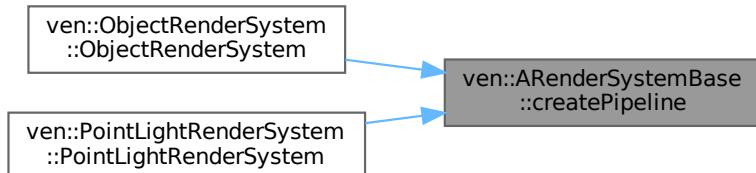
References [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

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()

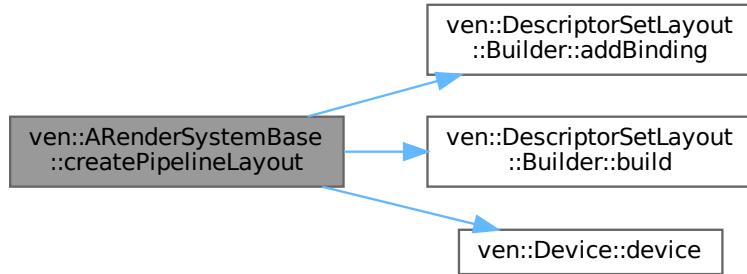
```
void ven::ARenderSystemBase::createPipelineLayout (
    VkDescriptorSetLayout globalsetLayout,
    uint32_t pushConstantSize) [protected]
```

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

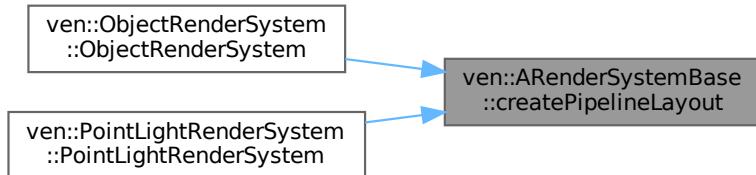
References [ven::DescriptorSetLayout::Builder::addBinding\(\)](#), [ven::DescriptorSetLayout::Builder::build\(\)](#), [ven::Device::device\(\)](#), [m_device](#), [m_pipelineLayout](#), and [renderSystemLayout](#).

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 34 of file [ABase.hpp](#).

References [m_device](#).

7.1.3.4 getPipelineLayout()

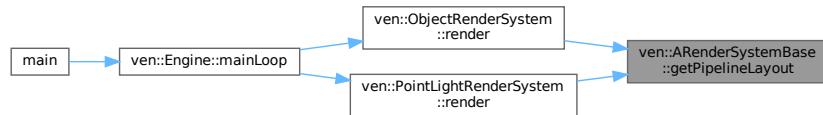
`VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]`

Definition at line 35 of file [ABase.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()

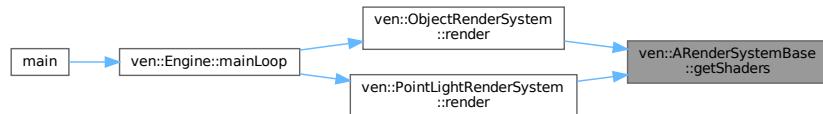
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],  
[nodiscard], [protected]
```

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

References [m_shaders](#).

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

Here is the caller graph for this function:



7.1.3.6 render()

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

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

7.1.4 Member Data Documentation

7.1.4.1 m_device

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

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

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

7.1.4.2 m_pipelineLayout

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

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

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

7.1.4.3 m_shaders

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

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

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

7.1.4.4 renderSystemLayout

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

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

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

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

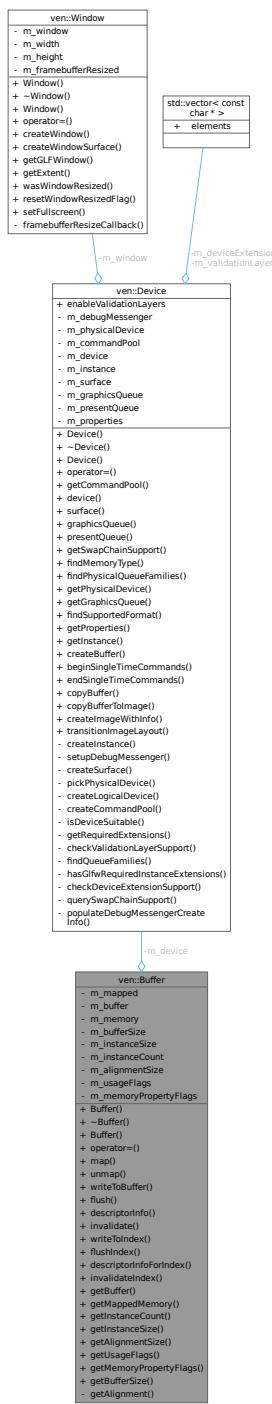
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[ABase.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/[base.cpp](#)

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](#) usage, [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 minOffsetAlignment`)

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

Private Attributes

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

7.2.1 Detailed Description

Class for buffer.

Definition at line 20 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 5 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 11 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

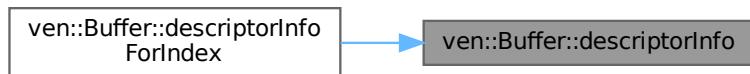
VkDescriptorBufferInfo of specified offset and range

Definition at line 76 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 index * alignmentSize |
|--------------|---|

Returns

VkDescriptorBufferInfo for instance at index

Definition at line 115 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 45 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]
```

Flush the memory range at $\text{index} * \text{alignmentSize}$ of the buffer to make it visible to the device.

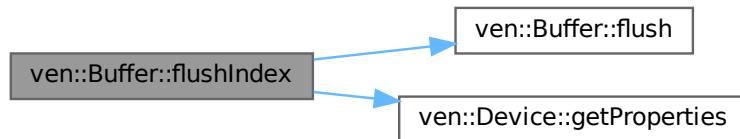
Parameters

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

Definition at line 105 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 147 of file [Buffer.hpp](#).

7.2.3.6 getAlignmentSize()

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

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

References [m_alignmentSize](#).

7.2.3.7 `getBuffer()`

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

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

References [m_buffer](#).

7.2.3.8 `getBufferSize()`

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

Definition at line 135 of file [Buffer.hpp](#).

References [m_bufferSize](#).

7.2.3.9 `getInstanceCount()`

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

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

References [m_instanceCount](#).

7.2.3.10 `getInstanceSize()`

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

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

References [m_instanceSize](#).

7.2.3.11 `getMappedMemory()`

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

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

References [m_mapped](#).

7.2.3.12 `getMemoryPropertyFlags()`

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

Definition at line 134 of file [Buffer.hpp](#).

References [m_memoryPropertyFlags](#).

7.2.3.13 getUsageFlags()

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

Definition at line 133 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

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

Returns

`VkResult` of the invalidate call

Definition at line 55 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: index * alignmentSize |
|--------------|---|

Returns

VkResult of the invalidate call

Definition at line 126 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 |

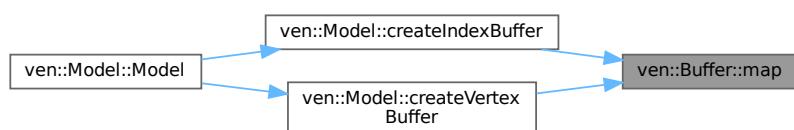
Returns

VkResult of the buffer mapping call

Definition at line 18 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 24 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 32 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 98 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 157 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 151 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 154 of file [Buffer.hpp](#).

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

7.2.4.4 m_device

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

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

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

7.2.4.5 m_instanceCount

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

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

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

7.2.4.6 m_instanceSize

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

Definition at line 155 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 150 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 152 of file [Buffer.hpp](#).

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

7.2.4.9 m_memoryPropertyFlags

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

Definition at line 159 of file [Buffer.hpp](#).

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

7.2.4.10 m_usageFlags

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

Definition at line 158 of file [Buffer.hpp](#).

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

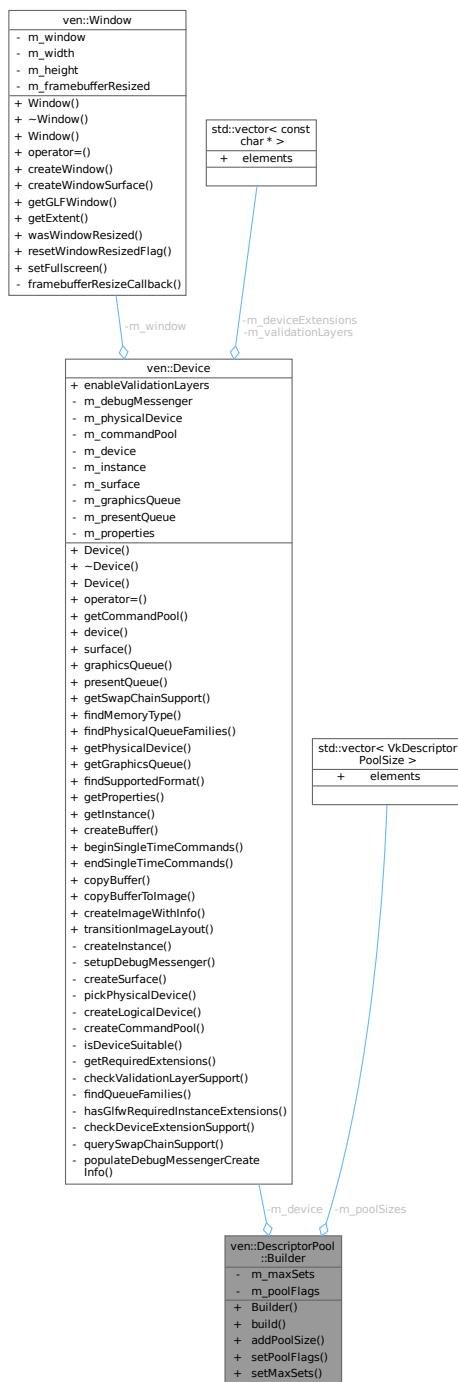
The documentation for this class was generated from the following files:

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

7.3 ven::DescriptorPool::Builder Class Reference

```
#include <Pool.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 [Pool.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 [Pool.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 [Pool.hpp](#).

References [m_poolSizes](#).

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

Here is the caller graph for this function:



7.3.3.2 build()

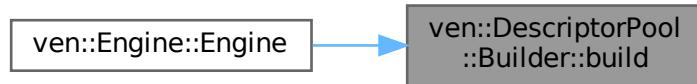
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const [inline],  
[nodiscard]
```

Definition at line 32 of file [Pool.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()

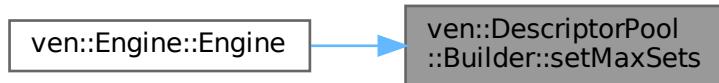
```
Builder & ven::DescriptorPool::Builder::setMaxSets (const uint32_t count) [inline]
```

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

References [m_maxSets](#).

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

Here is the caller graph for this function:



7.3.3.4 setPoolFlags()

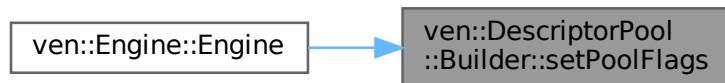
```
Builder & ven::DescriptorPool::Builder::setPoolFlags (
    const VkDescriptorPoolCreateFlags flags) [inline]
```

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

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

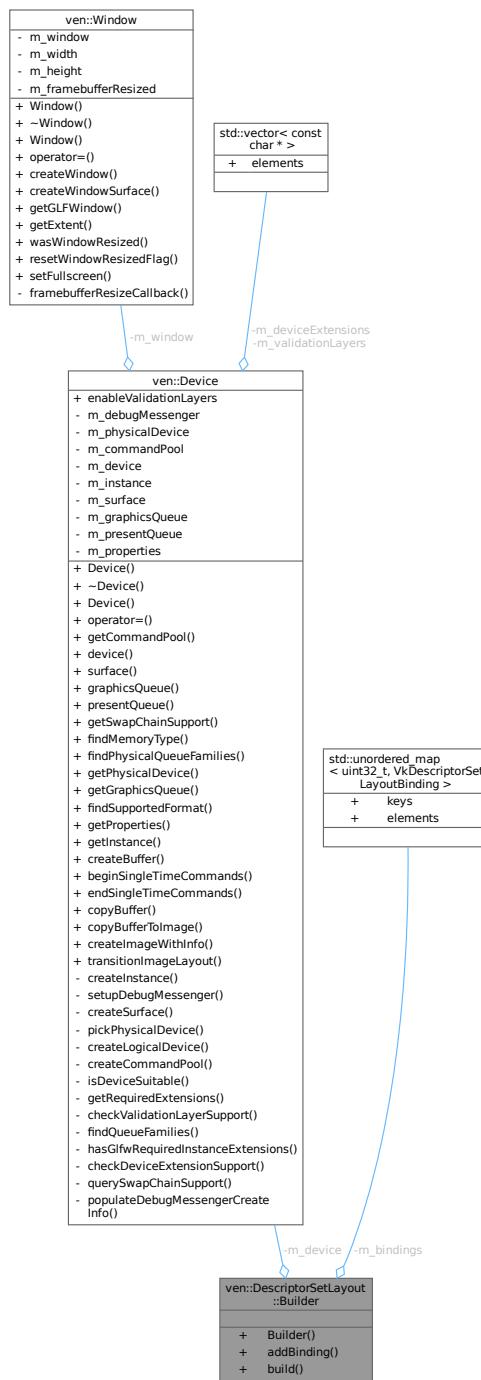
The documentation for this class was generated from the following file:

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

7.4 ven::DescriptorSetLayout::Builder Class Reference

```
#include <SetLayout.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 [SetLayout.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 [SetLayout.hpp](#).

7.4.3 Member Function Documentation

7.4.3.1 addBinding()

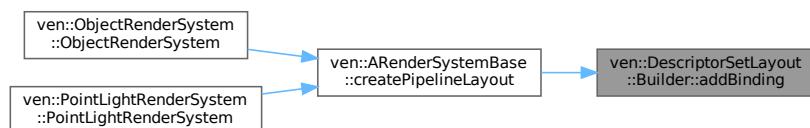
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
    uint32_t binding,
    VkDescriptorType descriptorType,
    VkShaderStageFlags stageFlags,
    uint32_t count = 1)
```

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

References [m_bindings](#).

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#).

Here is the caller graph for this function:



7.4.3.2 build()

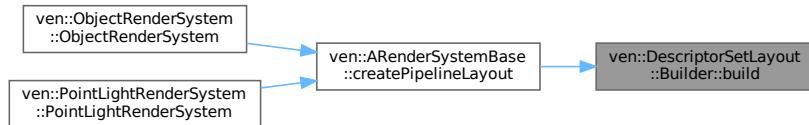
```
std::unique_ptr< DescriptorsetLayout > ven::DescriptorsetLayout::Builder::build () const
[inline]
```

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

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

Referenced by [ven::ARenderSystemBase::createPipelineLayout\(\)](#).

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 [SetLayout.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 [SetLayout.hpp](#).

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

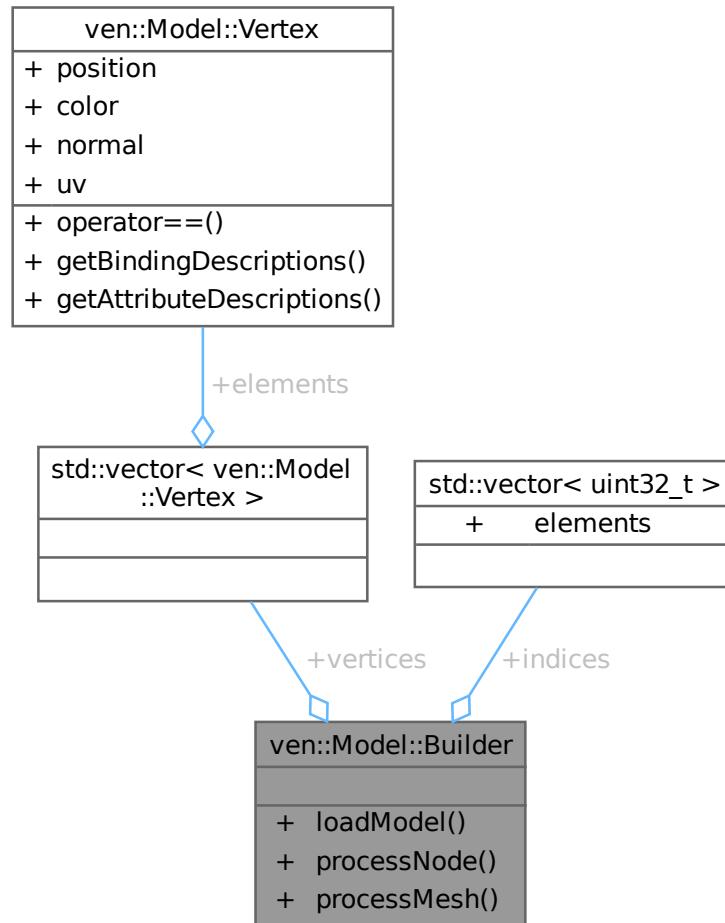
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[SetLayout.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[setLayout.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 45 of file [Model.hpp](#).

7.5.2 Member Function Documentation

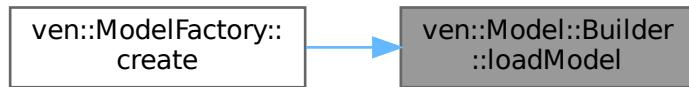
7.5.2.1 `loadModel()`

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

Definition at line 105 of file [model.cpp](#).

Referenced by [ven::ModelFactory::create\(\)](#).

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 132 of file [model.cpp](#).

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

7.5.2.3 `processNode()`

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

Definition at line 121 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 47 of file [Model.hpp](#).

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

7.5.3.2 vertices

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

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

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

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

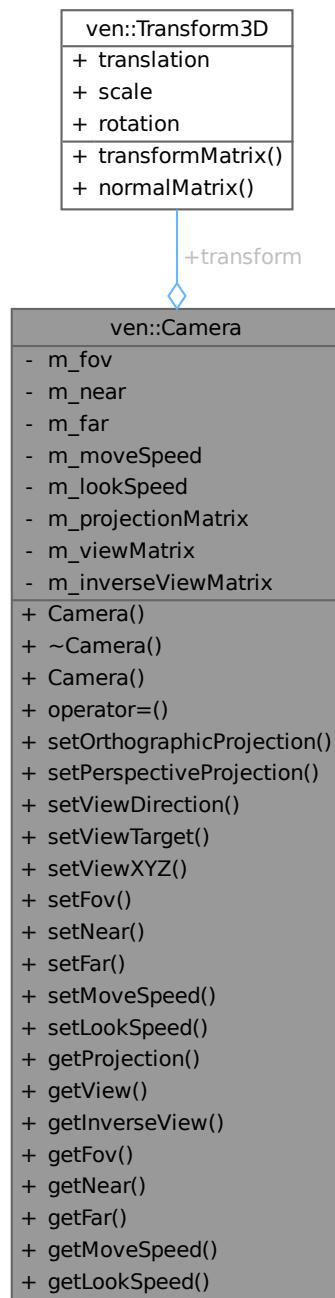
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

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()

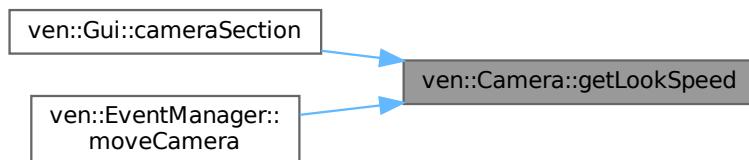
```
float ven::Camera::getLookSpeed () const [inline], [nodiscard]
```

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()

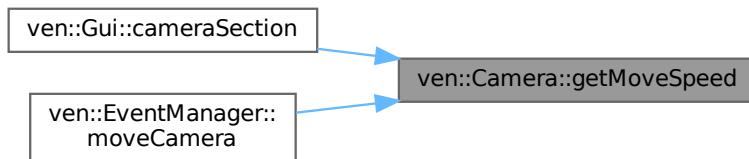
```
float ven::Camera::getMoveSpeed () const [inline], [nodiscard]
```

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

7.7 ven::Clock Class Reference

Class for clock.

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

Collaboration diagram for ven::Clock:

| ven::Clock |
|-----------------------------------|
| - m_startTime |
| - m_stopTime |
| - m_deltaTime |
| - m_isStopped |
| + Clock() |
| + ~Clock() |
| + Clock() |
| + operator=(const Clock &)=delete |
| + start() |
| + stop() |
| + resume() |
| + update() |
| + getDeltaTime() |
| + getDeltaTimeMS() |
| + getFPS() |
| + now() |

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`
- `float getDeltaTimeMS () const`
- `float getFPS () const`

Static Public Member Functions

- `static std::chrono::high_resolution_clock::time_point now ()`

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 36 of file [Clock.hpp](#).

References [m_deltaTime](#).

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

Here is the caller graph for this function:



7.7.3.2 getDeltaTimeMS()

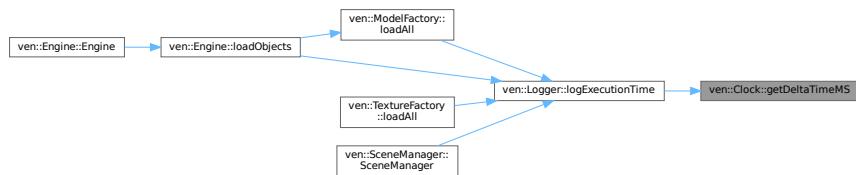
```
float ven::Clock::getDeltaTimeMS () const [inline], [nodiscard]
```

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

References [m_deltaTime](#).

Referenced by [ven::Logger::logExecutionTime\(\)](#).

Here is the caller graph for this function:



7.7.3.3 getFPS()

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

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

References [m_deltaTime](#).

7.7.3.4 now()

```
static std::chrono::high_resolution_clock::time_point ven::Clock::now () [inline], [static]
```

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

7.7.3.5 operator=()

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

7.7.3.6 resume()

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

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

7.7.3.7 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:



7.7.3.8 stop()

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

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

7.7.3.9 update()

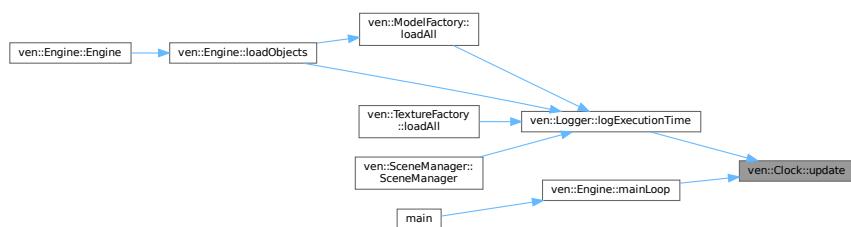
```
void ven::Clock::update ()
```

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

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

Referenced by [ven::Logger::logExecutionTime\(\)](#), and [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 44 of file [Clock.hpp](#).

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

7.7.4.2 m_isStopped

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

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

7.7.4.3 m_startTime

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

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

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

7.7.4.4 m_stopTime

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

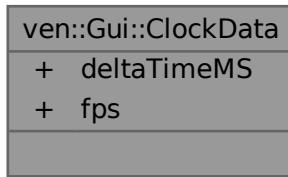
Definition at line 43 of file [Clock.hpp](#).

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::Gui::ClockData Struct Reference

Collaboration diagram for ven::Gui::ClockData:



Public Attributes

- float [deltaTimeMS](#) {0.0F}
- float [fps](#) {0.0F}

7.8.1 Detailed Description

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

7.8.2 Member Data Documentation

7.8.2.1 deltaTimeMS

```
float ven::Gui::ClockData::deltaTimeMS {0.0F}
```

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

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

7.8.2.2 fps

```
float ven::Gui::ClockData::fps {0.0F}
```

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

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

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

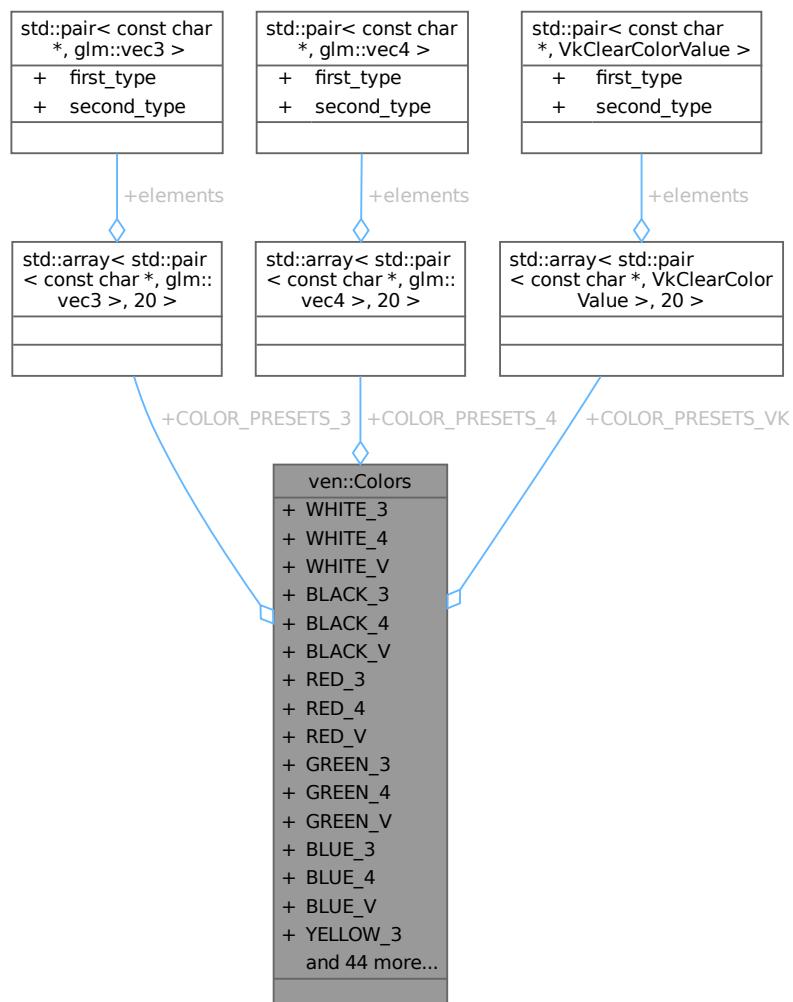
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Gui.hpp](#)

7.9 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**

7.9.1 Detailed Description

Class for colors.

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

7.9.2 Member Data Documentation

7.9.2.1 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](#).

7.9.2.2 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](#).

7.9.2.3 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](#).

7.9.2.4 BLACK_3

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

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

7.9.2.5 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](#).

7.9.2.6 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](#).

7.9.2.7 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](#).

7.9.2.8 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\(\)](#).

7.9.2.9 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](#).

7.9.2.10 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\(\)](#).

7.9.2.11 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\(\)](#).

7.9.2.12 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\(\)](#).

7.9.2.13 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](#).

7.9.2.14 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\(\)](#).

7.9.2.15 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](#).

7.9.2.16 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](#).

7.9.2.17 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](#).

7.9.2.18 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](#).

7.9.2.19 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](#).

7.9.2.20 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\(\)](#).

7.9.2.21 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](#).

7.9.2.22 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](#).

7.9.2.23 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\(\)](#).

7.9.2.24 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](#).

7.9.2.25 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](#).

7.9.2.26 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](#).

7.9.2.27 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](#).

7.9.2.28 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](#).

7.9.2.29 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\(\)](#).

7.9.2.30 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](#).

7.9.2.31 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](#).

7.9.2.32 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](#).

7.9.2.33 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](#).

7.9.2.34 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](#).

7.9.2.35 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](#).

7.9.2.36 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](#).

7.9.2.37 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](#).

7.9.2.38 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](#).

7.9.2.39 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](#).

7.9.2.40 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](#).

7.9.2.41 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](#).

7.9.2.42 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](#).

7.9.2.43 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](#).

7.9.2.44 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\(\)](#).

7.9.2.45 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](#).

7.9.2.46 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](#).

7.9.2.47 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](#).

7.9.2.48 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](#).

7.9.2.49 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](#).

7.9.2.50 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](#).

7.9.2.51 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](#).

7.9.2.52 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](#).

7.9.2.53 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](#).

7.9.2.54 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](#).

7.9.2.55 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](#).

7.9.2.56 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](#).

7.9.2.57 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](#).

7.9.2.58 WHITE_3

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

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

7.9.2.59 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](#).

7.9.2.60 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](#).

7.9.2.61 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](#).

7.9.2.62 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\(\)](#).

7.9.2.63 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:

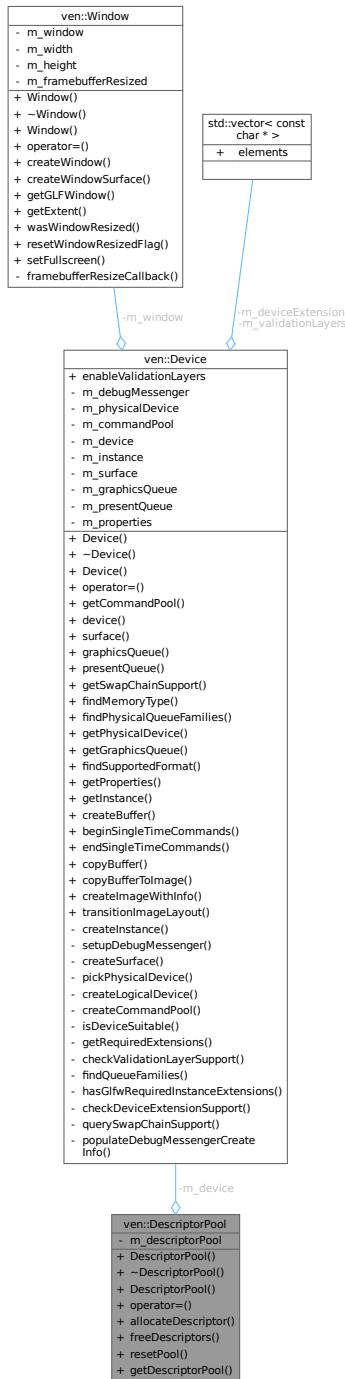
- [/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp](#)

7.10 ven::DescriptorPool Class Reference

Class for descriptor pool.

```
#include <Pool.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`

7.10.1 Detailed Description

Class for descriptor pool.

Definition at line 22 of file `Pool.hpp`.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 DescriptorPool() [1/2]

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

Definition at line 5 of file `pool.cpp`.

References `ven::Device::device()`, `m_descriptorPool`, and `m_device`.

Here is the call graph for this function:



7.10.2.2 ~DescriptorPool()

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

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

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

Here is the call graph for this function:



7.10.2.3 DescriptorPool() [2/2]

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

7.10.3 Member Function Documentation

7.10.3.1 allocateDescriptor()

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

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

7.10.3.2 freeDescriptors()

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

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

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

Here is the call graph for this function:



7.10.3.3 `getDescriptorPool()`

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

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

References [m_descriptorPool](#).

7.10.3.4 `operator=()`

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

7.10.3.5 `resetPool()`

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

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

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

Here is the call graph for this function:



7.10.4 Friends And Related Symbol Documentation

7.10.4.1 `DescriptorWriter`

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

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

7.10.5 Member Data Documentation

7.10.5.1 `m_descriptorPool`

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

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

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

7.10.5.2 m_device

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

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

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

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

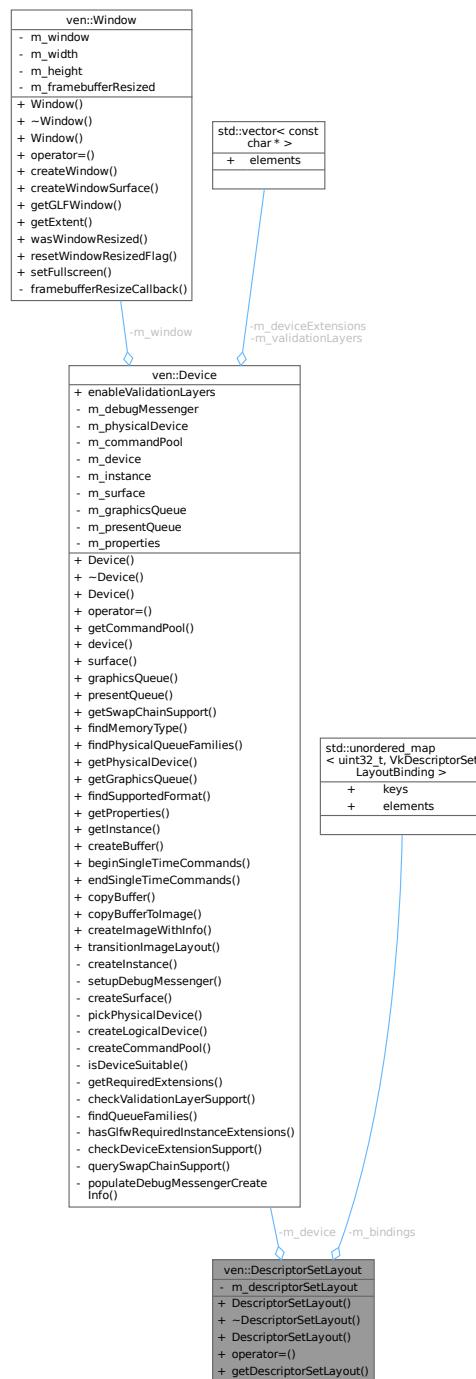
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[Pool.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[pool.cpp](#)

7.11 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

```
#include <SetLayout.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`

7.11.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file `SetLayout.hpp`.

7.11.2 Constructor & Destructor Documentation

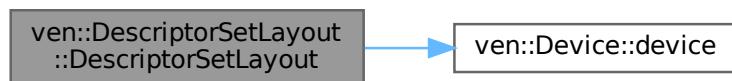
7.11.2.1 DescriptorsetLayout() [1/2]

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

Definition at line 17 of file `setLayout.cpp`.

References `ven::Device::device()`, `m_descriptorsetLayout`, and `m_device`.

Here is the call graph for this function:



7.11.2.2 ~DescriptorSetLayout()

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

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

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

Here is the call graph for this function:



7.11.2.3 DescriptorsetLayout() [2/2]

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

7.11.3 Member Function Documentation

7.11.3.1 getDescriptorsetLayout()

```
VkDescriptorsetLayout ven::DescriptorsetLayout::getDescriptorsetLayout () const [inline]
```

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

References [m_descriptorsetLayout](#).

7.11.3.2 operator=()

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

7.11.4 Friends And Related Symbol Documentation

7.11.4.1 DescriptorWriter

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

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

7.11.5 Member Data Documentation

7.11.5.1 m_bindings

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

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

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

7.11.5.2 m_descriptorsetLayout

```
VkDescriptorSetLayout ven::DescriptorsetLayout::m_descriptorsetLayout [private]
```

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

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

7.11.5.3 m_device

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

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

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

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

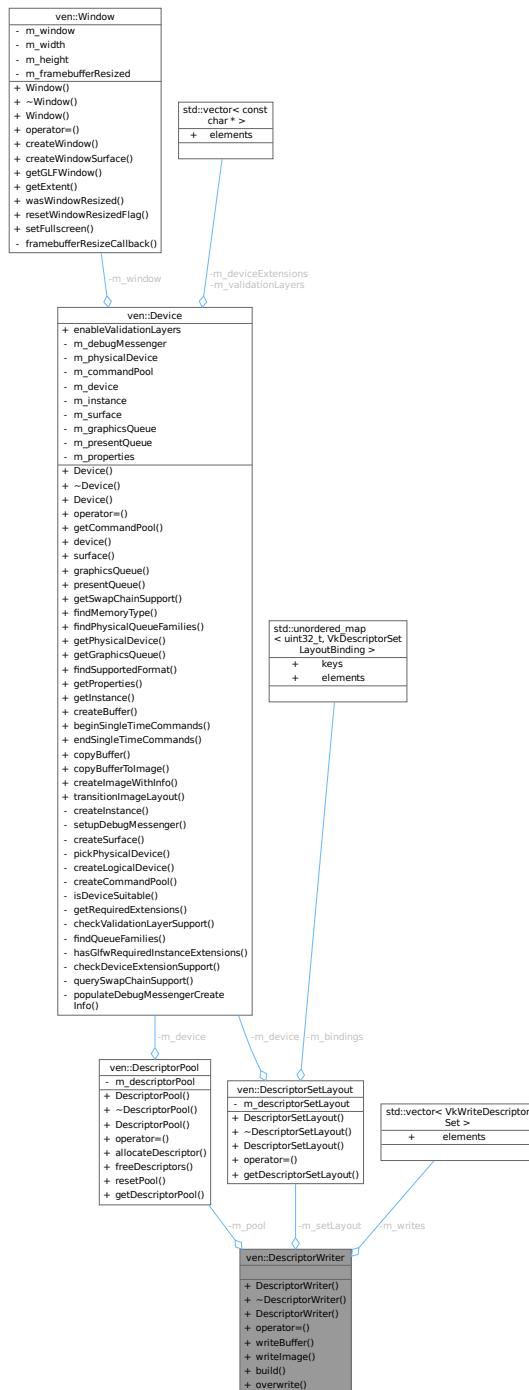
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[SetLayout.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[setLayout.cpp](#)

7.12 ven::DescriptorWriter Class Reference

Class for descriptor writer.

```
#include <Writer.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 & writelImage (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`

7.12.1 Detailed Description

Class for descriptor writer.

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

7.12.2 Constructor & Destructor Documentation

7.12.2.1 DescriptorWriter() [1/2]

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

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

7.12.2.2 ~DescriptorWriter()

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

7.12.2.3 DescriptorWriter() [2/2]

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

7.12.3 Member Function Documentation

7.12.3.1 build()

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

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

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

Here is the caller graph for this function:



7.12.3.2 operator=()

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

7.12.3.3 overwrite()

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

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

7.12.3.4 writeBuffer()

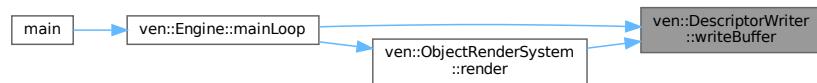
```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
    uint32_t binding,
    const VkDescriptorBufferInfo * bufferInfo)
```

Definition at line 5 of file [writer.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:



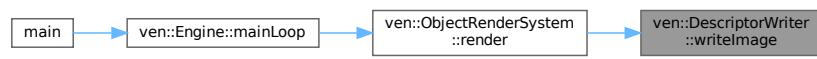
7.12.3.5 writeImage()

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

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

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

Here is the caller graph for this function:



7.12.4 Member Data Documentation

7.12.4.1 m_pool

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

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

7.12.4.2 m_setLayout

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

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

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

7.12.4.3 m_writes

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

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

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

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/[Wwriter.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/[writer.cpp](#)

7.13 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`
- `VkInstance getInstance () 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

- `Window & m_window`
- `VkDebugUtilsMessengerEXT m_debugMessenger`
- `VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE`
- `VkCommandPool m_commandPool`
- `VkDevice m_device`
- `VkInstance m_instance`
- `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}`

7.13.1 Detailed Description

Class for device.

Definition at line 35 of file `Device.hpp`.

7.13.2 Constructor & Destructor Documentation

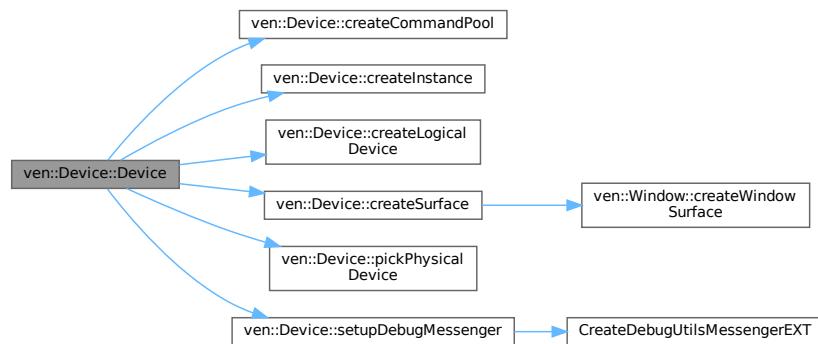
7.13.2.1 Device() [1/2]

```
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:



7.13.2.2 ~Device()

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

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

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

Here is the call graph for this function:



7.13.2.3 Device() [2/2]

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

7.13.3 Member Function Documentation

7.13.3.1 beginSingleTimeCommands()

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

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

7.13.3.2 checkDeviceExtensionSupport()

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

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

7.13.3.3 checkValidationLayerSupport()

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

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

7.13.3.4 copyBuffer()

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

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

7.13.3.5 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](#).

7.13.3.6 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:



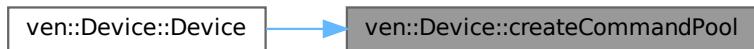
7.13.3.7 createCommandPool()

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

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

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

Here is the caller graph for this function:



7.13.3.8 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:



7.13.3.9 `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:



7.13.3.10 `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:



7.13.3.11 `createSurface()`

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

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

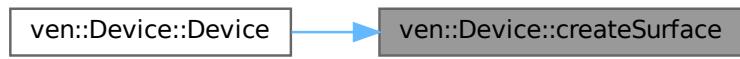
References [ven::Window::createWindowSurface\(\)](#), [m_instance](#), [m_surface](#), and [m_window](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.12 device()

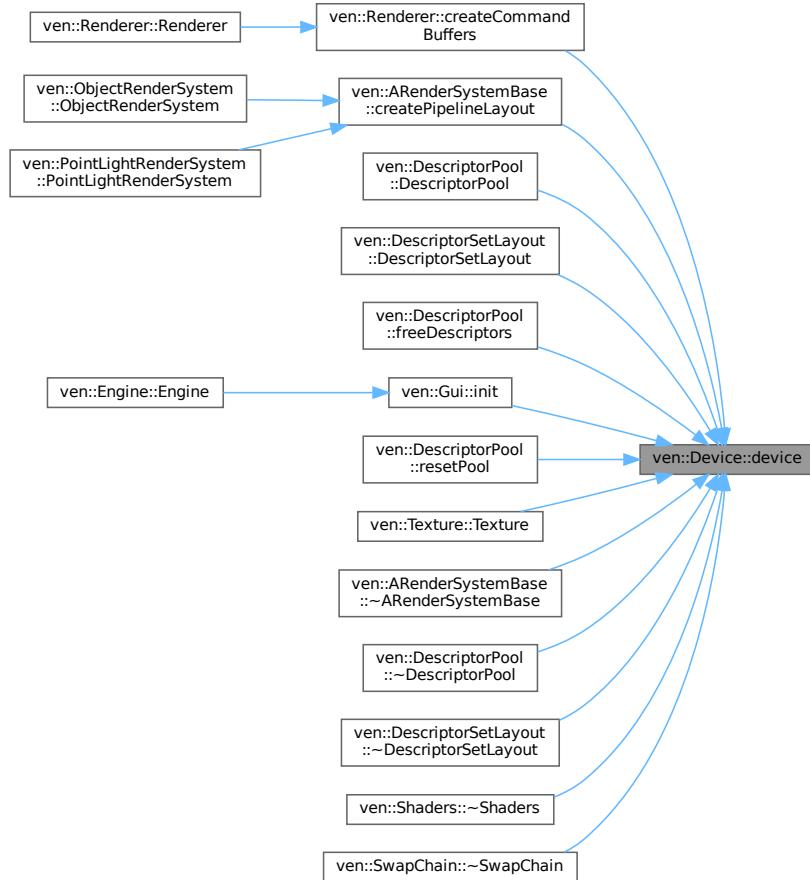
```
VkDevice ven::Device::device () const [inline], [nodiscard]
```

Definition at line 52 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:



7.13.3.13 endSingleTimeCommands()

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

Definition at line 432 of file [device.cpp](#).

7.13.3.14 findMemoryType()

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

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

7.13.3.15 findPhysicalQueueFamilies()

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

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

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

Here is the call graph for this function:



7.13.3.16 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::presentFamilyHasV](#)

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.17 `findSupportedFormat()`

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

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

7.13.3.18 `getCommandPool()`

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

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

References [m_commandPool](#).

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

Here is the caller graph for this function:



7.13.3.19 `getGraphicsQueue()`

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

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

References [m_graphicsQueue](#).

7.13.3.20 `getInstance()`

```
VkInstance ven::Device::getInstance () const [inline], [nodiscard]
```

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

References [m_instance](#).

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

Here is the caller graph for this function:



7.13.3.21 getPhysicalDevice()

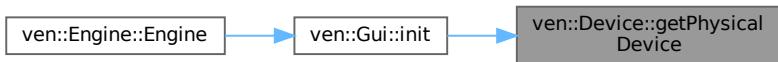
```
VkPhysicalDevice ven::Device::getPhysicalDevice () const [inline], [nodiscard]
```

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

References [m_physicalDevice](#).

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

Here is the caller graph for this function:



7.13.3.22 getProperties()

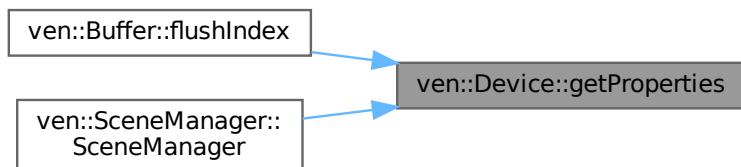
```
VkPhysicalDeviceProperties ven::Device::getProperties () const [inline], [nodiscard]
```

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

References [m_properties](#).

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

Here is the caller graph for this function:



7.13.3.23 getRequiredExtensions()

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

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

7.13.3.24 getSwapChainSupport()

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

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

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

Here is the call graph for this function:



7.13.3.25 graphicsQueue()

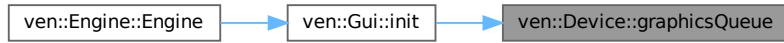
```
VkQueue ven::Device::graphicsQueue () const [inline], [nodiscard]
```

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

References [m_graphicsQueue](#).

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

Here is the caller graph for this function:



7.13.3.26 hasGlfwRequiredInstanceExtensions()

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

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

7.13.3.27 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:



7.13.3.28 operator=()

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

7.13.3.29 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:



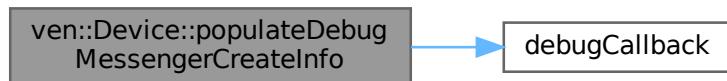
7.13.3.30 populateDebugMessengerCreateInfo()

```
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:



7.13.3.31 presentQueue()

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

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

References [m_presentQueue](#).

7.13.3.32 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::rasterizationLayers](#).

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

Here is the caller graph for this function:



7.13.3.33 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:



7.13.3.34 surface()

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

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

References [m_surface](#).

7.13.3.35 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](#).

7.13.4 Member Data Documentation

7.13.4.1 enableValidationLayers

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

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

7.13.4.2 m_commandPool

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

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

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

7.13.4.3 m_debugMessenger

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

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

7.13.4.4 m_device

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

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

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

7.13.4.5 m_deviceExtensions

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

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

7.13.4.6 m_graphicsQueue

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

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

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

7.13.4.7 m_instance

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

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

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

7.13.4.8 m_physicalDevice

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

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

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

7.13.4.9 m_presentQueue

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

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

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

7.13.4.10 m_properties

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

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

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

7.13.4.11 m_surface

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

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

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

7.13.4.12 m_validationLayers

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

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

7.13.4.13 m_window

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

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

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

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

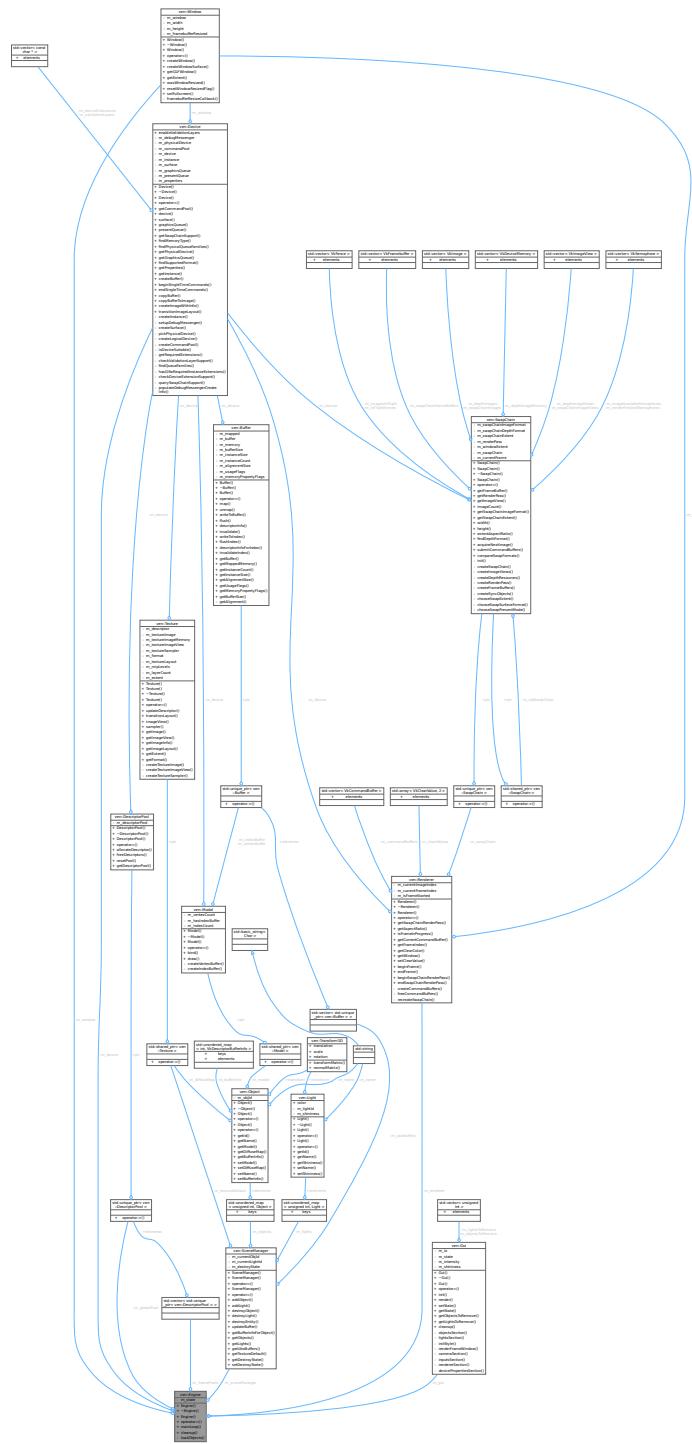
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp
- /home/runner/work/VEngine/VEngine/src/Core/[device.cpp](#)

7.14 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 ()
 - void cleanup ()

Private Member Functions

- void `loadObjects ()`

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 > > m_framePools`
- `SceneManager m_sceneManager {m_device}`

7.14.1 Detailed Description

Class for engine.

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

7.14.2 Constructor & Destructor Documentation

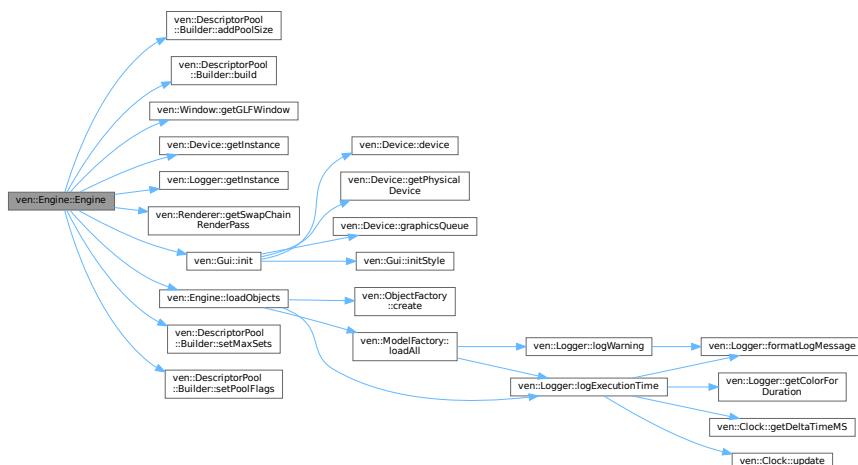
7.14.2.1 Engine() [1/2]

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

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

References `ven::DescriptorPool::Builder::addPoolSize()`, `ven::DescriptorPool::Builder::build()`, `ven::EDITOR`, `ven::Window::getGLFWWindow()`, `ven::Device::getInstance()`, `ven::Logger::getInstance()`, `ven::Renderer::getSwapChainRenderPass()`, `ven::Gui::init()`, `loadObjects()`, `m_device`, `m_framePools`, `m_globalPool`, `m_gui`, `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:



7.14.2.2 ~Engine()

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

7.14.2.3 Engine() [2/2]

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

7.14.3 Member Function Documentation

7.14.3.1 cleanup()

```
void ven::Engine::cleanup ()
```

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

References [ven::Gui::cleanup\(\)](#).

Here is the call graph for this function:



7.14.3.2 loadObjects()

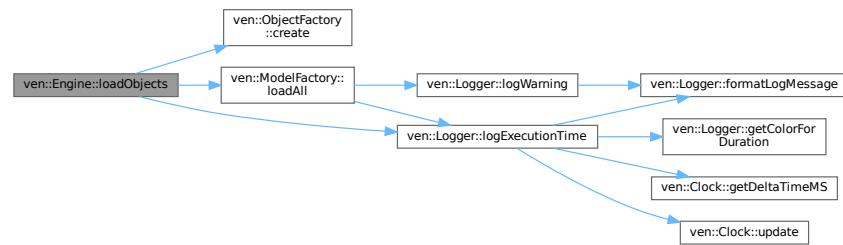
```
void ven::Engine::loadObjects () [private]
```

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

References [ven::Colors::BLUE_4](#), [ven::ObjectFactory::create\(\)](#), [ven::Colors::CYAN_4](#), [ven::Colors::GREEN_4](#), [ven::ModelFactory::loadAll\(\)](#), [ven::Logger::logExecutionTime\(\)](#), [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:



7.14.3.3 mainLoop()

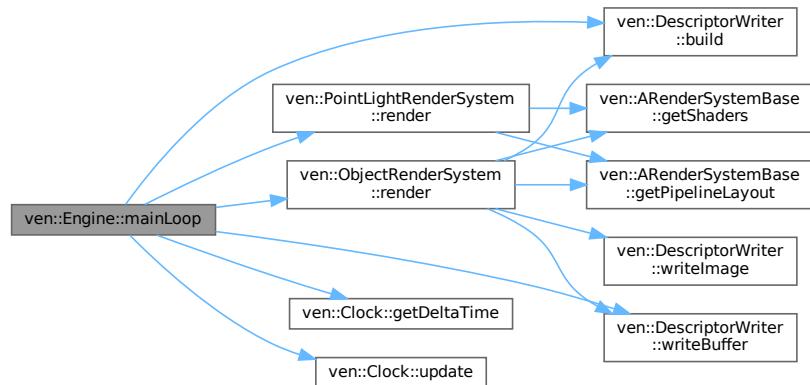
```
void ven::Engine::mainLoop ()
```

Definition at line 87 of file [engine.cpp](#).

References [ven::DescriptorWriter::build\(\)](#), [ven::EXIT](#), [ven::FrameInfo::frameIndex](#), [ven::Clock::getDeltaTime\(\)](#), [ven::HIDDEN](#), [ven::MAX_FRAMES_IN_FLIGHT](#), [ven::ObjectRenderSystem::render\(\)](#), [ven::PointLightRenderSystem::render\(\)](#), [ven::Clock::update\(\)](#), and [ven::DescriptorWriter::writeBuffer\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.3.4 operator=()

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

7.14.4 Member Data Documentation

7.14.4.1 m_device

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

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

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

7.14.4.2 m_framePools

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

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

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

7.14.4.3 m_globalPool

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

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

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

7.14.4.4 m_gui

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

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

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

7.14.4.5 m_renderer

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

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

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

7.14.4.6 m_sceneManager

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

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

7.14.4.7 m_state

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

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

7.14.4.8 m_window

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

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

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

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

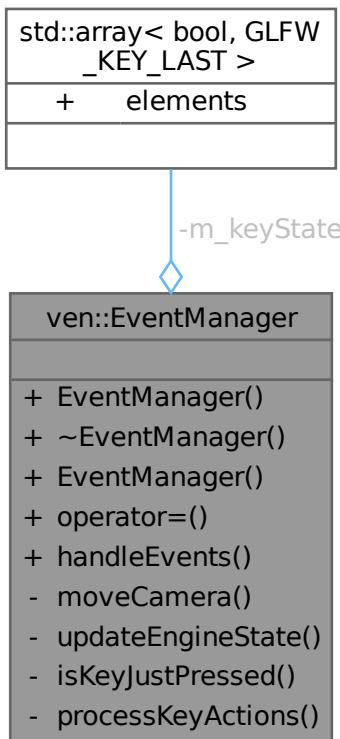
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Engine.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/[engine.cpp](#)

7.15 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, long unsigned int key, std::array< bool, GLFW_KEY_LAST > &keyStates)`
- `template<typename Iterator> static void processKeyActions (GLFWwindow *window, Iterator begin, Iterator end)`

Private Attributes

- `std::array< bool, GLFW_KEY_LAST > m_keyState {}`

7.15.1 Detailed Description

Class for event manager.

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

7.15.2 Constructor & Destructor Documentation

7.15.2.1 EventManager() [1/2]

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

7.15.2.2 ~EventManager()

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

7.15.2.3 EventManager() [2/2]

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

7.15.3 Member Function Documentation

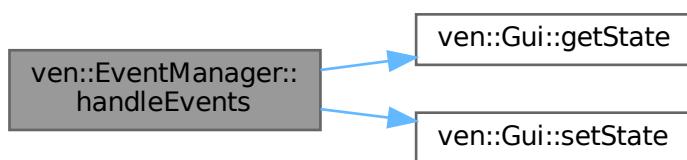
7.15.3.1 handleEvents()

```
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::EDITOR](#), [ven::EXIT](#), [ven::Gui::getState\(\)](#), [ven::HIDDEN](#), [ven::Gui::setState\(\)](#), [ven::SHOW_EDITOR](#), [ven::SHOW_PLAYER](#), and [ven::KeyMappings::toggleGui](#).

