

# ChArUco Board を使ったカメラキャリブレーション

0.0.1

構築: Doxygen 1.9.5



---

<b>1 ゲーム グラフィックス特論の宿題用補助プログラム <b>GLFW3</b> 版.</b>	<b>1</b>
<b>2 Calib</b>	<b>3</b>
2.1 libs ディレクトリについて . . . . .	3
<b>3 名前空間索引</b>	<b>5</b>
3.1 名前空間一覧 . . . . .	5
<b>4 階層索引</b>	<b>7</b>
4.1 クラス階層 . . . . .	7
<b>5 クラス索引</b>	<b>9</b>
5.1 クラス一覧 . . . . .	9
<b>6 ファイル索引</b>	<b>11</b>
6.1 ファイル一覧 . . . . .	11
<b>7 名前空間詳解</b>	<b>13</b>
7.1 gg 名前空間 . . . . .	13
7.1.1 詳解 . . . . .	16
7.1.2 列挙型詳解 . . . . .	16
7.1.2.1 BindingPoints . . . . .	16
7.1.3 関数詳解 . . . . .	16
7.1.3.1 ggError() . . . . .	16
7.1.3.2 ggFBOError() . . . . .	17
7.1.3.3 ggArraysObj() . . . . .	17
7.1.3.4 ggConjugate() . . . . .	18
7.1.3.5 ggCreateComputeShader() . . . . .	19
7.1.3.6 ggCreateNormalMap() . . . . .	19
7.1.3.7 ggCreateShader() . . . . .	20
7.1.3.8 ggCross() [1/2] . . . . .	21
7.1.3.9 ggCross() [2/2] . . . . .	22
7.1.3.10 ggDistance3() [1/2] . . . . .	22
7.1.3.11 ggDistance3() [2/2] . . . . .	23
7.1.3.12 ggDistance4() [1/2] . . . . .	24
7.1.3.13 ggDistance4() [2/2] . . . . .	25
7.1.3.14 ggDot3() [1/2] . . . . .	26
7.1.3.15 ggDot3() [2/2] . . . . .	27
7.1.3.16 ggDot4() [1/2] . . . . .	28
7.1.3.17 ggDot4() [2/2] . . . . .	28
7.1.3.18 ggElementsMesh() . . . . .	29
7.1.3.19 ggElementsObj() . . . . .	30
7.1.3.20 ggElementsSphere() . . . . .	31
7.1.3.21 ggEllipse() . . . . .	32

---

7.1.3.22 ggEulerQuaternion() [1/2] . . . . .	32
7.1.3.23 ggEulerQuaternion() [2/2] . . . . .	33
7.1.3.24 ggFrustum() . . . . .	34
7.1.3.25 ggIdentity() . . . . .	34
7.1.3.26 ggIdentityQuaternion() . . . . .	35
7.1.3.27 ggInit() . . . . .	36
7.1.3.28 ggInvert() [1/2] . . . . .	36
7.1.3.29 ggInvert() [2/2] . . . . .	37
7.1.3.30 ggLength3() [1/2] . . . . .	37
7.1.3.31 ggLength3() [2/2] . . . . .	38
7.1.3.32 ggLength4() [1/2] . . . . .	39
7.1.3.33 ggLength4() [2/2] . . . . .	40
7.1.3.34 ggLoadComputeShader() . . . . .	40
7.1.3.35 ggLoadHeight() . . . . .	41
7.1.3.36 ggLoadImage() . . . . .	42
7.1.3.37 ggLoadShader() [1/2] . . . . .	43
7.1.3.38 ggLoadShader() [2/2] . . . . .	43
7.1.3.39 ggLoadSimpleObj() [1/2] . . . . .	44
7.1.3.40 ggLoadSimpleObj() [2/2] . . . . .	45
7.1.3.41 ggLoadTexture() . . . . .	46
7.1.3.42 ggLookat() [1/3] . . . . .	47
7.1.3.43 ggLookat() [2/3] . . . . .	47
7.1.3.44 ggLookat() [3/3] . . . . .	48
7.1.3.45 ggMatrixQuaternion() [1/2] . . . . .	49
7.1.3.46 ggMatrixQuaternion() [2/2] . . . . .	49
7.1.3.47 ggNorm() . . . . .	50
7.1.3.48 ggNormal() . . . . .	51
7.1.3.49 ggNormalize() . . . . .	51
7.1.3.50 ggNormalize3() [1/4] . . . . .	52
7.1.3.51 ggNormalize3() [2/4] . . . . .	53
7.1.3.52 ggNormalize3() [3/4] . . . . .	53
7.1.3.53 ggNormalize3() [4/4] . . . . .	54
7.1.3.54 ggNormalize4() [1/4] . . . . .	55
7.1.3.55 ggNormalize4() [2/4] . . . . .	55
7.1.3.56 ggNormalize4() [3/4] . . . . .	56
7.1.3.57 ggNormalize4() [4/4] . . . . .	57
7.1.3.58 ggOrthogonal() . . . . .	57
7.1.3.59 ggPerspective() . . . . .	58
7.1.3.60 ggPointsCube() . . . . .	59
7.1.3.61 ggPointsSphere() . . . . .	59
7.1.3.62 ggQuaternion() [1/2] . . . . .	60
7.1.3.63 ggQuaternion() [2/2] . . . . .	60

---

7.1.3.64 ggQuaternionMatrix()	61
7.1.3.65 ggQuaternionTransposeMatrix()	62
7.1.3.66 ggReadImage()	63
7.1.3.67 ggRectangle()	64
7.1.3.68 ggRotate() [1/5]	64
7.1.3.69 ggRotate() [2/5]	65
7.1.3.70 ggRotate() [3/5]	66
7.1.3.71 ggRotate() [4/5]	66
7.1.3.72 ggRotate() [5/5]	67
7.1.3.73 ggRotateQuaternion() [1/3]	68
7.1.3.74 ggRotateQuaternion() [2/3]	68
7.1.3.75 ggRotateQuaternion() [3/3]	69
7.1.3.76 ggRotateX()	70
7.1.3.77 ggRotateY()	71
7.1.3.78 ggRotateZ()	72
7.1.3.79 ggSaveColor()	72
7.1.3.80 ggSaveDepth()	73
7.1.3.81 ggSaveTga()	73
7.1.3.82 ggScale() [1/3]	74
7.1.3.83 ggScale() [2/3]	75
7.1.3.84 ggScale() [3/3]	76
7.1.3.85 ggSlerp() [1/4]	76
7.1.3.86 ggSlerp() [2/4]	77
7.1.3.87 ggSlerp() [3/4]	78
7.1.3.88 ggSlerp() [4/4]	79
7.1.3.89 ggTranslate() [1/3]	80
7.1.3.90 ggTranslate() [2/3]	81
7.1.3.91 ggTranslate() [3/3]	82
7.1.3.92 ggTranspose()	83
7.1.3.93 operator*()	84
7.1.3.94 operator+() [1/2]	84
7.1.3.95 operator+() [2/2]	85
7.1.3.96 operator-() [1/2]	85
7.1.3.97 operator-() [2/2]	86
7.1.3.98 operator/()	86
7.1.4 変数詳解	87
7.1.4.1 ggBufferAlignment	87
<b>8 クラス詳解</b>	<b>89</b>
8.1 Buffer クラス	89
8.1.1 詳解	90
8.1.2 構築子と解体子	90

---

8.1.2.1 Buffer() [1/4] . . . . .	90
8.1.2.2 Buffer() [2/4] . . . . .	90
8.1.2.3 Buffer() [3/4] . . . . .	91
8.1.2.4 Buffer() [4/4] . . . . .	92
8.1.2.5 ~Buffer() . . . . .	92
8.1.3 関数詳解 . . . . .	92
8.1.3.1 bindBuffer() . . . . .	92
8.1.3.2 channelsToFormat() . . . . .	93
8.1.3.3 copy() . . . . .	94
8.1.3.4 copyBuffer() . . . . .	95
8.1.3.5 create() . . . . .	95
8.1.3.6 discard() . . . . .	96
8.1.3.7 formatToChannels() . . . . .	97
8.1.3.8 getAspect() . . . . .	97
8.1.3.9 getBufferName() . . . . .	98
8.1.3.10 getChannels() . . . . .	98
8.1.3.11 getData() . . . . .	98
8.1.3.12 getFormat() . . . . .	99
8.1.3.13 getHeight() . . . . .	100
8.1.3.14 getSize() . . . . .	100
8.1.3.15 getWidth() . . . . .	101
8.1.3.16 map() . . . . .	102
8.1.3.17 operator=() [1/2] . . . . .	102
8.1.3.18 operator=() [2/2] . . . . .	103
8.1.3.19 resize() [1/2] . . . . .	104
8.1.3.20 resize() [2/2] . . . . .	104
8.1.3.21 setData() . . . . .	105
8.1.3.22 unbindBuffer() . . . . .	105
8.1.3.23 unmap() . . . . .	106
8.2 Calibration クラス . . . . .	106
8.2.1 詳解 . . . . .	107
8.2.2 構築子と解体子 . . . . .	107
8.2.2.1 Calibration() [1/2] . . . . .	107
8.2.2.2 Calibration() [2/2] . . . . .	107
8.2.2.3 ~Calibration() . . . . .	108
8.2.3 関数詳解 . . . . .	108
8.2.3.1 calibrate() . . . . .	108
8.2.3.2 createBoard() . . . . .	108
8.2.3.3 detect() . . . . .	109
8.2.3.4 discardCorners() . . . . .	110
8.2.3.5 drawBoard() . . . . .	110
8.2.3.6 finished() . . . . .	111

---

8.2.3.7 getAllMarkerPoses()	111
8.2.3.8 getCameraMatrix()	112
8.2.3.9 getCornerCount()	112
8.2.3.10 getCornersCount()	113
8.2.3.11 getDistortionCoefficients()	113
8.2.3.12 getReprojectionError()	114
8.2.3.13 loadParameters()	114
8.2.3.14 operator=( )	115
8.2.3.15 recordCorners()	115
8.2.3.16 saveParameters()	116
8.2.3.17 setDictionary()	117
8.2.4 メンバ詳解	118
8.2.4.1 dictionaryList	118
8.3 CamCv クラス	119
8.3.1 詳解	120
8.3.2 構築子と解体子	120
8.3.2.1 CamCv()	120
8.3.2.2 ~CamCv()	120
8.3.3 関数詳解	120
8.3.3.1 close()	120
8.3.3.2 decreaseExposure()	121
8.3.3.3 decreaseGain()	121
8.3.3.4 getCodec() [1/2]	122
8.3.3.5 getCodec() [2/2]	122
8.3.3.6 getFps()	123
8.3.3.7 getPosition()	123
8.3.3.8 increaseExposure()	123
8.3.3.9 increaseGain()	124
8.3.3.10 open() [1/2]	124
8.3.3.11 open() [2/2]	125
8.3.3.12 setExposure()	125
8.3.3.13 setGain()	126
8.3.3.14 setPosition()	126
8.4 Camera クラス	127
8.4.1 詳解	128
8.4.2 構築子と解体子	128
8.4.2.1 Camera() [1/2]	129
8.4.2.2 Camera() [2/2]	129
8.4.2.3 ~Camera()	129
8.4.3 関数詳解	129
8.4.3.1 capture()	130
8.4.3.2 close()	130

---

8.4.3.3 copyFrame()	131
8.4.3.4 decreaseExposure()	131
8.4.3.5 decreaseGain()	131
8.4.3.6 getChannels()	132
8.4.3.7 getFps()	132
8.4.3.8 getFrames()	132
8.4.3.9 getHeight()	133
8.4.3.10 getSize()	133
8.4.3.11 getWidth()	133
8.4.3.12 increaseExposure()	133
8.4.3.13 increaseGain()	134
8.4.3.14 isRunning()	134
8.4.3.15 operator=()	134
8.4.3.16 start()	134
8.4.3.17 stop()	135
8.4.3.18 transmit() [1/2]	135
8.4.3.19 transmit() [2/2]	136
8.4.4 メンバ詳解	136
8.4.4.1 captured	136
8.4.4.2 channels	136
8.4.4.3 frame	136
8.4.4.4 in	137
8.4.4.5 interval	137
8.4.4.6 mtx	137
8.4.4.7 out	137
8.4.4.8 pixels	137
8.4.4.9 running	138
8.4.4.10 thr	138
8.4.4.11 total	138
8.5 CamImage クラス	138
8.5.1 詳解	139
8.5.2 構築子と解体子	139
8.5.2.1 CamImage() [1/2]	139
8.5.2.2 CamImage() [2/2]	140
8.5.2.3 ~CamImage()	140
8.5.3 関数詳解	140
8.5.3.1 isOpened()	140
8.5.3.2 load()	140
8.5.3.3 open()	141
8.5.3.4 save()	142
8.6 Capture クラス	143
8.6.1 詳解	144

---

8.6.2 構築子と解体子	144
8.6.2.1 Capture() [1/2]	144
8.6.2.2 Capture() [2/2]	144
8.6.2.3 ~Capture()	145
8.6.3 関数詳解	145
8.6.3.1 close()	145
8.6.3.2 getFps()	146
8.6.3.3 getSize()	146
8.6.3.4 isOpened()	146
8.6.3.5 openDevice()	146
8.6.3.6 openImage()	147
8.6.3.7 openMovie()	148
8.6.3.8 operator bool()	148
8.6.3.9 retrieve()	148
8.6.3.10 start()	149
8.6.3.11 stop()	150
8.7 Config クラス	150
8.7.1 詳解	151
8.7.2 構築子と解体子	151
8.7.2.1 Config()	151
8.7.2.2 ~Config()	151
8.7.3 関数詳解	152
8.7.3.1 getCheckerLength()	152
8.7.3.2 getDeviceCount()	152
8.7.3.3 getDeviceList()	152
8.7.3.4 getDeviceName()	153
8.7.3.5 getDictionaryName()	154
8.7.3.6 getHeight()	154
8.7.3.7 getInitialImage()	154
8.7.3.8 getTitle()	155
8.7.3.9 getWidth()	155
8.7.3.10 initialize()	155
8.7.3.11 load()	155
8.7.3.12 save()	156
8.7.4 フレンドと関連関数の詳解	157
8.7.4.1 Menu	157
8.8 Expand クラス	158
8.8.1 詳解	158
8.8.2 構築子と解体子	158
8.8.2.1 Expand() [1/2]	158
8.8.2.2 Expand() [2/2]	158
8.8.2.3 ~Expand()	160

---

8.8.3 関数詳解 . . . . .	160
8.8.3.1 getProgram() . . . . .	160
8.8.3.2 operator=() . . . . .	160
8.8.3.3 setup() . . . . .	161
8.9 Framebuffer クラス . . . . .	162
8.9.1 詳解 . . . . .	163
8.9.2 構築子と解体子 . . . . .	163
8.9.2.1 Framebuffer() [1/4] . . . . .	163
8.9.2.2 Framebuffer() [2/4] . . . . .	163
8.9.2.3 Framebuffer() [3/4] . . . . .	164
8.9.2.4 Framebuffer() [4/4] . . . . .	165
8.9.2.5 ~Framebuffer() . . . . .	166
8.9.3 関数詳解 . . . . .	166
8.9.3.1 bindFramebuffer() . . . . .	166
8.9.3.2 copy() . . . . .	167
8.9.3.3 create() . . . . .	168
8.9.3.4 discard() . . . . .	169
8.9.3.5 draw() . . . . .	169
8.9.3.6 getChannels() . . . . .	170
8.9.3.7 getFramebufferName() . . . . .	170
8.9.3.8 getSize() . . . . .	171
8.9.3.9 operator=() [1/2] . . . . .	171
8.9.3.10 operator=() [2/2] . . . . .	171
8.9.3.11 unbindFramebuffer() . . . . .	172
8.9.3.12 update() [1/2] . . . . .	172
8.9.3.13 update() [2/2] . . . . .	173
8.10 GgApp クラス . . . . .	174
8.10.1 詳解 . . . . .	175
8.10.2 構築子と解体子 . . . . .	175
8.10.2.1 GgApp() [1/2] . . . . .	175
8.10.2.2 GgApp() [2/2] . . . . .	175
8.10.2.3 ~GgApp() . . . . .	175
8.10.3 関数詳解 . . . . .	175
8.10.3.1 main() . . . . .	176
8.10.3.2 operator=() . . . . .	176
8.11 gg::GgBuffer< T > クラステンプレート . . . . .	177
8.11.1 詳解 . . . . .	177
8.11.2 構築子と解体子 . . . . .	177
8.11.2.1 GgBuffer() [1/2] . . . . .	177
8.11.2.2 ~GgBuffer() . . . . .	178
8.11.2.3 GgBuffer() [2/2] . . . . .	178
8.11.3 関数詳解 . . . . .	178

---

8.11.3.1 bind()	178
8.11.3.2 copy()	178
8.11.3.3 getBuffer()	179
8.11.3.4 getCount()	179
8.11.3.5 getStride()	179
8.11.3.6 getTarget()	180
8.11.3.7 map() [1/2]	180
8.11.3.8 map() [2/2]	180
8.11.3.9 operator=()	181
8.11.3.10 read()	181
8.11.3.11 send()	181
8.11.3.12 unbind()	182
8.11.3.13 unmap()	182
8.12 gg::GgColorTexture クラス	182
8.12.1 詳解	182
8.12.2 構築子と解体子	183
8.12.2.1 GgColorTexture() [1/3]	183
8.12.2.2 GgColorTexture() [2/3]	183
8.12.2.3 GgColorTexture() [3/3]	184
8.12.2.4 ~GgColorTexture()	185
8.12.3 関数詳解	185
8.12.3.1 load() [1/2]	185
8.12.3.2 load() [2/2]	186
8.13 gg::GgElements クラス	187
8.13.1 詳解	188
8.13.2 構築子と解体子	188
8.13.2.1 GgElements() [1/2]	188
8.13.2.2 GgElements() [2/2]	188
8.13.2.3 ~GgElements()	189
8.13.3 関数詳解	189
8.13.3.1 draw()	189
8.13.3.2 getIndexBuffer()	190
8.13.3.3 getIndexCount()	190
8.13.3.4 load()	190
8.13.3.5 send()	191
8.14 gg::GgMatrix クラス	191
8.14.1 詳解	194
8.14.2 構築子と解体子	194
8.14.2.1 GgMatrix() [1/4]	194
8.14.2.2 GgMatrix() [2/4]	194
8.14.2.3 GgMatrix() [3/4]	195
8.14.2.4 GgMatrix() [4/4]	195

---

8.14.3 関数詳解 . . . . .	196
8.14.3.1 frustum() . . . . .	196
8.14.3.2 get() [1/2] . . . . .	197
8.14.3.3 get() [2/2] . . . . .	198
8.14.3.4 invert() . . . . .	199
8.14.3.5 loadFrustum() . . . . .	199
8.14.3.6 loadIdentity() . . . . .	200
8.14.3.7 loadInvert() [1/2] . . . . .	201
8.14.3.8 loadInvert() [2/2] . . . . .	201
8.14.3.9 loadLookat() [1/3] . . . . .	202
8.14.3.10 loadLookat() [2/3] . . . . .	203
8.14.3.11 loadLookat() [3/3] . . . . .	203
8.14.3.12 loadNormal() [1/2] . . . . .	204
8.14.3.13 loadNormal() [2/2] . . . . .	205
8.14.3.14 loadOrthogonal() . . . . .	206
8.14.3.15 loadPerspective() . . . . .	206
8.14.3.16 loadRotate() [1/5] . . . . .	207
8.14.3.17 loadRotate() [2/5] . . . . .	208
8.14.3.18 loadRotate() [3/5] . . . . .	208
8.14.3.19 loadRotate() [4/5] . . . . .	209
8.14.3.20 loadRotate() [5/5] . . . . .	210
8.14.3.21 loadRotateX() . . . . .	211
8.14.3.22 loadRotateY() . . . . .	211
8.14.3.23 loadRotateZ() . . . . .	212
8.14.3.24 loadScale() [1/3] . . . . .	212
8.14.3.25 loadScale() [2/3] . . . . .	213
8.14.3.26 loadScale() [3/3] . . . . .	214
8.14.3.27 loadTranslate() [1/3] . . . . .	214
8.14.3.28 loadTranslate() [2/3] . . . . .	215
8.14.3.29 loadTranslate() [3/3] . . . . .	216
8.14.3.30 loadTranspose() [1/2] . . . . .	216
8.14.3.31 loadTranspose() [2/2] . . . . .	217
8.14.3.32 lookat() [1/3] . . . . .	218
8.14.3.33 lookat() [2/3] . . . . .	219
8.14.3.34 lookat() [3/3] . . . . .	219
8.14.3.35 normal() . . . . .	221
8.14.3.36 operator*() [1/3] . . . . .	221
8.14.3.37 operator*() [2/3] . . . . .	222
8.14.3.38 operator*() [3/3] . . . . .	222
8.14.3.39 operator*=(()) [1/2] . . . . .	223
8.14.3.40 operator*=(()) [2/2] . . . . .	224
8.14.3.41 operator+() [1/2] . . . . .	225

---

8.14.3.42 operator+() [2/2] . . . . .	225
8.14.3.43 operator+=() [1/2] . . . . .	226
8.14.3.44 operator+=() [2/2] . . . . .	227
8.14.3.45 operator-() [1/2] . . . . .	227
8.14.3.46 operator-() [2/2] . . . . .	228
8.14.3.47 operator-=() [1/2] . . . . .	228
8.14.3.48 operator-=() [2/2] . . . . .	229
8.14.3.49 operator/() [1/2] . . . . .	230
8.14.3.50 operator/() [2/2] . . . . .	230
8.14.3.51 operator/=() [1/2] . . . . .	231
8.14.3.52 operator/=() [2/2] . . . . .	232
8.14.3.53 operator=() . . . . .	232
8.14.3.54 orthogonal() . . . . .	233
8.14.3.55 perspective() . . . . .	234
8.14.3.56 projection() [1/4] . . . . .	235
8.14.3.57 projection() [2/4] . . . . .	235
8.14.3.58 projection() [3/4] . . . . .	235
8.14.3.59 projection() [4/4] . . . . .	236
8.14.3.60 rotate() [1/5] . . . . .	236
8.14.3.61 rotate() [2/5] . . . . .	237
8.14.3.62 rotate() [3/5] . . . . .	237
8.14.3.63 rotate() [4/5] . . . . .	238
8.14.3.64 rotate() [5/5] . . . . .	239
8.14.3.65 rotateX() . . . . .	240
8.14.3.66 rotateY() . . . . .	240
8.14.3.67 rotateZ() . . . . .	241
8.14.3.68 scale() [1/3] . . . . .	242
8.14.3.69 scale() [2/3] . . . . .	243
8.14.3.70 scale() [3/3] . . . . .	243
8.14.3.71 translate() [1/3] . . . . .	244
8.14.3.72 translate() [2/3] . . . . .	245
8.14.3.73 translate() [3/3] . . . . .	246
8.14.3.74 transpose() . . . . .	247
8.15 gg::GgNormalTexture クラス . . . . .	247
8.15.1 詳解 . . . . .	248
8.15.2 構築子と解体子 . . . . .	248
8.15.2.1 GgNormalTexture() [1/3] . . . . .	248
8.15.2.2 GgNormalTexture() [2/3] . . . . .	248
8.15.2.3 GgNormalTexture() [3/3] . . . . .	249
8.15.2.4 ~GgNormalTexture() . . . . .	249
8.15.3 関数詳解 . . . . .	249
8.15.3.1 load() [1/2] . . . . .	250

---

8.15.3.2 load() [2/2] . . . . .	250
8.16 gg::GgPoints クラス . . . . .	251
8.16.1 詳解 . . . . .	252
8.16.2 構築子と解体子 . . . . .	252
8.16.2.1 GgPoints() [1/2] . . . . .	252
8.16.2.2 GgPoints() [2/2] . . . . .	253
8.16.2.3 ~GgPoints() . . . . .	253
8.16.3 関数詳解 . . . . .	253
8.16.3.1 draw() . . . . .	253
8.16.3.2 getBuffer() . . . . .	254
8.16.3.3 getCount() . . . . .	254
8.16.3.4 load() . . . . .	254
8.16.3.5 operator bool() . . . . .	255
8.16.3.6 operator"!"() . . . . .	255
8.16.3.7 send() . . . . .	255
8.17 gg::GgPointShader クラス . . . . .	256
8.17.1 詳解 . . . . .	257
8.17.2 構築子と解体子 . . . . .	257
8.17.2.1 GgPointShader() [1/3] . . . . .	257
8.17.2.2 GgPointShader() [2/3] . . . . .	257
8.17.2.3 GgPointShader() [3/3] . . . . .	257
8.17.2.4 ~GgPointShader() . . . . .	258
8.17.3 関数詳解 . . . . .	258
8.17.3.1 get() . . . . .	258
8.17.3.2 load() [1/2] . . . . .	258
8.17.3.3 load() [2/2] . . . . .	259
8.17.3.4 loadMatrix() [1/2] . . . . .	260
8.17.3.5 loadMatrix() [2/2] . . . . .	260
8.17.3.6 loadModelviewMatrix() [1/2] . . . . .	261
8.17.3.7 loadModelviewMatrix() [2/2] . . . . .	261
8.17.3.8 loadProjectionMatrix() [1/2] . . . . .	262
8.17.3.9 loadProjectionMatrix() [2/2] . . . . .	262
8.17.3.10 unuse() . . . . .	262
8.17.3.11 use() [1/5] . . . . .	263
8.17.3.12 use() [2/5] . . . . .	263
8.17.3.13 use() [3/5] . . . . .	263
8.17.3.14 use() [4/5] . . . . .	264
8.17.3.15 use() [5/5] . . . . .	264
8.18 gg::GgQuaternion クラス . . . . .	265
8.18.1 詳解 . . . . .	268
8.18.2 構築子と解体子 . . . . .	268
8.18.2.1 GgQuaternion() [1/5] . . . . .	268

---

8.18.2.2 GgQuaternion()	[2/5]	268
8.18.2.3 GgQuaternion()	[3/5]	268
8.18.2.4 GgQuaternion()	[4/5]	269
8.18.2.5 GgQuaternion()	[5/5]	269
8.18.3 関数詳解		269
8.18.3.1 add()	[1/4]	269
8.18.3.2 add()	[2/4]	270
8.18.3.3 add()	[3/4]	271
8.18.3.4 add()	[4/4]	271
8.18.3.5 conjugate()		272
8.18.3.6 divide()	[1/4]	273
8.18.3.7 divide()	[2/4]	273
8.18.3.8 divide()	[3/4]	274
8.18.3.9 divide()	[4/4]	275
8.18.3.10 euler()	[1/2]	276
8.18.3.11 euler()	[2/2]	276
8.18.3.12 get()		277
8.18.3.13 getConjugateMatrix()	[1/3]	277
8.18.3.14 getConjugateMatrix()	[2/3]	278
8.18.3.15 getConjugateMatrix()	[3/3]	279
8.18.3.16 getMatrix()	[1/3]	279
8.18.3.17 getMatrix()	[2/3]	280
8.18.3.18 getMatrix()	[3/3]	281
8.18.3.19 invert()		281
8.18.3.20 load()	[1/4]	282
8.18.3.21 load()	[2/4]	283
8.18.3.22 load()	[3/4]	283
8.18.3.23 load()	[4/4]	284
8.18.3.24 loadAdd()	[1/4]	284
8.18.3.25 loadAdd()	[2/4]	285
8.18.3.26 loadAdd()	[3/4]	286
8.18.3.27 loadAdd()	[4/4]	286
8.18.3.28 loadConjugate()	[1/2]	287
8.18.3.29 loadConjugate()	[2/2]	288
8.18.3.30 loadDivide()	[1/4]	288
8.18.3.31 loadDivide()	[2/4]	289
8.18.3.32 loadDivide()	[3/4]	289
8.18.3.33 loadDivide()	[4/4]	290
8.18.3.34 loadEuler()	[1/2]	291
8.18.3.35 loadEuler()	[2/2]	292
8.18.3.36 loadIdentity()		293
8.18.3.37 loadInvert()	[1/2]	293

---

8.18.3.38 loadInvert() [2/2] . . . . .	294
8.18.3.39 loadMatrix() [1/2] . . . . .	295
8.18.3.40 loadMatrix() [2/2] . . . . .	295
8.18.3.41 loadMultiply() [1/4] . . . . .	296
8.18.3.42 loadMultiply() [2/4] . . . . .	297
8.18.3.43 loadMultiply() [3/4] . . . . .	297
8.18.3.44 loadMultiply() [4/4] . . . . .	298
8.18.3.45 loadNormalize() [1/2] . . . . .	299
8.18.3.46 loadNormalize() [2/2] . . . . .	299
8.18.3.47 loadRotate() [1/3] . . . . .	300
8.18.3.48 loadRotate() [2/3] . . . . .	301
8.18.3.49 loadRotate() [3/3] . . . . .	301
8.18.3.50 loadRotateX() . . . . .	302
8.18.3.51 loadRotateY() . . . . .	303
8.18.3.52 loadRotateZ() . . . . .	303
8.18.3.53 loadSlerp() [1/4] . . . . .	304
8.18.3.54 loadSlerp() [2/4] . . . . .	305
8.18.3.55 loadSlerp() [3/4] . . . . .	306
8.18.3.56 loadSlerp() [4/4] . . . . .	306
8.18.3.57 loadSubtract() [1/4] . . . . .	307
8.18.3.58 loadSubtract() [2/4] . . . . .	308
8.18.3.59 loadSubtract() [3/4] . . . . .	308
8.18.3.60 loadSubtract() [4/4] . . . . .	309
8.18.3.61 multiply() [1/4] . . . . .	310
8.18.3.62 multiply() [2/4] . . . . .	310
8.18.3.63 multiply() [3/4] . . . . .	310
8.18.3.64 multiply() [4/4] . . . . .	311
8.18.3.65 norm() . . . . .	311
8.18.3.66 normalize() . . . . .	312
8.18.3.67 operator*() [1/3] . . . . .	313
8.18.3.68 operator*() [2/3] . . . . .	313
8.18.3.69 operator*() [3/3] . . . . .	313
8.18.3.70 operator*=( ) [1/3] . . . . .	314
8.18.3.71 operator*=( ) [2/3] . . . . .	314
8.18.3.72 operator*=( ) [3/3] . . . . .	314
8.18.3.73 operator+() [1/3] . . . . .	315
8.18.3.74 operator+() [2/3] . . . . .	315
8.18.3.75 operator+() [3/3] . . . . .	316
8.18.3.76 operator+=( ) [1/3] . . . . .	316
8.18.3.77 operator+=( ) [2/3] . . . . .	317
8.18.3.78 operator+=( ) [3/3] . . . . .	317
8.18.3.79 operator-( ) [1/3] . . . . .	317

---

8.18.3.80 operator-() [2/3] . . . . .	318
8.18.3.81 operator-() [3/3] . . . . .	318
8.18.3.82 operator-=() [1/3] . . . . .	319
8.18.3.83 operator-=() [2/3] . . . . .	319
8.18.3.84 operator-=() [3/3] . . . . .	319
8.18.3.85 operator/() [1/3] . . . . .	320
8.18.3.86 operator/() [2/3] . . . . .	320
8.18.3.87 operator/() [3/3] . . . . .	321
8.18.3.88 operator/=(()) [1/3] . . . . .	321
8.18.3.89 operator/=(()) [2/3] . . . . .	322
8.18.3.90 operator/=(()) [3/3] . . . . .	322
8.18.3.91 operator=() [1/2] . . . . .	323
8.18.3.92 operator=() [2/2] . . . . .	323
8.18.3.93 rotate() [1/3] . . . . .	323
8.18.3.94 rotate() [2/3] . . . . .	324
8.18.3.95 rotate() [3/3] . . . . .	325
8.18.3.96 rotateX() . . . . .	326
8.18.3.97 rotateY() . . . . .	326
8.18.3.98 rotateZ() . . . . .	327
8.18.3.99 slerp() [1/2] . . . . .	327
8.18.3.100 slerp() [2/2] . . . . .	328
8.18.3.101 subtract() [1/4] . . . . .	328
8.18.3.102 subtract() [2/4] . . . . .	329
8.18.3.103 subtract() [3/4] . . . . .	329
8.18.3.104 subtract() [4/4] . . . . .	330
8.19 gg::GgShader クラス . . . . .	331
8.19.1 詳解 . . . . .	331
8.19.2 構築子と解体子 . . . . .	332
8.19.2.1 GgShader() [1/3] . . . . .	332
8.19.2.2 GgShader() [2/3] . . . . .	332
8.19.2.3 GgShader() [3/3] . . . . .	333
8.19.2.4 ~GgShader() . . . . .	333
8.19.3 関数詳解 . . . . .	333
8.19.3.1 get() . . . . .	333
8.19.3.2 operator=() . . . . .	333
8.19.3.3 unuse() . . . . .	334
8.19.3.4 use() . . . . .	334
8.20 gg::GgShape クラス . . . . .	334
8.20.1 詳解 . . . . .	335
8.20.2 構築子と解体子 . . . . .	335
8.20.2.1 GgShape() . . . . .	335
8.20.2.2 ~GgShape() . . . . .	335

---

8.20.3 関数詳解 . . . . .	336
8.20.3.1 draw() . . . . .	336
8.20.3.2 get() . . . . .	336
8.20.3.3 getMode() . . . . .	337
8.20.3.4 operator bool() . . . . .	337
8.20.3.5 operator"!"() . . . . .	338
8.20.3.6 setMode() . . . . .	338
8.21 gg::GgSimpleObj クラス . . . . .	338
8.21.1 詳解 . . . . .	339
8.21.2 構築子と解体子 . . . . .	339
8.21.2.1 GgSimpleObj() . . . . .	339
8.21.2.2 ~GgSimpleObj() . . . . .	339
8.21.3 関数詳解 . . . . .	340
8.21.3.1 draw() . . . . .	340
8.21.3.2 get() . . . . .	340
8.21.3.3 operator bool() . . . . .	341
8.21.3.4 operator"!"() . . . . .	341
8.22 gg::GgSimpleShader クラス . . . . .	341
8.22.1 詳解 . . . . .	343
8.22.2 構築子と解体子 . . . . .	343
8.22.2.1 GgSimpleShader() [1/4] . . . . .	343
8.22.2.2 GgSimpleShader() [2/4] . . . . .	343
8.22.2.3 GgSimpleShader() [3/4] . . . . .	344
8.22.2.4 GgSimpleShader() [4/4] . . . . .	344
8.22.2.5 ~GgSimpleShader() . . . . .	344
8.22.3 関数詳解 . . . . .	344
8.22.3.1 load() [1/2] . . . . .	344
8.22.3.2 load() [2/2] . . . . .	345
8.22.3.3 loadMatrix() [1/4] . . . . .	346
8.22.3.4 loadMatrix() [2/4] . . . . .	346
8.22.3.5 loadMatrix() [3/4] . . . . .	347
8.22.3.6 loadMatrix() [4/4] . . . . .	347
8.22.3.7 loadModelviewMatrix() [1/4] . . . . .	348
8.22.3.8 loadModelviewMatrix() [2/4] . . . . .	348
8.22.3.9 loadModelviewMatrix() [3/4] . . . . .	349
8.22.3.10 loadModelviewMatrix() [4/4] . . . . .	349
8.22.3.11 operator=() . . . . .	350
8.22.3.12 use() [1/13] . . . . .	350
8.22.3.13 use() [2/13] . . . . .	350
8.22.3.14 use() [3/13] . . . . .	351
8.22.3.15 use() [4/13] . . . . .	351
8.22.3.16 use() [5/13] . . . . .	352

---

8.22.3.17 use() [6/13] . . . . .	352
8.22.3.18 use() [7/13] . . . . .	353
8.22.3.19 use() [8/13] . . . . .	353
8.22.3.20 use() [9/13] . . . . .	354
8.22.3.21 use() [10/13] . . . . .	354
8.22.3.22 use() [11/13] . . . . .	355
8.22.3.23 use() [12/13] . . . . .	355
8.22.3.24 use() [13/13] . . . . .	356
8.23 gg::GgTexture クラス . . . . .	356
8.23.1 詳解 . . . . .	357
8.23.2 構築子と解体子 . . . . .	357
8.23.2.1 GgTexture() [1/2] . . . . .	357
8.23.2.2 ~GgTexture() . . . . .	357
8.23.2.3 GgTexture() [2/2] . . . . .	358
8.23.3 関数詳解 . . . . .	358
8.23.3.1 bind() . . . . .	358
8.23.3.2 getHeight() . . . . .	358
8.23.3.3 getSize() [1/2] . . . . .	359
8.23.3.4 getSize() [2/2] . . . . .	359
8.23.3.5 getTexture() . . . . .	359
8.23.3.6 getWidth() . . . . .	360
8.23.3.7 operator=() . . . . .	360
8.23.3.8 swapRandB() . . . . .	360
8.23.3.9 unbind() . . . . .	361
8.24 gg::GgTrackball クラス . . . . .	361
8.24.1 詳解 . . . . .	362
8.24.2 構築子と解体子 . . . . .	363
8.24.2.1 GgTrackball() . . . . .	363
8.24.2.2 ~GgTrackball() . . . . .	363
8.24.3 関数詳解 . . . . .	363
8.24.3.1 begin() . . . . .	363
8.24.3.2 end() . . . . .	364
8.24.3.3 get() . . . . .	364
8.24.3.4 getMatrix() . . . . .	365
8.24.3.5 getQuaternion() . . . . .	365
8.24.3.6 getScale() [1/3] . . . . .	366
8.24.3.7 getScale() [2/3] . . . . .	366
8.24.3.8 getScale() [3/3] . . . . .	366
8.24.3.9 getStart() [1/3] . . . . .	366
8.24.3.10 getStart() [2/3] . . . . .	367
8.24.3.11 getStart() [3/3] . . . . .	367
8.24.3.12 motion() . . . . .	367

---

8.24.3.13 operator=()	368
8.24.3.14 region() [1/2]	369
8.24.3.15 region() [2/2]	369
8.24.3.16 reset()	370
8.24.3.17 rotate()	370
8.25 gg::GgTriangles クラス	371
8.25.1 詳解	372
8.25.2 構築子と解体子	372
8.25.2.1 GgTriangles() [1/2]	372
8.25.2.2 GgTriangles() [2/2]	372
8.25.2.3 ~GgTriangles()	373
8.25.3 関数詳解	373
8.25.3.1 draw()	373
8.25.3.2 getBuffer()	374
8.25.3.3 getCount()	374
8.25.3.4 load()	374
8.25.3.5 send()	375
8.26 gg::GgUniformBuffer< T > クラステンプレート	375
8.26.1 詳解	376
8.26.2 構築子と解体子	376
8.26.2.1 GgUniformBuffer() [1/3]	376
8.26.2.2 GgUniformBuffer() [2/3]	376
8.26.2.3 GgUniformBuffer() [3/3]	377
8.26.2.4 ~GgUniformBuffer()	377
8.26.3 関数詳解	377
8.26.3.1 bind()	378
8.26.3.2 copy()	378
8.26.3.3 fill()	378
8.26.3.4 getBuffer()	379
8.26.3.5 getCount()	379
8.26.3.6 getStride()	379
8.26.3.7 getTarget()	380
8.26.3.8 load() [1/2]	380
8.26.3.9 load() [2/2]	380
8.26.3.10 map() [1/2]	381
8.26.3.11 map() [2/2]	381
8.26.3.12 read()	381
8.26.3.13 send()	382
8.26.3.14 unbind()	382
8.26.3.15 unmap()	383
8.27 gg::GgVector クラス	383
8.27.1 詳解	384

---

8.27.2 構築子と解体子	384
8.27.2.1 GgVector() [1/4]	384
8.27.2.2 GgVector() [2/4]	385
8.27.2.3 GgVector() [3/4]	386
8.27.2.4 GgVector() [4/4]	386
8.27.3 関数詳解	386
8.27.3.1 distance3()	387
8.27.3.2 distance4()	387
8.27.3.3 dot3()	388
8.27.3.4 dot4()	388
8.27.3.5 length3()	389
8.27.3.6 length4()	390
8.27.3.7 normalize3()	390
8.27.3.8 normalize4()	391
8.27.3.9 operator*() [1/2]	391
8.27.3.10 operator*() [2/2]	391
8.27.3.11 operator*=( ) [1/2]	392
8.27.3.12 operator*=( ) [2/2]	392
8.27.3.13 operator+() [1/2]	393
8.27.3.14 operator+() [2/2]	393
8.27.3.15 operator+=( ) [1/2]	393
8.27.3.16 operator+=( ) [2/2]	394
8.27.3.17 operator-() [1/2]	394
8.27.3.18 operator-() [2/2]	395
8.27.3.19 operator-=( ) [1/2]	395
8.27.3.20 operator-=( ) [2/2]	395
8.27.3.21 operator/() [1/2]	396
8.27.3.22 operator/() [2/2]	396
8.27.3.23 operator/=( ) [1/2]	397
8.27.3.24 operator/=( ) [2/2]	397
8.28 gg::GgVertex 構造体	397
8.28.1 詳解	398
8.28.2 構築子と解体子	398
8.28.2.1 GgVertex() [1/4]	399
8.28.2.2 GgVertex() [2/4]	399
8.28.2.3 GgVertex() [3/4]	399
8.28.2.4 GgVertex() [4/4]	400
8.28.3 メンバ詳解	400
8.28.3.1 normal	400
8.28.3.2 position	400
8.29 gg::GgVertexArray クラス	401
8.29.1 詳解	401