Here is the call graph for this function:



7.15.3.2 isKeyJustPressed()

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

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

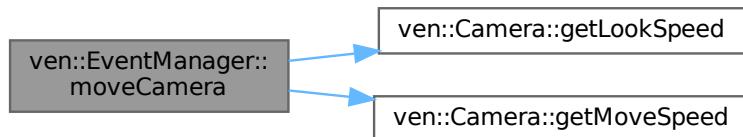
7.15.3.3 moveCamera()

```
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:



7.15.3.4 operator=()

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

7.15.3.5 processKeyActions()

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

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

7.15.3.6 updateEngineState()

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

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

7.15.4 Member Data Documentation

7.15.4.1 m_keyState

```
std::array<bool, GLFW_KEY_LAST> ven::EventManager::m_keyState {} [mutable], [private]
```

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

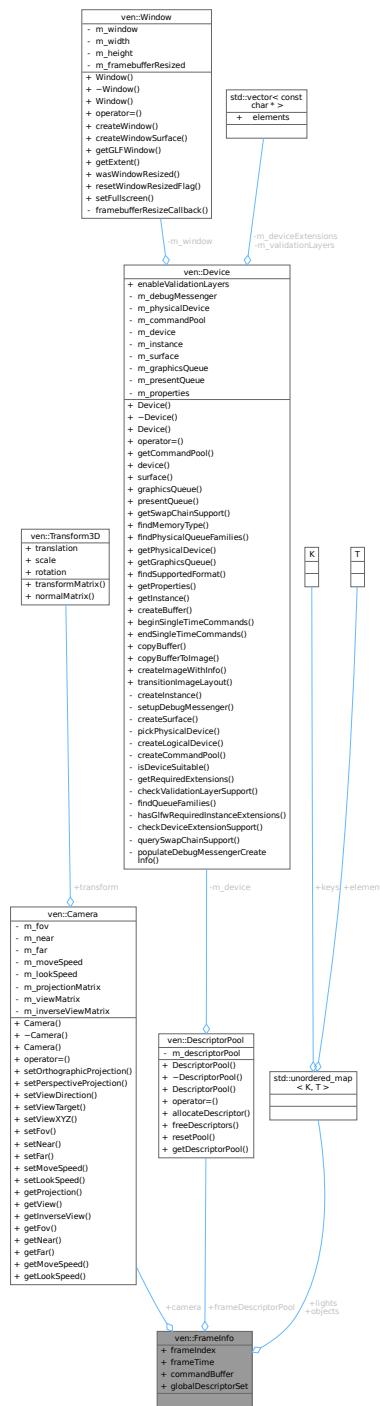
The documentation for this class was generated from the following files:

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

7.16 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`

7.16.1 Detailed Description

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

7.16.2 Member Data Documentation

7.16.2.1 camera

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

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

7.16.2.2 commandBuffer

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

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

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

7.16.2.3 frameDescriptorPool

`DescriptorPool&` `ven::FrameInfo::frameDescriptorPool`

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

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

7.16.2.4 frameIndex

`unsigned long` `ven::FrameInfo::frameIndex`

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

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

7.16.2.5 frameTime

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

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

7.16.2.6 globalDescriptorSet

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

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

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

7.16.2.7 lights

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

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

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

7.16.2.8 objects

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

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

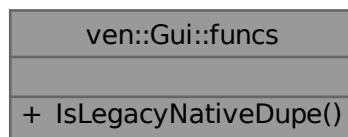
Referenced by [ven::ObjectRenderSystem::render\(\)](#).

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

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

7.17 ven::Gui::funcs Struct Reference

Collaboration diagram for ven::Gui::funcs:



Static Public Member Functions

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

7.17.1 Detailed Description

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

7.17.2 Member Function Documentation

7.17.2.1 `IsLegacyNativeDupe()`

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

Definition at line 66 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:



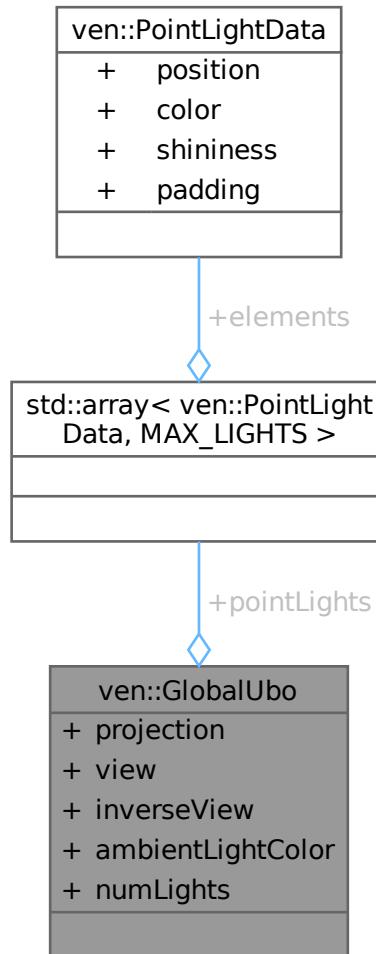
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp

7.18 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`

7.18.1 Detailed Description

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

7.18.2 Member Data Documentation

7.18.2.1 ambientLightColor

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

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

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

7.18.2.2 inverseView

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

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

7.18.2.3 numLights

```
uint8_t ven::GlobalUbo::numLights
```

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

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

7.18.2.4 pointLights

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

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

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

7.18.2.5 projection

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

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

7.18.2.6 view

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

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

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

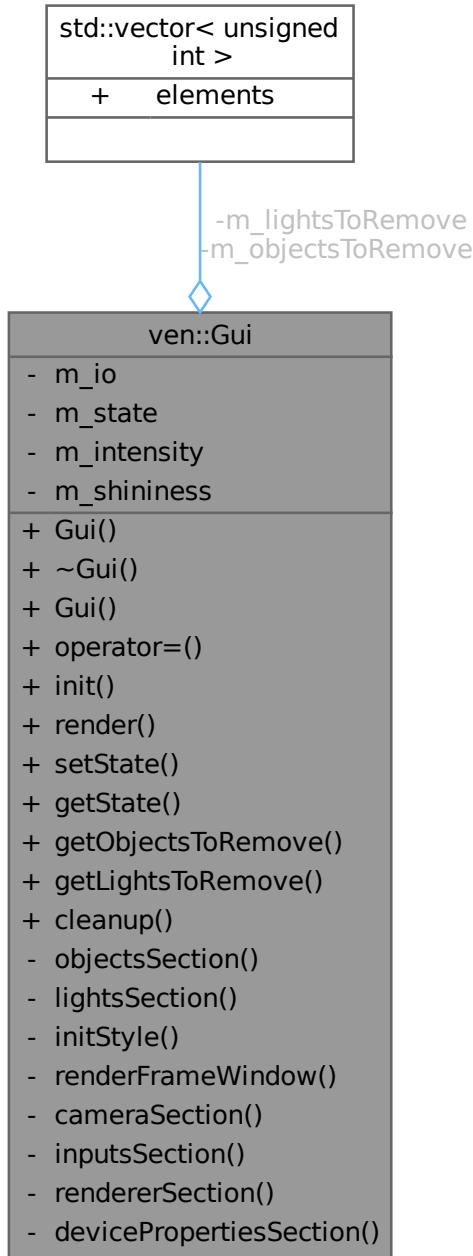
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[FrameInfo.hpp](#)

7.19 ven::Gui Class Reference

Class for [Gui](#).

```
#include <Gui.hpp>
```

Collaboration diagram for ven::Gui:



Classes

- struct [ClockData](#)
- struct [funcs](#)

Public Member Functions

- `Gui ()=default`
- `~Gui ()=default`
- `Gui (const Gui &)=delete`
- `Gui & operator= (const Gui &)=delete`
- `void init (GLFWwindow *window, VkInstance instance, const Device *device, VkRenderPass renderPass)`
- `void render (Renderer *renderer, SceneManager &sceneManager, Camera &camera, VkPhysicalDevice physicalDevice, GlobalUbo &ubo, const ClockData &clockData)`
- `void setState (const GUI_STATE state)`
- `GUI_STATE getState () const`
- `std::vector< unsigned int > * getObjectsToRemove ()`
- `std::vector< unsigned int > * getLightsToRemove ()`

Static Public Member Functions

- `static void cleanup ()`

Private Member Functions

- `void objectsSection (SceneManager &sceneManager)`
- `void lightsSection (SceneManager &sceneManager)`

Static Private Member Functions

- `static void initStyle ()`
- `static void renderFrameWindow (const ClockData &clockData)`
- `static void cameraSection (Camera &camera)`
- `static void inputsSection (const ImGuilO &io)`
- `static void rendererSection (Renderer *renderer, GlobalUbo &ubo)`
- `static void devicePropertiesSection (VkPhysicalDeviceProperties deviceProperties)`

Private Attributes

- `ImGuiIO * m_io {nullptr}`
- `GUI_STATE m_state {HIDDEN}`
- `float m_intensity {1.0F}`
- `float m_shininess {DEFAULT_SHININESS}`
- `std::vector< unsigned int > m_objectsToRemove`
- `std::vector< unsigned int > m_lightsToRemove`

7.19.1 Detailed Description

Class for [Gui](#).

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

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Gui() [1/2]

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

7.19.2.2 ~Gui()

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

7.19.2.3 Gui() [2/2]

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

7.19.3 Member Function Documentation

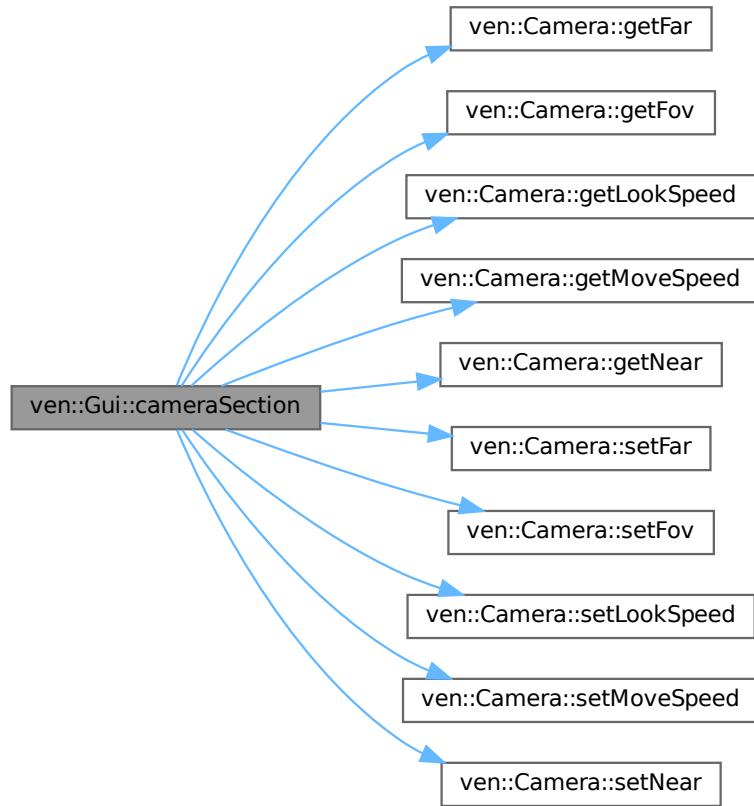
7.19.3.1 cameraSection()

```
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:



7.19.3.2 cleanup()

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

Definition at line 11 of file [render.cpp](#).

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

Here is the caller graph for this function:



7.19.3.3 devicePropertiesSection()

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

Definition at line 304 of file [render.cpp](#).

7.19.3.4 getLightsToRemove()

```
std::vector< unsigned int > * ven::Gui::getLightsToRemove () [inline], [nodiscard]
```

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

References [m_lightsToRemove](#).

7.19.3.5 getObjectsToRemove()

```
std::vector< unsigned int > * ven::Gui::getObjectsToRemove () [inline], [nodiscard]
```

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

References [m_objectsToRemove](#).

7.19.3.6 getState()

```
GUI_STATE ven::Gui::getState () const [inline], [nodiscard]
```

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

References [m_state](#).

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

Here is the caller graph for this function:



7.19.3.7 init()

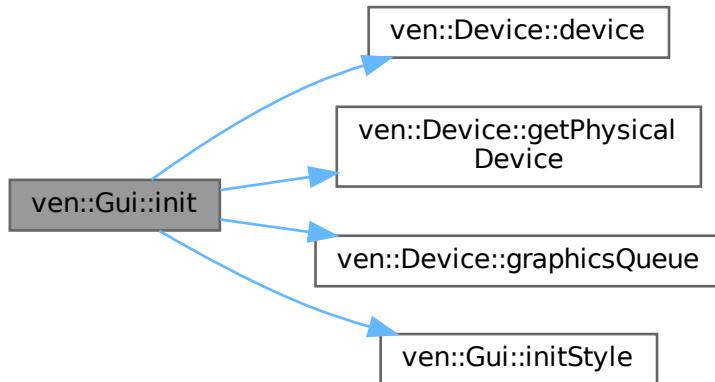
```
void ven::Gui::init (
    GLFWwindow * window,
    VkInstance instance,
    const Device * device,
    VkRenderPass renderPass)
```

Definition at line 6 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:



7.19.3.8 initStyle()

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

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

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

Here is the caller graph for this function:



7.19.3.9 inputsSection()

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

Definition at line 282 of file [render.cpp](#).

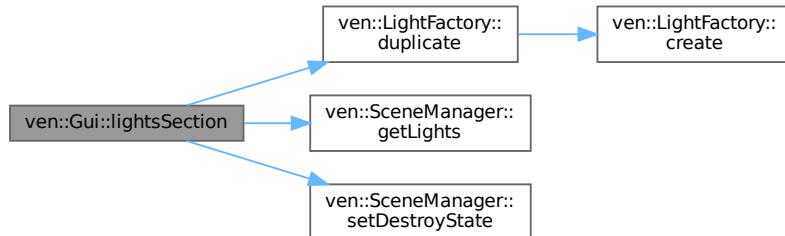
7.19.3.10 lightsSection()

```
void ven::Gui::lightsSection (
    SceneManager & sceneManager) [private]
```

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

References [ven::Colors::COLOR_PRESETS_3](#), [ven::DEFAULT_LIGHT_INTENSITY](#), [ven::DEFAULT_SHININESS](#), [ven::LightFactory::duplicate\(\)](#), [ven::SceneManager::getLights\(\)](#), and [ven::SceneManager::setDestroyState\(\)](#).

Here is the call graph for this function:



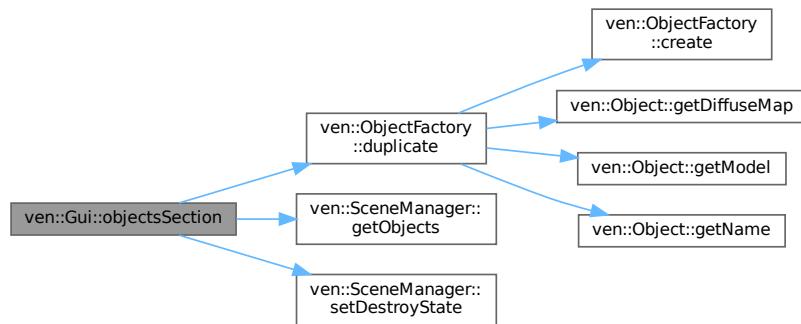
7.19.3.11 objectsSection()

```
void ven::Gui::objectsSection (
    SceneManager & sceneManager) [private]
```

Definition at line 158 of file [render.cpp](#).

References [ven::ObjectFactory::duplicate\(\)](#), [ven::SceneManager::getObjects\(\)](#), [ven::Colors::GRAY_4](#), and [ven::SceneManager::setDestroyState\(\)](#).

Here is the call graph for this function:



7.19.3.12 operator=()

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

7.19.3.13 render()

```
void ven::Gui::render (
    Renderer * renderer,
    SceneManager & sceneManager,
    Camera & camera,
    VkPhysicalDevice physicalDevice,
    GlobalUbo & ubo,
    const ClockData & clockData)
```

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

References [ven::Renderer::getCurrentCommandBuffer\(\)](#).

Here is the call graph for this function:



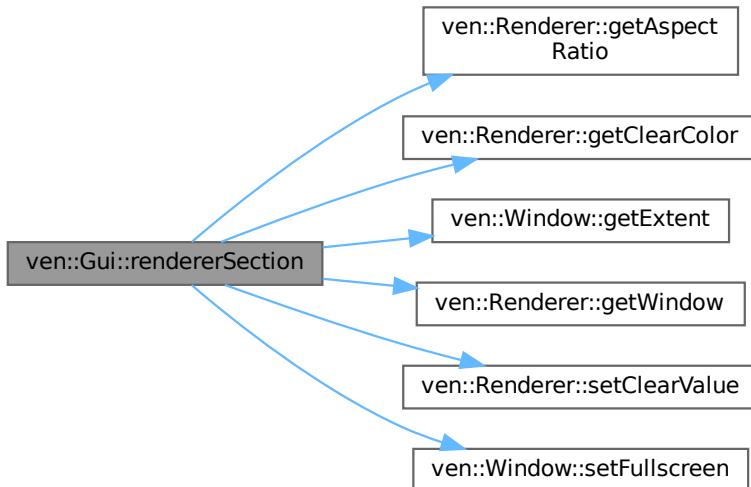
7.19.3.14 rendererSection()

```
void ven::Gui::rendererSection (
    Renderer * renderer,
    GlobalUbo & ubo) [static], [private]
```

Definition at line 48 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::setClearValue\(\)](#), and [ven::Window::setFullscreen\(\)](#).

Here is the call graph for this function:



7.19.3.15 renderFrameWindow()

```
void ven::Gui::renderFrameWindow (
    const ClockData & clockData) [static], [private]
```

Definition at line 39 of file [render.cpp](#).

References [ven::Gui::ClockData::deltaTimeMS](#), and [ven::Gui::ClockData::fps](#).

7.19.3.16 setState()

```
void ven::Gui::setState (
    const GUI_STATE state) [inline]
```

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

References [m_state](#).

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

Here is the caller graph for this function:



7.19.4 Member Data Documentation

7.19.4.1 m_intensity

```
float ven::Gui::m_intensity {1.0F} [private]
```

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

7.19.4.2 m_io

```
ImGuiIO* ven::Gui::m_io {nullptr} [private]
```

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

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

7.19.4.3 m_lightsToRemove

```
std::vector<unsigned int> ven::Gui::m_lightsToRemove [private]
```

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

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

7.19.4.4 m_objectsToRemove

```
std::vector<unsigned int> ven::Gui::m_objectsToRemove [private]
```

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

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

7.19.4.5 m_shininess

```
float ven::Gui::m_shininess {DEFAULT_SHININESS} [private]
```

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

7.19.4.6 m_state

```
GUI_STATE ven::Gui::m_state {HIDDEN} [private]
```

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

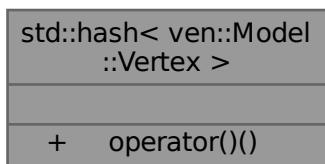
Referenced by [getState\(\)](#), and [setState\(\)](#).

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Gui.hpp
- /home/runner/work/VEngine/VEngine/src/Core/GUI/[init.cpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/GUI/[render.cpp](#)

7.20 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

7.20.1 Detailed Description

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

7.20.2 Member Function Documentation

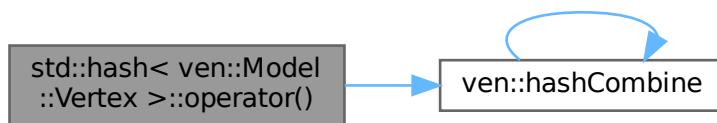
7.20.2.1 operator()()

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

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

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

Here is the call graph for this function:



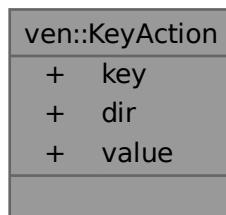
The documentation for this struct was generated from the following file:

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

7.21 ven::KeyAction Struct Reference

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

Collaboration diagram for ven::KeyAction:



Public Attributes

- `uint16_t key`
- `glm::vec3 * dir`
- `glm::vec3 value`

7.21.1 Detailed Description

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

7.21.2 Member Data Documentation

7.21.2.1 `dir`

```
glm::vec3* ven::KeyAction::dir
```

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

7.21.2.2 `key`

```
uint16_t ven::KeyAction::key
```

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

7.21.2.3 `value`

```
glm::vec3 ven::KeyAction::value
```

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

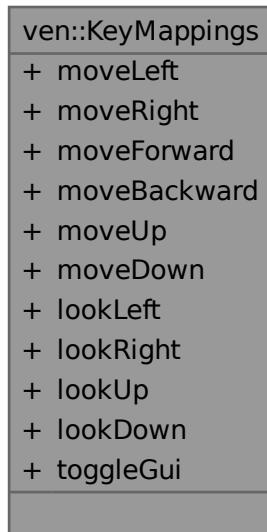
The documentation for this struct was generated from the following file:

- `/home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp`

7.22 ven::KeyMappings Struct Reference

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

Collaboration diagram for ven::KeyMappings:



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_0

7.22.1 Detailed Description

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

7.22.2 Member Data Documentation

7.22.2.1 lookDown

```
uint16_t ven::KeyMappings::lookDown = GLFW_KEY_DOWN
```

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

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

7.22.2.2 lookLeft

```
uint16_t ven::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

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

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

7.22.2.3 lookRight

```
uint16_t ven::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

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

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

7.22.2.4 lookUp

```
uint16_t ven::KeyMappings::lookUp = GLFW_KEY_UP
```

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

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

7.22.2.5 moveBackward

```
uint16_t ven::KeyMappings::moveBackward = GLFW_KEY_S
```

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

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

7.22.2.6 moveDown

```
uint16_t ven::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

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

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

7.22.2.7 moveForward

```
uint16_t ven::KeyMappings::moveForward = GLFW_KEY_W
```

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

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

7.22.2.8 moveLeft

```
uint16_t ven::KeyMappings::moveLeft = GLFW_KEY_A
```

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

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

7.22.2.9 moveRight

```
uint16_t ven::KeyMappings::moveRight = GLFW_KEY_D
```

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

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

7.22.2.10 moveUp

```
uint16_t ven::KeyMappings::moveUp = GLFW_KEY_SPACE
```

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

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

7.22.2.11 toggleGui

```
uint16_t ven::KeyMappings::toggleGui = GLFW_KEY_0
```

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

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

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

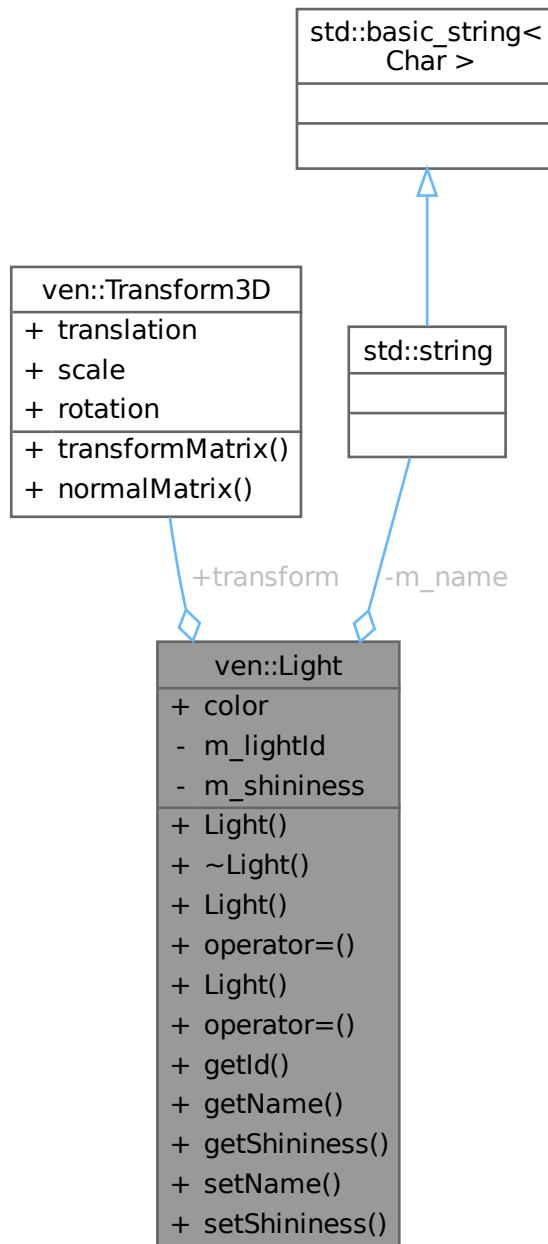
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp

7.23 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}`

7.23.1 Detailed Description

Class for light.

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

7.23.2 Member Typedef Documentation

7.23.2.1 Map

```
using ven::Light::Map = std::unordered_map<unsigned int, Light>
```

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

7.23.3 Constructor & Destructor Documentation

7.23.3.1 Light() [1/3]

```
ven::Light::Light (
    const unsigned int objId) [inline], [explicit]
```

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

7.23.3.2 ~Light()

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

7.23.3.3 Light() [2/3]

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

7.23.3.4 Light() [3/3]

```
ven::Light::Light (
    Light && ) [default]
```

7.23.4 Member Function Documentation

7.23.4.1 getId()

```
unsigned int ven::Light::getId () const [inline], [nodiscard]
```

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

References [m_lightId](#).

7.23.4.2 getName()

```
std::string ven::Light::getName () const [inline], [nodiscard]
```

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

References [m_name](#).

7.23.4.3 getShininess()

```
float ven::Light::getShininess () const [inline], [nodiscard]
```

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

References [m_shininess](#).

7.23.4.4 operator=() [1/2]

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

7.23.4.5 `operator=()` [2/2]

```
Light & ven::Light::operator= (
    Light &&) [default]
```

7.23.4.6 `setName()`

```
void ven::Light::setName (
    const std::string & name) [inline]
```

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

References [m_name](#).

7.23.4.7 `setShininess()`

```
void ven::Light::setShininess (
    const float shininess) [inline]
```

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

References [m_shininess](#).

7.23.5 Member Data Documentation

7.23.5.1 `color`

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

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

Referenced by [ven::LightFactory::duplicate\(\)](#).

7.23.5.2 `m_lightId`

```
unsigned int ven::Light::m_lightId [private]
```

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

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

7.23.5.3 `m_name`

```
std::string ven::Light::m_name {"point light"} [private]
```

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

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.23.5.4 m_shininess

```
float ven::Light::m_shininess {DEFAULT_SHININESS} [private]
```

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

Referenced by [getShininess\(\)](#), and [setShininess\(\)](#).

7.23.5.5 transform

```
Transform3D ven::Light::transform {}
```

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

Referenced by [ven::LightFactory::duplicate\(\)](#).

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

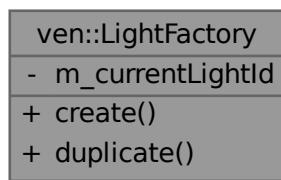
- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/[Light.hpp](#)

7.24 ven::LightFactory Class Reference

Class for light factory.

```
#include <Light.hpp>
```

Collaboration diagram for ven::LightFactory:



Static Public Member Functions

- static std::unique_ptr< [Light](#) > [create](#) (const [Transform3D](#) &transform=[DEFAULT_TRANSFORM](#), glm::vec4 color=[DEFAULT_LIGHT_COLOR](#))
- static std::unique_ptr< [Light](#) > [duplicate](#) (const [Light](#) &cpyLight)

Static Private Attributes

- static unsigned int [m_currentLightId](#) = 0

7.24.1 Detailed Description

Class for light factory.

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

7.24.2 Member Function Documentation

7.24.2.1 create()

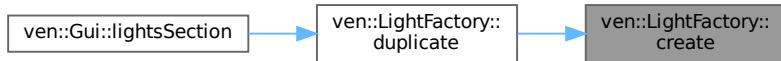
```
std::unique_ptr< ven::Light > ven::LightFactory::create (
    const Transform3D & transform = DEFAULT_TRANSFORM,
    glm::vec4 color = DEFAULT_LIGHT_COLOR) [static]
```

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

References [m_currentLightId](#), and [ven::MAX_LIGHTS](#).

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

Here is the caller graph for this function:



7.24.2.2 duplicate()

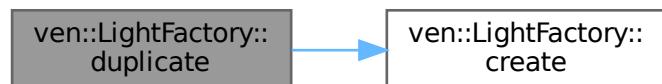
```
static std::unique_ptr< Light > ven::LightFactory::duplicate (
    const Light & cpyLight) [inline], [static]
```

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

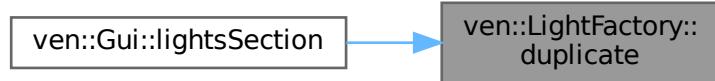
References [ven::Light::color](#), [create\(\)](#), and [ven::Light::transform](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.3 Member Data Documentation

7.24.3.1 m_currentLightId

```
unsigned int ven::LightFactory::m_currentLightId = 0 [static], [private]
```

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

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

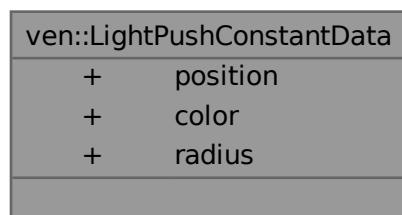
The documentation for this class was generated from the following files:

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

7.25 ven::LightPushConstantData Struct Reference

```
#include <PointLight.hpp>
```

Collaboration diagram for ven::LightPushConstantData:



Public Attributes

- `glm::vec4 position {}`
- `glm::vec4 color {}`
- `float radius`

7.25.1 Detailed Description

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

7.25.2 Member Data Documentation

7.25.2.1 `color`

```
glm::vec4 ven::LightPushConstantData::color {}
```

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

7.25.2.2 `position`

```
glm::vec4 ven::LightPushConstantData::position {}
```

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

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

7.25.2.3 `radius`

```
float ven::LightPushConstantData::radius
```

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

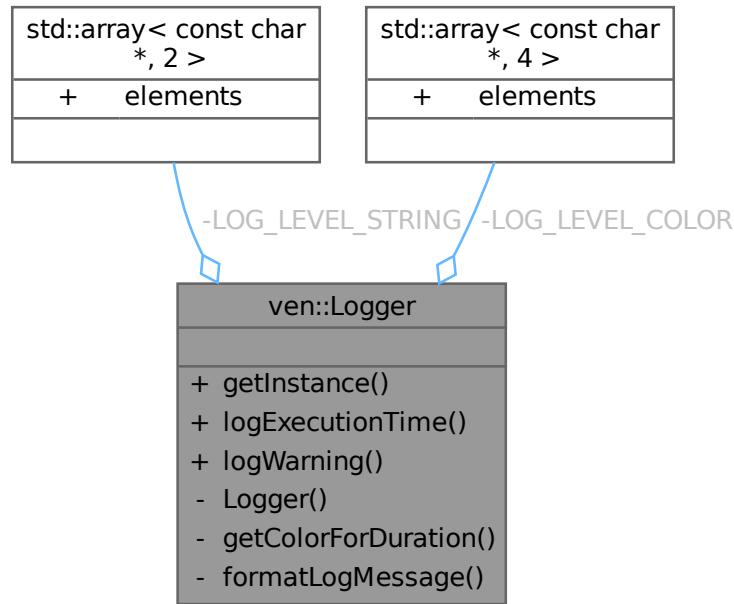
The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[PointLight.hpp](#)

7.26 ven::Logger Class Reference

```
#include <Logger.hpp>
```

Collaboration diagram for ven::Logger:



Static Public Member Functions

- static [Logger & getInstance \(\)](#)
- template<typename Func >
static void [logExecutionTime \(const std::string &message, Func &&func\)](#)
- static void [logWarning \(const std::string &message\)](#)

Private Member Functions

- [Logger \(\)](#)

Static Private Member Functions

- static const char * [getColorForDuration \(const float duration\)](#)
- static std::string [formatLogMessage \(LogLevel level, const std::string &message\)](#)

Static Private Attributes

- static constexpr std::array< const char *, 2 > [LOG_LEVEL_STRING = {"INFO", "WARNING"}](#)
- static constexpr std::array< const char *, 4 > [LOG_LEVEL_COLOR = {"\033\[31m", "\033\[32m", "\033\[33m", "\033\[0m\n"}](#)

7.26.1 Detailed Description

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

7.26.2 Constructor & Destructor Documentation

7.26.2.1 Logger()

```
ven::Logger::Logger () [private]
```

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

7.26.3 Member Function Documentation

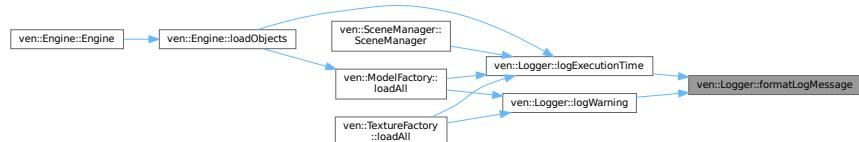
7.26.3.1 formatLogMessage()

```
std::string ven::Logger::formatLogMessage (
    LogLevel level,
    const std::string & message) [static], [nodiscard], [private]
```

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

Referenced by [logExecutionTime\(\)](#), and [logWarning\(\)](#).

Here is the caller graph for this function:



7.26.3.2 getColorForDuration()

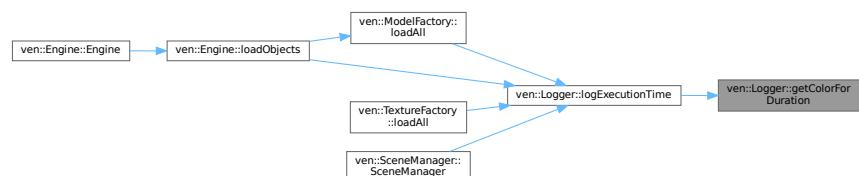
```
static const char * ven::Logger::getColorForDuration (
    const float duration) [inline], [static], [private]
```

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

References [LOG_LEVEL_COLOR](#).

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

Here is the caller graph for this function:



7.26.3.3 getInstance()

```
static Logger & ven::Logger::getInstance () [inline], [static]
```

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

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

Here is the caller graph for this function:



7.26.3.4 logExecutionTime()

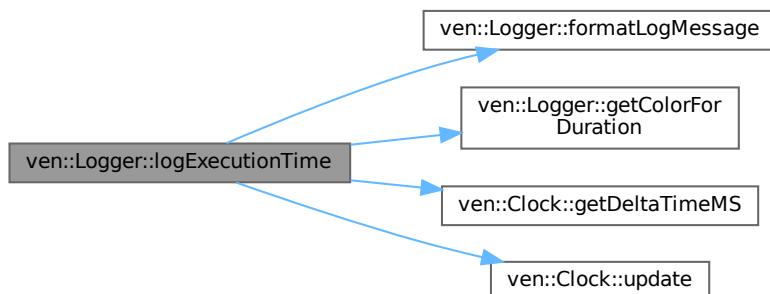
```
template<typename Func >
static void ven::Logger::logExecutionTime (
    const std::string & message,
    Func && func) [inline], [static]
```

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

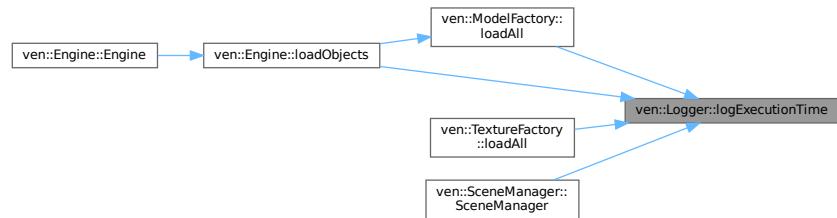
References [formatLogMessage\(\)](#), [getColorForDuration\(\)](#), [ven::Clock::getDeltaTimeMS\(\)](#), [ven::INFO](#), [LOG_LEVEL_COLOR](#), and [ven::Clock::update\(\)](#).

Referenced by [ven::ModelFactory::loadAll\(\)](#), [ven::TextureFactory::loadAll\(\)](#), [ven::Engine::loadObjects\(\)](#), and [ven::SceneManager::SceneManager\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.3.5 logWarning()

```
static void ven::Logger::logWarning (
    const std::string & message) [inline], [static]
```

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

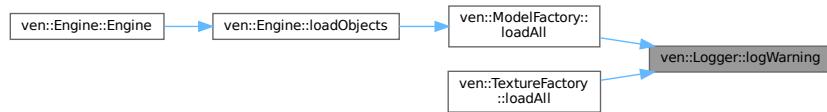
References [formatLogMessage\(\)](#), [LOG_LEVEL_COLOR](#), and [ven::WARNING](#).

Referenced by [ven::ModelFactory::loadAll\(\)](#), and [ven::TextureFactory::loadAll\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.4 Member Data Documentation

7.26.4.1 LOG_LEVEL_COLOR

```
std::array<const char*, 4> ven::Logger::LOG_LEVEL_COLOR = {"\033[31m", "\033[32m", "\033[33m",
"\033[0m\n"} [static], [constexpr], [private]
```

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

Referenced by [getColorForDuration\(\)](#), [logExecutionTime\(\)](#), and [logWarning\(\)](#).

7.26.4.2 LOG_LEVEL_STRING

```
std::array<const char*, 2> ven::Logger::LOG_LEVEL_STRING = {"INFO", "WARNING"} [static],  
[constexpr], [private]
```

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

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

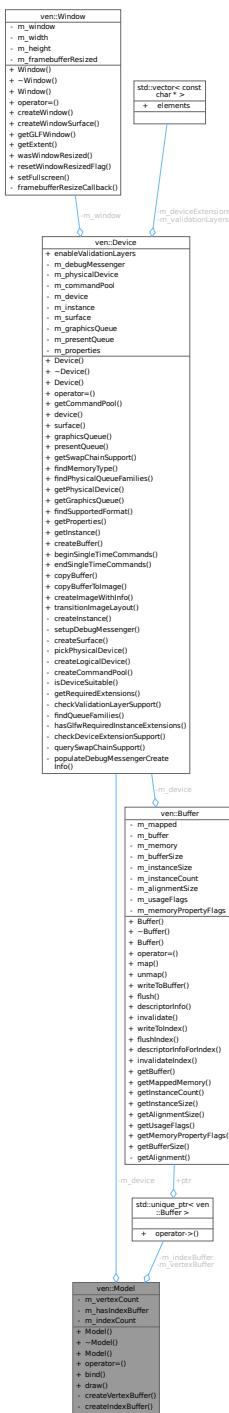
- /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp
- /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp

7.27 ven::Model Class Reference

Class for model.

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

Collaboration diagram for ven::Model:



Classes

- struct [Builder](#)
- struct [Vertex](#)

Public Member Functions

- [Model](#) ([Device](#) &device, const [Builder](#) &builder)

- `~Model ()=default`
- `Model (const Model &)=delete`
- `void operator= (const Model &)=delete`
- `void bind (VkCommandBuffer commandBuffer) const`
- `void draw (VkCommandBuffer commandBuffer) const`

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`

7.27.1 Detailed Description

Class for model.

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

7.27.2 Constructor & Destructor Documentation

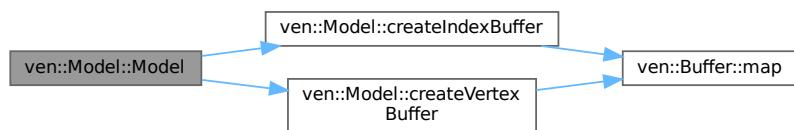
7.27.2.1 Model() [1/2]

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

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

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

Here is the call graph for this function:



7.27.2.2 ~Model()

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

7.27.2.3 Model() [2/2]

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

7.27.3 Member Function Documentation

7.27.3.1 bind()

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

Definition at line 73 of file [model.cpp](#).

7.27.3.2 createIndexBuffer()

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

Definition at line 43 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:



7.27.3.3 createVertexBuffer()

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

Definition at line 26 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:



7.27.3.4 draw()

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

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

7.27.3.5 operator=()

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

7.27.4 Member Data Documentation

7.27.4.1 m_device

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

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

7.27.4.2 m_hasIndexBuffer

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

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

7.27.4.3 m_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer [private]
```

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

7.27.4.4 m_indexCount

```
uint32_t ven::Model::m_indexCount [private]
```

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

7.27.4.5 m_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer [private]
```

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

7.27.4.6 m_vertexCount

```
uint32_t ven::Model::m_vertexCount [private]
```

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

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

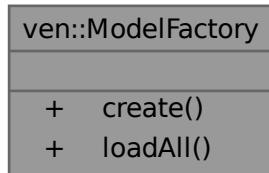
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

7.28 ven::ModelFactory Class Reference

Class for [Model](#) factory.

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

Collaboration diagram for ven::ModelFactory:



Static Public Member Functions

- static std::unique_ptr< [Model](#) > [create](#) ([Device](#) &device, const std::string &filepath)
- static std::unordered_map< std::string, std::shared_ptr< [Model](#) > > [loadAll](#) ([Device](#) &device, const std::string &folderPath)

7.28.1 Detailed Description

Class for [Model](#) factory.

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

7.28.2 Member Function Documentation

7.28.2.1 [create\(\)](#)

```
std::unique_ptr< ven::Model > ven::ModelFactory::create (
    Device & device,
    const std::string & filepath) [static]
```

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

References [ven::Model::Builder::loadModel\(\)](#).

Here is the call graph for this function:



7.28.2.2 loadAll()

```
std::unordered_map< std::string, std::shared_ptr< ven::Model > > ven::ModelFactory::loadAll (
    Device & device,
    const std::string & folderPath) [static]
```

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

References [ven::Logger::logExecutionTime\(\)](#), and [ven::Logger::logWarning\(\)](#).

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

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:

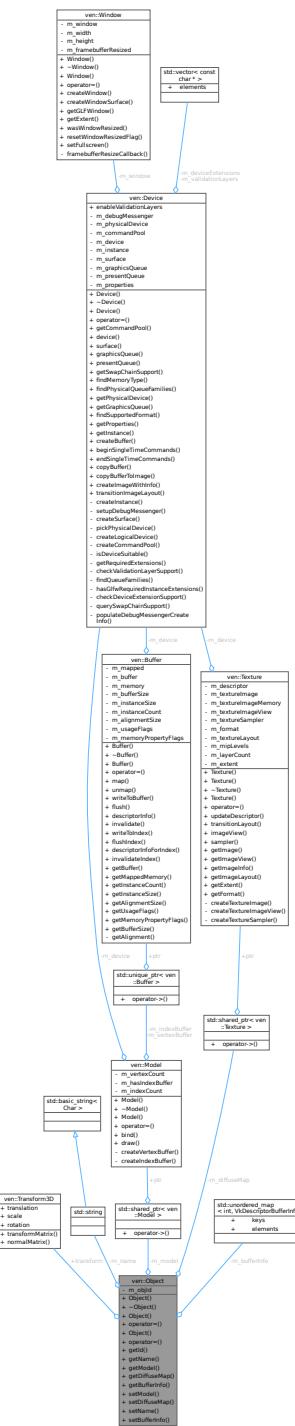
- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Model.cpp](#)

7.29 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 ()=default`
- `Object (const Object &)=delete`
- `Object & operator= (const Object &)=delete`
- `Object (Object &&)=default`
- `Object & operator= (Object &&)=default`
- `unsigned int getId () const`
- `std::string getName () const`
- `std::shared_ptr< Model > getModel () const`
- `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`

7.29.1 Detailed Description

Class for object.

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

7.29.2 Member Typedef Documentation

7.29.2.1 Map

```
using ven::Object::Map = std::unordered_map<unsigned int, Object>
```

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

7.29.3 Constructor & Destructor Documentation

7.29.3.1 Object() [1/3]

```
ven::Object::Object (
    const unsigned int objId) [inline], [explicit]
```

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

7.29.3.2 ~Object()

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

7.29.3.3 Object() [2/3]

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

7.29.3.4 Object() [3/3]

```
ven::Object::Object (
    Object && ) [default]
```

7.29.4 Member Function Documentation

7.29.4.1 getBufferInfo()

```
VkDescriptorBufferInfo ven::Object::getBufferInfo (
    const int frameIndex) const [inline], [nodiscard]
```

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

References [m_bufferInfo](#).

7.29.4.2 getDiffuseMap()

```
std::shared_ptr< Texture > ven::Object::getDiffuseMap () const [inline], [nodiscard]
```

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

References [m_diffuseMap](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



7.29.4.3 getId()

```
unsigned int ven::Object::getId () const [inline], [nodiscard]
```

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

References [m_objId](#).

7.29.4.4 getModel()

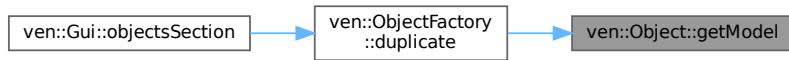
```
std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]
```

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

References [m_model](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



7.29.4.5 getName()

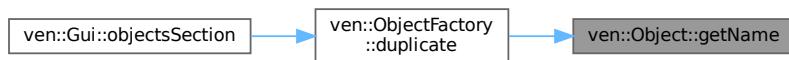
```
std::string ven::Object::getName () const [inline], [nodiscard]
```

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

References [m_name](#).

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

Here is the caller graph for this function:



7.29.4.6 operator=() [1/2]

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

7.29.4.7 operator=() [2/2]

```
Object & ven::Object::operator= (
    Object &&) [default]
```

7.29.4.8 setBufferInfo()

```
void ven::Object::setBufferInfo (
    const int frameIndex,
    const VkDescriptorBufferInfo & info) [inline]
```

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

References [m_bufferInfo](#).

7.29.4.9 setDiffuseMap()

```
void ven::Object::setDiffuseMap (
    const std::shared_ptr< Texture > & diffuseMap) [inline]
```

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

References [m_diffuseMap](#).

7.29.4.10 setModel()

```
void ven::Object::setModel (
    const std::shared_ptr< Model > & model) [inline]
```

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

References [m_model](#).

7.29.4.11 setName()

```
void ven::Object::setName (
    const std::string & name) [inline]
```

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

References [m_name](#).

7.29.5 Member Data Documentation

7.29.5.1 m_bufferInfo

```
std::unordered_map<int, VkDescriptorBufferInfo> ven::Object::m_bufferInfo [private]
```

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

Referenced by [getBufferInfo\(\)](#), and [setBufferInfo\(\)](#).

7.29.5.2 m_diffuseMap

```
std::shared_ptr<Texture> ven::Object::m_diffuseMap = nullptr [private]
```

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

Referenced by [getDiffuseMap\(\)](#), and [setDiffuseMap\(\)](#).

7.29.5.3 m_model

```
std::shared_ptr<Model> ven::Object::m_model = nullptr [private]
```

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

Referenced by [getModel\(\)](#), and [setModel\(\)](#).

7.29.5.4 m_name

```
std::string ven::Object::m_name [private]
```

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

Referenced by [getName\(\)](#), and [setName\(\)](#).

7.29.5.5 m_objId

```
unsigned int ven::Object::m_objId [private]
```

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

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

7.29.5.6 transform

```
Transform3D ven::Object::transform {}
```

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

Referenced by [ven::ObjectFactory::duplicate\(\)](#).

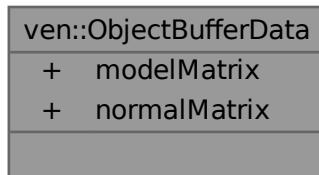
The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/[Object.hpp](#)

7.30 ven::ObjectBufferData Struct Reference

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

Collaboration diagram for ven::ObjectBufferData:



Public Attributes

- `glm::mat4 modelMatrix {1.F}`
- `glm::mat4 normalMatrix {1.F}`

7.30.1 Detailed Description

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

7.30.2 Member Data Documentation

7.30.2.1 modelMatrix

```
glm::mat4 ven::ObjectBufferData::modelMatrix {1.F}
```

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

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

7.30.2.2 normalMatrix

```
glm::mat4 ven::ObjectBufferData::normalMatrix {1.F}
```

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

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

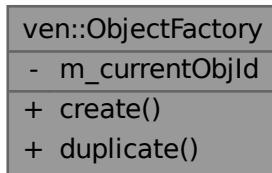
- [/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp](#)

7.31 ven::ObjectFactory Class Reference

Class for object factory.

```
#include <Object.hpp>
```

Collaboration diagram for ven::ObjectFactory:



Static Public Member Functions

- static std::unique_ptr<[Object](#)> [create](#) (const std::shared_ptr<[Texture](#)> &texture, const std::shared_ptr<[Model](#)> &model, const std::string &name, const [Transform3D](#) &transform)
- static std::unique_ptr<[Object](#)> [duplicate](#) (const [Object](#) &objSrc)

Static Private Attributes

- static unsigned int [m_currentObjId](#) = 0

7.31.1 Detailed Description

Class for object factory.

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

7.31.2 Member Function Documentation

7.31.2.1 [create\(\)](#)

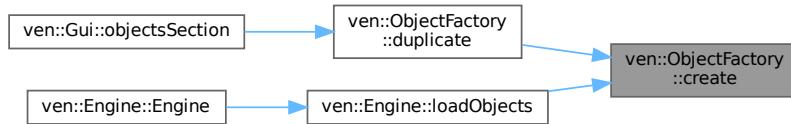
```
std::unique_ptr<ven::Object> ven::ObjectFactory::create (
    const std::shared_ptr<Texture> & texture,
    const std::shared_ptr<Model> & model,
    const std::string & name,
    const Transform3D & transform) [static]
```

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

References [m_currentObjId](#), and [ven::MAX_OBJECTS](#).

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

Here is the caller graph for this function:



7.31.2.2 duplicate()

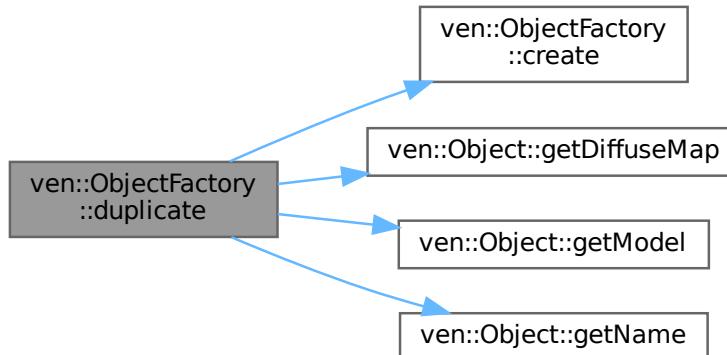
```
static std::unique_ptr< Object > ven::ObjectFactory::duplicate (
    const Object & objSrc) [inline], [static]
```

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

References [create\(\)](#), [ven::Object::getDiffuseMap\(\)](#), [ven::Object::getModel\(\)](#), [ven::Object::getName\(\)](#), and [ven::Object::transform](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.31.3 Member Data Documentation

7.31.3.1 m_currentObjId

```
unsigned int ven::ObjectFactory::m_currentObjId = 0 [static], [private]
```

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

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

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

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

7.32 ven::ObjectPushConstantData Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::ObjectPushConstantData:

| |
|-----------------------------|
| ven::ObjectPushConstantData |
| + modelMatrix |
| + normalMatrix |

Public Attributes

- `glm::mat4 modelMatrix {}`
- `glm::mat4 normalMatrix {}`

7.32.1 Detailed Description

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

7.32.2 Member Data Documentation

7.32.2.1 modelMatrix

```
glm::mat4 ven::ObjectPushConstantData::modelMatrix {}
```

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

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

7.32.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

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

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

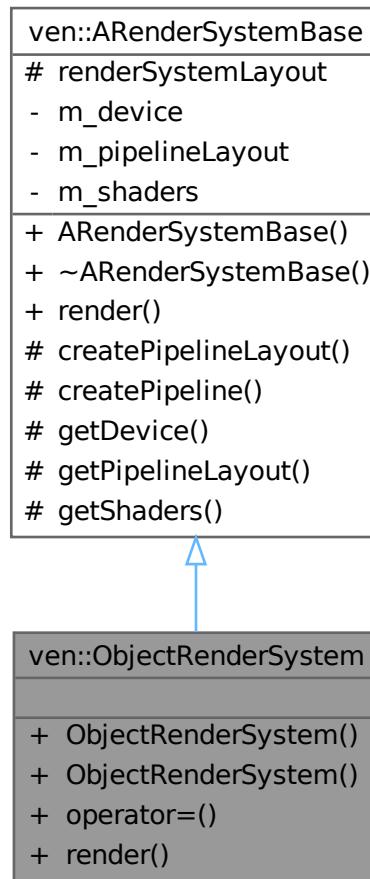
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[Object.hpp](#)

7.33 ven::ObjectRenderSystem Class Reference

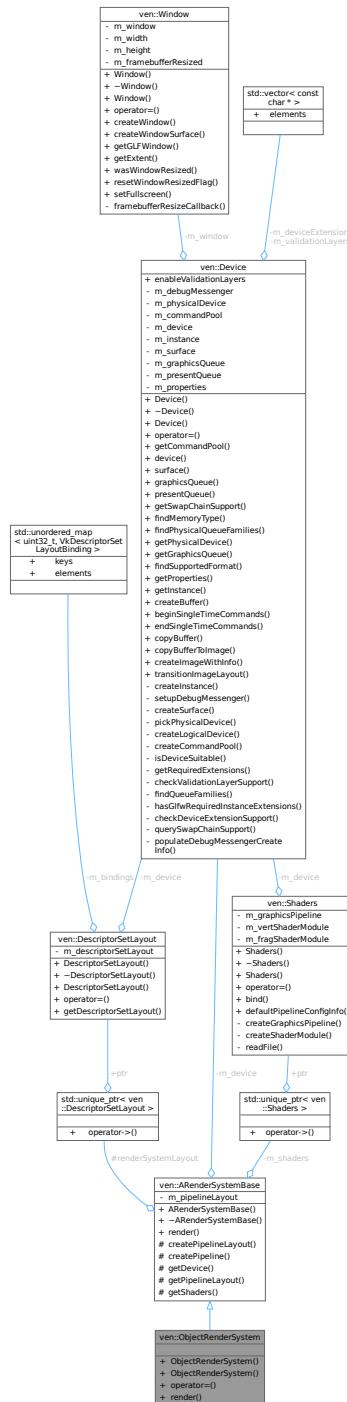
Class for object render system.

```
#include <Object.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](#)

7.33.1 Detailed Description

Class for object render system.

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

7.33.2 Constructor & Destructor Documentation

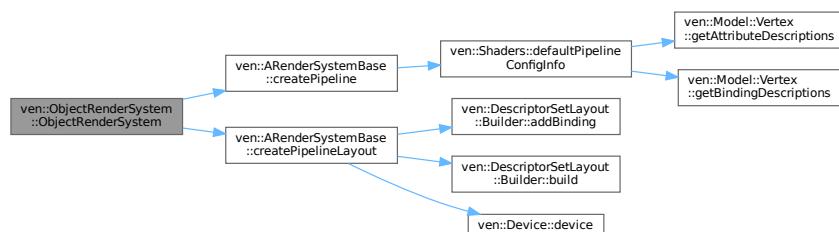
7.33.2.1 ObjectRenderSystem() [1/2]

```
ven::ObjectRenderSystem::ObjectRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalsetLayout) [inline], [explicit]
```

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

References [ven::ARenderSystemBase::createPipeline\(\)](#), [ven::ARenderSystemBase::createPipelineLayout\(\)](#), and [ven::SHADERS_BIN_PATH](#).

Here is the call graph for this function:



7.33.2.2 ObjectRenderSystem() [2/2]

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

7.33.3 Member Function Documentation

7.33.3.1 operator=()

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

7.33.3.2 render()

```
void ven::ObjectRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

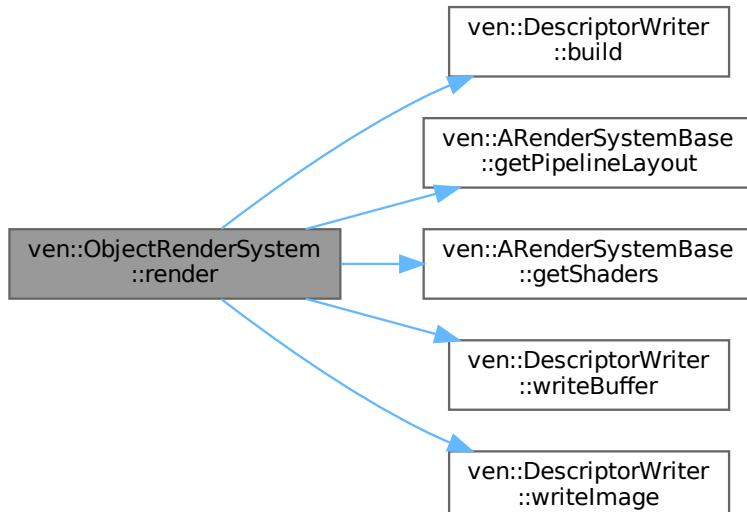
Implements [ven::ARenderSystemBase](#).

Definition at line 6 of file [object.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:



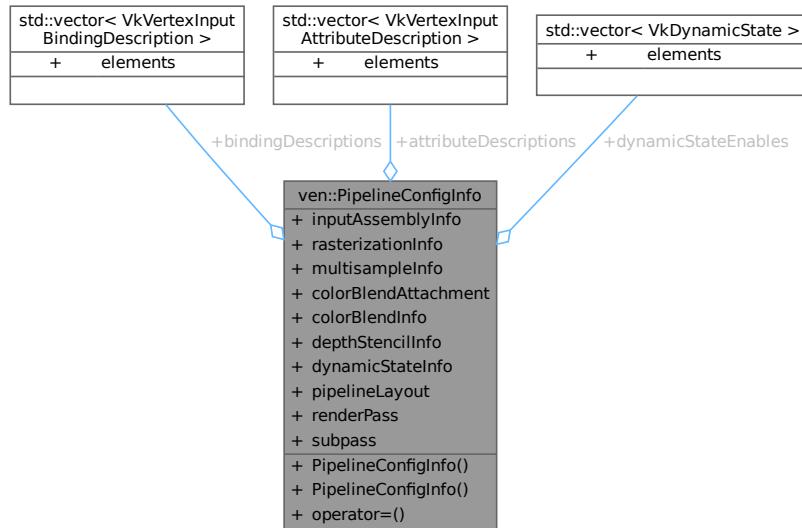
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/Object.hpp
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp

7.34 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

7.34.1 Detailed Description

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

7.34.2 Constructor & Destructor Documentation

7.34.2.1 PipelineConfigInfo() [1/2]

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

7.34.2.2 PipelineConfigInfo() [2/2]

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

7.34.3 Member Function Documentation

7.34.3.1 operator=()

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

7.34.4 Member Data Documentation

7.34.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

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

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

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

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.34.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#), and [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

7.34.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

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

Referenced by [ven::Shaders::createGraphicsPipeline\(\)](#).

7.34.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 32 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/Gfx/[Shaders.hpp](#)

7.35 ven::PointLightData Struct Reference

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

Collaboration diagram for ven::PointLightData:



Public Attributes

- glm::vec4 [position](#) {}
- glm::vec4 [color](#) {}
- float [shininess](#) {32.F}
- float [padding](#) [3]

7.35.1 Detailed Description

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

7.35.2 Member Data Documentation

7.35.2.1 [color](#)

```
glm::vec4 ven::PointLightData::color {}
```

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

7.35.2.2 [padding](#)

```
float ven::PointLightData::padding[3]
```

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

7.35.2.3 position

```
glm::vec4 ven::PointLightData::position {}
```

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

7.35.2.4 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:

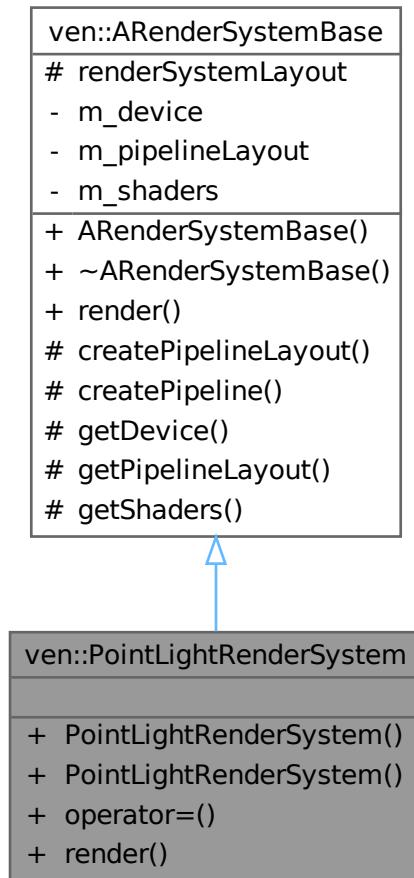
- [/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp](#)

7.36 ven::PointLightRenderSystem Class Reference

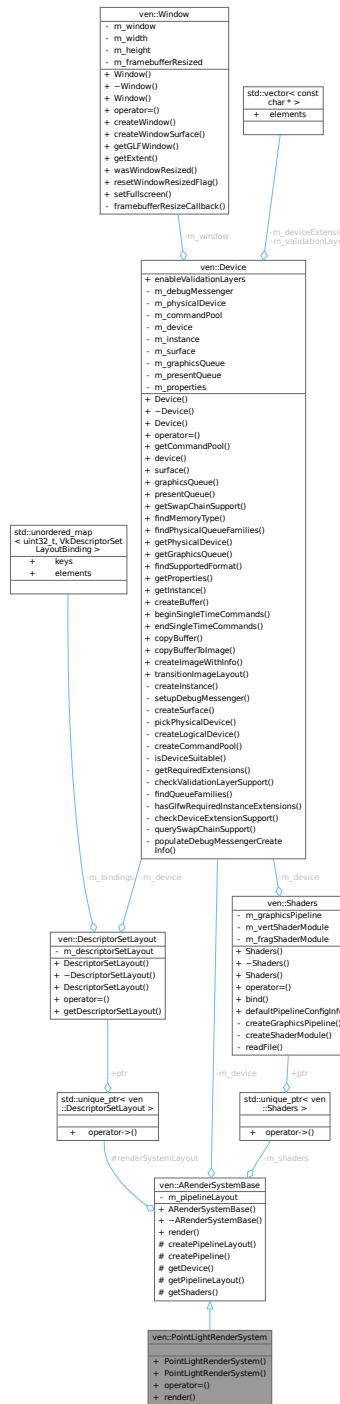
Class for point light system.