---

8.29.2 構築子と解体子	401
8.29.2.1 GgVertexArray() [1/2]	401
8.29.2.2 GgVertexArray() [2/2]	401
8.29.2.3 ~GgVertexArray()	402
8.29.3 関数詳解	402
8.29.3.1 bind()	402
8.29.3.2 get()	402
8.29.3.3 operator=()	402
8.30 Intrinsics 構造体	403
8.30.1 詳解	403
8.30.2 構築子と解体子	403
8.30.2.1 Intrinsics() [1/3]	404
8.30.2.2 Intrinsics() [2/3]	404
8.30.2.3 Intrinsics() [3/3]	404
8.30.3 関数詳解	405
8.30.3.1 setCenter()	405
8.30.3.2 setFov() [1/2]	405
8.30.3.3 setFov() [2/2]	406
8.30.3.4 setFps()	407
8.30.3.5 setSize()	407
8.30.4 メンバ詳解	407
8.30.4.1 center	407
8.30.4.2 fov	408
8.30.4.3 fps	408
8.30.4.4 sensorSize	408
8.30.4.5 size	408
8.31 gg::GgSimpleShader::Light 構造体	409
8.31.1 詳解	409
8.31.2 メンバ詳解	409
8.31.2.1 ambient	410
8.31.2.2 diffuse	410
8.31.2.3 position	410
8.31.2.4 specular	410
8.32 gg::GgSimpleShader::LightBuffer クラス	411
8.32.1 詳解	412
8.32.2 構築子と解体子	412
8.32.2.1 LightBuffer() [1/2]	412
8.32.2.2 LightBuffer() [2/2]	412
8.32.2.3 ~LightBuffer()	413
8.32.3 関数詳解	413
8.32.3.1 load() [1/2]	413
8.32.3.2 load() [2/2]	413

---

8.32.3.3 loadAmbient() [1/3] . . . . .	414
8.32.3.4 loadAmbient() [2/3] . . . . .	414
8.32.3.5 loadAmbient() [3/3] . . . . .	415
8.32.3.6 loadColor() . . . . .	415
8.32.3.7 loadDiffuse() [1/3] . . . . .	416
8.32.3.8 loadDiffuse() [2/3] . . . . .	416
8.32.3.9 loadDiffuse() [3/3] . . . . .	416
8.32.3.10 loadPosition() [1/4] . . . . .	417
8.32.3.11 loadPosition() [2/4] . . . . .	417
8.32.3.12 loadPosition() [3/4] . . . . .	418
8.32.3.13 loadPosition() [4/4] . . . . .	418
8.32.3.14 loadSpecular() [1/3] . . . . .	419
8.32.3.15 loadSpecular() [2/3] . . . . .	419
8.32.3.16 loadSpecular() [3/3] . . . . .	419
8.32.3.17 select() . . . . .	420
8.33 gg::GgSimpleShader::Material 構造体 . . . . .	420
8.33.1 詳解 . . . . .	421
8.33.2 メンバ詳解 . . . . .	421
8.33.2.1 ambient . . . . .	421
8.33.2.2 diffuse . . . . .	422
8.33.2.3 shininess . . . . .	422
8.33.2.4 specular . . . . .	422
8.34 gg::GgSimpleShader::MaterialBuffer クラス . . . . .	422
8.34.1 詳解 . . . . .	423
8.34.2 構築子と解体子 . . . . .	423
8.34.2.1 MaterialBuffer() [1/2] . . . . .	424
8.34.2.2 MaterialBuffer() [2/2] . . . . .	424
8.34.2.3 ~MaterialBuffer() . . . . .	424
8.34.3 関数詳解 . . . . .	424
8.34.3.1 load() [1/2] . . . . .	425
8.34.3.2 load() [2/2] . . . . .	425
8.34.3.3 loadAmbient() [1/3] . . . . .	425
8.34.3.4 loadAmbient() [2/3] . . . . .	426
8.34.3.5 loadAmbient() [3/3] . . . . .	426
8.34.3.6 loadAmbientAndDiffuse() [1/3] . . . . .	427
8.34.3.7 loadAmbientAndDiffuse() [2/3] . . . . .	427
8.34.3.8 loadAmbientAndDiffuse() [3/3] . . . . .	427
8.34.3.9 loadDiffuse() [1/3] . . . . .	428
8.34.3.10 loadDiffuse() [2/3] . . . . .	428
8.34.3.11 loadDiffuse() [3/3] . . . . .	429
8.34.3.12 loadShininess() [1/2] . . . . .	429
8.34.3.13 loadShininess() [2/2] . . . . .	430

---

8.34.3.14 loadSpecular() [1/3] . . . . .	430
8.34.3.15 loadSpecular() [2/3] . . . . .	430
8.34.3.16 loadSpecular() [3/3] . . . . .	431
8.34.3.17 select() . . . . .	431
8.35 Menu クラス . . . . .	432
8.35.1 詳解 . . . . .	432
8.35.2 構築子と解体子 . . . . .	432
8.35.2.1 Menu() [1/2] . . . . .	432
8.35.2.2 Menu() [2/2] . . . . .	433
8.35.2.3 ~Menu() . . . . .	433
8.35.3 関数詳解 . . . . .	433
8.35.3.1 draw() . . . . .	433
8.35.3.2 getMenuHeight() . . . . .	434
8.35.3.3 getPose() . . . . .	435
8.35.3.4 operator bool() . . . . .	435
8.35.3.5 operator=() . . . . .	435
8.35.3.6 saveImage() . . . . .	435
8.35.3.7 setSize() . . . . .	436
8.35.3.8 setup() . . . . .	437
8.35.4 メンバ詳解 . . . . .	438
8.35.4.1 detectBoard . . . . .	438
8.35.4.2 detectMarker . . . . .	438
8.36 Mesh クラス . . . . .	438
8.36.1 詳解 . . . . .	438
8.36.2 構築子と解体子 . . . . .	439
8.36.2.1 Mesh() [1/2] . . . . .	439
8.36.2.2 Mesh() [2/2] . . . . .	439
8.36.2.3 ~Mesh() . . . . .	439
8.36.3 関数詳解 . . . . .	439
8.36.3.1 draw() . . . . .	439
8.36.3.2 operator=() . . . . .	440
8.37 Preference クラス . . . . .	440
8.37.1 詳解 . . . . .	440
8.37.2 構築子と解体子 . . . . .	441
8.37.2.1 Preference() [1/3] . . . . .	441
8.37.2.2 Preference() [2/3] . . . . .	441
8.37.2.3 Preference() [3/3] . . . . .	441
8.37.2.4 ~Preference() . . . . .	442
8.37.3 関数詳解 . . . . .	442
8.37.3.1 buildShader() . . . . .	442
8.37.3.2 getDescription() . . . . .	442
8.37.3.3 getIntrinsics() . . . . .	443

---

8.37.3.4 getShader()	443
8.37.3.5 setPreference()	443
8.38 Settings 構造体	444
8.38.1 詳解	445
8.38.2 構築子と解体子	445
8.38.2.1 Settings()	445
8.38.3 関数詳解	445
8.38.3.1 getFocal()	445
8.38.4 メンバ詳解	446
8.38.4.1 checkerLength	446
8.38.4.2 defaultEuler	446
8.38.4.3 defaultFocal	446
8.38.4.4 defaultFocalRange	446
8.38.4.5 dictionaryName	446
8.38.4.6 euler	447
8.38.4.7 focal	447
8.38.4.8 focalRange	447
8.38.4.9 samples	447
8.39 Texture クラス	448
8.39.1 詳解	449
8.39.2 構築子と解体子	449
8.39.2.1 Texture() [1/4]	449
8.39.2.2 Texture() [2/4]	449
8.39.2.3 Texture() [3/4]	450
8.39.2.4 Texture() [4/4]	450
8.39.2.5 ~Texture()	451
8.39.3 関数詳解	451
8.39.3.1 bindTexture()	451
8.39.3.2 copy()	452
8.39.3.3 create()	453
8.39.3.4 discard()	454
8.39.3.5 drawPixels() [1/4]	455
8.39.3.6 drawPixels() [2/4]	455
8.39.3.7 drawPixels() [3/4]	456
8.39.3.8 drawPixels() [4/4]	456
8.39.3.9 getChannels()	457
8.39.3.10 getSize()	458
8.39.3.11 getTextureName()	458
8.39.3.12 operator=() [1/2]	458
8.39.3.13 operator=() [2/2]	459
8.39.3.14 readPixels() [1/3]	460
8.39.3.15 readPixels() [2/3]	460

---

8.39.3.16 readPixels() [3/3] . . . . .	461
8.39.3.17 unbindTexture() . . . . .	461
8.40 GgApp::Window クラス . . . . .	462
8.40.1 詳解 . . . . .	463
8.40.2 構築子と解体子 . . . . .	463
8.40.2.1 Window() [1/2] . . . . .	463
8.40.2.2 Window() [2/2] . . . . .	464
8.40.2.3 ~Window() . . . . .	464
8.40.3 関数詳解 . . . . .	464
8.40.3.1 get() . . . . .	464
8.40.3.2 getAltArrowX() . . . . .	465
8.40.3.3 getAltArrowY() . . . . .	465
8.40.3.4 getAltIArrow() . . . . .	466
8.40.3.5 getArrow() [1/2] . . . . .	467
8.40.3.6 getArrow() [2/2] . . . . .	467
8.40.3.7 getArrowX() . . . . .	468
8.40.3.8 getArrowY() . . . . .	469
8.40.3.9 getAspect() . . . . .	470
8.40.3.10 getControlArrow() . . . . .	470
8.40.3.11 getControlArrowX() . . . . .	470
8.40.3.12 getControlArrowY() . . . . .	471
8.40.3.13 getFboHeight() . . . . .	472
8.40.3.14 getFboSize() . . . . .	472
8.40.3.15 getFboWidth() . . . . .	472
8.40.3.16 getHeight() . . . . .	473
8.40.3.17 getKey() . . . . .	473
8.40.3.18 getLastKey() . . . . .	473
8.40.3.19 getMouse() [1/3] . . . . .	473
8.40.3.20 getMouse() [2/3] . . . . .	473
8.40.3.21 getMouse() [3/3] . . . . .	474
8.40.3.22 getMouseX() . . . . .	474
8.40.3.23 getMouseY() . . . . .	474
8.40.3.24 getRotation() . . . . .	475
8.40.3.25 getRotationMatrix() . . . . .	475
8.40.3.26 getScrollMatrix() . . . . .	475
8.40.3.27 getShiftArrow() . . . . .	476
8.40.3.28 getShiftArrowX() . . . . .	476
8.40.3.29 getShiftArrowY() . . . . .	477
8.40.3.30 getSize() . . . . .	478
8.40.3.31 getTranslation() . . . . .	478
8.40.3.32 getTranslationMatrix() . . . . .	478
8.40.3.33 getUserPointer() . . . . .	479

8.40.3.34 getWheel() [1/3] . . . . .	479
8.40.3.35 getWheel() [2/3] . . . . .	479
8.40.3.36 getWheel() [3/3] . . . . .	480
8.40.3.37 getWheelX() . . . . .	480
8.40.3.38 getWheelY() . . . . .	480
8.40.3.39 getWidth() . . . . .	481
8.40.3.40 operator bool() . . . . .	481
8.40.3.41 operator=( ) . . . . .	481
8.40.3.42 reset() . . . . .	482
8.40.3.43 resetRotation() . . . . .	482
8.40.3.44 resetTranslation() . . . . .	483
8.40.3.45 restoreViewport() . . . . .	483
8.40.3.46 selectInterface() . . . . .	483
8.40.3.47 setClose() . . . . .	483
8.40.3.48 setKeyboardFunc() . . . . .	485
8.40.3.49 setMouseFunc() . . . . .	485
8.40.3.50 setResizeFunc() . . . . .	485
8.40.3.51 setUserPointer() . . . . .	486
8.40.3.52 setVelocity() . . . . .	486
8.40.3.53 setWheelFunc() . . . . .	486
8.40.3.54 shouldClose() . . . . .	487
8.40.3.55 swapBuffers() . . . . .	487
8.40.3.56 updateViewport() . . . . .	487
<b>9 ファイル詳解</b> . . . . .	<b>489</b>
9.1 Buffer.cpp ファイル . . . . .	489
9.1.1 詳解 . . . . .	489
9.2 Buffer.cpp . . . . .	490
9.3 Buffer.h ファイル . . . . .	493
9.3.1 詳解 . . . . .	494
9.3.2 マクロ定義詳解 . . . . .	494
9.3.2.1 USE_PIXEL_BUFFER_OBJECT . . . . .	494
9.4 Buffer.h . . . . .	494
9.5 calib.cpp ファイル . . . . .	496
9.5.1 詳解 . . . . .	497
9.5.2 マクロ定義詳解 . . . . .	497
9.5.2.1 CONFIG_FILE . . . . .	497
9.6 calib.cpp . . . . .	497
9.7 Calibration.cpp ファイル . . . . .	498
9.7.1 詳解 . . . . .	499
9.8 Calibration.cpp . . . . .	499
9.9 Calibration.h ファイル . . . . .	504

---

9.9.1 詳解 . . . . .	505
9.10 Calibration.h . . . . .	506
9.11 CamCv.h ファイル . . . . .	507
9.11.1 詳解 . . . . .	508
9.12 CamCv.h . . . . .	509
9.13 Camera.h ファイル . . . . .	511
9.13.1 詳解 . . . . .	512
9.14 Camera.h . . . . .	513
9.15 CamImage.h ファイル . . . . .	515
9.15.1 詳解 . . . . .	516
9.16 CamImage.h . . . . .	517
9.17 Capture.cpp ファイル . . . . .	519
9.17.1 詳解 . . . . .	519
9.18 Capture.cpp . . . . .	519
9.19 Capture.h ファイル . . . . .	521
9.19.1 詳解 . . . . .	522
9.20 Capture.h . . . . .	522
9.21 Config.cpp ファイル . . . . .	523
9.21.1 詳解 . . . . .	524
9.21.2 関数詳解 . . . . .	524
9.21.2.1 getAnyList() . . . . .	524
9.22 Config.cpp . . . . .	525
9.23 Config.h ファイル . . . . .	530
9.23.1 詳解 . . . . .	531
9.24 Config.h . . . . .	531
9.25 Expand.cpp ファイル . . . . .	532
9.26 Expand.cpp . . . . .	533
9.27 Expand.h ファイル . . . . .	534
9.27.1 詳解 . . . . .	535
9.28 Expand.h . . . . .	536
9.29 Framebuffer.cpp ファイル . . . . .	537
9.29.1 詳解 . . . . .	537
9.30 Framebuffer.cpp . . . . .	538
9.31 Framebuffer.h ファイル . . . . .	541
9.31.1 詳解 . . . . .	542
9.32 Framebuffer.h . . . . .	542
9.33 gg.cpp ファイル . . . . .	543
9.33.1 詳解 . . . . .	543
9.34 gg.cpp . . . . .	544
9.35 gg.h ファイル . . . . .	620
9.35.1 詳解 . . . . .	624
9.35.2 マクロ定義詳解 . . . . .	624

---

9.35.2.1 ggError . . . . .	624
9.35.2.2 ggFBOError . . . . .	624
9.35.3 型定義詳解 . . . . .	625
9.35.3.1 pathChar . . . . .	625
9.35.3.2 pathString . . . . .	625
9.35.4 関数詳解 . . . . .	625
9.35.4.1 TCHARToUtf8() . . . . .	625
9.35.4.2 Utf8ToTCHAR() . . . . .	625
9.36 gg.h . . . . .	626
9.37 GgApp.cpp ファイル . . . . .	680
9.37.1 詳解 . . . . .	680
9.38 GgApp.cpp . . . . .	681
9.39 GgApp.h ファイル . . . . .	694
9.39.1 詳解 . . . . .	695
9.39.2 マクロ定義詳解 . . . . .	695
9.39.2.1 GG_BUTTON_COUNT . . . . .	695
9.39.2.2 GG_INTERFACE_COUNT . . . . .	695
9.39.2.3 GG_USE_IMGUI . . . . .	696
9.40 GgApp.h . . . . .	696
9.41 Intrinsics.cpp ファイル . . . . .	703
9.41.1 詳解 . . . . .	703
9.42 Intrinsics.cpp . . . . .	704
9.43 Intrinsics.h ファイル . . . . .	704
9.43.1 詳解 . . . . .	706
9.44 Intrinsics.h . . . . .	706
9.45 main.cpp ファイル . . . . .	707
9.45.1 詳解 . . . . .	707
9.45.2 マクロ定義詳解 . . . . .	708
9.45.2.1 HEADER_STR . . . . .	708
9.45.3 関数詳解 . . . . .	708
9.45.3.1 main() . . . . .	708
9.46 main.cpp . . . . .	708
9.47 Menu.cpp ファイル . . . . .	709
9.47.1 詳解 . . . . .	710
9.47.2 変数詳解 . . . . .	710
9.47.2.1 imageFilter . . . . .	710
9.47.2.2 jsonFilter . . . . .	711
9.47.2.3 movieFilter . . . . .	711
9.48 Menu.cpp . . . . .	711
9.49 Menu.h ファイル . . . . .	720
9.49.1 詳解 . . . . .	721
9.50 Menu.h . . . . .	721

---

9.51 Mesh.h ファイル . . . . .	722
9.51.1 詳解 . . . . .	723
9.52 Mesh.h . . . . .	723
9.53 parseconfig.h ファイル . . . . .	724
9.53.1 詳解 . . . . .	726
9.53.2 関数詳解 . . . . .	726
9.53.2.1 getString() [1/3] . . . . .	726
9.53.2.2 getString() [2/3] . . . . .	726
9.53.2.3 getString() [3/3] . . . . .	727
9.53.2.4 getValue() [1/2] . . . . .	727
9.53.2.5 getValue() [2/2] . . . . .	728
9.53.2.6 setString() [1/3] . . . . .	729
9.53.2.7 setString() [2/3] . . . . .	729
9.53.2.8 setString() [3/3] . . . . .	730
9.53.2.9 setValue() [1/2] . . . . .	730
9.53.2.10 setValue() [2/2] . . . . .	732
9.54 parseconfig.h . . . . .	733
9.55 Preference.cpp ファイル . . . . .	735
9.55.1 詳解 . . . . .	735
9.56 Preference.cpp . . . . .	736
9.57 Preference.h ファイル . . . . .	737
9.57.1 詳解 . . . . .	738
9.58 Preference.h . . . . .	739
9.59 README.md ファイル . . . . .	740
9.60 Texture.cpp ファイル . . . . .	740
9.60.1 詳解 . . . . .	740
9.61 Texture.cpp . . . . .	741
9.62 Texture.h ファイル . . . . .	743
9.62.1 詳解 . . . . .	744
9.63 Texture.h . . . . .	744
<b>Index</b>	<b>747</b>

## Chapter 1

# ゲームグラフィックス特論の宿題用補助 プログラム **GLFW3** 版.

著作権所有

Copyright (c) 2011-2024 Kohe Tokoi. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies or substantial portions of the Software.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



# Chapter 2

## Calib

ChArUco Board を使ったカメラキャリブレーションプログラム

Copyright (c) 2024 Kohe Tokoi. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies or substantial portions of the Software.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 2.1 libs ディレクトリについて

このリポジトリの `libs` ディレクトリ以下には、`OpenCV` の開発パッケージや `GStreamer` のランタイム用 DLL (ともに Windows 用と macOS 用) を置いています。これらを `Git LFS` に置こうとしたら、プライベートリポジトリのサイズの制限を超てしまい、追加料金が必要になったので、別に ZIP ファイルにまとめています。リポジトリを `clone` した後、[この ZIP ファイル](#) (488MB) を展開して、リポジトリのルートディレクトリに置いてください。



## Chapter 3

# 名前空間索引

### 3.1 名前空間一覧

全名前空間の一覧です。

[gg](#) ..... 13



# Chapter 4

## 階層索引

### 4.1 クラス階層

クラス階層一覧です。大雑把に文字符号順で並べられています。

std::array	191
gg::GgMatrix	191
gg::GgVector	383
gg::GgQuaternion	265
gg::GgTrackball	361
Buffer	89
Texture	448
Framebuffer	162
Calibration	106
Camera	127
CamCv	119
CamImage	138
Capture	143
Config	150
Expand	158
GgApp	174
gg::GgBuffer< T >	177
gg::GgColorTexture	182
gg::GgNormalTexture	247
gg::GgPointShader	256
gg::GgSimpleShader	341
gg::GgShader	331
gg::GgShape	334
gg::GgPoints	251
gg::GgTriangles	371
gg::GgElements	187
gg::GgSimpleObj	338
gg::GgTexture	356
gg::GgUniformBuffer< T >	375
gg::GgUniformBuffer< Light >	375
gg::GgSimpleShader::LightBuffer	411
gg::GgUniformBuffer< Material >	375
gg::GgSimpleShader::MaterialBuffer	422

---

gg::GgVertex . . . . .	397
gg::GgVertexArray . . . . .	401
Intrinsics . . . . .	403
gg::GgSimpleShader::Light . . . . .	409
gg::GgSimpleShader::Material . . . . .	420
Menu . . . . .	432
Mesh . . . . .	438
Preference . . . . .	440
Settings . . . . .	444
GgApp::Window . . . . .	462

# Chapter 5

## クラス索引

### 5.1 クラス一覧

クラス・構造体・共用体・インターフェースの一覧です。

Buffer	89
Calibration	106
CamCv	119
Camera	127
CamImage	138
Capture	143
Config	150
Expand	158
Framebuffer	162
GgApp	174
gg::GgBuffer< T >	177
gg::GgColorTexture	182
gg::GgElements	187
gg::GgMatrix	191
gg::GgNormalTexture	247
gg::GgPoints	251
gg::GgPointShader	256
gg::GgQuaternion	265
gg::GgShader	331
gg::GgShape	334
gg::GgSimpleObj	338
gg::GgSimpleShader	341
gg::GgTexture	356
gg::GgTrackball	361
gg::GgTriangles	371
gg::GgUniformBuffer< T >	375
gg::GgVector	383
gg::GgVertex	397
gg::GgVertexArray	401
Intrinsics	403
gg::GgSimpleShader::Light	409
gg::GgSimpleShader::LightBuffer	411
gg::GgSimpleShader::Material	420
gg::GgSimpleShader::MaterialBuffer	422
Menu	432

Mesh	438
Preference	440
Settings	444
Texture	448
GgApp::Window	462

# Chapter 6

## ファイル索引

### 6.1 ファイル一覧

ファイル一覧です。

Buffer.cpp	489
Buffer.h	493
calib.cpp	496
Calibration.cpp	498
Calibration.h	504
CamCv.h	507
Camera.h	511
CamImage.h	515
Capture.cpp	519
Capture.h	521
Config.cpp	523
Config.h	530
Expand.cpp	532
Expand.h	534
Framebuffer.cpp	537
Framebuffer.h	541
gg.cpp	543
gg.h	620
GgApp.cpp	680
GgApp.h	694
Intrinsics.cpp	703
Intrinsics.h	704
main.cpp	707
Menu.cpp	709
Menu.h	720
Mesh.h	722
parseconfig.h	724
Preference.cpp	735
Preference.h	737
Texture.cpp	740
Texture.h	743



# Chapter 7

## 名前空間詳解

### 7.1 gg 名前空間

#### クラス

- class `GgBuffer`
- class `GgColorTexture`
- class `GgElements`
- class `GgMatrix`
- class `GgNormalTexture`
- class `GgPoints`
- class `GgPointShader`
- class `GgQuaternion`
- class `GgShader`
- class `GgShape`
- class `GgSimpleObj`
- class `GgSimpleShader`
- class `GgTexture`
- class `GgTrackball`
- class `GgTriangles`
- class `GgUniformBuffer`
- class `GgVector`
- struct `GgVertex`
- class `GgVertexArray`

#### 列挙型

- enum `BindingPoints` { `LightBindingPoint` = 0 , `MaterialBindingPoint` }

## 関数

- void `ggInit ()`
- void `ggError (const std::string &name="", unsigned int line=0)`
- void `ggFBOError (const std::string &name="", unsigned int line=0)`
- void `ggCross (GLfloat *c, const GLfloat *a, const GLfloat *b)`
- GLfloat `ggDot3 (const GLfloat *a, const GLfloat *b)`
- GLfloat `ggLength3 (const GLfloat *a)`
- GLfloat `ggDistance3 (const GLfloat *a, const GLfloat *b)`
- void `ggNormalize3 (const GLfloat *a, GLfloat *b)`
- void `ggNormalize3 (GLfloat *a)`
- GLfloat `ggDot4 (const GLfloat *a, const GLfloat *b)`
- GLfloat `ggLength4 (const GLfloat *a)`
- GLfloat `ggDistance4 (const GLfloat *a, const GLfloat *b)`
- void `ggNormalize4 (const GLfloat *a, GLfloat *b)`
- void `ggNormalize4 (GLfloat *a)`
- const `GgVector & operator+ (const GgVector &v)`
- `GgVector operator+ (GLfloat a, const GgVector &b)`
- const `GgVector operator- (const GgVector &v)`
- `GgVector operator- (GLfloat a, const GgVector &b)`
- `GgVector operator* (GLfloat a, const GgVector &b)`
- `GgVector operator/ (GLfloat a, const GgVector &b)`
- `GgVector ggCross (const GgVector &a, const GgVector &b)`
- GLfloat `ggDot3 (const GgVector &a, const GgVector &b)`
- GLfloat `ggLength3 (const GgVector &a)`
- GLfloat `ggDistance3 (const GgVector &a, const GgVector &b)`
- `GgVector ggNormalize3 (const GgVector &a)`
- void `ggNormalize3 (GgVector *a)`
- GLfloat `ggDot4 (const GgVector &a, const GgVector &b)`
- GLfloat `ggLength4 (const GgVector &a)`
- GLfloat `ggDistance4 (const GgVector &a, const GgVector &b)`
- `GgVector ggNormalize4 (const GgVector &a)`
- void `ggNormalize4 (GgVector *a)`
- `GgMatrix ggIdentity ()`
- `GgMatrix ggTranslate (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix ggTranslate (const GLfloat *t)`
- `GgMatrix ggTranslate (const GgVector &t)`
- `GgMatrix ggScale (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix ggScale (const GLfloat *s)`
- `GgMatrix ggScale (const GgVector &s)`
- `GgMatrix ggRotateX (GLfloat a)`
- `GgMatrix ggRotateY (GLfloat a)`
- `GgMatrix ggRotateZ (GLfloat a)`
- `GgMatrix ggRotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgMatrix ggRotate (const GLfloat *r, GLfloat a)`
- `GgMatrix ggRotate (const GgVector &r, GLfloat a)`
- `GgMatrix ggRotate (const GLfloat *r)`
- `GgMatrix ggRotate (const GgVector &r)`
- `GgMatrix ggLookat (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)`
- `GgMatrix ggLookat (const GLfloat *e, const GLfloat *t, const GLfloat *u)`
- `GgMatrix ggLookat (const GgVector &e, const GgVector &t, const GgVector &u)`
- `GgMatrix ggOrthogonal (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix ggFrustum (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix ggPerspective (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)`

- `GgMatrix ggTranspose (const GgMatrix &m)`
- `GgMatrix ggInvert (const GgMatrix &m)`
- `GgMatrix ggNormal (const GgMatrix &m)`
- `GgQuaternion ggQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion ggQuaternion (const GLfloat *a)`
- `GgQuaternion ggIdentityQuaternion ()`
- `GgQuaternion ggMatrixQuaternion (const GLfloat *a)`
- `GgQuaternion ggMatrixQuaternion (const GgMatrix &m)`
- `GgMatrix ggQuaternionMatrix (const GgQuaternion &q)`
- `GgMatrix ggQuaternionTransposeMatrix (const GgQuaternion &q)`
- `GgQuaternion ggRotateQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgQuaternion ggRotateQuaternion (const GLfloat *v, GLfloat a)`
- `GgQuaternion ggRotateQuaternion (const GLfloat *v)`
- `GgQuaternion ggEulerQuaternion (GLfloat heading, GLfloat pitch, GLfloat roll)`
- `GgQuaternion ggEulerQuaternion (const GLfloat *e)`
- `GgQuaternion ggSlerp (const GLfloat *a, const GLfloat *b, GLfloat t)`
- `GgQuaternion ggSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
- `GgQuaternion ggSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
- `GgQuaternion ggSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
- `GLfloat ggNorm (const GgQuaternion &q)`
- `GgQuaternion ggNormalize (const GgQuaternion &q)`
- `GgQuaternion ggConjugate (const GgQuaternion &q)`
- `GgQuaternion ggInvert (const GgQuaternion &q)`
- `bool ggSaveTga (const std::string &name, const void *buffer, unsigned int width, unsigned int height, unsigned int depth)`
- `bool ggSaveColor (const std::string &name)`
- `bool ggSaveDepth (const std::string &name)`
- `bool ggReadImage (const std::string &name, std::vector< GLubyte > &image, GLsizei *pWidth, GLsizei *pHeight, GLenum *pFormat)`
- `GLuint ggLoadTexture (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE, bool swizzle=true)`
- `GLuint ggLoadImage (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE)`
- `void ggCreateNormalMap (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)`
- `GLuint ggLoadHeight (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)`
- `GLuint ggCreateShader (const std::string &vsr, const std::string &fsr="", const std::string &gsr="", GLint nvarying=0, const char *const *varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string &gtext="geometry shader")`
- `GLuint ggLoadShader (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)`
- `GLuint ggLoadShader (const std::array< std::string, 3 > &files, GLint nvarying=0, const char *const *varyings=nullptr)`
- `GLuint ggCreateComputeShader (const std::string &csr, const std::string &ctext="compute shader")`
- `GLuint ggLoadComputeShader (const std::string &comp)`
- `std::shared_ptr< GgPoints > ggPointsCube (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
- `std::shared_ptr< GgPoints > ggPointsSphere (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
- `std::shared_ptr< GgTriangles > ggRectangle (GLfloat width=1.0f, GLfloat height=1.0f)`
- `std::shared_ptr< GgTriangles > ggEllipse (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)`
- `std::shared_ptr< GgTriangles > ggArraysObj (const std::string &name, bool normalize=false)`
- `std::shared_ptr< GgElements > ggElementsObj (const std::string &name, bool normalize=false)`

- std::shared\_ptr< GgElements > ggElementsMesh (GLuint slices, GLuint stacks, const GLfloat(\*pos)[3], const GLfloat(\*norm)[3]=nullptr)
- std::shared\_ptr< GgElements > ggElementsSphere (GLfloat radius=1.0f, int slices=16, int stacks=8)
- bool ggLoadSimpleObj (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, bool normalize=false)
- bool ggLoadSimpleObj (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)

## 変数

- GLint ggBufferAlignment

使用している GPU のバッファアライメント.

### 7.1.1 詳解

ゲームグラフィックス特論の宿題用補助プログラムの名前空間

### 7.1.2 列挙型詳解

#### 7.1.2.1 BindingPoints

```
enum gg::BindingPoints
```

光源と材質の uniform buffer object の結合ポイント.

列挙値

LightBindingPoint	光源の uniform buffer object の結合ポイント.
MaterialBindingPoint	材質の uniform buffer object の結合ポイント.

gg.h の 1349 行目に定義があります。

### 7.1.3 関数詳解

#### 7.1.3.1 \_ggError()

```
void gg::_ggError (
    const std::string & name = "",
    unsigned int line = 0 )
```

OpenGL のエラーをチェックする.

引数

<i>name</i>	エラー発生時に標準エラー出力に出力する文字列, <code>nullptr</code> なら何も出力しない.
<i>line</i>	エラー発生時に標準エラー出力に出力する数値, 0 なら何も出力しない.

覚え書き

OpenGL の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する。

[gg.cpp](#) の 2614 行目に定義があります。

### 7.1.3.2 `ggFBOError()`

```
void gg::ggFBOError (
    const std::string & name = "",
    unsigned int line = 0 )
```

FBO のエラーをチェックする。

引数

<i>name</i>	エラー発生時に標準エラー出力に出力する文字列, <code>nullptr</code> なら何も出力しない.
<i>line</i>	エラー発生時に標準エラー出力に出力する数値, 0 なら何も出力しない.

覚え書き

FBO の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する。

[gg.cpp](#) の 2658 行目に定義があります。

### 7.1.3.3 `ggArraysObj()`

```
std::shared_ptr< gg::GgTriangles > gg::ggArraysObj (
    const std::string & name,
    bool normalize = false )
```

Wavefront OBJ ファイルを読み込む (Arrays 形式)

引数

<i>name</i>	ファイル名.
<i>normalize</i>	<code>true</code> なら大きさを正規化.

戻り値

`GgTriangles` 型のポインタ.

覚え書き

三角形分割された Wavefront OBJ ファイルを読み込んで `GgArrays` 形式の三角形データを生成する。

`gg.cpp` の 5260 行目に定義があります。

呼び出し関係図:



#### 7.1.3.4 `ggConjugate()`

```
GgQuaternion gg::ggConjugate (
    const GgQuaternion & q ) [inline]
```

共役四元数を返す。

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

四元数 `q` の共役四元数。

`gg.h` の 4747 行目に定義があります。

呼び出し関係図:



### 7.1.3.5 ggCreateComputeShader()

```
GLuint gg::ggCreateComputeShader (
    const std::string & csrc,
    const std::string & ctext = "compute shader" )
```

コンピュートシェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する。

引数

<i>csrc</i>	コンピュートシェーダのソースプログラムの文字列。
<i>ctext</i>	コンピュートシェーダのコンパイル時のメッセージに追加する文字列。

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0)。

gg.cpp の 5033 行目に定義があります。

被呼び出し関係図:



### 7.1.3.6 ggCreateNormalMap()

```
void gg::ggCreateNormalMap (
    const GLubyte * hmap,
    GLsizei width,
    GLsizei height,
    GLenum format,
    GLfloat nz,
    GLenum internal,
    std::vector< GgVector > & nmap )
```

グレースケール画像(8bit)から法線マップのデータを作成する。

引数

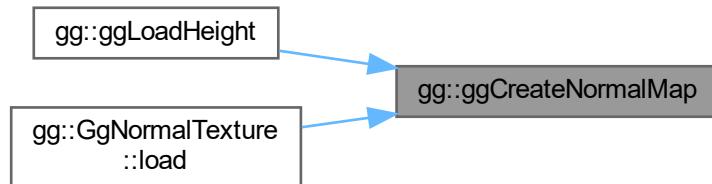
<i>hmap</i>	グレースケール画像のデータ。
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数。
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数。
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_RGBA)。
<small>構築</small> <b>Doxygen</b>	法線の z 成分の割合。
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット。
<i>nmap</i>	法線マップを格納する vector。

`gg.cpp` の 3846 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.7 `ggCreateShader()`

```

GLuint gg::ggCreateShader (
    const std::string & vsrc,
    const std::string & fsrc = "",
    const std::string & gsrc = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr,
    const std::string & vtext = "vertex shader",
    const std::string & ftext = "fragment shader",
    const std::string & gtext = "geometry shader" )
  
```

シェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する。

引数

<code>vsrc</code>	バーテックスシェーダのソースプログラムの文字列。
<code>fsrc</code>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用)。
<code>gsrc</code>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用)。
<code>nvarying</code>	フィードバックする <code>varying</code> 変数の数 (0 なら不使用)。
<code>varyings</code>	フィードバックする <code>varying</code> 変数のリスト ( <code>nullptr</code> なら不使用)。
<code>vtext</code>	バーテックスシェーダのコンパイル時のメッセージに追加する文字列。
<code>ftext</code>	フラグメントシェーダのコンパイル時のメッセージに追加する文字列。
<code>gtext</code>	ジオメトリシェーダのコンパイル時のメッセージに追加する文字列。

戻り値

プログラムオブジェクトのプログラム名(作成できなければ0).

`gg.cpp` の 4855 行目に定義があります。

呼び出し関係図:



### 7.1.3.8 ggCross() [1/2]

```
GgVector gg::ggCross (
    const GgVector & a,
    const GgVector & b ) [inline]
```

`GgVector` 型の 3 要素の外積.

引数

a	<code>GgVector</code> 型のベクトル.
b	<code>GgVector</code> 型のベクトル.

戻り値

a と b の外積.

覚え書き

戻り値の w (第4) 要素は 0.

`gg.h` の 1991 行目に定義があります。

呼び出し関係図:



### 7.1.3.9 ggCross() [2/2]

```
void gg::ggCross (
    GLfloat * c,
    const GLfloat * a,
    const GLfloat * b ) [inline]
```

3要素の外積.

引数

<i>a</i>	GLfloat 型の 3 要素の配列変数.
<i>b</i>	GLfloat 型の 3 要素の配列変数.
<i>c</i>	結果を格納する GLfloat 型の 3 要素の配列変数.

[gg.h](#) の 1429 行目に定義があります。

被呼び出し関係図:



### 7.1.3.10 ggDistance3() [1/2]

```
GLfloat gg::ggDistance3 (
    const GgVector & a,
    const GgVector & b ) [inline]
```

[GgVector](#) 型の 3 要素の距離.

引数

<i>a</i>	<a href="#">GgVector</a> 型のベクトル.
<i>b</i>	<a href="#">GgVector</a> 型のベクトル.

戻り値

a と b の距離.

gg.h の 2028 行目に定義があります。

呼び出し関係図:



### 7.1.3.11 ggDistance3() [2/2]

```
GLfloat gg::ggDistance3 (
    const GLfloat * a,
    const GLfloat * b )  [inline]
```

3 要素の距離.

引数

a	GLfloat 型の 3 要素の配列変数.
b	GLfloat 型の 3 要素の配列変数.

戻り値

a と b の距離.

gg.h の 1466 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



### 7.1.3.12 ggDistance4() [1/2]

```
GLfloat gg::ggDistance4 (
    const GgVector & a,
    const GgVector & b ) [inline]
```

**GgVector** 型の 4 要素の距離.

引数

a	<b>GgVector</b> 型の変数.
b	<b>GgVector</b> 型の変数.

戻り値

2 つの **GgVector** a, b の距離.

gg.h の 2094 行目に定義があります。

呼び出し関係図:



### 7.1.3.13 ggDistance4() [2/2]

```
GLfloat gg::ggDistance4 (
    const GLfloat * a,
    const GLfloat * b ) [inline]
```

GLfloat 型の 4 要素の距離.

引数

<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	GLfloat 型の 4 要素の配列変数.

戻り値

*a* と *b* のそれぞれの 4 要素の距離.

gg.h の 1531 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 7.1.3.14 ggDot3() [1/2]

```
GLfloat gg::ggDot3 (
    const GgVector & a,
    const GgVector & b ) [inline]
```

GgVector 型の 3 要素の内積.

引数

<i>a</i>	GgVector 型のベクトル.
<i>b</i>	GgVector 型のベクトル.

戻り値

$a$  と  $b$  のそれぞれの 3 要素の内積.

gg.h の 2005 行目に定義があります。

呼び出し関係図:



### 7.1.3.15 ggDot3() [2/2]

```
GLfloat gg::ggDot3 (
    const GLfloat * a,
    const GLfloat * b )  [inline]
```

3 要素の内積.

引数

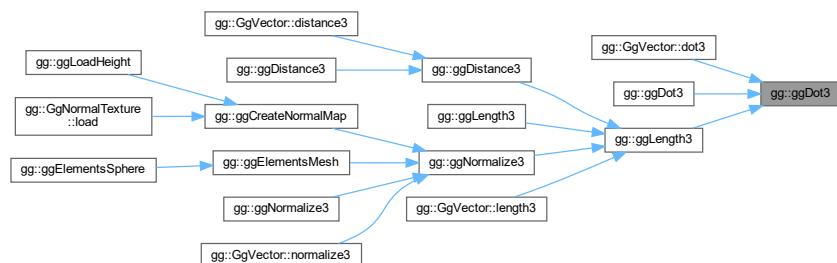
$a$	GLfloat 型の 3 要素の配列変数.
$b$	GLfloat 型の 3 要素の配列変数.

戻り値

$a$  と  $b$  のそれぞれの 3 要素の内積.

gg.h の 1443 行目に定義があります。

被呼び出し関係図:



### 7.1.3.16 ggDot4() [1/2]

```
GLfloat gg::ggDot4 (
    const GgVector & a,
    const GgVector & b ) [inline]
```

GgVector 型の 4 要素の内積.

引数

<i>a</i>	GgVector 型の変数.
<i>b</i>	GgVector 型の変数.

戻り値

*a* と *b* のそれぞれの 4 要素の内積.

gg.h の 2071 行目に定義があります。

呼び出し関係図:



### 7.1.3.17 ggDot4() [2/2]

```
GLfloat gg::ggDot4 (
    const GLfloat * a,
    const GLfloat * b ) [inline]
```

GLfloat 型の 4 要素の内積

引数

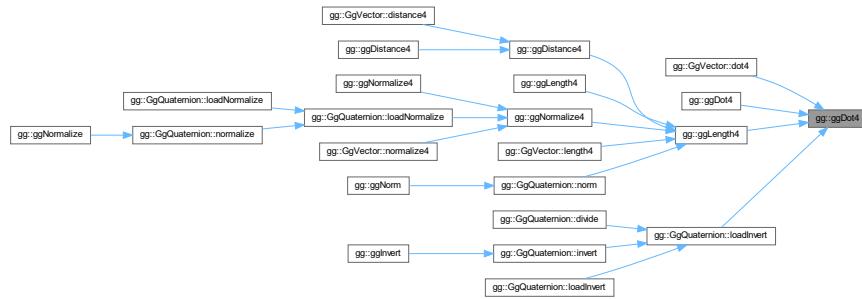
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	GLfloat 型の 4 要素の配列変数.