```
#include <PointLight.hpp>
```

Inheritance diagram for ven::PointLightRenderSystem:



Collaboration diagram for ven::PointLightRenderSystem:



Public Member Functions

- **PointLightRenderSystem** (`Device &device, const VkRenderPass renderPass, const VkDescriptorSetLayout globalsetLayout`)
- **PointLightRenderSystem** (`const PointLightRenderSystem &)=delete`
- **PointLightRenderSystem & operator=** (`(const PointLightRenderSystem &)=delete`)
- **void render** (`(const FrameInfo &frameInfo) const override`)

Public Member Functions inherited from `ven::ARenderSystemBase`

- `ARenderSystemBase (Device &device)`
- virtual `~ARenderSystemBase ()`

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`

7.36.1 Detailed Description

Class for point light system.

Definition at line 24 of file `PointLight.hpp`.

7.36.2 Constructor & Destructor Documentation

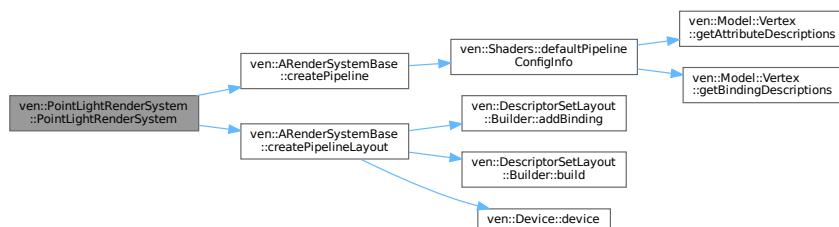
7.36.2.1 `PointLightRenderSystem() [1/2]`

```
ven::PointLightRenderSystem::PointLightRenderSystem (
    Device & device,
    const VkRenderPass renderPass,
    const VkDescriptorSetLayout globalsetLayout) [inline], [explicit]
```

Definition at line 28 of file `PointLight.hpp`.

References `ven::ARenderSystemBase::createPipeline()`, `ven::ARenderSystemBase::createPipelineLayout()`, and `ven::SHADERS_BIN_PATH`.

Here is the call graph for this function:



7.36.2.2 PointLightRenderSystem() [2/2]

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

7.36.3 Member Function Documentation

7.36.3.1 operator=()

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

7.36.3.2 render()

```
void ven::PointLightRenderSystem::render (
    const FrameInfo & frameInfo) const [override], [virtual]
```

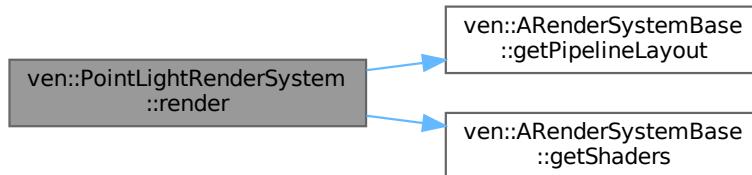
Implements [ven::ARenderSystemBase](#).

Definition at line 5 of file [pointLight.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:



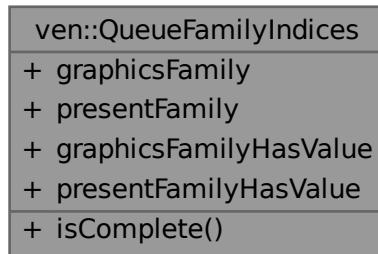
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/[PointLight.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/[pointLight.cpp](#)

7.37 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

7.37.1 Detailed Description

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

7.37.2 Member Function Documentation

7.37.2.1 [isComplete\(\)](#)

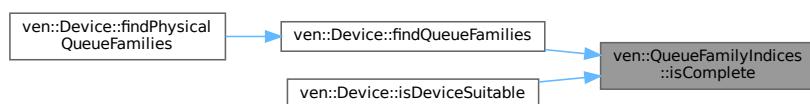
```
bool ven::QueueFamilyIndices::isComplete () const [inline], [nodiscard]
```

Definition at line [27](#) 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:



7.37.3 Member Data Documentation

7.37.3.1 `graphicsFamily`

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

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

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.37.3.2 `graphicsFamilyHasValue`

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

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

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

7.37.3.3 `presentFamily`

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

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

Referenced by [ven::Device::findQueueFamilies\(\)](#).

7.37.3.4 `presentFamilyHasValue`

```
bool ven::QueueFamilyIndices::presentFamilyHasValue = false
```

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

Referenced by [ven::Device::findQueueFamilies\(\)](#), and [isComplete\(\)](#).

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

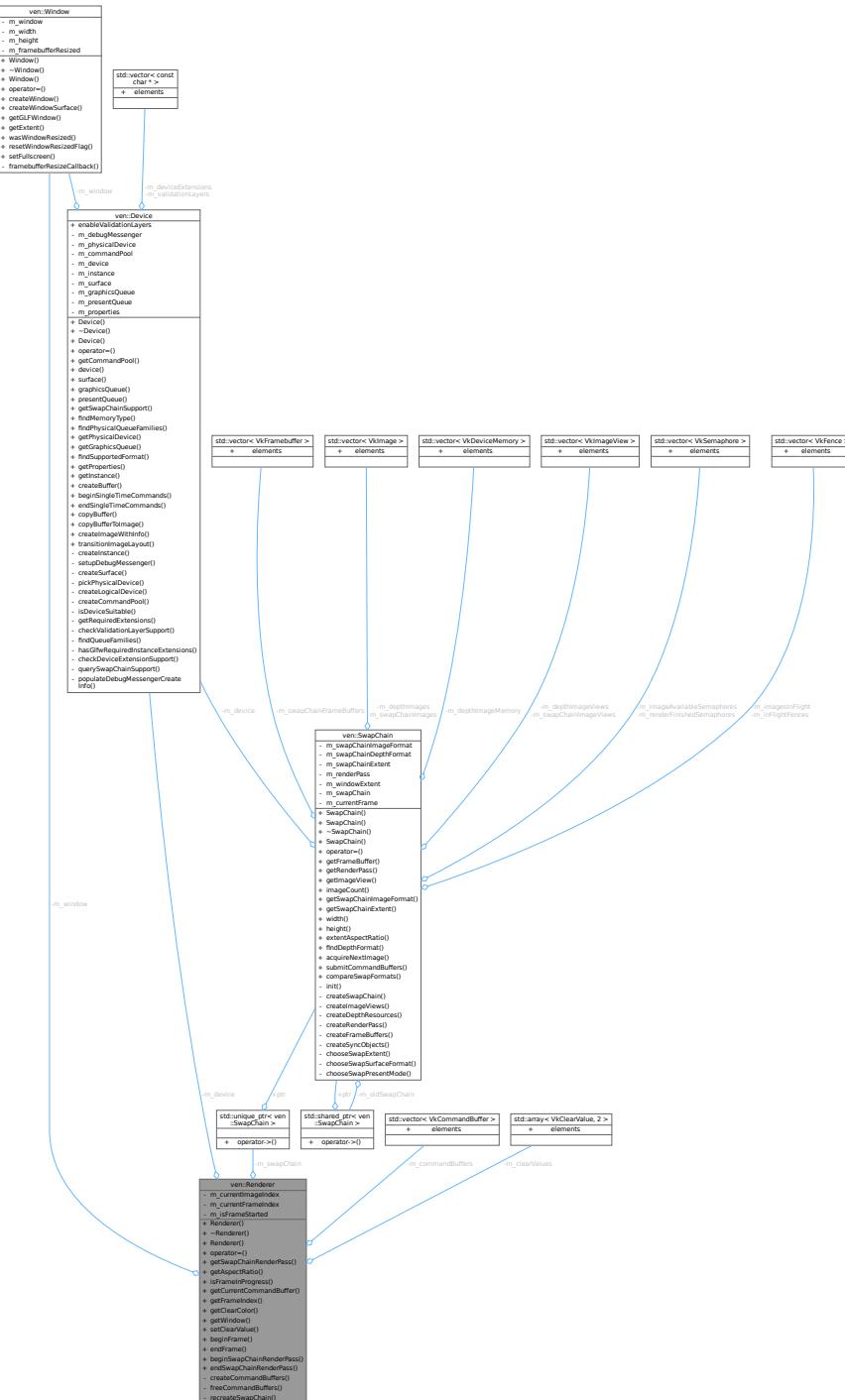
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Device.hpp](#)

7.38 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 setClearColor (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}`

7.38.1 Detailed Description

Class for renderer.

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

7.38.2 Constructor & Destructor Documentation

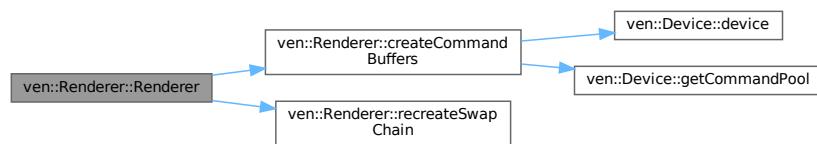
7.38.2.1 Renderer() [1/2]

```
ven::Renderer::Renderer (
    Window & window,
    Device & device) [inline]
```

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

References [createCommandBuffers\(\)](#), and [recreateSwapChain\(\)](#).

Here is the call graph for this function:



7.38.2.2 ~Renderer()

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

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

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

Here is the call graph for this function:



7.38.2.3 Renderer() [2/2]

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

7.38.3 Member Function Documentation

7.38.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

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

7.38.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

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

7.38.3.3 createCommandBuffers()

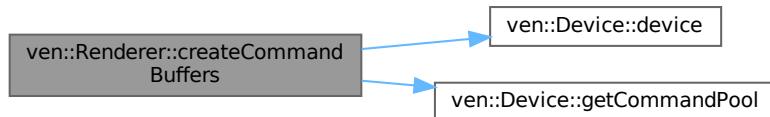
```
void ven::Renderer::createCommandBuffers () [private]
```

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:



7.38.3.4 `endFrame()`

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file [renderer.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.38.3.5 `endSwapChainRenderPass()`

```
void ven::Renderer::endSwapChainRenderPass (
    VkCommandBuffer commandBuffer) const
```

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

7.38.3.6 `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:



7.38.3.7 `getAspectRatio()`

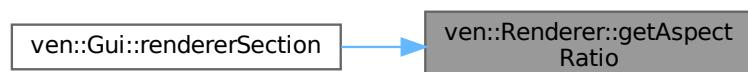
```
float ven::Renderer::getAspectRatio () const [inline], [nodiscard]
```

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

References [m_swapChain](#).

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

Here is the caller graph for this function:



7.38.3.8 getClearColor()

```
std::array< float, 4 > ven::Renderer::getClearColor () const [inline], [nodiscard]
```

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

References [m_clearValues](#).

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

Here is the caller graph for this function:



7.38.3.9 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const [inline], [nodiscard]
```

Definition at line 37 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:



7.38.3.10 getFrameIndex()

```
unsigned long ven::Renderer::getFrameIndex () const [inline], [nodiscard]
```

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

References [isFrameInProgress\(\)](#), and [m_currentFrameIndex](#).

Here is the call graph for this function:



7.38.3.11 getSwapChainRenderPass()

```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

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

References [m_swapChain](#).

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

Here is the caller graph for this function:



7.38.3.12 getWindow()

```
Window & ven::Renderer::getWindow () const [inline], [nodiscard]
```

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

References [m_window](#).

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

Here is the caller graph for this function:



7.38.3.13 isFrameInProgress()

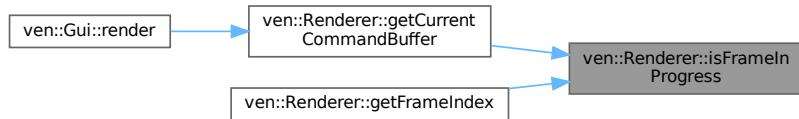
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

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

References [m_isFrameStarted](#).

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

Here is the caller graph for this function:



7.38.3.14 operator=(*)

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

7.38.3.15 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:



7.38.3.16 setClearValue()

```
void ven::Renderer::setClearValue (
    const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
    const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) [inline]
```

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

References [m_clearValues](#).

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

Here is the caller graph for this function:



7.38.4 Member Data Documentation

7.38.4.1 m_clearValues

```
std::array<VkClearColorValue, 2> ven::Renderer::m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F} [private]
```

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

Referenced by [getClearColor\(\)](#), and [setClearValue\(\)](#).

7.38.4.2 m_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers [private]
```

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

Referenced by [createCommandBuffers\(\)](#), and [getCurrentCommandBuffer\(\)](#).

7.38.4.3 m_currentFrameIndex

```
unsigned long ven::Renderer::m_currentFrameIndex {0} [private]
```

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

Referenced by [getCurrentCommandBuffer\(\)](#), and [getFrameIndex\(\)](#).

7.38.4.4 m_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0} [private]
```

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

7.38.4.5 m_device

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

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

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

7.38.4.6 m_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false} [private]
```

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

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

7.38.4.7 m_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain [private]
```

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

Referenced by [getAspectRatio\(\)](#), and [getSwapChainRenderPass\(\)](#).

7.38.4.8 m_window

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

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

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

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

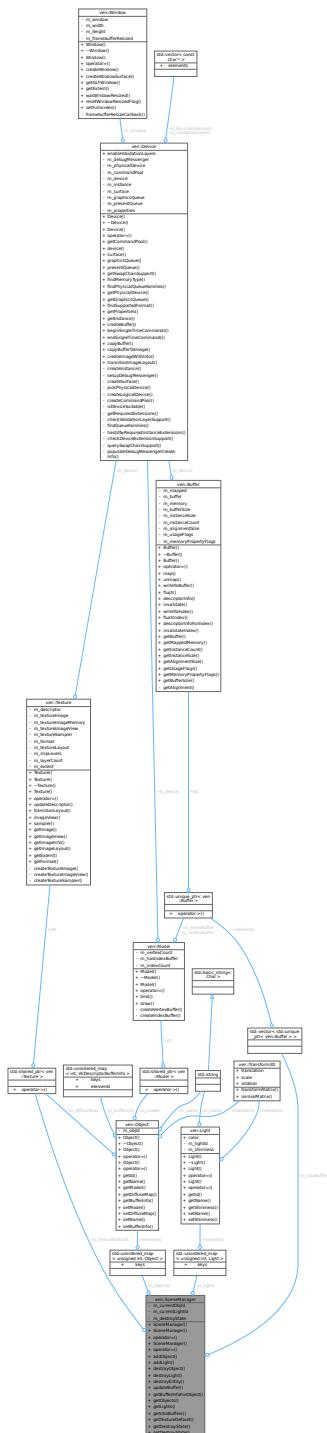
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Renderer.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[renderer.cpp](#)

7.39 ven::SceneManager Class Reference

Class for object manager.

```
#include <Manager.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`
- `void addObject (const std::unique_ptr< Object > &object)`
- `void addLight (const std::unique_ptr< Light > &light)`
- `void destroyObject (const unsigned int objectId)`
- `void destroyLight (const unsigned int lightId)`
- `void destroyEntity (std::vector< unsigned int > *objectIds, std::vector< unsigned int > *lightIds)`
- `void updateBuffer (GlobalUbo &ubo, unsigned long frameIndex, float frameTime)`
- `VkDescriptorBufferInfo getBufferInfoForObject (const int frameIndex, const unsigned int objectId) const`
- `Object::Map & getObjects ()`
- `Light::Map & getLights ()`
- `std::vector< std::unique_ptr< Buffer > > & getUboBuffers ()`
- `std::shared_ptr< Texture > getTextureDefault ()`
- `bool getDestroyState () const`
- `void setDestroyState (const bool state)`

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}`
- `bool m_destroyState {false}`

7.39.1 Detailed Description

Class for object manager.

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

7.39.2 Constructor & Destructor Documentation

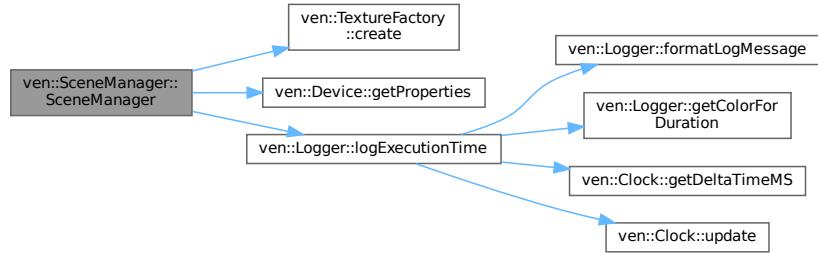
7.39.2.1 SceneManager() [1/3]

```
ven::SceneManager::SceneManager (
    Device & device) [explicit]
```

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

References `ven::TextureFactory::create()`, `ven::Device::getProperties()`, `ven::Logger::logExecutionTime()`, `m_textureDefault`, `m_uboBuffers`, and `ven::MAX_OBJECTS`.

Here is the call graph for this function:



7.39.2.2 SceneManager() [2/3]

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

7.39.2.3 SceneManager() [3/3]

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

7.39.3 Member Function Documentation

7.39.3.1 addLight()

```
void ven::SceneManager::addLight (
    const std::unique_ptr< Light > & light) [inline]
```

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

References [m_lights](#).

7.39.3.2 addObject()

```
void ven::SceneManager::addObject (
    const std::unique_ptr< Object > & object) [inline]
```

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

References [m_objects](#).

7.39.3.3 `destroyEntity()`

```
void ven::SceneManager::destroyEntity (
    std::vector< unsigned int > * objectsIds,
    std::vector< unsigned int > * lightsIds)
```

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

7.39.3.4 `destroyLight()`

```
void ven::SceneManager::destroyLight (
    const unsigned int lightId) [inline]
```

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

References [m_lights](#).

7.39.3.5 `destroyObject()`

```
void ven::SceneManager::destroyObject (
    const unsigned int objectId) [inline]
```

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

References [m_objects](#).

7.39.3.6 `getBufferInfoForObject()`

```
VkDescriptorBufferInfo ven::SceneManager::getBufferInfoForObject (
    const int frameIndex,
    const unsigned int objectId) const [inline], [nodiscard]
```

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

References [m_uboBuffers](#).

7.39.3.7 `getDestroyState()`

```
bool ven::SceneManager::getDestroyState () const [inline], [nodiscard]
```

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

References [m_destroyState](#).

7.39.3.8 getLights()

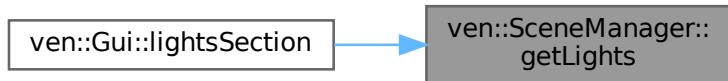
```
Light::Map & ven::SceneManager::getLights () [inline]
```

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

References [m_lights](#).

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

Here is the caller graph for this function:



7.39.3.9 getObjects()

```
Object::Map & ven::SceneManager::getObjects () [inline]
```

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

References [m_objects](#).

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

Here is the caller graph for this function:



7.39.3.10 getTextureDefault()

```
std::shared_ptr< Texture > ven::SceneManager::getTextureDefault () [inline]
```

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

References [m_textureDefault](#).

7.39.3.11 getUboBuffers()

```
std::vector< std::unique_ptr< Buffer > > & ven::SceneManager::getUboBuffers () [inline]
```

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

References [m_uboBuffers](#).

7.39.3.12 operator=(1/2)

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

7.39.3.13 operator=(2/2)

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

7.39.3.14 setDestroyState()

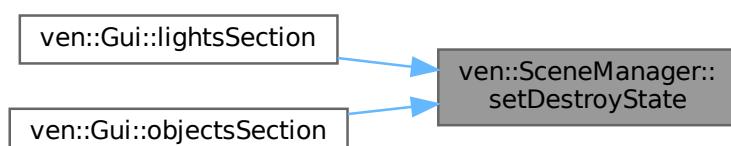
```
void ven::SceneManager::setDestroyState (
    const bool state) [inline]
```

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

References [m_destroyState](#).

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

Here is the caller graph for this function:



7.39.3.15 updateBuffer()

```
void ven::SceneManager::updateBuffer (
    GlobalUbo & ubo,
    unsigned long frameIndex,
    float frameTime)
```

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

References [ven::ObjectBufferData::modelMatrix](#), [ven::GlobalUbo::numLights](#), and [ven::GlobalUbo::pointLights](#).

7.39.4 Member Data Documentation

7.39.4.1 m_currentLightId

```
unsigned int ven::SceneManager::m_currentLightId {0} [private]
```

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

7.39.4.2 m_currentObjId

```
unsigned int ven::SceneManager::m_currentObjId {0} [private]
```

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

7.39.4.3 m_destroyState

```
bool ven::SceneManager::m_destroyState {false} [private]
```

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

Referenced by [getDestroyState\(\)](#), and [setDestroyState\(\)](#).

7.39.4.4 m_lights

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

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

Referenced by [addLight\(\)](#), [destroyLight\(\)](#), and [getLights\(\)](#).

7.39.4.5 m_objects

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

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

Referenced by [addObject\(\)](#), [destroyObject\(\)](#), and [getObjects\(\)](#).

7.39.4.6 m_textureDefault

```
std::shared_ptr<Texture> ven::SceneManager::m_textureDefault [private]
```

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

Referenced by [getTextureDefault\(\)](#), and [SceneManager\(\)](#).

7.39.4.7 m_uboBuffers

```
std::vector<std::unique_ptr<Buffer>> ven::SceneManager::m_uboBuffers {MAX_FRAMES_IN_FLIGHT}  
[private]
```

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

Referenced by [getBufferInfoForObject\(\)](#), [getUboBuffers\(\)](#), and [SceneManager\(\)](#).

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

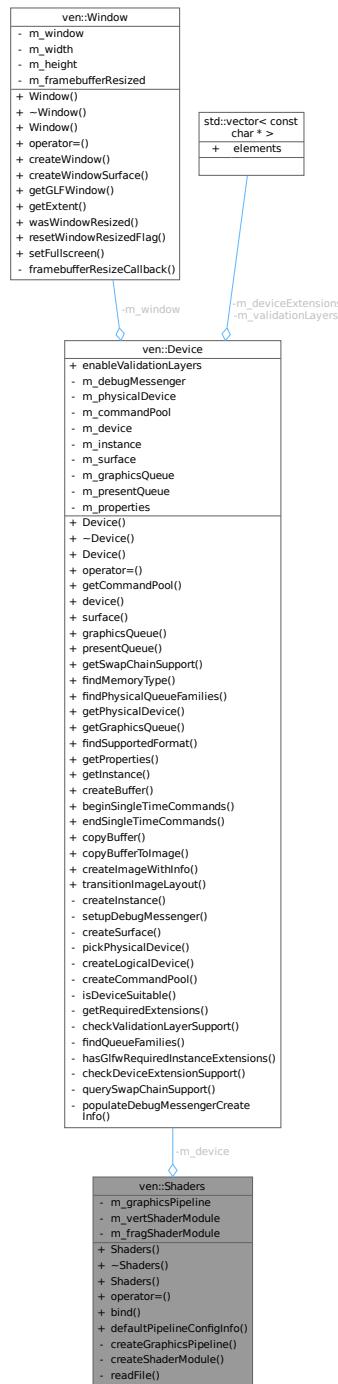
- /home/runner/work/VEngine/VEngine/include/VEngine/Scene/[Manager.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Scene/[manager.cpp](#)

7.40 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}](#)

7.40.1 Detailed Description

Class for shaders.

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

7.40.2 Constructor & Destructor Documentation**7.40.2.1 Shaders() [1/2]**

```
ven::Shaders::Shaders (
    Device & device,
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [inline]
```

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

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

Here is the call graph for this function:



7.40.2.2 ~Shaders()

```
ven::Shaders::~Shaders ()
```

Definition at line 7 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:



7.40.2.3 Shaders() [2/2]

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

7.40.3 Member Function Documentation

7.40.3.1 bind()

```
void ven::Shaders::bind (
    const VkCommandBuffer commandBuffer) const [inline]
```

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

References [m_graphicsPipeline](#).

7.40.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
    const std::string & vertFilepath,
    const std::string & fragFilepath,
    const PipelineConfigInfo & configInfo) [private]
```

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

References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorAttachmentCount](#), [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:



7.40.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
    const std::vector< char > & code,
    VkShaderModule * shaderModule) const [private]
```

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

7.40.3.4 defaultPipelineConfigInfo()

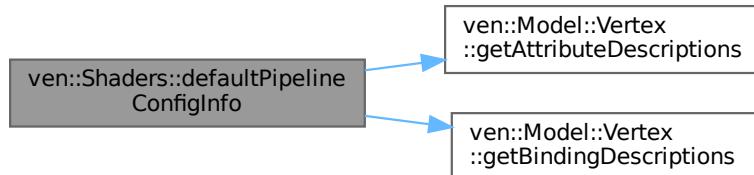
```
void ven::Shaders::defaultPipelineConfigInfo (
    PipelineConfigInfo & configInfo) [static]
```

Definition at line 108 of file [shaders.cpp](#).

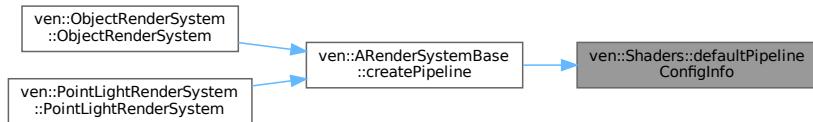
References [ven::PipelineConfigInfo::attributeDescriptions](#), [ven::PipelineConfigInfo::bindingDescriptions](#), [ven::PipelineConfigInfo::colorBlendInfo](#), [ven::PipelineConfigInfo::depthStencilInfo](#), [ven::PipelineConfigInfo::dynamicStateEnables](#), [ven::PipelineConfigInfo::dynamicStateInfo](#), [ven::Model::Vertex::getAttributeDescriptions\(\)](#), [ven::Model::Vertex::getBindingDescriptions\(\)](#), [ven::PipelineConfigInfo::inputAssemblyInfo](#), [ven::PipelineConfigInfo::multisampleInfo](#), and [ven::PipelineConfigInfo::rasterizationInfo](#).

Referenced by [ven::ARenderSystemBase::createPipeline\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.40.3.5 operator=()

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

7.40.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
    const std::string & filename) [static], [private]
```

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

7.40.4 Member Data Documentation

7.40.4.1 m_device

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

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

Referenced by [~Shaders\(\)](#).

7.40.4.2 m_fragShaderModule

```
VkShaderModule ven::Shaders::m_fragShaderModule {nullptr} [private]
```

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

Referenced by [~Shaders\(\)](#).

7.40.4.3 m_graphicsPipeline

```
VkPipeline ven::Shaders::m_graphicsPipeline {nullptr} [private]
```

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

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

7.40.4.4 m_vertShaderModule

```
VkShaderModule ven::Shaders::m_vertShaderModule {nullptr} [private]
```

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

Referenced by [~Shaders\(\)](#).

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

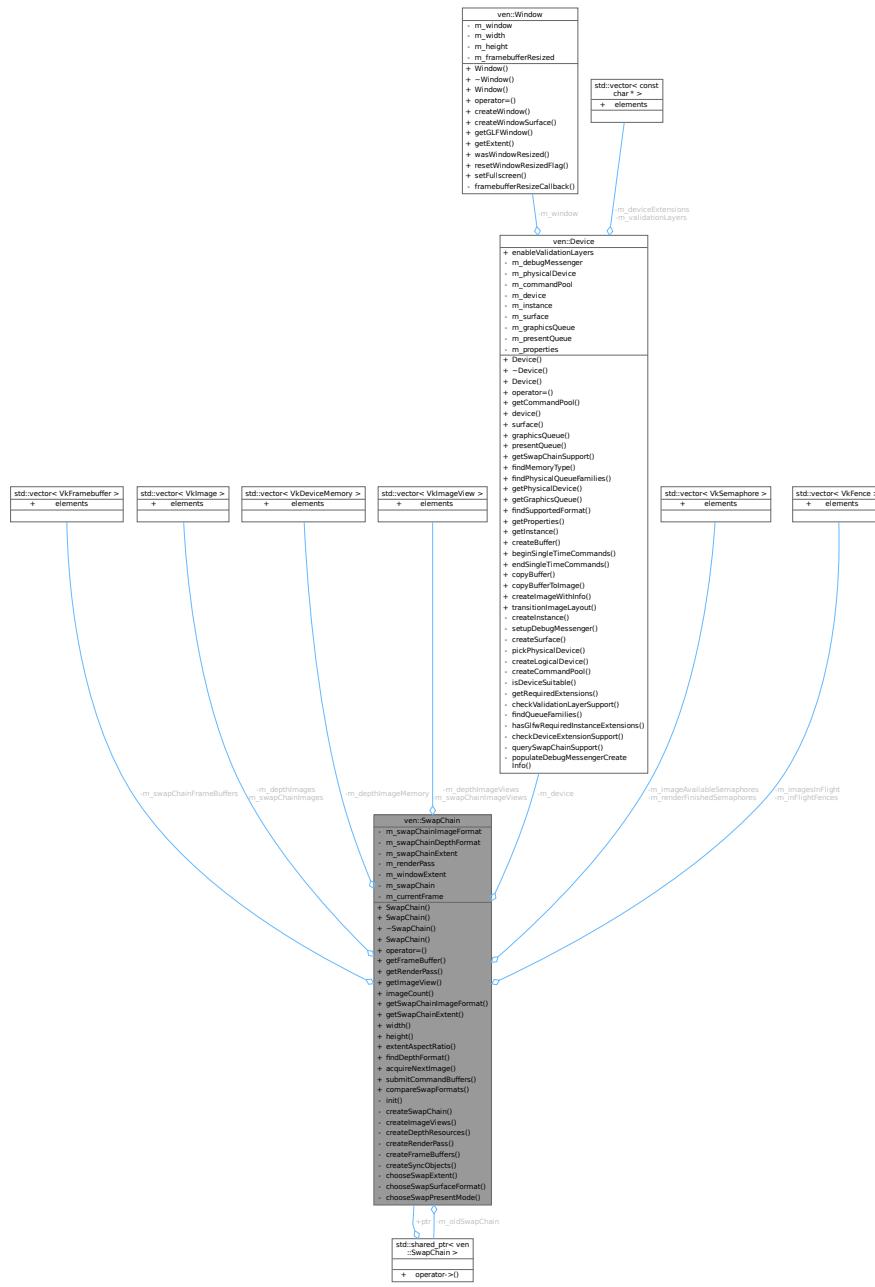
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Shaders.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[shaders.cpp](#)

7.41 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}`

7.41.1 Detailed Description

Class for swap chain.

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

7.41.2 Constructor & Destructor Documentation

7.41.2.1 SwapChain() [1/3]

```
ven::SwapChain::SwapChain (
    Device & deviceRef,
    const VkExtent2D windowExtentRef) [inline]
```

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

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

Here is the call graph for this function:



7.41.2.2 SwapChain() [2/3]

```
ven::SwapChain::SwapChain (
    Device & deviceRef,
    const VkExtent2D windowExtentRef,
    std::shared_ptr< SwapChain > previous) [inline]
```

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

References [init\(\)](#), and [m_oldSwapChain](#).

Here is the call graph for this function:



7.41.2.3 ~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:



7.41.2.4 SwapChain() [3/3]

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

7.41.3 Member Function Documentation

7.41.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
    uint32_t * imageIndex) const
```

Definition at line 49 of file [swapChain.cpp](#).

7.41.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
    const VkSurfaceCapabilitiesKHR & capabilities) const [nodiscard], [private]
```

Definition at line 362 of file [swapChain.cpp](#).

7.41.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
    const std::vector< VkPresentModeKHR > & availablePresentModes) [static], [private]
```

Definition at line 342 of file [swapChain.cpp](#).

7.41.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (
    const std::vector< VkSurfaceFormatKHR > & availableFormats) [static], [private]
```

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

7.41.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (
    const SwapChain & swapChain) const [inline], [nodiscard]
```

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

References [m_swapChainDepthFormat](#), and [m_swapChainImageFormat](#).

7.41.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources () [private]
```

Definition at line 262 of file [swapChain.cpp](#).

7.41.3.7 createFrameBuffers()

```
void ven::SwapChain::createFrameBuffers () [private]
```

Definition at line 240 of file [swapChain.cpp](#).

7.41.3.8 createImageViews()

```
void ven::SwapChain::createImageViews () [private]
```

Definition at line 160 of file [swapChain.cpp](#).

7.41.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass () [private]
```

Definition at line 181 of file [swapChain.cpp](#).

7.41.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain () [private]
```

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

7.41.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects () [private]
```

Definition at line 308 of file [swapChain.cpp](#).

References [ven::MAX_FRAMES_IN_FLIGHT](#).

7.41.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const [inline], [nodiscard]
```

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

References [m_swapChainExtent](#).

7.41.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const [nodiscard]
```

Definition at line 374 of file [swapChain.cpp](#).

7.41.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (
    const unsigned long index) const [inline], [nodiscard]
```

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

References [m_swapChainFrameBuffers](#).

7.41.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (
    const int index) const [inline], [nodiscard]
```

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

References [m_swapChainImageViews](#).

7.41.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const [inline], [nodiscard]
```

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

References [m_renderPass](#).

7.41.3.17 `getSwapChainExtent()`

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const [inline], [nodiscard]
```

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

References [m_swapChainExtent](#).

7.41.3.18 `getSwapChainImageFormat()`

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const [inline], [nodiscard]
```

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

References [m_swapChainImageFormat](#).

7.41.3.19 `height()`

```
uint32_t ven::SwapChain::height () const [inline], [nodiscard]
```

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

References [m_swapChainExtent](#).

7.41.3.20 `imageCount()`

```
size_t ven::SwapChain::imageCount () const [inline], [nodiscard]
```

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

References [m_swapChainImages](#).

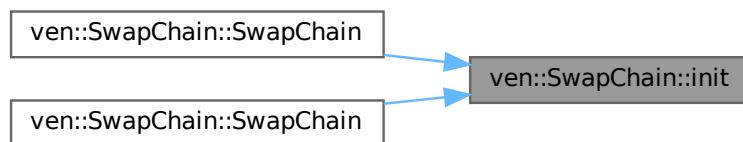
7.41.3.21 `init()`

```
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:



7.41.3.22 operator=()

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

7.41.3.23 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](#).

7.41.3.24 width()

```
uint32_t ven::SwapChain::width () const [inline], [nodiscard]
```

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

References [m_swapChainExtent](#).

7.41.4 Member Data Documentation

7.41.4.1 m_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0} [private]
```

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

7.41.4.2 m_depthImageMemory

```
std::vector<VkDeviceMemory> ven::SwapChain::m_depthImageMemory [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.3 m_depthImages

```
std::vector<VkImage> ven::SwapChain::m_depthImages [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.4 m_depthImageViews

```
std::vector<VkImageView> ven::SwapChain::m_depthImageViews [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.5 m_device

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

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

Referenced by [~SwapChain\(\)](#).

7.41.4.6 m_imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_imageAvailableSemaphores [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.7 m_imagesInFlight

```
std::vector<VkFence> ven::SwapChain::m_imagesInFlight [private]
```

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

7.41.4.8 m_inFlightFences

```
std::vector<VkFence> ven::SwapChain::m_inFlightFences [private]
```

Definition at line 85 of file [SwapChain.hpp](#).

Referenced by [~SwapChain\(\)](#).

7.41.4.9 m_oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::m_oldSwapChain [private]
```

Definition at line 81 of file [SwapChain.hpp](#).

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

7.41.4.10 m_renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::m_renderFinishedSemaphores [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.11 m_renderPass

```
VkRenderPass ven::SwapChain::m_renderPass {} [private]
```

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

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

7.41.4.12 m_swapChain

```
VkSwapchainKHR ven::SwapChain::m_swapChain {} [private]
```

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

Referenced by [~SwapChain\(\)](#).

7.41.4.13 m_swapChainDepthFormat

```
VkFormat ven::SwapChain::m_swapChainDepthFormat {} [private]
```

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

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

7.41.4.14 m_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {} [private]
```

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

Referenced by [extentAspectRatio\(\)](#), [getSwapChainExtent\(\)](#), [height\(\)](#), and [width\(\)](#).

7.41.4.15 m_swapChainFrameBuffers

```
std::vector<VkFramebuffer> ven::SwapChain::m_swapChainFrameBuffers [private]
```

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

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

7.41.4.16 m_swapChainImageFormat

```
VkFormat ven::SwapChain::m_swapChainImageFormat {} [private]
```

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

Referenced by [compareSwapFormats\(\)](#), and [getSwapChainImageFormat\(\)](#).

7.41.4.17 m_swapChainImages

```
std::vector<VkImage> ven::SwapChain::m_swapChainImages [private]
```

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

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

7.41.4.18 m_swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::m_swapChainImageViews [private]
```

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

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

7.41.4.19 m_windowExtent

```
VkExtent2D ven::SwapChain::m_windowExtent [private]
```

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

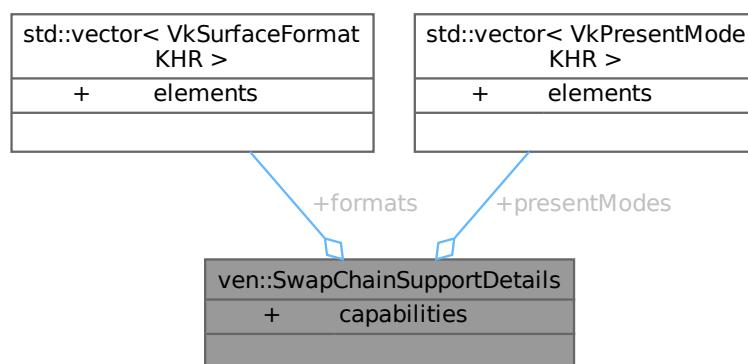
The documentation for this class was generated from the following files:

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

7.42 ven::SwapChainSupportDetails Struct Reference

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

Collaboration diagram for ven::SwapChainSupportDetails:



Public Attributes

- VkSurfaceCapabilitiesKHR [capabilities](#)
- std::vector< VkSurfaceFormatKHR > [formats](#)
- std::vector< VkPresentModeKHR > [presentModes](#)

7.42.1 Detailed Description

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

7.42.2 Member Data Documentation

7.42.2.1 capabilities

VkSurfaceCapabilitiesKHR [ven::SwapChainSupportDetails::capabilities](#)

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

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.42.2.2 formats

std::vector<VkSurfaceFormatKHR> [ven::SwapChainSupportDetails::formats](#)

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

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

7.42.2.3 presentModes

std::vector<VkPresentModeKHR> [ven::SwapChainSupportDetails::presentModes](#)

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

Referenced by [ven::Device::querySwapChainSupport\(\)](#).

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

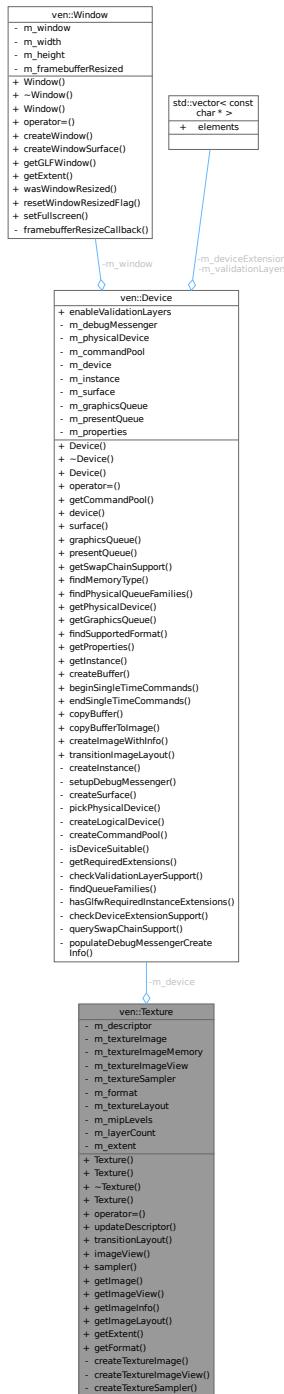
- /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp

7.43 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`

Private Member Functions

- `void createTextureImage (const std::string &filepath)`
- `void createTextureImageView (VkImageType 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 {}`

7.43.1 Detailed Description

Class for texture.

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

7.43.2 Constructor & Destructor Documentation

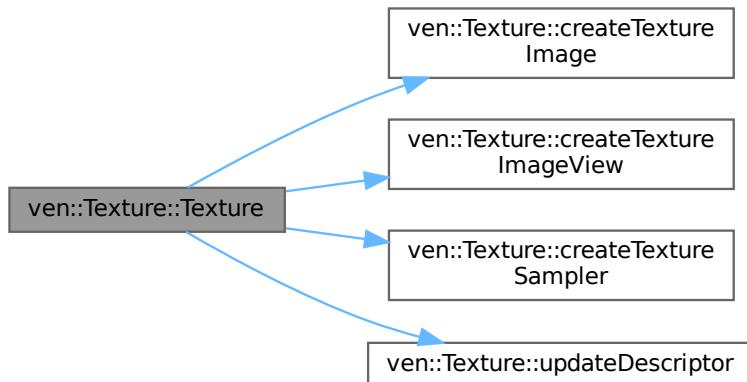
7.43.2.1 Texture() [1/3]

```
ven::Texture::Texture (
    Device & device,
    const std::string & textureFilepath)
```

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

References [createTextureImage\(\)](#), [createTextureImageView\(\)](#), [createTextureSampler\(\)](#), and [updateDescriptor\(\)](#).

Here is the call graph for this function:



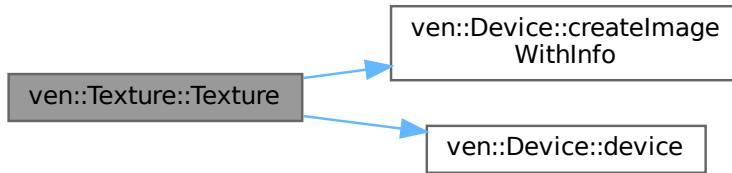
7.43.2.2 Texture() [2/3]

```
ven::Texture::Texture (
    Device & device,
    VkFormat format,
    VkExtent3D extent,
    VkImageUsageFlags usage,
    VkSampleCountFlagBits sampleCount)
```

Definition at line 14 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:



7.43.2.3 ~Texture()

```
ven::Texture::~Texture ()
```

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

7.43.2.4 Texture() [3/3]

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

7.43.3 Member Function Documentation

7.43.3.1 createTextureImage()

```
void ven::Texture::createTextureImage (
    const std::string & filepath) [private]
```

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

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

Here is the caller graph for this function:



7.43.3.2 `createTextureImageView()`

```
void ven::Texture::createTextureImageView (
    VkImageViewType viewType) [private]
```

Definition at line 190 of file [texture.cpp](#).

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

Here is the caller graph for this function:



7.43.3.3 `createTextureSampler()`

```
void ven::Texture::createTextureSampler () [private]
```

Definition at line 208 of file [texture.cpp](#).

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

Here is the caller graph for this function:



7.43.3.4 `getExtent()`

```
VkExtent3D ven::Texture::getExtent () const [inline], [nodiscard]
```

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

References [m_extent](#).

7.43.3.5 getFormat()

```
VkFormat ven::Texture::getFormat () const [inline], [nodiscard]
```

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

References [m_format](#).

7.43.3.6 getImage()

```
VkImage ven::Texture::getImage () const [inline], [nodiscard]
```

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

References [m_textureImage](#).

7.43.3.7 getImageInfo()

```
VkDescriptorImageInfo ven::Texture::getImageInfo () const [inline], [nodiscard]
```

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

References [m_descriptor](#).

7.43.3.8 getImageLayout()

```
VkImageLayout ven::Texture::getImageLayout () const [inline], [nodiscard]
```

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

References [m_textureLayout](#).

7.43.3.9 getImageView()

```
VkImageView ven::Texture::getImageView () const [inline], [nodiscard]
```

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

References [m_textureImageView](#).

7.43.3.10 imageView()

```
VkImageView ven::Texture::imageView () const [inline], [nodiscard]
```

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

References [m_textureImageView](#).

7.43.3.11 operator=()

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

7.43.3.12 sampler()

```
VkSampler ven::Texture::sampler () const [inline], [nodiscard]
```

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

References [m_textureSampler](#).

7.43.3.13 transitionLayout()

```
void ven::Texture::transitionLayout (
    VkCommandBuffer commandBuffer,
    VkImageLayout oldLayout,
    VkImageLayout newLayout) const
```

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

7.43.3.14 updateDescriptor()

```
void ven::Texture::updateDescriptor ()
```

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

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

Here is the caller graph for this function:



7.43.4 Member Data Documentation

7.43.4.1 m_descriptor

```
VkDescriptorImageInfo ven::Texture::m_descriptor {} [private]
```

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

Referenced by [getImageInfo\(\)](#), and [Texture\(\)](#).

7.43.4.2 m_device

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

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

7.43.4.3 m_extent

```
VkExtent3D ven::Texture::m_extent {} [private]
```

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

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

7.43.4.4 m_format

```
VkFormat ven::Texture::m_format [private]
```

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

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

7.43.4.5 m_layerCount

```
uint32_t ven::Texture::m_layerCount {1} [private]
```

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

7.43.4.6 m_mipLevels

```
uint32_t ven::Texture::m_mipLevels {1} [private]
```

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

7.43.4.7 m_textureImage

```
VkImage ven::Texture::m_textureImage = nullptr [private]
```

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

Referenced by [getImage\(\)](#), and [Texture\(\)](#).

7.43.4.8 m_textureImageMemory

```
VkDeviceMemory ven::Texture::m_textureImageMemory = nullptr [private]
```

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

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

7.43.4.9 m_textureImageView

```
VkImageView ven::Texture::m_textureImageView = nullptr [private]
```

Definition at line 53 of file [Texture.hpp](#).

Referenced by [getImageView\(\)](#), [imageView\(\)](#), and [Texture\(\)](#).

7.43.4.10 m_textureLayout

```
VkImageLayout ven::Texture::m_textureLayout {} [private]
```

Definition at line 56 of file [Texture.hpp](#).

Referenced by [getImageLayout\(\)](#).

7.43.4.11 m_textureSampler

```
VkSampler ven::Texture::m_textureSampler = nullptr [private]
```

Definition at line 54 of file [Texture.hpp](#).

Referenced by [sampler\(\)](#), and [Texture\(\)](#).

The documentation for this class was generated from the following files:

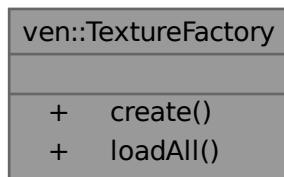
- /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/[Texture.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[texture.cpp](#)

7.44 ven::TextureFactory Class Reference

Class for [Texture](#) factory.

```
#include <Texture.hpp>
```

Collaboration diagram for ven::TextureFactory:



Static Public Member Functions

- static std::unique_ptr< [Texture](#) > [create](#) ([Device](#) &[device](#), const std::string &[filepath](#))
- static std::unordered_map< std::string, std::shared_ptr< [Texture](#) > > [loadAll](#) ([Device](#) &[device](#), const std::string &[FolderPath](#))

7.44.1 Detailed Description

Class for [Texture](#) factory.

Definition at line 21 of file [Texture.hpp](#).

7.44.2 Member Function Documentation

7.44.2.1 [create\(\)](#)

```
static std::unique_ptr< Texture > ven::TextureFactory::create (
    Device & device,
    const std::string & filepath) [inline], [static]
```

Definition at line 25 of file [Texture.hpp](#).

Referenced by [ven::SceneManager::SceneManager\(\)](#).

Here is the caller graph for this function:



7.44.2.2 [loadAll\(\)](#)

```
std::unordered_map< std::string, std::shared_ptr< ven::Texture > > ven::TextureFactory::loadAll (
    Device & device,
    const std::string & FolderPath) [static]
```

Definition at line 6 of file [Texture.cpp](#).

References [ven::Logger::logExecutionTime\(\)](#), and [ven::Logger::logWarning\(\)](#).

Here is the call graph for this function:



The documentation for this class was generated from the following files:

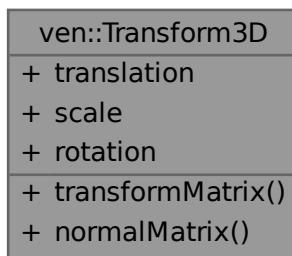
- /home/runner/work/VEngine/VEngine/include/VEngine/Factories/[Texture.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Factories/[Texture.cpp](#)

7.45 ven::Transform3D Class Reference

Class for 3D transformation.

```
#include <Transform3D.hpp>
```

Collaboration diagram for `ven::Transform3D`:



Public Member Functions

- `glm::mat4 transformMatrix () const`
- `glm::mat3 normalMatrix () const`

Public Attributes

- `glm::vec3 translation {}`
- `glm::vec3 scale {}`
- `glm::vec3 rotation {}`

7.45.1 Detailed Description

Class for 3D transformation.

Definition at line 18 of file [Transform3D.hpp](#).

7.45.2 Member Function Documentation

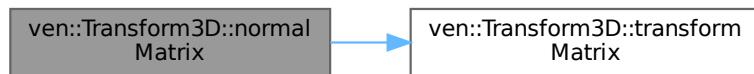
7.45.2.1 normalMatrix()

```
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:



7.45.2.2 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:



7.45.3 Member Data Documentation

7.45.3.1 rotation

```
glm::vec3 ven::Transform3D::rotation {}
```

Definition at line 38 of file [Transform3D.hpp](#).

Referenced by [ven::Gui::cameraSection\(\)](#), [ven::EventManager::moveCamera\(\)](#), and [transformMatrix\(\)](#).

7.45.3.2 scale

```
glm::vec3 ven::Transform3D::scale {}
```

Definition at line 37 of file [Transform3D.hpp](#).

Referenced by [transformMatrix\(\)](#).

7.45.3.3 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/Scene/[Transform3D.hpp](#)

7.46 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 {}`

7.46.1 Detailed Description

Definition at line 33 of file `Model.hpp`.

7.46.2 Member Function Documentation

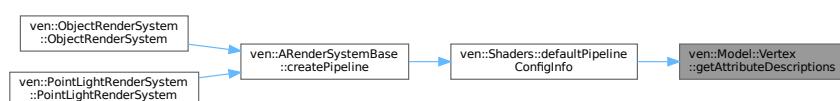
7.46.2.1 `getAttributeDescriptions()`

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions()
() [static]
```

Definition at line 93 of file `model.cpp`.

Referenced by `ven::Shaders::defaultPipelineConfigInfo()`.

Here is the caller graph for this function:



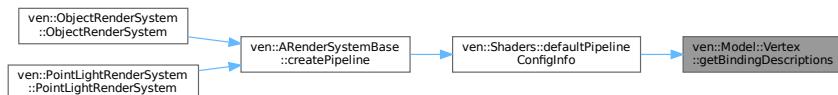
7.46.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()  
[static]
```

Definition at line 84 of file [model.cpp](#).

Referenced by [ven::Shaders::defaultPipelineConfigInfo\(\)](#).

Here is the caller graph for this function:



7.46.2.3 operator==()

```
bool ven::Model::Vertex::operator== (const Vertex & other) const [inline]
```

Definition at line 42 of file [Model.hpp](#).

References [color](#), [normal](#), [position](#), and [uv](#).

7.46.3 Member Data Documentation

7.46.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 35 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.46.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 36 of file [Model.hpp](#).

Referenced by [operator==\(\)](#).

7.46.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 34 of file [Model.hpp](#).

Referenced by [operator==\(\)](#), and [ven::Model::Builder::processMesh\(\)](#).

7.46.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 37 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/Gfx/[Model.hpp](#)
- /home/runner/work/VEngine/VEngine/src/Gfx/[model.cpp](#)

7.47 ven::Window Class Reference

Class for window.

```
#include <Window.hpp>
```

Collaboration diagram for ven::Window:

| ven::Window | |
|-------------|------------------------------|
| - | m_window |
| - | m_width |
| - | m_height |
| - | m_framebufferResized |
| + | 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

7.47.1 Detailed Description

Class for window.

Definition at line 26 of file `Window.hpp`.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 `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`.

7.47.2.2 `~Window()`

```
ven::Window::~Window () [inline]
```

Definition at line 31 of file `Window.hpp`.

References `m_window`.

7.47.2.3 Window() [2/2]

```
ven::Window::Window (
    const Window & ) [delete]
```

7.47.3 Member Function Documentation

7.47.3.1 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:



7.47.3.2 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:



7.47.3.3 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:



7.47.3.4 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::rendererSection\(\)](#).

Here is the caller graph for this function:



7.47.3.5 getGLFWwindow()

```
GLFWwindow * ven::Window::getGLFWwindow () const [inline], [nodiscard]
```

Definition at line 39 of file [Window.hpp](#).

References [m_window](#).

Referenced by [ven::Engine::Engine\(\)](#).

Here is the caller graph for this function:



7.47.3.6 operator=()

```
Window & ven::Window::operator= (
    const Window & ) [delete]
```

7.47.3.7 resetWindowResizedFlag()

```
void ven::Window::resetWindowResizedFlag () [inline]
```

Definition at line 43 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.47.3.8 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:



7.47.3.9 `wasWindowResized()`

```
bool ven::Window::wasWindowResized () const [inline], [nodiscard]
```

Definition at line 42 of file [Window.hpp](#).

References [m_framebufferResized](#).

7.47.4 Member Data Documentation

7.47.4.1 `m_framebufferResized`

```
bool ven::Window::m_framebufferResized = false [private]
```

Definition at line 55 of file [Window.hpp](#).

Referenced by [framebufferResizeCallback\(\)](#), [resetWindowResizedFlag\(\)](#), and [wasWindowResized\(\)](#).

7.47.4.2 `m_height`

```
uint32_t ven::Window::m_height {DEFAULT_HEIGHT} [private]
```

Definition at line 53 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.47.4.3 `m_width`

```
uint32_t ven::Window::m_width {DEFAULT_WIDTH} [private]
```

Definition at line 52 of file [Window.hpp](#).

Referenced by [getExtent\(\)](#).

7.47.4.4 `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/Core/Window.hpp
- /home/runner/work/VEngine/VEngine/src/Core/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     float cosDis = 0.5 * (cos(dis * M_PI) + 1.0);
00035     outColor = vec4(push.color.rgb + 0.5 * cosDis, cosDis);
00036 }
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

```

```

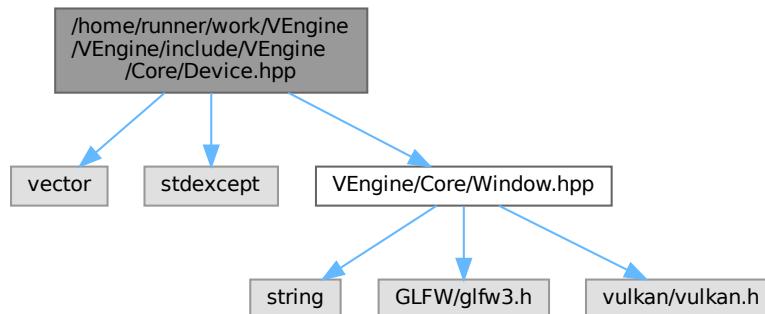
00033 layout(push_constant) uniform Push {
00034     mat4 modelMatrix;
00035     mat4 normalMatrix;
00036 } push;
00037
00038 void main() {
00039     vec4 positionWorld = object.modelMatrix * vec4(position, 1.0);
00040     gl_Position = ubo.projection * ubo.view * positionWorld;
00041     fragNormalWorld = normalize(mat3(object.normalMatrix) * normal);
00042     fragPosWorld = positionWorld.xyz;
00043     fragColor = color;
00044     fragUV = uv;
00045 }

```

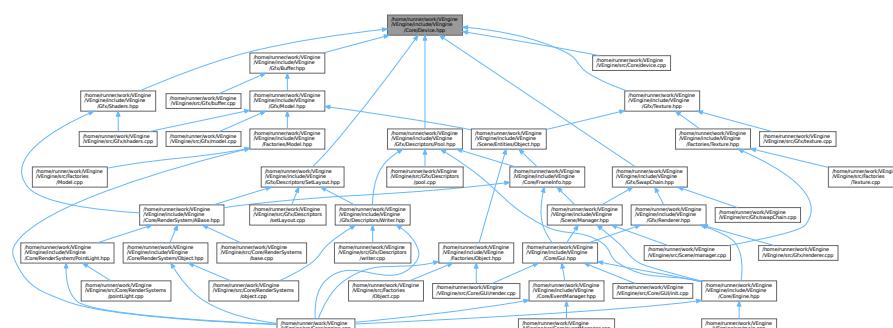
8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp File Reference

This file contains the Device class.

```
#include <vector>
#include <stdexcept>
#include "VEngine/Core/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`

8.9.1 Detailed Description

This file contains the `Device` class.

Definition in file [Device.hpp](#).

8.10 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 #include <stdexcept>
00011
00012 #include "VEngine/Core/Window.hpp"
00013
00014 namespace ven {
00015
00016     struct SwapChainSupportDetails {
00017         VkSurfaceCapabilitiesKHR capabilities;
00018         std::vector<VkSurfaceFormatKHR> formats;
00019         std::vector<VkPresentModeKHR> presentModes;
00020     };
00021
00022     struct QueueFamilyIndices {
00023         uint32_t graphicsFamily{};
00024         uint32_t presentFamily{};
00025         bool graphicsFamilyHasValue = false;
00026         bool presentFamilyHasValue = false;
00027         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
00028             presentFamilyHasValue; }
00029     };
00030
00031     /// @class Device
00032     /// @brief Class for device
00033     /// @namespace ven
00034     ///
00035     class Device {
00036
00037     public:
00038
00039         #ifdef NDEBUG
00040             const bool enableValidationLayers = false;
00041         #else
00042             const bool enableValidationLayers = true;
00043         #endif
00044
00045         explicit Device(Window &window);
00046         ~Device();
00047
00048         Device(const Device&) = delete;

```

```

00049     Device& operator=(const Device&) = delete;
00050
00051     [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
00052     [[nodiscard]] VkDevice device() const { return m_device; }
00053     [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
00054     [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
00055     [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }
00056
00057     [[nodiscard]] SwapChainSupportDetails querySwapChainSupport() const { return
00058         SwapChainSupport(m_physicalDevice); }
00059     [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags
00060     properties) const;
00061     [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
00062         findQueueFamilies(m_physicalDevice); }
00063     [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
00064     [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
00065     [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
00066     VkImageTiling tiling, VkFormatFeatureFlags features) const;
00067     [[nodiscard]] VkPhysicalDeviceProperties getProperties() const { return m_properties; }
00068     [[nodiscard]] VkInstance getInstance() const { return m_instance; }
00069
00070     // Buffer Helper Functions
00071     void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
00072     properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00073     [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00074     void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00075     void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00076     void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
00077     uint32_t layerCount) const;
00078     void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
00079     properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00080     void transitionImageLayout(VkImage image, VkFormat format, VkImageLayout oldLayout,
00081     VkImageLayout newLayout, uint32_t mipLevels = 1, uint32_t layerCount = 1) const;
00082
00083     private:
00084
00085     void createInstance();
00086     void setupDebugMessenger();
00087     void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
00088     void pickPhysicalDevice();
00089     void createLogicalDevice();
00090     void createCommandPool();
00091
00092     // helper functions
00093     bool isDeviceSuitable(VkPhysicalDevice device) const;
00094     [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00095     [[nodiscard]] bool checkValidationLayerSupport() const;
00096     QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00097     static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
00098     &createInfo);
00099     void hasGlfwRequiredInstanceExtensions() const;
00100     bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00101     SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00102
00103     Window &m_window;
00104     VkDebugUtilsMessengerEXT m_debugMessenger;
00105     VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
00106     VkCommandPool m_commandPool;
00107     VkDevice m_device;
00108     VkInstance m_instance;
00109     VkSurfaceKHR m_surface;
00110     VkQueue m_graphicsQueue;
00111     VkQueue m_presentQueue;
00112     VkPhysicalDeviceProperties m_properties;
00113
00114     const std::vector<const char *> m_validationLayers = {"VK_LAYER_KHRONOS_validation"};
00115     const std::vector<const char *> m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00116
00117 }; // class Device
00118 } // namespace ven

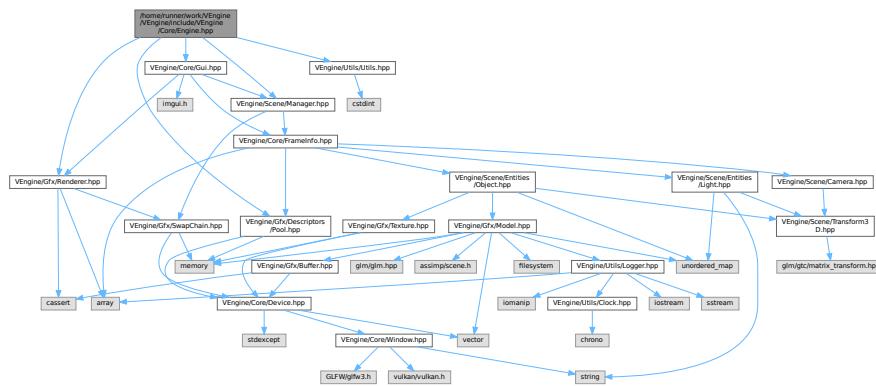
```

8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Core/[Engine.hpp](#) File Reference

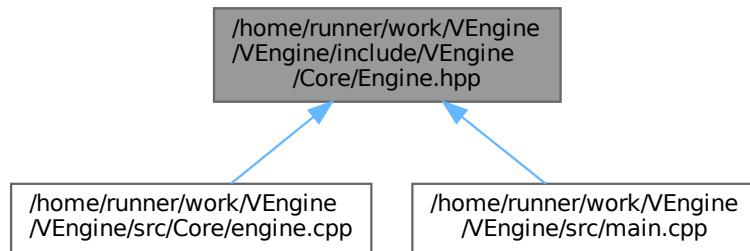
This file contains the Engine class.

```
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Gfx/Renderer.hpp"
```

```
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Scene/Manager.hpp"
#include "VEngine/Utils/Utils.hpp"
Include dependency graph for Engine.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Engine`
Class for engine.

Namespaces

- namespace `ven`

8.11.1 Detailed Description

This file contains the Engine class.

Definition in file [Engine.hpp](#).

8.12 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 "VEngine/Core/Gui.hpp"
00010 #include "VEngine/Gfx/Renderer.hpp"
00011 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00012 #include "VEngine/Scene/Manager.hpp"
00013 #include "VEngine/Utils/Utils.hpp"
00014
00015 namespace ven {
00016
00017
00018 /**
00019 /// @class Engine
00020 /// @brief Class for engine
00021 /// @namespace ven
00022 /**
00023 class Engine {
00024
00025     public:
00026
00027         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string
00028             &title = DEFAULT_TITLE.data());
00029             ~Engine() = default;
00030
00031         Engine(const Engine&) = delete;
00032         Engine operator=(const Engine&) = delete;
00033
00034         void mainLoop();
00035
00036         void cleanup();
00037
00038     private:
00039
00040         void loadObjects();
00041
00042         ENGINE_STATE m_state{EXIT};
00043
00044         Window m_window;
00045         Device m_device{m_window};
00046         Renderer m_renderer{m_window, m_device};
00047         Gui m_gui;
00048         std::unique_ptr<DescriptorPool> m_globalPool;
00049         std::vector<std::unique_ptr<DescriptorPool>> m_framePools;
00050         SceneManager m_sceneManager{m_device};
00051
00052     }; // class Engine
00053 } // namespace ven

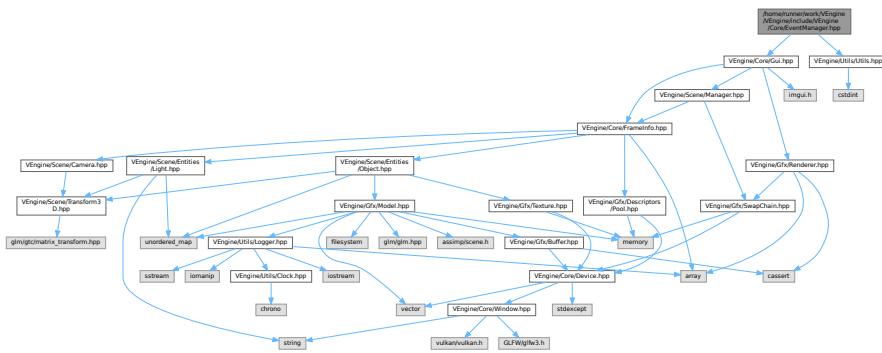
```

8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Core/EventManager.hpp File Reference

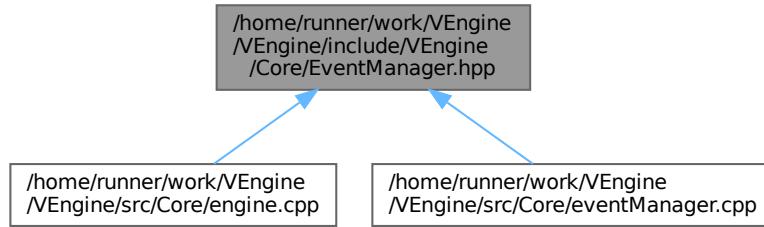
This file contains the EventManager class.

```
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Utils/Utils.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 {}

8.13.1 Detailed Description

This file contains the EventManager class.

Definition in file [EventManager.hpp](#).

8.14 EventManager.hpp

[Go to the documentation of this file.](#)

```

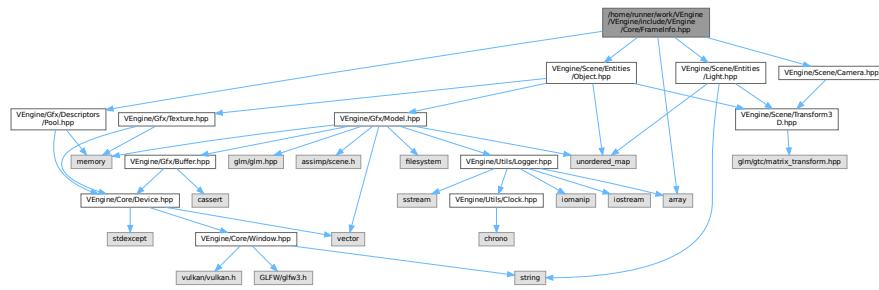
00001 /**
00002 /// @file EventManager.hpp
00003 /// @brief This file contains the EventManager class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/Gui.hpp"
00010 #include "VEngine/Utils/Utils.hpp"
00011
00012 namespace ven {
00013
00014     struct KeyAction {
00015         uint16_t key;
00016         glm::vec3 dir;
00017         glm::vec3 value;
00018     };
00019
00020     struct KeyMappings {
00021         uint16_t moveLeft = GLFW_KEY_A;
00022         uint16_t moveRight = GLFW_KEY_D;
00023         uint16_t moveForward = GLFW_KEY_W;
00024         uint16_t moveBackward = GLFW_KEY_S;
00025         uint16_t moveUp = GLFW_KEY_SPACE;
00026         uint16_t moveDown = GLFW_KEY_LEFT_SHIFT;
00027         uint16_t lookLeft = GLFW_KEY_LEFT;
00028         uint16_t lookRight = GLFW_KEY_RIGHT;
00029         uint16_t lookUp = GLFW_KEY_UP;
00030         uint16_t lookDown = GLFW_KEY_DOWN;
00031         uint16_t toggleGui = GLFW_KEY_0;
00032     };
00033
00034     static constexpr float EPSILON = std::numeric_limits<float>::epsilon();
00035     static constexpr KeyMappings DEFAULT_KEY_MAPPINGS{};
00036
00037 /**
00038 /// @class EventManager
00039 /// @brief Class for event manager
00040 /// @namespace ven
00041 /**
00042 class EventManager {
00043
00044     public:
00045
00046         EventManager() = default;
00047         ~EventManager() = default;
00048
00049         EventManager(const EventManager&) = delete;
00050         EventManager& operator=(const EventManager&) = delete;
00051
00052         void handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera, Gui& gui,
00053         float dt) const;
00054
00055     private:
00056
00057         static void moveCamera(GLFWwindow* window, Camera& camera, float dt);
00058         static void updateEngineState(ENGINE_STATE *engineState, const ENGINE_STATE newState) {
00059             *engineState = newState; }
00060         static bool isKeyJustPressed(GLFWwindow* window, long unsigned int key, std::array<bool,
00061             GLFW_KEY_LAST>& keyStates);
00062
00063         template<typename Iterator>
00064         static void processKeyActions(GLFWwindow* window, Iterator begin, Iterator end);
00065
00066         mutable std::array<bool, GLFW_KEY_LAST> m_keyState{};
00067     }; // class EventManager
00068
00069 } // namespace ven

```

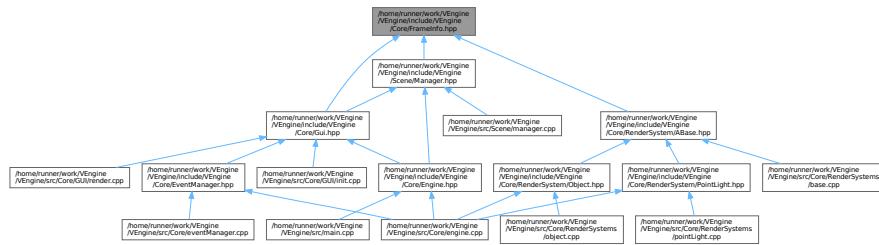
8.15 /home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp File Reference

This file contains the FrameInfo class.

```
#include <array>
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Scene/Camera.hpp"
#include "VEngine/Scene/Entities/Object.hpp"
#include "VEngine/Scene/Entities/Light.hpp"
Include dependency graph for FrameInfo.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `ven::PointLightData`
- struct `ven::ObjectBufferData`
- struct `ven::GlobalUbo`
- struct `ven::FrameInfo`

Namespaces

- namespace `ven`

Variables

- 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_INTENS`

8.15.1 Detailed Description

This file contains the `FrameInfo` class.

Definition in file [FrameInfo.hpp](#).

8.16 FrameInfo.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file FrameInfo.hpp
00003 /// @brief This file contains the FrameInfo class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <array>
00010
00011 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00012 #include "VEngine/Scene/Camera.hpp"
00013 #include "VEngine/Scene/Entities/Object.hpp"
00014 #include "VEngine/Scene/Entities/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),
00020                         DEFAULT_AMBIENT_LIGHT_INTENSITY};
00021
00022     struct PointLightData
00023     {
00024         glm::vec4 position{};
00025         glm::vec4 color{};
00026         float shininess{32.F};
00027         float padding[3]; // Pad to 32 bytes
00028     };
00029
00030     struct ObjectBufferData {
00031         glm::mat4 modelMatrix{1.F};
00032         glm::mat4 normalMatrix{1.F};
00033     };
00034
00035     struct GlobalUbo
00036     {
00037         glm::mat4 projection{1.F};
00038         glm::mat4 view{1.F};
00039         glm::mat4 inverseView{1.F};
00040         glm::vec4 ambientLightColor{DEFAULT_AMBIENT_LIGHT_COLOR};
00041         std::array<PointLightData, MAX_LIGHTS> pointLights;
00042         uint8_t numLights;
00043     };
00044
00045     struct FrameInfo
00046     {
00047         unsigned long frameIndex;
00048         float frameTime;
00049         VkCommandBuffer commandBuffer;
00050         Camera &camera;
00051         VkDescriptorSet globalDescriptorSet;
00052         DescriptorPool &frameDescriptorPool;
00053         Object::Map &objects;
00054         Light::Map &lights;
00055     };
00056 } // namespace ven

```

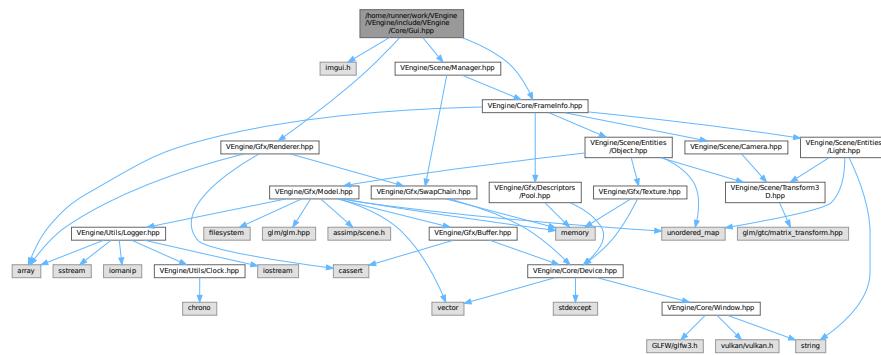
8.17 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← Gui.hpp File Reference

This file contains the ImGuiWindowManager class.

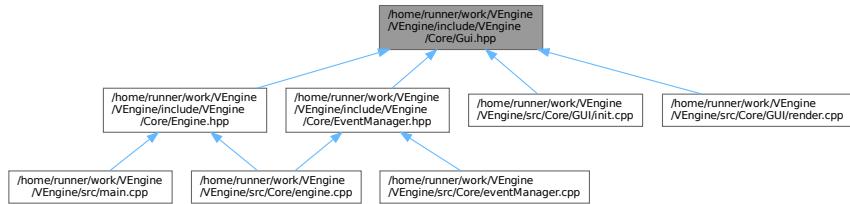
```
#include <imgui.h>
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Scene/Manager.hpp"
```

```
#include "VEngine/Gfx/Renderer.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::ClockData`
- struct `ven::Gui::funcs`

Namespaces

- namespace `ven`

Enumerations

- enum `ven::GUI_STATE` : `uint8_t` { `ven::SHOW_EDITOR` = 0 , `ven::SHOW_PLAYER` = 1 , `ven::HIDDEN` = 2 }

Variables

- static constexpr `uint16_t` `ven::DESCRIPTOR_COUNT` = 1000

8.17.1 Detailed Description

This file contains the `ImGuiWindowManager` class.

Definition in file [Gui.hpp](#).

8.18 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/Core/FrameInfo.hpp"
00012 #include "VEngine/Scene/Manager.hpp"
00013 #include "VEngine/Gfx/Renderer.hpp"
00014
00015 namespace ven {
00016
00017     static constexpr uint16_t DESCRIPTOR_COUNT = 1000;
00018
00019     enum GUI_STATE : uint8_t {
00020         SHOW_EDITOR = 0,
00021         SHOW_PLAYER = 1,
00022         HIDDEN = 2
00023     };
00024
00025 /**
00026  * @class Gui
00027  * @brief Class for Gui
00028  * @namespace ven
00029 /**
00030 class Gui {
00031
00032     struct ClockData {
00033         float deltaTimeMS{0.0F};
00034         float fps{0.0F};
00035     };
00036
00037     public:
00038
00039         Gui() = default;
00040         ~Gui() = default;
00041
00042         Gui(const Gui&) = delete;
00043         Gui& operator=(const Gui&) = delete;
00044
00045         void init(GLFWwindow* window, VkInstance instance, const Device* device, VkRenderPass
renderPass);
00046
00047         void render(Renderer *renderer, SceneManager& sceneManager, Camera& camera,
VkPhysicalDevice physicalDevice, GlobalUbo& ubo, const ClockData& clockData);
00048         static void cleanup();
00049
00050         void setState(GUI_STATE state) { m_state = state; }
00051         [[nodiscard]] GUI_STATE getState() const { return m_state; }
00052         [[nodiscard]] std::vector<unsigned int> *getObjectsToRemove() { return &m_objectsToRemove;
}
00053         [[nodiscard]] std::vector<unsigned int> *getLightsToRemove() { return &m_lightsToRemove; }
00054
00055     private:
00056
00057         static void initStyle();
00058         static void renderFrameWindow(const ClockData& clockData);
00059         static void cameraSection(Camera& camera);
00060         static void inputsSection(const ImGuiIO& io);
00061         static void rendererSection(Renderer *renderer, GlobalUbo& ubo);
00062         static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
00063         void objectsSection(SceneManager& sceneManager);
00064         void lightsSection(SceneManager& sceneManager);
00065
00066         struct funcs { static bool IsLegacyNativeDupe(const ImGuiKey key) { return key >= 0 && key
< 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist

```

```

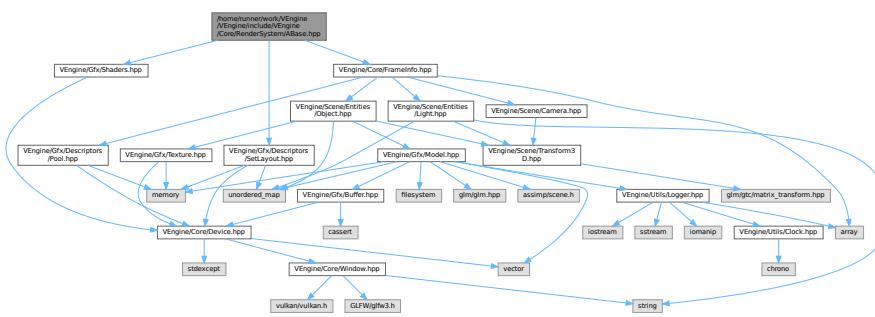
00067
00068     ImGuiIO* m_io{nullptr};
00069     GUI_STATE m_state{HIDDEN};
00070     float m_intensity{1.0f};
00071     float m_shininess{DEFAULT_SHININESS};
00072
00073     std::vector<unsigned int> m_objectsToRemove;
00074     std::vector<unsigned int> m_lightsToRemove;
00075
00076 }; // class Gui
00077
00078 } // namespace ven

```

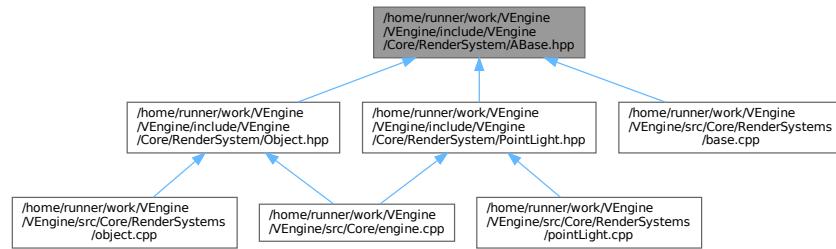
8.19 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/ABase.hpp File Reference

This file contains the ARenderSystemBase class.

```
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
#include "VEngine/Gfx/Shaders.hpp"
Include dependency graph for ABase.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::ARenderSystemBase](#)

Abstract class for render system base.

Namespaces

- namespace `ven`

8.19.1 Detailed Description

This file contains the `ARenderSystemBase` class.

Definition in file [ABase.hpp](#).

8.20 ABase.hpp

[Go to the documentation of this file.](#)

```

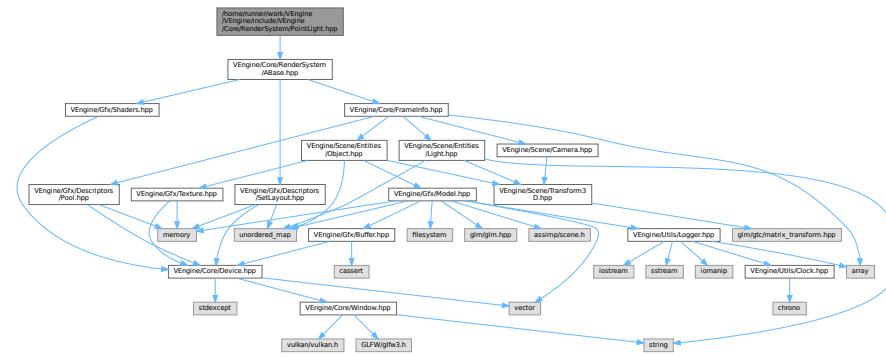
00001 /**
00002 // @file ABase.hpp
00003 // @brief This file contains the ARenderSystemBase class
00004 // @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/FrameInfo.hpp"
00010 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00011 #include "VEngine/Gfx/Shaders.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 // @class ARenderSystemBase
00017 // @brief Abstract class for render system base
00018 // @namespace ven
00019 /**
00020 class ARenderSystemBase {
00021
00022     public:
00023
00024         explicit ARenderSystemBase(Device& device) : m_device{device} {}
00025         virtual ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(),
00026             m_pipelineLayout, nullptr); }
00027         virtual void render(const FrameInfo &frameInfo) const = 0;
00028
00029     protected:
00030
00031         void createPipelineLayout(VkDescriptorSetLayout globalsetLayout, uint32_t
00032             pushConstantSize);
00032         void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
00033             std::string &shadersFragPath, bool isLight);
00034
00034         [[nodiscard]] Device& getDevice() const { return m_device; }
00035         [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
00036         [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }
00037
00038         std::unique_ptr<DescriptorsetLayout> renderSystemLayout;
00039
00040     private:
00041
00042         Device &m_device;
00043         VkPipelineLayout m_pipelineLayout{nullptr};
00044         std::unique_ptr<Shaders> m_shaders;
00045
00046
00047     }; // class ARenderSystemBase
00048
00049 } // namespace ven

```

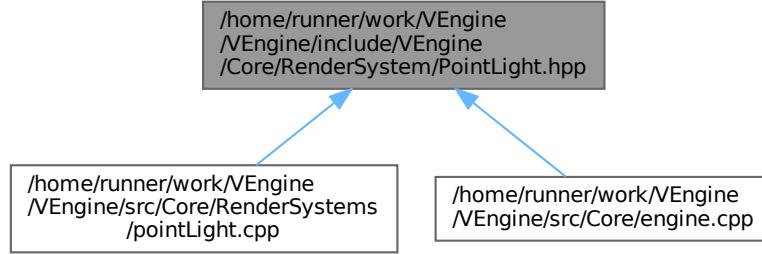
8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Core/RenderSystem/PointLight.hpp File Reference

This file contains the PointLightRenderSystem class.

```
#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for PointLight.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](#)

8.21.1 Detailed Description

This file contains the PointLightRenderSystem class.

Definition in file [PointLight.hpp](#).

8.22 PointLight.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file PointLight.hpp
00003 /// @brief This file contains the PointLightRenderSystem class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/RenderSystem/ABase.hpp"
00010
00011 namespace ven {
00012
00013     struct LightPushConstantData {
00014         glm::vec4 position{};
00015         glm::vec4 color{};
00016         float radius;
00017     };
00018
00019 /**
00020 /// @class PointLightRenderSystem
00021 /// @brief Class for point light system
00022 /// @namespace ven
00023 /**
00024 class PointLightRenderSystem final : public ARenderSystemBase {
00025
00026     public:
00027
00028     explicit PointLightRenderSystem(Device& device, const VkRenderPass renderPass, const
00029         VkDescriptorSetLayout globalsetLayout) : ARenderSystemBase(device) {
00030         createPipelineLayout(globalsetLayout, sizeof(LightPushConstantData));
00031         createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_point_light.spv",
00032             std::string(SHADERS_BIN_PATH) + "fragment_point_light.spv", true);
00033     }
00034
00035     PointLightRenderSystem(const PointLightRenderSystem&) = delete;
00036     PointLightRenderSystem& operator=(const PointLightRenderSystem&) = delete;
00037
00038     void render(const FrameInfo &frameInfo) const override;
00039 }; // class PointLightRenderSystem
00040 } // namespace ven

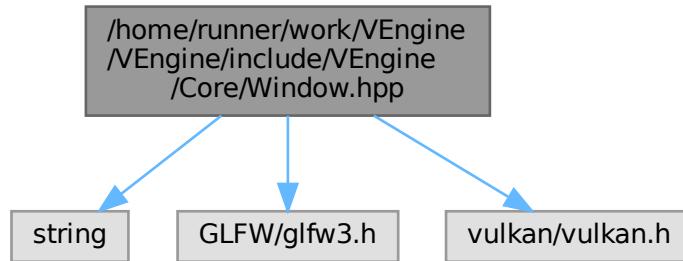
```

8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← Window.hpp File Reference

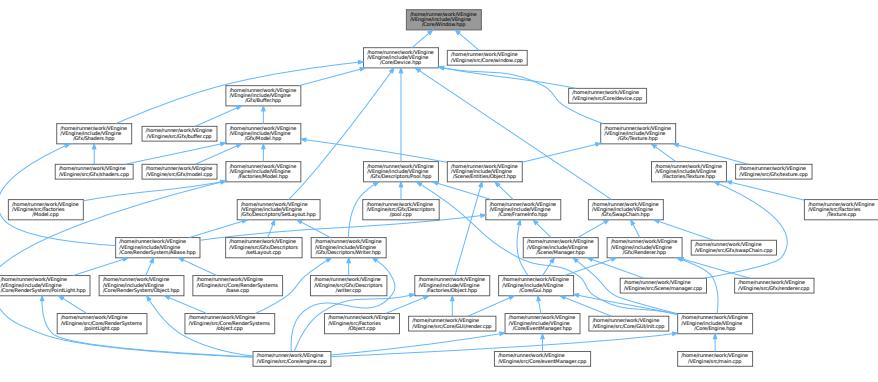
This file contains the Window class.

```
#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"

8.23.1 Detailed Description

This file contains the Window class.

Definition in file [Window.hpp](#).

8.23.2 Macro Definition Documentation

8.23.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file [Window.hpp](#).

8.24 Window.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00018     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00019     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00020
00021 /**
00022 /// @class Window
00023 /// @brief Class for window
00024 /// @namespace ven
00025 /**
00026 class Window {
00027
00028     public:
00029
00030         explicit Window(const uint32_t width = DEFAULT_WIDTH, const uint32_t height =
00031             DEFAULT_HEIGHT, const std::string &title = DEFAULT_TITLE.data()) : m_window(createWindow(
00032             width, height, title)), m_width(width), m_height(height) {}
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]] VkExtent2D getExtent() const { return {m_width, m_height}; }
00043         [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00044         void resetWindowResizedFlag() { m_framebufferResized = false; }
00045
00046         void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);
00047
00048     private:
00049
00050         static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00051         GLFWwindow* m_window{nullptr};
```

```

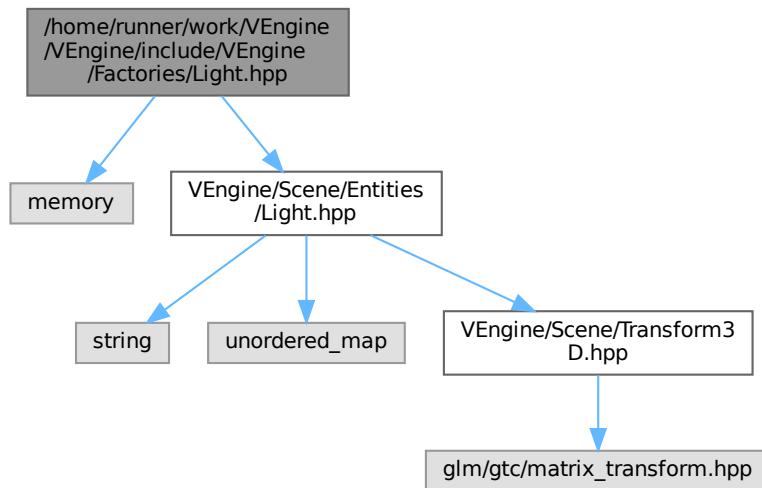
00052     uint32_t m_width{DEFAULT_WIDTH};
00053     uint32_t m_height{DEFAULT_HEIGHT};
00054     bool m_framebufferResized = false;
00055 };
00056 // class Window
00057
00058 } // namespace ven

```

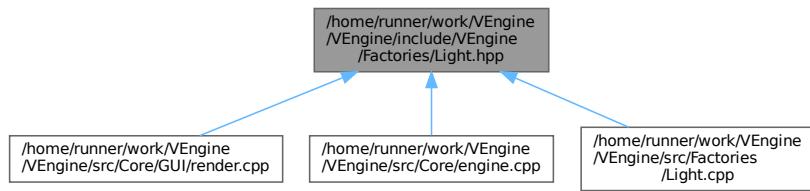
8.25 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/ Light.hpp File Reference

This file contains the Light Factory class.

```
#include <memory>
#include "VEngine/Scene/Entities/Light.hpp"
Include dependency graph for Light.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::LightFactory](#)
Class for light factory.

Namespaces

- namespace `ven`

Variables

- static constexpr `Transform3D ven::DEFAULT_TRANSFORM = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F, 0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}}`

8.25.1 Detailed Description

This file contains the Light Factory class.

Definition in file [Light.hpp](#).

8.26 Light.hpp

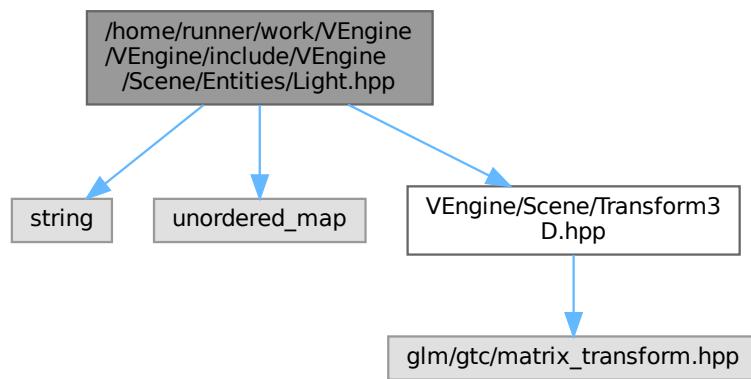
[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Light.hpp
00003 /// @brief This file contains the Light Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Scene/Entities/Light.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr Transform3D DEFAULT_TRANSFORM = {.translation = {0.F, 0.F, 0.F}, .scale = {0.1F,
0.F, 0.F}, .rotation = {0.F, 0.F, 0.F}};
00016
00017 /**
00018 /// @class LightFactory
00019 /// @brief Class for light factory
00020 /// @namespace ven
00021 /**
00022 class LightFactory {
00023
00024     public:
00025
00026         static std::unique_ptr<Light> create(const Transform3D &transform = DEFAULT_TRANSFORM,
00027             glm::vec4 color = DEFAULT_LIGHT_COLOR);
00028         static std::unique_ptr<Light> duplicate(const Light &cpyLight) { return
00029             create(cpyLight.transform, cpyLight.color); }
00030
00031     private:
00032
00033         static unsigned int m_currentLightId;
00034
00035 } // namespace ven
```

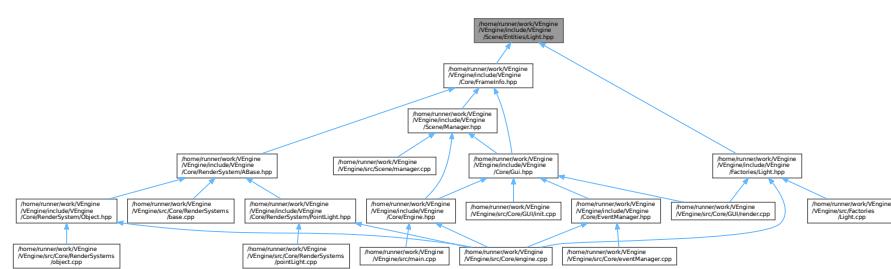
8.27 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp File Reference

This file contains the Light class.

```
#include <string>
#include <unordered_map>
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Light.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Light](#)
Class for light.

Namespaces

- namespace [ven](#)

Variables

- static constexpr float `ven::DEFAULT_LIGHT_INTENSITY` = .2F
- static constexpr float `ven::DEFAULT_LIGHT_RADIUS` = 0.1F
- static constexpr 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

8.27.1 Detailed Description

This file contains the Light class.

Definition in file [Light.hpp](#).

8.28 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 <string>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Scene/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 } // namespace ven

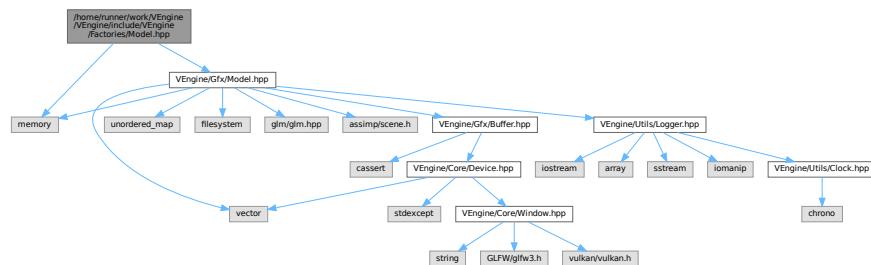
```

8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Model.hpp File Reference

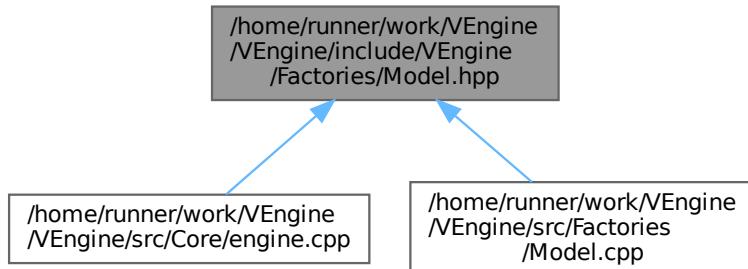
This file contains the Model Factory class.

```
#include <memory>
#include "VEngine/Gfx/Model.hpp"
```

Include dependency graph for Model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::ModelFactory](#)

Class for Model factory.

Namespaces

- namespace [ven](#)

8.29.1 Detailed Description

This file contains the Model Factory class.

Definition in file [Model.hpp](#).

8.30 Model.hpp

[Go to the documentation of this file.](#)

```

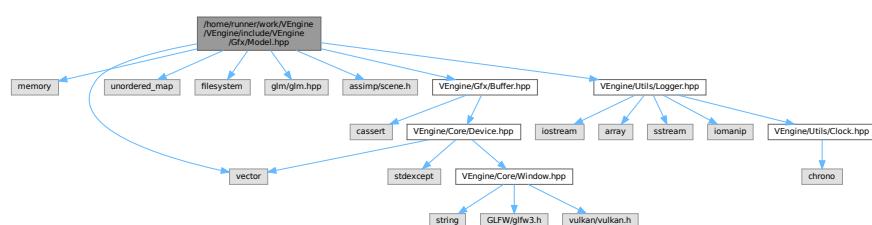
00001 /**
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Gfx/Model.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 /// @class ModelFactory
00017 /// @brief Class for Model factory
00018 /// @namespace ven
00019 /**
00020 class ModelFactory {
00021
00022     public:
00023
00024         static std::unique_ptr<Model> create(Device& device, const std::string& filepath);
00025         static std::unordered_map<std::string, std::shared_ptr<Model>> loadAll(Device& device,
00026         const std::string& folderPath);
00027     }; // class ModelFactory
00028
00029 } // namespace ven

```

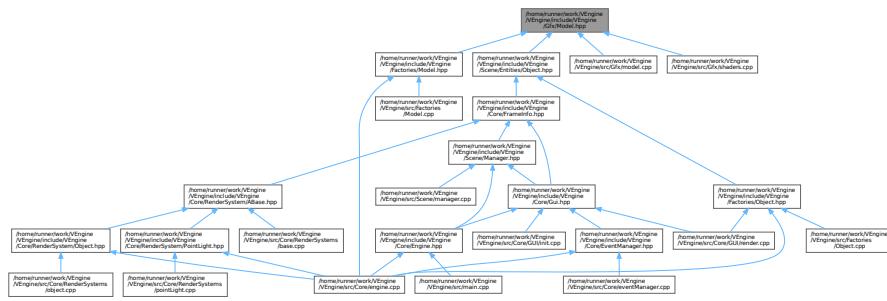
8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Model.hpp File Reference

This file contains the Model class.

```
#include <memory>
#include <vector>
#include <unordered_map>
#include <filesystem>
#include <glm/glm.hpp>
#include <assimp/scene.h>
#include "VEngine/Gfx/Buffer.hpp"
#include "VEngine/Utils/Logger.hpp"
Include dependency graph for Model.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Model](#)
Class for model.
- struct [ven::Model::Vertex](#)
- struct [ven::Model::Builder](#)

Namespaces

- namespace [ven](#)

8.31.1 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 #include <vector>
00011 #include <unordered_map>
00012 #include <filesystem>
00013
00014 #include <glm/glm.hpp>
00015
00016 #include <assimp/scene.h>
00017
00018 #include "VEngine/Gfx/Buffer.hpp"
00019 #include "VEngine/Utils/Logger.hpp"
00020
00021
00022 namespace ven {
00023
00024     /**
00025     /// @class Model

```

```

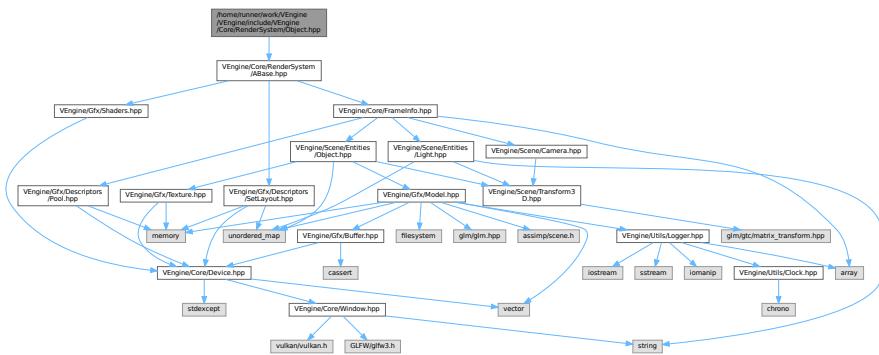
00026  /// @brief Class for model
00027  /// @namespace ven
00028  ///
00029  class Model {
00030      public:
00032      struct Vertex {
00034          glm::vec3 position{};
00035          glm::vec3 color{};
00036          glm::vec3 normal{};
00037          glm::vec2 uv{};
00038
00039          static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00040          static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00041
00042          bool operator==(const Vertex& other) const { return position == other.position &&
00043              color == other.color && normal == other.normal && uv == other.uv; }
00043      };
00044
00045      struct Builder {
00046          std::vector<Vertex> vertices;
00047          std::vector<uint32_t> indices;
00048
00049          void loadModel(const std::string &filename);
00050          void processNode(const aiNode* node, const aiScene* scene);
00051          void processMesh(const aiMesh* mesh, const aiScene* scene);
00052      };
00053
00054      Model(Device &device, const Builder &builder);
00055      ~Model() = default;
00056
00057      Model(const Model&) = delete;
00058      void operator=(const Model&) = delete;
00059
00060
00061      void bind(VkCommandBuffer commandBuffer) const;
00062      void draw(VkCommandBuffer commandBuffer) const;
00063
00064  private:
00065
00066      void createVertexBuffer(const std::vector<Vertex>& vertices);
00067      void createIndexBuffer(const std::vector<uint32_t>& indices);
00068
00069      Device& m_device;
00070      std::unique_ptr<Buffer> m_vertexBuffer;
00071      uint32_t m_vertexCount;
00072
00073      bool m_hasIndexBuffer{false};
00074      std::unique_ptr<Buffer> m_indexBuffer;
00075      uint32_t m_indexCount;
00076
00077  }; // class Model
00078
00079 } // namespace ven

```

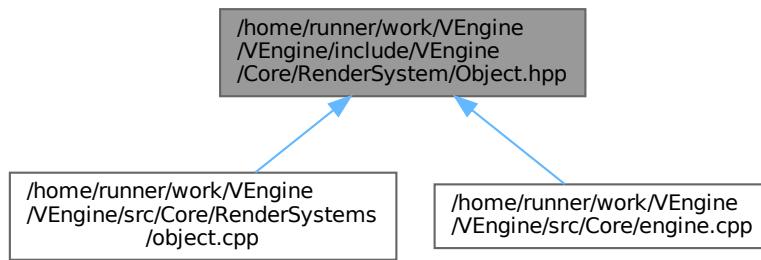
8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Core/← RenderSystem/Object.hpp File Reference

This file contains the ObjectRenderSystem class.

```
#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for Object.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `ven::ObjectPushConstantData`
 - class `ven::ObjectRenderSystem`
- Class for object render system.*

Namespaces

- namespace `ven`

8.33.1 Detailed Description

This file contains the `ObjectRenderSystem` class.

Definition in file [Object.hpp](#).

8.34 Object.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Object.hpp
00003 /// @brief This file contains the ObjectRenderSystem class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/RenderSystem/ABase.hpp"
00010
00011 namespace ven {
00012
00013     struct ObjectPushConstantData {
00014         glm::mat4 modelMatrix{};
00015         glm::mat4 normalMatrix{};
00016     };
00017
00018 /**
00019     /// @class ObjectRenderSystem
00020     /// @brief Class for object render system
00021     /// @namespace ven
00022 /**
00023     class ObjectRenderSystem final : public ARenderSystemBase {
00024
00025     public:
00026
00027         explicit ObjectRenderSystem(Device& device, const VkRenderPass renderPass, const
00028             VkDescriptorSetLayout globalsetLayout) : ARenderSystemBase(device) {
00029             createPipelineLayout(globalsetLayout, sizeof(ObjectPushConstantData));
00030             createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_shader.spv",
00031                         std::string(SHADERS_BIN_PATH) + "fragment_shader.spv", false);
00032         }
00033
00034         ObjectRenderSystem(const ObjectRenderSystem&) = delete;
00035         ObjectRenderSystem& operator=(const ObjectRenderSystem&) = delete;
00036
00037         void render(const FrameInfo &frameInfo) const override;
00038     }; // class ObjectRenderSystem
00039 } // namespace ven

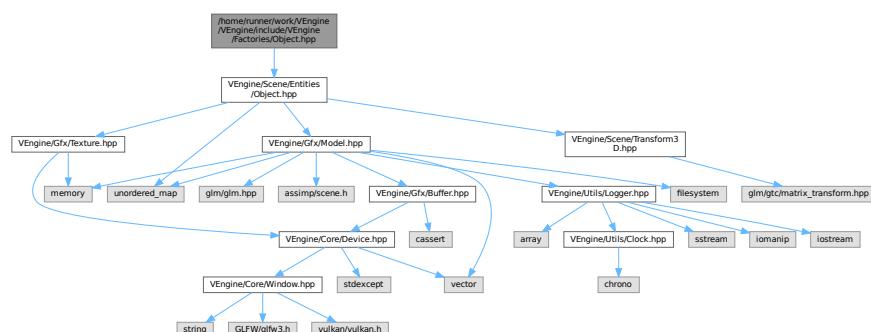
```

8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/← Object.hpp File Reference

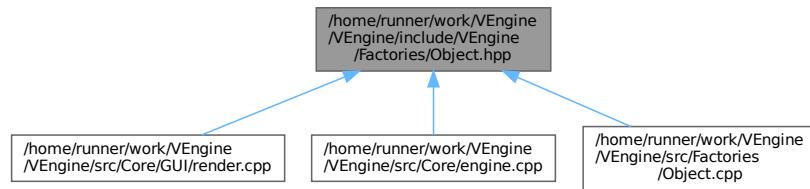
This file contains the Object Factory class.

```
#include "VEngine/Scene/Entities/Object.hpp"
```

Include dependency graph for Object.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::ObjectFactory](#)
Class for object factory.

Namespaces

- namespace [ven](#)

8.35.1 Detailed Description

This file contains the Object Factory class.

Definition in file [Object.hpp](#).

8.36 Object.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Object.hpp
00003  * @brief This file contains the Object Factory class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Scene/Entities/Object.hpp"
00010
00011 namespace ven {
00012
00013 /**
00014  * @class ObjectFactory
00015  * @brief Class for object factory
00016  * @namespace ven
00017 /**
00018 class ObjectFactory {
00019
00020     public:
00021
00022         static std::unique_ptr<Object> create(const std::shared_ptr<Texture>& texture, const
00023             std::shared_ptr<Model>& model, const std::string &name, const Transform3D &transform);
00024         static std::unique_ptr<Object> duplicate(const Object& objSrc) { return
00025             create(objSrc.getDiffuseMap(), objSrc.getModel(), objSrc.getName(), objSrc.transform); }
00026
00027     private:
00028
00029         static unsigned int m_currentObjId;
00030
00031 } // class ObjectFactory
00032 // namespace ven
  
```

8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp File Reference

This file contains the Object class.

```
#include <unordered_map>
#include "VEngine/Gfx/Texture.hpp"
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Object.hpp:
```

```

graph TD
    Root["VEngine/include/Engine/Scene/Entities/Object1.hpp"] --> Texture["VEngine/Gfx/Texture.hpp"]
    Root --> Model["VEngine/Gfx/Model.hpp"]
    Root --> Transform["VEngine/Scene/Transform3D.hpp"]

    Texture --> Memory["memory"]
    Texture --> UnorderedMap["unordered_map"]

    Model --> GlmGlm["glm/glm.hpp"]
    Model --> AssimpScene["assimp/scene.h"]
    Model --> Buffer["VEngine/Gfx/Buffer.hpp"]
    Model --> Logger["VEngine/Utils/Logger.hpp"]

    Transform --> Filesystem["filesystem"]
    Transform --> GltcMatrix["glm/gtc/matrix_transform.hpp"]

    Logger --> Array["array"]
    Logger --> Clock["VEngine/Utils/Clock.hpp"]
    Logger --> Sstream["sstream"]
    Logger --> Iomanip["iomanip"]
    Logger --> Iosstream["iostream"]

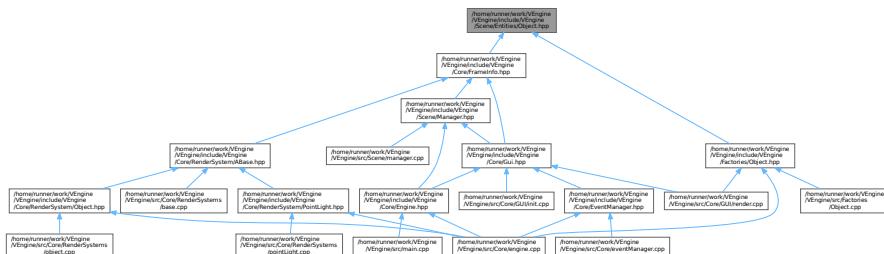
    Device["VEngine/Core/Device.hpp"] --> Casser["casser"]

    Window["VEngine/Core/Window.hpp"] --> StdExcept["std::exception"]
    Window --> Vector["vector"]

    Window --> String["string"]
    Window --> GiflW["GLFW/glfw3.h"]
    Window --> Vulkan["vulkan/vulkan.h"]

```

This graph shows which files directly or indirectly include this file:



Classes

- class `ven::Object`

Namespaces

- namespace `ven`

Variables

- static constexpr uint16_t **ven::MAX_OBJECTS** = 1000

8.37.1 Detailed Description

This file contains the Object class.

Definition in file [Object.hpp](#).

8.38 Object.hpp

[Go to the documentation of this file.](#)

```

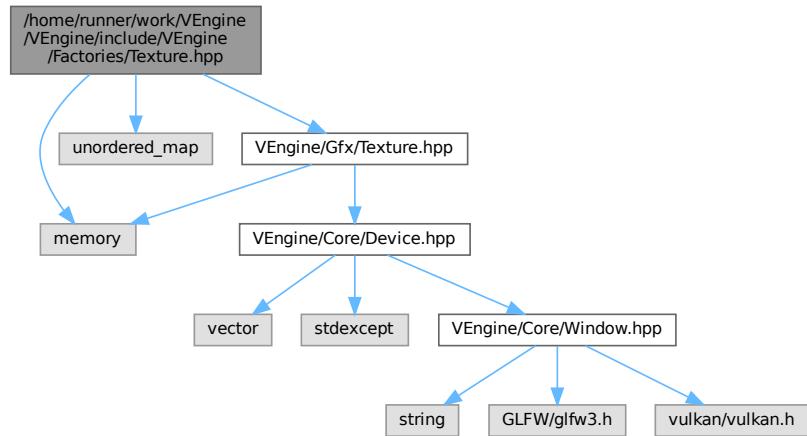
00001 /**
00002 /// @file Object.hpp
00003 /// @brief This file contains the Object class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <unordered_map>
00010
00011 #include "VEngine/Gfx/Texture.hpp"
00012 #include "VEngine/Gfx/Model.hpp"
00013 #include "VEngine/Scene/Transform3D.hpp"
00014
00015 namespace ven {
00016
00017     static constexpr uint16_t MAX_OBJECTS = 1000;
00018
00019 /**
00020 /// @class Object
00021 /// @brief Class for object
00022 /// @namespace ven
00023 /**
00024 class Object {
00025
00026     public:
00027
00028         using Map = std::unordered_map<unsigned int, Object>;
00029
00030         explicit Object(const unsigned int objId) : m_objId{objId} {}
00031
00032         ~Object() = default;
00033
00034         Object(const Object &) = delete;
00035         Object &operator=(const Object &) = delete;
00036         Object(Object &&) = default;
00037         Object &operator=(Object &&) = default;
00038
00039         [[nodiscard]] unsigned int getId() const { return m_objId; }
00040         [[nodiscard]] std::string getName() const { return m_name; }
00041         [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }
00042         [[nodiscard]] std::shared_ptr<Texture> getDiffuseMap() const { return m_diffuseMap; }
00043         [[nodiscard]] VkDescriptorBufferInfo getBufferInfo(const int frameIndex) const { return
00044             m_bufferInfo.at(frameIndex); }
00045         void setModel(const std::shared_ptr<Model> &model) { m_model = model; }
00046         void setDiffuseMap(const std::shared_ptr<Texture> &diffuseMap) { m_diffuseMap =
00047             diffuseMap; }
00048         void setName(const std::string &name) { m_name = name; }
00049         void setBufferInfo(const int frameIndex, const VkDescriptorBufferInfo& info) {
00050             m_bufferInfo[frameIndex] = info;
00051
00052         Transform3D transform{};
00053
00054     private:
00055         unsigned int m_objId;
00056         std::string m_name;
00057         std::shared_ptr<Model> m_model = nullptr;
00058         std::shared_ptr<Texture> m_diffuseMap = nullptr;
00059         std::unordered_map<int, VkDescriptorBufferInfo> m_bufferInfo;
00060
00061     }; // class Object
00062
00063 } // namespace ven

```

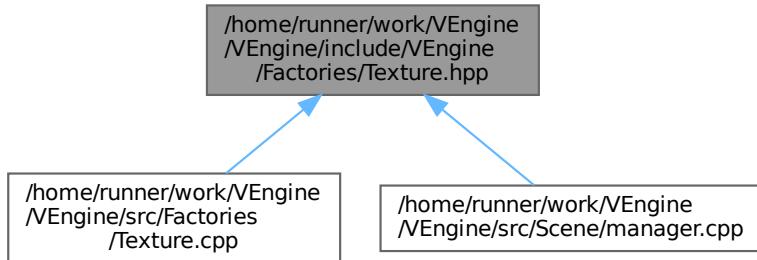
8.39 /home/runner/work/VEngine/VEngine/include/VEngine/Factories/Texture.hpp File Reference

This file contains the Texture Factory class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Gfx/Texture.hpp"
Include dependency graph for Texture.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class `ven::TextureFactory`
Class for Texture factory.

Namespaces

- namespace `ven`

8.39.1 Detailed Description

This file contains the Texture Factory class.

Definition in file [Texture.hpp](#).

8.40 Texture.hpp

[Go to the documentation of this file.](#)

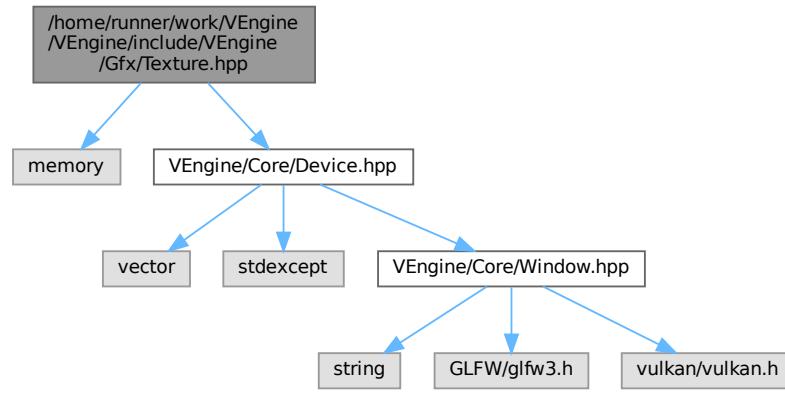
```
00001 /**
00002 /// @file Texture.hpp
00003 /// @brief This file contains the Texture Factory class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Gfx/Texture.hpp"
00013
00014 namespace ven {
00015
00016 /**
00017 /// @class TextureFactory
00018 /// @brief Class for Texture factory
00019 /// @namespace ven
00020 /**
00021 class TextureFactory {
00022
00023     public:
00024
00025         static std::unique_ptr<Texture> create(Device& device, const std::string& filepath) {
00026             return std::make_unique<Texture>(device, filepath);
00027             static std::unordered_map<std::string, std::shared_ptr<Texture>> loadAll(Device& device,
00028             const std::string& folderPath);
00029         }; // class TextureFactory
00030 } // namespace ven
```

8.41 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/← Texture.hpp File Reference

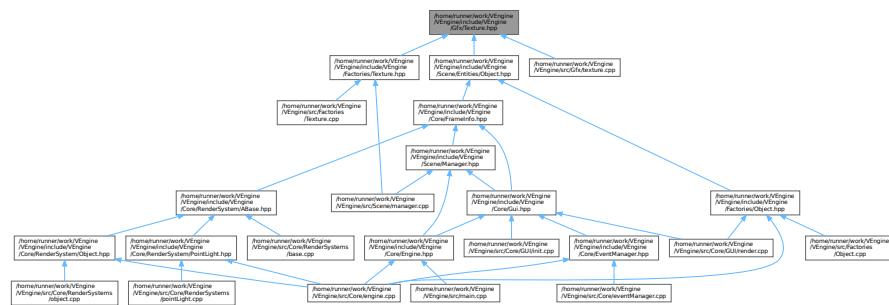
This file contains the Texture class.

```
#include <memory>
#include "VEngine/Core/Device.hpp"
```

Include dependency graph for Texture.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Texture](#)

Class for texture.

Namespaces

- namespace [ven](#)

8.41.1 Detailed Description

This file contains the Texture class.

Definition in file [Texture.hpp](#).