戻り値

`a` と `b` のそれぞれの 4 要素の内積.

`gg.h` の 1508 行目に定義があります。

被呼び出し関係図:



### 7.1.3.18 ggElementsMesh()

```
std::shared_ptr< gg::GgElements > gg::ggElementsMesh (
    GLuint slices,
    GLuint stacks,
    const GLfloat(*) pos[3],
    const GLfloat(*) norm[3] = nullptr )
```

メッシュ形状を作成する (Elements 形式).

引数

<code>slices</code>	メッシュの横方向の分割数.
<code>stacks</code>	メッシュの縦方向の分割数.
<code>pos</code>	メッシュの頂点の位置.
<code>norm</code>	メッシュの頂点の法線, <code>nullptr</code> なら頂点の位置から算出する.

戻り値

`GgElements` 型のポインタ.

覚え書き

メッシュ状に `GgElements` 形式の三角形データを生成する.

`gg.cpp` の 5294 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.19 ggElementsObj()

```
std::shared_ptr< gg::GgElements > gg::ggElementsObj (
    const std::string & name,
    bool normalize = false )
```

Wavefront OBJ ファイル を読み込む (Elements 形式).

引数

<i>name</i>	ファイル名.
<i>normalize</i>	true なら大きさを正規化.

戻り値

[GgElements](#) 型のポインタ.

覚え書き

三角形分割された Wavefront OBJ ファイル を読み込んで [GgElements](#) 形式の三角形データを生成する.

gg.cpp の 5276 行目に定義があります。

呼び出し関係図:



### 7.1.3.20 ggElementsSphere()

```
std::shared_ptr< gg::GgElements > gg::ggElementsSphere (
    GLfloat radius = 1.0f,
    int slices = 16,
    int stacks = 8 )
```

球状に三角形データを生成する (Elements 形式).

引数

<i>radius</i>	球の半径.
<i>slices</i>	球の経度方向の分割数.
<i>stacks</i>	球の緯度方向の分割数.

戻り値

[GgElements](#) 型のポインタ.

覚え書き

球状に [GgElements](#) 形式の三角形データを生成する.

[gg.cpp](#) の 5389 行目に定義があります。

呼び出し関係図:



### 7.1.3.21 ggEllipse()

```
std::shared_ptr< gg::GgTriangles > gg::ggEllipse (
    GLfloat width = 1.0f,
    GLfloat height = 1.0f,
    GLuint slices = 16 )
```

橢円状に三角形を生成する。

引数

<i>width</i>	橢円の横幅.
<i>height</i>	橢円の高さ.
<i>slices</i>	橢円の分割数.

戻り値

[GgTriangles](#) 型のポインタ.

[gg.cpp](#) の 5235 行目に定義があります。

### 7.1.3.22 ggEulerQuaternion() [1/2]

```
GgQuaternion gg::ggEulerQuaternion (
    const GLfloat * e ) [inline]
```

オイラー角 ( $e[0], e[1], e[2]$ ) で与えられた回転を表す四元数を返す。

引数

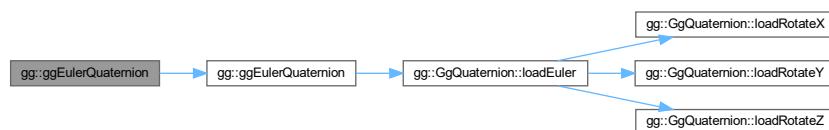
<i>e</i>	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転を表す四元数。

[gg.h](#) の 4661 行目に定義があります。

呼び出し関係図:



### 7.1.3.23 ggEulerQuaternion() [2/2]

```
GgQuaternion gg::ggEulerQuaternion (
    GLfloat heading,
    GLfloat pitch,
    GLfloat roll ) [inline]
```

オイラー角 (heading, pitch, roll) で与えられた回転を表す四元数を返す.

引数

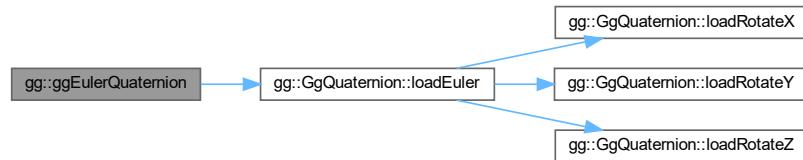
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転を表す四元数.

gg.h の 4649 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.24 ggFrustum()

```
GgMatrix gg::ggFrustum (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar ) [inline]
```

透視透視投影変換行列を返す。

引数

<i>left</i>	ウィンドウの左端の位置。
<i>right</i>	ウィンドウの右端の位置。
<i>bottom</i>	ウィンドウの下端の位置。
<i>top</i>	ウィンドウの上端の位置。
<i>zNear</i>	視点から前方面までの位置。
<i>zFar</i>	視点から後方面までの位置。

戻り値

求めた透視投影変換行列。

gg.h の 3360 行目に定義があります。

呼び出し関係図:



### 7.1.3.25 ggIdentity()

```
GgMatrix gg::ggIdentity ( ) [inline]
```

単位行列を返す。

戻り値

単位行列.

gg.h の 3085 行目に定義があります。

呼び出し関係図:



### 7.1.3.26 ggIdentityQuaternion()

`GgQuaternion gg::ggIdentityQuaternion( ) [inline]`

単位四元数を返す.

戻り値

単位四元数.

gg.h の 4547 行目に定義があります。

呼び出し関係図:



### 7.1.3.27 ggInit()

```
void gg::ggInit ( )
```

ゲームグラフィックス特論の都合にもとづく初期化を行う。

覚え書き

WindowsにおいてOpenGL 1.2以降のAPIを有効化する。

`gg.cpp` の 1359 行目に定義があります。

呼び出し関係図:



### 7.1.3.28 ggInvert() [1/2]

```
GgMatrix gg::ggInvert (
    const GgMatrix & m ) [inline]
```

逆行列を返す。

引数

<code>m</code>	元の変換行列。
----------------	---------

戻り値

`m` の逆行列。

`gg.h` の 3405 行目に定義があります。

呼び出し関係図:



### 7.1.3.29 ggInvert() [2/2]

```
GgQuaternion gg::ggInvert (
    const GgQuaternion & q ) [inline]
```

四元数の逆元を求める。

引数

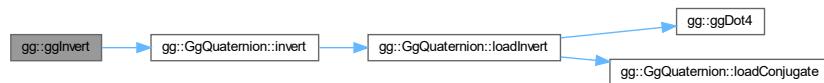
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

四元数 *q* の逆元。

gg.h の 4758 行目に定義があります。

呼び出し関係図:



### 7.1.3.30 ggLength3() [1/2]

```
GLfloat gg::ggLength3 (
    const GgVector & a ) [inline]
```

GgVector 型の 3 要素の長さ。

引数

<i>a</i>	GgVector 型のベクトル。
----------	------------------

戻り値

*a* の 3 要素の長さ。

gg.h の 2016 行目に定義があります。

呼び出し関係図:



### 7.1.3.31 ggLength3() [2/2]

```
GLfloat gg::ggLength3 (
    const GLfloat * a ) [inline]
```

3要素の長さ。

引数

a	GLfloat型の3要素の配列変数。
---	--------------------

戻り値

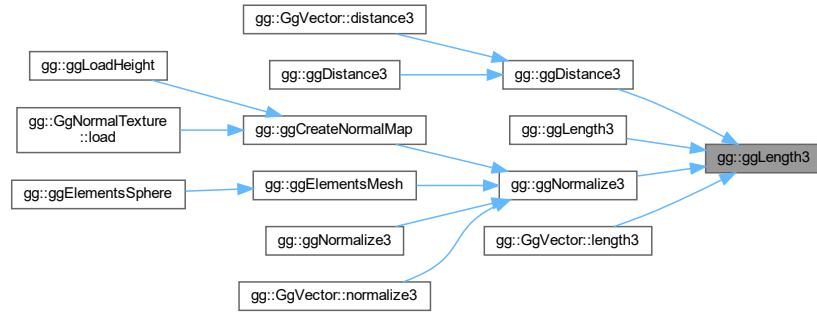
aの3要素の長さ。

gg.h の 1454 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.32 `ggLength4()` [1/2]

```
GLfloat gg::ggLength4 (
    const GgVector & a )  [inline]
```

`GgVector` 型の 4 要素の長さ。

引数

a	<code>GgVector</code> 型の変数。
---	-----------------------------

戻り値

`a` の 4 要素の長さ。

`gg.h` の 2082 行目に定義があります。

呼び出し関係図:



### 7.1.3.33 ggLength4() [2/2]

```
GLfloat gg::ggLength4 (
    const GLfloat * a ) [inline]
```

GLfloat 型の 4 要素の長さ.

引数

<i>a</i>	GLfloat 型の 4 要素の配列変数.
----------	-----------------------

戻り値

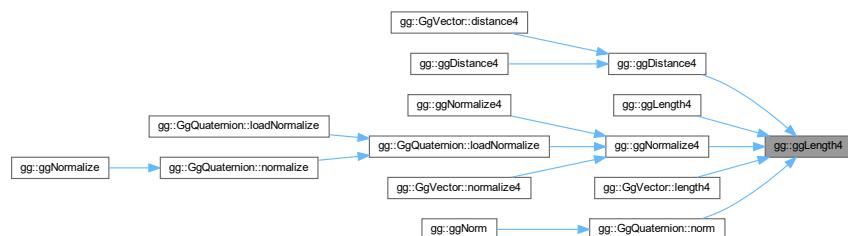
*a* の 4 要素の長さ.

[gg.h](#) の 1519 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.34 ggLoadComputeShader()

```
GLuint gg::ggLoadComputeShader (
    const std::string & comp )
```

コンピュートシェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.

引数

<i>comp</i>	コンピュートシェーダのソースファイル名.
-------------	----------------------

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

[gg.cpp](#) の 5074 行目に定義があります。

呼び出し関係図:



### 7.1.3.35 ggLoadHeight()

```
GLuint gg::ggLoadHeight (
    const std::string & name,
    GLfloat nz,
    GLsizei * pWidth = nullptr,
    GLsizei * pHeight = nullptr,
    GLenum internal = GL_RGBA )
```

TGA 画像ファイルの高さマップ読み込んで法線マップのテクスチャを作成する.

引数

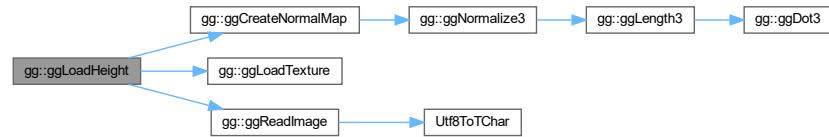
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

[gg.cpp](#) の 3931 行目に定義があります。

呼び出し関係図:



### 7.1.3.36 ggLoadImage()

```

GLuint gg::ggLoadImage (
    const std::string & name,
    GLsizei * pWidth = nullptr,
    GLsizei * pHeight = nullptr,
    GLenum internal = 0,
    GLenum wrap = GL_CLAMP_TO_EDGE )
  
```

テクスチャを作成して TGA フォーマットの画像ファイルを読み込む.

引数

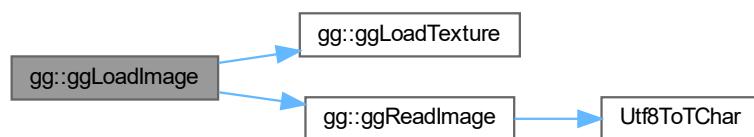
<i>name</i>	読み込むファイル名.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

gg.cpp の 3797 行目に定義があります。

呼び出し関係図:



## 7.1.3.37 ggLoadShader() [1/2]

```
GLuint gg::ggLoadShader (
    const std::array< std::string, 3 > & files,
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列。
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用)。

戻り値

プログラムオブジェクトのプログラム名 (作成できなければ 0)。

gg.h の 5149 行目に定義があります。

呼び出し関係図:



## 7.1.3.38 ggLoadShader() [2/2]

```
GLuint gg::ggLoadShader (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr )
```

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

<i>vert</i>	バーテックスシェーダのソースファイル名。
<i>frag</i>	フラグメントシェーダのソースファイル名 (空文字列なら不使用)。
<i>geom</i>	ジオメトリシェーダのソースファイル名 (空文字列なら不使用)。
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用)。

## 戻り値

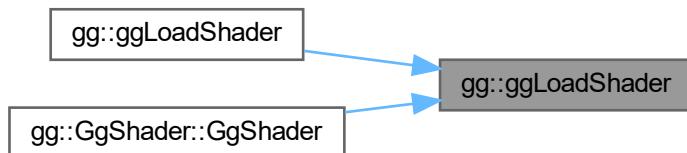
プログラムオブジェクトのプログラム名(作成できなければ 0).

`gg.cpp` の 5000 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.39 `ggLoadSimpleObj()` [1/2]

```

bool gg::ggLoadSimpleObj (
    const std::string & name,
    std::vector< std::array< GLuint, 3 > > & group,
    std::vector< GgSimpleShader::Material > & material,
    std::vector< GgVertex > & vert,
    bool normalize = false )
  
```

三角形分割された OBJ ファイルと MTL ファイルを読み込む (Arrays 形式)

引数

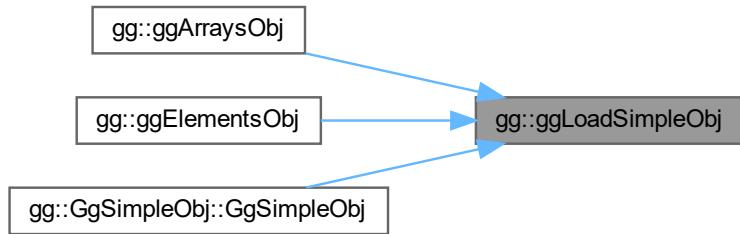
<code>name</code>	読み込む Wavefront OBJ ファイル名.
<code>group</code>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<code>material</code>	読み込んだデータのポリゴングループごとの <code>GgSimpleShader::Material</code> 型の材質.
<code>vert</code>	読み込んだデータの頂点属性.
<code>normalize</code>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

[gg.cpp](#) の 4607 行目に定義があります。

被呼び出し関係図:



### 7.1.3.40 ggLoadSimpleObj() [2/2]

```

bool gg::ggLoadSimpleObj (
    const std::string & name,
    std::vector< std::array< GLuint, 3 > > & group,
    std::vector< GgSimpleShader::Material > & material,
    std::vector< GgVertex > & vert,
    std::vector< GLuint > & face,
    bool normalize = false )
  
```

三角形分割された OBJ ファイルを読み込む (Elements 形式).

引数

<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>face</i>	読み込んだデータの三角形の頂点インデックス.
<i>normalize</i>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

[gg.cpp](#) の 4696 行目に定義があります。

### 7.1.3.41 ggLoadTexture()

```
GLuint gg::ggLoadTexture (
    const GLvoid * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RGB,
    GLenum type = GL_UNSIGNED_BYTE,
    GLenum internal = GL_RGB,
    GLenum wrap = GL_CLAMP_TO_EDGE,
    bool swizzle = true )
```

テクスチャを作成して確保して画像データをテクスチャとして読み込む。

引数

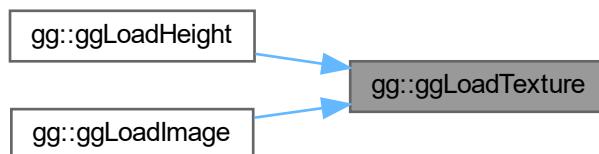
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャの作成のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <code>image</code> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <code>image</code> の縦の画素数.
<i>format</i>	<code>image</code> のフォーマット.
<i>type</i>	<code>image</code> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは <code>GL_CLAMP_TO_EDGE</code> .
<i>swizzle</i>	<code>true</code> ならテクスチャの赤と青を入れ替える, デフォルトは <code>true</code> .

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

`gg.cpp` の 3749 行目に定義があります。

被呼び出し関係図:



## 7.1.3.42 ggLookat() [1/3]

```
GgMatrix gg::ggLookat (
    const GgVector & e,
    const GgVector & t,
    const GgVector & u ) [inline]
```

ビュー変換行列を返す。

引数

<i>e</i>	視点の位置を格納した <code>GgVector</code> 型の変数.
<i>t</i>	目標点の位置を格納した <code>GgVector</code> 型の変数.
<i>u</i>	上方向のベクトルを格納した <code>GgVector</code> 型の変数.

戻り値

求めたビュー変換行列.

`gg.h` の 3320 行目に定義があります。

呼び出し関係図:



## 7.1.3.43 ggLookat() [2/3]

```
GgMatrix gg::ggLookat (
    const GLfloat * e,
    const GLfloat * t,
    const GLfloat * u ) [inline]
```

ビュー変換行列を返す。

引数

<i>e</i>	視点の位置を格納した <code>GLfloat</code> 型の 3 要素の配列変数.
<i>t</i>	目標点の位置を格納した <code>GLfloat</code> 型の 3 要素の配列変数.
<i>u</i>	上方向のベクトルを格納した <code>GLfloat</code> 型の 3 要素の配列変数.

戻り値

求めたビュー変換行列.

[gg.h](#) の 3302 行目に定義があります。

呼び出し関係図:



#### 7.1.3.44 `ggLookat()` [3/3]

```
GgMatrix gg::ggLookat (
    GLfloat ex,
    GLfloat ey,
    GLfloat ez,
    GLfloat tx,
    GLfloat ty,
    GLfloat tz,
    GLfloat ux,
    GLfloat uy,
    GLfloat uz ) [inline]
```

ビュー変換行列を返す.

引数

<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

求めたビュー変換行列.

`gg.h` の 3284 行目に定義があります。

呼び出し関係図:



#### 7.1.3.45 ggMatrixQuaternion() [1/2]

```
GgQuaternion gg::ggMatrixQuaternion (
    const GgMatrix & m ) [inline]
```

回転の変換行列 `m` を表す四元数を返す。

引数

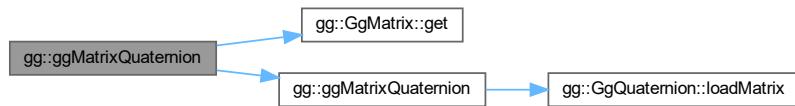
<code>m</code>	<code>GgMatrix</code> 型の変換行列。
----------------	-------------------------------

戻り値

`m` による回転の変換に相当する四元数。

`gg.h` の 4571 行目に定義があります。

呼び出し関係図:



#### 7.1.3.46 ggMatrixQuaternion() [2/2]

```
GgQuaternion gg::ggMatrixQuaternion (
    const GLfloat * a ) [inline]
```

回転の変換行列 `m` を表す四元数を返す。

引数

<code>a</code>	GLfloat 型の 16 要素の配列変数.
----------------	------------------------

戻り値

`a` による回転の変換に相当する四元数.

`gg.h` の 4559 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 7.1.3.47 `ggNorm()`

```
GLfloat gg::ggNorm (
    const GgQuaternion & q ) [inline]
```

四元数のノルムを返す.

引数

<code>q</code>	GgQuaternion 型の四元数.
----------------	---------------------

戻り値

四元数 `q` のノルム.

gg.h の 4725 行目に定義がります。

呼び出し関係図:



### 7.1.3.48 ggNormal()

```
GgMatrix gg::ggNormal (
    const GgMatrix & m ) [inline]
```

法線変換行列を返す.

引数

<code>m</code>	元の変換行列.
----------------	---------

戻り値

`m` の法線変換行列.

gg.h の 3416 行目に定義がります。

呼び出し関係図:



### 7.1.3.49 ggNormalize()

```
GgQuaternion gg::ggNormalize (
    const GgQuaternion & q ) [inline]
```

正規化した四元数を返す.

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

四元数 `q` を正規化した四元数.

`gg.h` の 4736 行目に定義がります。

呼び出し関係図:



### 7.1.3.50 ggNormalize3() [1/4]

```
GgVector gg::ggNormalize3 (
    const GgVector & a ) [inline]
```

`GgVector` 型の 3 要素の正規化.

引数

<code>a</code>	<code>GgVector</code> 型のベクトル.
----------------	-------------------------------

戻り値

`a` の 3 要素を正規化したもの.

覚え書き

戻り値の `w` (第4) 要素は 0 になる.

`gg.h` の 2042 行目に定義がります。

呼び出し関係図:



7.1.3.51 `ggNormalize3()` [2/4]

```
void gg::ggNormalize3 (
    const GLfloat * a,
    GLfloat * b ) [inline]
```

3要素の正規化。

引数

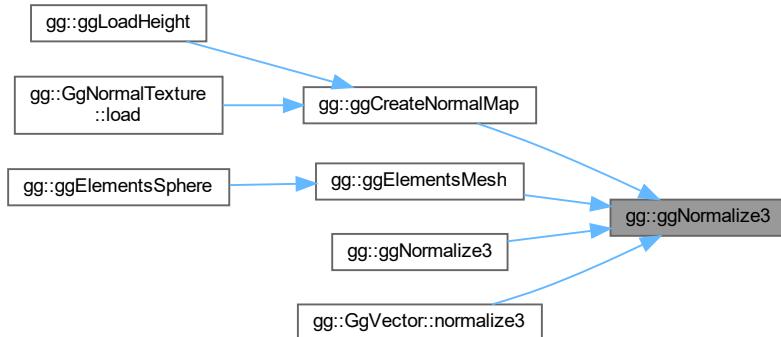
<i>a</i>	GLfloat 型の 3 要素の配列変数。
<i>b</i>	<i>a</i> の 3 要素を正規化した結果を格納する GLfloat 型の 3 要素の配列変数。

`gg.h` の 1478 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

7.1.3.52 `ggNormalize3()` [3/4]

```
void gg::ggNormalize3 (
    GgVector * a ) [inline]
```

`GgVector` 型の 3 要素の正規化。

引数

a	GgVector 型の変数のポインタ.
---	---------------------

覚え書き

a の w (第4) 要素は 0 になる。

gg.h の 2058 行目に定義があります。

呼び出し関係図:



### 7.1.3.53 ggNormalize3() [4/4]

```
void gg::ggNormalize3 (
    GLfloat * a ) [inline]
```

3 要素の正規化。

引数

a	GLfloat 型の 3 要素の配列変数.
---	-----------------------

gg.h の 1492 行目に定義があります。

呼び出し関係図:



### 7.1.3.54 ggNormalize4() [1/4]

```
GgVector gg::ggNormalize4 (
    const GgVector & a )  [inline]
```

GgVector 型の 4 要素の正規化.

引数

a	GgVector 型の変数.
---	----------------

戻り値

a の 4 要素を正規化した結果.

gg.h の 2105 行目に定義があります。

呼び出し関係図:



### 7.1.3.55 ggNormalize4() [2/4]

```
void gg::ggNormalize4 (
    const GLfloat * a,
    GLfloat * b )  [inline]
```

GLfloat 型の 4 要素の正規化.

引数

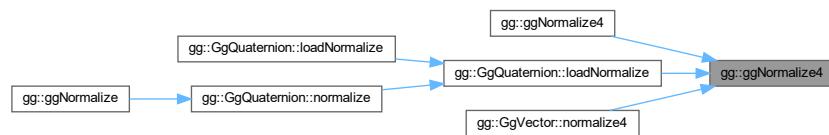
a	GLfloat 型の 4 要素の配列変数.
b	a を正規化した結果を格納する GLfloat 型の 4 要素の配列変数.

gg.h の 1543 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.56 ggNormalize4() [3/4]

```
void gg::ggNormalize4 (
    GgVector * a ) [inline]
```

GgVector 型の 4 要素の正規化。

引数

a	GLfloat 型の 4 要素の配列変数。
---	-----------------------

gg.h の 2117 行目に定義があります。

呼び出し関係図:



### 7.1.3.57 ggNormalize4() [4/4]

```
void gg::ggNormalize4 (
    GLfloat * a ) [inline]
```

GLfloat 型の 4 要素の正規化.

引数

<i>a</i>	正規化する GLfloat 型の 4 要素の配列変数.
----------	-----------------------------

gg.h の 1558 行目に定義があります。

呼び出し関係図:



### 7.1.3.58 ggOrthogonal()

```
GgMatrix gg::ggOrthogonal (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar ) [inline]
```

直交投影変換行列を返す.

引数

<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

求めた直交投影変換行列.

[gg.h](#) の 3341 行目に定義があります。

呼び出し関係図:



### 7.1.3.59 ggPerspective()

```
GgMatrix gg::ggPerspective (
    GLfloat fovy,
    GLfloat aspect,
    GLfloat zNear,
    GLfloat zFar ) [inline]
```

画角を指定して透視投影変換行列を返す.

引数

<i>fovy</i>	y 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

求めた透視投影変換行列.

[gg.h](#) の 3379 行目に定義があります。

呼び出し関係図:



### 7.1.3.60 ggPointsCube()

```
std::shared_ptr< gg::GgPoints > gg::ggPointsCube (
    GLsizei countv,
    GLfloat length = 1.0f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f )
```

点群を立方体状に生成する。

引数

<i>countv</i>	生成する点の数。
<i>length</i>	点群を生成する立方体の一辺の長さ。
<i>cx</i>	点群の中心の x 座標。
<i>cy</i>	点群の中心の y 座標。
<i>cz</i>	点群の中心の z 座標。

戻り値

`GgPoints` 型の ポインタ。

`gg.cpp` の 5161 行目に定義があります。

### 7.1.3.61 ggPointsSphere()

```
std::shared_ptr< gg::GgPoints > gg::ggPointsSphere (
    GLsizei countv,
    GLfloat radius = 0.5f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f )
```

点群を球状に生成する。

引数

<i>countv</i>	生成する点の数。
<i>radius</i>	点群を生成する半径。
<i>cx</i>	点群の中心の x 座標。
<i>cy</i>	点群の中心の y 座標。
<i>cz</i>	点群の中心の z 座標。

戻り値

`GgPoints` 型のポインタ。

`gg.cpp` の 5187 行目に定義がります。

### 7.1.3.62 `ggQuaternion()` [1/2]

```
GgQuaternion gg::ggQuaternion (
    const GLfloat * a ) [inline]
```

四元数を返す。

引数

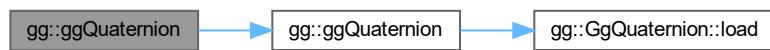
<code>a</code>	GLfloat 型の GLfloat 型の 4 要素の配列変数に格納した四元数。
----------------	--

戻り値

四元数。

`gg.h` の 4537 行目に定義がります。

呼び出し関係図:



### 7.1.3.63 `ggQuaternion()` [2/2]

```
GgQuaternion gg::ggQuaternion (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline]
```

四元数を返す。

引数

<code>x</code>	四元数の x 要素。
<code>y</code>	四元数の y 要素。
<code>z</code>	四元数の z 要素。
<code>w</code>	四元数の w 要素。

戻り値

四元数.

gg.h の 4525 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 7.1.3.64 ggQuaternionMatrix()

```
GgMatrix gg::ggQuaternionMatrix (  
    const GgQuaternion & q ) [inline]
```

四元数 q の回転の変換行列を返す.

引数

<code>q</code>	元の四元数.
----------------	--------

戻り値

四元数 q が表す回転に相当する GgMatrix 型の変換行列.

gg.h の 4582 行目に定義があります。

呼び出し関係図:



### 7.1.3.65 ggQuaternionTransposeMatrix()

```
GgMatrix gg::ggQuaternionTransposeMatrix (
    const GgQuaternion & q ) [inline]
```

四元数  $q$  の回転の転置した変換行列を返す。

引数

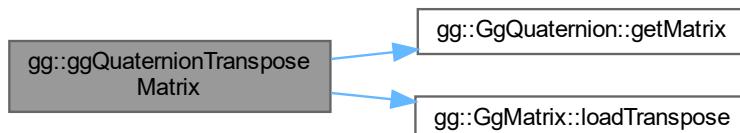
$q$	元の四元数。
-----	--------

戻り値

四元数  $q$  が表す回転に相当する転置した [GgMatrix](#) 型の変換行列。

[gg.h](#) の 4595 行目に定義があります。

呼び出し関係図:



### 7.1.3.66 ggReadImage()

```
bool gg::ggReadImage (
    const std::string & name,
    std::vector< GLubyte > & image,
    GLsizei * pWidth,
    GLsizei * pHeight,
    GLenum * pFormat )
```

TGA ファイル (8/16/24/32bit) をメモリに読み込む。

引数

<i>name</i>	読み込むファイル名。
<i>image</i>	読み込んだデータを格納する vector。
<i>pWidth</i>	読み込んだ画像の横の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない。
<i>pHeight</i>	読み込んだ画像の縦の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない。
<i>pFormat</i>	読み込んだファイルの書式 ( <code>GL_RED</code> , <code>G_RG</code> , <code>GL_RGB</code> , <code>G_RGBA</code> ) の格納先のポインタ, <code>nullptr</code> なら格納しない。

戻り値

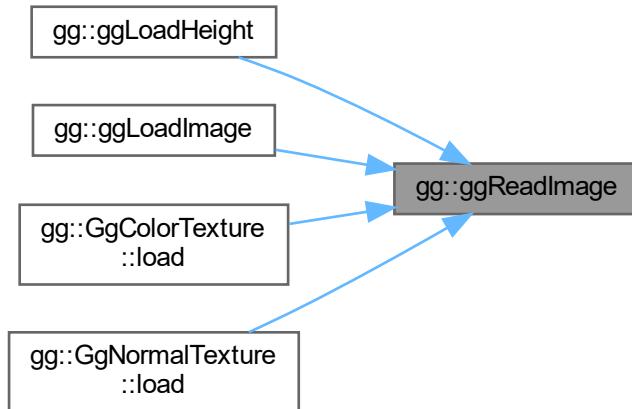
読み込みに成功すれば `true`, 失敗すれば `false`。

`gg.cpp` の 3628 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.67 ggRectangle()

```
std::shared_ptr< gg::GgTriangles > gg::ggRectangle (
    GLfloat width = 1.0f,
    GLfloat height = 1.0f )
```

矩形状に 2 枚の三角形を生成する。

引数

<i>width</i>	矩形の横幅。
<i>height</i>	矩形の高さ。

戻り値

GgTriangles 型のポインタ。

gg.cpp の 5214 行目に定義があります。

### 7.1.3.68 ggRotate() [1/5]

```
GgMatrix gg::ggRotate (
    const GgVector & r ) [inline]
```

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す。

引数

<i>r</i>	回転軸のベクトルと回転角を表す <a href="#">GgVector</a> 型の変数.
----------	--

戻り値

(*r*[0], *r*[1], *r*[2]) を軸に *r*[3] だけ回転する変換行列.

[gg.h](#) の 3264 行目に定義がります。

呼び出し関係図:



### 7.1.3.69 ggRotate() [2/5]

```
GgMatrix gg::ggRotate (
    const GgVector & r,
    GLfloat a ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<i>r</i>	回転軸のベクトルを表す <a href="#">GgVector</a> 型の変数.
<i>a</i>	回転角.

戻り値

*r* を軸に *a* だけ回転する変換行列.

[gg.h](#) の 3240 行目に定義がります。

呼び出し関係図:



### 7.1.3.70 ggRotate() [3/5]

```
GgMatrix gg::ggRotate (
    const GLfloat * r ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<i>r</i>	回転軸のベクトルと回転角を表す GLfloat 型の 4 要素の配列変数.
----------	---------------------------------------

戻り値

(*r*[0], *r*[1], *r*[2]) を軸に *r*[3] だけ回転する変換行列.

[gg.h](#) の 3252 行目に定義があります。

呼び出し関係図:



### 7.1.3.71 ggRotate() [4/5]

```
GgMatrix gg::ggRotate (
    const GLfloat * r,
    GLfloat a ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<i>r</i>	回転軸のベクトルを表す <code>GLfloat</code> 型の 3 要素の配列変数.
<i>a</i>	回転角.

戻り値

*r* を軸に *a* だけ回転する変換行列.

`gg.h` の 3227 行目に定義があります。

呼び出し関係図:



### 7.1.3.72 `ggRotate()` [5/5]

```
GgMatrix gg::ggRotate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a ) [inline]
```

$(x, y, z)$  方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<i>x</i>	回転軸の x 成分.
<i>y</i>	回転軸の y 成分.
<i>z</i>	回転軸の z 成分.
<i>a</i>	回転角.

戻り値

$(x, y, z)$  を軸にさらに *a* 回転する変換行列.

`gg.h` の 3214 行目に定義があります。

呼び出し関係図:



### 7.1.3.73 ggRotateQuaternion() [1/3]

```
GgQuaternion gg::ggRotateQuaternion (
    const GLfloat * v ) [inline]
```

(v[0], v[1], v[2]) を軸として角度 v[3] 回転する四元数を返す.

引数

v	軸ベクトルを表す GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

回転を表す四元数.

gg.h の 4636 行目に定義があります。

呼び出し関係図:



### 7.1.3.74 ggRotateQuaternion() [2/3]

```
GgQuaternion gg::ggRotateQuaternion (
    const GLfloat * v,
    GLfloat a ) [inline]
```

(v[0], v[1], v[2]) を軸として角度 a 回転する四元数を返す.

引数

<i>v</i>	軸ベクトルを表す GLfloat 型の 3 要素の配列変数.
<i>a</i>	回転角.

戻り値

回転を表す四元数.

gg.h の 4625 行目に定義があります。

呼び出し関係図:



### 7.1.3.75 ggRotateQuaternion() [3/3]

```
GgQuaternion gg::ggRotateQuaternion (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a ) [inline]
```

(x, y, z) を軸として角度 a 回転する四元数を返す.

引数

<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

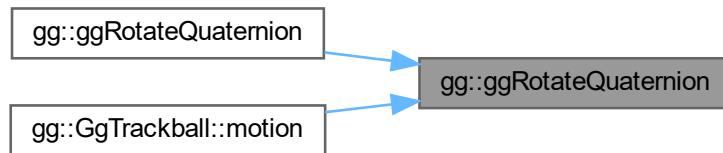
回転を表す四元数.

gg.h の 4612 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.76 ggRotateX()

```
GgMatrix gg::ggRotateX (
    GLfloat a ) [inline]
```

$x$  軸中心の回転の変換行列を返す。

引数

a	回転角.
---	------

戻り値

$x$  軸中心に  $a$  だけ回転する変換行列。

gg.h の 3175 行目に定義があります。

呼び出し関係図:



#### 7.1.3.77 ggRotateY()

```
GgMatrix gg::ggRotateY (  
    GLfloat a ) [inline]
```

y 軸中心の回転の変換行列を返す。

引数

a	回転角。
---	------

戻り値

y 軸中心に a だけ回転する変換行列。

gg.h の 3187 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.78 ggRotateZ()

```
GgMatrix gg::ggRotateZ (
    GLfloat a ) [inline]
```

*z* 軸中心の回転の変換行列を返す。

引数

<i>a</i>	回転角.
----------	------

戻り値

*z* 軸中心に *a* だけ回転する変換行列。

[gg.h](#) の 3199 行目に定義があります。

呼び出し関係図:



### 7.1.3.79 ggSaveColor()

```
bool gg::ggSaveColor (
    const std::string & name )
```

カラーバッファの内容を TGA ファイルに保存する。

引数

<i>name</i>	保存するファイル名.
-------------	------------

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 3572 行目に定義がります。

呼び出し関係図:



#### 7.1.3.80 ggSaveDepth()

```
bool gg::ggSaveDepth (
    const std::string & name )
```

デプスバッファの内容を TGA ファイルに保存する。

引数

<i>name</i>	保存するファイル名。
-------------	------------

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 3598 行目に定義がります。

呼び出し関係図:



#### 7.1.3.81 ggSaveTga()

```
bool gg::ggSaveTga (
    const std::string & name,
```

```
const void * buffer,
unsigned int width,
unsigned int height,
unsigned int depth )
```

配列の内容を TGA ファイルに保存する。

引数

<i>name</i>	保存するファイル名。
<i>buffer</i>	画像データを格納した配列。
<i>width</i>	画像の横の画素数。
<i>height</i>	画像の縦の画素数。
<i>depth</i>	1画素のバイト数。

戻り値

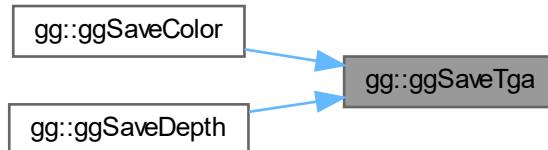
保存に成功すれば `true`, 失敗すれば `false`.

`gg.cpp` の 3482 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.82 `ggScale()` [1/3]

```
GgMatrix gg::ggScale (
    const GgVector & s ) [inline]
```

拡大縮小の変換行列を返す。

引数

<code>s</code>	拡大率の <code>GgVector</code> 型の変数.
----------------	----------------------------------

戻り値

拡大縮小の変換行列.

`gg.h` の 3163 行目に定義がります。

呼び出し関係図:



### 7.1.3.83 `ggScale()` [2/3]

```
GgMatrix gg::ggScale (
    const GLfloat * s ) [inline]
```

拡大縮小の変換行列を返す.

引数

<code>s</code>	拡大率の <code>GLfloat</code> 型の 3 要素の配列変数 ( <code>x, y, z</code> ).
----------------	--

戻り値

拡大縮小の変換行列.

`gg.h` の 3151 行目に定義がります。

呼び出し関係図:



### 7.1.3.84 ggScale() [3/3]

```
GgMatrix gg::ggScale (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f )  [inline]
```

拡大縮小の変換行列を返す。

引数

<i>x</i>	<i>x</i> 方向の拡大率。
<i>y</i>	<i>y</i> 方向の拡大率。
<i>z</i>	<i>z</i> 方向の拡大率。
<i>w</i>	拡大率のスケールファクタ (= 1.0f)。

戻り値

拡大縮小の変換行列。

gg.h の 3139 行目に定義があります。

呼び出し関係図:



### 7.1.3.85 ggSlerp() [1/4]

```
GgQuaternion gg::ggSlerp (
    const GgQuaternion & q,
    const GgQuaternion & r,
    GLfloat t )  [inline]
```

二つの四元数の球面線形補間の結果を返す。

引数

<i>q</i>	GgQuaternion 型の四元数.
<i>r</i>	GgQuaternion 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

*q, r* を *t* で内分した四元数.

gg.h の 4688 行目に定義があります。

呼び出し関係図:



### 7.1.3.86 ggSlerp() [2/4]

```
GgQuaternion gg::ggSlerp (
    const GgQuaternion & q,
    const GLfloat * a,
    GLfloat t ) [inline]
```

二つの四元数の球面線形補間の結果を返す.

引数

<i>q</i>	GgQuaternion 型の四元数.
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

*q, a* を *t* で内分した四元数.

gg.h の 4701 行目に定義があります。

呼び出し関係図:



### 7.1.3.87 ggSlerp() [3/4]

```
GgQuaternion gg::ggSlerp (
    const GLfloat * a,
    const GgQuaternion & q,
    GLfloat t ) [inline]
```

二つの四元数の球面線形補間の結果を返す。

引数

<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
<i>q</i>	GgQuaternion 型の四元数。
<i>t</i>	補間パラメータ。

戻り値

*a, q* を *t* で内分した四元数。

gg.h の 4714 行目に定義があります。

呼び出し関係図:



### 7.1.3.88 ggSlerp() [4/4]

```
GgQuaternion gg::ggSlerp (
    const GLfloat * a,
    const GLfloat * b,
    GLfloat t )  [inline]
```

二つの四元数の球面線形補間の結果を返す。

引数

<i>a</i>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

*a, b* を *t* で内分した四元数.

`gg.h` の 4674 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.89 `ggTranslate()` [1/3]

```
GgMatrix gg::ggTranslate (
    const GgVector & t ) [inline]
```

平行移動の変換行列を返す.

引数

<i>t</i>	移動量の <code>GgVector</code> 型の変数.
----------	----------------------------------

戻り値

平行移動の変換行列.

gg.h の 3124 行目に定義があります。

呼び出し関係図:



### 7.1.3.90 ggTranslate() [2/3]

```
GgMatrix gg::ggTranslate (
    const GLfloat * t ) [inline]
```

平行移動の変換行列を返す.

引数

<code>t</code>	移動量の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------------	--------------------------------------

戻り値

平行移動の変換行列.

gg.h の 3112 行目に定義があります。

呼び出し関係図:



### 7.1.3.91 ggTranslate() [3/3]

```
GgMatrix gg::ggTranslate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f )  [inline]
```

平行移動の変換行列を返す。

引数

<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	移動量のスケールファクタ (= 1.0f).

戻り値

平行移動の変換行列.

[gg.h](#) の 3100 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 7.1.3.92 ggTranspose()

```
GgMatrix gg::ggTranspose (
    const GgMatrix & m ) [inline]
```

転置行列を返す。

引数

<code>m</code>	元の変換行列.
----------------	---------

戻り値

`m` の転置行列.

`gg.h` の 3394 行目に定義があります。

呼び出し関係図:



### 7.1.3.93 operator\*()

```
GgVector gg::operator* (
    GLfloat a,
    const GgVector & b ) [inline]
```

スカラーに `GgVector` 型の各要素を乗じた積を返す.