8.42 Texture.hpp

[Go to the documentation of this file.](#)

```

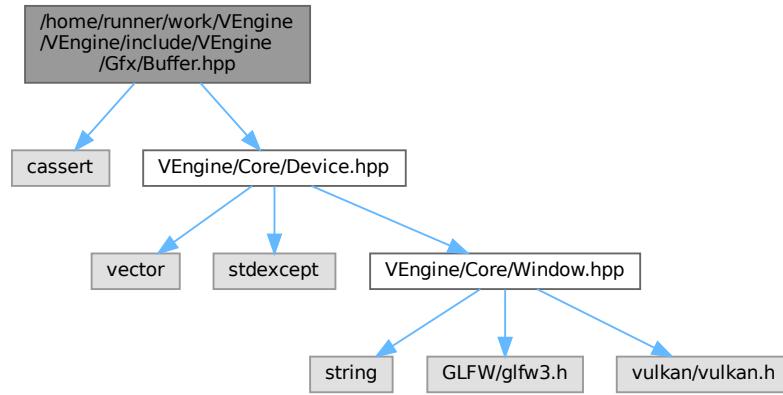
00001 /**
00002 /// @file Texture.hpp
00003 /// @brief This file contains the Texture class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 /// @class Texture
00017 /// @brief Class for texture
00018 /// @namespace ven
00019 /**
00020 class Texture {
00021
00022     public:
00023
00024         Texture(Device &device, const std::string &textureFilepath);
00025         Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00026         VkSampleCountFlagBits sampleCount);
00027         ~Texture();
00028
00029         Texture(const Texture &) = delete;
00030         Texture &operator=(const Texture &) = delete;
00031
00032         void updateDescriptor();
00033         void transitionLayout(VkCommandBuffer commandBuffer, VkImageLayout oldLayout,
00034         VkImageLayout newLayout) const;
00035
00036         [[nodiscard]] VkImageView imageView() const { return m_textureImageView; }
00037         [[nodiscard]] VkSampler sampler() const { return m_textureSampler; }
00038         [[nodiscard]] VkImage getImage() const { return m_textureImage; }
00039         [[nodiscard]] VkImageView getImageView() const { return m_textureImageView; }
00040         [[nodiscard]] VkDescriptorImageInfo getImageInfo() const { return m_descriptor; }
00041         [[nodiscard]] VkImageLayout getLayout() const { return m_textureLayout; }
00042         [[nodiscard]] VkExtent3D getExtent() const { return m_extent; }
00043         [[nodiscard]] VkFormat getFormat() const { return m_format; }
00044
00045     private:
00046
00047         void createTextureImage(const std::string &filepath);
00048         void createTextureImageView(VkImageViewType viewType);
00049         void createTextureSampler();
00050
00051         VkDescriptorImageInfo m_descriptor{};
00052         Device &m_device;
00053         VkImage m_textureImage = nullptr;
00054         VkDeviceMemory m_textureImageMemory = nullptr;
00055         VkImageView m_textureImageView = nullptr;
00056         VkSampler m_textureSampler = nullptr;
00057         VkFormat m_format;
00058         VkImageLayout m_textureLayout{};
00059         uint32_t m_mipLevels{1};
00060         uint32_t m_layerCount{1};
00061         VkExtent3D m_extent{};
00062
00063 } // class Texture
00064
00065 } // namespace ven

```

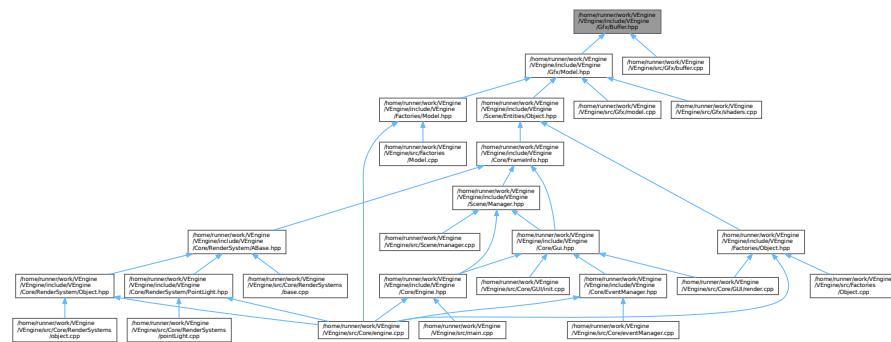
8.43 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Buffer.hpp File Reference

This file contains the Buffer class.

```
#include <cassert>
#include "VEngine/Core/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](#)

8.43.1 Detailed Description

This file contains the Buffer class.

Definition in file [Buffer.hpp](#).

8.44 Buffer.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Buffer.hpp
00003 /// @brief This file contains the Buffer class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <cassert>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015 /**
00016 /// @class Buffer
00017 /// @brief Class for buffer
00018 /// @namespace ven
00019 /**
00020 class Buffer {
00021
00022     public:
00023
00024         Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
00025             VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
00026             minOffsetAlignment = 1);
00027         ~Buffer();
00028
00029         Buffer(const Buffer&) = delete;
00030         Buffer& operator=(const Buffer&) = delete;
00031
00032         /**
00033             /// @brief Map a memory range of this buffer. If successful, mapped points to the
00034             specified buffer range.
00035             /**
00036                 /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the
00037                 complete buffer range.
00038                 /**
00039                     /// @param offset (Optional) Byte offset from beginning
00040                     /**
00041                     /// @return VkResult of the buffer mapping call
00042                     /**
00043                     VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00044
00045         void unmap();
00046
00047         /**
00048             /// @brief Copies the specified data to the mapped buffer. Default value writes whole
00049             buffer range
00050             /**
00051                 /// @param data Pointer to the data to copy
00052                 /// @param size (Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the
00053                 complete buffer range.
00054                 /**
00055                     /// @param offset (Optional) Byte offset from beginning of mapped region
00056                     /**
00057                     void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
00058                         offset = 0) const;
00059
00060         /**
00061             /// @brief Flush a memory range of the buffer to make it visible to the device
00062             /**
00063                 /// @param size (Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush
00064                 the complete buffer range.
00065                 /**
00066                     /// @param offset (Optional) Byte offset from beginning
00067                     /**
00068                     /// @return VkResult of the flush call
00069                     /**
00070                     VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0) const;
00071
00072         /**
00073             /// @brief Create a buffer info descriptor
00074             /**
00075                 /// @param size (Optional) Size of the memory range of the descriptor
00076                 /// @param offset (Optional) Byte offset from beginning
00077                 /**
00078                     /// @return VkDescriptorBufferInfo of specified offset and range
```

```

00075      /**
00076       * [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size =
00077           VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset,
00078           size, }; }
00079
00080      /**
00081      //brief Invalidate a memory range of the buffer to make it visible to the host
00082      /**
00083      //note Only required for non-coherent memory
00084      /**
00085      //param size (Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to
00086      //invalidate
00087      //the complete buffer range.
00088      //param offset (Optional) Byte offset from beginning
00089      /**
00090      //return VkResult of the invalidate call
00091      /**
00092      [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset =
00093          0) const;
00094
00095      /**
00096      // Copies "instanceSize" bytes of data to the mapped buffer at an offset of index *
00097      //alignmentSize
00098      /**
00099      //param data Pointer to the data to copy
00100      //param index Used in offset calculation
00101      /**
00102      void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
00103          m_instanceSize, index * m_alignmentSize); }
00104
00105      /**
00106      // Flush the memory range at index * alignmentSize of the buffer to make it visible to
00107      //the device
00108      /**
00109      // Create a buffer info descriptor
00110      /**
00111      //param index Specifies the region given by index * alignmentSize
00112      /**
00113      //return VkDescriptorBufferInfo for instance at index
00114      /**
00115      [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index)
00116          const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }
00117
00118      /**
00119      // Invalidate a memory range of the buffer to make it visible to the host
00120      /**
00121      //note Only required for non-coherent memory
00122      /**
00123      //param index Specifies the region to invalidate: index * alignmentSize
00124      /**
00125      //return VkResult of the invalidate call
00126      /**
00127      [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return
00128          invalidate(m_alignmentSize, index * m_alignmentSize); }
00129
00130      [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
00131      [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
00132      [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
00133      [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }
00134      [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_alignmentSize; }
00135      [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00136      [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
00137          m_memoryPropertyFlags; }
00138
00139      private:
00140
00141      /**
00142      // Returns the minimum instance size required to be compatible with devices
00143      minOffsetAlignment
00144      /**
00145      //param instanceSize The size of an instance
00146      //param minOffsetAlignment The minimum required alignment, in bytes, for the offset
00147      //member (eg
00148      //minUniformBufferOffsetAlignment)
00149      /**
00150      //return VkResult of the buffer mapping call
00151      /**

```

```

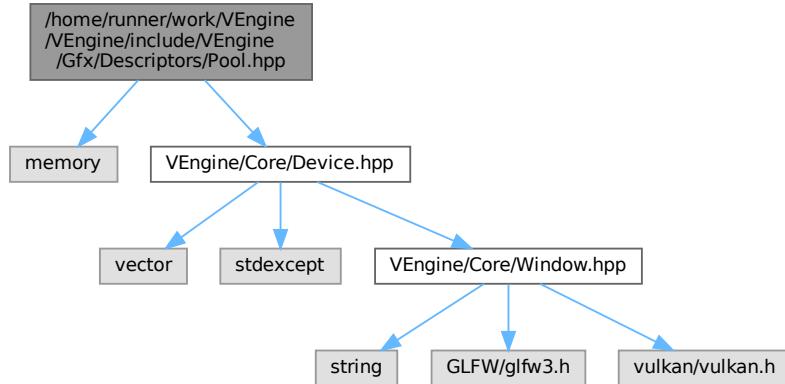
00147     static VkDeviceSize getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
00148         minOffsetAlignment) { return (minOffsetAlignment > 0) ? (instanceSize + minOffsetAlignment - 1) &
00149             ~(minOffsetAlignment - 1) : instanceSize; }
00150
00151     Device& m_device;
00152     void* m_mapped = nullptr;
00153     VkBuffer m_buffer = VK_NULL_HANDLE;
00154     VkDeviceMemory m_memory = VK_NULL_HANDLE;
00155
00156     VkDeviceSize m_bufferSize;
00157     VkDeviceSize m_instanceSize;
00158     uint32_t m_instanceCount;
00159     VkDeviceSize m_alignmentSize;
00160     VkBufferUsageFlags m_usageFlags;
00161     VkMemoryPropertyFlags m_memoryPropertyFlags;
00162
00163 }; // class Buffer
00164 } // namespace ven

```

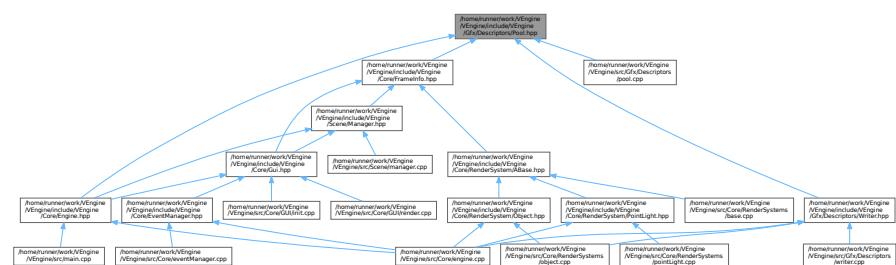
8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Pool.hpp File Reference

This file contains the DescriptorPool class.

```
#include <memory>
#include "VEngine/Core/Device.hpp"
Include dependency graph for Pool.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorPool](#)
Class for descriptor pool.
 - class [ven::DescriptorPool::Builder](#)

Namespaces

- namespace `ven`

Variables

- static constexpr uint32_t `ven::DEFAULT_MAX_SETS` = 1000

8.45.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file [Pool.hpp](#).

8.46 Pool.hpp

[Go to the documentation of this file.](#)

```
00001 /**
00002 /// @file Pool.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/Core/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
00033                     std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00034
00035                 Builder &addPoolSize(const VkDescriptorType descriptorType, const uint32_t count)
00036                     { m_poolSizes.push_back({descriptorType, count}); return *this; }
00037
00038                 Builder &setPoolFlags(const VkDescriptorPoolCreateFlags flags) { m_poolFlags =
00039                     flags; return *this; }
00040
00041                 Builder &setMaxSets(const uint32_t count) { m_maxSets = count; return *this; }
00042
00043         private:
```

```

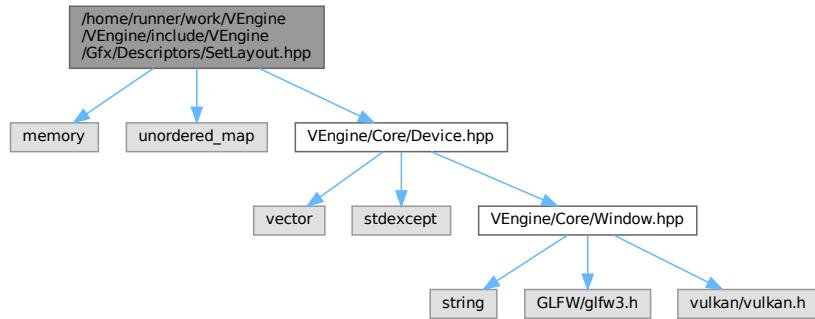
00040             Device &m_device;
00041             std::vector<VkDescriptorPoolSize> m_poolSizes;
00042             uint32_t m_maxSets{DEFAULT_MAX_SETS};
00043             VkDescriptorPoolCreateFlags m_poolFlags{0};
00044
00045         }; // class Builder
00046
00047     DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
00048         const std::vector<VkDescriptorPoolSize> &poolSizes);
00049     ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
00050 }
00051
00052     DescriptorPool(const DescriptorPool &) = delete;
00053     DescriptorPool &operator=(const DescriptorPool &) = delete;
00054
00055     bool allocateDescriptor(VkDescriptorSetLayout descriptorsetLayout, VkDescriptorSet
00056         &descriptor) const;
00057     void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
00058         vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
00059         descriptors.data()); }
00060     void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00061
00062     [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }
00063
00064     private:
00065         Device &m_device;
00066         VkDescriptorPool m_descriptorPool;
00067         friend class DescriptorWriter;
00068
00069     }; // class DescriptorPool
00070
00071 } // namespace ven

```

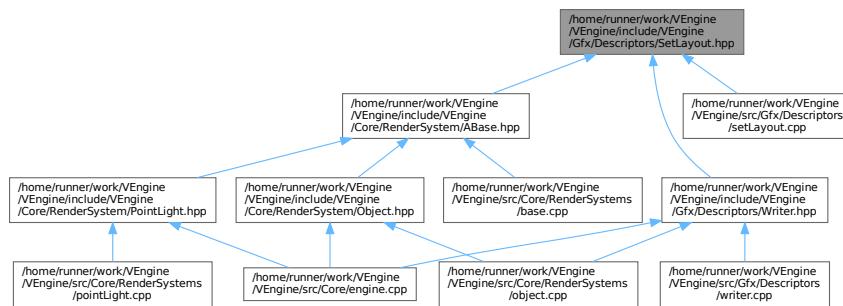
8.47 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/SetLayout.hpp File Reference

This file contains the DescriptorsetLayout class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Core/Device.hpp"
Include dependency graph for SetLayout.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorSetLayout](#)
Class for descriptor set layout.
- class [ven::DescriptorSetLayout::Builder](#)

Namespaces

- namespace [ven](#)

8.47.1 Detailed Description

This file contains the DescriptorsetLayout class.

Definition in file [SetLayout.hpp](#).

8.48 SetLayout.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /**
00003 /**
00004 /**
00005 /**
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Core/Device.hpp"
00013
00014 namespace ven {
00015
00016 /**
00017 /**
00018 /**
00019 /**
00020 /**
00021 class DescriptorsetLayout {
00022
00023     public:
00024

```

```

00025     class Builder {
00026
00027     public:
00028
00029         explicit Builder(Device &device) : m_device{device} {}
00030
00031         Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType,
00032             VkShaderStageFlags stageFlags, uint32_t count = 1);
00033         std::unique_ptr<DescriptorsetLayout> build() const { return
00034             std::make_unique<DescriptorsetLayout>(m_device, m_bindings); }
00035
00036     private:
00037         Device &m_device;
00038         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00039
00040     }; // class Builder
00041
00042     DescriptorsetLayout(Device &device, const std::unordered_map<uint32_t,
00043         VkDescriptorSetLayoutBinding>& bindings);
00044     ~DescriptorsetLayout() { vkDestroyDescriptorsetLayout(m_device.device(),
00045         m_descriptorsetLayout, nullptr); }
00046
00047     DescriptorsetLayout(const DescriptorsetLayout &) = delete;
00048     DescriptorsetLayout &operator=(const DescriptorsetLayout &) = delete;
00049
00050     VkDescriptorsetLayout getDescriptorsetLayout() const { return m_descriptorsetLayout; }
00051
00052     private:
00053         Device &m_device;
00054         VkDescriptorsetLayout m_descriptorsetLayout;
00055         std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00056
00057     friend class DescriptorWriter;
00058
00059 } // namespace ven

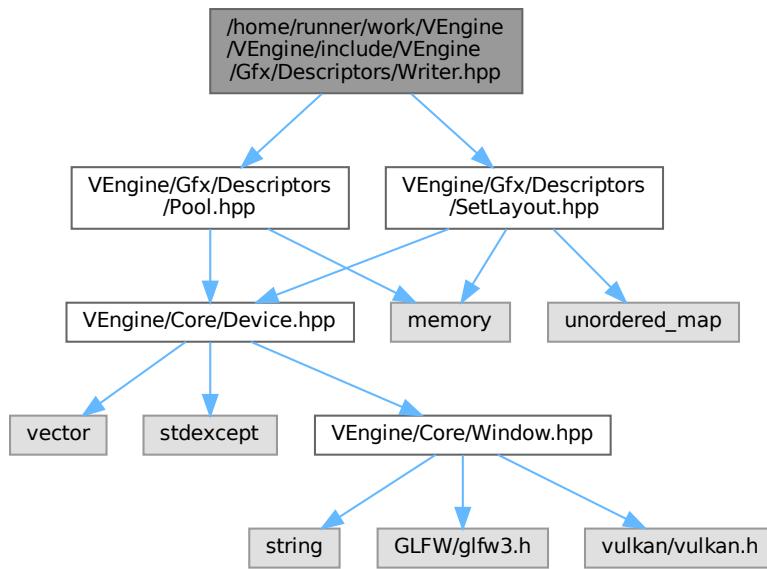
```

8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Descriptors/Writer.hpp File Reference

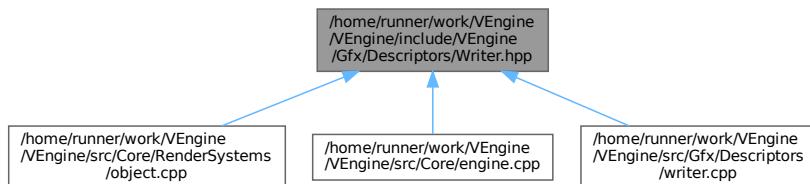
This file contains the DescriptorsWriter class.

```
#include "VEngine/Gfx/Descriptors/Pool.hpp"
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
```

Include dependency graph for Writer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::DescriptorWriter](#)
Class for descriptor writer.

Namespaces

- namespace [ven](#)

8.49.1 Detailed Description

This file contains the `DescriptorsWriter` class.

Definition in file [Writer.hpp](#).

8.50 Writer.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Writer.hpp
00003 /// @brief This file contains the DescriptorsWriter class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00010 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00011
00012 namespace ven {
00013
00014 /**
00015 /// @class DescriptorWriter
00016 /// @brief Class for descriptor writer
00017 /// @namespace ven
00018 /**
00019 class DescriptorWriter {
00020
00021     public:
00022
00023         DescriptorWriter(DescriptorsetLayout &setLayout, DescriptorPool &pool) :
00024             m_setLayout{setLayout}, m_pool{pool} {}
00025             ~DescriptorWriter() = default;
00026
00027         DescriptorWriter(const DescriptorWriter &) = delete;
00028         DescriptorWriter &operator=(const DescriptorWriter &) = delete;
00029
00030         DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00031         DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00032
00033         bool build(VkDescriptorSet &set);
00034         void overwrite(const VkDescriptorSet &set);
00035
00036     private:
00037         DescriptorsetLayout &m_setLayout;
00038         DescriptorPool &m_pool;
00039         std::vector<VkWriteDescriptorSet> m_writes;
00040
00041     }; // class DescriptorWriter
00042
00043 } // namespace ven

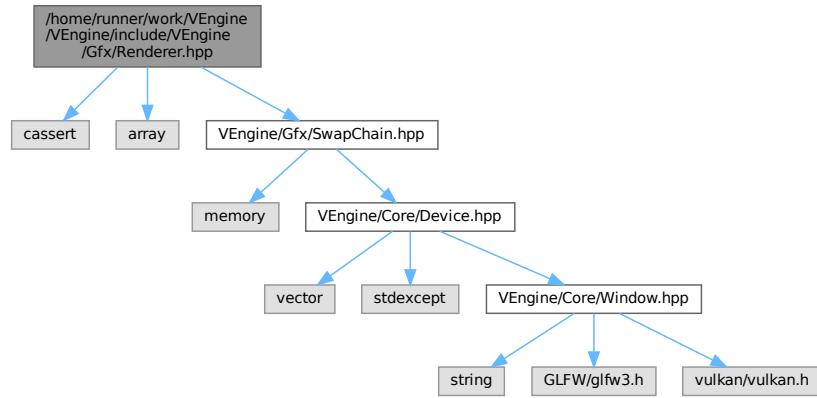
```

8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/← Renderer.hpp File Reference

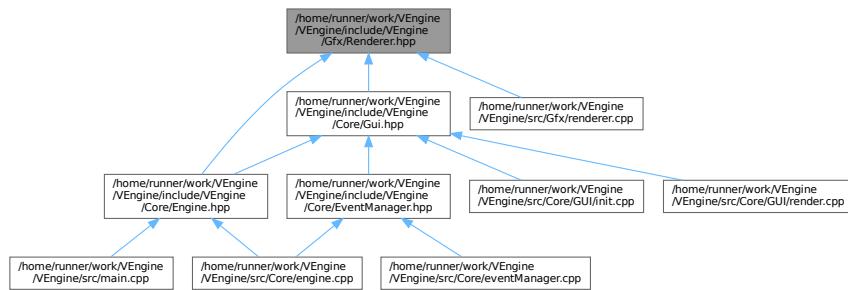
This file contains the Renderer class.

```
#include <cassert>
#include <array>
#include "VEngine/Gfx/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}

8.51.1 Detailed Description

This file contains the Renderer class.

Definition in file [Renderer.hpp](#).

8.52 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 <cassert>
00010 #include <array>
00011
00012 #include "VEngine/Gfx/SwapChain.hpp"
00013
00014 namespace ven {
00015
00016     static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}};
00017     static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0};
00018
00019 /**
00020 /// @class Renderer
00021 /// @brief Class for renderer
00022 /// @namespace ven
00023 /**
00024 class Renderer {
00025
00026     public:
00027
00028         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
00029             recreateSwapChain(); createCommandBuffers(); }
00030         ~Renderer() { freeCommandBuffers(); }
00031
00032         Renderer(const Renderer &) = delete;
00033         Renderer& operator=(const Renderer &) = delete;
00034
00035         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
00036             m_swapChain->getRenderPass(); }
00037         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00038         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00039         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress()
00040             && "cannot get command m_buffer when frame not in progress"); return
00041             m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00042
00043         [[nodiscard]] unsigned long getFrameIndex() const { assert(isFrameInProgress() && "cannot
00044             get frame index when frame not in progress"); return m_currentFrameIndex; }
00045
00046         [[nodiscard]] Window& getWindow() const { return m_window; }
00047
00048         void setClearColorValue(const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR, const
00049             VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
00050                 clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
00051         VkCommandBuffer beginFrame();
00052         void endFrame();
00053         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00054         void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00055
00056     private:
00057
00058         void createCommandBuffers();
00059         void freeCommandBuffers();
00060         void recreateSwapChain();
00061
00062         Window &m_window;
00063         Device &m_device;
00064         std::unique_ptr<SwapChain> m_swapChain;
00065         std::vector<VkCommandBuffer> m_commandBuffers;
00066         std::array<VkClearColorValue, 2> m_clearValues{DEFAULT_CLEAR_COLOR, 1.0F, 0.F};
00067
00068         uint32_t m_currentImageIndex{0};
00069         unsigned long m_currentFrameIndex{0};
00070         bool m_isFrameStarted{false};
00071
00072     }; // class Renderer
00073 } // namespace ven

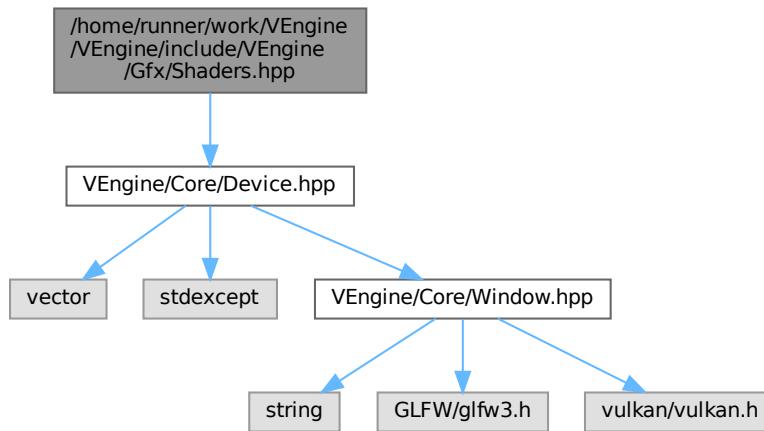
```

8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp File Reference

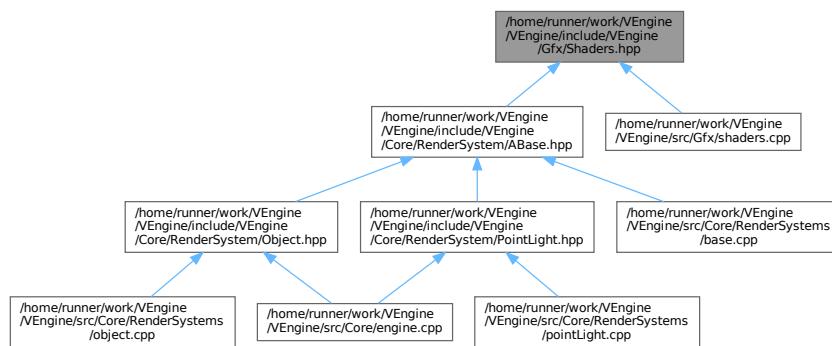
This file contains the Shader class.

```
#include "VEngine/Core/Device.hpp"
```

Include dependency graph for Shaders.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ven::PipelineConfigInfo](#)
- class [ven::Shaders](#)

Class for shaders.

Namespaces

- namespace [ven](#)

Variables

- static constexpr std::string_view ven::SHADERS_BIN_PATH = "build/shaders/"

8.53.1 Detailed Description

This file contains the Shader class.

Definition in file [Shaders.hpp](#).

8.54 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 "VEngine/Core/Device.hpp"
00010
00011 namespace ven {
00012
00013     static constexpr std::string_view SHADERS_BIN_PATH = "build/shaders/";
00014
00015     struct PipelineConfigInfo {
00016         PipelineConfigInfo() = default;
00017         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00018         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00019
00020         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00021         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00022         VKPipelineInputAssemblyCreateInfo inputAssemblyInfo{};
00023         VKPipelineRasterizationStateCreateInfo rasterizationInfo{};
00024         VKPipelineMultisampleStateCreateInfo multisampleInfo{};
00025         VKPipelineColorBlendAttachmentState colorBlendAttachment{};
00026         VKPipelineColorBlendStateCreateInfo colorBlendInfo{};
00027         VKPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00028         std::vector<VkDynamicState> dynamicStateEnables;
00029         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00030         VKPipelineLayout pipelineLayout = nullptr;
00031         VkRenderPass renderPass = nullptr;
00032         uint32_t subpass = 0;
00033     };
00034
00035 /**
00036 /// @class Shaders
00037 /// @brief Class for shaders
00038 /// @namespace ven
00039 /**
00040     class Shaders {
00041
00042     public:
00043
00044         Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
00045             const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
00046             fragFilepath, configInfo); }
00047         ~Shaders();
00048
00049         Shaders(const Shaders&) = delete;
00050         Shaders& operator=(const Shaders&) = delete;
00051
00052         static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00053         void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
00054             VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00055
00056     private:
00057
00058         static std::vector<char> readFile(const std::string &filename);
00059         void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo);

```

```

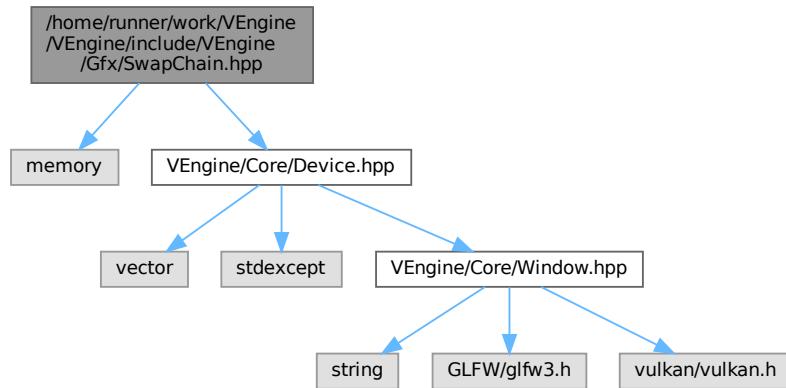
00057     void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
00058     const;
00059     Device& m_device;
00060     VkPipeline m_graphicsPipeline{nullptr};
00061     VkShaderModule m_vertShaderModule{nullptr};
00062     VkShaderModule m_fragShaderModule{nullptr};
00063   };
00064   // class Shaders
00065
00066 } // namespace ven

```

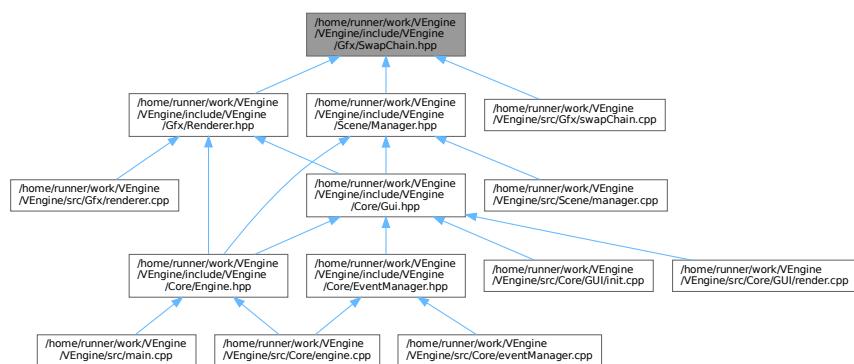
8.55 /home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp File Reference

This file contains the Shader class.

```
#include <memory>
#include "VEngine/Core/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

8.55.1 Detailed Description

This file contains the Shader class.

Definition in file [SwapChain.hpp](#).

8.56 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 <memory>
00010
00011 #include "VEngine/Core/Device.hpp"
00012
00013 namespace ven {
00014
00015     static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00016
00017 /**
00018     /// @class SwapChain
00019     /// @brief Class for swap chain
00020     /// @namespace ven
00021 /**
00022     class SwapChain {
00023
00024         public:
00025
00026             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : m_device(deviceRef),
00027             m_windowExtent(windowExtentRef) { init(); }
00028             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
00029             previous) : m_device(deviceRef), m_windowExtent(windowExtentRef), m_oldSwapChain(std::move(previous))
00030             { init(); m_oldSwapChain = nullptr; }
00031             ~SwapChain();
00032
00033             [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return
00034                 m_swapChainFrameBuffers[index]; }
00035             [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }
00036             [[nodiscard]] VkImageView getImageView(const int index) const { return
00037                 m_swapChainImageViews[static_cast<unsigned long>(index)]; }
00038             [[nodiscard]] size_t getCount() const { return m_swapChainImages.size(); }
00039             [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_swapChainImageFormat; }
00040             [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00041             [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }

```

```

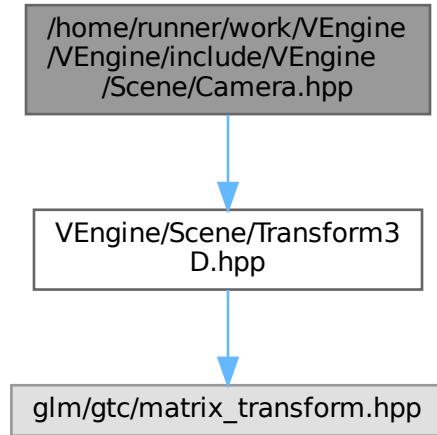
00040     [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00041
00042     [[nodiscard]] float extentAspectRatio() const { return
00043         static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
00044
00045     VkResult acquireNextImage(uint32_t *imageIndex) const;
00046     VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00047
00048     [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return
00049         m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat ==
00050         swapChain.m_swapChainDepthFormat; }
00051
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
00068         VkFormat m_swapChainImageFormat{};
00069         VkFormat m_swapChainDepthFormat{};
00070         VkExtent2D m_swapChainExtent{};
00071
00072         std::vector<VkFramebuffer> m_swapChainFrameBuffers;
00073         VkRenderPass m_renderPass{};
00074
00075         std::vector<VKImage> m_depthImages;
00076         std::vector<VkDeviceMemory> m_depthImageMemory;
00077         std::vector<VkImageView> m_depthImageViews;
00078         std::vector<VKImage> m_swapChainImages;
00079         std::vector<VkImageView> m_swapChainImageViews;
00080
00081         Device &m_device;
00082         VkExtent2D m_windowExtent;
00083
00084         VkSwapchainKHR m_swapChain{};
00085         std::shared_ptr<SwapChain> m_oldSwapChain;
00086
00087         std::vector<VkSemaphore> m_imageAvailableSemaphores;
00088         std::vector<VkSemaphore> m_renderFinishedSemaphores;
00089         std::vector<VkFence> m_inFlightFences;
00090         std::vector<VkFence> m_imagesInFlight;
00091         size_t m_currentFrame{0};
00092
00093     }; // class SwapChain
00094
00095 } // namespace ven

```

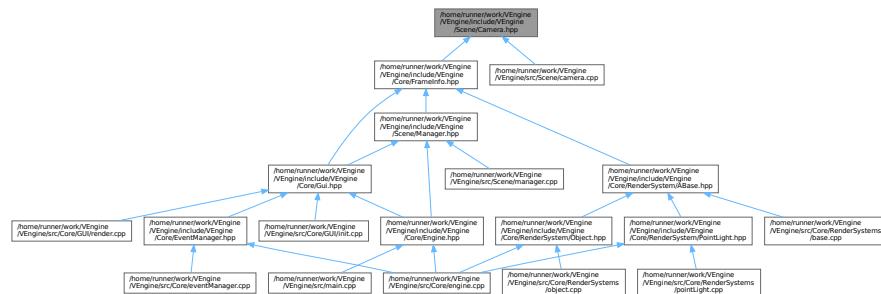
8.57 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/← Camera.hpp File Reference

This file contains the Camera class.

```
#include "VEngine/Scene/Transform3D.hpp"
Include dependency graph for Camera.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Camera](#)
Class for camera.

Namespaces

- namespace [ven](#)

Variables

- static constexpr glm::vec3 [ven::DEFAULT_POSITION](#) {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 [ven::DEFAULT_ROTATION](#) {0.F, 0.F, 0.F}
- static constexpr float [ven::DEFAULT_FOV](#) = glm::radians(50.0F)
- static constexpr float [ven::DEFAULT_NEAR](#) = 0.1F
- static constexpr float [ven::DEFAULT_FAR](#) = 100.0F
- static constexpr float [ven::DEFAULT_MOVE_SPEED](#) = 3.F
- static constexpr float [ven::DEFAULT_LOOK_SPEED](#) = 1.5F

8.57.1 Detailed Description

This file contains the Camera class.

Definition in file [Camera.hpp](#).

8.58 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/Scene/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
00039             near, float far);
00040         void setPerspectiveProjection(float aspect);
00041         void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up = {0.F, -1.F,
00042             0.F});
00043         void setViewTarget(const glm::vec3 position, const glm::vec3 target, const glm::vec3 up =
00044             {0.F, -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00045         void setViewXYZ(glm::vec3 position, glm::vec3 rotation);
00046         void setFov(const float fov) { m_fov = fov; }
00047         void setNear(const float near) { m_near = near; }
00048         void setFar(const float far) { m_far = far; }
00049         void setMoveSpeed(const float moveSpeed) { m_moveSpeed = moveSpeed; }
00050         void setLookSpeed(const float lookSpeed) { m_lookSpeed = lookSpeed; }
00051
00052         [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00053         [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00054         [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00055         [[nodiscard]] float getFov() const { return m_fov; }
00056         [[nodiscard]] float getNear() const { return m_near; }
00057         [[nodiscard]] float getFar() const { return m_far; }
00058         [[nodiscard]] float getMoveSpeed() const { return m_moveSpeed; }
00059         [[nodiscard]] float getLookSpeed() const { return m_lookSpeed; }
00060
00061     private:
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};

```

```

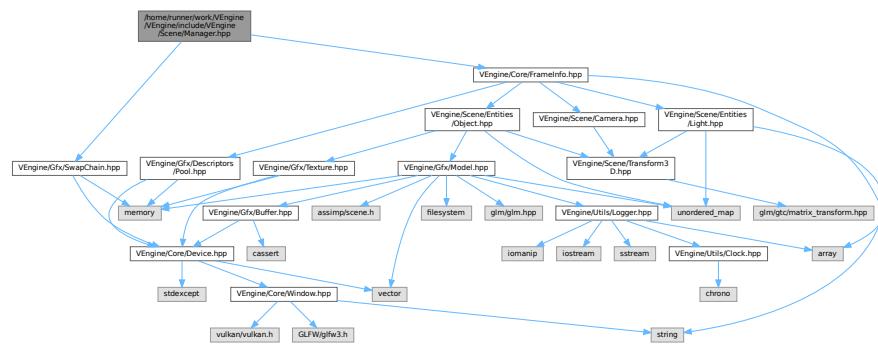
00067     glm::mat4 m_projectionMatrix{1.F};
00068     glm::mat4 m_viewMatrix{1.F};
00069     glm::mat4 m_inverseViewMatrix{1.F};
00070
00071 } // class Camera
00072
00073 } // namespace ven

```

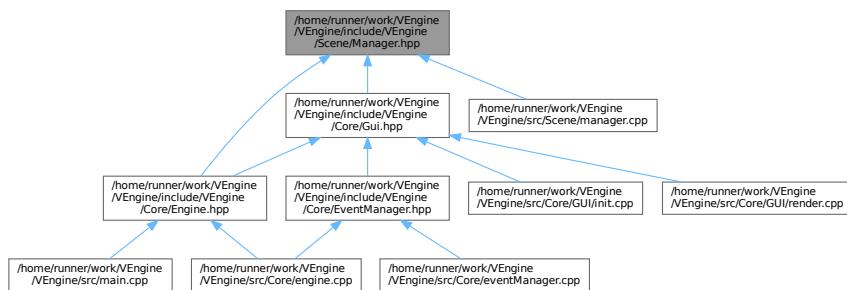
8.59 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/SceneManager.hpp File Reference

This file contains the SceneManager class.

```
#include "VEngine/Core/FrameInfo.hpp"
#include "VEngine/Gfx/SwapChain.hpp"
Include dependency graph for Manager.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::SceneManager](#)

Class for object manager.

Namespaces

- namespace [ven](#)

8.59.1 Detailed Description

This file contains the SceneManager class.

Definition in file [Manager.hpp](#).

8.60 Manager.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002  * @file Manager.hpp
00003  * @brief This file contains the SceneManager class
00004  * @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include "VEngine/Core/FrameInfo.hpp"
00010 #include "VEngine/Gfx/SwapChain.hpp"
00011
00012 namespace ven {
00013
00014 /**
00015  * @class SceneManager
00016  * @brief Class for object manager
00017  * @namespace ven
00018 /**
00019 class SceneManager {
00020
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     void addObject(const std::unique_ptr<Object> &object) { m_objects.insert({object->getId(),
00031         std::move(*object)}); }
00032     void addLight(const std::unique_ptr<Light> &light) { m_lights.insert({light->getId(),
00033         std::move(*light)}); }
00034
00035     void destroyObject(const unsigned int objectId) { m_objects.erase(objectId); }
00036     void destroyLight(const unsigned int lightId) { m_lights.erase(lightId); }
00037     void destroyEntity(std::vector<unsigned int> *objectIds, std::vector<unsigned int>
00038         *lightsIds);
00039
00040     [[nodiscard]] VkDescriptorBufferInfo getBufferInfoForObject(const int frameIndex, const
00041         unsigned int objectId) const { return m_uboBuffers.at(static_cast<long unsigned
00042             int>(frameIndex))>descriptorInfoForIndex(objectId); }
00043     Object::Map& getObjects() { return m_objects; }
00044     Light::Map& getLights() { return m_lights; }
00045     std::vector<std::unique_ptr<Buffer>> &getUboBuffers() { return m_uboBuffers; }
00046     std::shared_ptr<Texture> getTextureDefault() { return m_textureDefault; }
00047     [[nodiscard]] bool getDestroyState() const { return m_destroyState; }
00048
00049     private:
00050     unsigned int m_currentObjId{0};
00051     unsigned int m_currentLightId{0};
00052     std::shared_ptr<Texture> m_textureDefault;
00053     Object::Map m_objects;
00054     Light::Map m_lights;
00055     std::vector<std::unique_ptr<Buffer>> m_uboBuffers{MAX_FRAMES_IN_FLIGHT};
00056     bool m_destroyState{false};
00057
00058 }; // class SceneManager
00059
00060 } // namespace ven

```

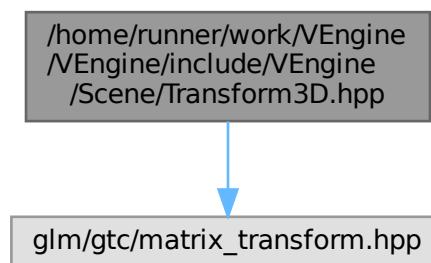
8.61 /home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp File Reference ↵

Transform3D.hpp File Reference

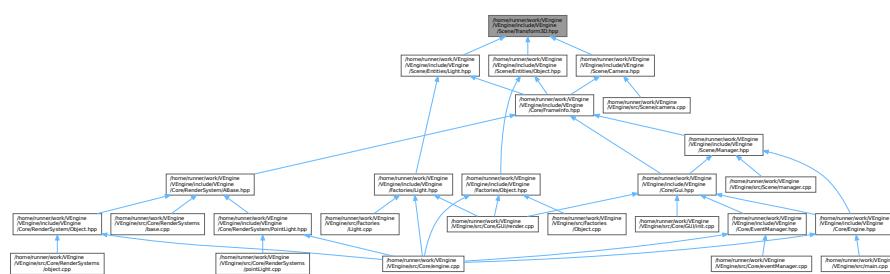
This file contains the Transform3D class.

```
#include <glm/gtc/matrix_transform.hpp>
```

Include dependency graph for Transform3D.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Transform3D](#)

Class for 3D transformation.

Namespaces

- namespace [ven](#)

8.61.1 Detailed Description

This file contains the Transform3D class.

Definition in file [Transform3D.hpp](#).

8.62 Transform3D.hpp

[Go to the documentation of this file.](#)

```

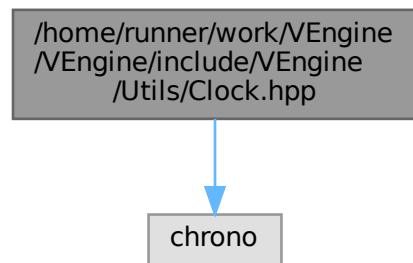
00001 /**
00002 /// @file Transform3D.hpp
00003 /// @brief This file contains the Transform3D class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <glm/gtc/matrix_transform.hpp>
00010
00011 namespace ven {
00012
00013 /**
00014 /// @class Transform3D
00015 /// @brief Class for 3D transformation
00016 /// @namespace ven
00017 /**
00018 class Transform3D {
00019
00020     public:
00021
00022         [[nodiscard]] glm::mat4 transformMatrix() const {
00023             auto rotationMatrix = glm::mat4(1.0F);
00024
00025             rotationMatrix = rotate(rotationMatrix, rotation.x, glm::vec3(1.0F, 0.0F, 0.0F));
00026             rotationMatrix = rotate(rotationMatrix, rotation.y, glm::vec3(0.0F, 1.0F, 0.0F));
00027             rotationMatrix = rotate(rotationMatrix, rotation.z, glm::vec3(0.0F, 0.0F, 1.0F));
00028
00029             const glm::mat4 scaleMatrix = glm::scale(glm::mat4(1.0F), scale);
00030             const glm::mat4 translationMatrix = translate(glm::mat4(1.0F), translation);
00031
00032             return translationMatrix * rotationMatrix * scaleMatrix;
00033         }
00034         [[nodiscard]] glm::mat3 normalMatrix() const { return
00035             transpose(inverse(glm::mat3(transformMatrix())));
00036         }
00037         glm::vec3 translation{};
00038         glm::vec3 scale{};
00039         glm::vec3 rotation{};
00040     }; // class Transform3D
00041
00042 } // namespace ven

```

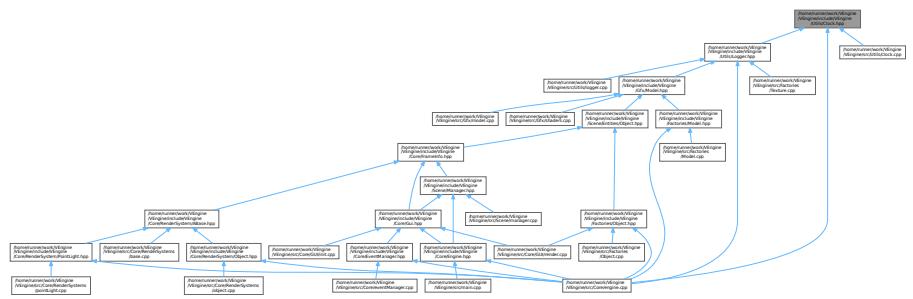
8.63 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Clock.hpp File Reference

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>

8.63.1 Detailed Description

This file contains the Clock class.

Definition in file [Clock.hpp](#).

8.64 Clock.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /**
00003 /**
00004 /**
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 /**
00017 /**
00018 /**
00019 /**
00020 class Clock {
00021

```

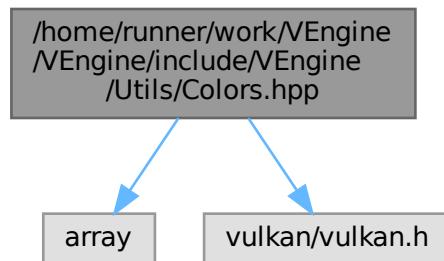
```

00022     public:
00023
00024     Clock() { start(); }
00025     ~Clock() = default;
00026
00027     Clock(const Clock&) = delete;
00028     Clock& operator=(const Clock&) = delete;
00029
00030     void start() { m_startTime = std::chrono::high_resolution_clock::now(); }
00031     void stop();
00032     void resume();
00033     void update();
00034     static std::chrono::high_resolution_clock::time_point now() { return
00035         std::chrono::high_resolution_clock::now(); }
00036     [[nodiscard]] float getDeltaTime() const { return m_deltaTime.count(); }
00037     [[nodiscard]] float getDeltaTimeMS() const { return m_deltaTime.count() * 1000.F; }
00038     [[nodiscard]] float getFPS() const { return 1.F / m_deltaTime.count(); }
00039
00040     private:
00041
00042         TimePoint mStartTime;
00043         TimePoint mStopTime;
00044         std::chrono::duration<float> mDeltaTime{0.F};
00045
00046         bool m_isStopped{false};
00047
00048     }; // class Clock
00049
00050 } // namespace ven

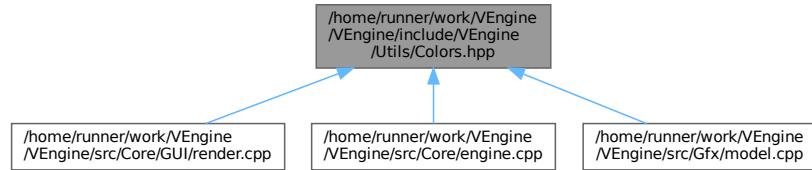
```

8.65 /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

8.66 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;
  
```

```

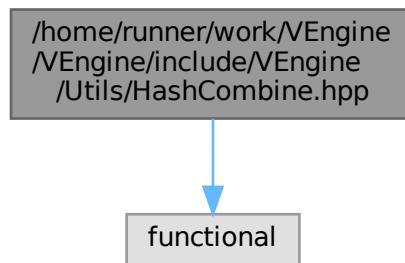
00035     static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F };
00036     static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } };
00037
00038     static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
00039     static constexpr glm::vec4 GREEN_4 = { 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},

```

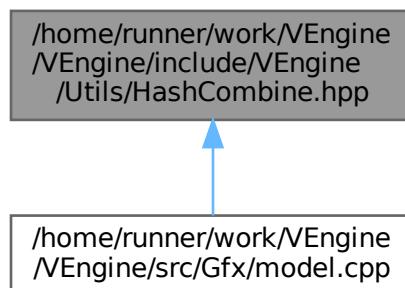
```
00122         {"Teal", TEAL_3},
00123         {"Navy", NAVY_3},
00124         {"Fuchsia", FUCHSIA_3},
00125         {"Night Blue", NIGHT_BLUE_3},
00126         {"Sky Blue", SKY_BLUE_3},
00127         {"Sunset", SUNSET_3}
00128     }};
00129
00130     static constexpr std::array<std::pair<const char *, 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>
00154     COLOR_PRESETS_VK = {{
00155         {"White", WHITE_V},
00156         {"Black", BLACK_V},
00157         {"Red", RED_V},
00158         {"Green", GREEN_V},
00159         {"Blue", BLUE_V},
00160         {"Yellow", YELLOW_V},
00161         {"Cyan", CYAN_V},
00162         {"Magenta", MAGENTA_V},
00163         {"Silver", SILVER_V},
00164         {"Gray", GRAY_V},
00165         {"Maroon", MAROON_V},
00166         {"Olive", OLIVE_V},
00167         {"Lime", LIME_V},
00168         {"Aqua", AQUA_V},
00169         {"Teal", TEAL_V},
00170         {"Navy", NAVY_V},
00171         {"Fuchsia", FUCHSIA_V},
00172         {"Night Blue", NIGHT_BLUE_V},
00173         {"Sky Blue", SKY_BLUE_V},
00174         {"Sunset", SUNSET_V}
00175     }};
00176 } // class Colors
00177
00178 } // namespace ven
```

8.67 /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)

8.68 HashCombine.hpp

[Go to the documentation of this file.](#)

```

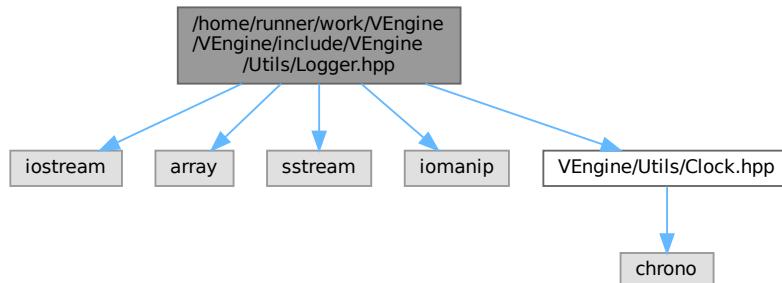
00001 /**
00002 /// @file HashCombine.hpp
00003 /// @brief
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018 } // namespace ven

```

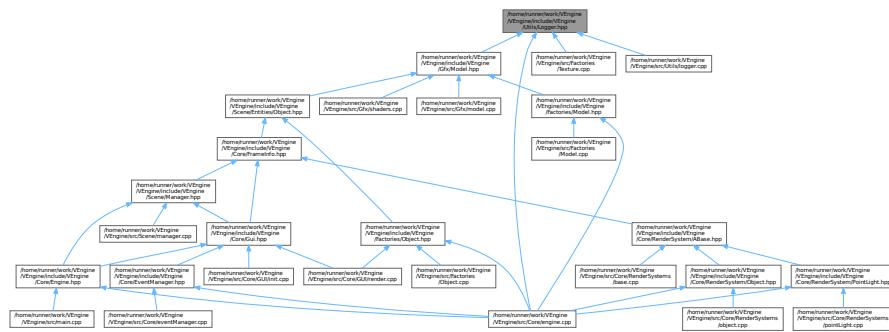
8.69 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp File Reference

This file contains the Logger class.

```
#include <iostream>
#include <array>
#include <sstream>
#include <iomanip>
#include "VEngine/Utils/Clock.hpp"
Include dependency graph for Logger.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [ven::Logger](#)

Namespaces

- namespace [ven](#)

Enumerations

- enum class [ven::LogLevel](#) : uint8_t { [ven::INFO](#) , [ven::WARNING](#) }

8.69.1 Detailed Description

This file contains the Logger class.

Definition in file [Logger.hpp](#).

8.70 Logger.hpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 /// @file Logger.hpp
00003 /// @brief This file contains the Logger class
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <iostream>
00010 #include <array>
00011 #include <sstream>
00012 #include <iomanip>
00013
00014 #ifdef _WIN32
00015     #include <windows.h>
00016 #endif
00017
00018 #include "VEngine/Utils/Clock.hpp"
00019
00020
00021 namespace ven {
  
```

```

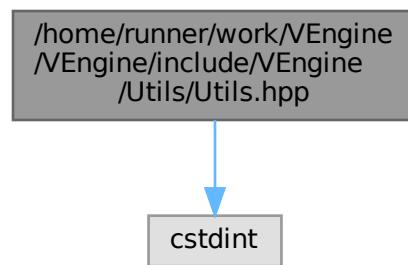
00022
00023     enum class LogLevel : uint8_t {
00024         INFO,
00025         WARNING
00026     };
00027
00028     class Logger {
00029         public:
00030             static Logger& getInstance() { static Logger instance; return instance; }
00031
00032             template <typename Func>
00033             static void logExecutionTime(const std::string& message, Func&& func)
00034             {
00035                 Clock clock;
00036                 func();
00037                 clock.update();
00038                 const float duration = clock.getDeltaTimeMS();
00039
00040                 std::cout << getColorForDuration(duration) << formatLogMessage(LogLevel::INFO, message +
00041 " took " + std::to_string(duration) + " ms") << LOG_LEVEL_COLOR.at(3);
00042             }
00043
00044             static void logWarning(const std::string& message) { std::cout << LOG_LEVEL_COLOR.at(2) <<
00045                 formatLogMessage(LogLevel::WARNING, message) << LOG_LEVEL_COLOR.at(3); }
00046
00047         private:
00048             static constexpr std::array<const char*, 2> LOG_LEVEL_STRING = {"INFO", "WARNING"};
00049             static constexpr std::array<const char*, 4> LOG_LEVEL_COLOR = {"\033[31m", "\033[32m",
00050             "\033[33m", "\033[0m\n"};
00051
00052             Logger();
00053
00054             static const char* getColorForDuration(const float duration) { return duration < 20.0F ? LOG_LEVEL_COLOR.at(1) : (duration < 90.0F ? LOG_LEVEL_COLOR.at(2) : LOG_LEVEL_COLOR.at(0)); }
00055
00056             [[nodiscard]] static std::string formatLogMessage(LogLevel level, const std::string&
00057             message);
00058     }; // class Logger
00059
00060 } // namespace ven

```

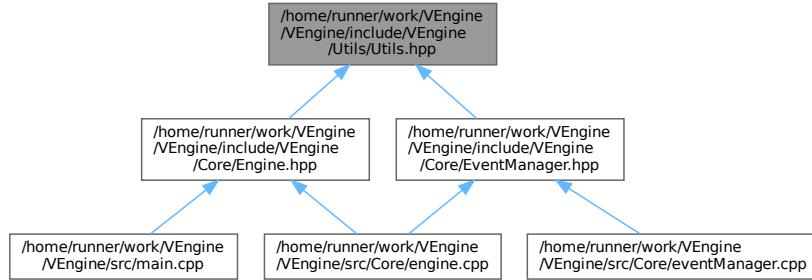
8.71 /home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp File Reference

This file contains utils for VEngine.

```
#include <cstdint>
Include dependency graph for Utils.hpp:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [ven](#)

Enumerations

- enum [ven::ENGINE_STATE](#) : uint8_t { [ven::EDITOR](#) = 0 , [ven::PLAYER](#) = 1 , [ven::PAUSED](#) = 2 , [ven::EXIT](#) = 3 }

8.71.1 Detailed Description

This file contains utils for VEngine.

Definition in file [Utils.hpp](#).

8.72 Utils.hpp

[Go to the documentation of this file.](#)

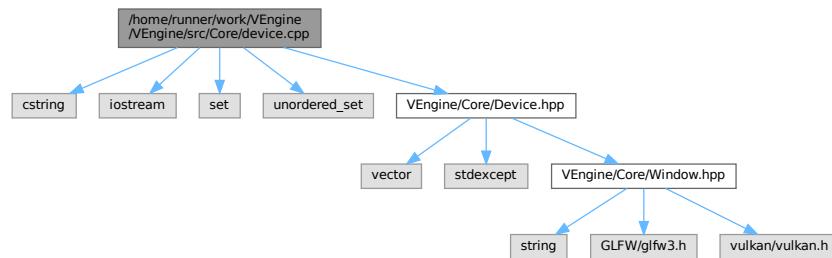
```

00001 /**
00002 /// @file Utils.hpp
00003 /// @brief This file contains utils for VEngine
00004 /// @namespace ven
00005 /**
00006
00007 #pragma once
00008
00009 #include <cstdint>
00010
00011 namespace ven {
00012
00013     enum ENGINE_STATE : uint8_t {
00014         EDITOR = 0,
00015         PLAYER = 1,
00016         PAUSED = 2,
00017         EXIT = 3
00018     };
00019
00020 } // namespace ven
  
```

8.73 /home/runner/work/VEngine/VEngine/README.md File Reference

8.74 /home/runner/work/VEngine/VEngine/src/Core/device.cpp File Reference

```
#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>
#include "VEngine/Core/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)

8.74.1 Function Documentation

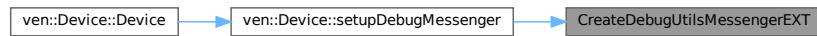
8.74.1.1 CreateDebugUtilsMessenger()

```
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:



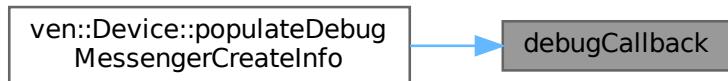
8.74.1.2 debugCallback()

```
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:



8.74.1.3 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:



8.75 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/Core/Device.hpp"
00007
00008 static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT
00009   messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const
00010   VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
00011 {
00012     (void) pUserData; (void) messageSeverity; (void) messageType;
00013     std::cerr << "validation layer: " << pCallbackData->pMessage << '\n';
00014     return VK_FALSE;
00015 }
00016
00017 VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const
00018   VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator,
00019   VkDebugUtilsMessengerEXT *pDebugMessenger)
00020 {
00021     if (const auto func =
00022       reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
00023         "vkCreateDebugUtilsMessengerEXT")); func != nullptr) {
00024         func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00025     }
00026     return VK_ERROR_EXTENSION_NOT_PRESENT;
00027 }
00028
00029 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
00030   debugMessenger, const VkAllocationCallbacks *pAllocator)
00031 {
00032     if (const auto func =
00033       reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
00034         "vkDestroyDebugUtilsMessengerEXT")); func != nullptr) {
00035         func(instance, debugMessenger, pAllocator);
00036     }
00037 }
00038
00039 ven::Device::Device(Window &window) : m_window{window}
00040 {
00041     createInstance();
00042     setupDebugMessenger();
00043     createSurface();
00044     pickPhysicalDevice();
00045     createLogicalDevice();
00046     createCommandPool();
00047 }
00048
00049 ven::Device::~Device()
00050 {
00051     vkDestroyCommandPool(m_device, m_commandPool, nullptr);
00052     vkDestroyDevice(m_device, nullptr);
00053
00054     if (enableValidationLayers) {
00055         DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
00056     }
00057     vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
00058     vkDestroyInstance(m_instance, nullptr);
00059 }
00060
00061 void ven::Device::createInstance()
00062 {
00063     if (enableValidationLayers && !checkValidationLayerSupport()) {
00064         throw std::runtime_error("validation layers requested, but not available!");
00065     }
00066
00067     constexpr VkApplicationInfo appInfo = {
00068         .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO,
00069         .pNext = nullptr,
00070         .pApplicationName = "VEngine App",
00071         .applicationVersion = VK_MAKE_VERSION(1, 0, 0),
00072         .pEngineName = "VEngine",
00073         .engineVersion = VK_MAKE_VERSION(1, 0, 0),
00074         .apiVersion = VK_API_VERSION_1_0
00075     };
00076
00077     VkInstanceCreateInfo createInfo = {};
00078     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00079     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] =
00127     findQueueFamilies(m_physicalDevice);
00128
00129     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00130     const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, presentFamily};
00131     float queuePriority = 1.0F;
00132
00133     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00134         VkDeviceQueueCreateInfo queueCreateInfo = {};
00135         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00136         queueCreateInfo.queueFamilyIndex = queueFamily;
00137         queueCreateInfo.queueCount = 1;
00138         queueCreateInfo.pQueuePriorities = &queuePriority;
00139         queueCreateInfos.push_back(queueCreateInfo);
00140     }
00141
00142     VkPhysicalDeviceFeatures deviceFeatures = {};
00143     deviceFeatures.samplerAnisotropy = VK_TRUE;
00144
00145     VkDeviceCreateInfo createInfo = {};
00146     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00147
00148     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00149     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00150
00151     createInfo.pEnabledFeatures = &deviceFeatures;
00152     createInfo.enabledExtensionCount = static_cast<uint32_t>(m_deviceExtensions.size());
00153     createInfo.ppEnabledExtensionNames = m_deviceExtensions.data();
00154
00155     // might not really be necessary anymore because device specific validation layers
00156     // have been deprecated
00157     if (enableValidationLayers) {
00158         createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
00159         createInfo.ppEnabledLayerNames = m_validationLayers.data();
00160     } 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
00206 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00207 {
00208     createInfo = {};
00209     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00210     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
00211                                     VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00212     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
00213                             VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
00214                             VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00215     createInfo.pfnUserCallback = debugCallback;
00216     createInfo.pUserData = nullptr; // Optional
00217 }
00218
00219 void ven::Device::setupDebugMessenger()
00220 {
00221     if (!enableValidationLayers) { return; }
00222     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00223     populateDebugMessengerCreateInfo(createInfo);
00224     if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) !=
00225         VK_SUCCESS) {
00226         throw std::runtime_error("failed to set up debug messenger!");
00227     }
00228 }
00229
00230 bool ven::Device::checkValidationLayerSupport() const
00231 {
00232     uint32_t layerCount = 0;
00233     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00234
00235     std::vector<VkLayerProperties> availableLayers(layerCount);
00236     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00237
00238     for (const char *validationLayer : m_validationLayers) {
00239         bool layerFound = false;
00240
00241         for (const auto &[layerName, specVersion, implementationVersion, description] :
00242             availableLayers) {
00243             if (strcmp(layerName, validationLayer) == 0) {
00244                 layerFound = true;
00245                 break;
00246             }
00247         }
00248         if (layerFound) {
00249             m_validationLayers.erase(std::remove(m_validationLayers.begin(),
00250                                     m_validationLayers.end(),
00251                                     validationLayer));
00252         }
00253     }
00254 }

```

```

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
00283      &required : requiredExtensions) {
00284         std::cout << "\t" << required << '\n';
00285         if (!available.contains(required)) {
00286             throw std::runtime_error("Missing required glfw extension");
00287         }
00288     }
00289 }
00290
00291 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00292 {
00293     uint32_t extensionCount = 0;
00294     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00295
00296     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00297     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
00298                                         availableExtensions.data());
00299
00300     std::set<std::string> requiredExtensions(m_deviceExtensions.begin(), m_deviceExtensions.end());
00301     for (const auto &[extensionName, specVersion] : availableExtensions) {
00302         requiredExtensions.erase(extensionName);
00303     }
00304
00305     return requiredExtensions.empty();
00306 }
00307
00308 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00309 {
00310     QueueFamilyIndices indices;
00311     uint32_t queueFamilyCount = 0;
00312     uint32_t index = 0;
00313     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00314     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00315     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00316
00317     for (const auto &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
00318          queueFamilies) {
00319         if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00320             indices.graphicsFamily = index;
00321             indices.graphicsFamilyHasValue = true;
00322         }
00323         VkBool32 presentSupport = 0U;
00324         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
00325         if (queueCount > 0 && (presentSupport != 0U)) {
00326             indices.presentFamily = index;
00327             indices.presentFamilyHasValue = true;
00328         }
00329     }
00330 }
```

```

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
tiling, const VkFormatFeatureFlags features) const
00356 {
00357     for (const VkFormat format : candidates) {
00358         VKFormatProperties props;
00359         vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
00360         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00361             return format;
00362         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
features) {
00363             return format;
00364         }
00365     }
00366     throw std::runtime_error("failed to find supported format!");
00367 }
00368
00369 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
properties) const
00370 {
00371     VkPhysicalDeviceMemoryProperties memProperties;
00372     vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);
00373
00374     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00375         if (((typeFilter & (1 << i)) != 0U) &&
00376             (memProperties.memoryTypes[i].propertyFlags & properties) == properties) {
00377             return i;
00378         }
00379     }
00380
00381     throw std::runtime_error("failed to find suitable m_memory type!");
00382 }
00383
00384 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
VkMemoryPropertyFlags properties, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00385 {
00386     VkBufferCreateInfo bufferInfo{};
00387     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00388     bufferInfo.size = size;
00389     bufferInfo.usage = usage;
00390     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00391
00392     if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00393         throw std::runtime_error("failed to create vertex m_buffer!");
00394     }
00395
00396     VkMemoryRequirements memRequirements;
00397     vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);
00398
00399     const VkMemoryAllocateInfo allocInfo{
00400         .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,
00401         .pNext = nullptr,
00402         .allocationSize = memRequirements.size,
00403         .memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties)
00404     };
00405
00406     if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00407         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00408 }

```

```

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{};

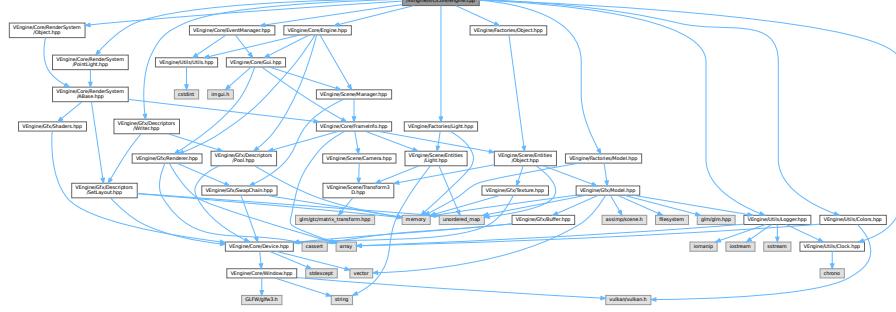

```

```
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
00505     VkImageLayout oldLayout, const VkImageLayout newLayout, const uint32_t mipLevels, const uint32_t
00506     layerCount) const {
00507     // uses an image memory barrier transition image layouts and transfer queue
00508     // family ownership when VK_SHARING_MODE_EXCLUSIVE is used. There is an
00509     // equivalent buffer memory barrier to do this for buffers
00510     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00511
00512     VkImageMemoryBarrier barrier{};
00513     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00514     barrier.oldLayout = oldLayout;
00515     barrier.newLayout = newLayout;
00516
00517     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00518     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00519
00520     barrier.image = image;
00521     barrier.subresourceRange.baseMipLevel = 0;
00522     barrier.subresourceRange.levelCount = mipLevels;
00523     barrier.subresourceRange.baseArrayLayer = 0;
00524     barrier.subresourceRange.layerCount = layerCount;
00525
00526     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00527         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00528         if (format == VK_FORMAT_D32_SFLOAT_S8_UINT || format == VK_FORMAT_D24_UNORM_S8_UINT) {
00529             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00530         } else {
00531             barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00532         }
00533
00534     VkPipelineStageFlags sourceStage = 0;
00535     VkPipelineStageFlags destinationStage = 0;
00536
00537     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00538         barrier.srcAccessMask = 0;
00539         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00540
00541         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00542         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00543     } else if (
00544         oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00545         barrier.srcAccessMask = 0;
00546         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00547
00548         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00549         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00550     } else if (
00551         oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL &&
00552         newLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00553         barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00554         barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00555
00556         sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00557         destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00558     } else if (
00559         oldLayout == VK_IMAGE_LAYOUT_UNDEFINED &&
00560         newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00561         barrier.srcAccessMask = 0;
00562         barrier.dstAccessMask =
00563             VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00564
00565         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00566         destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00567     } else {
00568         throw std::invalid_argument("unsupported layout transition!");
00569     }
00570     vkCmdPipelineBarrier(
00571         commandBuffer,
00572         sourceStage,
00573         destinationStage,
00574         0,
00575         0,
00576         nullptr,
```

```
00576     0,  
00577     nullptr,  
00578     1,  
00579     &barrier);  
00580  
00581 endSingleTimeCommands(commandBuffer);  
00582 }
```

8.76 /home/runner/work/VEngine/VEngine/src/Core/engine.cpp File Reference

```
#include "VEngine/Core/Engine.hpp"
#include "VEngine/Core/EventManager.hpp"
#include "VEngine/Core/RenderSystem/Object.hpp"
#include "VEngine/Core/RenderSystem/PointLight.hpp"
#include "VEngine/Gfx/Descriptors/Writer.hpp"
#include "VEngine/Factories/Light.hpp"
#include "VEngine/Factories/Object.hpp"
#include "VEngine/Factories/Model.hpp"
#include "VEngine/Utils/Clock.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Utils/Logger.hpp"
Include dependency graph for engine.cpp:
```



8.77 engine.cpp

[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Core/Engine.hpp"
00002 #include "VEngine/Core/EventManager.hpp"
00003 #include "VEngine/Core/RenderSystem/Object.hpp"
00004 #include "VEngine/Core/RenderSystem/PointLight.hpp"
00005 #include "VEngine/Gfx/Descriptors/Writer.hpp"
00006 #include "VEngine/Factories/Light.hpp"
00007 #include "VEngine/Factories/Object.hpp"
00008 #include "VEngine/Factories/Model.hpp"
00009 #include "VEngine/Utils/Clock.hpp"
00010 #include "VEngine/Utils/Colors.hpp"
00011 #include "VEngine/Utils/Logger.hpp"
00012
00013 ven::Engine::Engine(const uint32_t width, const uint32_t height, const std::string &title) :
    m_state(EDITOR), m_window(width, height, title) {
00014     m_gui.init(m_window.getGLFWWindow(), m_device.getInstance(), &m_device,
    m_renderer.getSwapChainRenderPass());
00015     m_globalPool =
        DescriptorPool::Builder(m_device).setMaxSets(MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    MAX_FRAMES_IN_FLIGHT).build();
00016     m_framePools.resize(MAX_FRAMES_IN_FLIGHT);
00017     const auto framePoolBuilder = DescriptorPool::Builder(m_device)
00018         .setMaxSets(1000)
```

```

00019             .addPoolSize(VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1000)
00020             .addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1000)
00021             .setPoolFlags(VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT);
00022     for (auto & framePool : m_framePools) {
00023         framePool = framePoolBuilder.build();
00024     }
00025     Logger::getInstance();
00026     loadObjects();
00027 }
00028
00029 void ven::Engine::loadObjects()
00030 {
00031     constexpr std::array lightColors{Colors::RED_4, Colors::GREEN_4, Colors::BLUE_4, Colors::YELLOW_4,
00032     Colors::CYAN_4, Colors::MAGENTA_4};
00033     const std::unordered_map<std::string, std::shared_ptr<Model>> modelCache =
00034     ModelFactory::loadAll(m_device, "assets/models/");
00035     const std::shared_ptr<Texture> defaultTexture = m_sceneManager.getTextureDefault();
00036
00037     Logger::logExecutionTime("Creating object quad", [&] {
00038         m_sceneManager.addObject(ObjectFactory::create(
00039             defaultTexture,
00040             modelCache.at("assets/models/quad.obj"),
00041             "quad",
00042             {
00043                 .translation = {0.F, .5F, 0.F},
00044                 .scale = {3.F, 1.F, 3.F},
00045                 .rotation = {0.F, 0.F, 0.F}
00046             }));
00047     });
00048
00049     Logger::logExecutionTime("Creating object smooth vase", [&]{
00050         m_sceneManager.addObject(ObjectFactory::create(
00051             defaultTexture,
00052             modelCache.at("assets/models/smooth_vase.obj"),
00053             "smooth vase",
00054             {
00055                 .translation = {.5F, .5F, 0.F},
00056                 .scale = {3.F, 1.5F, 3.F},
00057                 .rotation = {0.F, 0.F, 0.F}
00058             }));
00059     });
00060     Logger::logExecutionTime("Creating object flat vase", [&]{
00061         m_sceneManager.addObject(ObjectFactory::create(
00062             defaultTexture,
00063             modelCache.at("assets/models/flat_vase.obj"),
00064             "flat vase",
00065             {
00066                 .translation = {-0.5F, .5F, 0.F},
00067                 .scale = {3.F, 1.5F, 3.F},
00068                 .rotation = {0.F, 0.F, 0.F}
00069             }));
00070     });
00071     for (std::size_t i = 0; i < lightColors.size(); i++) {
00072         Logger::logExecutionTime("Creating light n" + std::to_string(i), [&] {
00073             const glm::mat4 rotateLight = rotate(
00074                 glm::mat4(1.F),
00075                 static_cast<float>(i) * glm::two_pi<float>() / 6.0F, // 6 = num of lights
00076                 {0.F, -1.F, 0.F});
00077             m_sceneManager.addLight(LightFactory::create({
00078                 .translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F, 1.F)),
00079                 .scale = {0.1F, 0.0F, 0.0F},
00080                 .rotation = {0.F, 0.F, 0.F},
00081                 lightColors.at(i)
00082             }));
00083         });
00084     }
00085 }
00086
00087 void ven::Engine::mainLoop()
00088 {
00089     Clock clock;
00090     Camera camera{};
00091     EventManager eventManager{};
00092     GlobalUbo ubo{};
00093     VkCommandBuffer_T *commandBuffer = nullptr;
00094     VkDescriptorBufferInfo bufferInfo{};
00095     float frameTime = 0.0F;
00096     unsigned long frameIndex = 0;
00097     std::unique_ptr globalsetLayout(DescriptorSetLayout::Builder(m_device).addBinding(0,
00098         VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_SHADER_STAGE_ALL_GRAPHICS).build());
00099     std::vector<std::unique_ptr<Buffer>> uboBuffers(MAX_FRAMES_IN_FLIGHT);
00100     std::vector<VkDescriptorSet> globalDescriptorSets(MAX_FRAMES_IN_FLIGHT);
00101     ObjectRenderSystem objectRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
00102     globalsetLayout->getDescriptorSetLayout());
00103     PointLightRenderSystem pointLightRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),

```

```

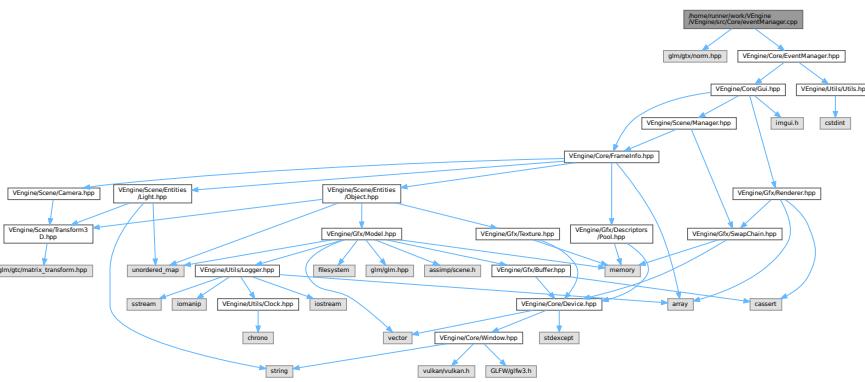
00102     globalsetLayout->getDescriptorsetLayout());
00103     for (auto& uboBuffer : uboBuffers)
00104     {
00105         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
00106         VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00107         uboBuffer->map();
00108     }
00109     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00110         bufferInfo = uboBuffers[i]->descriptorInfo();
00111         DescriptorWriter(*globalsetLayout, *m_globalPool).writeBuffer(0,
00112         &bufferInfo).build(globalDescriptorSets[i]);
00113     }
00114     while (m_state != EXIT)
00115     {
00116         clock.update();
00117         frameTime = clock.getDeltaTime();
00118         eventManager.handleEvents(m_window.getGLFWWindow(), &m_state, camera, m_gui, frameTime);
00119         commandBuffer = m_renderer.beginFrame();
00120
00121         camera.setViewXYZ(camera.transform.translation, camera.transform.rotation);
00122         camera.setPerspectiveProjection(m_renderer.getAspectRatio());
00123
00124         if (commandBuffer != nullptr) {
00125             frameIndex = m_renderer.getFrameIndex();
00126             m_framePools[frameIndex]->resetPool();
00127             FrameInfo frameInfo{
00128                 .frameIndex=frameIndex,
00129                 .frameTime=frameTime,
00130                 .commandBuffer=commandBuffer,
00131                 .camera=camera,
00132                 .globalDescriptorSet=globalDescriptorSets[frameIndex],
00133                 .frameDescriptorPool=&m_framePools[frameIndex],
00134                 .objects=m_sceneManager.getObjects(),
00135                 .lights=m_sceneManager.getLights()
00136             };
00137             ubo.projection=camera.getProjection();
00138             ubo.view=camera.getView();
00139             ubo.inverseView=camera.getInverseView();
00140             m_sceneManager.updateBuffer(ubo, frameIndex, frameTime);
00141             uboBuffers.at(frameIndex)->writeToBuffer(&ubo);
00142             uboBuffers.at(frameIndex)->flush();
00143             m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
00144             objectRenderSystem.render(frameInfo);
00145             pointLightRenderSystem.render(frameInfo);
00146
00147             if (m_gui.getState() != HIDDEN) {
00148                 m_gui.render(
00149                     &m_renderer,
00150                     m_sceneManager,
00151                     camera,
00152                     m_device.getPhysicalDevice(),
00153                     ubo,
00154                     { .deltaTimeMS=clock.getDeltaTimeMS(), .fps=clock.getFPS() }
00155                 );
00156
00157                 m_renderer.endSwapChainRenderPass(commandBuffer);
00158                 m_renderer.endFrame();
00159                 commandBuffer = nullptr;
00160             }
00161             if (m_sceneManager.getDestroyState()) {
00162                 vkDeviceWaitIdle(m_device.device());
00163                 m_sceneManager.destroyEntity(m_gui.getObjectsToRemove(), m_gui.getLightsToRemove());
00164             }
00165         }
00166         vkDeviceWaitIdle(m_device.device());
00167     }
00168
00169 void ven::Engine::cleanup()
00170 {
00171     Gui::cleanup();
00172 }
```

8.78 /home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp

File Reference

```
#include <glm/gtx/norm.hpp>
#include "VEngine/Core/EventManager.hpp"
```

Include dependency graph for eventManager.cpp:



Macros

- `#define GLM_ENABLE_EXPERIMENTAL`

8.78.1 Macro Definition Documentation

8.78.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 1 of file [eventManager.cpp](#).

8.79 eventManager.cpp

[Go to the documentation of this file.](#)

```
00001 #define GLM_ENABLE_EXPERIMENTAL
00002 #include <glm/gtx/norm.hpp>
00003
00004 #include "VEngine/Core/EventManager.hpp"
00005
00006 bool ven::EventManager::isKeyJustPressed(GLFWwindow* window, const long unsigned int key,
00007     std::array<bool, GLFW_KEY_LAST>& keyStates)
00008 {
00009     const bool isPressed = glfwGetKey(window, static_cast<int>(key)) == GLFW_PRESS;
00010     const bool wasPressed = keyStates.at(key);
00011
00012     keyStates.at(key) = isPressed;
00013
00014     return isPressed && !wasPressed;
00015 }
00016 template<typename Iterator>
00017 void ven::EventManager::processKeyActions(GLFWwindow* window, Iterator begin, Iterator end)
00018 {
00019     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};
```

```

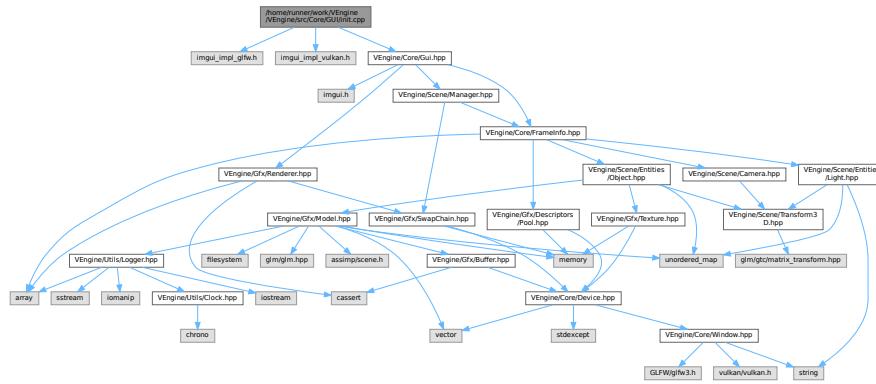
00030     static constexpr glm::vec3 upDir{0.F, -1.F, 0.F};
00031     const float yaw = camera.transform.rotation.y;
00032     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
00033     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00034     const std::array<KeyAction, 10> 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 (const float lengthRotate = length2(rotate); lengthRotate > EPSILON) {
00050         camera.transform.rotation += camera.getLookSpeed() * dt * rotate / std::sqrt(lengthRotate);
00051     }
00052     if (const float lengthMove = length2(moveDir); lengthMove > EPSILON) {
00053         camera.transform.translation += camera.getMoveSpeed() * dt * moveDir / std::sqrt(lengthMove);
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,
00061     Gui& gui, const float dt) const
00062 {
00063     glfwPollEvents();
00064     if (glfwWindowShouldClose(window) == GLFW_TRUE) {
00065         updateEngineState(engineState, EXIT);
00066     }
00067     if (isKeyJustPressed(window, DEFAULT_KEY_MAPPINGS.toggleGui, m_keyState)) {
00068         if (gui.getState() != HIDDEN) {
00069             gui.setState(HIDDEN);
00070         } else {
00071             if (*engineState == EDITOR) {
00072                 gui.setState(SHOW_EDITOR);
00073             } else {
00074                 gui.setState(SHOW_PLAYER);
00075             }
00076         }
00077     }
00078     moveCamera(window, camera, dt);
00079 }

```

8.80 /home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include "VEngine/Core/Gui.hpp"
```

Include dependency graph for init.cpp:



8.81 init.cpp

[Go to the documentation of this file.](#)

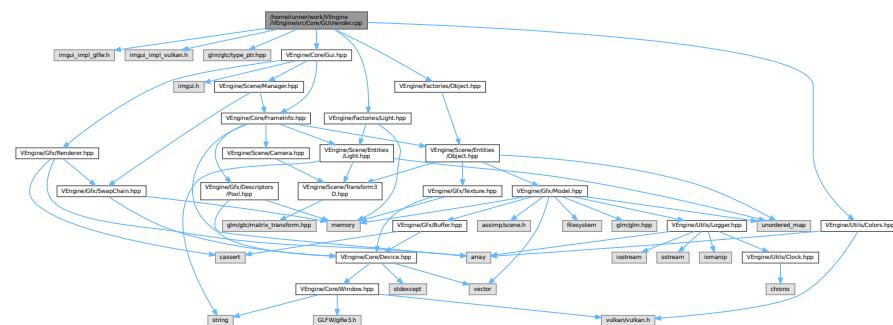
```

00001 #include <imgui_impl_glfw.h>
00002 #include <imgui_impl_vulkan.h>
00003
00004 #include "VEngine/Core/Gui.hpp"
00005
00006 void ven::Gui::init(GLFWwindow* window, const VkInstance instance, const Device* device, const
    VkRenderPass renderPass)
00007 {
00008     VkDescriptorPool pool = nullptr;
00009     ImGui_ImplVulkan_InitInfo init_info{};
00010     ImGui::CreateContext();
00011     m_io = &ImGui::GetIO();
00012     m_io->IniFilename = nullptr;
00013
00014     constexpr std::array<VkDescriptorPoolSize, 11> pool_sizes = {{
00015         { .type=VK_DESCRIPTOR_TYPE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00016         { .type=VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
00017         { .type=VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00018         { .type=VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
00019         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00020         { .type=VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00021         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00022         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
00023         { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00024         { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
00025         { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
00026     }};
00027     const VkDescriptorPoolCreateInfo pool_info = {
00028         VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
00029         nullptr,
00030         VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
00031         DESCRIPTOR_COUNT,
00032         std::size(pool_sizes),
00033         pool_sizes.data()
00034     };
00035
00036     if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
00037         throw std::runtime_error("Failed to create ImGui descriptor pool");
00038     }
00039
00040     init_info.Instance = instance;
00041     init_info.PhysicalDevice = device->getPhysicalDevice();
00042     init_info.Device = device->device();
00043     init_info.Queue = device->graphicsQueue();
00044     init_info.DescriptorPool = pool;
00045     init_info.MinImageCount = 3;
00046     init_info.ImageCount = 3;
00047     init_info.MSAASamples = VK_SAMPLE_COUNT_1_BIT;
00048
00049     ImGui_ImplGlfw_InitForVulkan(window, true);
00050     ImGui_ImplVulkan_Init(&init_info, renderPass);
00051     initStyle();
00052 }
```

```
00053
00054 void ven::Gui::initStyle()
00055 {
00056     ImGuiStyle& style = ImGui::GetStyle();
00057     style.Alpha = 1.0;
00058     style.WindowRounding = 3;
00059     style.GrabRounding = 1;
00060     style.GrabMinSize = 20;
00061     style.FrameRounding = 3;
00062
00063     style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00064     style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
00065     style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
00066     style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
00067     style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
00068     style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
00069     style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
00070     style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
00071     style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
00072     style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
00073     style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
00074     style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
00075     style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
00076     style.Colors[ImGuiCol_ScrollbarGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.44F);
00077     style.Colors[ImGuiCol_ScrollbarGrabHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00078     style.Colors[ImGuiCol_ScrollbarGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00079     style.Colors[ImGuiCol_CheckMark] = ImVec4(0.00F, 1.00F, 1.00F, 0.68F);
00080     style.Colors[ImGuiCol_SliderGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.36F);
00081     style.Colors[ImGuiCol_SliderGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.76F);
00082     style.Colors[ImGuiCol_Button] = ImVec4(0.00F, 0.65F, 0.65F, 0.46F);
00083     style.Colors[ImGuiCol_ButtonHovered] = ImVec4(0.01F, 1.00F, 1.00F, 0.43F);
00084     style.Colors[ImGuiCol_ButtonActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.62F);
00085     style.Colors[ImGuiCol_Header] = ImVec4(0.00F, 1.00F, 1.00F, 0.33F);
00086     style.Colors[ImGuiCol_HeaderHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.42F);
00087     style.Colors[ImGuiCol_HeaderActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00088     style.Colors[ImGuiCol_ResizeGrip] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
00089     style.Colors[ImGuiCol_ResizeGripHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
00090     style.Colors[ImGuiCol_ResizeGripActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00091     style.Colors[ImGuiCol_PlotLines] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00092     style.Colors[ImGuiCol_PlotLinesHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00093     style.Colors[ImGuiCol_PlotHistogram] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00094     style.Colors[ImGuiCol_PlotHistogramHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
00095     style.Colors[ImGuiCol_TextSelectedBg] = ImVec4(0.00F, 1.00F, 1.00F, 0.22F);
00096 }
```

8.82 /home/runner/work/VEngine/VEngine/src/Core/GUI/render.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/Core/Gui.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Factories/Object.hpp"
#include "VEngine/Factories/Light.hpp"
Include dependency graph for render.cpp:
```



8.83 render.cpp

[Go to the documentation of this file.](#)

```

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",
00083         &item_current,
00084         [] (void*, const int idx, const char** out_text) -> bool {
00085             if (idx < 0 || idx >=
00086                 static_cast<int>(std::size(Colors::COLOR_PRESETS_4))) { return false; }
00087             *out_text = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
00088                 long>(idx)).first;
00089             return true;
00090         },
00091         nullptr,
00092         std::size(Colors::COLOR_PRESETS_4))) {
00093             ubo.ambientLightColor = Colors::COLOR_PRESETS_4.at(static_cast<unsigned
00094                 long>(item_current)).second;
00095         }
00096     ImGui::TableNextColumn();
00097     if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a =
00098         DEFAULT_AMBIENT_LIGHT_INTENSITY; }
00099     ImGui::EndTable();
00100 }
00101
00102     static bool fullscreen = false;
00103     if (ImGui::Checkbox("Fullscreen", &fullscreen)) {
00104         renderer->getWindow().setFullscreen(fullscreen, renderer->getWindow().getExtent().width,
00105         renderer->getWindow().getExtent().height);
00106     }
00107 }
00108
00109 void ven::Gui::cameraSection(Camera &camera)
00110 {
00111     if (ImGui::CollapsingHeader("Camera")) {
00112         float fov = camera.getFov();
00113         float tnear = camera.getNear();
00114         float tfar = camera.getFar();
00115         if (ImGui::BeginTable("CameraTable", 2)) {
00116             ImGui::TableNextColumn();
00117             ImGui::DragFloat3("Position", glm::value_ptr(camera.transform.translation), 0.1F);
00118             ImGui::TableNextColumn();
00119             if (ImGui::Button("Reset##position")) { camera.transform.translation = DEFAULT_POSITION; }
00120
00121             ImGui::TableNextColumn();
00122             ImGui::DragFloat3("Rotation", glm::value_ptr(camera.transform.rotation), 0.1F);
00123             ImGui::TableNextColumn();
00124             if (ImGui::Button("Reset##rotation")) { camera.transform.rotation = DEFAULT_ROTATION; }
00125
00126             ImGui::TableNextColumn();
00127             if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) {
00128                 camera.setFov(fov);
00129             if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }
00130
00131             ImGui::TableNextColumn();
00132             if (ImGui::SliderFloat("Near", &tnear, 0.001F, 10.0F)) { camera.setNear(tnear); }
00133             ImGui::TableNextColumn();
00134             if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }
00135
00136             ImGui::TableNextColumn();
00137             if (ImGui::SliderFloat("Far", &tfar, 1.F, 1000.0F)) { camera.setFar(tfar); }
00138             ImGui::TableNextColumn();
00139             if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }
00140
00141             ImGui::TableNextColumn();
00142             float moveSpeed = camera.getMoveSpeed();
00143             if (ImGui::SliderFloat("Move speed", &moveSpeed, 0.1F, 10.0F)) {
00144                 camera.setMoveSpeed(moveSpeed);
00145             if (ImGui::Button("Reset##moveSpeed")) { camera.setMoveSpeed(DEFAULT_MOVE_SPEED); }
00146
00147             ImGui::TableNextColumn();
00148             float lookSpeed = camera.getLookSpeed();
00149             if (ImGui::SliderFloat("Look speed", &lookSpeed, 0.1F, 10.0F)) {
00150                 camera.setLookSpeed(lookSpeed);
00151             if (ImGui::Button("Reset##lookSpeed")) { camera.setLookSpeed(DEFAULT_LOOK_SPEED); }
00152
00153             ImGui::EndTable();
00154     }

```

```

00155     }
00156 }
00157
00158 void ven::Gui::objectsSection(SceneManager& sceneManager)
00159 {
00160     if (ImGui::CollapsingHeader("Objects")) {
00161         bool open = false;
00162         for (Object::Map::iterator objects = sceneManager.getObjects(); auto& [id, object] : objects) {
00163             ImGui::PushStyleColor(ImGuiCol_Text, { Colors::GRAY_4.r, Colors::GRAY_4.g,
00164                                         Colors::GRAY_4.b, 1.0F });
00165             open = ImGui::TreeNode(std::string(object.getName()) + " [" +
00166                                   std::to_string(object.getId()) + "]").c_str());
00167             ImGui::PopStyleColor(1);
00168             if (open) {
00169                 if (ImGui::Button(("Delete##" + object.getName()).c_str())) {
00170                     m_objectsToRemove.push_back(id);
00171                     sceneManager.setDestroyState(true);
00172                 }
00173                 ImGui::SameLine();
00174                 if (ImGui::Button(("Duplicate##" + object.getName()).c_str())) {
00175                     ObjectFactory::duplicate(object);
00176                 }
00177                 ImGui::Text("Address: %p", static_cast<void*>(&object));
00178                 ImGui::DragFloat3(("Position##" + object.getName()).c_str(),
00179                                   glm::value_ptr(object.transform.translation), 0.1F);
00180                 ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(),
00181                                   glm::value_ptr(object.transform.rotation), 0.1F);
00182                 ImGui::DragFloat3(("Scale##" + object.getName()).c_str(),
00183                                   glm::value_ptr(object.transform.scale), 0.1F);
00184             }
00185             ImGui::TreePop();
00186         }
00187     }
00188 }
00189
00190 void ven::Gui::lightsSection(SceneManager& sceneManager)
00191 {
00192     if (ImGui::CollapsingHeader("Lights")) {
00193         bool open = false;
00194         float tempIntensity = m_intensity;
00195         float tempShininess = m_shininess;
00196         Light::Map& lights = sceneManager.getLights();
00197
00198         if (ImGui::BeginTable("LightTable", 2)) {
00199             ImGui::TableNextColumn();
00200             if (ImGui::SliderFloat("Global Intensity", &tempIntensity, 0.0F, 5.F)) {
00201                 m_intensity = tempIntensity;
00202                 for (auto&[fst, snd] : lights) {
00203                     snd.color.a = m_intensity;
00204                 }
00205             }
00206             ImGui::TableNextColumn();
00207             if (ImGui::Button("Reset")) {
00208                 m_intensity = DEFAULT_LIGHT_INTENSITY;
00209                 tempIntensity = m_intensity;
00210                 for (auto&[fst, snd] : lights) {
00211                     snd.color.a = m_intensity;
00212                 }
00213             }
00214             ImGui::TableNextColumn();
00215             if (ImGui::SliderFloat("Global Shininess", &tempShininess, 0.0F, 512.F)) {
00216                 m_shininess = tempShininess;
00217                 for (auto&[fst, snd] : lights) {
00218                     snd.setShininess(m_shininess);
00219                 }
00220             }
00221             ImGui::TableNextColumn();
00222             if (ImGui::Button("Reset")) {
00223                 m_shininess = DEFAULT_SHININESS;
00224                 tempShininess = m_shininess;
00225                 for (auto&[fst, snd] : lights) {
00226                     snd.setShininess(m_shininess);
00227                 }
00228             }
00229             ImGui::EndTable();
00230         }
00231         for (auto& [id, light] : lights) {
00232             ImGui::PushStyleColor(ImGuiCol_Text, {light.color.r, light.color.g, light.color.b, 1.0F});
00233             open = ImGui::TreeNode(std::string(light.getName()) + " [" + std::to_string(light.getId())
00234                                   + "]").c_str());
00235             ImGui::PopStyleColor(1);
00236             if (open) {

```

```

00236         if (ImGui::Button(("Delete##" + light.getName()).c_str())) {
00237             m_lightsToRemove.push_back(id);
00238             sceneManager.setDestroyState(true);
00239         }
00240         ImGui::SameLine();
00241         if (ImGui::Button(("Duplicate##" + light.getName()).c_str())) {
00242             LightFactory::duplicate(light);
00243         }
00244         ImGui::Text("Address: %p", static_cast<void*>(&light));
00245         ImGui::DragFloat3(("Position##" + std::to_string(light.getId())).c_str(),
00246                           glm::value_ptr(light.transform.translation), 0.1F);
00246         ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId())).c_str(),
00247                           glm::value_ptr(light.transform.rotation), 0.1F);
00247         ImGui::DragFloat3(("Scale##" + std::to_string(light.getId())).c_str(),
00248                           glm::value_ptr(light.transform.scale), 0.1F);
00248         if (ImGui::BeginTable("ColorTable", 2)) {
00249             ImGui::TableNextColumn();
00250             ImGui::ColorEdit4(("Color##" + std::to_string(light.getId())).c_str(),
00251                               glm::value_ptr(light.color));
00252             ImGui::TableNextColumn();
00253             static int item_current = 0;
00254             if (ImGui::Combo("Color Presets",
00255                             &item_current,
00256                             [] (void*, const int idx, const char** out_text) -> bool {
00257                                 if (idx < 0 || idx >=
00258                                     static_cast<int>(std::size(Colors::COLOR_PRESETS_3))) { return false; }
00259                                 *out_text = Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00260                                     long>(idx)).first;
00261                                 return true;
00262                             },
00263                             nullptr,
00264                             std::size(Colors::COLOR_PRESETS_3))) {
00265                             light.color = {Colors::COLOR_PRESETS_3.at(static_cast<unsigned
00266                                     long>(item_current)).second, light.color.a};
00267                         }
00268                         ImGui::EndTable();
00269                         ImGui::SliderFloat(("Intensity##" + std::to_string(light.getId())).c_str(),
00270                               &light.color.a, 0.0F, 5.F);
00271                         ImGui::SameLine();
00272                         if (ImGui::Button(("Reset##" + std::to_string(light.getId())).c_str())) {
00273                             light.color.a = DEFAULT_LIGHT_INTENSITY;
00274                             float shininess = light.getShininess();
00275                             if (ImGui::SliderFloat("Shininess", &shininess, 0.0F, 512.F)) {
00276                                 light.setShininess(shininess);
00277                             }
00278                             ImGui::SameLine();
00279                             if (ImGui::Button("Reset#shininess")) { light.setShininess(DEFAULT_SHININESS); }
00280                         }
00281                         ImGui::TreePop();
00282                     }
00283                 }
00284                 if (ImGui::CollapsingHeader("Input")) {
00285                     ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io.MousePos.x, io.MousePos.y) :
00286                     ImGui::Text("Mouse pos: <INVALID>");
00287                     ImGui::Text("Mouse delta: (%g, %g)", io.MouseDelta.x, io.MouseDelta.y);
00288                     ImGui::Text("Mouse down:");
00289                     for (int i = 0; i < static_cast<int>(std::size(io.MouseDown)); i++) {
00290                         if (ImGui::IsMouseDown(i)) {
00291                             ImGui::SameLine();
00292                             ImGui::Text("b%d (%.02f secs)", i, io.MouseDownDuration[i]);
00293                         }
00294                         ImGui::Text("Mouse wheel: %.1f", io.MouseWheel);
00295                         ImGui::Text("Keys down:");
00296                         for (auto key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key =
00297                             static_cast<ImGuiKey>(key + 1)) {
00298                             if (func::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
00299                             ImGui::SameLine();
00300                             ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\"%s\" : \"%s\" %d",
00301                             ImGui::GetKeyName(key), key);
00302                         }
00303                     }
00304                 void ven::Gui::devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties)
00305                 {
00306                     if (ImGui::CollapsingHeader("Device Properties")) {
00307                         if (ImGui::BeginTable("DevicePropertiesTable", 2)) {
00308                             ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
00309                             ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d",
00310

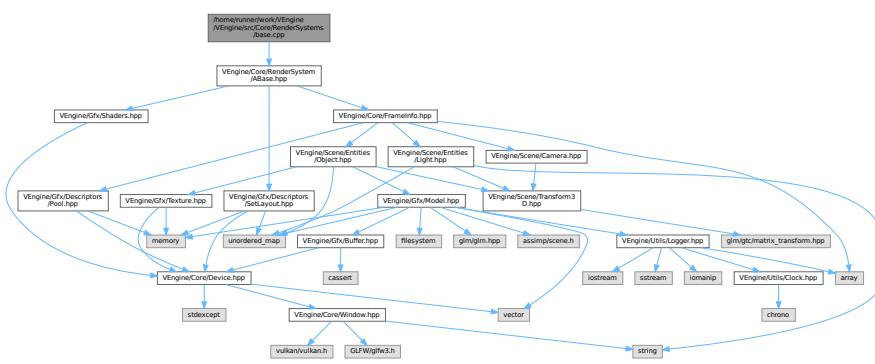
```

```

VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion),
VK_VERSION_PATCH(deviceProperties.apiVersion);
00311     ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d",
VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion),
VK_VERSION_PATCH(deviceProperties.driverVersion));
00312         ImGui::TableNextColumn(); ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
00313         ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
00314         ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
00315         ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d",
deviceProperties.limits.discreteQueuePriorities);
00316         ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d",
deviceProperties.limits.maxPushConstantSize);
00317         ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d",
deviceProperties.limits.maxMemoryAllocationCount);
00318         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d",
deviceProperties.limits.maxImageDimension1D);
00319         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d",
deviceProperties.limits.maxImageDimension2D);
00320         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d",
deviceProperties.limits.maxImageDimension3D);
00321         ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d",
deviceProperties.limits.maxImageDimensionCube);
00322         ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d",
deviceProperties.limits.maxImageArrayLayers);
00323         ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d",
deviceProperties.limits.maxTexelBufferElements);
00324         ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d",
deviceProperties.limits.maxUniformBufferRange);
00325         ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d",
deviceProperties.limits.maxStorageBufferRange);
00326         ImGui::EndTable();
00327     }
00328 }
00329 }
```

8.84 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp File Reference

#include "VEngine/Core/RenderSystem/ABase.hpp"
Include dependency graph for base.cpp:



8.85 base.cpp

[Go to the documentation of this file.](#)

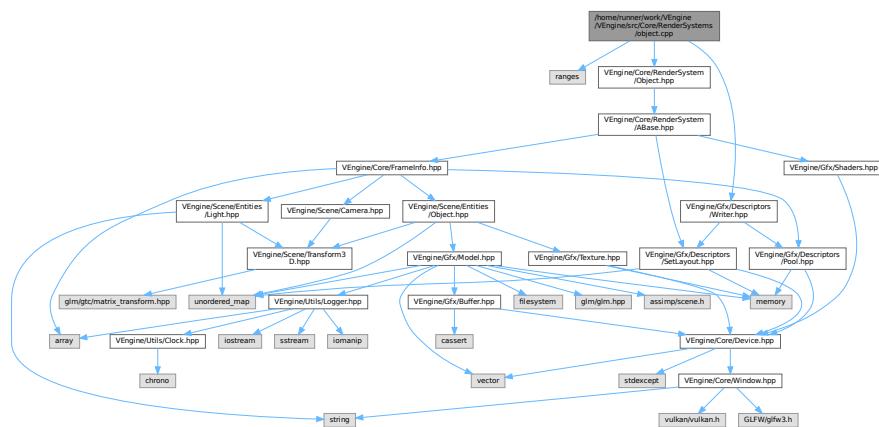
```

00001 #include "VEngine/Core/RenderSystem/ABase.hpp"
00002
00003 void ven::ARenderSystemBase::createPipelineLayout(const VkDescriptorSetLayout globalsetLayout, const
uint32_t pushConstantSize)
00004 {
00005     VkPushConstantRange pushConstantRange{};
00006     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00007     pushConstantRange.offset = 0;
```

```
00008 pushConstantRange.size = pushConstantSize;
00009
00010 renderSystemLayout =
00011 DescriptorSetLayout::Builder(m_device)
00012     .addBinding(
00013         0,
00014         VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
00015         VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT)
00016     .addBinding(1, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_SHADER_STAGE_FRAGMENT_BIT)
00017     .build();
00018
00019 const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{
00020     globalsetLayout,
00021     renderSystemLayout->getDescriptorSetLayout() };
00022
00023 VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00024 pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00025 pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00026 pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00027 pipelineLayoutInfo.pushConstantRangeCount = 1;
00028 pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00029 if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) != VK_SUCCESS)
00030 {
00031     throw std::runtime_error("Failed to create pipeline layout");
00032 }
00033 }
00034
00035 void ven::ARenderSystemBase::createPipeline(const VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, const bool isLight)
00036 {
00037     assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
00038     PipelineConfigInfo pipelineConfig{};
00039     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00040     if (isLight) {
00041         pipelineConfig.attributeDescriptions.clear();
00042         pipelineConfig.bindingDescriptions.clear();
00043     }
00044     pipelineConfig.renderPass = renderPass;
00045     pipelineConfig.pipelineLayout = m_pipelineLayout;
00046     m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
00047 }
```

8.86 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp File Reference

```
#include <ranges>
#include "VEngine/Gfx/Descriptors/Writer.hpp"
#include "VEngine/Core/RenderSystem/Object.hpp"
Include dependency graph for object.cpp:
```



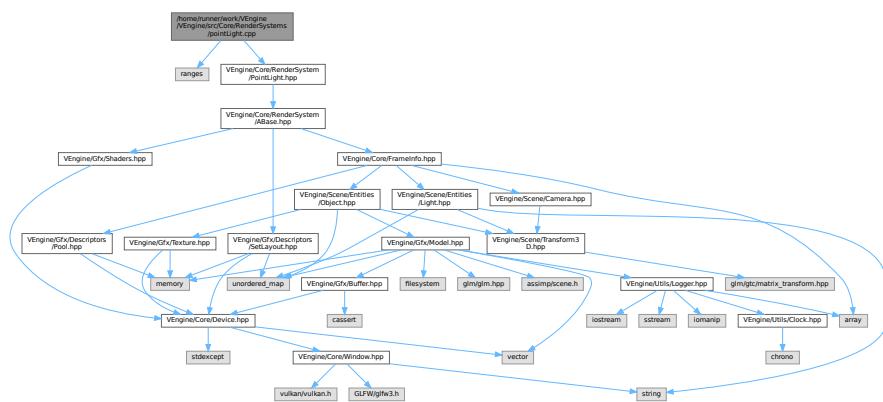
8.87 object.cpp

[Go to the documentation of this file.](#)

```
00001 #include <ranges>
00002
00003 #include "VEngine/Gfx/Descriptors/Writer.hpp"
00004 #include "VEngine/Core/RenderSystem/Object.hpp"
00005
00006 void ven::ObjectRenderSystem::render(const FrameInfo &frameInfo) const
00007 {
00008     getShaders()->bind(frameInfo.commandBuffer);
00009
00010     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
00011         getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00012
00013     for (Object& object : frameInfo.objects | std::views::values) {
00014         if (object.getModel() == nullptr) { continue; }
00015         auto bufferInfo = object.getBufferInfo(static_cast<int>(frameInfo.frameIndex));
00016         auto imageInfo = object.getDiffuseMap()->getImageInfo();
00017         VkDescriptorSet objectDescriptorSet = nullptr;
00018         DescriptorWriter(*renderSystemLayout, frameInfo.frameDescriptorPool)
00019             .writeBuffer(0, &bufferInfo)
00020             .writeImage(1, &imageInfo)
00021             .build(objectDescriptorSet);
00022
00023         vkCmdBindDescriptorSets(
00024             frameInfo.commandBuffer,
00025             VK_PIPELINE_BIND_POINT_GRAPHICS,
00026             getPipelineLayout(),
00027             1, // starting set (0 is the globalDescriptorSet, 1 is the set specific to this system)
00028             1, // set count
00029             &objectDescriptorSet,
00030             0,
00031             nullptr);
00032
00033     const ObjectPushConstantData push{
00034         .modelMatrix = object.transform.transformMatrix(),
00035         .normalMatrix = object.transform.normalMatrix()
00036     };
00037     vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
00038         VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(ObjectPushConstantData), &push);
00039     object.getModel()->bind(frameInfo.commandBuffer);
00040     object.getModel()->draw(frameInfo.commandBuffer);
00041 }
```

8.88 /home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.cpp File Reference

```
#include <ranges>
#include "VEngine/Core/RenderSystem/PointLight.hpp"
Include dependency graph for pointLight.cpp:
```



8.89 pointLight.cpp

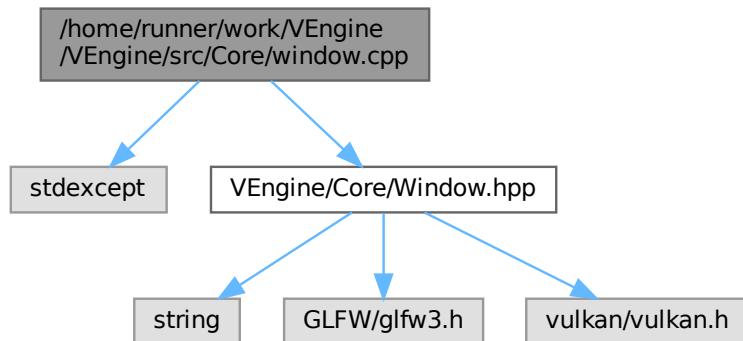
[Go to the documentation of this file.](#)

```
00001 #include <ranges>
00002
00003 #include "VEngine/Core/RenderSystem/PointLight.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,
00009     getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00010
00011     for (const Light &light : frameInfo.lights | std::views::values) {
00012         const LightPushConstantData push{
00013             .position = glm::vec4(light.transform.translation, 1.F),
00014             .color = light.color,
00015             .radius = light.transform.scale.x
00016         };
00017         vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
00018             VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
00019     }
00019 }
```

8.90 /home/runner/work/VEngine/VEngine/src/Core/window.cpp File Reference

```
#include <stdexcept>
#include "VEngine/Core/Window.hpp"
```

Include dependency graph for window.cpp:



8.91 window.cpp

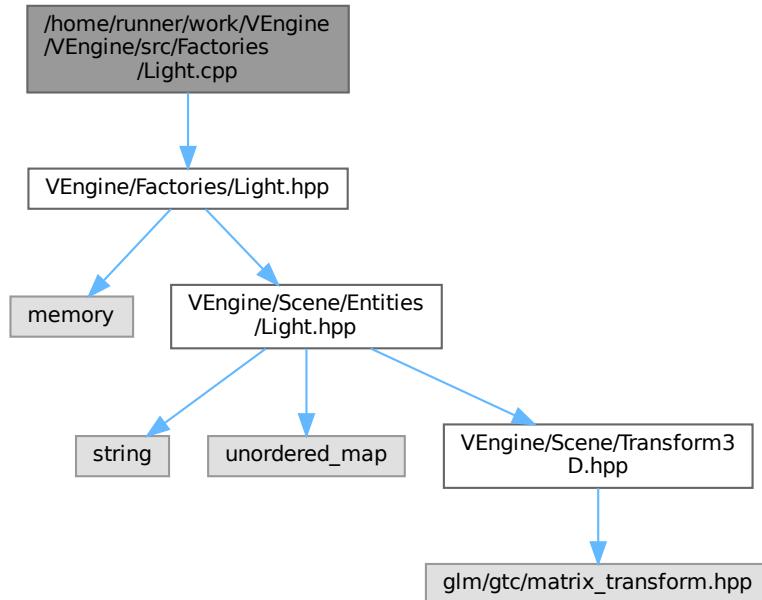
[Go to the documentation of this file.](#)

```
00001 #include <stdexcept>
00002
00003 #include "VEngine/Core/Window.hpp"
00004
00005 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string
00006 &title)
00006 {
00007     if (glfwInit() == GLFW_FALSE) {
```

```
00008     throw std::runtime_error("Failed to initialize GLFW");
00009 }
0010
0011     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
0012     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
0013
0014     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
0015         title.c_str(), nullptr, nullptr);
0016     if (window == nullptr) {
0017         glfwTerminate();
0018         throw std::runtime_error("Failed to create window");
0019     }
0020     glfwSetWindowUserPointer(window, this);
0021     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
0022     return window;
0023 }
0024 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
0025 {
0026     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
0027         throw std::runtime_error("Failed to create window surface");
0028     }
0029 }
0030
0031 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
0032 {
0033     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
0034     app->m_framebufferResized = true;
0035     app->m_width = static_cast<uint32_t>(width);
0036     app->m_height = static_cast<uint32_t>(height);
0037 }
0038
0039 void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
0040 {
0041     GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
0042     const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);
0043
0044     /*
0045     if (fullscreen) {
0046         glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
0047             mode->refreshRate);
0048     } else {
0049         // To restore a window that was originally windowed to its original size and position,
0050         // save these before making it full screen and then pass them in as above
0051         glfwSetWindowMonitor(m_window, nullptr, 0, 0, static_cast<int>(width),
0052             static_cast<int>(height), mode->refreshRate);
0053     }
0054     m_width = width;
0055     m_height = height;
0056 */
0057 }
```

8.92 /home/runner/work/VEngine/VEngine/src/Factories/Light.cpp File Reference

```
#include "VEngine/Factories/Light.hpp"
Include dependency graph for Light.cpp:
```



8.93 Light.cpp

[Go to the documentation of this file.](#)

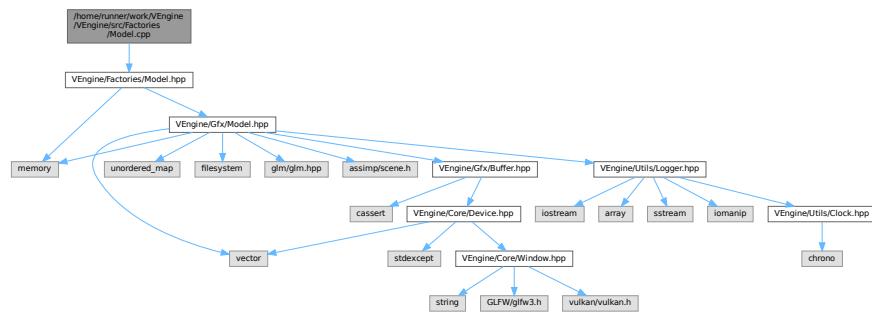
```

0001 #include "VEngine/Factories/Light.hpp"
0002
0003 unsigned int ven::LightFactory::m_currentLightId = 0;
0004
0005 std::unique_ptr<ven::Light> ven::LightFactory::create(const Transform3D &transform, const glm::vec4
0006   color)
0006 {
0007     assert(m_currentLightId < MAX_LIGHTS && "Max light count exceeded!");
0008     auto light = std::make_unique<Light>(++m_currentLightId);
0009     light->color = color;
0010     light->transform = transform;
0011     return light;
0012 }
```

8.94 /home/runner/work/VEngine/VEngine/src/Factories/Model.cpp File Reference

```
#include "VEngine/Factories/Model.hpp"
```

Include dependency graph for Model.cpp:



8.95 Model.cpp

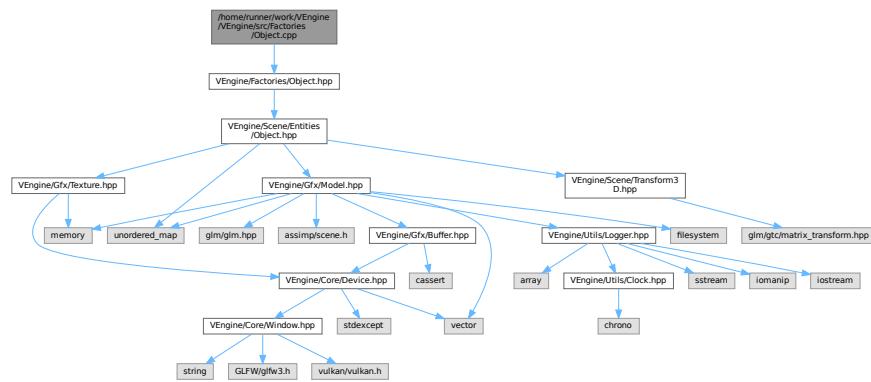
[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Factories/Model.hpp"
00002
00003 std::unordered_map<std::string, std::shared_ptr<ven::Model>> ven::ModelFactory::loadAll(Device& device,
00004   const std::string& folderPath)
00005 {
00006     std::unordered_map<std::string, std::shared_ptr<Model>> modelCache;
00007
00008     for (const auto &entry : std::filesystem::directory_iterator(folderPath)) {
00009         if (entry.is_regular_file()) {
00010             Logger::logExecutionTime("Creating model " + entry.path().string(), [&]() {
00011                 const std::string &filepath = entry.path().string();
00012                 modelCache[filepath] = create(device, filepath);
00013             });
00014         } else {
00015             Logger::logWarning("Skipping non-regular file " + entry.path().string());
00016         }
00017     }
00018     return modelCache;
00019 }
00020 std::unique_ptr<ven::Model> ven::ModelFactory::create(Device& device, const std::string& filepath)
00021 {
00022     Model::Builder builder{};
00023     builder.loadModel(filepath);
00024     return std::make_unique<Model>(device, builder);
00025 }
```

8.96 /home/runner/work/VEngine/VEngine/src/Factories/Object.cpp File Reference

```
#include "VEngine/Factories/Object.hpp"
Include dependency graph for Object.cpp:
```



8.97 Object.cpp

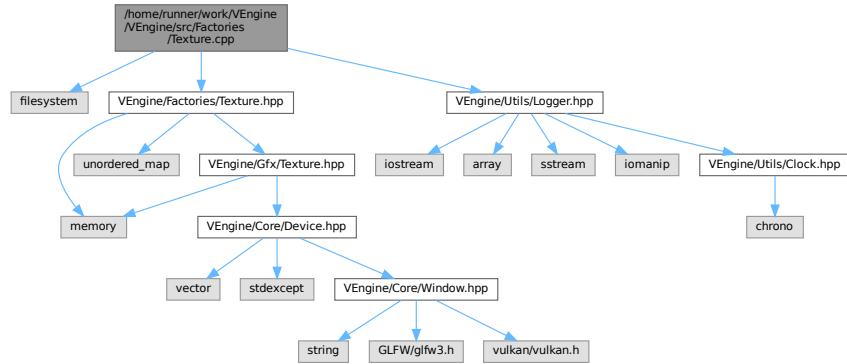
[Go to the documentation of this file.](#)

```
00001 #include "VEngine/Factories/Object.hpp"
00002
00003 unsigned int ven::ObjectFactory::m_currentObjId = 0;
00004
00005 std::unique_ptr<ven::Object> ven::ObjectFactory::create(const std::shared_ptr<Texture>& texture, const
00006   std::shared_ptr<Model>& model, const std::string &name, const Transform3D &transform)
00007 {
00008     assert(m_currentObjId < MAX_OBJECTS && "Max object count exceeded!");
00009     auto object = std::make_unique<Object>(++m_currentObjId);
00010     object->setDiffuseMap(texture);
00011     object->setModel(model);
00012     object->setName(name);
00013     object->transform = transform;
00014     return object;
00015 }
```

8.98 /home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp File Reference