引数

<code>a</code>	<code>GLfloat</code> 型の値.
<code>b</code>	<code>GgVector</code> 型のベクトル.

戻り値

`a` に `b` の各要素を乗じた積.

`gg.h` の 1964 行目に定義があります。

### 7.1.3.94 operator+() [1/2]

```
const GgVector & gg::operator+ (
    const GgVector & v ) [inline]
```

何もしない.

引数

v	GgVector 型のベクトル.
---	------------------

戻り値

v の値.

gg.h の 1917 行目に定義がります。

### 7.1.3.95 operator+() [2/2]

```
GgVector gg::operator+ (
    GLfloat a,
    const GgVector & b )  [inline]
```

スカラーに GgVector 型の各要素を足した和を返す.

引数

a	GLfloat 型の値.
b	GgVector 型のベクトル.

戻り値

a に b の各要素を足した和.

gg.h の 1929 行目に定義がります。

### 7.1.3.96 operator-() [1/2]

```
const GgVector gg::operator- (
    const GgVector & v )  [inline]
```

符号の反転.

引数

v	GgVector 型のベクトル.
---	------------------

戻り値

$v$  の値の符号を反転した結果.

gg.h の 1940 行目に定義があります。

### 7.1.3.97 operator-() [2/2]

```
GgVector gg::operator- (
    GLfloat a,
    const GgVector & b )  [inline]
```

スカラーから GgVector 型の各要素を引いた差を返す.

引数

<i>a</i>	GLfloat 型の値.
<i>b</i>	GgVector 型のベクトル.

戻り値

$a$  から  $b$  の各要素を引いた差.

gg.h の 1952 行目に定義があります。

### 7.1.3.98 operator/()

```
GgVector gg::operator/ (
    GLfloat a,
    const GgVector & b )  [inline]
```

スカラーを GgVector 型の各要素で割った商を返す.

引数

<i>a</i>	GLfloat 型の値.
<i>b</i>	GgVector 型のベクトル.

戻り値

$a$  を  $b$  の各要素で割った商.

gg.h の 1976 行目に定義があります。

## 7.1.4 変数詳解

### 7.1.4.1 ggBufferAlignment

```
GLint gg::ggBufferAlignment [extern]
```

使用している GPU のバッファアライメント.

使用している GPU のバッファオブジェクトのアライメント, 初期化に取得される.



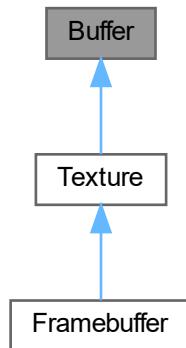
# Chapter 8

## クラス詳解

### 8.1 Buffer クラス

```
#include <Buffer.h>
```

Buffer の継承関係図



### 公開メンバ関数

- `Buffer ()`
- `Buffer (GLsizei width, GLsizei height, int channels)`
- `Buffer (const Buffer &buffer)`
- `Buffer (Buffer &&buffer) noexcept`
- `virtual ~Buffer ()`
- `Buffer & operator= (const Buffer &buffer)`
- `Buffer & operator= (Buffer &&buffer) noexcept`
- `virtual void create (GLsizei width, GLsizei height, int channels)`
- `void resize (GLsizei width, GLsizei height, int channels)`
- `void resize (const Buffer &buffer)`

- virtual void `copy` (const `Buffer` &buffer) noexcept
- virtual void `discard` ()
- auto `getBufferName` () const
- virtual const std::array<GLsizei, 2> & `getSize` () const
- GLsizei `getWidth` () const
- GLsizei `getHeight` () const
- virtual int `getChannels` () const
- auto `getFormat` () const
- auto `getAspect` () const
- void `bindBuffer` (GLenum target=GL\_PIXEL\_PACK\_BUFFER) const
- void `unbindBuffer` (GLenum target=GL\_PIXEL\_PACK\_BUFFER) const
- GLvoid \* `map` () const
- void `unmap` () const
- void `getData` (GLsizei size, GLvoid \*data) const
- void `setData` (GLsizei size, const GLvoid \*data) const

## 限定公開 メンバ関数

- auto `channelsToFormat` (int channels) const
- int `formatToChannels` (GLenum format) const
- void `copyBuffer` (const `Buffer` &buffer) noexcept

### 8.1.1 詳解

バッファクラス

`@description` フレームを格納するピクセルバッファオブジェクト

`Buffer.h` の 24 行目に定義があります。

### 8.1.2 構築子と解体子

#### 8.1.2.1 `Buffer()` [1/4]

`Buffer::Buffer ( )` [inline]

バッファのデフォルトコンストラクタ

`Buffer.h` の 78 行目に定義があります。

#### 8.1.2.2 `Buffer()` [2/4]

```
Buffer::Buffer (
    GLsizei width,
    GLsizei height,
    int channels)
```

バッファを作成するコンストラクタ

引数

<i>width</i>	格納するフレームの横の画素数
<i>height</i>	格納するフレームの縦の画素数
<i>channels</i>	格納するフレームのチャネル数

覚え書き

ピクセルバッファオブジェクトを作成して、そこにフレームのデータを格納する。

Buffer.h の 98 行目に定義がります。

呼び出し関係図:



### 8.1.2.3 Buffer() [3/4]

```
Buffer::Buffer (
    const Buffer & buffer ) [inline]
```

コピー構造

引数

<i>texture</i>	コピー元のバッファ
----------------	-----------

Buffer.h の 108 行目に定義がります。

呼び出し関係図:



### 8.1.2.4 Buffer() [4/4]

```
Buffer::Buffer (
    Buffer && buffer )  [inline], [noexcept]
```

ムーブコンストラクタ

引数

<i>texture</i>	ムーブ元のバッファ
----------------	-----------

[Buffer.h](#) の 118 行目に定義があります。

### 8.1.2.5 ~Buffer()

```
virtual Buffer::~Buffer ( )  [inline], [virtual]
```

デストラクタ

[Buffer.h](#) の 126 行目に定義があります。

呼び出し関係図:



## 8.1.3 関数詳解

### 8.1.3.1 bindBuffer()

```
void Buffer::bindBuffer (
    GLenum target = GL_PIXEL_PACK_BUFFER ) const  [inline]
```

バッファのピクセルバッファオブジェクトを結合する

[Buffer.h](#) の 270 行目に定義があります。

### 8.1.3.2 channelsToFormat()

```
auto Buffer::channelsToFormat (   
    int channels ) const [inline], [protected]
```

チャネル数からフォーマットを求める

引数

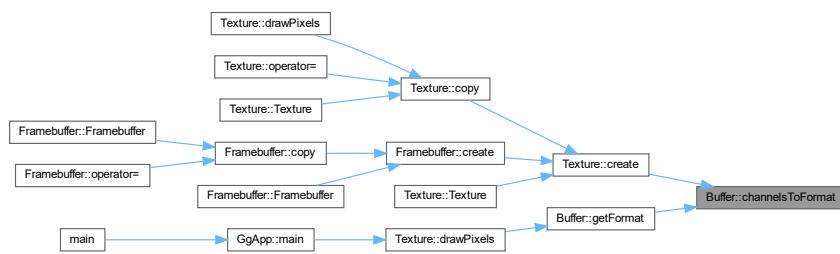
<i>channels</i>	フレームのチャネル数
-----------------	------------

戻り値

テクスチャのフォーマット

[Buffer.h](#) の 49 行目に定義がります。

被呼び出し関係図:



### 8.1.3.3 copy()

```
void Buffer::copy (
    const Buffer & buffer ) [virtual], [noexcept]
```

バッファをコピーする

引数

<i>buffer</i>	コピー元のバッファ
---------------	-----------

覚え書き

このバッファのサイズがコピー元のバッファのサイズと異なれば、このバッファを削除して新しいバッファを作り直してコピーする。

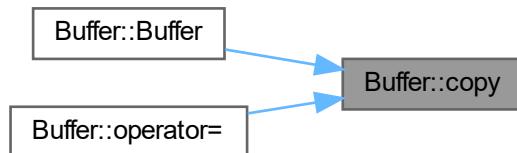
[Framebuffer](#), [Texture](#)で再実装されています。

[Buffer.cpp](#) の 101 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



#### 8.1.3.4 copyBuffer()

```
void Buffer::copyBuffer (
    const Buffer & buffer ) [protected], [noexcept]
```

既存のバッファにコピーする

引数

buffer	コピー元のバッファ
--------	-----------

Buffer.cpp の 36 行目に定義があります。

#### 8.1.3.5 create()

```
void Buffer::create (
    GLsizei width,
    GLsizei height,
    int channels ) [virtual]
```

バッファを作成する

## 引数

<i>width</i>	バッファに格納するフレームの横の画素数
<i>height</i>	バッファに格納するフレームの縦の画素数
<i>channels</i>	バッファに格納するフレームのチャネル数

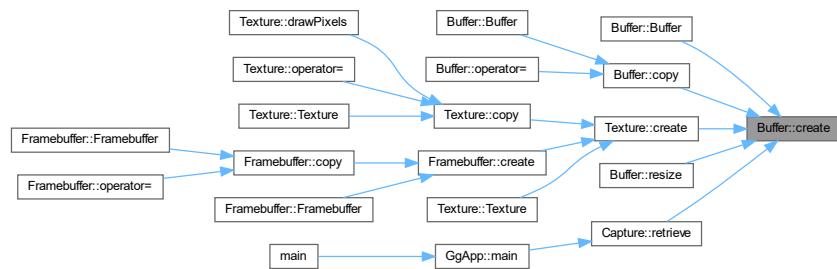
## 覚え書き

このバッファのサイズが引数で指定したサイズと異なれば、このバッファを削除して新しいバッファを作り直す。

[Framebuffer](#), [Texture](#)で再実装されています。

[Buffer.cpp](#) の 68 行目に定義があります。

被呼び出し関係図:



### 8.1.3.6 `discard()`

```
void Buffer::discard ( ) [virtual]
```

バッファを破棄する

[Framebuffer](#), [Texture](#)で再実装されています。

[Buffer.cpp](#) の 242 行目に定義があります。

被呼び出し関係図:



### 8.1.3.7 formatToChannels()

```
int Buffer::formatToChannels (
    GLenum format ) const [protected]
```

フォーマットからチャネル数を求める

引数

<i>format</i>	テクスチャのフォーマット
---------------	--------------

戻り値

フレームのチャネル数

[Buffer.cpp](#) の 13 行目に定義があります。

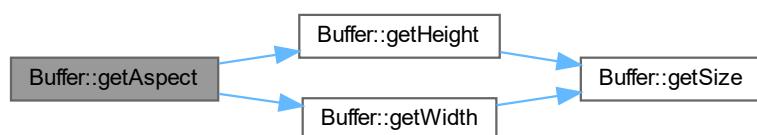
### 8.1.3.8 getAspect()

```
auto Buffer::getAspect () const [inline]
```

格納されているフレームの縦横比を得る

[Buffer.h](#) の 262 行目に定義があります。

呼び出し関係図:



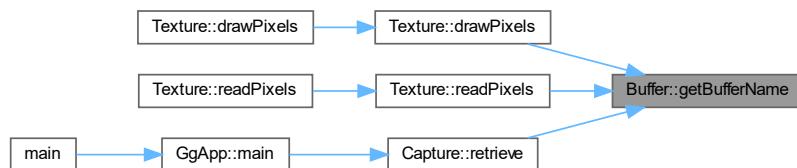
### 8.1.3.9 getBufferName()

```
auto Buffer::getBufferName ( ) const [inline]
```

バッファのピクセルバッファオブジェクト名を得る

[Buffer.h](#) の 211 行目に定義があります。

被呼び出し関係図:



### 8.1.3.10 getChannels()

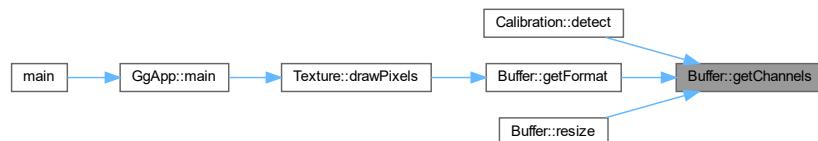
```
virtual int Buffer::getChannels ( ) const [inline], [virtual]
```

格納されているフレームのチャネル数を得る

[Framebuffer](#), [Texture](#) で再実装されています。

[Buffer.h](#) の 246 行目に定義があります。

被呼び出し関係図:



### 8.1.3.11 getData()

```
void Buffer::getData (
    GLsizei size,
    GLvoid * data ) const
```

バッファのピクセルバッファオブジェクトのデータを読み出す

引数

<i>size</i>	読み出すデータのサイズ
<i>data</i>	読み出すデータのポインタ

Buffer.cpp の 183 行目に定義があります。

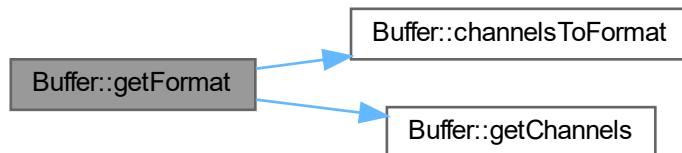
#### 8.1.3.12 getFormat()

```
auto Buffer::getFormat ( ) const [inline]
```

格納されているフレームのフォーマットを得る

Buffer.h の 254 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.1.3.13 getHeight()

```
GLsizei Buffer::getHeight ( ) const [inline]
```

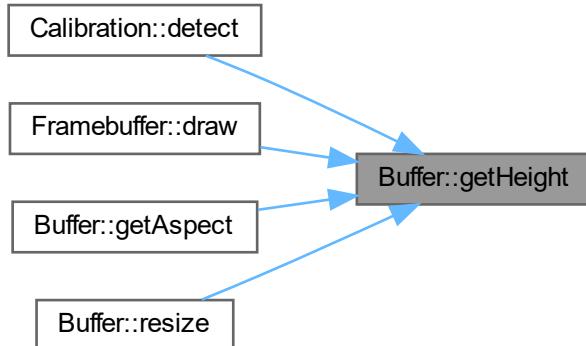
格納されているフレームの縦の画素数を得る

[Buffer.h](#) の 238 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.1.3.14 getSize()

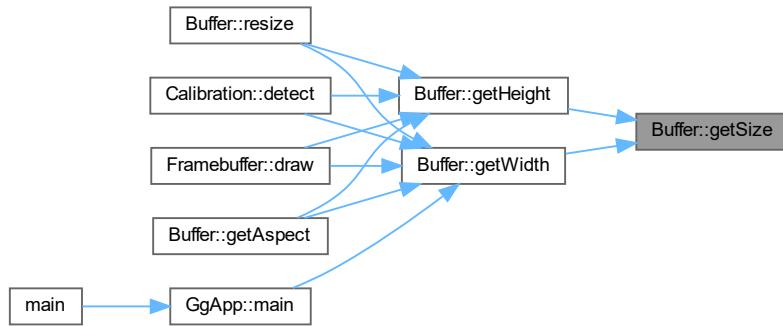
```
virtual const std::array< GLsizei, 2 > & Buffer::getSize ( ) const [inline], [virtual]
```

格納されているフレームのサイズを得る

[Framebuffer](#), [Texture](#)で再実装されています。

[Buffer.h](#) の 222 行目に定義があります。

被呼び出し関係図:



### 8.1.3.15 getWidth()

```
GLsizei Buffer::getWidth ( ) const [inline]
```

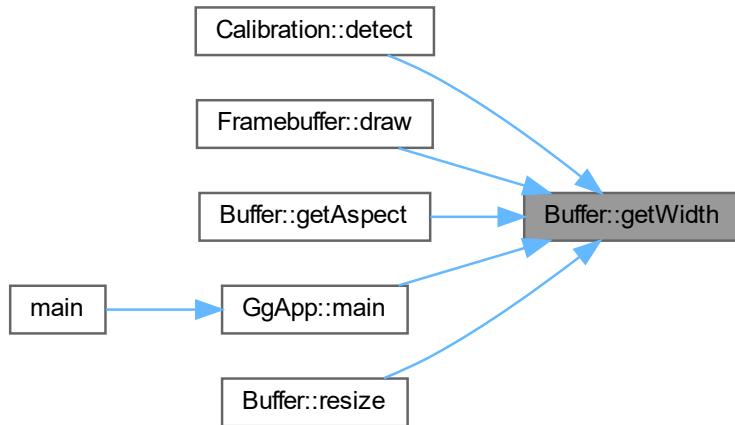
格納されているフレームの横の画素数を得る

Buffer.h の 230 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.1.3.16 map()

`GLvoid * Buffer::map ( ) const`  
バッファのピクセルバッファオブジェクトをマップする

戻り値

ピクセルバッファオブジェクトをマップしたメモリ

`Buffer.cpp` の 149 行目に定義があります。

被呼び出し関係図:



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

```
Buffer & Buffer::operator= (
    Buffer && buffer ) [noexcept]
```

ムーブ代入演算子

引数

buffer	ムーブ代入元のバッファ
--------	-------------

戻り値

ムーブ代入結果のバッファ

[Buffer.cpp](#) の 130 行目に定義があります。

呼び出し関係図:



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

```
Buffer & Buffer::operator= (
    const Buffer & buffer )
```

代入演算子

引数

buffer	代入元のバッファ
--------	----------

戻り値

代入結果のバッファ

[Buffer.cpp](#) の 114 行目に定義があります。

呼び出し関係図:



### 8.1.3.19 `resize()` [1/2]

```
void Buffer::resize (
    const Buffer & buffer ) [inline]
```

オブジェクトが保持するフレームのサイズを引数に指定したオブジェクトと同じにする  
引数

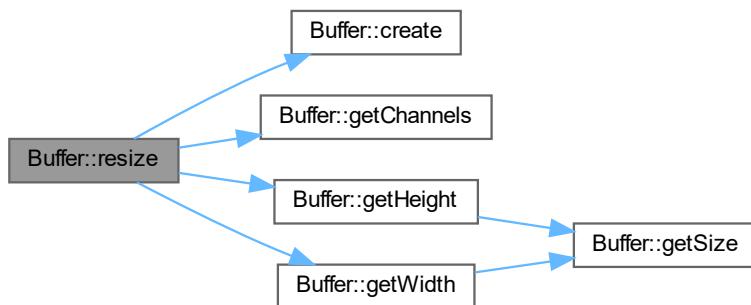
<code>buffer</code>	サイズの基準に用いるオブジェクト
<code>pixels</code>	作成するオブジェクトに格納するフレームのデータのポインタ

覚え書き

このオブジェクトのサイズが引数で指定したオブジェクトのサイズと異なれば、このオブジェクトを削除して新しいオブジェクトを作り直す。

[Buffer.h](#) の 186 行目に定義があります。

呼び出し関係図:



### 8.1.3.20 `resize()` [2/2]

```
void Buffer::resize (
    GLsizei width,
    GLsizei height,
    int channels ) [inline]
```

オブジェクトが保持するフレームのサイズを変更する

引数

<i>width</i>	フレームの横の画素数
<i>height</i>	フレームの縦の画素数
<i>channels</i>	フレームのチャネル数

覚え書き

このオブジェクトのサイズが引数で指定したオブジェクトのサイズと異なれば、このオブジェクトを削除して新しいオブジェクトを作り直す。

[Buffer.h](#) の 171 行目に定義があります。

呼び出し関係図:



### 8.1.3.21 setData()

```
void Buffer::setData (
    GLsizei size,
    const GLvoid * data ) const
```

バッファのピクセルバッファオブジェクトにデータを転送する

引数

<i>size</i>	転送するデータのサイズ
<i>data</i>	転送するデータのポインタ

[Buffer.cpp](#) の 211 行目に定義があります。

### 8.1.3.22 unbindBuffer()

```
void Buffer::unbindBuffer (
    GLenum target = GL_PIXEL_PACK_BUFFER ) const [inline]
```

バッファのピクセルバッファオブジェクトの結合を解除する

[Buffer.h](#) の 280 行目に定義があります。

### 8.1.3.23 unmap()

```
void Buffer::unmap ( ) const
```

バッファのピクセルバッファオブジェクトをアンマップする

[Buffer.cpp](#) の 169 行目に定義があります。

被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [Buffer.h](#)
- [Buffer.cpp](#)

## 8.2 Calibration クラス

```
#include <Calibration.h>
```

### 公開メンバ関数

- [Calibration](#) (const std::string &dictionaryName, const std::array< float, 2 > &length)
- [Calibration](#) (const [Calibration](#) &calibration)=delete
- virtual [~Calibration](#) ()
- [Calibration & operator=](#) (const [Calibration](#) &calibration)=delete
- void [createBoard](#) (const std::array< float, 2 > &length)
- void [setDictionary](#) (const std::string &dictionaryName, const std::array< float, 2 > &length)
- void [drawBoard](#) (cv::Mat &boardImage, int width, int height)
- bool [detect](#) (Buffer &buffer, bool detectBoard)
- const auto [getCornersCount](#) () const
- void [recordCorners](#) ()
- const auto [getCornerCount](#) () const
- void [discardCorners](#) ()
- void [calibrate](#) ()
- const auto & [getCameraMatrix](#) () const
- const auto & [getDistortionCoefficients](#) () const
- auto [getReprojectionError](#) () const
- auto [finished](#) ()
- void [getAllMarkerPoses](#) (std::map< int, GgMatrix > &poses, float markerLength)
- bool [loadParameters](#) (const std::string &filename)
- bool [saveParameters](#) (const std::string &filename) const

## 静的公開変数類

- static const std::map< const std::string, const cv::aruco::PredefinedDictionaryType > [dictionaryList](#)  
*ArUco Marker 辞書のリスト*

### 8.2.1 詳解

#### 較正クラス

[Calibration.h](#) の 26 行目に定義があります。

### 8.2.2 構築子と解体子

#### 8.2.2.1 Calibration() [1/2]

```
Calibration::Calibration (
    const std::string & dictionaryName,
    const std::array< float, 2 > & length )
```

較正オブジェクトのコンストラクタ

引数

<i>dictionaryName</i>	ArUco Marker の辞書名
<i>length</i>	ChArUco Board のマス目の一辺の長さと ArUco Marker の一辺の長さ (単位 cm)

[Calibration.cpp](#) の 23 行目に定義があります。

呼び出し関係図:



#### 8.2.2.2 Calibration() [2/2]

```
Calibration::Calibration (
    const Calibration & calibration ) [delete]
```

コピー構造子は使用しない

引数

<i>calibration</i>	コピー元
--------------------	------

### 8.2.2.3 ~Calibration()

Calibration::~Calibration ( ) [virtual]

デストラクタ

[Calibration.cpp](#) の 34 行目に定義があります。

## 8.2.3 関数詳解

### 8.2.3.1 calibrate()

void Calibration::calibrate ( )

較正する

[Calibration.cpp](#) の 189 行目に定義があります。

被呼び出し関係図:



### 8.2.3.2 createBoard()

```
void Calibration::createBoard (const std::array< float, 2 > & length )
```

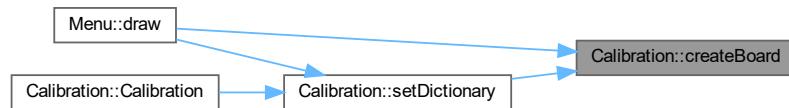
ChArUco Board を作成する

引数

<i>length</i>	ChArUco Board のマス目の一辺の長さと ArUco Marker の一辺の長さ (単位 cm)
---------------	---

Calibration.cpp の 41 行目に定義があります。

被呼び出し関係図:



### 8.2.3.3 `detect()`

```
bool Calibration::detect (
    Buffer & buffer,
    bool detectBoard )
```

ArUco Marker を検出する

引数

<i>buffer</i>	ArUco Marker を検出するフレームを格納したバッファ
<i>detectBoard</i>	ChArUco Board を検出するなら true

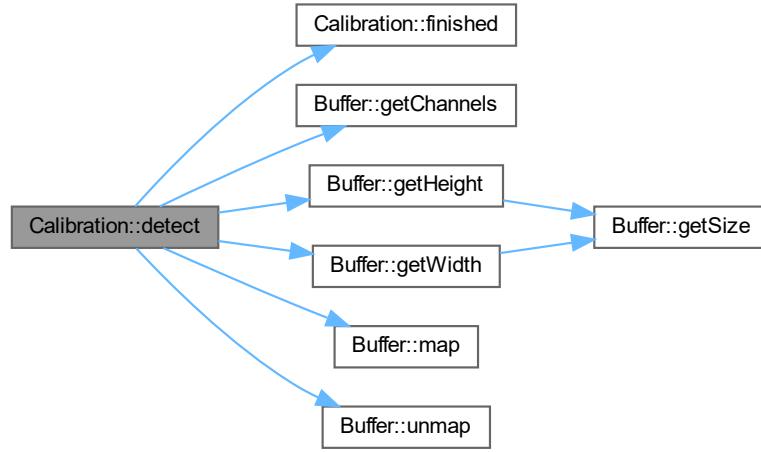
戻り値

ArUco Marker が見つかれば true

マーカの姿勢

Calibration.cpp の 84 行目に定義があります。

呼び出し関係図:



#### 8.2.3.4 `discardCorners()`

```
void Calibration::discardCorners ( )
```

標本を破棄する

[Calibration.cpp](#) の 175 行目に定義があります。

被呼び出し関係図:



#### 8.2.3.5 `drawBoard()`

```
void Calibration::drawBoard (
    cv::Mat & boardImage,
    int width,
    int height )
```

ChArUco Board を描く

引数

<i>boardImage</i>	ChArUco Board の画像の格納先
<i>width</i>	ChArUco Board の横の画素数
<i>height</i>	ChArUco Board の縦の画素数

Calibration.cpp の 76 行目に定義があります。

被呼び出し関係図:



#### 8.2.3.6 finished()

auto Calibration::finished () [inline]

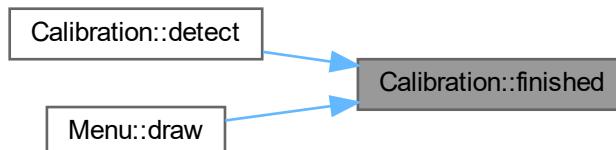
較正が完了したかどうかを調べる

戻り値

較正が完了していたら true

Calibration.h の 202 行目に定義があります。

被呼び出し関係図:



#### 8.2.3.7 getAllMarkerPoses()

```

void Calibration::getAllMarkerPoses (
    std::map< int, GgMatrix > & poses,
    float markerLength )
  
```

ArUco Marker の 3 次元姿勢の変換行列を求める

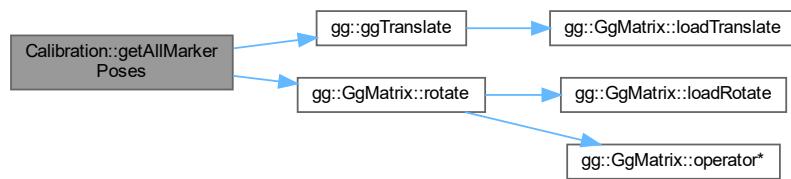
引数

<i>poses</i>	推定した ArUco Marker の 3 次元姿勢
<i>markerLength</i>	ChArUco Board 上の ArUco Marker の一辺の長さ (単位 cm)

マーカの姿勢

[Calibration.cpp](#) の 233 行目に定義があります。

呼び出し関係図:



### 8.2.3.8 getCameraMatrix()

```
const auto & Calibration::getCameraMatrix() const [inline]
```

カメラ行列を取り出す

戻り値

カメラ行列

[Calibration.h](#) の 172 行目に定義があります。

### 8.2.3.9 getCornerCount()

```
const auto Calibration::getCornerCount() const [inline]
```

標本数を取得する

戻り値

保存された標本の数

Calibration.h の 152 行目に定義があります。

被呼び出し関係図:



#### 8.2.3.10 getCornerCount()

```
const auto Calibration::getCornerCount ( ) const [inline]
```

検出数を取得する

戻り値

検出された角の数

Calibration.h の 137 行目に定義があります。

被呼び出し関係図:



#### 8.2.3.11 getDistortionCoefficients()

```
const auto & Calibration::getDistortionCoefficients ( ) const [inline]
```

歪パラメータを取り出す

戻り値

歪パラメータ

Calibration.h の 182 行目に定義があります。

### 8.2.3.12 getReprojectionError()

```
auto Calibration::getReprojectionError ( ) const [inline]
```

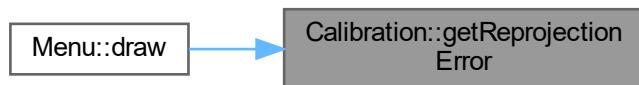
再投影誤差を得る

戻り値

再投影誤差

[Calibration.h](#) の 192 行目に定義があります。

被呼び出し関係図:



### 8.2.3.13 loadParameters()

```
bool Calibration::loadParameters (
    const std::string & filename )
```

ファイルからキャリブレーションパラメータを読み込む

引数

<i>filename</i>	保存するファイル名
-----------------	-----------

戻り値

パラメータの読み込みに成功したら true

[Calibration.cpp](#) の 328 行目に定義があります。

呼び出し関係図:

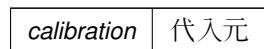


#### 8.2.3.14 operator=( )

```
Calibration & Calibration::operator= (
    const Calibration & calibration ) [delete]
```

代入演算子は使用しない

引数



#### 8.2.3.15 recordCorners( )

```
void Calibration::recordCorners ( )
```

標本を取得する

Calibration.cpp の 153 行目に定義があります。

被呼び出し関係図:



### 8.2.3.16 saveParameters()

```
bool Calibration::saveParameters (
```

```
    const std::string & filename ) const
```

キャリブレーションパラメータをファイルに保存する

引数

<code>filename</code>	保存するファイル名
-----------------------	-----------

戻り値

パラメータの書き込みに成功したら `true`

[Calibration.cpp](#) の 361 行目に定義がります。

呼び出し関係図:



### 8.2.3.17 setDictionary()

```
void Calibration::setDictionary (
    const std::string & dictionaryName,
    const std::array< float, 2 > & length )
```

ArUco Marker の辞書と検出器を設定する

引数

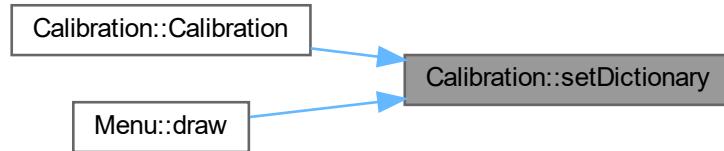
<code>dictionaryName</code>	ArUco Marker の辞書名
<code>length</code>	ChArUco Board のマス目の一边の長さと ArUco Marker の一边の長さ (単位 cm)

[Calibration.cpp](#) の 54 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



## 8.2.4 メンバ詳解

### 8.2.4.1 dictionaryList

```
const std::map< const std::string, const cv::aruco::PredefinedDictionaryType > Calibration::dictionaryList [static]
```

初期値:

```
{
    {"DICT_4X4_50", cv::aruco::DICT_4X4_50 },
    {"DICT_4X4_100", cv::aruco::DICT_4X4_100 },
    {"DICT_4X4_250", cv::aruco::DICT_4X4_250 },
    {"DICT_4X4_1000", cv::aruco::DICT_4X4_1000 },
    {"DICT_5X5_50", cv::aruco::DICT_5X5_50 },
    {"DICT_5X5_100", cv::aruco::DICT_5X5_100 },
    {"DICT_5X5_250", cv::aruco::DICT_5X5_250 },
    {"DICT_5X5_1000", cv::aruco::DICT_5X5_1000 },
    {"DICT_6X6_50", cv::aruco::DICT_6X6_50 },
    {"DICT_6X6_100", cv::aruco::DICT_6X6_100 },
    {"DICT_6X6_250", cv::aruco::DICT_6X6_250 },
    {"DICT_6X6_1000", cv::aruco::DICT_6X6_1000 },
    {"DICT_7X7_50", cv::aruco::DICT_7X7_50 },
    {"DICT_7X7_100", cv::aruco::DICT_7X7_100 },
    {"DICT_7X7_250", cv::aruco::DICT_7X7_250 },
    {"DICT_7X7_1000", cv::aruco::DICT_7X7_1000 },
    {"DICT_ARUCO_ORIGINAL", cv::aruco::DICT_ARUCO_ORIGINAL },
    {"DICT_APRILTAG_16h5", cv::aruco::DICT_APRILTAG_16h5 },
    {"DICT_APRILTAG_25h9", cv::aruco::DICT_APRILTAG_25h9 },
    {"DICT_APRILTAG_36h10", cv::aruco::DICT_APRILTAG_36h10 },
    {"DICT_APRILTAG_36h11", cv::aruco::DICT_APRILTAG_36h11 }
}
```

ArUco Marker 辞書のリスト

[Calibration.h](#) の 232 行目に定義があります。

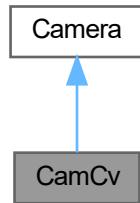
このクラス詳解は次のファイルから抽出されました:

- [Calibration.h](#)
- [Calibration.cpp](#)

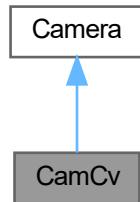
### 8.3 CamCv クラス

```
#include <CamCv.h>
```

CamCv の継承関係図



CamCv 連携図



### 公開 メンバ関数

- `CamCv ()`
- `virtual ~CamCv ()`
- `auto open (int device, int width=0, int height=0, double fps=0.0, const char *fourcc="", int pref=cv::CAP_ANY)`
- `void close ()`
- `auto open (const std::string &file, int width=0, int height=0, double fps=0.0, const char *fourcc="", int pref=cv::CAP_ANY)`
- `virtual double getFps () const`
- `auto getCodec () const`
- `void getCodec (char *fourcc) const`
- `auto getPosition () const`
- `void setPosition (double frame)`
- `void setExposure (double exposure)`
- `void increaseExposure ()`
- `void decreaseExposure ()`
- `void setGain (double gain)`
- `void increaseGain ()`
- `void decreaseGain ()`

その他の継承メンバ

### 8.3.1 詳解

OpenCV を使ってビデオをキャプチャするクラス

[CamCv.h](#) の 17 行目に定義があります。

### 8.3.2 構築子と解体子

#### 8.3.2.1 CamCv()

```
CamCv::CamCv ( ) [inline]
```

コンストラクタ

[CamCv.h](#) の 160 行目に定義があります。

#### 8.3.2.2 ~CamCv()

```
virtual CamCv::~CamCv ( ) [inline], [virtual]
```

デストラクタ

[CamCv.h](#) の 169 行目に定義があります。

### 8.3.3 関数詳解

#### 8.3.3.1 close()

```
void CamCv::close ( ) [inline], [virtual]
```

キャプチャデバイスの使用を終了する

[Camera](#)を再実装しています。

[CamCv.h](#) の 192 行目に定義があります。

呼び出し関係図:



### 8.3.3.2 decreaseExposure()

```
void CamCv::decreaseExposure ( ) [inline], [virtual]
```

露出を一段階下げる

[Camera](#)を再実装しています。

[CamCv.h](#) の 294 行目に定義があります。

呼び出し関係図:



### 8.3.3.3 decreaseGain()

```
void CamCv::decreaseGain ( ) [inline], [virtual]
```

利得を一段階下げる

[Camera](#)を再実装しています。

[CamCv.h](#) の 320 行目に定義があります。

呼び出し関係図:



### 8.3.3.4 `getCodec()` [1/2]

```
auto CamCv::getCodec ( ) const [inline]
```

コードックを調べる

戻り値

使用しているコードックを表す4バイト

[CamCv.h](#) の 232 行目に定義があります。

呼び出し関係図:



### 8.3.3.5 `getCodec()` [2/2]

```
void CamCv::getCodec ( char * fourcc ) const [inline]
```

コードックを調べる

引数

<code>fourcc</code>	使用しているコードックを表す4文字の格納先
---------------------	-----------------------

[CamCv.h](#) の 242 行目に定義があります。

呼び出し関係図:



### 8.3.3.6 getFps()

```
virtual double CamCv::getFps ( ) const [inline], [virtual]
```

キャプチャしたフレームのフレームレートを得る

戻り値

キャプチャしたフレームのフレームレート

[Camera](#)を再実装しています。

[CamCv.h](#) の 222 行目に定義があります。

### 8.3.3.7 getPosition()

```
auto CamCv::getPosition ( ) const [inline]
```

ファイルから入力しているとき現在のフレーム番号を得る

戻り値

現在入力しているフレーム番号

[CamCv.h](#) の 258 行目に定義があります。

### 8.3.3.8 increaseExposure()

```
void CamCv::increaseExposure ( ) [inline], [virtual]
```

露出を一段階上げる

[Camera](#)を再実装しています。

[CamCv.h](#) の 286 行目に定義があります。

呼び出し関係図:



### 8.3.3.9 increaseGain()

```
void CamCv::increaseGain ( ) [inline], [virtual]
```

利得を一段階上げる

[Camera](#)を再実装しています。

[CamCv.h](#) の 312 行目に定義があります。

呼び出し関係図:



### 8.3.3.10 open() [1/2]

```
auto CamCv::open (
    const std::string & file,
    int width = 0,
    int height = 0,
    double fps = 0.0,
    const char * fourcc = "",
    int pref = cv::CAP_ANY ) [inline]
```

ファイル / ネットワーク / [GStreamer](#) から入力する

引数

<i>device</i>	入力するファイルの名前
<i>width</i>	入力するファイルを開く際に期待するフレームの横の画素数, 0ならお任せ
<i>height</i>	入力するファイルを開く際に期待するフレームの縦の画素数, 0ならお任せ
<i>fps</i>	入力するファイルを開く際に期待するフレームフレームレート, 0ならお任せ
<i>fourcc</i>	入力するファイルを開く際に期待するコーデックの 4 文字, ""ならお任せ

戻り値

入力するファイルが使用可能なら true

[CamCv.h](#) の 211 行目に定義があります。

## 8.3.3.11 open() [2/2]

```
auto CamCv::open (
    int device,
    int width = 0,
    int height = 0,
    double fps = 0.0,
    const char * fourcc = "",
    int pref = cv::CAP_ANY ) [inline]
```

キャプチャデバイスを開く

引数

<i>device</i>	キャプチャデバイスの番号
<i>width</i>	キャプチャデバイスを開く際に期待するフレームの横の画素数, 0ならお任せ
<i>height</i>	キャプチャデバイスを開く際に期待するフレームの縦の画素数, 0ならお任せ
<i>fps</i>	キャプチャデバイスを開く際に期待するフレームフレームレート, 0ならお任せ
<i>fourcc</i>	キャプチャデバイスを開く際に期待するコーデックの4文字, ""ならお任せ

戻り値

キャプチャデバイスが使用可能なら true

CamCv.h の 183 行目に定義があります。

## 8.3.3.12 setExposure()

```
void CamCv::setExposure (
    double exposure ) [inline]
```

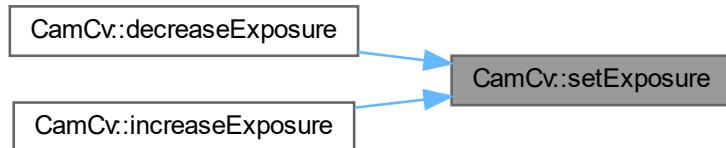
露出を設定する

引数

<i>exposure</i>	設定する露出
-----------------	--------

CamCv.h の 278 行目に定義があります。

被呼び出し関係図:



### 8.3.3.13 setGain()

```
void CamCv::setGain (
    double gain ) [inline]
```

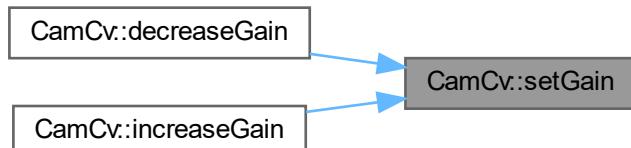
利得を設定する

引数

<i>gain</i>	設定する利得
-------------	--------

[CamCv.h](#) の 304 行目に定義があります。

被呼び出し関係図:



### 8.3.3.14 setPosition()

```
void CamCv::setPosition (
    double frame ) [inline]
```

ファイルから入力しているとき再生位置を指定する

引数

frame	再生位置
-------	------

CamCv.h の 268 行目に定義があります。

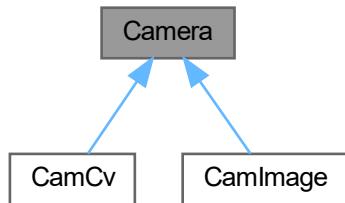
このクラス詳解は次のファイルから抽出されました:

- [CamCv.h](#)

## 8.4 Camera クラス

```
#include <Camera.h>
```

Camera の継承関係図



公開メンバ関数

- [Camera \(\)](#)
- [Camera \(const Camera &camera\)=delete](#)
- [virtual ~Camera \(\)](#)
- [Camera & operator= \(const Camera &camera\)=delete](#)
- [void start \(\)](#)
- [void stop \(\)](#)
- [void transmit \(GLuint buffer\)](#)
- [void transmit \(decltype\(pixels\)&buffer\)](#)
- [virtual void close \(\)](#)
- [auto isRunning \(\) const](#)
- [std::array< int, 2 > getSize \(\) const](#)
- [auto getWidth \(\) const](#)
- [auto getHeight \(\) const](#)
- [auto getChannels \(\) const](#)
- [auto getFrames \(\) const](#)
- [virtual double getFps \(\) const](#)
- [virtual void increaseExposure \(\)](#)
- [virtual void decreaseExposure \(\)](#)
- [virtual void increaseGain \(\)](#)
- [virtual void decreaseGain \(\)](#)