```
#include <filesystem>
#include "VEngine/Factories/Texture.hpp"
#include "VEngine/Utils/Logger.hpp"
```

Include dependency graph for Texture.cpp:



8.99 Texture.cpp

[Go to the documentation of this file.](#)

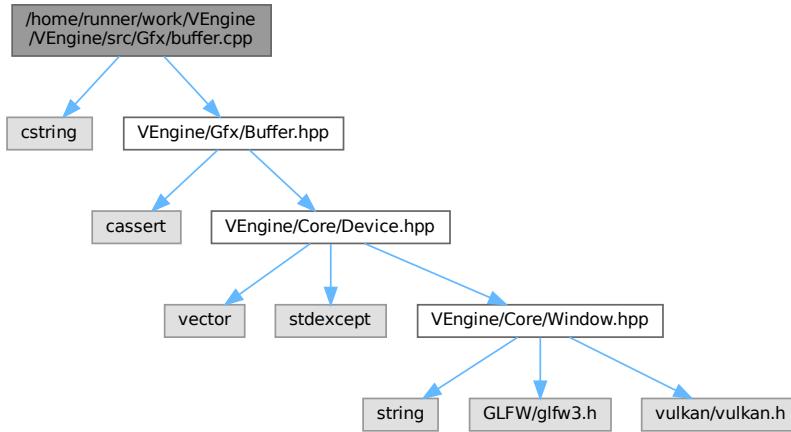
```

00001 #include <filesystem>
00002
00003 #include "VEngine/Factories/Texture.hpp"
00004 #include "VEngine/Utils/Logger.hpp"
00005
00006 std::unordered_map<std::string, std::shared_ptr<ven::Texture>> ven::TextureFactory::loadAll(Device&
device, const std::string& folderPath)
00007 {
00008     std::unordered_map<std::string, std::shared_ptr<Texture>> modelCache;
00009
0010     for (const auto &entry : std::filesystem::directory_iterator(folderPath)) {
0011         if (entry.is_regular_file()) {
0012             Logger::logExecutionTime("Creating texture " + entry.path().string(), [&]() {
0013                 const std::string &filepath = entry.path().string();
0014                 modelCache[filepath] = create(device, filepath);
0015             });
0016         } else {
0017             Logger::logWarning("Skipping non-regular file " + entry.path().string());
0018         }
0019     }
0020     return modelCache;
0021 }
```

8.100 /home/runner/work/VEngine/VEngine/src/Gfx/Buffer.cpp File Reference

```
#include <cstring>
#include "VEngine/Gfx/Buffer.hpp"
```

Include dependency graph for buffer.cpp:



8.101 buffer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cstring>
00002
00003 #include "VEngine/Gfx/Buffer.hpp"
00004
00005 ven::Buffer::Buffer(Device &device, const VkDeviceSize instanceSize, const uint32_t instanceCount,
00006   const VkBufferUsageFlags usageFlags, const VkMemoryPropertyFlags memoryPropertyFlags, const
00007   VkDeviceSize minOffsetAlignment) : m_device{device}, m_instanceSize{instanceSize},
00008   m_instanceCount{instanceCount}, m_alignmentSize{getAlignment(instanceSize, minOffsetAlignment)},
00009   m_usageFlags{usageFlags}, m_memoryPropertyFlags{memoryPropertyFlags}
00010 {
00011     m_bufferSize = m_alignmentSize * m_instanceCount;
00012     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00013 }
00014
00015 ven::Buffer::~Buffer()
00016 {
00017     unmmap();
00018     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00019     vkFreeMemory(m_device.device(), m_memory, nullptr);
00020 }
00021
00022 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00023 {
00024     assert(m_buffer && m_memory && "Called map on buffer before create");
00025     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00026 }
00027
00028 void ven::Buffer::unmap()
00029 {
00030     if (m_mapped != nullptr) {
00031         vkUnmapMemory(m_device.device(), m_memory);
00032         m_mapped = nullptr;
00033     }
00034 }
00035
00036 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
00037 {
00038     const
00039     assert(m_mapped && "Cannot copy to unmapped buffer");
00040     if (size == VK_WHOLE_SIZE) {
00041         memcpy(m_mapped, data, m_bufferSize);
00042     } else {
00043         auto *memOffset = static_cast<char *>(m_mapped);
00044         memOffset += offset;
00045         memcpy(memOffset, data, size);
00046     }
  
```

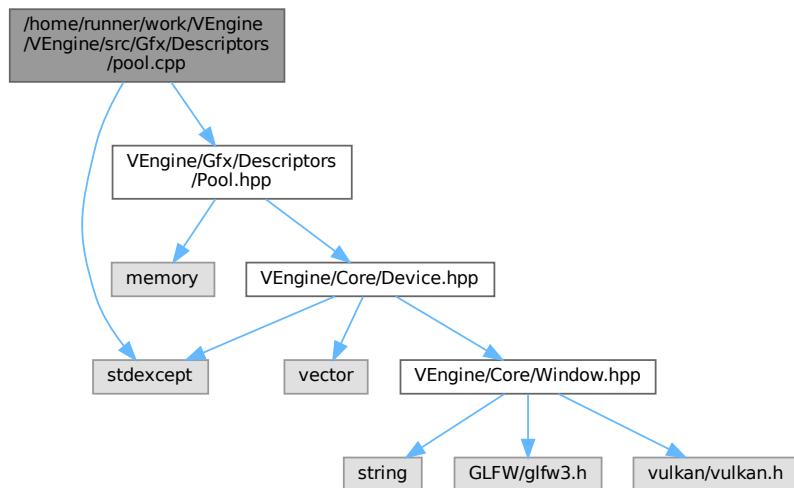
```

00043 }
00044
00045 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00046 {
00047     VkMappedMemoryRange mappedRange = {};
00048     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00049     mappedRange.memory = m_memory;
00050     mappedRange.offset = offset;
00051     mappedRange.size = size;
00052     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00053 }
00054
00055 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00056 {
00057     VkMappedMemoryRange mappedRange = {};
00058     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00059     mappedRange.memory = m_memory;
00060     mappedRange.offset = offset;
00061     mappedRange.size = size;
00062     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00063 }

```

8.102 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/pool.cpp File Reference

```
#include <stdexcept>
#include "VEngine/Gfx/Descriptors/Pool.hpp"
Include dependency graph for pool.cpp:
```



8.103 pool.cpp

[Go to the documentation of this file.](#)

```

00001 #include <stdexcept>
00002
00003 #include "VEngine/Gfx/Descriptors/Pool.hpp"
00004
00005 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device(device)
00006 {

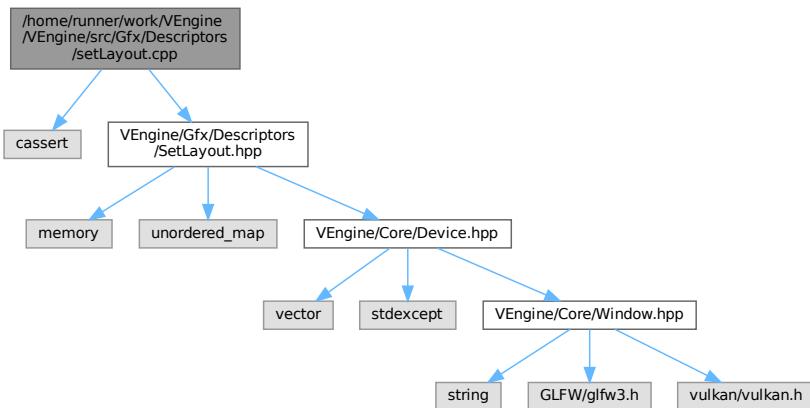
```

```

00007     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00008     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00009     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00010     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00011     descriptorPoolInfo.maxSets = maxSets;
00012     descriptorPoolInfo.flags = poolFlags;
00013
00014     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) != VK_SUCCESS) {
00015         throw std::runtime_error("failed to create descriptor pool!");
00016     }
00017 }
00018 }
00019
00020 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorsetLayout,
00021     VkDescriptorSet &descriptor) const
00022 {
00023     VkDescriptorSetAllocateInfo allocInfo{};
00024     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00025     allocInfo.descriptorPool = m_descriptorPool;
00026     allocInfo.pSetLayouts = &descriptorsetLayout;
00027     allocInfo.descriptorSetCount = 1;
00028
00029     // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00030     // a new pool whenever an old pool fills up. But this is beyond our current scope
00031     return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00032 }
```

8.104 /home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/setLayout.cpp File Reference

```
#include <cassert>
#include "VEngine/Gfx/Descriptors/SetLayout.hpp"
Include dependency graph for setLayout.cpp:
```



8.105 setLayout.cpp

[Go to the documentation of this file.](#)

```

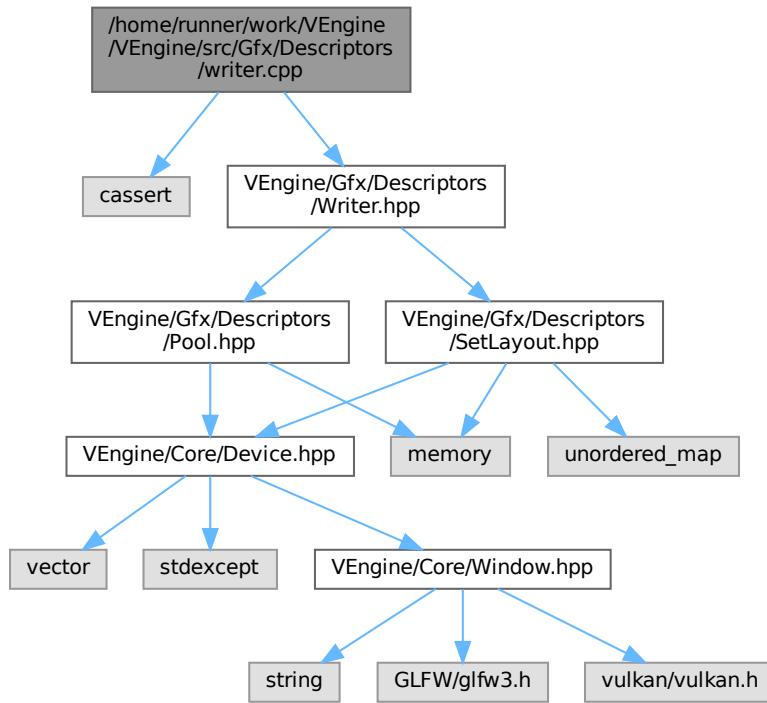
00001 #include <cassert>
00002
00003 #include "VEngine/Gfx/Descriptors/SetLayout.hpp"
00004
00005 ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t
00006     binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t
00007     count)
00008 {
```

```
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,
00018                                                 VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
00019 {
00020     std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
00021     setLayoutBindings.reserve(bindings.size());
00022     for (auto [fst, snd] : bindings) {
00023         setLayoutBindings.push_back(snd);
00024     }
00025
00026     VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
00027     descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
00028     descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
00029     descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();
00030
00031     if (vkCreateDescriptorSetLayout(
00032         m_device.device(),
00033         &descriptorSetLayoutInfo,
00034         nullptr,
00035         &m_descriptorSetLayout) != VK_SUCCESS) {
00036         throw std::runtime_error("failed to create descriptor set layout!");
00037 }
```

8.106 /home/runner/work/VEngine/VEngine/src/Gfx/[Descriptor](#)s/writer.cpp File Reference

```
#include <cassert>
#include "VEngine/Gfx/Descriptors/Writer.hpp"
```

Include dependency graph for writer.cpp:



8.107 writer.cpp

[Go to the documentation of this file.](#)

```

00001 #include <cassert>
00002
00003 #include "VEngine/Gfx/Descriptors/Writer.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
00031     expects multiple");
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,
00054     pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
00055         dstSet = set;
00056         vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
00057         m_writes.data(), 0, nullptr);
00057 }

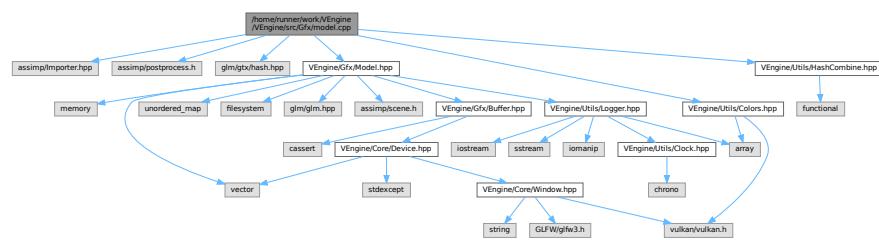
```

8.108 /home/runner/work/VEngine/src/Gfx/model.cpp File Reference

```

#include <assimp/Importer.hpp>
#include <assimp/postprocess.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Utils/Colors.hpp"
#include "VEngine/Utils/HashCombine.hpp"
Include dependency graph for model.cpp:

```



Classes

- struct `std::hash< ven::Model::Vertex >`

Macros

- #define `GLM_ENABLE_EXPERIMENTAL`

8.108.1 Macro Definition Documentation

8.108.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 4 of file [model.cpp](#).

8.109 model.cpp

[Go to the documentation of this file.](#)

```
00001 #include <assimp/Importer.hpp>
00002 #include <assimp/postprocess.h>
00003
00004 #define GLM_ENABLE_EXPERIMENTAL
00005 #include <glm/gtx/hash.hpp>
00006
00007 #include "VEngine/Gfx/Model.hpp"
00008 #include "VEngine/Utils/Colors.hpp"
00009 #include "VEngine/Utils/HashCombine.hpp"
00010
00011 template<>
00012 struct std::hash<ven::Model::Vertex> {
00013     size_t operator()(ven::Model::Vertex const &vertex) const noexcept {
00014         size_t seed = 0;
00015         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00016         return seed;
00017     }
00018 };
00019
00020 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
m_indexCount(0)
00021 {
00022     createVertexBuffer(builder.vertices);
00023     createIndexBuffer(builder.indices);
00024 }
00025
00026 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00027 {
00028     m_vertexCount = static_cast<uint32_t>(vertices.size());
00029     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00030     constexpr unsigned long vertexSize = sizeof(vertices[0]);
00031     const VkDeviceSize bufferSize = vertexSize * m_vertexCount;
00032
00033     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};
00034
00035     stagingBuffer.map();
00036     stagingBuffer.writeToBuffer(vertices.data());
00037
00038     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);
00039
00040     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00041 }
00042
00043 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00044 {
00045     m_indexCount = static_cast<uint32_t>(indices.size());
00046     m_hasIndexBuffer = m_indexCount > 0;
00047
00048     if (!m_hasIndexBuffer) {
00049         return;
00050     }
00051
00052     constexpr uint32_t indexSize = sizeof(indices[0]);
00053
00054     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};
00055
00056     stagingBuffer.map();
00057     stagingBuffer.writeToBuffer(indices.data());
00058
00059     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);
```

```

00060
00061     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), sizeof(indices[0]) *
00062     m_indexCount);
00063
00064 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00065 {
00066     if (m_hasIndexBuffer) {
00067         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
00068     } else {
00069         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00070     }
00071 }
00072
00073 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00074 {
00075     const std::array buffers{m_vertexBuffer->getBuffer()};
00076     constexpr std::array<VkDeviceSize, 1> offsets{};
00077     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets.data());
00078
00079     if (m_hasIndexBuffer) {
00080         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00081     }
00082 }
00083
00084 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00085 {
00086     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00087     bindingDescriptions[0].binding = 0;
00088     bindingDescriptions[0].stride = sizeof(Vertex);
00089     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00090     return bindingDescriptions;
00091 }
00092
00093 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00094 {
00095     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00096
00097     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00098     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00099     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00100    attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00101
00102     return attributeDescriptions;
00103 }
00104
00105 void ven::Model::Builder::loadModel(const std::string &filename)
00106 {
00107     Assimp::Importer importer;
00108
00109     const aiScene* scene = importer.ReadFile(filename, aiProcess_Triangulate | aiProcess_FlipUVs |
00110     aiProcess_CalcTangentSpace | aiProcess_GenNormals);
00111
00112     if ((scene == nullptr) || ((scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE) != 0U) || (scene->mRootNode
00113     == nullptr)) {
00114         throw std::runtime_error("Failed to load model with Assimp: " +
00115         std::string(importer.GetErrorString()));
00116     }
00117
00118     vertices.clear();
00119     indices.clear();
00120
00121 void ven::Model::Builder::processNode(const aiNode* node, const aiScene* scene) {
00122     for (unsigned int i = 0; i < node->mNumMeshes; i++) {
00123         const aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
00124         processMesh(mesh, scene);
00125     }
00126
00127     for (unsigned int i = 0; i < node->mNumChildren; i++) {
00128         processNode(node->mChildren[i], scene);
00129     }
00130 }
00131
00132 void ven::Model::Builder::processMesh(const aiMesh* mesh, const aiScene* scene) {
00133     std::unordered_map<Vertex, uint32_t> uniqueVertices;
00134
00135     for (unsigned int i = 0; i < mesh->mNumVertices; i++) {
00136         Vertex vertex{};
00137
00138         vertex.position = glm::vec3(
00139             mesh->mVertices[i].x,
00140             mesh->mVertices[i].y,
00141             mesh->mVertices[i].z
00142         );

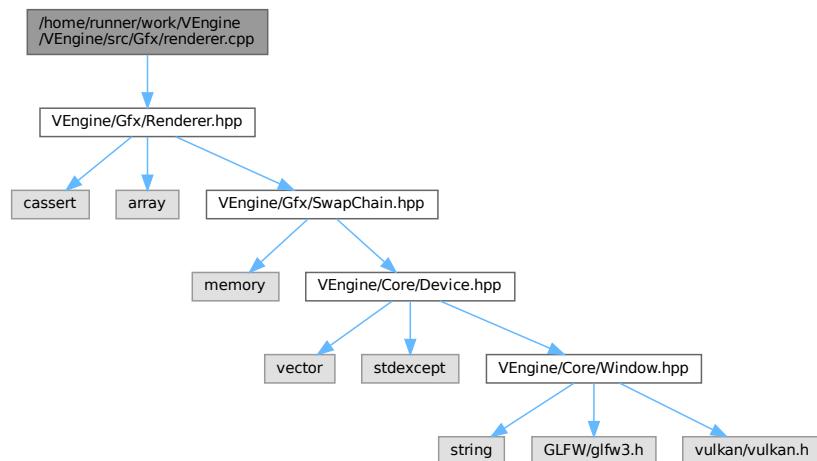
```

```

00143
00144     if (mesh->HasNormals()) {
00145         vertex.normal = glm::vec3(
00146             mesh->mNormals[i].x,
00147             mesh->mNormals[i].y,
00148             mesh->mNormals[i].z
00149         );
00150     }
00151
00152     if (mesh->mTextureCoords[0] != nullptr) {
00153         vertex.uv = glm::vec2(
00154             mesh->mTextureCoords[0][i].x,
00155             mesh->mTextureCoords[0][i].y
00156         );
00157     } else {
00158         vertex.uv = glm::vec2(0.0F, 0.0F);
00159     }
00160
00161     if (!uniqueVertices.contains(vertex)) {
00162         uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00163         vertices.push_back(vertex);
00164     }
00165
00166     indices.push_back(uniqueVertices[vertex]);
00167 }
00168 }
```

8.110 /home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp File Reference

#include "VEngine/Gfx/Renderer.hpp"
Include dependency graph for renderer.cpp:



8.111 renderer.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Gfx/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();
0010     allocInfo.commandBufferCount = static_cast<uint32_t>(m_commandBuffers.size());
0011
0012     if (vkAllocateCommandBuffers(m_device.device(), &allocInfo, m_commandBuffers.data()) != VK_SUCCESS) {
0013         throw std::runtime_error("Failed to allocate command buffers");
0014     }
0015 }
0016
0017 void ven::Renderer::freeCommandBuffers()
0018 {
0019     vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(),
0020     static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
0021     m_commandBuffers.clear();
0022
0023 void ven::Renderer::recreateSwapChain()
0024 {
0025     VkExtent2D extent = m_window.getExtent();
0026     while (extent.width == 0 || extent.height == 0) {
0027         extent = m_window.getExtent();
0028         glfwWaitEvents();
0029     }
0030     vkDeviceWaitIdle(m_device.device());
0031     if (m_swapChain == nullptr) {
0032         m_swapChain = std::make_unique<SwapChain>(m_device, extent);
0033     } else {
0034         std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
0035         m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
0036         if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
0037             throw std::runtime_error("Swap chain image/depth format changed");
0038         }
0039     }
0040     // well be back
0041 }
0042
0043 VkCommandBuffer ven::Renderer::beginFrame()
0044 {
0045     assert(!m_isFrameStarted && "Can't start new frame while previous one is still in progress");
0046
0047     const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
0048     if (result == VK_ERROR_OUT_OF_DATE_KHR) {
0049         recreateSwapChain();
0050         return nullptr;
0051     }
0052
0053     if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
0054         throw std::runtime_error("Failed to acquire swap chain image");
0055     }
0056
0057     m_isFrameStarted = true;
0058
0059     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
0060     VkCommandBufferBeginInfo beginInfo{};
0061     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
0062
0063     if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
0064         throw std::runtime_error("Failed to begin recording command buffer");
0065     }
0066     return commandBuffer;
0067 }
0068
0069 void ven::Renderer::endFrame()
0070 {
0071     assert(m_isFrameStarted && "Can't end frame that hasn't been started");
0072
0073     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
0074     if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
0075         throw std::runtime_error("Failed to record command buffer");
0076     }
0077     if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer,
0078     &m_currentImageIndex); result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
0079     m_window.wasWindowResized()) {
0080         m_window.resetWindowResizedFlag();
0081         recreateSwapChain();
0082     }
0083     else if (result != VK_SUCCESS) {
0084         throw std::runtime_error("Failed to submit command buffer");
0085     }
0086     m_isFrameStarted = false;
0087     m_currentFrameIndex = (m_currentFrameIndex + 1) % MAX_FRAMES_IN_FLIGHT;
0088 }
0089
0090 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 }

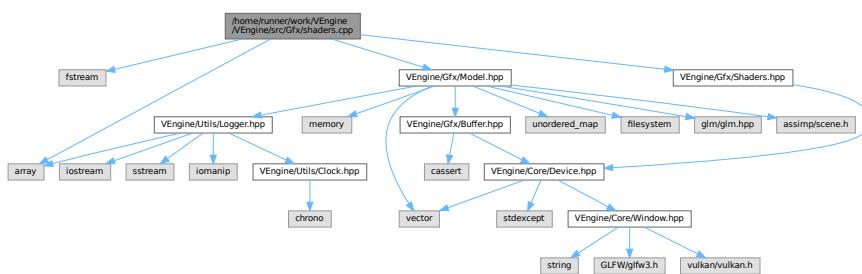
```

8.112 /home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp File Reference

```

#include <fstream>
#include <array>
#include "VEngine/Gfx/Model.hpp"
#include "VEngine/Gfx/Shaders.hpp"
Include dependency graph for shaders.cpp:

```



8.113 shaders.cpp

[Go to the documentation of this file.](#)

```

00001 #include <fstream>
00002 #include <array>

```

```
00003
00004 #include "VEngine/Gfx/Model.hpp"
00005 #include "VEngine/Gfx/Shaders.hpp"
00006
00007 ven::Shaders::~Shaders()
00008 {
00009     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00010    vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00011    vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
00012 }
00013
00014 std::vector<char> ven::Shaders::readFile(const std::string &filename) {
00015     std::ifstream file(filename, std::ios::binary | std::ios::ate);
00016     if (!file.is_open()) {
00017         throw std::runtime_error("failed to open file!");
00018     }
00019
00020     const long int fileSize = file.tellg();
00021     std::vector<char> buffer(static_cast<long unsigned int>(fileSize));
00022     file.seekg(0);
00023     file.read(buffer.data(), fileSize);
00024     return buffer;
00025 }
00026
00027 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
fragFilepath, const PipelineConfigInfo& configInfo)
00028 {
00029     const std::vector<char> vertCode = readFile(vertFilepath);
00030     const std::vector<char> fragCode = readFile(fragFilepath);
00031
00032     createShaderModule(vertCode, &m_vertShaderModule);
00033     createShaderModule(fragCode, &m_fragShaderModule);
00034
00035     std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
00036     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00037     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00038     shaderStages[0].module = m_vertShaderModule;
00039     shaderStages[0].pName = "main";
00040     shaderStages[0].flags = 0;
00041     shaderStages[0].pNext = nullptr;
00042     shaderStages[0].pSpecializationInfo = nullptr;
00043
00044     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00045     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00046     shaderStages[1].module = m_fragShaderModule;
00047     shaderStages[1].pName = "main";
00048     shaderStages[1].flags = 0;
00049     shaderStages[1].pNext = nullptr;
00050     shaderStages[1].pSpecializationInfo = nullptr;
00051
00052     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00053     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00054     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00055     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00056     vertexInputInfo.vertexAttributeDescriptionCount =
00057         static_cast<uint32_t>(attributeDescriptions.size());
00058     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00059     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00060     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00061
00062     VkPipelineViewportStateCreateInfo viewportInfo{};
00063     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00064     viewportInfo.viewportCount = 1;
00065     viewportInfo.pViewports = nullptr;
00066     viewportInfo.scissorCount = 1;
00067     viewportInfo.pScissors = nullptr;
00068
00069
00070     VkGraphicsPipelineCreateInfo pipelineInfo{};
00071     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00072     pipelineInfo.stageCount = 2;
00073     pipelineInfo.pStages = shaderStages.data();
00074     pipelineInfo.pVertexInputState = &vertexInputInfo;
00075     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00076     pipelineInfo.pViewportState = &viewportInfo;
00077     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00078     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00079
00080     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00081     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00082     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00083
00084     pipelineInfo.layout = configInfo.pipelineLayout;
00085     pipelineInfo.renderPass = configInfo.renderPass;
00086     pipelineInfo.subpass = configInfo.subpass;
00087 }
```

```

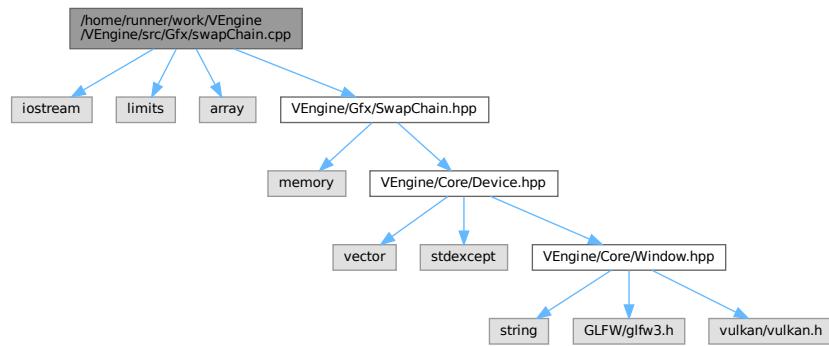
00088     pipelineInfo.basePipelineIndex = -1;
00089     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00090
00091     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
00092         &m_graphicsPipeline) != VK_SUCCESS) {
00093         throw std::runtime_error("failed to create graphics pipeline");
00094     }
00095
00096 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
00097 {
00098     VkShaderModuleCreateInfo createInfo{};
00099     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00100     createInfo.codeSize = code.size();
00101     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00102
00103     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00104         throw std::runtime_error("failed to create shader module");
00105     }
00106 }
00107
00108 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00109 {
00110     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00111     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00112     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00113
00114     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00115     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00116     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00117     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00118     configInfo.rasterizationInfo.lineWidth = 1.0F;
00119     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE; // to enable later
00120     configInfo.rasterizationInfo.backFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00121     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00122     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00123     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00124     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00125
00126     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00127     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00128     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00129     configInfo.multisampleInfo.minSampleShading = 1.0F;
00130     configInfo.multisampleInfo.pSampleMask = nullptr;
00131     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00132     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00133
00134     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
00135         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 =
00169         static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00170     configInfo.dynamicStateInfo.flags = 0;
00171     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();

```

```
00170     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00171 }
```

8.114 /home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp File Reference

```
#include <iostream>
#include <limits>
#include <array>
#include "VEngine/Gfx/SwapChain.hpp"
Include dependency graph for swapChain.cpp:
```



8.115 swapChain.cpp

[Go to the documentation of this file.](#)

```
00001 #include <iostream>
00002 #include <limits>
00003 #include <array>
00004
00005 #include "VEngine/Gfx/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         vkDestroyImage(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,
00052     std::numeric_limits<uint64_t>::max());
00053
00054     return vkAcquireNextImageKHR(m_device.device(), m_swapChain, std::numeric_limits<uint64_t>::max(),
00055     m_imageAvailableSemaphores[m_currentFrame], VK_NULL_HANDLE, imageIndex);
00056 }
00057
00058 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
00059 *imageIndex)
00060 {
00061     if (m_imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00062         vkWaitForFences(m_device.device(), 1, &m_imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00063     }
00064     m_imagesInFlight[*imageIndex] = m_inFlightFences[m_currentFrame];
00065
00066     VkSubmitInfo submitInfo = {};
00067     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00068
00069     const std::array<VkSemaphore, 1> waitSemaphores = {m_imageAvailableSemaphores[m_currentFrame]};
00070     constexpr std::array<VkPipelineStageFlags, 1> waitStages =
00071     {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00072     submitInfo.waitSemaphoreCount = 1;
00073     submitInfo.pWaitSemaphores = waitSemaphores.data();
00074     submitInfo.pWaitDstStageMask = waitStages.data();
00075
00076     submitInfo.commandBufferCount = 1;
00077     submitInfo.pCommandBuffers = buffers;
00078
00079     const std::array<VkSemaphore, 1> signalSemaphores = {m_renderFinishedSemaphores[m_currentFrame]};
00080     submitInfo.signalSemaphoreCount = 1;
00081     submitInfo.pSignalSemaphores = signalSemaphores.data();
00082
00083     vkResetFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame]);
00084     if (vkQueueSubmit(m_device.graphicsQueue(), 1, &submitInfo, m_inFlightFences[m_currentFrame]) !=
00085     VK_SUCCESS) {
00086         throw std::runtime_error("failed to submit draw command buffer!");
00087     }
00088
00089     VkPresentInfoKHR presentInfo = {};
00090     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00091
00092     presentInfo.waitSemaphoreCount = 1;
00093     presentInfo.pWaitSemaphores = signalSemaphores.data();
00094
00095     presentInfo.pImageIndices = imageIndex;
00096
00097     const VkResult result = vkQueuePresentKHR(m_device.presentQueue(), &presentInfo);
00098
00099     m_currentFrame = (m_currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00100
00101     return result;
00102 }
00103
00104 void ven::SwapChain::createSwapChain()
00105 {
00106     const auto [capabilities, formats, presentModes] = m_device.getSwapChainSupport();
00107
00108     const auto [format, colorSpace] = chooseSwapSurfaceFormat(formats);
00109     const VkPresentModeKHR presentMode = chooseSwapPresentMode(presentModes);
00110     const VkExtent2D extent = chooseSwapExtent(capabilities);
00111
00112     uint32_t imageCount = capabilities.minImageCount + 1;
00113     if (capabilities.maxImageCount > 0 && imageCount > capabilities.maxImageCount) {
00114         imageCount = capabilities.maxImageCount;
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] =
00128         m_device.findPhysicalQueueFamilies();
00129     const std::array<uint32_t, 2> queueFamilyIndices = {graphicsFamily, presentFamily};
00130
00131     if (graphicsFamily != presentFamily) {
00132         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00133         createInfo.queueFamilyIndexCount = 2;
00134         createInfo.pQueueFamilyIndices = queueFamilyIndices.data();
00135     } else {
00136         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00137         createInfo.queueFamilyIndexCount = 0; // Optional
00138         createInfo.pQueueFamilyIndices = nullptr; // Optional
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 :
00147         m_oldSwapChain->m_swapChain;
00148
00149     if (vkCreateSwapchainKHR(m_device.device(), &createInfo, nullptr, &m_swapChain) != VK_SUCCESS) {
00150         throw std::runtime_error("failed to create swap chain!");
00151     }
00152
00153     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, nullptr);
00154     m_swapChainImages.resize(imageCount);
00155     vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, m_swapChainImages.data());
00156
00157     m_swapChainImageFormat = format;
00158     m_swapChainExtent = extent;
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]) !=
00176             VK_SUCCESS) {
00177             throw std::runtime_error("failed to create texture image view!");
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 |
00221         VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00221     dependency.dstSubpass = 0;
00222     dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
00223         VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00223     dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
00224         VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00224
00225     const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
00226     VkRenderPassCreateInfo renderPassInfo = {};
00227     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
00228     renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00229     renderPassInfo.pAttachments = attachments.data();
00230     renderPassInfo.subpassCount = 1;
00231     renderPassInfo.pSubpasses = &subpass;
00232     renderPassInfo.dependencyCount = 1;
00233     renderPassInfo.pDependencies = &dependency;
00234
00235     if (vkCreateRenderPass(m_device.device(), &renderPassInfo, nullptr, &m_renderPass) != VK_SUCCESS)
00236     {
00237         throw std::runtime_error("failed to create render pass!");
00238     }
00239
00240 void ven::SwapChain::createFrameBuffers()
00241 {
00242     m_swapChainFrameBuffers.resize(imageCount());
00243     for (size_t i = 0; i < imageCount(); i++) {
00244         std::array<VkImageView, 2> attachments = {m_swapChainImageViews[i], m_depthImageViews[i]};
00245
00246         const auto [width, height] = getSwapChainExtent();
00247         VkFramebufferCreateInfo framebufferInfo = {};
00248         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00249         framebufferInfo.renderPass = m_renderPass;
00250         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00251         framebufferInfo.pAttachments = attachments.data();
00252         framebufferInfo.width = width;
00253         framebufferInfo.height = height;
00254         framebufferInfo.layers = 1;
00255
00256         if (vkCreateFramebuffer(m_device.device(), &framebufferInfo, nullptr,
00257             &m_swapChainFrameBuffers[i]) != VK_SUCCESS) {
00258             throw std::runtime_error("failed to create framebuffer!");
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],
00290     m_depthImageMemory[i]);
00291
00292     VkImageViewCreateInfo viewInfo{};
00293     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00294     viewInfo.image = m_depthImages[i];
00295     viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00296     viewInfo.format = depthFormat;
00297     viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00298     viewInfo.subresourceRange.baseMipLevel = 0;
00299     viewInfo.subresourceRange.levelCount = 1;
00300     viewInfo.subresourceRange.baseArrayLayer = 0;
00301     viewInfo.subresourceRange.layerCount = 1;
00302
00303     if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_depthImageViews[i]) != VK_SUCCESS) {
00304         throw std::runtime_error("failed to create texture image view!");
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,
00324             &m_imageAvailableSemaphores[i]) != VK_SUCCESS ||
00325             vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr,
00326             &m_renderFinishedSemaphores[i]) != VK_SUCCESS ||
00327             vkCreateFence(m_device.device(), &fenceInfo, nullptr, &m_inFlightFences[i]) != VK_SUCCESS)
00328     {
00329         throw std::runtime_error("failed to create synchronization objects for a frame!");
00330     }
00331 }
00332
00333     VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
00334     &availableFormats)
00335 {
00336     for (const auto &availableFormat : availableFormats) {
00337         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
00338             VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00339             return availableFormat;
00340         }
00341     }
00342     return availableFormats[0];
00343 }
00344
00345     VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
00346     &availablePresentModes)
00347 {
00348     for (const auto &availablePresentMode : availablePresentModes) {
00349         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00350             std::cout << "Present mode: Mailbox\n";
00351             return availablePresentMode;
00352         }
00353     }
00354
00355     for (const auto &availablePresentMode : availablePresentModes) {
00356         if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00357             std::cout << "Present mode: Immediate" << '\n';
00358             return availablePresentMode;
00359         }
00360     }
00361
00362     std::cout << "Present mode: V-Sync\n";
00363     return VK_PRESENT_MODE_FIFO_KHR;
00364 }
```

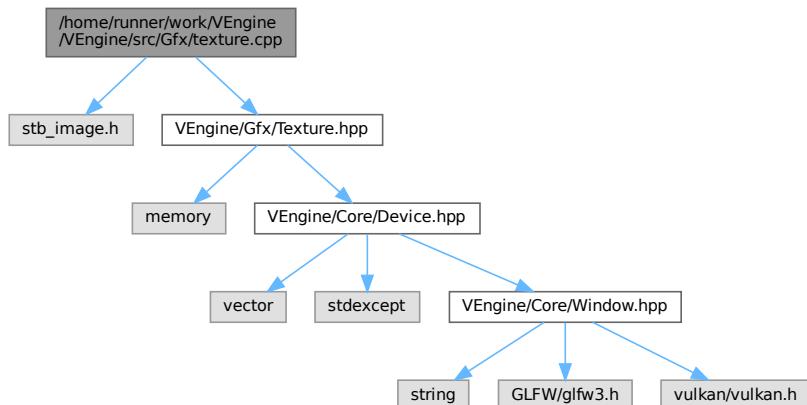
```

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,
00369         std::min(capabilities.maxImageExtent.width, actualExtent.width));
00369     actualExtent.height = std::max(capabilities.minImageExtent.height,
00370         std::min(capabilities.maxImageExtent.height, actualExtent.height));
00370
00371     return actualExtent;
00372 }
00373
00374 VkFormat ven::SwapChain::findDepthFormat() const
00375 {
00376     return m_device.findSupportedFormat(
00377         {VK_FORMAT_D32_SFLOAT, VK_FORMAT_D32_SFLOAT_S8_UINT, VK_FORMAT_D24_UNORM_S8_UINT},
00378         VK_IMAGE_TILING_OPTIMAL,
00379         VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT);
00380 }

```

8.116 /home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp File Reference

```
#include <stb_image.h>
#include "VEngine/Gfx/Texture.hpp"
Include dependency graph for texture.cpp:
```



Macros

- #define STB_IMAGE_IMPLEMENTATION

8.116.1 Macro Definition Documentation

8.116.1.1 STB_IMAGE_IMPLEMENTATION

```
#define STB_IMAGE_IMPLEMENTATION
```

Definition at line 1 of file [texture.cpp](#).

8.117 texture.cpp

[Go to the documentation of this file.](#)

```

00001 #define STB_IMAGE_IMPLEMENTATION
00002 #include <stb_image.h>
00003
00004 #include "VEngine/Gfx/Texture.hpp"
00005
00006 ven::Texture::Texture(Device &device, const std::string &textureFilepath) : m_device{device}
00007 {
00008     createTextureImage(textureFilepath);
00009     createTextureImageView(VK_IMAGE_VIEW_TYPE_2D);
00010     createTextureSampler();
00011     updateDescriptor();
00012 }
00013
00014 ven::Texture::Texture(Device &device, VkFormat format, VkExtent3D extent, VkImageUsageFlags usage,
00015                         VkSampleCountFlagBits sampleCount)
00016 : m_device{device}, m_format(format), m_extent(extent)
00017 {
00018     VkImageAspectFlags aspectMask = 0;
00019     VkImageLayout imageLayout;
00020
00021     if ((usage & VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT) != 0u) {
00022         aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00023         imageLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00024     }
00025     if ((usage & VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT) != 0u) {
00026         aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00027         imageLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00028     }
00029
00030     // Don't like this, should I be using an image array instead of multiple images?
00031     VkImageCreateInfo imageInfo{};
00032     imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00033     imageInfo.imageType = VK_IMAGE_TYPE_2D;
00034     imageInfo.format = format;
00035     imageInfo.extent = extent;
00036     imageInfo.mipLevels = 1;
00037     imageInfo.arrayLayers = 1;
00038     imageInfo.samples = sampleCount;
00039     imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00040     imageInfo.usage = usage;
00041     imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00042     device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_textureImage,
00043                               m_textureImageMemory);
00044
00045     VkImageViewCreateInfo viewInfo{};
00046     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00047     viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00048     viewInfo.format = format;
00049     viewInfo.subresourceRange = {};
00050     viewInfo.subresourceRange.aspectMask = aspectMask;
00051     viewInfo.subresourceRange.baseMipLevel = 0;
00052     viewInfo.subresourceRange.levelCount = 1;
00053     viewInfo.subresourceRange.baseArrayLayer = 0;
00054     viewInfo.subresourceRange.layerCount = 1;
00055     viewInfo.image = m_textureImage;
00056     if (vkCreateImageView(device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00057         throw std::runtime_error("failed to create texture image view!");
00058     }
00059
00060     // Sampler should be separated out
00061     if ((usage & VK_IMAGE_USAGE_SAMPLED_BIT) != 0u) {
00062         // Create sampler to sample from the attachment in the fragment shader
00063         VkSamplerCreateInfo samplerInfo{};
00064         samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00065         samplerInfo.magFilter = VK_FILTER_LINEAR;
00066         samplerInfo.minFilter = VK_FILTER_LINEAR;
00067         samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00068         samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER;
00069         samplerInfo.addressModeV = samplerInfo.addressModeU;
00070         samplerInfo.addressModeW = samplerInfo.addressModeU;
00071         samplerInfo.mipLodBias = 0.0f;
00072         samplerInfo.maxAnisotropy = 1.0f;
00073         samplerInfo.minLod = 0.0f;
00074         samplerInfo.maxLod = 1.0f;
00075         samplerInfo.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
00076         if (vkCreateSampler(device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS)
00077             throw std::runtime_error("failed to create sampler!");
00078     }
00079     VkImageLayout samplerImageLayout = imageLayout == VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL

```

```

00080             ? VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
00081             : VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL;
00082     m_descriptor.sampler = m_textureSampler;
00083     m_descriptor.imageView = m_textureImageView;
00084     m_descriptor.imageLayout = samplerImageLayout;
00085 }
00086 }
00087
00088 ven::Texture::~Texture()
00089 {
00090     vkDestroySampler(m_device.device(), m_textureSampler, nullptr);
00091     vkDestroyImageView(m_device.device(), m_textureImageView, nullptr);
00092     vkDestroyImage(m_device.device(), m_textureImage, nullptr);
00093     vkFreeMemory(m_device.device(), m_textureImageMemory, nullptr);
00094 }
00095
00096 void ven::Texture::updateDescriptor()
00097 {
00098     m_descriptor.sampler = m_textureSampler;
00099     m_descriptor.imageView = m_textureImageView;
00100    m_descriptor.imageLayout = m_textureLayout;
00101 }
00102
00103 void ven::Texture::createTextureImage(const std::string &filepath)
00104 {
00105     int texWidth = 0;
00106     int texHeight = 0;
00107     int texChannels = 0;
00108     void *data = nullptr;
00109     stbi_uc *pixels = nullptr;
00110
00111     stbi_set_flip_vertically_on_load(1);
00112     pixels = stbi_load(filepath.c_str(), &texWidth, &texHeight, &texChannels, STBI_rgb_alpha);
00113     const auto imageSize = static_cast<VkDeviceSize>(texWidth * texHeight * 4);
00114
00115     if (pixels == nullptr) {
00116         throw std::runtime_error("failed to load texture image!");
00117     }
00118
00119 // mMipLevels = static_cast<uint32_t>(std::floor(std::log2(std::max(texWidth, texHeight)))) + 1;
00120 m_mipLevels = 1;
00121
00122 VkBuffer stagingBuffer = nullptr;
00123 VkDeviceMemory stagingBufferMemory = nullptr;
00124
00125 m_device.createBuffer(
00126     imageSize,
00127     VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
00128     VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT,
00129     stagingBuffer,
00130     stagingBufferMemory);
00131
00132 vkMapMemory(m_device.device(), stagingBufferMemory, 0, imageSize, 0, &data);
00133 memcpy(data, pixels, imageSize);
00134 vkUnmapMemory(m_device.device(), stagingBufferMemory);
00135
00136 stbi_image_free(pixels);
00137
00138 m_format = VK_FORMAT_R8G8B8A8_SRGB;
00139 m_extent = {.width=static_cast<uint32_t>(texWidth), .height=static_cast<uint32_t>(texHeight),
00140 .depth=1};
00141
00142 VkImageCreateInfo imageInfo{};
00143 imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00144 imageInfo.imageType = VK_IMAGE_TYPE_2D;
00145 imageInfo.extent = m_extent;
00146 imageInfo.mipLevels = m_mipLevels;
00147 imageInfo.arrayLayers = m_layerCount;
00148 imageInfo.format = m_format;
00149 imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00150 imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00151 imageInfo.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT |
VK_IMAGE_USAGE_SAMPLED_BIT;
00152 imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00153 imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00154
00155 m_device.createImageWithInfo(
00156     imageInfo,
00157     VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT,
00158     m_textureImage,
00159     m_textureImageMemory);
00160 m_device.transitionImageLayout(
00161     m_textureImage,
00162     VK_FORMAT_R8G8B8A8_SRGB,
00163     VK_IMAGE_LAYOUT_UNDEFINED,
00164     VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00165     m_mipLevels,
```

```

00165     m_layerCount);
00166     m_device.copyBufferToImage(
00167         stagingBuffer,
00168         m_textureImage,
00169         static_cast<uint32_t>(texWidth),
00170         static_cast<uint32_t>(texHeight),
00171         m_layerCount);
00172
00173     // comment this out if using mips
00174     m_device.transitionImageLayout(
00175         m_textureImage,
00176         VK_FORMAT_R8G8B8A8_SRGB,
00177         VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
00178         VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL,
00179         m_mipLevels,
00180         m_layerCount);
00181
00182     // If we generate mip maps then the final image will already be READ_ONLY_OPTIMAL
00183     // mDevice.generateMipmaps(mTextureImage, mFormat, texWidth, texHeight, mMipLevels);
00184     m_textureLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
00185
00186     vkDestroyBuffer(m_device.device(), stagingBuffer, nullptr);
00187     vkFreeMemory(m_device.device(), stagingBufferMemory, nullptr);
00188 }
00189
00190 void ven::Texture::createTextureImageView(const VkImageViewType viewType)
00191 {
00192     VkImageViewCreateInfo viewInfo{};
00193     viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00194     viewInfo.image = m_textureImage;
00195     viewInfo.viewType = viewType;
00196     viewInfo.format = VK_FORMAT_R8G8B8A8_SRGB;
00197     viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00198     viewInfo.subresourceRange.baseMipLevel = 0;
00199     viewInfo.subresourceRange.levelCount = m_mipLevels;
00200     viewInfo.subresourceRange.baseArrayLayer = 0;
00201     viewInfo.subresourceRange.layerCount = m_layerCount;
00202
00203     if (vkCreateImageView(m_device.device(), &viewInfo, nullptr, &m_textureImageView) != VK_SUCCESS) {
00204         throw std::runtime_error("failed to create texture image view!");
00205     }
00206 }
00207
00208 void ven::Texture::createTextureSampler()
00209 {
00210     VkSamplerCreateInfo samplerInfo{};
00211     samplerInfo.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
00212     samplerInfo.magFilter = VK_FILTER_LINEAR;
00213     samplerInfo.minFilter = VK_FILTER_LINEAR;
00214
00215     samplerInfo.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00216     samplerInfo.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00217     samplerInfo.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
00218
00219     samplerInfo.anisotropyEnable = VK_TRUE;
00220     samplerInfo.maxAnisotropy = 16.0F;
00221     samplerInfo.borderColor = VK_BORDER_COLOR_INT_OPAQUE_BLACK;
00222     samplerInfo.unnormalizedCoordinates = VK_FALSE;
00223
00224     // these fields useful for percentage close filtering for shadow maps
00225     samplerInfo.compareEnable = VK_FALSE;
00226     samplerInfo.compareOp = VK_COMPARE_OP_ALWAYS;
00227
00228     samplerInfo.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
00229     samplerInfo.mipLodBias = 0.0F;
00230     samplerInfo.minLod = 0.0F;
00231     samplerInfo.maxLod = static_cast<float>(m_mipLevels);
00232
00233     if (vkCreateSampler(m_device.device(), &samplerInfo, nullptr, &m_textureSampler) != VK_SUCCESS) {
00234         throw std::runtime_error("failed to create texture sampler!");
00235     }
00236 }
00237
00238 void ven::Texture::transitionLayout(const VkCommandBuffer commandBuffer, const VkImageLayout
00239     oldLayout, const VkImageLayout newLayout) const
00240 {
00241     VkPipelineStageFlags sourceStage = 0;
00242     VkPipelineStageFlags destinationStage = 0;
00243     VkImageMemoryBarrier barrier{};
00244
00245     barrier.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
00246     barrier.oldLayout = oldLayout;
00247     barrier.newLayout = newLayout;
00248
00249     barrier.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00250     barrier.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
00251 }
```

```

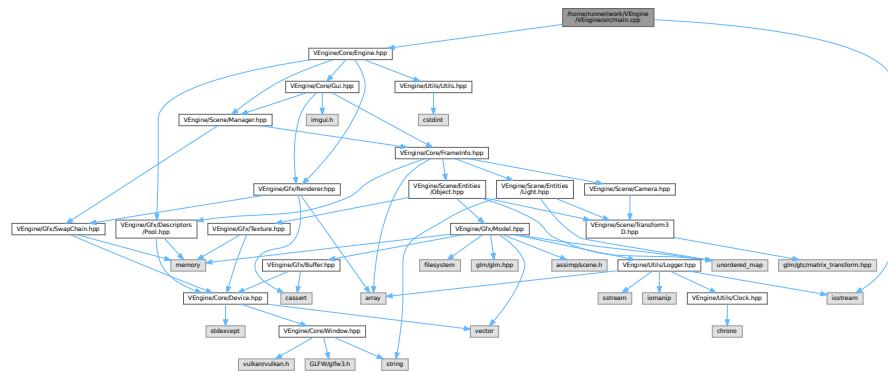
00251     barrier.image = m_textureImage;
00252     barrier.subresourceRange.baseMipLevel = 0;
00253     barrier.subresourceRange.levelCount = m_mipLevels;
00254     barrier.subresourceRange.baseArrayLayer = 0;
00255     barrier.subresourceRange.layerCount = m_layerCount;
00256
00257     if (newLayout == VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00258         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00259         if (m_format == VK_FORMAT_D32_SFLOAT_S8_UINT || m_format == VK_FORMAT_D24_UNORM_S8_UINT) {
00260             barrier.subresourceRange.aspectMask |= VK_IMAGE_ASPECT_STENCIL_BIT;
00261         }
00262     } else {
00263         barrier.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00264     }
00265     if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL) {
00266         barrier.srcAccessMask = 0;
00267         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00268         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00269         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00270     } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL) {
00271         barrier.srcAccessMask = 0;
00272         barrier.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00273         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00274         destinationStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00275     } else if (oldLayout == VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL && newLayout ==
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL) {
00276         barrier.srcAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
00277         barrier.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
00278
00279         sourceStage = VK_PIPELINE_STAGE_TRANSFER_BIT;
00280         destinationStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00281     } else if (oldLayout == VK_IMAGE_LAYOUT_UNDEFINED && newLayout ==
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL) {
00282         barrier.srcAccessMask = 0;
00283         barrier.dstAccessMask = VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT |
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00284         sourceStage = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
00285         destinationStage = VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00286     } else if (oldLayout == VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL && newLayout ==
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL) {
00287         // This says that any cmd that acts in color output or after (dstStage)
00288         // that needs read or write access to a resource
00289         // must wait until all previous read accesses in fragment shader
00290         barrier.srcAccessMask = VK_ACCESS_SHADER_READ_BIT | VK_ACCESS_SHADER_WRITE_BIT;
00291         barrier.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT;
00292         sourceStage = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
00293         destinationStage = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
00294     } else {
00295         throw std::invalid_argument("unsupported layout transition!");
00296     }
00297     vkCmdPipelineBarrier(commandBuffer, sourceStage, destinationStage, 0, 0, nullptr, 0, nullptr, 1,
&barrier);
00298 }

```

8.118 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```
#include <iostream>
#include "VEngine/Core/Engine.hpp"
```

Include dependency graph for main.cpp:



Functions

- int [main \(\)](#)

8.118.1 Function Documentation

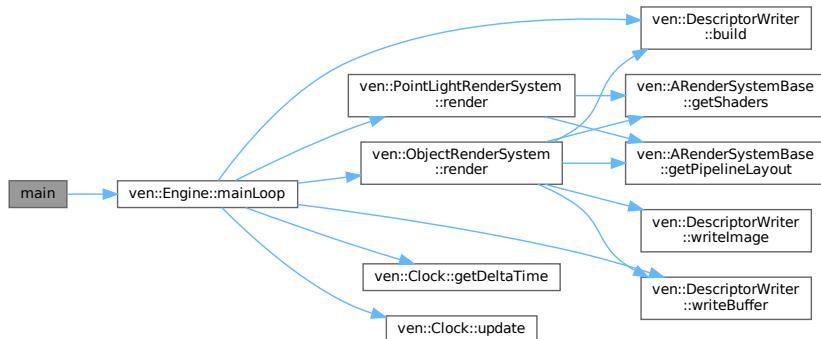
8.118.1.1 [main\(\)](#)

```
int main ()
```

Definition at line 7 of file [main.cpp](#).

References [ven::Engine::mainLoop\(\)](#).