## 公開変数類

- **double in**  
ムービーファイルのインポイント
- **double out**  
ムービーファイルのアウトポイント

## 限定公開メンバ関数

- **void copyFrame ()**
- **virtual void capture ()**

## 限定公開変数類

- **double total**  
ムービーファイルの総フレーム数
- **double interval**  
キャプチャした画像のフレーム間隔
- **int channels**  
キャプチャするムービーのチャネル数
- **cv::Mat frame**  
*OpenCV*のキャプチャデバイスから取得したフレーム
- **std::vector< GLubyte > pixels**  
キャプチャフレームを *GPU*に送るために用いる一時メモリ
- **bool captured**  
新しいフレームが取得されたら *true*
- **std::thread thr**  
キャプチャを非同期に行うためのスレッド
- **std::mutex mtx**  
キャプチャを非同期に行うためのミューテックス
- **bool running**  
キャプチャスレッドが実行中なら *true*

### 8.4.1 詳解

キャプチャデバイス関連の基底クラス

[Camera.h](#) の 39 行目に定義があります。

### 8.4.2 構築子と解体子

#### 8.4.2.1 Camera() [1/2]

```
Camera::Camera ( ) [inline]
```

コンストラクタ

[Camera.h](#) の 125 行目に定義があります。

#### 8.4.2.2 Camera() [2/2]

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

コピーコンストラクタは使用しない

引数

camera	コピー元
--------	------

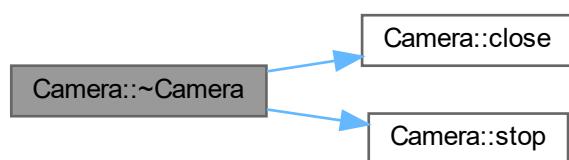
#### 8.4.2.3 ~Camera()

```
virtual Camera::~Camera ( ) [inline], [virtual]
```

デストラクタ

[Camera.h](#) の 146 行目に定義があります。

呼び出し関係図:



### 8.4.3 関数詳解

### 8.4.3.1 capture()

```
virtual void Camera::capture () [inline], [protected], [virtual]
```

フレームをキャプチャする

[Camera.h](#) の 107 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.4.3.2 close()

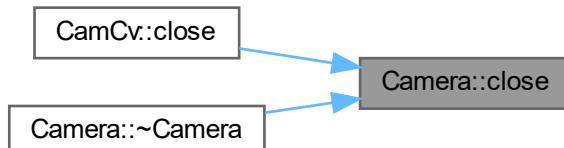
```
virtual void Camera::close () [inline], [virtual]
```

キャプチャデバイスの使用を終了する

[CamCv](#)で再実装されています。

[Camera.h](#) の 241 行目に定義があります。

被呼び出し関係図:



#### 8.4.3.3 copyFrame()

```
void Camera::copyFrame ( ) [inline], [protected]
```

フレームを一時メモリにコピーする

引数

<i>frame</i>	コピーするフレーム
--------------	-----------

[Camera.h](#) の 75 行目に定義があります。

被呼び出し関係図:



#### 8.4.3.4 decreaseExposure()

```
virtual void Camera::decreaseExposure ( ) [inline], [virtual]
```

露出を下げる

[CamCv](#)で再実装されています。

[Camera.h](#) の 325 行目に定義があります。

#### 8.4.3.5 decreaseGain()

```
virtual void Camera::decreaseGain ( ) [inline], [virtual]
```

利得を下げる

[CamCv](#)で再実装されています。

[Camera.h](#) の 335 行目に定義があります。

#### 8.4.3.6 getChannels()

```
auto Camera::getChannels ( ) const [inline]
```

キャプチャしたフレームのチャネル数を調べる

戻り値

キャプチャしたフレームのチャネル数

[Camera.h](#) の 292 行目に定義があります。

#### 8.4.3.7 getFps()

```
virtual double Camera::getFps ( ) const [inline], [virtual]
```

キャプチャデバイスのフレームレートを得る

戻り値

キャプチャデバイスのフレームレート

[CamCv](#)で再実装されています。

[Camera.h](#) の 312 行目に定義があります。

#### 8.4.3.8 getFrames()

```
auto Camera::getFrames ( ) const [inline]
```

ムービーファイルの総フレーム数を得る

戻り値

ムービーファイルの総フレーム数

[Camera.h](#) の 302 行目に定義があります。

#### 8.4.3.9 getHeight()

```
auto Camera::getHeight ( ) const [inline]
```

キャプチャしたフレームの縦の画素数を得る

戻り値

キャプチャ中のフレームの縦の画素数

[Camera.h](#) の 282 行目に定義があります。

#### 8.4.3.10 getSize()

```
std::array< int, 2 > Camera::getSize ( ) const [inline]
```

キャプチャしたフレームのサイズを得る

戻り値

キャプチャしたフレームのサイズ

[Camera.h](#) の 262 行目に定義があります。

#### 8.4.3.11 getWidth()

```
auto Camera::getWidth ( ) const [inline]
```

キャプチャしたフレームの横の画素数を得る

戻り値

キャプチャ中のフレームの横の画素数

[Camera.h](#) の 272 行目に定義があります。

#### 8.4.3.12 increaseExposure()

```
virtual void Camera::increaseExposure ( ) [inline], [virtual]
```

露出を上げる

[CamCv](#)で再実装されています。

[Camera.h](#) の 320 行目に定義があります。

### 8.4.3.13 increaseGain()

```
virtual void Camera::increaseGain ( ) [inline], [virtual]
```

利得を上げる

[CamCv](#)で再実装されています。

[Camera.h](#) の 330 行目に定義があります。

### 8.4.3.14 isRunning()

```
auto Camera::isRunning ( ) const [inline]
```

キャプチャスレッドが実行中かどうか調べる

戻り値

キャプチャ中なら true

[Camera.h](#) の 252 行目に定義があります。

### 8.4.3.15 operator=()

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

代入演算子は使用しない

引数

<i>camera</i>	代入元
---------------	-----

### 8.4.3.16 start()

```
void Camera::start ( ) [inline]
```

キャプチャスレッドを起動する

[Camera.h](#) の 168 行目に定義があります。

呼び出し関係図:



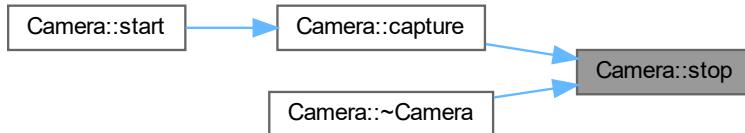
#### 8.4.3.17 stop()

```
void Camera::stop ( ) [inline]
```

キャプチャスレッドを停止する

[Camera.h](#) の 180 行目に定義があります。

被呼び出し関係図:



#### 8.4.3.18 transmit() [1/2]

```
void Camera::transmit (
    decltype(pixels)& buffer ) [inline]
```

キャプチャデバイスをロックしてフレームをメモリに転送する

引数

<code>buffer</code>	転送先のメモリ
---------------------	---------

[Camera.h](#) の 221 行目に定義があります。

#### 8.4.3.19 `transmit()` [2/2]

```
void Camera::transmit (
    GLuint buffer ) [inline]
```

キャプチャデバイスをロックしてフレームをピクセルバッファオブジェクトに転送する

引数

<code>buffer</code>	転送先のピクセルバッファオブジェクト
---------------------	--------------------

[Camera.h](#) の 198 行目に定義があります。

### 8.4.4 メンバ詳解

#### 8.4.4.1 `captured`

```
bool Camera::captured [protected]
```

新しいフレームが取得されたら true

[Camera.h](#) の 59 行目に定義があります。

#### 8.4.4.2 `channels`

```
int Camera::channels [protected]
```

キャプチャするムービーのチャネル数

[Camera.h](#) の 50 行目に定義があります。

#### 8.4.4.3 `frame`

```
cv::Mat Camera::frame [protected]
```

OpenCV のキャプチャデバイスから取得したフレーム

[Camera.h](#) の 53 行目に定義があります。

#### 8.4.4.4 in

```
double Camera::in
```

ムービーファイルのインポイント

[Camera.h](#) の 117 行目に定義があります。

#### 8.4.4.5 interval

```
double Camera::interval [protected]
```

キャプチャした画像のフレーム間隔

[Camera.h](#) の 47 行目に定義があります。

#### 8.4.4.6 mtx

```
std::mutex Camera::mtx [protected]
```

キャプチャを非同期に行うためのミューテックス

[Camera.h](#) の 65 行目に定義があります。

#### 8.4.4.7 out

```
double Camera::out
```

ムービーファイルのアウトポイント

[Camera.h](#) の 120 行目に定義があります。

#### 8.4.4.8 pixels

```
std::vector<GLubyte> Camera::pixels [protected]
```

キャプチャフレームを GPU に送るために用いる一時メモリ

[Camera.h](#) の 56 行目に定義があります。

#### 8.4.4.9 running

```
bool Camera::running [protected]
```

キャプチャスレッドが実行中なら true

[Camera.h](#) の 68 行目に定義があります。

#### 8.4.4.10 thr

```
std::thread Camera::thr [protected]
```

キャプチャを非同期に行うためのスレッド

[Camera.h](#) の 62 行目に定義があります。

#### 8.4.4.11 total

```
double Camera::total [protected]
```

ムービーファイルの総フレーム数

[Camera.h](#) の 44 行目に定義があります。

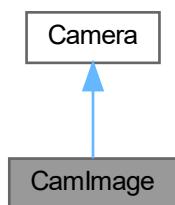
このクラス詳解は次のファイルから抽出されました:

- [Camera.h](#)

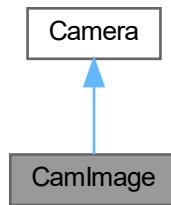
### 8.5 CamImage クラス

```
#include <CamImage.h>
```

CamImage の継承関係図



## CamImage 連携図



## 公開メンバ関数

- [CamImage \(\)](#)
- [CamImage \(std::string &filename, bool flip=false\)](#)
- [virtual ~CamImage \(\)](#)
- [bool open \(const std::string &filename, bool flip=false\)](#)
- [bool isOpened \(\) const](#)

## 静的公開メンバ関数

- [static cv::Mat load \(const std::string &filename\)](#)
- [static bool save \(const std::string &filename, const cv::Mat &image\)](#)

## その他の継承メンバ

## 8.5.1 詳解

OpenCV を使って画像ファイルを読み込むクラス

[CamImage.h](#) の 20 行目に定義があります。

## 8.5.2 構築子と解体子

8.5.2.1 [CamImage\(\)](#) [1/2]

```
CamImage::CamImage ( ) [inline]
```

コンストラクタ

[CamImage.h](#) の 27 行目に定義があります。

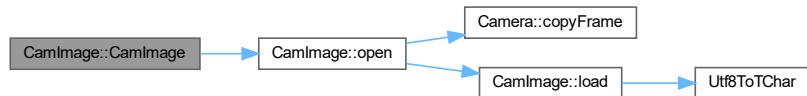
### 8.5.2.2 CamImage() [2/2]

```
CamImage::CamImage (
    std::string & filename,
    bool flip = false ) [inline]
```

画像ファイルを開いて読み込むコンストラクタ

[CamImage.h](#) の 34 行目に定義があります。

呼び出し関係図:



### 8.5.2.3 ~CamImage()

```
virtual CamImage::~CamImage ( ) [inline], [virtual]
```

デストラクタ

[CamImage.h](#) の 42 行目に定義があります。

## 8.5.3 関数詳解

### 8.5.3.1 isOpened()

```
bool CamImage::isOpened ( ) const [inline]
```

画像ファイルが開けたかどうか調べる

戻り値

ファイルが開けていたら true

[CamImage.h](#) の 78 行目に定義があります。

### 8.5.3.2 load()

```
static cv::Mat CamImage::load (
    const std::string & filename ) [inline], [static]
```

画像ファイルを読み込む

引数

<i>filename</i>	読み込む画像ファイルのパス
-----------------	---------------

戻り値

読み込んだ画像

[CamImage.h](#) の 90 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.5.3.3 open()

```
bool CamImage::open (
    const std::string & filename,
    bool flip = false ) [inline]
```

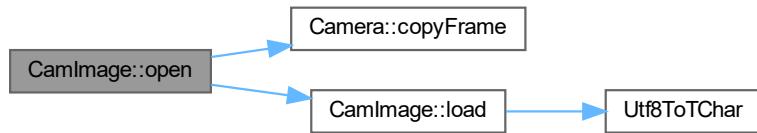
画像ファイルを開く

引数

<i>filename</i>	画像ファイル名
<i>flip</i>	上下を反転するときは true

[CamImage.h](#) の 52 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.5.3.4 save()

```
static bool CamImage::save (
    const std::string & filename,
    const cv::Mat & image ) [inline], [static]
```

配列に格納されている画像をファイルに保存する

引数

<i>filename</i>	保存する画像ファイルのパス
<i>image</i>	保存する画像を保持している配列

戻り値

保存に成功したら true

CamImage.h の 130 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [CamImage.h](#)

## 8.6 Capture クラス

```
#include <Capture.h>
```

### 公開メンバ関数

- [Capture \(\)](#)
- [Capture \(const std::string &filename\)](#)
- [virtual ~Capture \(\)](#)
- [bool openImage \(const std::string &filename\)](#)
- [bool openMovie \(const std::string &filename, cv::VideoCaptureAPIs backend=cv::CAP\\_FFMPEG\)](#)
- [bool openDevice \(int deviceNumber, std::array< int, 2 > &size, double fps, cv::VideoCaptureAPIs backend=cv::CAP\\_FFMPEG, const char \\*fourcc=""\)](#)
- [void start \(\)](#)
- [void stop \(\)](#)
- [void close \(\)](#)
- [`operator bool \(\) const`](#)
- [`bool isOpened \(\) const`](#)
- [`std::array< int, 2 > getSize \(\) const`](#)
- [`double getFps \(\) const`](#)
- [`void retrieve \(Buffer &buffer\)`](#)

## 8.6.1 詳解

キャプチャクラス

[Capture.h](#) の 23 行目に定義があります。

## 8.6.2 構築子と解体子

### 8.6.2.1 Capture() [1/2]

```
Capture::Capture ( ) [inline]
```

キャプチャオブジェクトのデフォルトコンストラクタ

[Capture.h](#) の 33 行目に定義があります。

### 8.6.2.2 Capture() [2/2]

```
Capture::Capture ( const std::string & filename ) [inline]
```

キャプチャするファイルを指定するコンストラクタ

引数

<i>filename</i>	キャプチャするファイルのパス名
-----------------	-----------------

[Capture.h](#) の 42 行目に定義があります。

呼び出し関係図:



### 8.6.2.3 ~Capture()

```
virtual Capture::~Capture () [inline], [virtual]
```

デストラクタ

[Capture.h](#) の 50 行目に定義がります。

呼び出し関係図:



## 8.6.3 関数詳解

### 8.6.3.1 close()

```
void Capture::close ()
```

キャプチャデバイスを閉じる

[Capture.cpp](#) の 96 行目に定義がります。

被呼び出し関係図:



### 8.6.3.2 getFps()

```
double Capture::getFps ( ) const
キャプチャデバイスのフレームレートを得る
戻り値
キャプチャデバイスのフレームレート
```

[Capture.cpp](#) の 118 行目に定義があります。

### 8.6.3.3 getSize()

```
std::array< int, 2 > Capture::getSize ( ) const
キャプチャデバイスのフレームの解像度を得る
戻り値
キャプチャデバイスのフレームの解像度
```

[Capture.cpp](#) の 109 行目に定義があります。

被呼び出し関係図:



### 8.6.3.4 isOpend()

```
bool Capture::isOpend ( ) const [inline]
キャプチャデバイスが有効かどうか
Capture.h の 116 行目に定義があります。
```

### 8.6.3.5 openDevice()

```
bool Capture::openDevice (
    int deviceNumber,
    std::array< int, 2 > & size,
    double fps,
    cv::VideoCaptureAPIs backend = cv::CAP_FFMPEG,
    const char * fourcc = "" )
```

キャプチャデバイスを開く

引数

<i>deviceNumber</i>	開くデバイス番号
<i>size</i>	キャプチャデバイスのフレームの解像度
<i>fps</i>	キャプチャデバイスのフレームレート
<i>backend</i>	バックエンドの種類
<i>fourcc</i>	コーデックの 4 文字

戻り値

開くことができたら true

[Capture.cpp](#) の 54 行目に定義があります。

### 8.6.3.6 openImage()

```
bool Capture::openImage (
    const std::string & filename )
```

画像ファイルを開く

引数

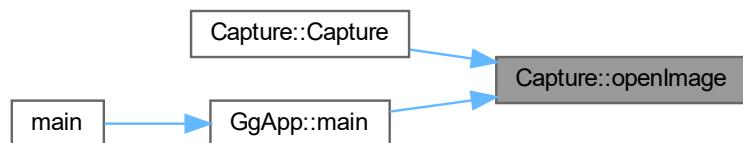
開く画像ファイル名	<input type="text"/>
-----------	----------------------

戻り値

開くことができたら true

[Capture.cpp](#) の 13 行目に定義があります。

被呼び出し関係図:



### 8.6.3.7 openMovie()

```
bool Capture::openMovie (
    const std::string & filename,
    cv::VideoCaptureAPIs backend = cv::CAP_FFMPEG )
```

動画ファイルを開く

引数

<i>filename</i>	開く動画ファイル名
<i>backend</i>	バックエンドの種類

戻り値

開くことができたら true

[Capture.cpp](#) の 33 行目に定義があります。

### 8.6.3.8 operator bool()

```
Capture::operator bool () const [inline], [explicit]
```

キャプチャ中か否か

戻り値

キャプチャ中なら true

[Capture.h](#) の 108 行目に定義があります。

### 8.6.3.9 retrieve()

```
void Capture::retrieve (
    Buffer & buffer )
```

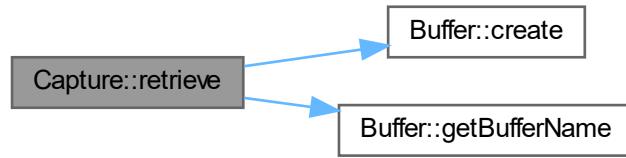
フレームを取得する

引数

<i>buffer</i>	取得したフレームを格納するバッファ
---------------	-------------------

[Capture.cpp](#) の 127 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.6.3.10 start()

```
void Capture::start ( )
```

キャプチャ開始

[Capture.cpp](#) の 78 行目に定義があります。

被呼び出し関係図:



### 8.6.3.11 stop()

```
void Capture::stop ( )
```

キャプチャ終了

[Capture.cpp](#) の 87 行目に定義があります。

被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [Capture.h](#)
- [Capture.cpp](#)

## 8.7 Config クラス

```
#include <Config.h>
```

### 公開メンバ関数

- [Config \(const std::string &filename\)](#)
- [virtual ~Config \(\)](#)
- [void initialize \(\)](#)
- [bool load \(const pathString &filename\)](#)
- [bool save \(const pathString &filename\) const](#)
- [const auto & getTitle \(\) const](#)
- [auto getWidth \(\) const](#)
- [auto getHeight \(\) const](#)
- [const auto & getInitialImage \(\) const](#)
- [const auto & getDictionaryName \(\) const](#)
- [const auto & getCheckerLength \(\) const](#)
- [const auto & getDeviceList \(cv::VideoCaptureAPIs api\) const](#)
- [auto getDeviceCount \(cv::VideoCaptureAPIs api\) const](#)
- [const auto & getDeviceName \(cv::VideoCaptureAPIs api, int number\) const](#)

### フレンド

- [class Menu](#)

プライベートメンバは [Menu](#) クラスで設定する

### 8.7.1 詳解

構成データクラス

[Config.h](#) の 74 行目に定義があります。

### 8.7.2 構築子と解体子

#### 8.7.2.1 Config()

```
Config::Config (
    const std::string & filename )
```

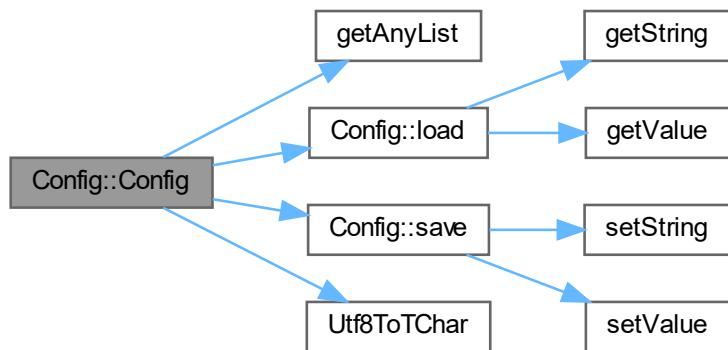
コンストラクタ

引数

<i>filename</i>	構成データの初期化に用いる JSON 形式のファイル名
-----------------	-----------------------------

[Config.cpp](#) の 192 行目に定義があります。

呼び出し関係図:



#### 8.7.2.2 ~Config()

```
Config::~Config ( ) [virtual]
```

デストラクタ

[Config.cpp](#) の 252 行目に定義があります。

## 8.7.3 関数詳解

### 8.7.3.1 getCheckerLength()

```
const auto & Config::getCheckerLength ( ) const [inline]
```

ChArUco Board のマス目の一辺の長さと ArUco Marker の一辺の長さを得る

[Config.h](#) の 198 行目に定義があります。

### 8.7.3.2 getDeviceCount()

```
auto Config::getDeviceCount (
    cv::VideoCaptureAPIs api ) const [inline]
```

キャプチャデバイスの数を調べる

引数

<i>api</i>	使用しているバックエンドの API 名
------------	---------------------

戻り値

キャプチャデバイスの数

[Config.h](#) の 220 行目に定義があります。

被呼び出し関係図:



### 8.7.3.3 getDeviceList()

```
const auto & Config::getDeviceList (
    cv::VideoCaptureAPIs api ) const [inline]
```

キャプチャデバイスのリストを取り出す

引数

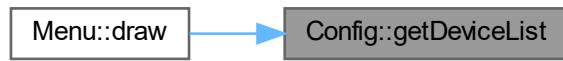
<i>api</i>	使用しているバックエンドの API 名
------------	---------------------

戻り値

キャプチャデバイスのリスト

[Config.h](#) の 209 行目に定義があります。

被呼び出し関係図:



#### 8.7.3.4 getDeviceName()

```
const auto & Config::getDeviceName (
    cv::VideoCaptureAPIs api,
    int number ) const [inline]
```

キャプチャデバイスの名前を調べる

引数

<i>api</i>	使用しているバックエンドの API 名
<i>number</i>	キャプチャデバイスの番号

戻り値

キャプチャデバイスの名前

[Config.h](#) の 232 行目に定義があります。

被呼び出し関係図:



### 8.7.3.5 getDictionaryName()

```
const auto & Config::getDictionaryName ( ) const [inline]
```

ArUco Marker の辞書名を得る

[Config.h](#) の 190 行目に定義があります。

### 8.7.3.6 getHeight()

```
auto Config::getHeight ( ) const [inline]
```

ウィンドウの高さを得る

戻り値

ウィンドウの高さ

[Config.h](#) の 174 行目に定義があります。

### 8.7.3.7 getInitialImage()

```
const auto & Config::getInitialImage ( ) const [inline]
```

初期画像ファイル名を得る

[Config.h](#) の 182 行目に定義があります。

### 8.7.3.8 getTitle()

```
const auto & Config::getTitle () const [inline]
```

ウィンドウタイトルの文字列を得る

戻り値

ウィンドウのタイトル文字列

[Config.h](#) の 154 行目に定義があります。

### 8.7.3.9 getWidth()

```
auto Config::getWidth () const [inline]
```

ウィンドウの横幅を得る

戻り値

ウィンドウの横幅

[Config.h](#) の 164 行目に定義があります。

### 8.7.3.10 initialize()

```
void Config::initialize ()
```

構成データを初期化する

覚え書き

OpenGL のレンダリングコンテキスト作成後に実行する

[Config.cpp](#) の 259 行目に定義があります。

### 8.7.3.11 load()

```
bool Config::load (
    const pathString & filename )
```

構成ファイルを読み込む

引数

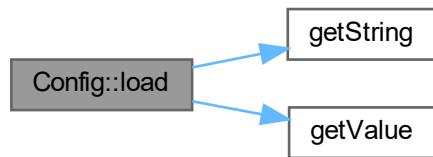
<i>filename</i>	構成データの JSON 形式のファイル名
-----------------	----------------------

戻り値

構成ファイルの読み込みに成功したら true

[Config.cpp](#) の 271 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.7.3.12 save()

```
bool Config::save (
    const pathString & filename ) const
```

構成ファイルを保存する

引数

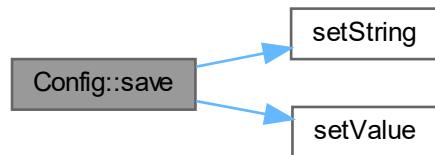
<i>filename</i>	構成データの JSON 形式のファイル名
-----------------	----------------------

戻り値

構成ファイルの保存に成功したら true

[Config.cpp](#) の 341 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



## 8.7.4 フレンドと関連関数の詳解

### 8.7.4.1 Menu

```
friend class Menu [friend]
```

プライベートメンバは `Menu` クラスで設定する

[Config.h](#) の 77 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [Config.h](#)
- [Config.cpp](#)

## 8.8 Expand クラス

```
#include <Expand.h>
```

### 公開メンバ関数

- `Expand (const std::string &vert, const std::string &frag)`
- `Expand (const Expand &shader)=delete`
- `virtual ~Expand ()`
- `Expand & operator= (const Expand &shader)=delete`
- `auto getProgram () const`
- `std::array< GLsizei, 2 > setup (int samples, GLfloat aspect, const gg::GgMatrix &pose, const std::array< GLfloat, 2 > &fov, const std::array< GLfloat, 2 > &center, GLfloat focal, const std::array< GLfloat, 4 > &border, int unit=0) const`

### 8.8.1 詳解

展開用シェーダクラス

`Expand.h` の 17 行目に定義があります。

### 8.8.2 構築子と解体子

#### 8.8.2.1 `Expand()` [1/2]

```
Expand::Expand (
    const std::string & vert,
    const std::string & frag )
```

コンストラクタ

引数

<code>vert</code>	バーテックスシェーダのソースファイル名
<code>frag</code>	フレグメントシェーダのソースファイル名

`Expand.cpp` の 12 行目に定義があります。

#### 8.8.2.2 `Expand()` [2/2]

```
Expand::Expand (
    const Expand & shader ) [delete]
```

コピー・コンストラクタは使用しない

引数

<code>shader</code>	コピー元のシェーダ
---------------------	-----------

### 8.8.2.3 ~Expand()

`Expand::~Expand () [virtual]`

デストラクタ

[Expand.cpp](#) の 29 行目に定義があります。

## 8.8.3 関数詳解

### 8.8.3.1 getProgram()

`auto Expand::getProgram () const [inline]`

展開用シェーダのプログラムオブジェクトを得る

戻り値

展開用シェーダのプログラムオブジェクト

[Expand.h](#) の 77 行目に定義があります。

### 8.8.3.2 operator=()

`Expand & Expand::operator= (`  
`const Expand & shader) [delete]`

代入演算子は使用しない

引数

<code>shader</code>	代入元のシェーダ
---------------------	----------

### 8.8.3.3 setup()

```
std::array< int, 2 > Expand::setup (
    int samples,
    GLfloat aspect,
    const gg::GgMatrix & pose,
    const std::array< GLfloat, 2 > & fov,
    const std::array< GLfloat, 2 > & center,
    GLfloat focal,
    const std::array< GLfloat, 4 > & border,
    int unit = 0 ) const
```

展開

引数

<i>samples</i>	展開するテクスチャをサンプリングする数
<i>aspect</i>	展開するテクスチャの縦横比
<i>pose</i>	サンプリングに用いるカメラの姿勢
<i>fov</i>	サンプリングに用いるカメラの相対画角（単位は度）
<i>center</i>	サンプリングに用いるカメラの撮像面上の中心(主点) 位置 @oaram focal サンプリングに用いるカメラの主点とスクリーンの距離
<i>border</i>	展開後のフレームの境界色
<i>unit</i>	テクスチャユニット番号

戻り値

描画すべきメッシュの横と縦の格子点数

覚え書き

展開するテクスチャをマッピングするメッシュの解像度を *samples* と *aspect* から個々のメッシュが正方形に近くなるよう決定し、それをもとに格子の間隔を求める

展開

[Expand.cpp](#) の 37 行目に定義があります。

呼び出し関係図:



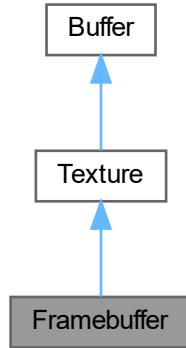
このクラス詳解は次のファイルから抽出されました:

- [Expand.h](#)
- [Expand.cpp](#)

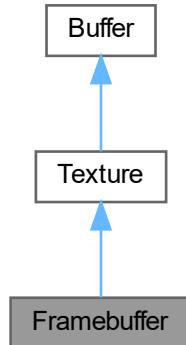
## 8.9 Framebuffer クラス

```
#include <Framebuffer.h>
```

Framebuffer の継承関係図



Framebuffer 連携図



公開メンバ関数

- [Framebuffer \(\)](#)
- [Framebuffer \(GLsizei width, GLsizei height, int channels=3, GLenum attachment=GL\\_COLOR\\_ATTACHMENT0\)](#)
- [Framebuffer \(const Framebuffer &framebuffer\)](#)
- [Framebuffer \(Framebuffer &&framebuffer\) noexcept](#)
- [virtual ~Framebuffer \(\)](#)

- `Framebuffer & operator= (const Framebuffer &framebuffer)`
- `Framebuffer & operator= (Framebuffer &&framebuffer) noexcept`
- `virtual void create (GLsizei width, GLsizei height, int channels)`
- `virtual void copy (const Buffer &framebuffer) noexcept`
- `virtual void discard ()`
- `auto getFramebufferName () const`
- `virtual const std::array< GLsizei, 2 > & getSize () const`
- `virtual int getChannels () const`
- `void bindFramebuffer ()`
- `void unbindFramebuffer () const`
- `void update (const std::array< int, 2 > &size)`
- `void update (const std::array< int, 2 > &size, const Texture &frame, int unit=0)`
- `void draw (GLsizei width, GLsizei height) const`

その他の継承メンバ

### 8.9.1 詳解

フレームバッファオブジェクトクラス

`Framebuffer.h` の 20 行目に定義があります。

### 8.9.2 構築子と解体子

#### 8.9.2.1 `Framebuffer()` [1/4]

```
Framebuffer::Framebuffer ( ) [inline]
```

デフォルトコンストラクタ

`Framebuffer.h` の 54 行目に定義があります。

#### 8.9.2.2 `Framebuffer()` [2/4]

```
Framebuffer::Framebuffer (
    GLsizei width,
    GLsizei height,
    int channels = 3,
    GLenum attachment = GL_COLOR_ATTACHMENT0 )
```

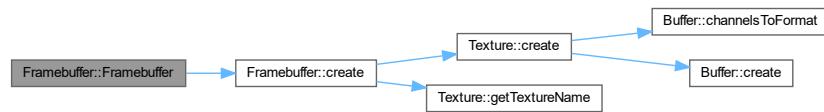
フレームバッファオブジェクトを作成するコンストラクタ

引数

<i>width</i>	作成するフレームバッファオブジェクトの横の画素数
<i>height</i>	作成するフレームバッファオブジェクトの縦の画素数
<i>channels</i>	作成するフレームバッファオブジェクトのチャネル数
<i>attachment</i>	フレームバッファオブジェクトのカラー・バッファのアタッチメント

Framebuffer.cpp の 28 行目に定義がります。

呼び出し関係図:



### 8.9.2.3 Framebuffer() [3/4]

```
Framebuffer::Framebuffer (
    const Framebuffer & framebuffer )
```

コピー・コンストラクタ

引数

<i>framebuffer</i>	コピー元のフレームバッファオブジェクト
--------------------	---------------------

Framebuffer.cpp の 41 行目に定義がります。

呼び出し関係図:



### 8.9.2.4 Framebuffer() [4/4]

```
Framebuffer::Framebuffer (
    Framebuffer && framebuffer ) [noexcept]
```

ムーブコンストラクタ

引数

<code>framebuffer</code>	ムーブ元のフレームバッファオブジェクト
--------------------------	---------------------

ムーブコンストラクタ

引数

<code>framebuffer</code>	ムーブ元
--------------------------	------

[Framebuffer.cpp](#) の 53 行目に定義があります。

### 8.9.2.5 ~Framebuffer()

`Framebuffer::~Framebuffer () [virtual]`

デストラクタ

[Framebuffer.cpp](#) の 61 行目に定義があります。

呼び出し関係図:



## 8.9.3 関数詳解

### 8.9.3.1 bindFramebuffer()

`void Framebuffer::bindFramebuffer ()`

レンダリング先をフレームバッファオブジェクトに切り替える

## 覚え書き

この後から [unbindFramebuffer\(\)](#) の前まで、レンダリング先がフレームバッファオブジェクトになる。この時点のビューポートを保存する。この間でレンダリング処理を実行する。

[Framebuffer.cpp](#) の 170 行目に定義があります。

呼び出し関係図:



## 8.9.3.2 copy()

```
void Framebuffer::copy (
    const Buffer & framebuffer ) [virtual], [noexcept]
```

フレームバッファオブジェクトをコピーする

引数

<code>framebuffer</code>	コピー元のフレームバッファオブジェクト
--------------------------	---------------------

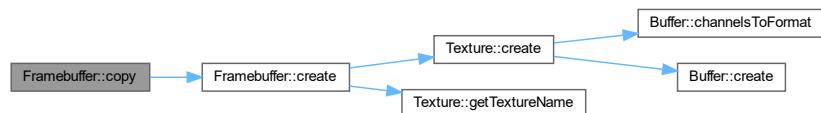
## 覚え書き

このフレームバッファオブジェクトのサイズがコピー元と異なれば、このフレームバッファオブジェクトを削除して、新しいフレームバッファオブジェクトを作り直してコピーする。引数の型が `Buffer` なのは、このオブジェクトの型が `Buffer` のとき [Buffer::copy\(\)](#)、`Texture` のとき [Texture::copy\(\)](#)、`Framebuffer` のとき [Framebuffer::copy\(\)](#) を呼ぶようにするため。

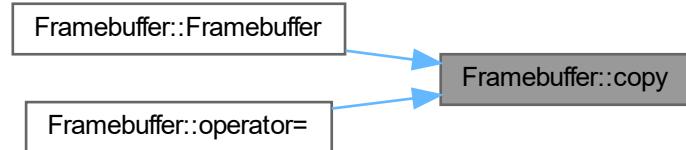
`Texture`を再実装しています。

[Framebuffer.cpp](#) の 157 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.9.3.3 create()

```
void Framebuffer::create (
    GLsizei width,
    GLsizei height,
    int channels ) [virtual]
```

フレームバッファオブジェクトを作成する

引数

<i>width</i>	作成するフレームバッファオブジェクトの横の画素数
<i>height</i>	作成するフレームバッファオブジェクトの縦の画素数
<i>channels</i>	作成するフレームバッファオブジェクトのチャネル数

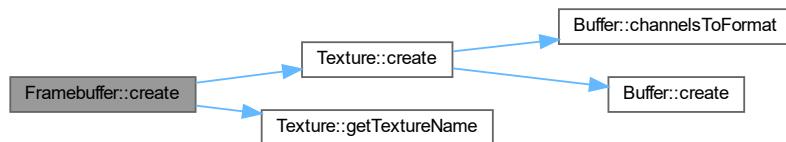
覚え書き

このフレームバッファオブジェクトのサイズが引数で指定したサイズと異なれば、このフレームバッファオブジェクトを削除して、新しいフレームバッファオブジェクトを作り直す。

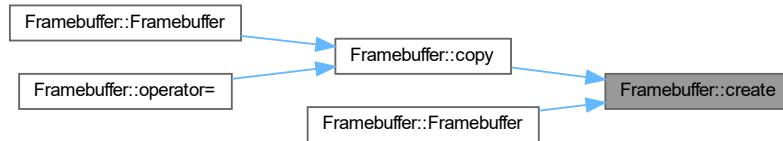
[Texture](#)を再実装しています。

[Framebuffer.cpp](#) の 129 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.9.3.4 discard()

```
void Framebuffer::discard ( ) [virtual]
```

フレームバッファオブジェクトを破棄する

[Texture](#)を再実装しています。

[Framebuffer.cpp](#) の 110 行目に定義があります。

被呼び出し関係図:



#### 8.9.3.5 draw()

```
void Framebuffer::draw (
    GLsizei width,
    GLsizei height ) const
```

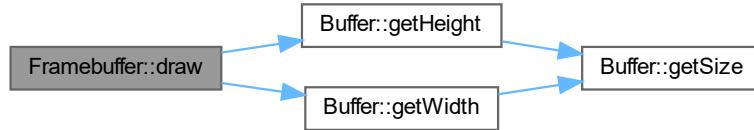
フレームバッファオブジェクトの内容を表示する

引数

<i>width</i>	フレームバッファオブジェクトの内容を表示する横の画素数
<i>height</i>	フレームバッファオブジェクトの内容を表示する縦の画素数

[Framebuffer.cpp](#) の 231 行目に定義がります。

呼び出し関係図:



#### 8.9.3.6 getChannels()

```
virtual int Framebuffer::getChannels ( ) const [inline], [virtual]
```

フレームバッファオブジェクトのチャネル数を得る

戻り値

フレームバッファオブジェクトのチャネル数

[Texture](#)を再実装しています。

[Framebuffer.h](#) の 169 行目に定義がります。

#### 8.9.3.7 getFramebufferName()

```
auto Framebuffer::getFramebufferName ( ) const [inline]
```

フレームバッファオブジェクト名を得る

戻り値

フレームバッファオブジェクト名

[Framebuffer.h](#) の 149 行目に定義がります。

### 8.9.3.8 getSize()

```
virtual const std::array< GLsizei, 2 > & Framebuffer::getSize() const [inline], [virtual]
```

フレームバッファオブジェクトのサイズを得る

戻り値

フレームバッファオブジェクトのサイズ

[Texture](#)を再実装しています。

[Framebuffer.h](#) の 159 行目に定義があります。

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

```
Framebuffer & Framebuffer::operator= (
    const Framebuffer & framebuffer )
```

代入演算子

引数

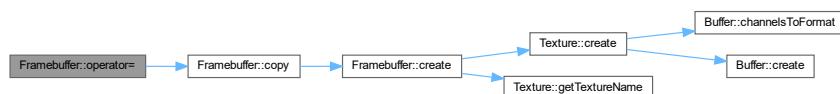
<code>framebuffer</code>	代入元のフレームバッファオブジェクト
--------------------------	--------------------

戻り値

代入結果のテクスチャ

[Framebuffer.cpp](#) の 69 行目に定義があります。

呼び出し関係図:



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

```
Framebuffer & Framebuffer::operator= (
    Framebuffer && framebuffer ) [noexcept]
```

ムーブ代入演算子

引数

<code>framebuffer</code>	ムーブ代入元のフレームバッファオブジェクト
--------------------------	-----------------------

戻り値

ムーブ代入結果のフレームバッファオブジェクト

[Framebuffer.cpp](#) の 85 行目に定義があります。

呼び出し関係図:



### 8.9.3.11 unbindFramebuffer()

```
void Framebuffer::unbindFramebuffer() const
```

レンダリング先を通常のフレームバッファに戻す

覚え書き

[bindFramebuffer\(\)](#) の後からこの前まで、レンダリング先がフレームバッファオブジェクトになる。保存したビューポートはここで復帰する。この間でレンダリング処理を実行する。

[Framebuffer.cpp](#) の 185 行目に定義があります。

被呼び出し関係図:



### 8.9.3.12 update() [1/2]

```
void Framebuffer::update(
    const std::array<int, 2> & size)
```

テクスチャを展開してフレームバッファオブジェクトを更新する

引数

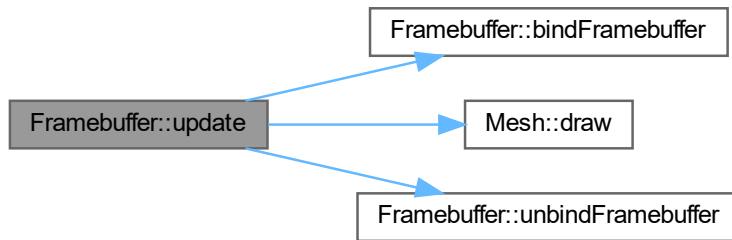
<code>size</code>	展開に用いるメッシュの分割数
-------------------	----------------

覚え書き

レンダリング先をフレームバッファオブジェクトに切り替えて、フレームバッファオブジェクト全体を覆うメッシュを描画する。これより前でシェーダの選択やテクスチャの結合を行う。

[Framebuffer.cpp](#) の 201 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.9.3.13 update() [2/2]

```

void Framebuffer::update (
    const std::array< int, 2 > & size,
    const Texture & frame,
    int unit = 0 )
  
```

テクスチャを展開してフレームバッファオブジェクトを更新する

## 引数

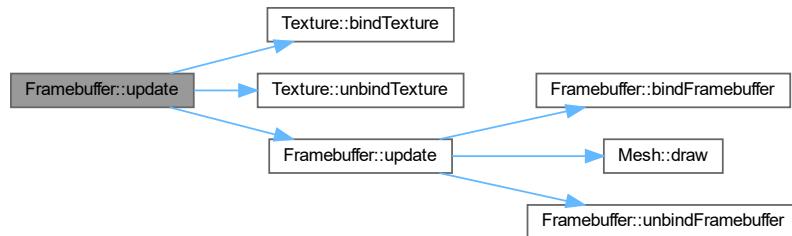
<i>size</i>	展開に用いるメッシュの分割数
<i>frame</i>	展開するフレームを格納したテクスチャ
<i>unit</i>	テクスチャのマッピングに使うテクスチャユニットの番号

## 覚え書き

レンダリング先をフレームバッファオブジェクトに切り替えて、テクスチャユニット *unit* を指定して *frame* に格納されたテクスチャを結合してから、フレームバッファオブジェクト全体を覆うメッシュを描画する。これより前でシェーダの選択を行う。*unit* にはシェーダにおいてテクスチャのサンプラーの *uniform* 変数に設定する *GL\_TEXTURE*i** の *i* と一致させること。

[Framebuffer.cpp](#) の 216 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [Framebuffer.h](#)
- [Framebuffer.cpp](#)

## 8.10 GgApp クラス

```
#include <GgApp.h>
```

### クラス

- class [Window](#)

### 公開メンバ関数

- [GgApp \(int major=0, int minor=1\)](#)
- [GgApp \(const GgApp &w\)=delete](#)
- virtual [~GgApp \(\)](#)
- [GgApp & operator= \(const GgApp &w\)=delete](#)
- int [main \(int argc, const char \\*const \\*argv\)](#)

### 8.10.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラス

GgApp.h の 64 行目に定義がります。

### 8.10.2 構築子と解体子

#### 8.10.2.1 GgApp() [1/2]

```
GgApp::GgApp (
    int major = 0,
    int minor = 1 )
```

コンストラクタ.

引数

<i>major</i>	使用する OpenGL の major 番号, 0 なら無指定.
<i>minor</i>	使用する OpenGL の minor 番号, major 番号が 0 なら無視.

GgApp.cpp の 24 行目に定義がります。

#### 8.10.2.2 GgApp() [2/2]

```
GgApp::GgApp (
    const GgApp & w ) [delete]
```

コピー構築子は使用しない

#### 8.10.2.3 ~GgApp()

```
GgApp::~GgApp ( ) [virtual]
```

デストラクタ.

GgApp.cpp の 71 行目に定義がります。

### 8.10.3 関数詳解

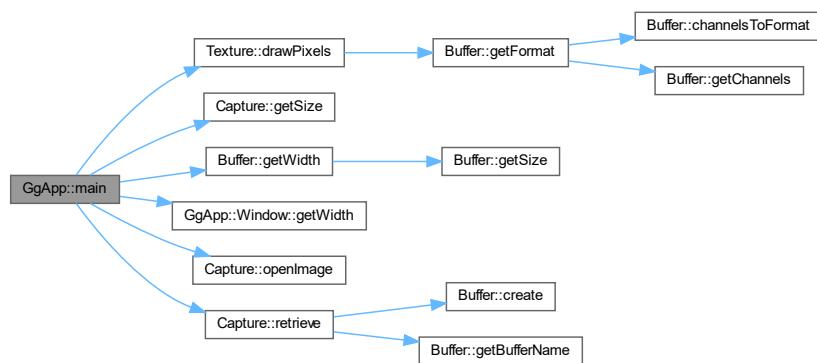
### 8.10.3.1 main()

```
int GgApp::main (
    int argc,
    const char *const * argv )
```

アプリケーション本体。

[calib.cpp](#) の 33 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.10.3.2 operator=()

```
GgApp & GgApp::operator= (
    const GgApp & w ) [delete]
```

代入演算子は使用しない

このクラス詳解は次のファイルから抽出されました:

- [GgApp.h](#)
- [calib.cpp](#)
- [GgApp.cpp](#)

## 8.11 gg::GgBuffer< T > クラステンプレート

```
#include <gg.h>
```

### 公開メンバ関数

- `GgBuffer` (GLenum target, const T \*data, GLsizei stride, GLsizei count, GLenum usage)
- virtual `~GgBuffer` ()
- `GgBuffer` (const `GgBuffer< T >` &o)=delete
- `GgBuffer< T >` & `operator=` (const `GgBuffer< T >` &o)=delete
- const GLuint & `getTarget` () const
- GLsizeiptr `getStride` () const
- const GLsizei & `getCount` () const
- const GLuint & `getBuffer` () const
- void `bind` () const
- void `unbind` () const
- void \* `map` () const
- void \* `map` (GLint first, GLsizei count) const
- void `unmap` () const
- void `send` (const T \*data, GLint first, GLsizei count) const
- void `read` (T \*data, GLint first, GLsizei count) const
- void `copy` (GLuint src\_buffer, GLint src\_first=0, GLint dst\_first=0, GLsizei count=0) const

### 8.11.1 詳解

```
template<typename T>
class gg::GgBuffer< T >
```

バッファオブジェクト。

#### 覚え書き

頂点属性／頂点インデックス／ユニフォーム変数を格納するバッファオブジェクトの基底クラス。

`gg.h` の 5544 行目に定義があります。

### 8.11.2 構築子と解体子

#### 8.11.2.1 GgBuffer() [1/2]

```
template<typename T >
gg::GgBuffer< T >::GgBuffer (
    GLenum target,
    const T * data,
    GLsizei stride,
    GLsizei count,
    GLenum usage ) [inline]
```

コンストラクタ。

引数

<i>target</i>	バッファオブジェクトのターゲット.
<i>data</i>	データが格納されている領域の先頭のポインタ ( <code>nullptr</code> ならデータを転送しない).
<i>count</i>	データの数.
<i>stride</i>	データの間隔.
<i>usage</i>	バッファオブジェクトの使い方.

`gg.h` の 5556 行目に定義があります。

### 8.11.2.2 ~GgBuffer()

```
template<typename T >
virtual gg::GgBuffer< T >::~GgBuffer ( ) [inline], [virtual]
デストラクタ.
```

`gg.h` の 5556 行目に定義があります。

### 8.11.2.3 GgBuffer() [2/2]

```
template<typename T >
gg::GgBuffer< T >::GgBuffer (
    const GgBuffer< T > & o ) [delete]
コピー構造は使用禁止.
```

## 8.11.3 関数詳解

### 8.11.3.1 bind()

```
template<typename T >
void gg::GgBuffer< T >::bind ( ) const [inline]
バッファオブジェクトを結合する.
gg.h の 5649 行目に定義があります。
```

### 8.11.3.2 copy()

```
template<typename T >
void gg::GgBuffer< T >::copy (
    GLuint src_buffer,
    GLint src_first = 0,
    GLint dst_first = 0,
    GLsizei count = 0 ) const [inline]
別のバッファオブジェクトからデータを複写する.
```

引数

<i>src_buffer</i>	複写元のバッファオブジェクト名.
<i>src_first</i>	複写元 ( <i>src_buffer</i> ) の先頭のデータの位置.
<i>dst_first</i>	複写先 ( <i>getBuffer()</i> ) の先頭のデータの位置.
<i>count</i>	複写するデータの数 (0 ならバッファオブジェクト全体).

gg.h の 5754 行目に定義があります。

#### 8.11.3.3 getBuffer()

```
template<typename T >
const GLuint & gg::GgBuffer< T >::getBuffer ( ) const [inline]
バッファオブジェクト名を取り出す.
```

戻り値

このバッファオブジェクト名.

gg.h の 5641 行目に定義があります。

#### 8.11.3.4 getCount()

```
template<typename T >
const GLsizei & gg::GgBuffer< T >::getCount ( ) const [inline]
バッファオブジェクトが保持するデータの数を取り出す.
```

戻り値

このバッファオブジェクトが保持するデータの数.

gg.h の 5631 行目に定義があります。

#### 8.11.3.5 getStride()

```
template<typename T >
GLsizeiptr gg::GgBuffer< T >::getStride ( ) const [inline]
バッファオブジェクトのアライメントを考慮したデータの間隔を取り出す.
```

戻り値

このバッファオブジェクトのデータの間隔.

gg.h の 5621 行目に定義があります。

### 8.11.3.6 `getTarget()`

```
template<typename T >
const GLuint & gg::GgBuffer< T >::getTarget ( ) const [inline]
```

バッファオブジェクトのターゲットを取り出す。

戻り値

このバッファオブジェクトのターゲット。

`gg.h` の 5611 行目に定義があります。

### 8.11.3.7 `map()` [1/2]

```
template<typename T >
void * gg::GgBuffer< T >::map ( ) const [inline]
```

バッファオブジェクトをマップする。

戻り値

マップしたメモリの先頭のポインタ。

`gg.h` の 5667 行目に定義があります。

### 8.11.3.8 `map()` [2/2]

```
template<typename T >
void * gg::GgBuffer< T >::map (
    GLint first,
    GLsizei count ) const [inline]
```

バッファオブジェクトの指定した範囲をマップする。

引数

<code>first</code>	マップする範囲のバッファオブジェクトの先頭からの位置。
<code>count</code>	マップするデータの数(0ならバッファオブジェクト全体)。

戻り値

マップしたメモリの先頭のポインタ。

`gg.h` の 5684 行目に定義があります。

### 8.11.3.9 operator=()

```
template<typename T >
GgBuffer< T > & gg::GgBuffer< T >::operator= (
    const GgBuffer< T > & o ) [delete]
```

代入演算子は使用禁止.

### 8.11.3.10 read()

```
template<typename T >
void gg::GgBuffer< T >::read (
    T * data,
    GLint first,
    GLsizei count ) const [inline]
```

バッファオブジェクトのデータから抽出する.

引数

<i>data</i>	抽出先の領域の先頭のポインタ.
<i>first</i>	抽出元のバッファオブジェクトの取り出すデータの領域の先頭の要素番号.
<i>count</i>	抽出するデータの数 (0 ならバッファオブジェクト全体).

gg.h の 5727 行目に定義があります。

### 8.11.3.11 send()

```
template<typename T >
void gg::GgBuffer< T >::send (
    const T * data,
    GLint first,
    GLsizei count ) const [inline]
```

すでに確保したバッファオブジェクトにデータを転送する.

引数

<i>data</i>	転送元のデータが格納されている領域の先頭のポインタ.
<i>first</i>	転送先のバッファオブジェクトの先頭の要素番号.
<i>count</i>	転送するデータの数 (0 ならバッファオブジェクト全体).

gg.h の 5709 行目に定義があります。

### 8.11.3.12 unbind()

```
template<typename T >
void gg::GgBuffer< T >::unbind ( ) const [inline]
```

バッファオブジェクトを解放する。

[gg.h の 5657 行目](#)に定義があります。

### 8.11.3.13 unmap()

```
template<typename T >
void gg::GgBuffer< T >::unmap ( ) const [inline]
```

バッファオブジェクトをアンマップする。

[gg.h の 5697 行目](#)に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

## 8.12 gg::GgColorTexture クラス

```
#include <gg.h>
```

### 公開メンバ関数

- [GgColorTexture \(\)](#)
- [GgColorTexture \(const GLvoid \\*image, GLsizei width, GLsizei height, GLenum format=GL\\_RGB, GLenum type=GL\\_UNSIGNED\\_BYTE, GLenum internal=GL\\_RGB, GLenum wrap=GL\\_CLAMP\\_TO\\_EDGE, bool swizzle=true\)](#)
- [GgColorTexture \(const std::string &name, GLenum internal=0, GLenum wrap=GL\\_CLAMP\\_TO\\_EDGE\)](#)
- [virtual ~GgColorTexture \(\)](#)
- [void load \(const GLvoid \\*image, GLsizei width, GLsizei height, GLenum format=GL\\_RGB, GLenum type=GL\\_UNSIGNED\\_BYTE, GLenum internal=GL\\_RGB, GLenum wrap=GL\\_CLAMP\\_TO\\_EDGE, bool swizzle=true\)](#)
- [void load \(const std::string &name, GLenum internal=0, GLenum wrap=GL\\_CLAMP\\_TO\\_EDGE\)](#)

### 8.12.1 詳解

カラーマップ。

#### 覚え書き

カラー画像を読み込んでテクスチャを作成する。

[gg.h の 5323 行目](#)に定義があります。

## 8.12.2 構築子と解体子

### 8.12.2.1 GgColorTexture() [1/3]

```
gg::GgColorTexture::GgColorTexture ( ) [inline]
```

コンストラクタ.

[gg.h](#) の 5333 行目に定義があります。

### 8.12.2.2 GgColorTexture() [2/3]

```
gg::GgColorTexture::GgColorTexture (
    const GLvoid * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RGB,
    GLenum type = GL_UNSIGNED_BYTE,
    GLenum internal = GL_RGB,
    GLenum wrap = GL_CLAMP_TO_EDGE,
    bool swizzle = true ) [inline]
```

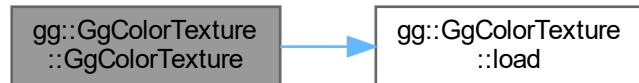
メモリ上のデータからカラーのテクスチャを作成するコンストラクタ.

引数

<i>image</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	読み込む画像の横の画素数.
<i>height</i>	読み込む画像の縦の画素数.
<i>format</i>	読み込む画像のフォーマット.
<i>type</i>	読み込む画像のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード.
<i>swizzle</i>	<code>true</code> ならテクスチャの赤と青を入れ替える, デフォルトは <code>true</code> .

[gg.h](#) の 5349 行目に定義があります。

呼び出し関係図:



### 8.12.2.3 GgColorTexture() [3/3]

```
gg::GgColorTexture::GgColorTexture (
    const std::string & name,
    GLenum internal = 0,
    GLenum wrap = GL_CLAMP_TO_EDGE ) [inline]
```

TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成するコンストラクタ.

引数

<i>name</i>	読み込むファイル名.
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる.
<i>wrap</i>	テクスチャのラッピングモード, GL_TEXTURE_WRAP_S および GL_TEXTURE_WRAP_T に設定する値.

gg.h の 5370 行目に定義があります。

呼び出し関係図:



### 8.12.2.4 ~GgColorTexture()

```
virtual gg::GgColorTexture::~GgColorTexture() [inline], [virtual]
```

デストラクタ.

[gg.h](#) の 5382 行目に定義があります。

## 8.12.3 関数詳解

### 8.12.3.1 load() [1/2]

```
void gg::GgColorTexture::load (
    const GLvoid * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RGB,
    GLenum type = GL_UNSIGNED_BYTE,
    GLenum internal = GL_RGB,
    GLenum wrap = GL_CLAMP_TO_EDGE,
    bool swizzle = true ) [inline]
```

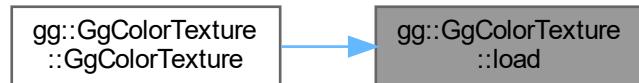
メモリ上のデータを読み込んでテクスチャを作成する.

引数

<i>image</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャの横の画素数.
<i>height</i>	テクスチャの縦の画素数.
<i>format</i>	読み込む画像のフォーマット.
<i>type</i>	読み込む画像のデータ型.
<i>internal</i>	<code>glTexImage2D()</code> に指定するテクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード ( <code>GL_CLAMP_TO_EDGE</code> , <code>GL_CLAMP_TO_BORDER</code> , <code>GL_REPEAT</code> , <code>GL_MIRRORED_REPEAT</code> ).
<i>swizzle</i>	<code>true</code> ならテクスチャの赤と青を入れ替える, デフォルトは <code>true</code> .

[gg.h](#) の 5398 行目に定義があります。

被呼び出し関係図:



### 8.12.3.2 load() [2/2]

```
void gg::GgColorTexture::load (
    const std::string & name,
    GLenum internal = 0,
    GLenum wrap = GL_CLAMP_TO_EDGE )
```

TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成する。

引数

<i>name</i>	読み込むファイル名。
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 ならファイルの画像フォーマットに合わせる。
<i>wrap</i>	テクスチャのラッピングモード (GL_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER, GL_REPEAT, GL_MIRRORED_REPEAT)。

gg.cpp の 3993 行目に定義があります。

呼び出し関係図:



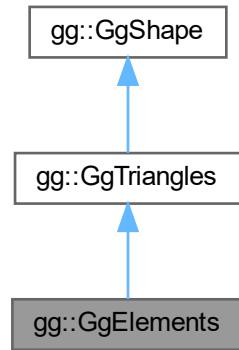
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

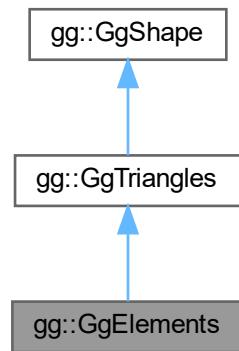
## 8.13 gg::GgElements クラス

```
#include <gg.h>
```

gg::GgElements の継承関係図



gg::GgElements 連携図



## 公開メンバ関数

- [GgElements \(GLenum mode=GL\\_TRIANGLES\)](#)
- [GgElements \(const GgVertex \\*vert, GLsizei countv, const GLuint \\*face, GLsizei countf, GLenum mode=GL\\_TRIANGLES, GLenum usage=GL\\_STATIC\\_DRAW\)](#)
- virtual [~GgElements \(\)](#)
- const GLsizei & [getIndexCount \(\) const](#)
- const GLuint & [getIndexBuffer \(\) const](#)

- void `send` (const `GgVertex` \*vert, GLuint firstv, GLsizei countv, const GLuint \*face=nullptr, GLuint firstf=0, GLsizei countf=0) const
- void `load` (const `GgVertex` \*vert, GLsizei countv, const GLuint \*face, GLsizei countf, GLenum usage=GL\_STATIC\_DRAW)
- virtual void `draw` (GLint first=0, GLsizei count=0) const

### 8.13.1 詳解

三角形で表した形状データ (Elements 形式).

`gg.h` の 6539 行目に定義があります。

### 8.13.2 構築子と解体子

#### 8.13.2.1 `GgElements()` [1/2]

```
gg::GgElements::GgElements (
    GLenum mode = GL_TRIANGLES ) [inline]
```

コンストラクタ.

引数

<code>mode</code>	描画する基本図形の種類.
-------------------	--------------

`gg.h` の 6552 行目に定義があります。

#### 8.13.2.2 `GgElements()` [2/2]

```
gg::GgElements::GgElements (
    const GgVertex * vert,
    GLsizei countv,
    const GLuint * face,
    GLsizei countf,
    GLenum mode = GL_TRIANGLES,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

コンストラクタ.

引数

<code>vert</code>	この図形の頂点属性の配列 (nullptr ならデータを転送しない).
<code>countv</code>	頂点数.

引数

<i>face</i>	三角形の頂点インデックス.
<i>countf</i>	三角形の頂点数.
<i>mode</i>	描画する基本図形の種類.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6567 行目に定義がります。

### 8.13.2.3 ~GgElements()

```
virtual gg::GgElements::~GgElements () [inline], [virtual]
```

デストラクタ.

gg.h の 6583 行目に定義がります。

## 8.13.3 関数詳解

### 8.13.3.1 draw()

```
void gg::GgElements::draw (
    GLint first = 0,
    GLsizei count = 0 ) const [virtual]
```

インデックスを使った三角形の描画.

引数

<i>first</i>	描画を開始する最初の三角形番号.
<i>count</i>	描画する三角形数, 0 なら全部の三角形を描く.

gg::GgTrianglesを再実装しています。

gg.cpp の 5148 行目に定義がります。

呼び出し関係図:



### 8.13.3.2 getIndexBuffer()

```
const GLuint & gg::GgElements::getIndexBuffer () const [inline]
```

三角形の頂点インデックスデータを格納した頂点バッファオブジェクト名を取り出す.

戻り値

この図形の三角形の頂点インデックスデータを格納した頂点バッファオブジェクト名.

[gg.h](#) の 6601 行目に定義があります。

### 8.13.3.3 getIndexCount()

```
const GLsizei & gg::GgElements::getIndexCount () const [inline]
```

データの数を取り出す.

戻り値

この図形の三角形数.

[gg.h](#) の 6591 行目に定義があります。

### 8.13.3.4 load()

```
void gg::GgElements::load (
    const GgVertex * vert,
    GLsizei countv,
    const GLuint * face,
    GLsizei countf,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

バッファオブジェクトを確保して頂点属性と三角形の頂点インデックスデータを格納する.

引数

<i>vert</i>	頂点属性が格納されている領域の先頭のポインタ.
<i>countv</i>	頂点のデータの数(頂点数).
<i>face</i>	三角形の頂点インデックスデータ.
<i>countf</i>	三角形の頂点数.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6638 行目に定義があります。

#### 8.13.3.5 send()

```
void gg::GgElements::send (
    const GgVertex * vert,
    GLuint firstv,
    GLsizei countv,
    const GLuint * face = nullptr,
    GLuint firstf = 0,
    GLsizei countf = 0 ) const [inline]
```

既存のバッファオブジェクトに頂点属性と三角形の頂点インデックスデータを転送する。

引数

<i>vert</i>	頂点属性が格納されている領域の先頭のポインタ.
<i>firstv</i>	頂点属性の転送先のバッファオブジェクトの先頭の要素番号.
<i>countv</i>	頂点のデータの数(頂点数).
<i>face</i>	三角形の頂点インデックスデータ.
<i>firstf</i>	インデックスの転送先のバッファオブジェクトの先頭の要素番号.
<i>countf</i>	三角形の頂点数.

gg.h の 6616 行目に定義があります。

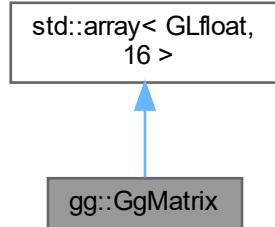
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

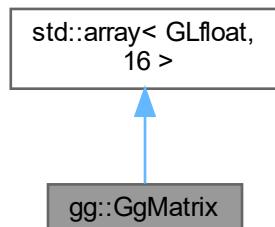
## 8.14 gg::GgMatrix クラス

```
#include <gg.h>
```

gg::GgMatrix の継承関係図



gg::GgMatrix 連携図



## 公開 メンバ関数

- [GgMatrix \(\)](#)
- [constexpr GgMatrix \(GLfloat m00, GLfloat m01, GLfloat m02, GLfloat m03, GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13, GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23, GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33\)](#)
- [constexpr GgMatrix \(GLfloat c\)](#)
- [GgMatrix \(const GLfloat \\*a\)](#)
- [GgMatrix & operator= \(const GLfloat \\*a\)](#)
- [GgMatrix operator+ \(const GLfloat \\*a\) const](#)
- [GgMatrix operator+ \(const GgMatrix &m\) const](#)
- [GgMatrix & operator+= \(const GLfloat \\*a\)](#)
- [GgMatrix & operator+= \(const GgMatrix &m\)](#)
- [GgMatrix operator- \(const GLfloat \\*a\) const](#)
- [GgMatrix operator- \(const GgMatrix &m\) const](#)
- [GgMatrix & operator-= \(const GLfloat \\*a\)](#)
- [GgMatrix & operator-= \(const GgMatrix &m\)](#)
- [GgMatrix operator\\* \(const GLfloat \\*a\) const](#)
- [GgMatrix operator\\* \(const GgMatrix &m\) const](#)

- `GgMatrix & operator*=(const GLfloat *a)`
- `GgMatrix & operator*=(const GgMatrix &m)`
- `GgMatrix operator/(const GLfloat *a) const`
- `GgMatrix operator/(const GgMatrix &m) const`
- `GgMatrix & operator/= (const GLfloat *a)`
- `GgMatrix & operator/= (const GgMatrix &m)`
- `GgMatrix & loadIdentity ()`
- `GgMatrix & loadTranslate (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix & loadTranslate (const GLfloat *t)`
- `GgMatrix & loadTranslate (const GgVector &t)`
- `GgMatrix & loadScale (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix & loadScale (const GLfloat *s)`
- `GgMatrix & loadScale (const GgVector &s)`
- `GgMatrix & loadRotateX (GLfloat a)`
- `GgMatrix & loadRotateY (GLfloat a)`
- `GgMatrix & loadRotateZ (GLfloat a)`
- `GgMatrix & loadRotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgMatrix & loadRotate (const GLfloat *r, GLfloat a)`
- `GgMatrix & loadRotate (const GgVector &r, GLfloat a)`
- `GgMatrix & loadRotate (const GLfloat *r)`
- `GgMatrix & loadRotate (const GgVector &r)`
- `GgMatrix & loadLookat (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)`
- `GgMatrix & loadLookat (const GLfloat *e, const GLfloat *t, const GLfloat *u)`
- `GgMatrix & loadLookat (const GgVector &e, const GgVector &t, const GgVector &u)`
- `GgMatrix & loadOrthogonal (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix & loadFrustum (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix & loadPerspective (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)`
- `GgMatrix & loadTranspose (const GLfloat *a)`
- `GgMatrix & loadTranspose (const GgMatrix &m)`
- `GgMatrix & loadInvert (const GLfloat *a)`
- `GgMatrix & loadInvert (const GgMatrix &m)`
- `GgMatrix & loadNormal (const GLfloat *a)`
- `GgMatrix & loadNormal (const GgMatrix &m)`
- `GgMatrix translate (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f) const`
- `GgMatrix translate (const GLfloat *t) const`
- `GgMatrix translate (const GgVector &t) const`
- `GgMatrix scale (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f) const`
- `GgMatrix scale (const GLfloat *s) const`
- `GgMatrix scale (const GgVector &s) const`
- `GgMatrix rotateX (GLfloat a) const`
- `GgMatrix rotateY (GLfloat a) const`
- `GgMatrix rotateZ (GLfloat a) const`
- `GgMatrix rotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a) const`
- `GgMatrix rotate (const GLfloat *r, GLfloat a) const`
- `GgMatrix rotate (const GgVector &r, GLfloat a) const`
- `GgMatrix rotate (const GLfloat *r) const`
- `GgMatrix rotate (const GgVector &r) const`
- `GgMatrix lookat (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz) const`
- `GgMatrix lookat (const GLfloat *e, const GLfloat *t, const GLfloat *u) const`
- `GgMatrix lookat (const GgVector &e, const GgVector &t, const GgVector &u) const`
- `GgMatrix orthogonal (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar) const`

- `GgMatrix frustum (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar) const`
- `GgMatrix perspective (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar) const`
- `GgMatrix transpose () const`
- `GgMatrix invert () const`
- `GgMatrix normal () const`
- `void projection (GLfloat *c, const GLfloat *v) const`
- `void projection (GLfloat *c, const GgVector &v) const`
- `void projection (GgVector &c, const GLfloat *v) const`
- `void projection (GgVector &c, const GgVector &v) const`
- `GgVector operator* (const GgVector &v) const`
- `const GLfloat * get () const`
- `void get (GLfloat *a) const`

### 8.14.1 詳解

変換行列。

`gg.h` の 2125 行目に定義があります。

### 8.14.2 構築子と解体子

#### 8.14.2.1 `GgMatrix()` [1/4]

`gg::GgMatrix::GgMatrix ( ) [inline]`

コンストラクタ。

`gg.h` の 2138 行目に定義があります。

#### 8.14.2.2 `GgMatrix()` [2/4]

```
constexpr gg::GgMatrix::GgMatrix (
    GLfloat m00,
    GLfloat m01,
    GLfloat m02,
    GLfloat m03,
    GLfloat m10,
    GLfloat m11,
    GLfloat m12,
    GLfloat m13,
    GLfloat m20,
    GLfloat m21,
    GLfloat m22,
    GLfloat m23,
    GLfloat m30,
    GLfloat m31,
    GLfloat m32,
    GLfloat m33 ) [inline], [constexpr]
```

コンストラクタ。

引数

<i>m00</i>	GLfloat 型の値.
<i>m01</i>	GLfloat 型の値.
<i>m02</i>	GLfloat 型の値.
<i>m03</i>	GLfloat 型の値.
<i>m10</i>	GLfloat 型の値.
<i>m11</i>	GLfloat 型の値.
<i>m12</i>	GLfloat 型の値.
<i>m13</i>	GLfloat 型の値.
<i>m20</i>	GLfloat 型の値.
<i>m21</i>	GLfloat 型の値.
<i>m22</i>	GLfloat 型の値.
<i>m23</i>	GLfloat 型の値.
<i>m30</i>	GLfloat 型の値.
<i>m31</i>	GLfloat 型の値.
<i>m32</i>	GLfloat 型の値.
<i>m33</i>	GLfloat 型の値.

gg.h の 2162 行目に定義がります。

#### 8.14.2.3 GgMatrix() [3/4]

```
constexpr gg::GgMatrix::GgMatrix (
    GLfloat c ) [inline], [constexpr]
```

コンストラクタ.

引数

<i>c</i>	GLfloat 型の値.
----------	--------------

gg.h の 2177 行目に定義がります。

#### 8.14.2.4 GgMatrix() [4/4]

```
gg::GgMatrix::GgMatrix (
    const GLfloat * a ) [inline]
```

コンストラクタ.

引数

a	GLfloat 型の 16 要素の配列変数.
---	------------------------

gg.h の 2187 行目に定義があります。

呼び出し関係図:



### 8.14.3 関数詳解

#### 8.14.3.1 frustum()

```
GgMatrix gg::GgMatrix::frustum (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar ) const [inline]
```

透視投影変換を乗じた結果を返す。

引数

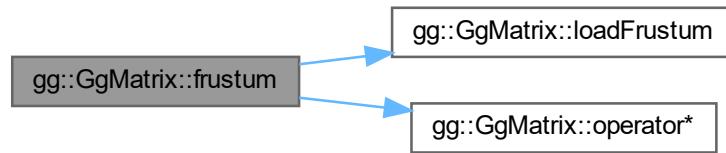
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2941 行目に定義があります。

呼び出し関係図:



#### 8.14.3.2 `get()` [1/2]

```
const GLfloat * gg::GgMatrix::get ( ) const [inline]
```

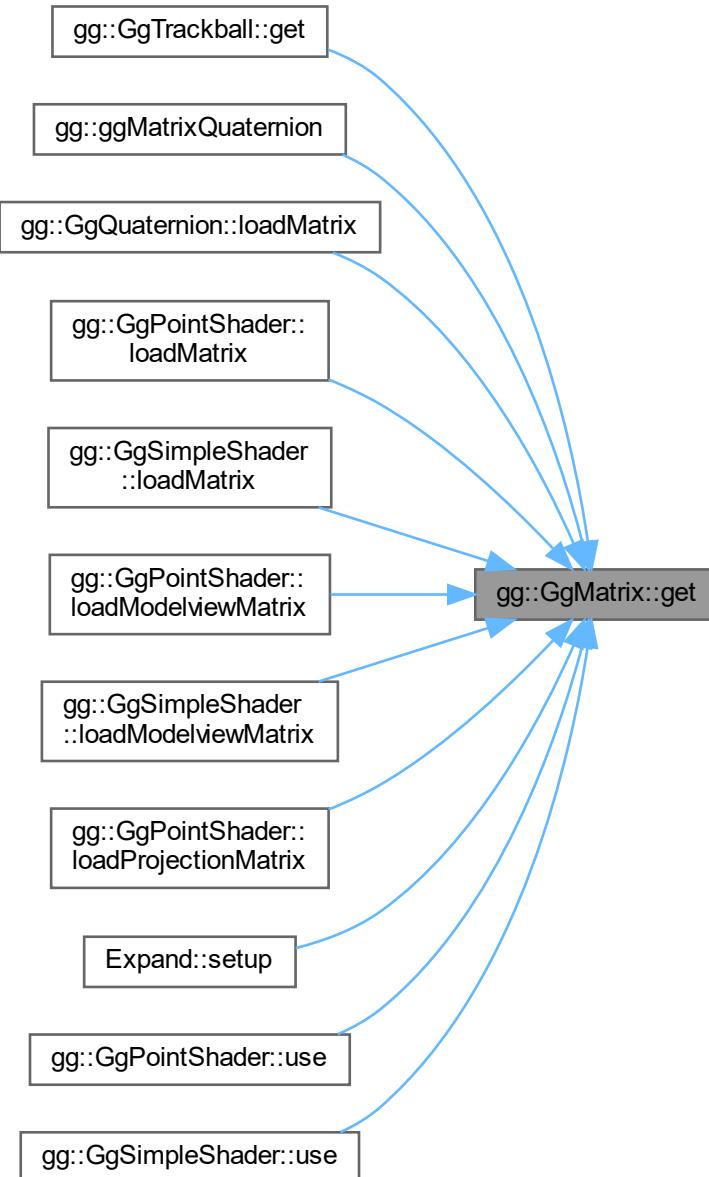
変換行列を取り出す.

戻り値

変換行列を格納した `GLfloat` 型の 16 要素の配列変数.

`gg.h` の 3064 行目に定義がります。

被呼び出し関係図:



### 8.14.3.3 `get()` [2/2]

```
void gg::GgMatrix::get (
    GLfloat * a ) const [inline]
```

変換行列を取り出す。

引数

a	変換行列を格納する GLfloat 型の 16 要素の配列変数.
---	----------------------------------

gg.h の 3074 行目に定義があります。

#### 8.14.3.4 invert()

`GgMatrix gg::GgMatrix::invert () const [inline]`

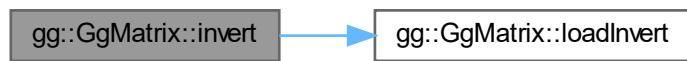
逆行列を返す。

戻り値

逆行列。

gg.h の 2985 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.14.3.5 loadFrustum()

```

gg::GgMatrix & gg::GgMatrix::loadFrustum (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar )
  
```

透視透視投影変換行列を格納する。

引数

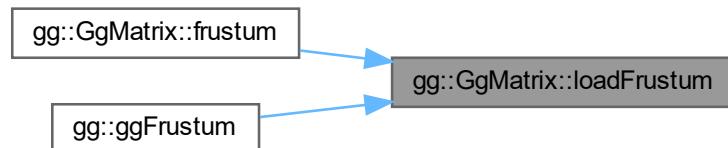
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

`gg.cpp` の 3086 行目に定義がります。

被呼び出し関係図:



#### 8.14.3.6 loadIdentity()

`gg::GgMatrix & gg::GgMatrix::loadIdentity( )`

単位行列を格納する.

`gg.cpp` の 2739 行目に定義がります。

被呼び出し関係図:



### 8.14.3.7 loadInvert() [1/2]

```
GgMatrix & gg::GgMatrix::loadInvert (
    const GgMatrix & m ) [inline]
```

逆行列を格納する。

引数

<code>m</code>	<code>GgMatrix</code> 型の変換行列。
----------------	-------------------------------

戻り値

設定した `m` の逆行列。

`gg.h` の 2664 行目に定義があります。

呼び出し関係図:



### 8.14.3.8 loadInvert() [2/2]

```
gg::GgMatrix & gg::GgMatrix::loadInvert (
    const GLfloat * a )
```

逆行列を格納する。

引数

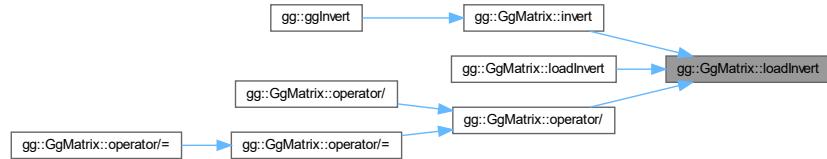
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列。
----------------	-------------------------------------

戻り値

設定した `a` の逆行列。

`gg.cpp` の 2896 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.9 `loadLookat()` [1/3]

```
GgMatrix & gg::GgMatrix::loadLookat (
    const GgVector & e,
    const GgVector & t,
    const GgVector & u ) [inline]
```

ビュー変換行列を格納する。

引数

<i>e</i>	視点の位置の <code>GgVector</code> 型の変数。
<i>t</i>	目標点の位置の <code>GgVector</code> 型の変数。
<i>u</i>	上方向のベクトルの <code>GgVector</code> 型の変数。

戻り値

設定したビュー変換行列。

`gg.h` の 2584 行目に定義があります。

呼び出し関係図:



#### 8.14.3.10 loadLookat() [2/3]

```
GgMatrix & gg::GgMatrix::loadLookat (
    const GLfloat * e,
    const GLfloat * t,
    const GLfloat * u ) [inline]
```

ビュー変換行列を格納する。

引数

e	視点の位置の配列変数.
t	目標点の位置の配列変数.
u	上方向のベクトルの配列変数.

戻り値

設定したビュー変換行列。

gg.h の 2571 行目に定義があります。

呼び出し関係図:



#### 8.14.3.11 loadLookat() [3/3]

```
gg::GgMatrix & gg::GgMatrix::loadLookat (
    GLfloat ex,
    GLfloat ey,
    GLfloat ez,
    GLfloat tx,
    GLfloat ty,
    GLfloat tz,
    GLfloat ux,
    GLfloat uy,
    GLfloat uz )
```

ビュー変換行列を格納する。

引数

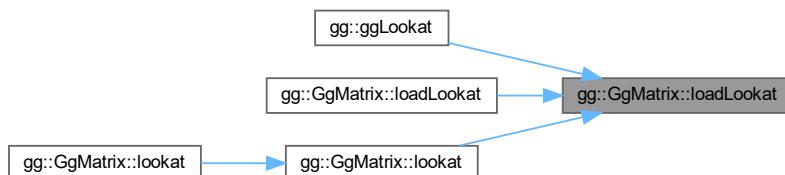
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列.

`gg.cpp` の 2998 行目に定義があります。

被呼び出し関係図:



### 8.14.3.12 `loadNormal()` [1/2]

```
GgMatrix & gg::GgMatrix::loadNormal (
    const GgMatrix & m ) [inline]
```

法線変換行列を格納する.

引数

<code>m</code>	<code>GgMatrix</code> 型の変換行列.
----------------	-------------------------------

戻り値

設定した `m` の法線変換行列.

gg.h の 2683 行目に定義があります。

呼び出し関係図:



#### 8.14.3.13 loadNormal() [2/2]

```
gg::GgMatrix & gg::GgMatrix::loadNormal (
    const GLfloat * a )
```

法線変換行列を格納する。

引数

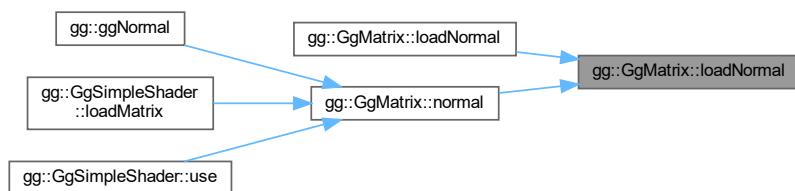
a	GLfloat 型の 16 要素の変換行列。
---	------------------------

戻り値

設定した m の法線変換行列。

gg.cpp の 2978 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.14 loadOrthogonal()

```
gg::GgMatrix & gg::GgMatrix::loadOrthogonal (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar )
```

直交投影変換行列を格納する。

引数

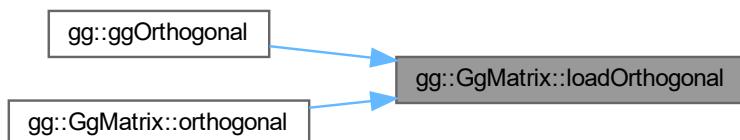
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した直交投影変換行列。

gg.cpp の 3056 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.15 loadPerspective()

```
gg::GgMatrix & gg::GgMatrix::loadPerspective (
    GLfloat fovy,
    GLfloat aspect,
    GLfloat zNear,
    GLfloat zFar )
```

画角を指定して透視投影変換行列を格納する。

引数

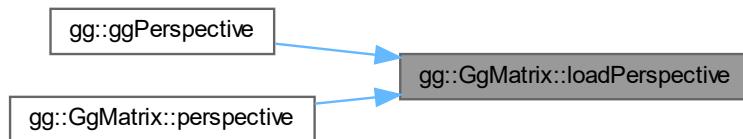
<i>fovy</i>	<i>y</i> 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 3116 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.16 loadRotate() [1/5]

```
GgMatrix & gg::GgMatrix::loadRotate (
    const GgVector & r ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を格納する.

引数

<i>r</i>	回転軸の方向ベクトルと回転角を格納した GgVector 型のベクトル, 第 4 要素を回転角に用いる.
----------	--

戻り値

設定した変換行列.

gg.h の 2540 行目に定義があります。

呼び出し関係図:



#### 8.14.3.17 loadRotate() [2/5]

```
GgMatrix & gg::GgMatrix::loadRotate (
    const GgVector & r,
    GLfloat a ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を格納する.

引数

<i>r</i>	回転軸の方向ベクトルを格納した <a href="#">GgVector</a> 型のベクトル, 第 4 要素は無視する.
<i>a</i>	回転角.

戻り値

設定した変換行列.

[gg.h](#) の 2518 行目に定義があります。

呼び出し関係図:



#### 8.14.3.18 loadRotate() [3/5]

```
GgMatrix & gg::GgMatrix::loadRotate (
    const GLfloat * r ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を格納する.

引数

<i>r</i>	回転軸の方向ベクトルと回転角を格納した GLfloat 型の 4 要素の配列変数, 第 4 要素を回転角に用いる.
----------	---

戻り値

設定した変換行列.

gg.h の 2529 行目に定義があります。

呼び出し関係図:



#### 8.14.3.19 loadRotate() [4/5]

```
GgMatrix & gg::GgMatrix::loadRotate (
    const GLfloat * r,
    GLfloat a ) [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を格納する.