Here is the call graph for this function:



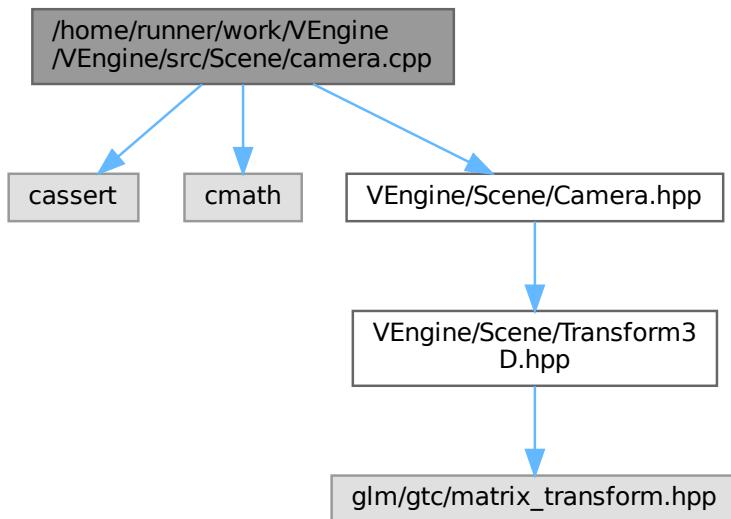
8.119 main.cpp

[Go to the documentation of this file.](#)

```
00001 #include <iostream>
00002
00003 #include "VEngine/Core/Engine.hpp"
00004
00005 using namespace ven;
00006
00007 int main()
00008 {
00009     try {
00010         Engine engine{};
00011         engine.mainLoop();
00012         engine.cleanup();
00013     } catch (const std::exception &e) {
00014         std::cerr << "std exception: " << e.what() << '\n';
00015         return EXIT_FAILURE;
00016     } catch (...) {
00017         std::cerr << "Unknown error\n";
00018         return EXIT_SUCCESS;
00019     }
00020     return EXIT_SUCCESS;
00021 }
```

8.120 /home/runner/work/VEngine/VEngine/src/Scene/camera.cpp File Reference

```
#include <cassert>
#include <cmath>
#include "VEngine/Scene/Camera.hpp"
Include dependency graph for camera.cpp:
```



8.121 camera.cpp

[Go to the documentation of this file.](#)

```
00001 #include <cassert>
00002 #include <cmath>
00003
00004 #include "VEngine/Scene/Camera.hpp"
00005
00006 void ven::Camera::setOrthographicProjection(const float left, const float right, const float top,
00007   const float bottom, const float near, const float far)
00008 {
00009     m_projectionMatrix = glm::mat4{1.0F};
00010     m_projectionMatrix[0][0] = 2.F / (right - left);
00011     m_projectionMatrix[1][1] = 2.F / (top - bottom);
00012     m_projectionMatrix[2][2] = 1.F / (far - near);
00013     m_projectionMatrix[3][0] = -(right + left) / (right - left);
00014     m_projectionMatrix[3][1] = -(bottom + top) / (top - bottom);
00015     m_projectionMatrix[3][2] = -near / (far - near);
00016 }
00017 void ven::Camera::setPerspectiveProjection(const float aspect)
00018 {
00019     assert(glm::abs(aspect - std::numeric_limits<float>::epsilon()) > 0.0F);
00020     const float tanHalfFov = std::tan(m_fov / 2.F);
00021     m_projectionMatrix = glm::mat4{0.0F};
00022     m_projectionMatrix[0][0] = 1.F / (aspect * tanHalfFov);
00023     m_projectionMatrix[1][1] = 1.F / (tanHalfFov);
00024     m_projectionMatrix[2][2] = m_far / (m_far - m_near);
00025     m_projectionMatrix[2][3] = 1.F;
00026     m_projectionMatrix[3][2] = -(m_far * m_near) / (m_far - m_near);
00027 }
00028
00029 void ven::Camera::setViewDirection(const glm::vec3 position, const glm::vec3 direction, const
00030   glm::vec3 up)
00031 {
00032     const glm::vec3 w{normalize(direction)};
00033     const glm::vec3 u{normalize(cross(w, up))};
00034     const glm::vec3 v{cross(w, u)};
00035
00036     m_viewMatrix = glm::mat4{1.F};
00037     m_viewMatrix[0][0] = u.x;
00038     m_viewMatrix[1][0] = u.y;
00039     m_viewMatrix[2][0] = u.z;
00040     m_viewMatrix[0][1] = v.x;
00041     m_viewMatrix[1][1] = v.y;
00042     m_viewMatrix[2][1] = v.z;
00043     m_viewMatrix[0][2] = w.x;
00044     m_viewMatrix[1][2] = w.y;
00045     m_viewMatrix[2][2] = w.z;
00046     m_viewMatrix[3][0] = -dot(u, position);
00047     m_viewMatrix[3][1] = -dot(v, position);
00048     m_viewMatrix[3][2] = -dot(w, position);
00049
00050     m_inverseViewMatrix = glm::mat4{1.F};
00051     m_inverseViewMatrix[0][0] = u.x;
00052     m_inverseViewMatrix[0][1] = u.y;
00053     m_inverseViewMatrix[0][2] = u.z;
00054     m_inverseViewMatrix[1][0] = v.x;
00055     m_inverseViewMatrix[1][1] = v.y;
00056     m_inverseViewMatrix[1][2] = v.z;
00057     m_inverseViewMatrix[2][0] = w.x;
00058     m_inverseViewMatrix[2][1] = w.y;
00059     m_inverseViewMatrix[2][2] = w.z;
00060     m_inverseViewMatrix[3][0] = position.x;
00061     m_inverseViewMatrix[3][1] = position.y;
00062     m_inverseViewMatrix[3][2] = position.z;
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
00076     m_viewMatrix = glm::mat4{1.F};
00077     m_viewMatrix[0][0] = u.x;
00078     m_viewMatrix[1][0] = u.y;
00079     m_viewMatrix[2][0] = u.z;
00080     m_viewMatrix[0][1] = v.x;
00081     m_viewMatrix[1][1] = v.y;
00082     m_viewMatrix[2][1] = v.z;
00083     m_viewMatrix[0][2] = w.x;
00084     m_viewMatrix[1][2] = w.y;
00085     m_viewMatrix[2][2] = w.z;
00086     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 }

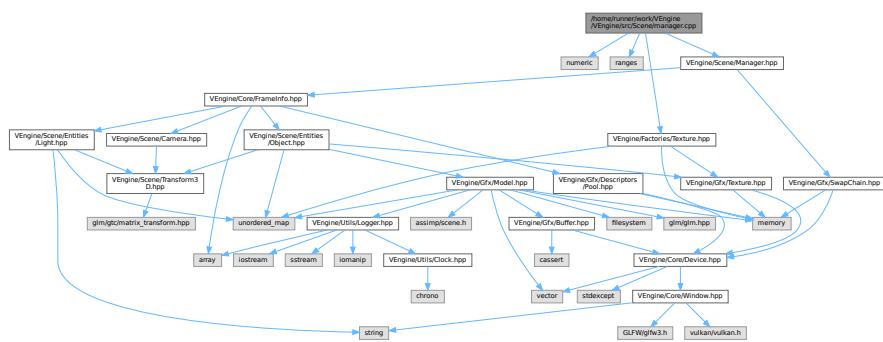
```

8.122 /home/runner/work/VEngine/VEngine/src/Scene/manager.cpp File Reference

```

#include <numeric>
#include <ranges>
#include "VEngine/Scene/Manager.hpp"
#include "VEngine/Factories/Texture.hpp"
Include dependency graph for manager.cpp:

```



8.123 manager.cpp

Go to the documentation of this file.

```

00001 #include <numeric>
00002 #include <ranges>
00003
00004 #include "VEngine/Scene/Manager.hpp"
00005 #include "VEngine/Factories/Texture.hpp"
00006
00007 ven::SceneManager::SceneManager(Device& device)
00008 {
00009     // including nonCoherentAtomSize allows us to flush a specific index at once
0010     unsigned long alignment = std::lcm(
0011         device.getProperties().limits.nonCoherentAtomSize,
0012         device.getProperties().limits.minUniformBufferOffsetAlignment
0013     );
0014     for (auto & uboBuffer : m_uboBuffers) {
0015         uboBuffer = std::make_unique<Buffer>(
0016             device,
0017             sizeof(ManagedObjectData),
0018             MAX_OBJECTS,
0019             VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT,
0020             VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT,
0021             alignment);

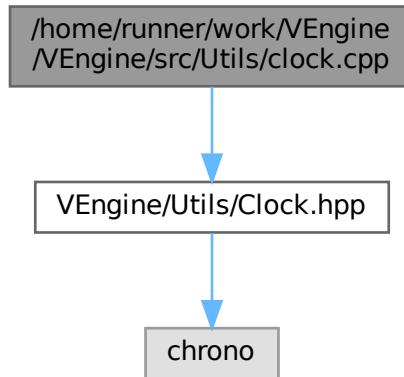
```

```
00022         uboBuffer->map();
00023     }
00024     Logger::logExecutionTime("Creating default texture", [&] {
00025         m_textureDefault = TextureFactory::create(device, "assets/textures/default.png");
00026     });
00027 }
00028
00029 void ven::SceneManager::updateBuffer(GlobalUbo &ubo, const unsigned long frameIndex, const float
frameTime)
00030 {
00031     uint8_t lightIndex = 0;
00032     const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameTime, {0.F, -1.F, 0.F});
00033
00034     for (Object& object : m_objects | std::views::values) {
00035         const ObjectBufferData data{
00036             .modelMatrix = object.transform.transformMatrix(),
00037             .normalMatrix = object.transform.normalMatrix()
00038         };
00039         m_uboBuffers.at(frameIndex)->writeToIndex(&data, object.getId());
00040         object.setBufferInfo(static_cast<int>(frameIndex),
00041             m_uboBuffers.at(frameIndex)->descriptorInfoForIndex(object.getId()));
00042     }
00043
00044     for (Light &light : m_lights | std::views::values) {
00045         auto&[position, color, shininess, padding] = ubo.pointLights.at(lightIndex);
00046         light.transform.translation = glm::vec3(rotateLight * glm::vec4(light.transform.translation,
00047             light.transform.scale.x));
00048         position = glm::vec4(light.transform.translation, light.transform.scale.x);
00049         color = light.color;
00050         shininess = light.getShininess();
00051         lightIndex++;
00052     }
00053
00054 void ven::SceneManager::destroyEntity(std::vector<unsigned int> *objectsIds, std::vector<unsigned int>
*lightsIds)
00055 {
00056     for (const unsigned int objectId : *objectsIds) {
00057         m_objects.erase(objectId);
00058     }
00059     for (const unsigned int lightId : *lightsIds) {
00060         m_lights.erase(lightId);
00061     }
00062     objectsIds->clear();
00063     lightsIds->clear();
00064     m_destroyState = false;
00065 }
```

8.124 /home/runner/work/VEngine/VEngine/src/Utils/clock.cpp File Reference

```
#include "VEngine/Utils/Clock.hpp"
```

Include dependency graph for clock.cpp:



8.125 clock.cpp

[Go to the documentation of this file.](#)

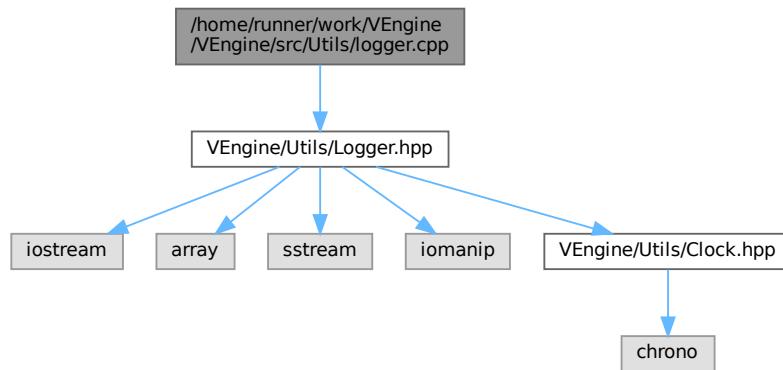
```

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 }
```

8.126 /home/runner/work/VEngine/VEngine/src/Utils/logger.cpp File Reference

```
#include "VEngine/Utils/Logger.hpp"
```

Include dependency graph for logger.cpp:



8.127 logger.cpp

[Go to the documentation of this file.](#)

```

00001 #include "VEngine/Utils/Logger.hpp"
00002
00003 ven::Logger::Logger()
00004 {
00005 #ifdef _WIN32
00006     const HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
00007     DWORD dwMode = 0;
00008     if (hOut != INVALID_HANDLE_VALUE && (GetConsoleMode(hOut, &dwMode) != 0)) {
00009         SetConsoleMode(hOut, dwMode | ENABLE_VIRTUAL_TERMINAL_PROCESSING);
00010     }
00011 #endif
00012 }
00013
00014 std::string ven::Logger::formatLogMessage(const LogLevel level, const std::string& message)
00015 {
00016     const auto inTime = std::chrono::system_clock::to_time_t(std::chrono::system_clock::now());
00017
00018     std::ostringstream ss;
00019     ss << "[" << std::put_time(std::localtime(&inTime), "%Y-%m-%d %X") << "] ";
00020     ss << "[" << LOG_LEVEL_STRING.at(static_cast<uint8_t>(level)) << "] " << message;
00021
00022     return ss.str();
00023 }
```


Index

/home/runner/work/VEngine/VEngine/README.md, 319
/home/runner/work/VEngine/VEngine/assets/shaders/fragments/lighting/pointLight.frag, 249
/home/runner/work/VEngine/VEngine/assets/shaders/fragments/lighting/swapChain.frag, 249
/home/runner/work/VEngine/VEngine/assets/shaders/fragments/madeForWork/Texture.hpp, 249
/home/runner/work/VEngine/VEngine/assets/shaders/vertexes/lighting/lightVerteX.hpp, 250
/home/runner/work/VEngine/VEngine/assets/shaders/vertexes/lighting/lightVerteX.hpp, 250
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Shaders.hpp, 298, 299
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/SwapChain.hpp, 300, 301
/home/runner/work/VEngine/VEngine/include/VEngine/Gfx/Texture.hpp, 283, 285
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Camera.hpp, 302, 304
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Light.hpp, 271, 272
/home/runner/work/VEngine/VEngine/include/VEngine/Core/Device.hpp, 280, 281
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Entities/Object.hpp, 252, 253
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Manager.hpp, 305, 306
/home/runner/work/VEngine/VEngine/include/VEngine/Scene/Transform3D.hpp, 307, 308
/home/runner/work/VEngine/VEngine/include/VEngine/Core/FrameInfo.hpp, 308, 309
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Colors.hpp, 310, 311
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/HashCombine.hpp, 314, 315
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Logger.hpp, 315, 316
/home/runner/work/VEngine/VEngine/include/VEngine/Utils/Utils.hpp, 317, 318
/home/runner/work/VEngine/VEngine/src/Core/GUI/init.cpp, 332, 333
/home/runner/work/VEngine/VEngine/src/Core/render.cpp, 334, 335
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/base.cpp, 339
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/object.cpp, 340, 341
/home/runner/work/VEngine/VEngine/src/Core/RenderSystems/pointLight.hpp, 341, 342
/home/runner/work/VEngine/VEngine/src/Core/device.cpp, 319, 321
/home/runner/work/VEngine/VEngine/src/Core/engine.cpp, 328
/home/runner/work/VEngine/VEngine/src/Core/eventManager.cpp, 330, 331
/home/runner/work/VEngine/VEngine/src/Core/window.cpp, 342
/home/runner/work/VEngine/VEngine/src/Factories/Light.cpp, 344
/home/runner/work/VEngine/VEngine/src/Factories/Model.cpp, 345

/home/runner/work/VEngine/VEngine/src/Factories/Object.cpp
 Object
 ven::Object, 168

/home/runner/work/VEngine/VEngine/src/Factories/Texture.cpp
 Texture
 ven::Texture, 231

/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/postProcessors
 postProcessors
 ven::PostProcessors, 211

/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/swapChain
 SwapChain
 ven::SwapChain, 218

/home/runner/work/VEngine/VEngine/src/Gfx/Descriptors/writeTexture
 writeTexture
 ven::WriteTexture, 231

/home/runner/work/VEngine/VEngine/src/Gfx/buffer.cpp, ~Window
 ~Window
 ven::Window, 244

/home/runner/work/VEngine/VEngine/src/Gfx/model.cpp,
 Model
 acquireNextImage
 ven::AcquireNextImage

/home/runner/work/VEngine/VEngine/src/Gfx/renderer.cpp, ven::SwapChain
 SwapChain
 addBinding
 ven::AddBinding

/home/runner/work/VEngine/VEngine/src/Gfx/shaders.cpp, ven::DescriptorsetLayout::Builder
 DescriptorsetLayout
 addLight
 ven::AddLight

/home/runner/work/VEngine/VEngine/src/Gfx/swapChain.cpp, ven::SceneManager
 SceneManager
 addObject
 ven::AddObject

/home/runner/work/VEngine/VEngine/src/Gfx/texture.cpp, ven::SceneManager
 SceneManager
 addPoolSize
 ven::AddPoolSize

/home/runner/work/VEngine/VEngine/src/Scene/camera.cpp, ven::DescriptorPool::Builder
 DescriptorPool
 allocateDescriptor
 ven::AllocateDescriptor

/home/runner/work/VEngine/VEngine/src/Scene/manager.cpp, ven::DescriptorPool
 DescriptorPool
 ambientLightColor
 ven::AmbientLightColor

/home/runner/work/VEngine/VEngine/src/Utils/clock.cpp, ven::GlobalUbo
 GlobalUbo
 AQUA_3
 ven::AQUA_3

/home/runner/work/VEngine/VEngine/src/Utils/logger.cpp, ven::Colors
 Colors
 AQUA_4
 ven::AQUA_4

/home/runner/work/VEngine/VEngine/src/main.cpp,
 Main
 AQUA_V
 ven::AQUA_V

~ARenderSystemBase
 ARenderSystemBase
 ven::ARenderSystemBase, 25

~Buffer
 Buffer
 ven::Buffer, 32

~Camera
 Camera
 ven::Camera, 55

~Clock
 Clock
 ven::Clock, 65

~DescriptorPool
 DescriptorPool
 ven::DescriptorPool, 84

~DescriptorsetLayout
 DescriptorsetLayout
 ven::DescriptorsetLayout, 89

~DescriptorWriter
 DescriptorWriter
 ven::DescriptorWriter, 93

~Device
 Device
 ven::Device, 98

~Engine
 Engine
 ven::Engine, 116

~EventManager
 EventManager
 ven::EventManager, 122

~Gui
 Gui
 ven::Gui, 133

~Light
 Light
 ven::Light, 148

~Model
 Model
 ven::Model, 161

BLACK_3
 ven::Colors, 72

BLACK_4
 ven::Colors, 72

BLACK_V
 ven::Colors, 73

BLUE_3
 ven::Colors, 73

BLUE_4
 ven::Colors, 73

BLUE_V
 ven::Colors, 73

Buffer
 ven::Buffer, 32

build
 ven::DescriptorPool::Builder, 44
 ven::DescriptorSetLayout::Builder, 49
 ven::DescriptorWriter, 93

Builder
 ven::DescriptorPool::Builder, 44
 ven::DescriptorSetLayout::Builder, 49

Camera
 ven::Camera, 55

camera
 ven::FrameInfo, 126

cameraSection
 ven::Gui, 133

capabilities
 ven::SwapChainSupportDetails, 227

checkDeviceExtensionSupport
 ven::Device, 99

checkValidationLayerSupport
 ven::Device, 99

chooseSwapExtent
 ven::SwapChain, 219

chooseSwapPresentMode
 ven::SwapChain, 219

chooseSwapSurfaceFormat
 ven::SwapChain, 219

cleanup
 ven::Engine, 117
 ven::Gui, 134

Clock
 ven::Clock, 65

color
 ven::Light, 150
 ven::LightPushConstantData, 154
 ven::Model::Vertex, 242
 ven::PointLightData, 185

COLOR_MAX
 ven, 17

COLOR_PRESETS_3
 ven::Colors, 73

COLOR_PRESETS_4
 ven::Colors, 73

COLOR_PRESETS_VK
 ven::Colors, 74

colorBlendAttachment
 ven::PipelineConfigInfo, 183

colorBlendInfo
 ven::PipelineConfigInfo, 183

commandBuffer
 ven::FrameInfo, 126

compareSwapFormats
 ven::SwapChain, 220

copyBuffer
 ven::Device, 99

copyBufferToImage
 ven::Device, 100

create
 ven::LightFactory, 152
 ven::ModelFactory, 165
 ven::ObjectFactory, 174
 ven::TextureFactory, 237

createBuffer
 ven::Device, 100

createCommandBuffers
 ven::Renderer, 195

createCommandPool
 ven::Device, 100

CreateDebugUtilsMessengerEXT
 device.cpp, 319

createDepthResources
 ven::SwapChain, 220

createFrameBuffers
 ven::SwapChain, 220

createGraphicsPipeline
 ven::Shaders, 212

createImageViews
 ven::SwapChain, 220

createImageWithInfo
 ven::Device, 101

createIndexBuffer
 ven::Model, 162

createInstance
 ven::Device, 101

createLogicalDevice
 ven::Device, 102

createPipeline
 ven::ARenderSystemBase, 26

createPipelineLayout
 ven::ARenderSystemBase, 26

createRenderPass
 ven::SwapChain, 220

createShaderModule
 ven::Shaders, 213

createSurface
 ven::Device, 102

createSwapChain
 ven::SwapChain, 220

createSyncObjects
 ven::SwapChain, 220

createTextureImage
 ven::Texture, 231

createTextureImageView
 ven::Texture, 231

createTextureSampler
 ven::Texture, 232

createVertexBuffer
 ven::Model, 162

createWindow
 ven::Window, 245

createWindowSurface
 ven::Window, 245

CYAN_3
 ven::Colors, 74

CYAN_4
 ven::Colors, 74

CYAN_V
 ven::Colors, 75

debugCallback
 device.cpp, 319

DEFAULT_AMBIENT_LIGHT_COLOR
 ven, 17

DEFAULT_AMBIENT_LIGHT_INTENSITY
 ven, 17

DEFAULT_CLEAR_COLOR
 ven, 17

DEFAULT_CLEAR_DEPTH
 ven, 18

DEFAULT_FAR
 ven, 18

DEFAULT_FOV
 ven, 18

DEFAULT_HEIGHT
 ven, 18

DEFAULT_KEY_MAPPINGS
 ven, 18

DEFAULT_LIGHT_COLOR
 ven, 18

DEFAULT_LIGHT_INTENSITY
 ven, 18

DEFAULT_LIGHT_RADIUS
 ven, 19

DEFAULT_LOOK_SPEED
 ven, 19

DEFAULT_MAX_SETS
 ven, 19

DEFAULT_MOVE_SPEED
 ven, 19

DEFAULT_NEAR
 ven, 19

DEFAULT_POSITION
 ven, 19

DEFAULT_ROTATION
 ven, 20

DEFAULT_SHININESS
 ven, 20

DEFAULT_TITLE
 ven, 20

DEFAULT_TRANSFORM
 ven, 20

DEFAULT_WIDTH
 ven, 20

defaultPipelineConfigInfo
 ven::Shaders, 213

deltaTimeMS
 ven::Gui::ClockData, 69

depthStencilInfo
 ven::PipelineConfigInfo, 183

DESCRIPTOR_COUNT
 ven, 20

descriptorInfo
 ven::Buffer, 32

descriptorInfoForIndex
 ven::Buffer, 33

DescriptorPool
 ven::DescriptorPool, 84, 85

DescriptorSetLayout
 ven::DescriptorSetLayout, 89, 90

DescriptorWriter
 ven::DescriptorPool, 86
 ven::DescriptorSetLayout, 90
 ven::DescriptorWriter, 93

DestroyDebugUtilsMessengerEXT
 device.cpp, 320

destroyEntity
 ven::SceneManager, 204

destroyLight
 ven::SceneManager, 205

destroyObject
 ven::SceneManager, 205

Device
 ven::Device, 98, 99

device
 ven::Device, 103

device.cpp
 CreateDebugUtilsMessengerEXT, 319
 debugCallback, 319
 DestroyDebugUtilsMessengerEXT, 320

devicePropertiesSection
 ven::Gui, 134

dir
 ven::KeyAction, 143

draw
 ven::Model, 163

duplicate
 ven::LightFactory, 152
 ven::ObjectFactory, 175

dynamicStateEnables
 ven::PipelineConfigInfo, 183

dynamicStateInfo
 ven::PipelineConfigInfo, 183

EDITOR
 ven, 16

enableValidationLayers
 ven::Device, 112

endFrame
 ven::Renderer, 195

endSingleTimeCommands
 ven::Device, 104

endSwapChainRenderPass
 ven::Renderer, 196

Engine
 ven::Engine, 116, 117

ENGINE_STATE
 ven, 15

EPSILON
 ven, 20

EventManager
 ven::EventManager, 122

eventManager.cpp

GLM_ENABLE_EXPERIMENTAL, 331
EXIT
 ven, 16
extentAspectRatio
 ven::SwapChain, 221

findDepthFormat
 ven::SwapChain, 221
findMemoryType
 ven::Device, 104
findPhysicalQueueFamilies
 ven::Device, 104
findQueueFamilies
 ven::Device, 105
findSupportedFormat
 ven::Device, 105
flush
 ven::Buffer, 33
flushIndex
 ven::Buffer, 34
formatLogMessage
 ven::Logger, 156
formats
 ven::SwapChainSupportDetails, 227
fps
 ven::Gui::ClockData, 69
framebufferResizeCallback
 ven::Window, 245
frameDescriptorPool
 ven::FrameInfo, 126
frameIndex
 ven::FrameInfo, 126
frameTime
 ven::FrameInfo, 126
freeCommandBuffers
 ven::Renderer, 196
freeDescriptors
 ven::DescriptorPool, 85
FUCHSIA_3
 ven::Colors, 75
FUCHSIA_4
 ven::Colors, 75
FUCHSIA_V
 ven::Colors, 75

getAlignment
 ven::Buffer, 35
getAlignmentSize
 ven::Buffer, 35
getAspectRatio
 ven::Renderer, 196
getAttributeDescriptions
 ven::Model::Vertex, 241
getBindingDescriptions
 ven::Model::Vertex, 241
getBuffer
 ven::Buffer, 35
getBufferInfo
 ven::Object, 169

getBufferInfoForObject
 ven::SceneManager, 205
getBufferSize
 ven::Buffer, 36
getClearColor
 ven::Renderer, 196
getColorForDuration
 ven::Logger, 156
getCommandPool
 ven::Device, 106
getCurrentCommandBuffer
 ven::Renderer, 197
getDeltaTime
 ven::Clock, 66
getDeltaTimeMS
 ven::Clock, 66
getDescriptorPool
 ven::DescriptorPool, 85
getDescriptorSetLayout
 ven::DescriptorsetLayout, 90
getDestroyState
 ven::SceneManager, 205
getDevice
 ven::ARenderSystemBase, 27
getDiffuseMap
 ven::Object, 169
getExtent
 ven::Texture, 232
 ven::Window, 246
getFar
 ven::Camera, 56
getFormat
 ven::Texture, 232
getFov
 ven::Camera, 56
getFPS
 ven::Clock, 66
getFrameBuffer
 ven::SwapChain, 221
getFrameIndex
 ven::Renderer, 197
getGLFWwindow
 ven::Window, 246
getGraphicsQueue
 ven::Device, 106
getId
 ven::Light, 149
 ven::Object, 169
getImage
 ven::Texture, 233
getImageInfo
 ven::Texture, 233
getImageLayout
 ven::Texture, 233
getImageView
 ven::SwapChain, 221
 ven::Texture, 233
getInstance

ven::Device, 106
 ven::Logger, 156
 getInstanceCount
 ven::Buffer, 36
 getInstanceSize
 ven::Buffer, 36
 getInverseView
 ven::Camera, 56
 getLights
 ven::SceneManager, 205
 getLightsToRemove
 ven::Gui, 135
 getLookSpeed
 ven::Camera, 57
 getMappedMemory
 ven::Buffer, 36
 getMemoryPropertyFlags
 ven::Buffer, 36
 getModel
 ven::Object, 170
 getMoveSpeed
 ven::Camera, 57
 getName
 ven::Light, 149
 ven::Object, 170
 getNear
 ven::Camera, 57
 getObjects
 ven::SceneManager, 206
 getObjectsToRemove
 ven::Gui, 135
 getPhysicalDevice
 ven::Device, 106
 getPipelineLayout
 ven::ARenderSystemBase, 27
 getProjection
 ven::Camera, 58
 getProperties
 ven::Device, 107
 getRenderPass
 ven::SwapChain, 221
 getRequiredExtensions
 ven::Device, 107
 getShaders
 ven::ARenderSystemBase, 28
 getShininess
 ven::Light, 149
 getState
 ven::Gui, 135
 getSwapChainExtent
 ven::SwapChain, 221
 getSwapChainImageFormat
 ven::SwapChain, 222
 getSwapChainRenderPass
 ven::Renderer, 198
 getSwapChainSupport
 ven::Device, 107
 getTextureDefault
 ven::SceneManager, 206
 getUboBuffers
 ven::SceneManager, 206
 getUsageFlags
 ven::Buffer, 36
 getView
 ven::Camera, 58
 getWindow
 ven::Renderer, 198
 GLFW_INCLUDE_VULKAN
 Window.hpp, 268
 GLM_ENABLE_EXPERIMENTAL
 eventManager.cpp, 331
 model.cpp, 354
 globalDescriptorSet
 ven::FrameInfo, 126
 graphicsFamily
 ven::QueueFamilyIndices, 191
 graphicsFamilyHasValue
 ven::QueueFamilyIndices, 191
 graphicsQueue
 ven::Device, 108
 GRAY_3
 ven::Colors, 75
 GRAY_4
 ven::Colors, 75
 GRAY_V
 ven::Colors, 75
 GREEN_3
 ven::Colors, 76
 GREEN_4
 ven::Colors, 76
 GREEN_V
 ven::Colors, 76
 Gui
 ven::Gui, 133
 GUI_STATE
 ven, 16
 handleEvents
 ven::EventManager, 122
 hasGlfwRequiredInstanceExtensions
 ven::Device, 108
 hashCombine
 ven, 16
 height
 ven::SwapChain, 222
 HIDDEN
 ven, 16
 imageCount
 ven::SwapChain, 222
 imageView
 ven::Texture, 233
 indices
 ven::Model::Builder, 53
 INFO
 ven, 16
 init

ven::Gui, 135
ven::SwapChain, 222
initStyle
 ven::Gui, 136
inputAssemblyInfo
 ven::PipelineConfigInfo, 183
inputsSection
 ven::Gui, 137
invalidate
 ven::Buffer, 37
invalidateIndex
 ven::Buffer, 37
inverseView
 ven::GlobalUbo, 130
isComplete
 ven::QueueFamilyIndices, 190
isDeviceSuitable
 ven::Device, 108
isFrameInProgress
 ven::Renderer, 198
isKeyJustPressed
 ven::EventManager, 122
IsLegacyNativeDupe
 ven::Gui::funcs, 128

key
 ven::KeyAction, 143

Light
 ven::Light, 148, 149
lights
 ven::FrameInfo, 127
lightsSection
 ven::Gui, 137
LIME_3
 ven::Colors, 76
LIME_4
 ven::Colors, 76
LIME_V
 ven::Colors, 76
loadAll
 ven::ModelFactory, 165
 ven::TextureFactory, 237
loadModel
 ven::Model::Builder, 52
loadObjects
 ven::Engine, 117
LOG_LEVEL_COLOR
 ven::Logger, 158
LOG_LEVEL_STRING
 ven::Logger, 158
logExecutionTime
 ven::Logger, 157
Logger
 ven::Logger, 156
LogLevel
 ven, 16
logWarning
 ven::Logger, 158

lookDown
 ven::KeyMappings, 145
lookLeft
 ven::KeyMappings, 145
lookRight
 ven::KeyMappings, 145
lookUp
 ven::KeyMappings, 145

m_alignmentSize
 ven::Buffer, 40
m_bindings
 ven::DescriptorsetLayout, 91
 ven::DescriptorsetLayout::Builder, 50
m_buffer
 ven::Buffer, 40
m_bufferInfo
 ven::Object, 171
m_bufferSize
 ven::Buffer, 40
m_clearValues
 ven::Renderer, 200
m_commandBuffers
 ven::Renderer, 200
m_commandPool
 ven::Device, 112
m_currentFrame
 ven::SwapChain, 223
m_currentFrameIndex
 ven::Renderer, 200
m_currentImageIndex
 ven::Renderer, 200
m_currentLightId
 ven::LightFactory, 153
 ven::SceneManager, 208
m_currentObjId
 ven::ObjectFactory, 176
 ven::SceneManager, 208
m_debugMessenger
 ven::Device, 112
m_deltaTime
 ven::Clock, 68
m_depthImageMemory
 ven::SwapChain, 223
m_depthImages
 ven::SwapChain, 223
m_depthImageViews
 ven::SwapChain, 223
m_descriptor
 ven::Texture, 234
m_descriptorPool
 ven::DescriptorPool, 86
m_descriptorsetLayout
 ven::DescriptorsetLayout, 91
m_destroyState
 ven::SceneManager, 208
m_device
 ven::ARenderSystemBase, 28
 ven::Buffer, 40

ven::DescriptorPool, 86
 ven::DescriptorPool::Builder, 46
 ven::DescriptorSetLayout, 91
 ven::DescriptorSetLayout::Builder, 50
 ven::Device, 112
 ven::Engine, 119
 ven::Model, 164
 ven::Renderer, 201
 ven::Shaders, 214
 ven::SwapChain, 224
 ven::Texture, 234
 m_deviceExtensions
 ven::Device, 112
 m_diffuseMap
 ven::Object, 171
 m_extent
 ven::Texture, 235
 m_far
 ven::Camera, 62
 m_format
 ven::Texture, 235
 m_fov
 ven::Camera, 62
 m_fragShaderModule
 ven::Shaders, 214
 m_framebufferResized
 ven::Window, 248
 m_framePools
 ven::Engine, 119
 m_globalPool
 ven::Engine, 119
 m_graphicsPipeline
 ven::Shaders, 214
 m_graphicsQueue
 ven::Device, 112
 m_gui
 ven::Engine, 119
 m_hasIndexBuffer
 ven::Model, 164
 m_height
 ven::Window, 248
 m_imageAvailableSemaphores
 ven::SwapChain, 224
 m_imagesInFlight
 ven::SwapChain, 224
 m_indexBuffer
 ven::Model, 164
 m_indexCount
 ven::Model, 164
 m_inFlightFences
 ven::SwapChain, 224
 m_instance
 ven::Device, 112
 m_instanceCount
 ven::Buffer, 41
 m_instanceSize
 ven::Buffer, 41
 m_intensity
 ven::Gui, 140
 m_inverseViewMatrix
 ven::Camera, 62
 m_io
 ven::Gui, 140
 m_isFrameStarted
 ven::Renderer, 201
 m_isStopped
 ven::Clock, 68
 m_keyState
 ven::EventManager, 124
 m_layerCount
 ven::Texture, 235
 m_lightId
 ven::Light, 150
 m_lights
 ven::SceneManager, 208
 m_lightsToRemove
 ven::Gui, 140
 m_lookSpeed
 ven::Camera, 62
 m_mapped
 ven::Buffer, 41
 m_maxSets
 ven::DescriptorPool::Builder, 46
 m_memory
 ven::Buffer, 41
 m_memoryPropertyFlags
 ven::Buffer, 41
 m_mipLevels
 ven::Texture, 235
 m_model
 ven::Object, 172
 m_moveSpeed
 ven::Camera, 62
 m_name
 ven::Light, 150
 ven::Object, 172
 m_near
 ven::Camera, 63
 m_objects
 ven::SceneManager, 208
 m_objectsToRemove
 ven::Gui, 140
 m_objId
 ven::Object, 172
 m_oldSwapChain
 ven::SwapChain, 224
 m_physicalDevice
 ven::Device, 113
 m_pipelineLayout
 ven::ARenderSystemBase, 28
 m_pool
 ven::DescriptorWriter, 95
 m_poolFlags
 ven::DescriptorPool::Builder, 46
 m_poolSizes
 ven::DescriptorPool::Builder, 46

m_presentQueue
 ven::Device, 113
m_projectionMatrix
 ven::Camera, 63
m_properties
 ven::Device, 113
m_renderer
 ven::Engine, 119
m_renderFinishedSemaphores
 ven::SwapChain, 224
m_renderPass
 ven::SwapChain, 225
m_sceneManager
 ven::Engine, 120
m_setLayout
 ven::DescriptorWriter, 95
m_shaders
 ven::ARenderSystemBase, 29
m_shininess
 ven::Gui, 140
 ven::Light, 150
m_startTime
 ven::Clock, 68
m_state
 ven::Engine, 120
 ven::Gui, 141
m_stopTime
 ven::Clock, 68
m_surface
 ven::Device, 113
m_swapChain
 ven::Renderer, 201
 ven::SwapChain, 225
m_swapChainDepthFormat
 ven::SwapChain, 225
m_swapChainExtent
 ven::SwapChain, 225
m_swapChainFrameBuffers
 ven::SwapChain, 225
m_swapChainImageFormat
 ven::SwapChain, 225
m_swapChainImages
 ven::SwapChain, 226
m_swapChainImageViews
 ven::SwapChain, 226
m_textureDefault
 ven::SceneManager, 208
m_textureImage
 ven::Texture, 235
m_textureImageMemory
 ven::Texture, 235
m_textureImageView
 ven::Texture, 235
m_textureLayout
 ven::Texture, 236
m_textureSampler
 ven::Texture, 236
m_uboBuffers
 ven::SceneManager, 208
m_usageFlags
 ven::Buffer, 41
m_validationLayers
 ven::Device, 113
m_vertexBuffer
 ven::Model, 164
m_vertexCount
 ven::Model, 164
m_vertShaderModule
 ven::Shaders, 214
m_viewMatrix
 ven::Camera, 63
m_width
 ven::Window, 248
m_window
 ven::Device, 113
 ven::Engine, 120
 ven::Renderer, 201
 ven::Window, 248
m_windowExtent
 ven::SwapChain, 226
m_writes
 ven::DescriptorWriter, 95
MAGENTA_3
 ven::Colors, 76
MAGENTA_4
 ven::Colors, 77
MAGENTA_V
 ven::Colors, 77
main
 main.cpp, 371
main.cpp
 main, 371
mainLoop
 ven::Engine, 118
Map
 ven::Light, 148
 ven::Object, 168
map
 ven::Buffer, 38
MAROON_3
 ven::Colors, 77
MAROON_4
 ven::Colors, 77
MAROON_V
 ven::Colors, 77
MAX_FRAMES_IN_FLIGHT
 ven, 21
MAX_LIGHTS
 ven, 21
MAX_OBJECTS
 ven, 21
Model
 ven::Model, 161, 162
model.cpp
 GLM_ENABLE_EXPERIMENTAL, 354
modelMatrix

ven::ObjectBufferData, 173
 ven::ObjectPushConstantData, 176
 moveBackward
 ven::KeyMappings, 145
 moveCamera
 ven::EventManager, 123
 moveDown
 ven::KeyMappings, 145
 moveForward
 ven::KeyMappings, 145
 moveLeft
 ven::KeyMappings, 146
 moveRight
 ven::KeyMappings, 146
 moveUp
 ven::KeyMappings, 146
 multisampleInfo
 ven::PipelineConfigInfo, 184

 NAVY_3
 ven::Colors, 77
 NAVY_4
 ven::Colors, 77
 NAVY_V
 ven::Colors, 78
 NIGHT_BLUE_3
 ven::Colors, 78
 NIGHT_BLUE_4
 ven::Colors, 78
 NIGHT_BLUE_V
 ven::Colors, 78
 normal
 ven::Model::Vertex, 242
 normalMatrix
 ven::ObjectBufferData, 173
 ven::ObjectPushConstantData, 176
 ven::Transform3D, 239
 now
 ven::Clock, 66
 numLights
 ven::GlobalUbo, 130

 Object
 ven::Object, 168, 169
 ObjectRenderSystem
 ven::ObjectRenderSystem, 179
 objects
 ven::FrameInfo, 127
 objectsSection
 ven::Gui, 137
 OLIVE_3
 ven::Colors, 78
 OLIVE_4
 ven::Colors, 78
 OLIVE_V
 ven::Colors, 78
 operator()
 std::hash< ven::Model::Vertex >, 142
 operator=

 ven::Buffer, 38
 ven::Camera, 58
 ven::Clock, 67
 ven::DescriptorPool, 86
 ven::DescriptorSetLayout, 90
 ven::DescriptorWriter, 93
 ven::Device, 109
 ven::Engine, 119
 ven::EventManager, 123
 ven::Gui, 138
 ven::Light, 149
 ven::Model, 163
 ven::Object, 170
 ven::ObjectRenderSystem, 180
 ven::PipelineConfigInfo, 182
 ven::PointLightRenderSystem, 189
 ven::Renderer, 199
 ven::SceneManager, 207
 ven::Shaders, 214
 ven::SwapChain, 222
 ven::Texture, 233
 ven::Window, 247
 operator==
 ven::Model::Vertex, 242
 overwrite
 ven::DescriptorWriter, 94

 padding
 ven::PointLightData, 185
 PAUSED
 ven, 16
 pickPhysicalDevice
 ven::Device, 109
 PipelineConfigInfo
 ven::PipelineConfigInfo, 182
 pipelineLayout
 ven::PipelineConfigInfo, 184
 PLAYER
 ven, 16
 PointLightRenderSystem
 ven::PointLightRenderSystem, 188
 pointLights
 ven::GlobalUbo, 130
 populateDebugMessengerCreateInfo
 ven::Device, 109
 position
 ven::LightPushConstantData, 154
 ven::Model::Vertex, 242
 ven::PointLightData, 185
 presentFamily
 ven::QueueFamilyIndices, 191
 presentFamilyHasValue
 ven::QueueFamilyIndices, 191
 presentModes
 ven::SwapChainSupportDetails, 227
 presentQueue
 ven::Device, 110
 processKeyActions
 ven::EventManager, 123

processMesh
 ven::Model::Builder, 52

processNode
 ven::Model::Builder, 52

projection
 ven::GlobalUbo, 130

querySwapChainSupport
 ven::Device, 110

radius
 ven::LightPushConstantData, 154

rasterizationInfo
 ven::PipelineConfigInfo, 184

readFile
 ven::Shaders, 214

recreateSwapChain
 ven::Renderer, 199

RED_3
 ven::Colors, 79

RED_4
 ven::Colors, 79

RED_V
 ven::Colors, 79

render
 ven::ARenderSystemBase, 28
 ven::Gui, 138
 ven::ObjectRenderSystem, 180
 ven::PointLightRenderSystem, 189

Renderer
 ven::Renderer, 194

rendererSection
 ven::Gui, 138

renderFrameWindow
 ven::Gui, 139

renderPass
 ven::PipelineConfigInfo, 184

renderSystemLayout
 ven::ARenderSystemBase, 29

resetPool
 ven::DescriptorPool, 86

resetWindowResizedFlag
 ven::Window, 247

resume
 ven::Clock, 67

rotation
 ven::Transform3D, 240

sampler
 ven::Texture, 234

scale
 ven::Transform3D, 240

SceneManager
 ven::SceneManager, 203, 204

setBufferInfo
 ven::Object, 171

setClearValue
 ven::Renderer, 199

setDestroyState
 ven::SceneManager, 207

setDiffuseMap
 ven::Object, 171

setFar
 ven::Camera, 58

setFov
 ven::Camera, 59

setFullscreen
 ven::Window, 247

setLookSpeed
 ven::Camera, 59

setMaxSets
 ven::DescriptorPool::Builder, 45

setModel
 ven::Object, 171

setMoveSpeed
 ven::Camera, 60

setName
 ven::Light, 150
 ven::Object, 171

setNear
 ven::Camera, 60

setOrthographicProjection
 ven::Camera, 61

setPerspectiveProjection
 ven::Camera, 61

setPoolFlags
 ven::DescriptorPool::Builder, 45

setShininess
 ven::Light, 150

setState
 ven::Gui, 139

setupDebugMessenger
 ven::Device, 110

setViewDirection
 ven::Camera, 61

setViewTarget
 ven::Camera, 61

setViewXYZ
 ven::Camera, 62

Shaders
 ven::Shaders, 211, 212

SHADERS_BIN_PATH
 ven, 21

shininess
 ven::PointLightData, 186

SHOW_EDITOR
 ven, 16

SHOW_PLAYER
 ven, 16

SILVER_3
 ven::Colors, 79

SILVER_4
 ven::Colors, 79

SILVER_V
 ven::Colors, 79

SKY_BLUE_3
 ven::Colors, 79

SKY_BLUE_4
 ven::Colors, 80
 SKY_BLUE_V
 ven::Colors, 80
 start
 ven::Clock, 67
 STB_IMAGE_IMPLEMENTATION
 texture.cpp, 366
 std::hash< ven::Model::Vertex >, 141
 operator(), 142
 stop
 ven::Clock, 67
 submitCommandBuffers
 ven::SwapChain, 223
 subpass
 ven::PipelineConfigInfo, 184
 SUNSET_3
 ven::Colors, 80
 SUNSET_4
 ven::Colors, 80
 SUNSET_V
 ven::Colors, 80
 surface
 ven::Device, 111
 SwapChain
 ven::SwapChain, 218, 219

 TEAL_3
 ven::Colors, 80
 TEAL_4
 ven::Colors, 80
 TEAL_V
 ven::Colors, 81
 Texture
 ven::Texture, 230, 231
 texture.cpp
 STB_IMAGE_IMPLEMENTATION, 366
 TimePoint
 ven, 15
 toggleGui
 ven::KeyMappings, 146
 transform
 ven::Camera, 63
 ven::Light, 151
 ven::Object, 172
 transformMatrix
 ven::Transform3D, 239
 transitionImageLayout
 ven::Device, 111
 transitionLayout
 ven::Texture, 234
 translation
 ven::Transform3D, 240

 unmap
 ven::Buffer, 39
 update
 ven::Clock, 67
 updateBuffer
 ven::SceneManager, 207
 updateDescriptor
 ven::Texture, 234
 updateEngineState
 ven::EventManager, 123
 uv
 ven::Model::Vertex, 243

 value
 ven::KeyAction, 143
 ven, 13
 COLOR_MAX, 17
 DEFAULT_AMBIENT_LIGHT_COLOR, 17
 DEFAULT_AMBIENT_LIGHT_INTENSITY, 17
 DEFAULT_CLEAR_COLOR, 17
 DEFAULT_CLEAR_DEPTH, 18
 DEFAULT_FAR, 18
 DEFAULT_FOV, 18
 DEFAULT_HEIGHT, 18
 DEFAULT_KEY_MAPPINGS, 18
 DEFAULT_LIGHT_COLOR, 18
 DEFAULT_LIGHT_INTENSITY, 18
 DEFAULT_LIGHT_RADIUS, 19
 DEFAULT_LOOK_SPEED, 19
 DEFAULT_MAX_SETS, 19
 DEFAULT_MOVE_SPEED, 19
 DEFAULT_NEAR, 19
 DEFAULT_POSITION, 19
 DEFAULT_ROTATION, 20
 DEFAULT_SHININESS, 20
 DEFAULT_TITLE, 20
 DEFAULT_TRANSFORM, 20
 DEFAULT_WIDTH, 20
 DESCRIPTOR_COUNT, 20
 EDITOR, 16
 ENGINE_STATE, 15
 EPSILON, 20
 EXIT, 16
 GUI_STATE, 16
 hashCombine, 16
 HIDDEN, 16
 INFO, 16
 LogLevel, 16
 MAX_FRAMES_IN_FLIGHT, 21
 MAX_LIGHTS, 21
 MAX_OBJECTS, 21
 PAUSED, 16
 PLAYER, 16
 SHADERS_BIN_PATH, 21
 SHOW_EDITOR, 16
 SHOW_PLAYER, 16
 TimePoint, 15
 WARNING, 16
 ven::ARenderSystemBase, 23
 ~ARenderSystemBase, 25
 ARenderSystemBase, 25
 createPipeline, 26
 createPipelineLayout, 26
 getDevice, 27

getPipelineLayout, 27
getShaders, 28
m_device, 28
m_pipelineLayout, 28
m_shaders, 29
render, 28
renderSystemLayout, 29
ven::Buffer, 29
~Buffer, 32
Buffer, 32
descriptorInfo, 32
descriptorInfoForIndex, 33
flush, 33
flushIndex, 34
getAlignment, 35
getAlignmentSize, 35
getBuffer, 35
getBufferSize, 36
getInstanceCount, 36
getInstanceSize, 36
getMappedMemory, 36
getMemoryPropertyFlags, 36
getUsageFlags, 36
invalidate, 37
invalidateIndex, 37
m_alignmentSize, 40
m_buffer, 40
m_bufferSize, 40
m_device, 40
m_instanceCount, 41
m_instanceSize, 41
m_mapped, 41
m_memory, 41
m_memoryPropertyFlags, 41
m_usageFlags, 41
map, 38
operator=, 38
unmap, 39
writeToBuffer, 39
writeToIndex, 39
ven::Camera, 53
~Camera, 55
Camera, 55
getFar, 56
getFov, 56
getInverseView, 56
getLookSpeed, 57
getMoveSpeed, 57
getNear, 57
getProjection, 58
getView, 58
m_far, 62
m_fov, 62
m_inverseViewMatrix, 62
m_lookSpeed, 62
m_moveSpeed, 62
m_near, 63
m_projectionMatrix, 63
m_viewMatrix, 63
operator=, 58
setFar, 58
setFov, 59
setLookSpeed, 59
setMoveSpeed, 60
setNear, 60
setOrthographicProjection, 61
setPerspectiveProjection, 61
setViewDirection, 61
setViewTarget, 61
setViewXYZ, 62
transform, 63
ven::Clock, 64
~Clock, 65
Clock, 65
getDeltaTime, 66
getDeltaTimeMS, 66
getFPS, 66
m_deltaTime, 68
m_isStopped, 68
m_startTime, 68
m_stopTime, 68
now, 66
operator=, 67
resume, 67
start, 67
stop, 67
update, 67
ven::Colors, 70
AQUA_3, 72
AQUA_4, 72
AQUA_V, 72
BLACK_3, 72
BLACK_4, 72
BLACK_V, 73
BLUE_3, 73
BLUE_4, 73
BLUE_V, 73
COLOR_PRESETS_3, 73
COLOR_PRESETS_4, 73
COLOR_PRESETS_VK, 74
CYAN_3, 74
CYAN_4, 74
CYAN_V, 75
FUCHSIA_3, 75
FUCHSIA_4, 75
FUCHSIA_V, 75
GRAY_3, 75
GRAY_4, 75
GRAY_V, 75
GREEN_3, 76
GREEN_4, 76
GREEN_V, 76
LIME_3, 76
LIME_4, 76
LIME_V, 76
MAGENTA_3, 76

MAGENTA_4, 77
 MAGENTA_V, 77
 MAROON_3, 77
 MAROON_4, 77
 MAROON_V, 77
 NAVY_3, 77
 NAVY_4, 77
 NAVY_V, 78
 NIGHT_BLUE_3, 78
 NIGHT_BLUE_4, 78
 NIGHT_BLUE_V, 78
 OLIVE_3, 78
 OLIVE_4, 78
 OLIVE_V, 78
 RED_3, 79
 RED_4, 79
 RED_V, 79
 SILVER_3, 79
 SILVER_4, 79
 SILVER_V, 79
 SKY_BLUE_3, 79
 SKY_BLUE_4, 80
 SKY_BLUE_V, 80
 SUNSET_3, 80
 SUNSET_4, 80
 SUNSET_V, 80
 TEAL_3, 80
 TEAL_4, 80
 TEAL_V, 81
 WHITE_3, 81
 WHITE_4, 81
 WHITE_V, 81
 YELLOW_3, 81
 YELLOW_4, 81
 YELLOW_V, 81

ven::DescriptorPool, 82
 ~DescriptorPool, 84
 allocateDescriptor, 85
 DescriptorPool, 84, 85
 DescriptorWriter, 86
 freeDescriptors, 85
 getDescriptorPool, 85
 m_descriptorPool, 86
 m_device, 86
 operator=, 86
 resetPool, 86

ven::DescriptorPool::Builder, 42
 addPoolSize, 44
 build, 44
 Builder, 44
 m_device, 46
 m_maxSets, 46
 m_poolFlags, 46
 m_poolSizes, 46
 setMaxSets, 45
 setPoolFlags, 45

ven::DescriptorsetLayout, 87
 ~DescriptorsetLayout, 89

DescriptorsetLayout, 89, 90
 DescriptorWriter, 90
 getDescriptorsetLayout, 90
 m_bindings, 91
 m_descriptorsetLayout, 91
 m_device, 91
 operator=, 90

ven::DescriptorsetLayout::Builder, 47
 addBinding, 49
 build, 49
 Builder, 49
 m_bindings, 50
 m_device, 50

ven::DescriptorWriter, 91
 ~DescriptorWriter, 93
 build, 93
 DescriptorWriter, 93
 m_pool, 95
 m_setLayout, 95
 m_writes, 95
 operator=, 93
 overwrite, 94
 writeBuffer, 94
 writeImage, 94

ven::Device, 95
 ~Device, 98
 beginSingleTimeCommands, 99
 checkDeviceExtensionSupport, 99
 checkValidationLayerSupport, 99
 copyBuffer, 99
 copyBufferToImage, 100
 createBuffer, 100
 createCommandPool, 100
 createImageWithInfo, 101
 createInstance, 101
 createLogicalDevice, 102
 createSurface, 102
 Device, 98, 99
 device, 103
 enableValidationLayers, 112
 endSingleTimeCommands, 104
 findMemoryType, 104
 findPhysicalQueueFamilies, 104
 findQueueFamilies, 105
 findSupportedFormat, 105
 getCommandPool, 106
 getGraphicsQueue, 106
 getInstance, 106
 getPhysicalDevice, 106
 getProperties, 107
 getRequiredExtensions, 107
 getSwapChainSupport, 107
 graphicsQueue, 108
 hasGlfwRequiredInstanceExtensions, 108
 isDeviceSuitable, 108
 m_commandPool, 112
 m_debugMessenger, 112
 m_device, 112

m_deviceExtensions, 112
m_graphicsQueue, 112
m_instance, 112
m_physicalDevice, 113
m_presentQueue, 113
m_properties, 113
m_surface, 113
m_validationLayers, 113
m_window, 113
operator=, 109
pickPhysicalDevice, 109
populateDebugMessengerCreateInfo, 109
presentQueue, 110
querySwapChainSupport, 110
setupDebugMessenger, 110
surface, 111
transitionImageLayout, 111
ven::Engine, 114
~Engine, 116
cleanup, 117
Engine, 116, 117
loadObjects, 117
m_device, 119
m_framePools, 119
m_globalPool, 119
m_gui, 119
m_renderer, 119
m_sceneManager, 120
m_state, 120
m_window, 120
mainLoop, 118
operator=, 119
ven::EventManager, 120
~EventManager, 122
EventManager, 122
handleEvents, 122
isKeyJustPressed, 122
m_keyState, 124
moveCamera, 123
operator=, 123
processKeyActions, 123
updateEngineState, 123
ven::FrameInfo, 124
camera, 126
commandBuffer, 126
frameDescriptorPool, 126
frameIndex, 126
frameTime, 126
globalDescriptorSet, 126
lights, 127
objects, 127
ven::GlobalUbo, 129
ambientLightColor, 130
inverseView, 130
numLights, 130
pointLights, 130
projection, 130
view, 130
ven::Gui, 131
~Gui, 133
cameraSection, 133
cleanup, 134
devicePropertiesSection, 134
getLightsToRemove, 135
getObjectsToRemove, 135
getState, 135
Gui, 133
init, 135
initStyle, 136
inputsSection, 137
lightsSection, 137
m_intensity, 140
m_io, 140
m_lightsToRemove, 140
m_objectsToRemove, 140
m_shininess, 140
m_state, 141
objectsSection, 137
operator=, 138
render, 138
rendererSection, 138
renderFrameWindow, 139
setState, 139
ven::Gui::ClockData, 69
deltaTimeMS, 69
fps, 69
ven::Gui::funcs, 127
IsLegacyNativeDupe, 128
ven::KeyAction, 142
dir, 143
key, 143
value, 143
ven::KeyMappings, 144
lookDown, 145
lookLeft, 145
lookRight, 145
lookUp, 145
moveBackward, 145
moveDown, 145
moveForward, 145
moveLeft, 146
moveRight, 146
moveUp, 146
toggleGui, 146
ven::Light, 147
~Light, 148
color, 150
getId, 149
getName, 149
getShininess, 149
Light, 148, 149
m_lightId, 150
m_name, 150
m_shininess, 150
Map, 148
operator=, 149

setName, 150
 setShininess, 150
 transform, 151
 ven::LightFactory, 151
 create, 152
 duplicate, 152
 m_currentLightId, 153
 ven::LightPushConstantData, 153
 color, 154
 position, 154
 radius, 154
 ven::Logger, 155
 formatLogMessage, 156
 getColorForDuration, 156
 getInstance, 156
 LOG_LEVEL_COLOR, 158
 LOG_LEVEL_STRING, 158
 logExecutionTime, 157
 Logger, 156
 logWarning, 158
 ven::Model, 159
 ~Model, 161
 bind, 162
 createIndexBuffer, 162
 createVertexBuffer, 162
 draw, 163
 m_device, 164
 m_hasIndexBuffer, 164
 m_indexBuffer, 164
 m_indexCount, 164
 m_vertexBuffer, 164
 m_vertexCount, 164
 Model, 161, 162
 operator=, 163
 ven::Model::Builder, 51
 indices, 53
 loadModel, 52
 processMesh, 52
 processNode, 52
 vertices, 53
 ven::Model::Vertex, 240
 color, 242
 getAttributeDescriptions, 241
 getBindingDescriptions, 241
 normal, 242
 operator==, 242
 position, 242
 uv, 243
 ven::ModelFactory, 165
 create, 165
 loadAll, 165
 ven::Object, 166
 ~Object, 168
 getBufferInfo, 169
 getDiffuseMap, 169
 getId, 169
 getModel, 170
 getName, 170
 m_bufferInfo, 171
 m_diffuseMap, 171
 m_model, 172
 m_name, 172
 m_objId, 172
 Map, 168
 Object, 168, 169
 operator=, 170
 setBufferInfo, 171
 setDiffuseMap, 171
 setModel, 171
 setName, 171
 transform, 172
 ven::ObjectBufferData, 173
 modelMatrix, 173
 normalMatrix, 173
 ven::ObjectFactory, 174
 create, 174
 duplicate, 175
 m_currentObjId, 176
 ven::ObjectPushConstantData, 176
 modelMatrix, 176
 normalMatrix, 176
 ven::ObjectRenderSystem, 177
 ObjectRenderSystem, 179
 operator=, 180
 render, 180
 ven::PipelineConfigInfo, 181
 attributeDescriptions, 182
 bindingDescriptions, 182
 colorBlendAttachment, 183
 colorBlendInfo, 183
 depthStencilInfo, 183
 dynamicStateEnables, 183
 dynamicStateInfo, 183
 inputAssemblyInfo, 183
 multisampleInfo, 184
 operator=, 182
 PipelineConfigInfo, 182
 pipelineLayout, 184
 rasterizationInfo, 184
 renderPass, 184
 subpass, 184
 ven::PointLightData, 185
 color, 185
 padding, 185
 position, 185
 shininess, 186
 ven::PointLightRenderSystem, 186
 operator=, 189
 PointLightRenderSystem, 188
 render, 189
 ven::QueueFamilyIndices, 190
 graphicsFamily, 191
 graphicsFamilyHasValue, 191
 isComplete, 190
 presentFamily, 191
 presentFamilyHasValue, 191

ven::Renderer, 192
~Renderer, 194
beginFrame, 195
beginSwapChainRenderPass, 195
createCommandBuffers, 195
endFrame, 195
endSwapChainRenderPass, 196
freeCommandBuffers, 196
getAspectRatio, 196
getClearColor, 196
getCurrentCommandBuffer, 197
getFrameIndex, 197
getSwapChainRenderPass, 198
getWindow, 198
isFrameInProgress, 198
m_clearValues, 200
m_commandBuffers, 200
m_currentFrameIndex, 200
m_currentImageIndex, 200
m_device, 201
m_isFrameStarted, 201
m_swapChain, 201
m_window, 201
operator=, 199
recreateSwapChain, 199
Renderer, 194
setClearValue, 199
ven::SceneManager, 202
addLight, 204
addObject, 204
destroyEntity, 204
destroyLight, 205
destroyObject, 205
getBufferInfoForObject, 205
getDestroyState, 205
getLights, 205
getObjects, 206
getTextureDefault, 206
getUboBuffers, 206
m_currentLightId, 208
m_currentObjId, 208
m_destroyState, 208
m_lights, 208
m_objects, 208
m_textureDefault, 208
m_uboBuffers, 208
operator=, 207
SceneManager, 203, 204
setDestroyState, 207
updateBuffer, 207
ven::Shaders, 209
~Shaders, 211
bind, 212
createGraphicsPipeline, 212
createShaderModule, 213
defaultPipelineConfigInfo, 213
m_device, 214
m_fragShaderModule, 214
m_graphicsPipeline, 214
m_vertShaderModule, 214
operator=, 214
readFile, 214
Shaders, 211, 212
ven::SwapChain, 215
~SwapChain, 218
acquireNextImage, 219
chooseSwapExtent, 219
chooseSwapPresentMode, 219
chooseSwapSurfaceFormat, 219
compareSwapFormats, 220
createDepthResources, 220
createFrameBuffers, 220
createImageViews, 220
createRenderPass, 220
createSwapChain, 220
createSyncObjects, 220
extentAspectRatio, 221
findDepthFormat, 221
getFrameBuffer, 221
getImageView, 221
getRenderPass, 221
getSwapChainExtent, 221
getSwapChainImageFormat, 222
height, 222
imageCount, 222
init, 222
m_currentFrame, 223
m_depthImageMemory, 223
m_depthImages, 223
m_depthImageViews, 223
m_device, 224
m_imageAvailableSemaphores, 224
m_imagesInFlight, 224
m_inFlightFences, 224
m_oldSwapChain, 224
m_renderFinishedSemaphores, 224
m_renderPass, 225
m_swapChain, 225
m_swapChainDepthFormat, 225
m_swapChainExtent, 225
m_swapChainFrameBuffers, 225
m_swapChainImageFormat, 225
m_swapChainImages, 226
m_swapChainImageViews, 226
m_windowExtent, 226
operator=, 222
submitCommandBuffers, 223
SwapChain, 218, 219
width, 223
ven::SwapChainSupportDetails, 226
capabilities, 227
formats, 227
presentModes, 227
ven::Texture, 228
~Texture, 231
createTextureImage, 231

createTextureImageView, 231
 createTextureSampler, 232
 getExtent, 232
 getFormat, 232
 getImage, 233
 getImageInfo, 233
 getImageLayout, 233
 getImageView, 233
 imageView, 233
 m_descriptor, 234
 m_device, 234
 m_extent, 235
 m_format, 235
 m_layerCount, 235
 m_mipLevels, 235
 m_textureImage, 235
 m_textureImageMemory, 235
 m_textureImageView, 235
 m_textureLayout, 236
 m_textureSampler, 236
 operator=, 233
 sampler, 234
 Texture, 230, 231
 transitionLayout, 234
 updateDescriptor, 234
 ven::TextureFactory, 236
 create, 237
 loadAll, 237
 ven::Transform3D, 238
 normalMatrix, 239
 rotation, 240
 scale, 240
 transformMatrix, 239
 translation, 240
 ven::Window, 243
 ~Window, 244
 createWindow, 245
 createWindowSurface, 245
 framebufferResizeCallback, 245
 getExtent, 246
 getGLFWWindow, 246
 m_framebufferResized, 248
 m_height, 248
 m_width, 248
 m_window, 248
 operator=, 247
 resetWindowResizedFlag, 247
 setFullscreen, 247
 wasWindowResized, 247
 Window, 244
 vengine, 1
 vertices
 ven::Model::Builder, 53
 view
 ven::GlobalUbo, 130

WARNING

ven, 16

wasWindowResized

ven::Window, 247
 WHITE_3
 ven::Colors, 81
 WHITE_4
 ven::Colors, 81
 WHITE_V
 ven::Colors, 81
 width
 ven::SwapChain, 223
 Window
 ven::Window, 244
 Window.hpp
 GLFW_INCLUDE_VULKAN, 268
 writeBuffer
 ven::DescriptorWriter, 94
 writeImage
 ven::DescriptorWriter, 94
 writeToBuffer
 ven::Buffer, 39
 writeToIndex
 ven::Buffer, 39
 YELLOW_3
 ven::Colors, 81
 YELLOW_4
 ven::Colors, 81
 YELLOW_V
 ven::Colors, 81