引数

<i>r</i>	回転軸の方向ベクトルを格納した GLfloat 型の 3 要素の配列変数 (x, y, z).
<i>a</i>	回転角.

戻り値

設定した変換行列.

gg.h の 2506 行目に定義があります。

呼び出し関係図:



#### 8.14.3.20 loadRotate() [5/5]

```
gg::GgMatrix & gg::GgMatrix::loadRotate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a )
```

(x, y, z) 方向のベクトルを軸とする回転の変換行列を格納する。

引数

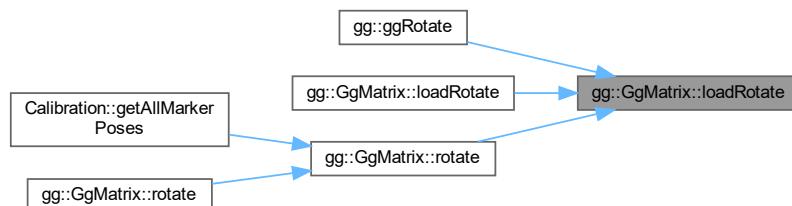
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

設定した変換行列.

gg.cpp の 2832 行目に定義があります。

被呼び出し関係図:



### 8.14.3.21 loadRotateX()

```
gg::GgMatrix & gg::GgMatrix::loadRotateX (
    GLfloat a )
```

x 軸中心の回転の変換行列を格納する。

引数

<i>a</i>	回転角.
----------	------

戻り値

設定した変換行列。

gg.cpp の 2784 行目に定義があります。

被呼び出し関係図:



### 8.14.3.22 loadRotateY()

```
gg::GgMatrix & gg::GgMatrix::loadRotateY (
    GLfloat a )
```

y 軸中心の回転の変換行列を格納する。

引数

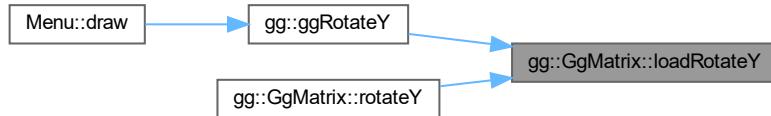
<i>a</i>	回転角.
----------	------

戻り値

設定した変換行列。

gg.cpp の 2800 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.23 loadRotateZ()

```
gg::GgMatrix & gg::GgMatrix::loadRotateZ (
    GLfloat a )
```

*z* 軸中心の回転の変換行列を格納する。

引数

a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2816 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.24 loadScale() [1/3]

```
GgMatrix & gg::GgMatrix::loadScale (
    const GgVector & s ) [inline]
```

拡大縮小の変換行列を格納する。

引数

s	拡大率の GgVector 型の変数.
---	---------------------

戻り値

設定した変換行列.

gg.h の 2459 行目に定義がります。

呼び出し関係図:



#### 8.14.3.25 loadScale() [2/3]

```
GgMatrix & gg::GgMatrix::loadScale (
    const GLfloat * s ) [inline]
```

拡大縮小の変換行列を格納する.

引数

s	拡大率の GLfloat 型の配列 (x, y, z).
---	------------------------------

戻り値

設定した変換行列.

gg.h の 2448 行目に定義がります。

呼び出し関係図:



### 8.14.3.26 loadScale() [3/3]

```
gg::GgMatrix & gg::GgMatrix::loadScale (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f )
```

拡大縮小の変換行列を格納する。

引数

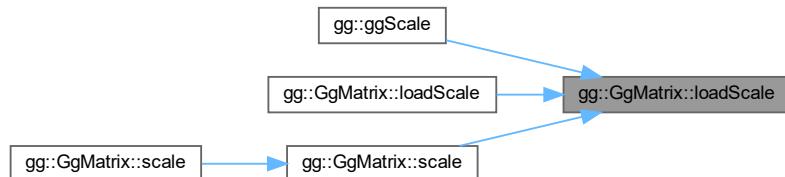
<i>x</i>	<i>x</i> 方向の拡大率。
<i>y</i>	<i>y</i> 方向の拡大率。
<i>z</i>	<i>z</i> 方向の拡大率。
<i>w</i>	<i>w</i> 拡大率のスケールファクタ (= 1.0f)。

戻り値

設定した変換行列。

gg.cpp の 2768 行目に定義があります。

被呼び出し関係図:



### 8.14.3.27 loadTranslate() [1/3]

```
GgMatrix & gg::GgMatrix::loadTranslate (
    const GgVector & t ) [inline]
```

平行移動の変換行列を格納する。

引数

<code>t</code>	移動量の GgVector 型の変数.
----------------	---------------------

戻り値

設定した変換行列.

gg.h の 2426 行目に定義がります。

呼び出し関係図:



#### 8.14.3.28 loadTranslate() [2/3]

```
GgMatrix & gg::GgMatrix::loadTranslate (const GLfloat * t) [inline]
```

平行移動の変換行列を格納する.

引数

<code>t</code>	移動量の GLfloat 型の配列 (x, y, z).
----------------	------------------------------

戻り値

設定した変換行列.

gg.h の 2415 行目に定義がります。

呼び出し関係図:



### 8.14.3.29 loadTranslate() [3/3]

```
gg::GgMatrix & gg::GgMatrix::loadTranslate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f )
```

平行移動の変換行列を格納する。

引数

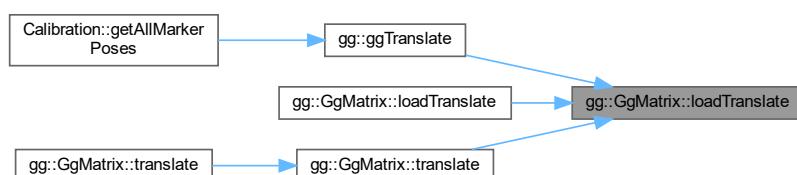
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2752 行目に定義があります。

被呼び出し関係図:



### 8.14.3.30 loadTranspose() [1/2]

```
GgMatrix & gg::GgMatrix::loadTranspose (
    const GgMatrix & m ) [inline]
```

転置行列を格納する。

引数

<code>m</code>	GgMatrix 型の変換行列.
----------------	------------------

戻り値

設定した `m` の転置行列.

`gg.h` の 2645 行目に定義があります。

呼び出し関係図:



#### 8.14.3.31 loadTranspose() [2/2]

```
gg::GgMatrix & gg::GgMatrix::loadTranspose ( const GLfloat * a )
```

転置行列を格納する.

引数

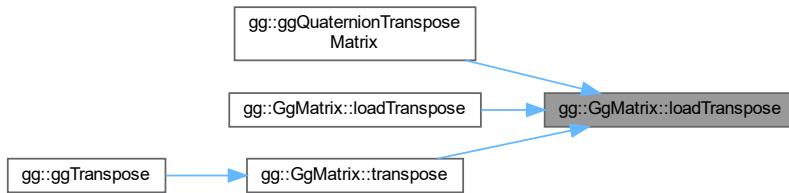
<code>a</code>	GLfloat 型の 16 要素の変換行列.
----------------	------------------------

戻り値

設定した  $\mathbf{a}$  の転置行列.

`gg.cpp` の 2871 行目に定義があります。

呼び出し関係図:



#### 8.14.3.32 `lookat()` [1/3]

```
GgMatrix gg::GgMatrix::lookat (
    const GgVector & e,
    const GgVector & t,
    const GgVector & u ) const [inline]
```

ビュー変換を乗じた結果を返す.

引数

$e$	視点の位置を格納した <code>GgVector</code> 型の変数.
$t$	目標点の位置を格納した <code>GgVector</code> 型の変数.
$u$	上方向のベクトルを格納した <code>GgVector</code> 型の変数.

戻り値

ビュー変換行列を乗じた変換行列.

`gg.h` の 2904 行目に定義があります。

呼び出し関係図:



## 8.14.3.33 lookat() [2/3]

```
GgMatrix gg::GgMatrix::lookat (
    const GLfloat * e,
    const GLfloat * t,
    const GLfloat * u ) const [inline]
```

ビュー変換を乗じた結果を返す。

引数

<i>e</i>	視点の位置を格納した <code>GLfloat</code> 型の 3 要素の配列変数.
<i>t</i>	目標点の位置を格納した <code>GLfloat</code> 型の 3 要素の配列変数.
<i>u</i>	上方向のベクトルを格納した <code>GLfloat</code> 型の 3 要素の配列変数.

戻り値

ビュー変換行列を乗じた変換行列.

`gg.h` の 2891 行目に定義があります。

呼び出し関係図:



## 8.14.3.34 lookat() [3/3]

```
GgMatrix gg::GgMatrix::lookat (
    GLfloat ex,
    GLfloat ey,
    GLfloat ez,
    GLfloat tx,
    GLfloat ty,
    GLfloat tz,
    GLfloat ux,
    GLfloat uy,
    GLfloat uz ) const [inline]
```

ビュー変換を乗じた結果を返す。

## 引数

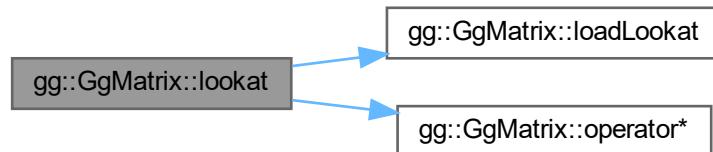
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

## 戻り値

ビュー変換行列を乗じた変換行列.

`gg.h` の 2873 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



**8.14.3.35 normal()**

```
GgMatrix gg::GgMatrix::normal () const [inline]
```

法線変換行列を返す。

戻り値

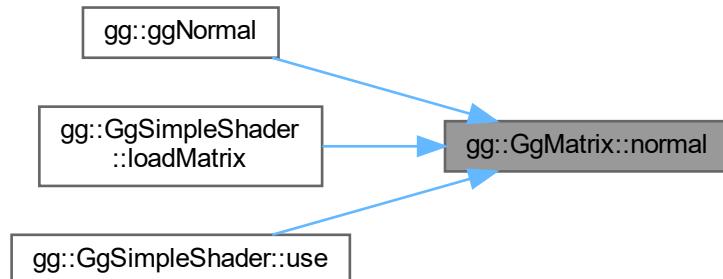
法線変換行列。

gg.h の 2996 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

**8.14.3.36 operator\*() [1/3]**

```
GgMatrix gg::GgMatrix::operator* (
    const GgMatrix & m ) const [inline]
```

変換行列に別の変換行列を乗算した値を返す。

引数

<code>m</code>	<code>GgMatrix</code> 型の変換行列.
----------------	-------------------------------

戻り値

変換行列に `a` を乗じた `GgMatrix` 型の変換行列.

`gg.h` の 2317 行目に定義があります。

呼び出し関係図:



#### 8.14.3.37 operator\*() [2/3]

```
GgVector gg::GgMatrix::operator* (
    const GgVector & v ) const [inline]
```

ベクトルに対して投影変換を行う.

引数

<code>v</code>	元のベクトルの <code>GgVector</code> 型の変数.
----------------	-------------------------------------

戻り値

`v` 変換結果の `GgVector` 型のベクトル.

`gg.h` の 3052 行目に定義があります。

#### 8.14.3.38 operator\*() [3/3]

```
GgMatrix gg::GgMatrix::operator* (
    const GLfloat * a ) const [inline]
```

変換行列に配列に格納した変換行列を乗算した値を返す.

引数

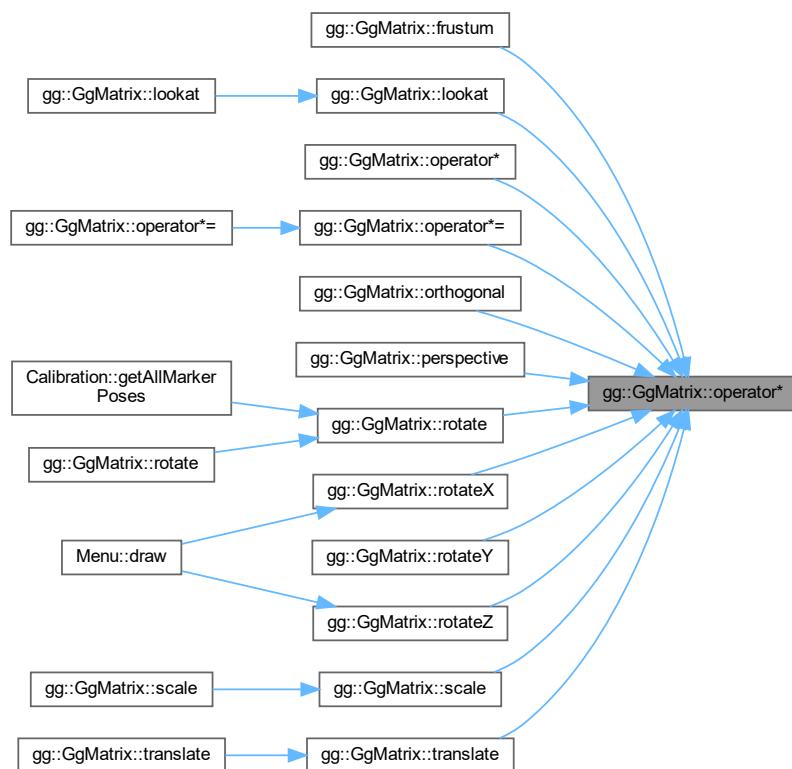
a	変換行列を格納した GLfloat 型の 16 要素の配列変数.
---	----------------------------------

戻り値

変換行列に a を乗じた GgMatrix 型の変換行列.

gg.h の 2304 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.39 operator\*=( ) [1/2]

```
GgMatrix & gg::GgMatrix::operator*=
  const GgMatrix & m) [inline]
```

変換行列に別の変換行列を乗算した結果を格納する.

引数

<code>m</code>	GgMatrix 型の変換行列.
----------------	------------------

戻り値

`m` を掛けた変換行列の参照.

`gg.h` の 2340 行目に定義があります。

呼び出し関係図:



#### 8.14.3.40 operator\*=( ) [2/2]

```
GgMatrix & gg::GgMatrix::operator*=
    const GLfloat * a ) [inline]
```

変換行列に配列に格納した変換行列を乗算した結果を格納する.

引数

<code>a</code>	変換行列を格納した GLfloat 型の 16 要素の配列変数.
----------------	----------------------------------

戻り値

`a` を掛けた変換行列の参照.

`gg.h` の 2328 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.14.3.41 operator+() [1/2]

```
GgMatrix gg::GgMatrix::operator+ (
    const GgMatrix & m ) const [inline]
```

変換行列に別の変換行列を加算した値を返す.

引数

<i>m</i>	GgMatrix 型の変換行列.
----------	------------------

戻り値

変換行列に *a* を加えた GgMatrix 型の変換行列.

gg.h の 2223 行目に定義があります。

呼び出し関係図:



#### 8.14.3.42 operator+() [2/2]

```
GgMatrix gg::GgMatrix::operator+ (
    const GLfloat * a ) const [inline]
```

変換行列に配列に格納した変換行列を加算した値を返す.

引数

a	変換行列を格納した <code>GLfloat</code> 型の 16 要素の配列変数.
---	---

戻り値

変換行列に a を加えた `GgMatrix` 型の変換行列.

`gg.h` の 2210 行目に定義があります。

呼び出し関係図:



#### 8.14.3.43 operator+=() [1/2]

```
GgMatrix & gg::GgMatrix::operator+= (
    const GgMatrix & m ) [inline]
```

変換行列に別の変換行列を加算した結果を格納する.

引数

m	<code>GgMatrix</code> 型の変換行列.
---	-------------------------------

戻り値

m を加えた変換行列の参照.

`gg.h` の 2246 行目に定義があります。

呼び出し関係図:



#### 8.14.3.44 operator+=() [2/2]

```
GgMatrix & gg::GgMatrix::operator+= (
    const GLfloat * a ) [inline]
```

変換行列に配列に格納した変換行列を加算した結果を格納する。

引数

a	変換行列を格納した GLfloat 型の 16 要素の配列変数.
---	----------------------------------

戻り値

a を加えた変換行列の参照。

gg.h の 2234 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.45 operator-() [1/2]

```
GgMatrix gg::GgMatrix::operator- (
    const GgMatrix & m ) const [inline]
```

変換行列から別の変換行列を減算した値を返す。

引数

m	GgMatrix 型の変換行列.
---	------------------

戻り値

変換行列から m を引いた GgMatrix 型の変換行列。

gg.h の 2270 行目に定義があります。

呼び出し関係図:



#### 8.14.3.46 operator-() [2/2]

```
GgMatrix gg::GgMatrix::operator- (
    const GLfloat * a ) const [inline]
```

変換行列から配列に格納した変換行列を減算した結果を格納する.

引数

<i>a</i>	変換行列を格納した GLfloat 型の 16 要素の配列変数.
----------	----------------------------------

戻り値

変換行列から *a* を引いた GgMatrix 型の変換行列.

gg.h の 2257 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.47 operator-() [1/2]

```
GgMatrix & gg::GgMatrix::operator-= (
    const GgMatrix & m ) [inline]
```

変換行列に別の変換行列を減算した結果を格納する.

引数

<code>m</code>	GgMatrix 型の変換行列.
----------------	------------------

戻り値

`m` を引いた変換行列の参照.

`gg.h` の 2293 行目に定義があります。

呼び出し関係図:



#### 8.14.3.48 operator-() [2/2]

```
GgMatrix & gg::GgMatrix::operator-= (
    const GLfloat * a ) [inline]
```

変換行列に配列に格納した変換行列を減算した結果を格納する.

引数

<code>a</code>	変換行列を格納した GLfloat 型の 16 要素の配列変数.
----------------	----------------------------------

戻り値

`a` を引いた変換行列の参照.

`gg.h` の 2281 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.49 operator/() [1/2]

```
GgMatrix gg::GgMatrix::operator/ (
    const GgMatrix & m ) const [inline]
```

変換行列を変換行列で除算した値を返す.

引数

<i>m</i>	GgMatrix 型の変換行列.
----------	------------------

戻り値

変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2365 行目に定義があります。

呼び出し関係図:



#### 8.14.3.50 operator/() [2/2]

```
GgMatrix gg::GgMatrix::operator/ (
    const GLfloat * a ) const [inline]
```

変換行列を配列に格納した変換行列で除算した値を返す.

引数

<i>a</i>	変換行列を格納した GLfloat 型の 16 要素の配列変数.
----------	----------------------------------

戻り値

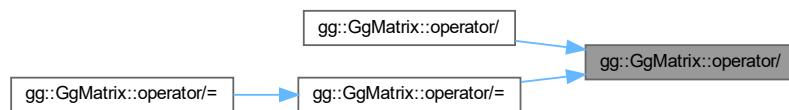
変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2351 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.14.3.51 operator/() [1/2]

```
GgMatrix & gg::GgMatrix::operator/= (
    const GgMatrix & m ) [inline]
```

変換行列を別の変換行列で除算した結果を格納する。

引数

<i>m</i>	GgMatrix 型の変換行列。
----------	------------------

戻り値

*m* で割った変換行列の参照。

gg.h の 2388 行目に定義があります。

呼び出し関係図:



### 8.14.3.52 operator/() [2/2]

```
GgMatrix & gg::GgMatrix::operator/=
    const GLfloat * a ) [inline]
```

変換行列を配列に格納した変換行列で除算した結果を格納する。

引数

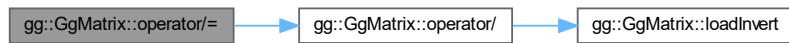
<b>a</b>	変換行列を格納した GLfloat 型の 16 要素の配列変数。
----------	----------------------------------

戻り値

a で割った変換行列の参照。

gg.h の 2376 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.14.3.53 operator=(())

```
GgMatrix & gg::GgMatrix::operator= (
    const GLfloat * a ) [inline]
```

配列変数の値を格納する。

引数

<b>a</b>	GLfloat 型の 16 要素の配列変数。
----------	------------------------

戻り値

a を代入したこのオブジェクトの参照.

gg.h の 2198 行目に定義があります。

被呼び出し関係図:



#### 8.14.3.54 orthogonal()

```
GgMatrix gg::GgMatrix::orthogonal (
    GLfloat left,
    GLfloat right,
    GLfloat bottom,
    GLfloat top,
    GLfloat zNear,
    GLfloat zFar ) const [inline]
```

直交投影変換を乗じた結果を返す.

引数

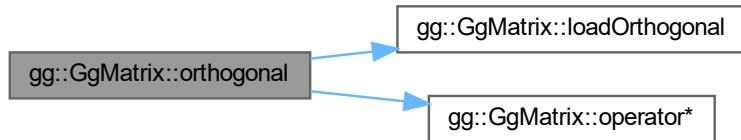
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

直交投影変換行列を乗じた変換行列.

gg.h の 2920 行目に定義があります。

呼び出し関係図:



#### 8.14.3.55 perspective()

```
GgMatrix gg::GgMatrix::perspective (
    GLfloat fovy,
    GLfloat aspect,
    GLfloat zNear,
    GLfloat zFar ) const [inline]
```

画角を指定して透視投影変換を乗じた結果を返す。

引数

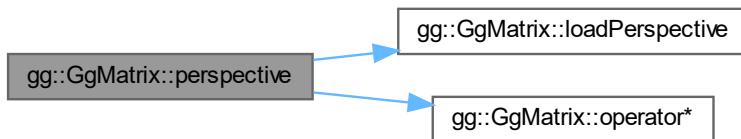
<i>fovy</i>	y 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2960 行目に定義があります。

呼び出し関係図:



## 8.14.3.56 projection() [1/4]

```
void gg::GgMatrix::projection (
    GgVector & c,
    const GgVector & v ) const [inline]
```

ベクトルに対して投影変換を行う。

引数

<i>c</i>	変換結果を格納する GgVector 型の変数。
<i>v</i>	元のベクトルの GgVector 型の変数。

gg.h の 3041 行目に定義があります。

## 8.14.3.57 projection() [2/4]

```
void gg::GgMatrix::projection (
    GgVector & c,
    const GLfloat * v ) const [inline]
```

ベクトルに対して投影変換を行う。

引数

<i>c</i>	変換結果を格納する GgVector 型の変数。
<i>v</i>	元のベクトルの GLfloat 型の 4 要素の配列変数。

gg.h の 3030 行目に定義があります。

## 8.14.3.58 projection() [3/4]

```
void gg::GgMatrix::projection (
    GLfloat * c,
    const GgVector & v ) const [inline]
```

ベクトルに対して投影変換を行う。

引数

<i>c</i>	変換結果を格納する GLfloat 型の 4 要素の配列変数。
<i>v</i>	元のベクトルの GgVector 型の変数。

gg.h の 3019 行目に定義があります。

### 8.14.3.59 projection() [4/4]

```
void gg::GgMatrix::projection (
    GLfloat * c,
    const GLfloat * v ) const [inline]
```

ベクトルに対して投影変換を行う。

引数

<i>c</i>	変換結果を格納する GLfloat 型の 4 要素の配列変数.
<i>v</i>	元のベクトルの GLfloat 型の 4 要素の配列変数.

[gg.h](#) の 3008 行目に定義があります。

### 8.14.3.60 rotate() [1/5]

```
GgMatrix gg::GgMatrix::rotate (
    const GgVector & r ) const [inline]
```

*r* 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す。

引数

<i>r</i>	回転軸の方向ベクトルと回転角を格納した GgVector 型の変数).
----------	-------------------------------------

戻り値

(*r*[0], *r*[1], *r*[2]) を軸にさらに *r*[3] 回転した変換行列。

[gg.h](#) の 2854 行目に定義があります。

呼び出し関係図:



## 8.14.3.61 rotate() [2/5]

```
GgMatrix gg::GgMatrix::rotate (
    const GgVector & r,
    GLfloat a ) const [inline]
```

$r$  方向のベクトルを軸とする回転変換を乗じた結果を返す.

引数

$r$	回転軸の方向ベクトルを格納した <code>GgVector</code> 型の変数.
$a$	回転角.

戻り値

$(r[0], r[1], r[2])$  を軸にさらに  $a$  回転した変換行列.

`gg.h` の 2832 行目に定義があります。

呼び出し関係図:



## 8.14.3.62 rotate() [3/5]

```
GgMatrix gg::GgMatrix::rotate (
    const GLfloat * r ) const [inline]
```

$r$  方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

$r$	回転軸の方向ベクトルと回転角を格納した <code>GLfloat</code> 型の 4 要素の配列変数 ( $x, y, z, a$ ).
-----	---

戻り値

$(r[0], r[1], r[2])$  を軸にさらに  $r[3]$  回転した変換行列.

`gg.h` の 2843 行目に定義があります。

呼び出し関係図:



#### 8.14.3.63 `rotate()` [4/5]

```
GgMatrix gg::GgMatrix::rotate (
    const GLfloat * r,
    GLfloat a ) const [inline]
```

`r` 方向のベクトルを軸とする回転変換を乗じた結果を返す.

引数

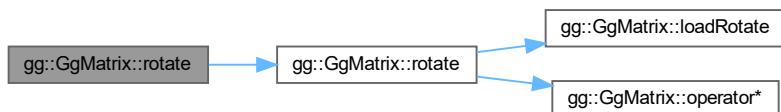
<code>r</code>	回転軸の方向ベクトルを格納した <code>GLfloat</code> 型の 3 要素の配列変数 ( <code>x, y, z</code> ).
<code>a</code>	回転角.

戻り値

`(r[0], r[1], r[2])` を軸にさらに `a` 回転した変換行列.

`gg.h` の 2820 行目に定義があります。

呼び出し関係図:



8.14.3.64 `rotate()` [5/5]

```
GgMatrix gg::GgMatrix::rotate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a ) const [inline]
```

$(x, y, z)$  方向のベクトルを軸とする回転変換を乗じた結果を返す。

引数

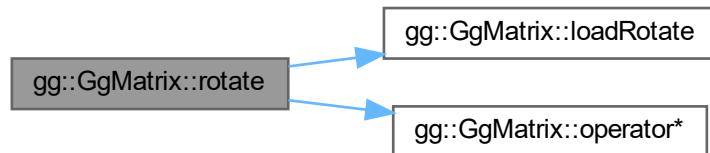
<i>x</i>	回転軸の x 成分.
<i>y</i>	回転軸の y 成分.
<i>z</i>	回転軸の z 成分.
<i>a</i>	回転角.

戻り値

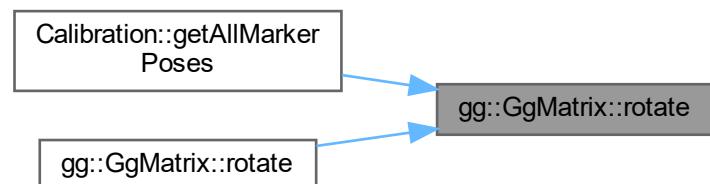
$(x, y, z)$  を軸にさらに *a* 回転した変換行列.

gg.h の 2807 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.14.3.65 rotateX()

```
GgMatrix gg::GgMatrix::rotateX (
    GLfloat a ) const [inline]
```

x 軸中心の回転変換を乗じた結果を返す。

引数

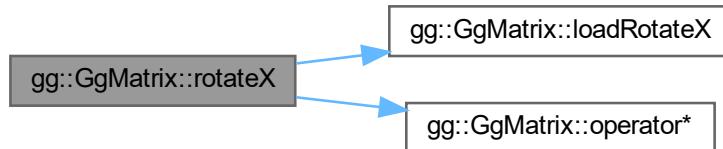
<i>a</i>	回転角.
----------	------

戻り値

x 軸中心にさらに a 回転した変換行列。

gg.h の 2768 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.14.3.66 rotateY()

```
GgMatrix gg::GgMatrix::rotateY (
    GLfloat a ) const [inline]
```

y 軸中心の回転変換を乗じた結果を返す。

引数

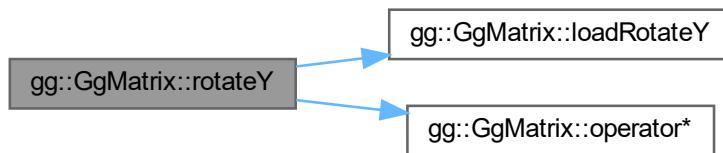
a	回転角.
---	------

戻り値

y 軸中心にさらに a 回転した変換行列.

gg.h の 2780 行目に定義がります。

呼び出し関係図:



#### 8.14.3.67 rotateZ()

```
GgMatrix gg::GgMatrix::rotateZ (
    GLfloat a ) const [inline]
```

z 軸中心の回転変換を乗じた結果を返す.

引数

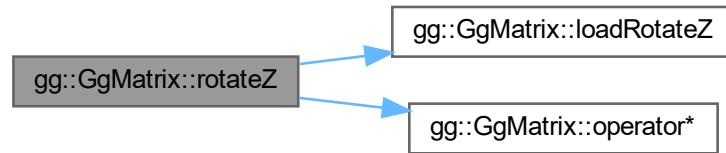
a	回転角.
---	------

戻り値

z 軸中心にさらに a 回転した変換行列.

gg.h の 2792 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



#### 8.14.3.68 scale() [1/3]

```
GgMatrix gg::GgMatrix::scale (
    const GgVector & s ) const [inline]
```

拡大縮小変換を乗じた結果を返す。

引数

<code>s</code>	拡大率の <code>GgVector</code> 型の変数。
----------------	----------------------------------

戻り値

拡大縮小した結果の変換行列。

`gg.h` の 2757 行目に定義があります。

呼び出し関係図:



#### 8.14.3.69 scale() [2/3]

```
GgMatrix gg::GgMatrix::scale (
    const GLfloat * s ) const [inline]
```

拡大縮小変換を乗じた結果を返す.

引数

<b>s</b>	拡大率の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

拡大縮小した結果の変換行列.

gg.h の 2746 行目に定義があります。

呼び出し関係図:



#### 8.14.3.70 scale() [3/3]

```
GgMatrix gg::GgMatrix::scale (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f ) const [inline]
```

拡大縮小変換を乗じた結果を返す.

引数

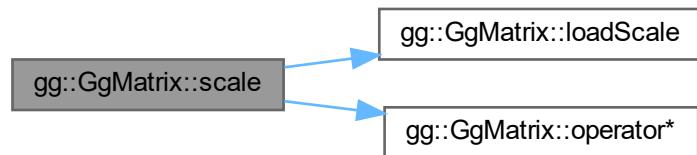
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

拡大縮小した結果の変換行列.

gg.h の 2734 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.14.3.71 translate() [1/3]

```
GgMatrix gg::GgMatrix::translate (
    const GgVector & t ) const [inline]
```

平行移動変換を乗じた結果を返す.

引数

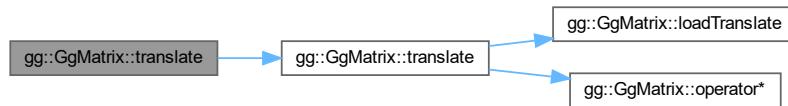
<i>t</i>	移動量の GgVector 型の変数.
----------	---------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2720 行目に定義があります。

呼び出し関係図:



#### 8.14.3.72 translate() [2/3]

```
GgMatrix gg::GgMatrix::translate (
    const GLfloat * t ) const [inline]
```

平行移動変換を乗じた結果を返す.

引数

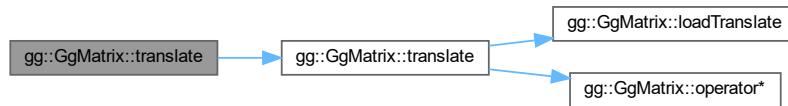
<i>t</i>	移動量の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2709 行目に定義があります。

呼び出し関係図:



### 8.14.3.73 translate() [3/3]

```
GgMatrix gg::GgMatrix::translate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f ) const [inline]
```

平行移動変換を乗じた結果を返す.

引数

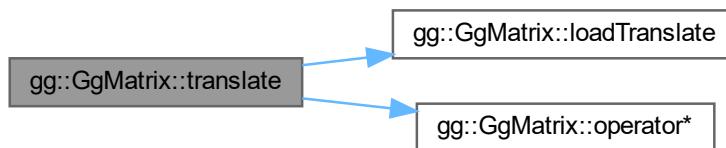
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

平行移動した結果の変換行列.

gg.h の 2697 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.14.3.74 transpose()

```
GgMatrix gg::GgMatrix::transpose () const [inline]
```

転置行列を返す。

戻り値

転置行列。

gg.h の 2974 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.15 gg::GgNormalTexture クラス

```
#include <gg.h>
```

公開メンバ関数

- [GgNormalTexture \(\)](#)
- [GgNormalTexture \(const GLubyte \\*image, GLsizei width, GLsizei height, GLenum format=GL\\_RED, GLfloat nz=1.0f, GLenum internal=GL\\_RGBA\)](#)
- [GgNormalTexture \(const std::string &name, GLfloat nz=1.0f, GLenum internal=GL\\_RGBA\)](#)
- [virtual ~GgNormalTexture \(\)](#)
- [void load \(const GLubyte \\*hmap, GLsizei width, GLsizei height, GLenum format=GL\\_RED, GLfloat nz=1.0f, GLenum internal=GL\\_RGBA\)](#)
- [void load \(const std::string &name, GLfloat nz=1.0f, GLenum internal=GL\\_RGBA\)](#)

### 8.15.1 詳解

法線マップ.

覚え書き

高さマップ（グレイスケール画像）を読み込んで法線マップのテクスチャを作成する。

[gg.h](#) の 5433 行目に定義があります。

### 8.15.2 構築子と解体子

#### 8.15.2.1 GgNormalTexture() [1/3]

```
gg::GgNormalTexture::GgNormalTexture ( ) [inline]
```

コンストラクタ.

[gg.h](#) の 5443 行目に定義があります。

#### 8.15.2.2 GgNormalTexture() [2/3]

```
gg::GgNormalTexture::GgNormalTexture (
    const GLubyte * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RED,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA ) [inline]
```

メモリ上のデータから法線マップのテクスチャを作成するコンストラクタ.

引数

<i>image</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット ( <code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code> , <code>GL_RGBA</code> ).
<i>nz</i>	法線マップの z 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

[gg.h](#) の 5457 行目に定義があります。

呼び出し関係図:



### 8.15.2.3 GgNormalTexture() [3/3]

```
gg::GgNormalTexture::GgNormalTexture (
    const std::string & name,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA ) [inline]
```

ファイルからデータを読み込んで法線マップのテクスチャを作成するコンストラクタ.

引数

<i>name</i>	画像ファイル名.
<i>nz</i>	法線マップの z 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5477 行目に定義がります。

呼び出し関係図:



### 8.15.2.4 ~GgNormalTexture()

```
virtual gg::GgNormalTexture::~GgNormalTexture( ) [inline], [virtual]
```

デストラクタ.

gg.h の 5490 行目に定義がります。

## 8.15.3 関数詳解

### 8.15.3.1 load() [1/2]

```
void gg::GgNormalTexture::load (
    const GLubyte * hmap,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RED,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA ) [inline]
```

メモリ上のデータから法線マップのテクスチャを作成する。

引数

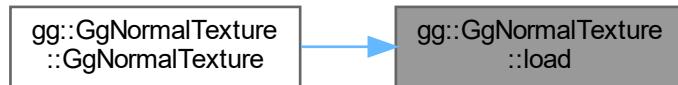
<i>hmap</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット ( <code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code> , <code>GL_RGBA</code> ).
<i>nz</i>	法線マップの z 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

`gg.h` の 5504 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.15.3.2 load() [2/2]

```
void gg::GgNormalTexture::load (
    const std::string & name,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA )
```

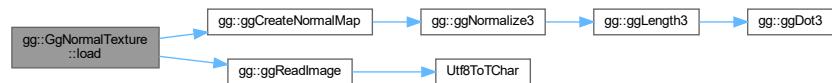
TGA フォーマットの画像ファイルから高さマップ読み込んで法線マップのテクスチャを作成する。

引数

<i>name</i>	画像ファイル名(1チャネルのTGA画像).
<i>nz</i>	法線マップのz成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.cpp の 4029 行目に定義がります。

呼び出し関係図:



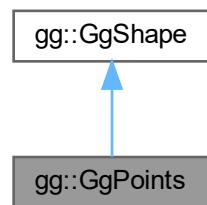
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

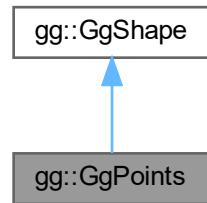
## 8.16 gg::GgPoints クラス

```
#include <gg.h>
```

gg::GgPoints の継承関係図



## gg::GgPoints 連携図



## 公開メンバ関数

- `GgPoints (GLenum mode=GL_POINTS)`
- `GgPoints (const GgVector *pos, GLsizei count, GLenum mode=GL_POINTS, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~GgPoints ()`
- `operator bool () const noexcept`
- `bool operator! () const noexcept`
- `const GLsizei & getCount () const`
- `const GLuint & getBuffer () const`
- `void send (const GgVector *pos, GLint first=0, GLsizei count=0) const`
- `void load (const GgVector *pos, GLsizei count, GLenum usage=GL_STATIC_DRAW)`
- `virtual void draw (GLint first=0, GLsizei count=0) const`

**8.16.1 詳解**

点.

[gg.h](#) の 6266 行目に定義がります。

**8.16.2 構築子と解体子****8.16.2.1 GgPoints() [1/2]**

```
gg::GgPoints::GgPoints (
    GLenum mode = GL_POINTS ) [inline]
```

コンストラクタ.

[gg.h](#) の 6277 行目に定義がります。

### 8.16.2.2 GgPoints() [2/2]

```
gg::GgPoints::GgPoints (
    const GgVector * pos,
    GLsizei countv,
    GLenum mode = GL_POINTS,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	この図形の頂点の位置のデータの配列 (nullptr ならデータを転送しない).
<i>countv</i>	頂点数.
<i>mode</i>	描画する基本図形の種類.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6290 行目に定義があります。

### 8.16.2.3 ~GgPoints()

```
virtual gg::GgPoints::~GgPoints () [inline], [virtual]
```

デストラクタ.

gg.h の 6304 行目に定義があります。

## 8.16.3 関数詳解

### 8.16.3.1 draw()

```
void gg::GgPoints::draw (
    GLint first = 0,
    GLsizei count = 0 ) const [virtual]
```

点の描画.

引数

<i>first</i>	描画を開始する最初の点の番号.
<i>count</i>	描画する点の数, 0 なら全部の点を描く.

gg::GgShape を再実装しています。

[gg.cpp](#) の 5105 行目に定義があります。

呼び出し関係図:



#### 8.16.3.2 `getBuffer()`

`const GLuint & gg::GgPoints::getBuffer () const [inline]`

頂点の位置データを格納した頂点バッファオブジェクト名を取り出す.

戻り値

この図形の頂点の位置データを格納した頂点バッファオブジェクト名.

[gg.h](#) の 6343 行目に定義があります。

#### 8.16.3.3 `getCount()`

`const GLsizei & gg::GgPoints::getCount () const [inline]`

データの数を取り出す.

戻り値

この図形の頂点の位置データの数 (頂点数).

[gg.h](#) の 6333 行目に定義があります。

#### 8.16.3.4 `load()`

```

void gg::GgPoints::load (
    const GgVector * pos,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW )
  
```

バッファオブジェクトを確保して頂点の位置データを格納する.

引数

<i>pos</i>	頂点の位置データが格納されている領域の先頭のポインタ.
<i>count</i>	頂点のデータの数(頂点数).
<i>usage</i>	バッファオブジェクトの使い方.

gg.cpp の 5092 行目に定義があります。

#### 8.16.3.5 operator bool()

gg::GgPoints::operator bool ( ) const [inline], [explicit], [virtual], [noexcept]

バッファが有効かどうか調べる。

戻り値

バッファが有効なら true

gg::GgShapeを再実装しています。

gg.h の 6313 行目に定義があります。

#### 8.16.3.6 operator”!”()

bool gg::GgPoints::operator! ( ) const [inline], [virtual], [noexcept]

バッファが有効かどうかの結果を反転する。

戻り値

バッファが有効なら false, 無効なら true.

gg::GgShapeを再実装しています。

gg.h の 6323 行目に定義があります。

#### 8.16.3.7 send()

```
void gg::GgPoints::send (
    const GgVector * pos,
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

既存のバッファオブジェクトに頂点の位置データを転送する。

## 引数

<i>pos</i>	転送元の頂点の位置データが格納されている領域の先頭のポインタ.
<i>first</i>	転送先のバッファオブジェクトの先頭の要素番号.
<i>count</i>	転送する頂点の位置データの数(0ならバッファオブジェクト全体).

[gg.h](#) の 6355 行目に定義があります。

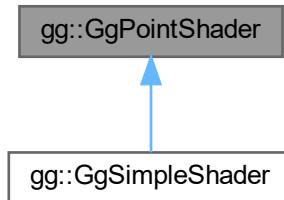
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.17 gg::GgPointShader クラス

```
#include <gg.h>
```

gg::GgPointShader の継承関係図



### 公開 メンバ 関数

- [GgPointShader \(\)](#)
- [GgPointShader \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- [GgPointShader \(const std::array< std::string, 3 > &files, int nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- virtual [~GgPointShader \(\)](#)
- bool [load \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- bool [load \(const std::array< std::string, 3 > &files, GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- virtual void [loadProjectionMatrix \(const GLfloat \\*mp\) const](#)
- virtual void [loadProjectionMatrix \(const GgMatrix &mp\) const](#)
- virtual void [loadModelviewMatrix \(const GLfloat \\*mv\) const](#)
- virtual void [loadModelviewMatrix \(const GgMatrix &mv\) const](#)
- virtual void [loadMatrix \(const GLfloat \\*mp, const GLfloat \\*mv\) const](#)
- virtual void [loadMatrix \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- virtual void [use \(\) const](#)
- void [use \(const GLfloat \\*mp\) const](#)
- void [use \(const GgMatrix &mp\) const](#)
- void [use \(const GLfloat \\*mp, const GLfloat \\*mv\) const](#)
- void [use \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- void [unuse \(\) const](#)
- GLuint [get \(\) const](#)

### 8.17.1 詳解

点のシェーダ.

[gg.h の 6890 行目](#)に定義がります。

### 8.17.2 構築子と解体子

#### 8.17.2.1 GgPointShader() [1/3]

```
gg::GgPointShader::GgPointShader ( ) [inline]
```

コンストラクタ.

[gg.h の 6906 行目](#)に定義がります。

#### 8.17.2.2 GgPointShader() [2/3]

```
gg::GgPointShader::GgPointShader (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

<i>vert</i>	パーティクルシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

[gg.h の 6921 行目](#)に定義がります。

#### 8.17.2.3 GgPointShader() [3/3]

```
gg::GgPointShader::GgPointShader (
    const std::array< std::string, 3 > & files,
```

```
int nvarying = 0,
const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列.
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).

[gg.h](#) の 6940 行目に定義があります。

#### 8.17.2.4 ~GgPointShader()

```
virtual gg::GgPointShader::~GgPointShader() [inline], [virtual]
```

デストラクタ.

[gg.h](#) の 6952 行目に定義があります。

### 8.17.3 関数詳解

#### 8.17.3.1 get()

```
GLuint gg::GgPointShader::get() const [inline]
```

シェーダのプログラム名を得る.

戻り値

シェーダのプログラム名.

[gg.h](#) の 7136 行目に定義があります。

#### 8.17.3.2 load() [1/2]

```
bool gg::GgPointShader::load(
    const std::array<std::string, 3> & files,
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.

## 引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列.
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).

## 戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 6999 行目に定義があります。

## 8.17.3.3 load() [2/2]

```
bool gg::GgPointShader::load (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

シェーダのソースファイルを読み込む.

## 引数

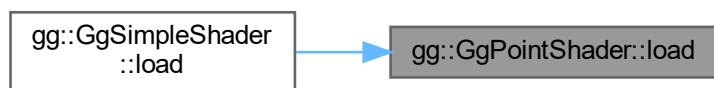
<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

## 戻り値

プログラムオブジェクトが作成できれば true.

gg.h の 6966 行目に定義があります。

被呼び出し関係図:



### 8.17.3.4 loadMatrix() [1/2]

```
virtual void gg::GgPointShader::loadMatrix (
    const GgMatrix & mp,
    const GgMatrix & mv ) const [inline], [virtual]
```

投影変換行列とモデルビュー変換行列を設定する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
<i>mv</i>	GgMatrix 型のモデルビュー変換行列.

gg::GgSimpleShaderで再実装されています。

gg.h の 7066 行目に定義があります。

呼び出し関係図:



### 8.17.3.5 loadMatrix() [2/2]

```
virtual void gg::GgPointShader::loadMatrix (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline], [virtual]
```

投影変換行列とモデルビュー変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.

gg::GgSimpleShaderで再実装されています。

gg.h の 7054 行目に定義がります。

#### 8.17.3.6 loadModelviewMatrix() [1/2]

```
virtual void gg::GgPointShader::loadModelviewMatrix (
    const GgMatrix & mv ) const [inline], [virtual]
```

モデルビュー変換行列を設定する。

引数

<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
-----------	------------------------

gg::GgSimpleShaderで再実装されています。

gg.h の 7043 行目に定義がります。

呼び出し関係図:



#### 8.17.3.7 loadModelviewMatrix() [2/2]

```
virtual void gg::GgPointShader::loadModelviewMatrix (
    const GLfloat * mv ) const [inline], [virtual]
```

モデルビュー変換行列を設定する。

引数

<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
-----------	--

gg::GgSimpleShaderで再実装されています。

gg.h の 7033 行目に定義がります。

### 8.17.3.8 loadProjectionMatrix() [1/2]

```
virtual void gg::GgPointShader::loadProjectionMatrix (
    const GgMatrix & mp ) const [inline], [virtual]
```

投影変換行列を設定する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
-----------	--------------------

gg.h の 7023 行目に定義がります。

呼び出し関係図:



### 8.17.3.9 loadProjectionMatrix() [2/2]

```
virtual void gg::GgPointShader::loadProjectionMatrix (
    const GLfloat * mp ) const [inline], [virtual]
```

投影変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
-----------	------------------------------------

gg.h の 7013 行目に定義がります。

### 8.17.3.10 unuse()

```
void gg::GgPointShader::unuse ( ) const [inline]
```

シェーダプログラムの使用を終了する。

gg.h の 7126 行目に定義がります。

### 8.17.3.11 use() [1/5]

```
virtual void gg::GgPointShader::use ( ) const [inline], [virtual]
```

シェーダプログラムの使用を開始する。

[gg::GgSimpleShader](#)で再実装されています。

[gg.h](#) の 7074 行目に定義があります。

### 8.17.3.12 use() [2/5]

```
void gg::GgPointShader::use (
    const GgMatrix & mp ) const [inline]
```

投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	<a href="#">GgMatrix</a> 型の投影変換行列。
-----------	------------------------------------

[gg.h](#) の 7095 行目に定義があります。

呼び出し関係図:



### 8.17.3.13 use() [3/5]

```
void gg::GgPointShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv ) const [inline]
```

投影変換行列とモデルビューを設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	<a href="#">GgMatrix</a> 型の投影変換行列。
<i>mv</i>	<a href="#">GgMatrix</a> 型のモデルビュー変換行列。

[gg.h](#) の 7118 行目に定義があります。

呼び出し関係図:



#### 8.17.3.14 use() [4/5]

```
void gg::GgPointShader::use (
    const GLfloat * mp ) const [inline]
```

投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列。
-----------	------------------------------------

[gg.h](#) の 7084 行目に定義があります。

#### 8.17.3.15 use() [5/5]

```
void gg::GgPointShader::use (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline]
```

投影変換行列とモデルビューを設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列。
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列。

[gg.h](#) の 7106 行目に定義があります。

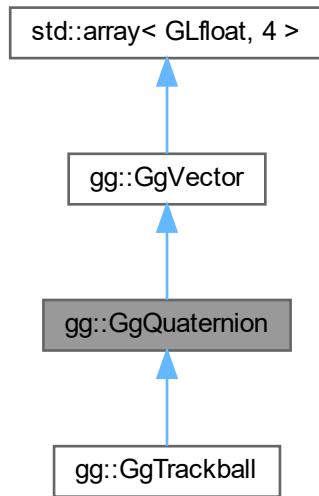
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

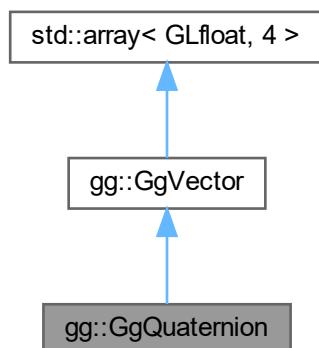
## 8.18 gg::GgQuaternion クラス

```
#include <gg.h>
```

gg::GgQuaternion の継承関係図



gg::GgQuaternion 連携図



公開メンバ関数

- [GgQuaternion \(\)](#)

- `constexpr GgQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `constexpr GgQuaternion (GLfloat c)`
- `GgQuaternion (const GLfloat *a)`
- `GgQuaternion (const GgVector &v)`
- `GLfloat norm () const`
- `GgQuaternion & load (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion & load (const GLfloat *a)`
- `GgQuaternion & load (const GgVector &v)`
- `GgQuaternion & load (const GgQuaternion &q)`
- `GgQuaternion & loadAdd (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion & loadAdd (const GLfloat *a)`
- `GgQuaternion & loadAdd (const GgVector &v)`
- `GgQuaternion & loadAdd (const GgQuaternion &q)`
- `GgQuaternion & loadSubtract (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion & loadSubtract (const GLfloat *a)`
- `GgQuaternion & loadSubtract (const GgVector &v)`
- `GgQuaternion & loadSubtract (const GgQuaternion &q)`
- `GgQuaternion & loadMultiply (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion & loadMultiply (const GLfloat *a)`
- `GgQuaternion & loadMultiply (const GgVector &v)`
- `GgQuaternion & loadMultiply (const GgQuaternion &q)`
- `GgQuaternion & loadDivide (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion & loadDivide (const GLfloat *a)`
- `GgQuaternion & loadDivide (const GgVector &v)`
- `GgQuaternion & loadDivide (const GgQuaternion &q)`
- `GgQuaternion add (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const`
- `GgQuaternion add (const GLfloat *a) const`
- `GgQuaternion add (const GgVector &v) const`
- `GgQuaternion add (const GgQuaternion &q) const`
- `GgQuaternion subtract (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const`
- `GgQuaternion subtract (const GLfloat *a) const`
- `GgQuaternion subtract (const GgVector &v) const`
- `GgQuaternion subtract (const GgQuaternion &q) const`
- `GgQuaternion multiply (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const`
- `GgQuaternion multiply (const GLfloat *a) const`
- `GgQuaternion multiply (const GgVector &v) const`
- `GgQuaternion multiply (const GgQuaternion &q) const`
- `GgQuaternion divide (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const`
- `GgQuaternion divide (const GLfloat *a) const`
- `GgQuaternion divide (const GgVector &v) const`
- `GgQuaternion divide (const GgQuaternion &q) const`
- `GgQuaternion & operator= (const GLfloat *a)`
- `GgQuaternion & operator= (const GgVector &v)`
- `GgQuaternion & operator+= (const GLfloat *a)`
- `GgQuaternion & operator+= (const GgVector &v)`
- `GgQuaternion & operator+= (const GgQuaternion &q)`
- `GgQuaternion & operator-= (const GLfloat *a)`
- `GgQuaternion & operator-= (const GgVector &v)`
- `GgQuaternion & operator-= (const GgQuaternion &q)`
- `GgQuaternion & operator*=(const GLfloat *a)`
- `GgQuaternion & operator*=(const GgVector &v)`
- `GgQuaternion & operator*=(const GgQuaternion &q)`
- `GgQuaternion & operator/=(const GLfloat *a)`
- `GgQuaternion & operator/=(const GgVector &v)`
- `GgQuaternion & operator/=(const GgQuaternion &q)`

- `GgQuaternion operator+ (const GLfloat *a) const`
- `GgQuaternion operator+ (const GgVector &v) const`
- `GgQuaternion operator+ (const GgQuaternion &q) const`
- `GgQuaternion operator- (const GLfloat *a) const`
- `GgQuaternion operator- (const GgVector &v) const`
- `GgQuaternion operator- (const GgQuaternion &q) const`
- `GgQuaternion operator* (const GLfloat *a) const`
- `GgQuaternion operator* (const GgVector &v) const`
- `GgQuaternion operator* (const GgQuaternion &q) const`
- `GgQuaternion operator/ (const GLfloat *a) const`
- `GgQuaternion operator/ (const GgVector &v) const`
- `GgQuaternion operator/ (const GgQuaternion &q) const`
- `GgQuaternion & loadMatrix (const GLfloat *a)`
- `GgQuaternion & loadMatrix (const GgMatrix &m)`
- `GgQuaternion & loadIdentity ()`
- `GgQuaternion & loadRotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgQuaternion & loadRotate (const GLfloat *v, GLfloat a)`
- `GgQuaternion & loadRotate (const GLfloat *v)`
- `GgQuaternion & loadRotateX (GLfloat a)`
- `GgQuaternion & loadRotateY (GLfloat a)`
- `GgQuaternion & loadRotateZ (GLfloat a)`
- `GgQuaternion rotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a) const`
- `GgQuaternion rotate (const GLfloat *v, GLfloat a) const`
- `GgQuaternion rotate (const GLfloat *v) const`
- `GgQuaternion rotateX (GLfloat a) const`
- `GgQuaternion rotateY (GLfloat a) const`
- `GgQuaternion rotateZ (GLfloat a) const`
- `GgQuaternion & loadEuler (GLfloat heading, GLfloat pitch, GLfloat roll)`
- `GgQuaternion & loadEuler (const GLfloat *e)`
- `GgQuaternion euler (GLfloat heading, GLfloat pitch, GLfloat roll) const`
- `GgQuaternion euler (const GLfloat *e) const`
- `GgQuaternion & loadSlerp (const GLfloat *a, const GLfloat *b, GLfloat t)`
- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
- `GgQuaternion & loadSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
- `GgQuaternion & loadNormalize (const GLfloat *a)`
- `GgQuaternion & loadNormalize (const GgQuaternion &q)`
- `GgQuaternion & loadConjugate (const GLfloat *a)`
- `GgQuaternion & loadConjugate (const GgQuaternion &q)`
- `GgQuaternion & loadInvert (const GLfloat *a)`
- `GgQuaternion & loadInvert (const GgQuaternion &q)`
- `GgQuaternion slerp (GLfloat *a, GLfloat t) const`
- `GgQuaternion slerp (const GgQuaternion &q, GLfloat t) const`
- `GgQuaternion normalize () const`
- `GgQuaternion conjugate () const`
- `GgQuaternion invert () const`
- `void get (GLfloat *a) const`
- `void getMatrix (GLfloat *a) const`
- `void getMatrix (GgMatrix &m) const`
- `GgMatrix getMatrix () const`
- `void getConjugateMatrix (GLfloat *a) const`
- `void getConjugateMatrix (GgMatrix &m) const`
- `GgMatrix getConjugateMatrix () const`

### 8.18.1 詳解

四元数.

[gg.h の 3424 行目](#)に定義がります。

### 8.18.2 構築子と解体子

#### 8.18.2.1 GgQuaternion() [1/5]

```
gg::GgQuaternion::GgQuaternion ( ) [inline]
```

コンストラクタ.

[gg.h の 3443 行目](#)に定義がります。

#### 8.18.2.2 GgQuaternion() [2/5]

```
constexpr gg::GgQuaternion::GgQuaternion (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline], [constexpr]
```

コンストラクタ.

引数

<i>x</i>	四元数の <i>x</i> 要素.
<i>y</i>	四元数の <i>y</i> 要素.
<i>z</i>	四元数の <i>z</i> 要素.
<i>w</i>	四元数の <i>w</i> 要素.

[gg.h の 3455 行目](#)に定義がります。

#### 8.18.2.3 GgQuaternion() [3/5]

```
constexpr gg::GgQuaternion::GgQuaternion (
    GLfloat c ) [inline], [constexpr]
```

コンストラクタ.

引数

c	GLfloat 型の値.
---	--------------

gg.h の 3465 行目に定義があります。

#### 8.18.2.4 GgQuaternion() [4/5]

```
gg::GgQuaternion::GgQuaternion (
    const GLfloat * a ) [inline]
```

コンストラクタ.

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

gg.h の 3475 行目に定義があります。

#### 8.18.2.5 GgQuaternion() [5/5]

```
gg::GgQuaternion::GgQuaternion (
    const GgVector & v ) [inline]
```

コンストラクタ.

引数

v	四元数を格納した GgVector 型の変数.
---	-------------------------

gg.h の 3485 行目に定義があります。

### 8.18.3 関数詳解

#### 8.18.3.1 add() [1/4]

```
GgQuaternion gg::GgQuaternion::add (
    const GgQuaternion & q ) const [inline]
```

四元数に別の四元数を加算した結果を返す.

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

`q` を加えた四元数.

`gg.h` の 3794 行目に定義がります。

呼び出し関係図:



### 8.18.3.2 add() [2/4]

```
GgQuaternion gg::GgQuaternion::add (
    const GgVector & v ) const [inline]
```

四元数に別の四元数を加算した結果を返す.

引数

<code>v</code>	四元数を格納した <code>GgVector</code> 型の変数.
----------------	--------------------------------------

戻り値

`v` を加えた四元数.

`gg.h` の 3783 行目に定義がります。

呼び出し関係図:



## 8.18.3.3 add() [3/4]

```
GgQuaternion gg::GgQuaternion::add (
    const GLfloat * a ) const [inline]
```

四元数に別の四元数を加算した結果を返す.

引数

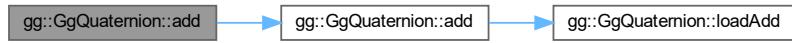
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------	--------------------------------

戻り値

*a* を加えた四元数.

gg.h の 3772 行目に定義があります。

呼び出し関係図:



## 8.18.3.4 add() [4/4]

```
GgQuaternion gg::GgQuaternion::add (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) const [inline]
```

四元数に別の四元数を加算した結果を返す.

引数

<i>x</i>	加える四元数の x 要素.
<i>y</i>	加える四元数の y 要素.
<i>z</i>	加える四元数の z 要素.
<i>w</i>	加える四元数の w 要素.

戻り値

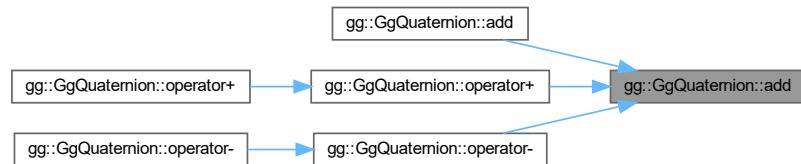
(*x*, *y*, *z*, *w*) を加えた四元数.

[gg.h](#) の 3760 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.5 conjugate()

[GgQuaternion](#) gg::GgQuaternion::conjugate ( ) const [inline]

共役四元数に変換する。

戻り値

共役四元数。

[gg.h](#) の 4417 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.18.3.6 divide() [1/4]

```
GgQuaternion gg::GgQuaternion::divide (
    const GgQuaternion & q ) const [inline]
```

四元数を別の四元数で除算した結果を返す。

引数

<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

*q* で割った四元数。

gg.h の 3943 行目に定義があります。

呼び出し関係図:



#### 8.18.3.7 divide() [2/4]

```
GgQuaternion gg::GgQuaternion::divide (
    const GgVector & v ) const [inline]
```

四元数を別の四元数で除算した結果を返す。

引数

<code>v</code>	四元数を格納した <code>GgVector</code> 型の変数.
----------------	--------------------------------------

戻り値

`v` で割った四元数.

`gg.h` の 3932 行目に定義があります。

呼び出し関係図:



#### 8.18.3.8 divide() [3/4]

```
GgQuaternion gg::GgQuaternion::divide (
    const GLfloat * a ) const [inline]
```

四元数を別の四元数で除算した結果を返す.

引数

<code>a</code>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
----------------	---

戻り値

`a` で割った四元数.

`gg.h` の 3918 行目に定義があります。

呼び出し関係図:



### 8.18.3.9 divide() [4/4]

```
GgQuaternion gg::GgQuaternion::divide (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) const [inline]
```

四元数を別の四元数で除算した結果を返す。

引数

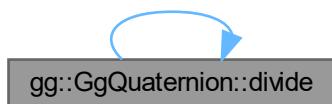
<i>x</i>	割る四元数の <i>x</i> 要素.
<i>y</i>	割る四元数の <i>y</i> 要素.
<i>z</i>	割る四元数の <i>z</i> 要素.
<i>w</i>	割る四元数の <i>w</i> 要素.

戻り値

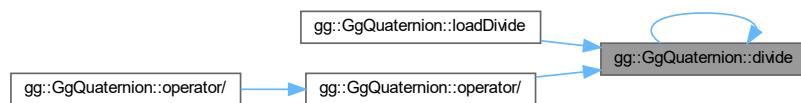
(*x*, *y*, *z*, *w*) を割った四元数.

gg.h の 3906 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.10 euler() [1/2]

```
GgQuaternion gg::GgQuaternion::euler (
    const GLfloat * e ) const [inline]
```

四元数をオイラー角 ( $e[0], e[1], e[2]$ ) で回転した四元数を返す.

引数

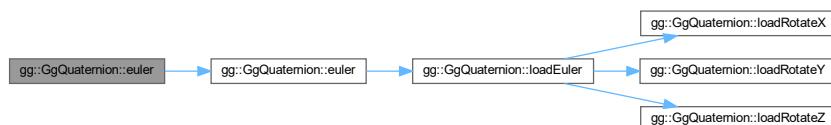
<b>e</b>	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転した四元数.

gg.h の 4257 行目に定義があります。

呼び出し関係図:



### 8.18.3.11 euler() [2/2]

```
GgQuaternion gg::GgQuaternion::euler (
    GLfloat heading,
    GLfloat pitch,
    GLfloat roll ) const [inline]
```

四元数をオイラー角 (heading, pitch, roll) で回転した四元数を返す.

引数

<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転した四元数.

gg.h の 4245 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.12 get()

```
void gg::GgQuaternion::get (
    GLfloat * a ) const [inline]
```

四元数を取り出す。

引数

a	四元数を格納する GLfloat 型の 4 要素の配列変数。
---	--------------------------------

gg.h の 4441 行目に定義がります。

### 8.18.3.13 getConjugateMatrix() [1/3]

```
GgMatrix gg::GgQuaternion::getConjugateMatrix ( ) const [inline]
```

四元数の共役が表す回転の変換行列を取り出す。

戻り値

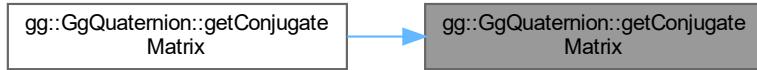
回転の変換を表す `GgMatrix` 型の変換行列.

`gg.h` の 4508 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.18.3.14 `getConjugateMatrix()` [2/3]

```
void gg::GgQuaternion::getConjugateMatrix (
    GgMatrix & m ) const [inline]
```

四元数の共役が表す回転の変換行列を `m` に求める.

引数

<code>m</code>	回転の変換行列を格納する <code>GgMatrix</code> 型の変数.
----------------	--

`gg.h` の 4498 行目に定義があります。

呼び出し関係図:



#### 8.18.3.15 getConjugateMatrix() [3/3]

```
void gg::GgQuaternion::getConjugateMatrix (
    GLfloat * a ) const [inline]
```

四元数の共役が表す回転の変換行列を a に求める。

引数

a	回転の変換行列を格納する GLfloat 型の 16 要素の配列変数。
---	-------------------------------------

gg.h の 4486 行目に定義があります。

呼び出し関係図:



#### 8.18.3.16 getMatrix() [1/3]

```
GgMatrix gg::GgQuaternion::getMatrix ( ) const [inline]
```

四元数が表す回転の変換行列を取り出す。

戻り値

回転の変換を表す [GgMatrix](#) 型の変換行列.

[gg.h](#) の 4474 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.18.3.17 `getMatrix()` [2/3]

```
void gg::GgQuaternion::getMatrix (
    GgMatrix & m ) const [inline]
```

四元数が表す回転の変換行列を `m` に求める.

引数

<code>m</code>	回転の変換行列を格納する <a href="#">GgMatrix</a> 型の変数.
----------------	---

[gg.h](#) の 4464 行目に定義があります。

呼び出し関係図:



#### 8.18.3.18 getMatrix() [3/3]

```
void gg::GgQuaternion::getMatrix (
    GLfloat * a ) const [inline]
```

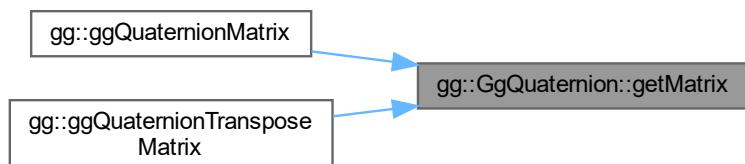
四元数が表す回転の変換行列を a に求める。

引数

a	回転の変換行列を格納する GLfloat 型の 16 要素の配列変数。
---	-------------------------------------

gg.h の 4454 行目に定義があります。

被呼び出し関係図:



#### 8.18.3.19 invert()

```
GgQuaternion gg::GgQuaternion::invert ( ) const [inline]
```

逆元に変換する。

戻り値

四元数の逆元.

`gg.h` の 4429 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.20 `load()` [1/4]

```
GgQuaternion & gg::GgQuaternion::load (
    const GgQuaternion & q ) [inline]
```

四元数を格納する.

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

設定した四元数.

`gg.h` の 3547 行目に定義があります。

### 8.18.3.21 load() [2/4]

```
GgQuaternion & gg::GgQuaternion::load (
    const GgVector & v ) [inline]
```

四元数を格納する。

引数

v	四元数を格納した GgVector 型の変数。
---	-------------------------

戻り値

設定した四元数。

gg.h の 3535 行目に定義があります。

### 8.18.3.22 load() [3/4]

```
GgQuaternion & gg::GgQuaternion::load (
    const GLfloat * a ) [inline]
```

四元数を格納する。

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数。
---	--------------------------------

戻り値

設定した四元数。

gg.h の 3524 行目に定義があります。

呼び出し関係図:



### 8.18.3.23 load() [4/4]

```
GgQuaternion & gg::GgQuaternion::load (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w )  [inline]
```

四元数を格納する。

引数

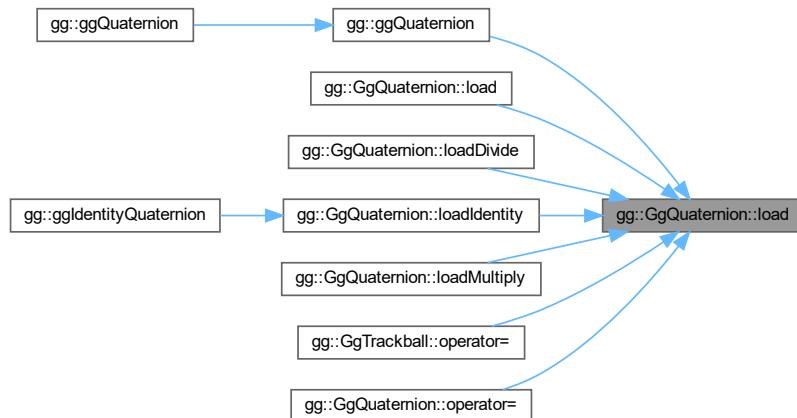
<i>x</i>	四元数の <i>x</i> 要素。
<i>y</i>	四元数の <i>y</i> 要素。
<i>z</i>	四元数の <i>z</i> 要素。
<i>w</i>	四元数の <i>w</i> 要素。

戻り値

設定した四元数。

gg.h の 3509 行目に定義があります。

被呼び出し関係図:



### 8.18.3.24 loadAdd() [1/4]

```
GgQuaternion & gg::GgQuaternion::loadAdd (
    const GgQuaternion & q )  [inline]
```

四元数に別の四元数を加算した結果を格納する。

引数

<code>q</code>	GgQuaternion 型の四元数.
----------------	---------------------

戻り値

`v` を加えた四元数.

gg.h の 3599 行目に定義がります。

呼び出し関係図:



#### 8.18.3.25 loadAdd() [2/4]

```
GgQuaternion & gg::GgQuaternion::loadAdd ( const GgVector & v ) [inline]
```

四元数に別の四元数を加算した結果を格納する.

引数

<code>v</code>	四元数を格納した GgVector 型の変数.
----------------	-------------------------

戻り値

`v` を加えた四元数.

gg.h の 3588 行目に定義がります。

呼び出し関係図:



### 8.18.3.26 loadAdd() [3/4]

```
GgQuaternion & gg::GgQuaternion::loadAdd (
    const GLfloat * a ) [inline]
```

四元数に別の四元数を加算した結果を格納する。

引数

<b>a</b>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------	--------------------------------

戻り値

**a** を加えた四元数.

gg.h の 3577 行目に定義があります。

呼び出し関係図:



### 8.18.3.27 loadAdd() [4/4]

```
GgQuaternion & gg::GgQuaternion::loadAdd (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline]
```

四元数に別の四元数を加算した結果を格納する。

引数

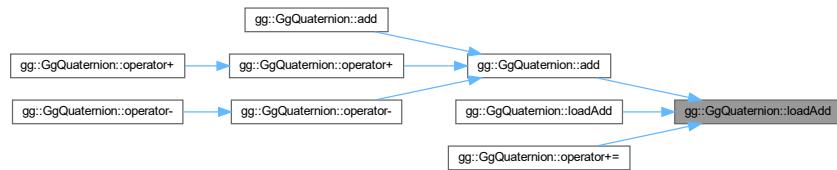
<b>x</b>	加える四元数の x 要素.
<b>y</b>	加える四元数の y 要素.
<b>z</b>	加える四元数の z 要素.
<b>w</b>	加える四元数の w 要素.

戻り値

$(x, y, z, w)$  を加えた四元数.

gg.h の 3562 行目に定義があります。

被呼び出し関係図:



### 8.18.3.28 loadConjugate() [1/2]

```
GgQuaternion & gg::GgQuaternion::loadConjugate (
    const GgQuaternion & q ) [inline]
```

引数に指定した四元数の共役四元数を格納する.

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

共役四元数.

gg.h の 4348 行目に定義があります。

呼び出し関係図:



### 8.18.3.29 loadConjugate() [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadConjugate (
    const GLfloat * a )
```

引数に指定した四元数の共役四元数を格納する。

引数

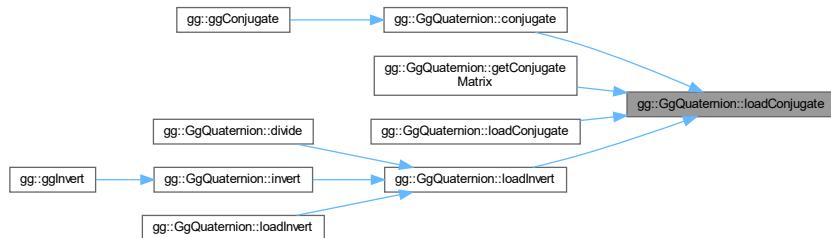
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
----------	--------------------------------

戻り値

共役四元数。

gg.cpp の 3326 行目に定義があります。

被呼び出し関係図:



### 8.18.3.30 loadDivide() [1/4]

```
GgQuaternion & gg::GgQuaternion::loadDivide (
    const GgQuaternion & q ) [inline]
```

四元を別の四元数で除算した結果を格納する。

引数

<i>q</i>	<code>GgQuaternion</code> 型の四元数。
----------	----------------------------------

戻り値

*q* で割った四元数。

gg.h の 3746 行目に定義があります。

呼び出し関係図:



#### 8.18.3.31 loadDivide() [2/4]

```
GgQuaternion & gg::GgQuaternion::loadDivide (
    const GgVector & v ) [inline]
```

四元を別の四元数で除算した結果を格納する。

引数

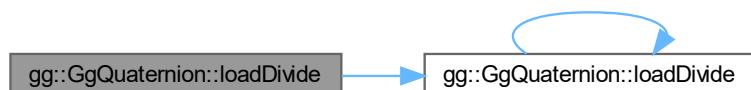
v	四元数を格納した GgVector 型の変数。
---	-------------------------

戻り値

v で割った四元数。

gg.h の 3735 行目に定義があります。

呼び出し関係図:



#### 8.18.3.32 loadDivide() [3/4]

```
GgQuaternion & gg::GgQuaternion::loadDivide (
    const GLfloat * a ) [inline]
```

四元を別の四元数で除算した結果を格納する。

引数

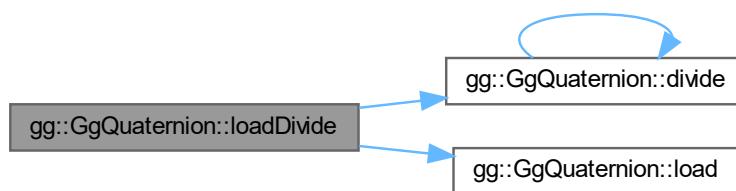
a	四元数を格納した GLfloat 型の 4 要素の配列変数。
---	--------------------------------

戻り値

a で割った四元数。

gg.h の 3724 行目に定義があります。

呼び出し関係図:



### 8.18.3.33 loadDivide() [4/4]

```
GgQuaternion & gg::GgQuaternion::loadDivide (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline]
```

四元を別の四元数で除算した結果を格納する。

引数

x	割る四元数の x 要素。
y	割る四元数の y 要素。
z	割る四元数の z 要素。
w	割る四元数の w 要素。

戻り値

(x, y, z, w) を割った四元数。

gg.h の 3712 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.34 loadEuler() [1/2]

```
GgQuaternion & gg::GgQuaternion::loadEuler (
    const GLfloat * e ) [inline]
```

オイラー角 ( $e[0], e[1], e[2]$ ) で与えられた回転を表す四元数を格納する.

引数

<code>e</code>	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------------	---

戻り値

格納した回転を表す四元数.

gg.h の 4232 行目に定義があります。

呼び出し関係図:



### 8.18.3.35 loadEuler() [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadEuler (
    GLfloat heading,
    GLfloat pitch,
    GLfloat roll )
```

オイラー角 (heading, pitch, roll) で与えられた回転を表す四元数を格納する。

引数

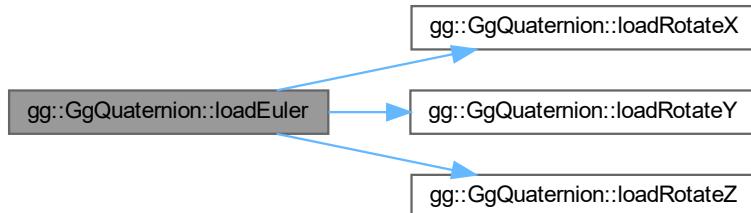
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

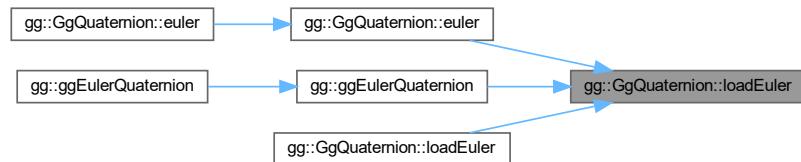
格納した回転を表す四元数。

gg.cpp の 3295 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.36 loadIdentity()

```
GgQuaternion & gg::GgQuaternion::loadIdentity ( ) [inline]
```

単位元を格納する。

戻り値

格納された単位元。

gg.h の 4082 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.37 loadInvert() [1/2]

```
GgQuaternion & gg::GgQuaternion::loadInvert ( const GgQuaternion & q ) [inline]
```

引数に指定した四元数の逆元を格納する。

引数

<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

四元数の逆元.

`gg.h` の 4367 行目に定義があります。

呼び出し関係図:



### 8.18.3.38 `loadInvert()` [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadInvert (
    const GLfloat * a )
```

引数に指定した四元数の逆元を格納する.

引数

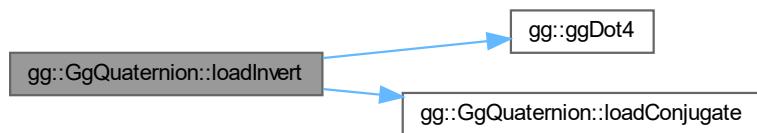
a	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
---	---

戻り値

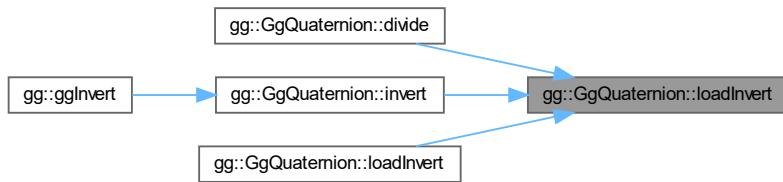
四元数の逆元.

`gg.cpp` の 3340 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.39 loadMatrix() [1/2]

```
GgQuaternion & gg::GgQuaternion::loadMatrix (
    const GgMatrix & m ) [inline]
```

回転の変換行列 `m` を表す四元数を格納する。

引数

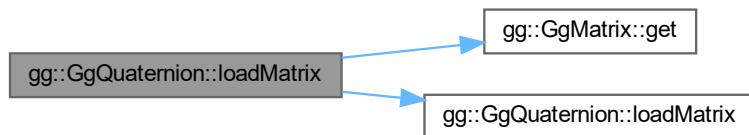
<code>m</code>	<code>Ggmatrix</code> 型の変換行列。
----------------	-------------------------------

戻り値

`m` による回転の変換に相当する四元数。

`gg.h` の 4072 行目に定義があります。

呼び出し関係図:



### 8.18.3.40 loadMatrix() [2/2]

```
GgQuaternion & gg::GgQuaternion::loadMatrix (
    const GLfloat * a ) [inline]
```

回転の変換行列を表す四元数を格納する。

引数

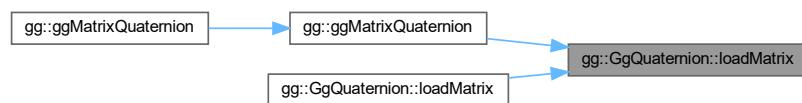
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 4060 行目に定義があります。

呼び出し関係図:



#### 8.18.3.41 loadMultiply() [1/4]

```
GgQuaternion & gg::GgQuaternion::loadMultiply (
    const GgQuaternion & q ) [inline]
```

四元数に別の四元数を乗算した結果を格納する.

引数

q	GgQuaternion 型の四元数.
---	---------------------

戻り値

q を乗じた四元数.

gg.h の 3698 行目に定義があります。

呼び出し関係図:



### 8.18.3.42 loadMultiply() [2/4]

```
GgQuaternion & gg::GgQuaternion::loadMultiply (
    const GgVector & v ) [inline]
```

四元数に別の四元数を乗算した結果を格納する.

引数

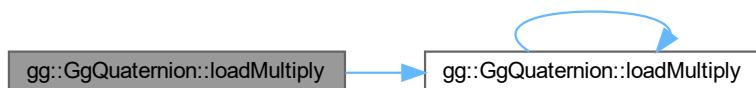
v	四元数を格納した GgVector 型の変数.
---	-------------------------

戻り値

v を乗じた四元数.

gg.h の 3687 行目に定義があります。

呼び出し関係図:



### 8.18.3.43 loadMultiply() [3/4]

```
GgQuaternion & gg::GgQuaternion::loadMultiply (
    const GLfloat * a ) [inline]
```

四元数に別の四元数を乗算した結果を格納する.

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a を乗じた四元数.

gg.h の 3676 行目に定義がります。

呼び出し関係図:



#### 8.18.3.44 loadMultiply() [4/4]

```
GgQuaternion & gg::GgQuaternion::loadMultiply (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline]
```

四元数に別の四元数を乗算した結果を格納する。

引数

<i>x</i>	掛ける四元数の <i>x</i> 要素。
<i>y</i>	掛ける四元数の <i>y</i> 要素。
<i>z</i>	掛ける四元数の <i>z</i> 要素。
<i>w</i>	掛ける四元数の <i>w</i> 要素。

戻り値

(*x*, *y*, *z*, *w*) を掛けた四元数。

gg.h の 3664 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



#### 8.18.3.45 loadNormalize() [1/2]

```
GgQuaternion & gg::GgQuaternion::loadNormalize (
    const GgQuaternion & q ) [inline]
```

引数に指定した四元数を正規化して格納する。

引数

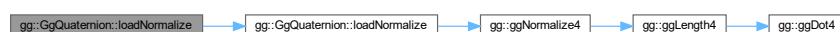
<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

正規化された四元数。

`gg.h` の 4329 行目に定義があります。

呼び出し関係図:



#### 8.18.3.46 loadNormalize() [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadNormalize (
    const GLfloat * a )
```

引数に指定した四元数を正規化して格納する。

引数

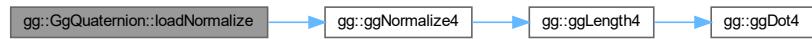
<code>a</code>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
----------------	---

戻り値

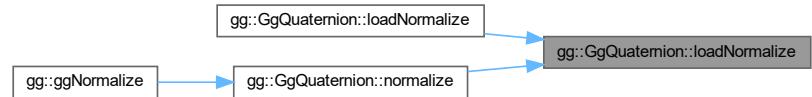
正規化された四元数.

`gg.cpp` の 3311 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.18.3.47 `loadRotate()` [1/3]

```
GgQuaternion & gg::GgQuaternion::loadRotate (
    const GLfloat * v) [inline]
```

( $v[0], v[1], v[2]$ ) を軸として角度  $v[3]$  回転する四元数を格納する.

引数

<code>v</code>	軸ベクトルと回転角を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
----------------	---

戻り値

格納された回転を表す四元数.

`gg.h` の 4116 行目に定義があります。

呼び出し関係図:



#### 8.18.3.48 loadRotate() [2/3]

```
GgQuaternion & gg::GgQuaternion::loadRotate (
    const GLfloat * v,
    GLfloat a ) [inline]
```

(v[0], v[1], v[2]) を軸として角度 a 回転する四元数を格納する。

引数

<i>v</i>	軸ベクトルを表す GLfloat 型の 3 要素の配列変数。
<i>a</i>	回転角。

戻り値

格納された回転を表す四元数。

gg.h の 4105 行目に定義があります。

呼び出し関係図:



#### 8.18.3.49 loadRotate() [3/3]

```
gg::GgQuaternion & gg::GgQuaternion::loadRotate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a )
```

(x, y, z) を軸として角度 a 回転する四元数を格納する。

引数

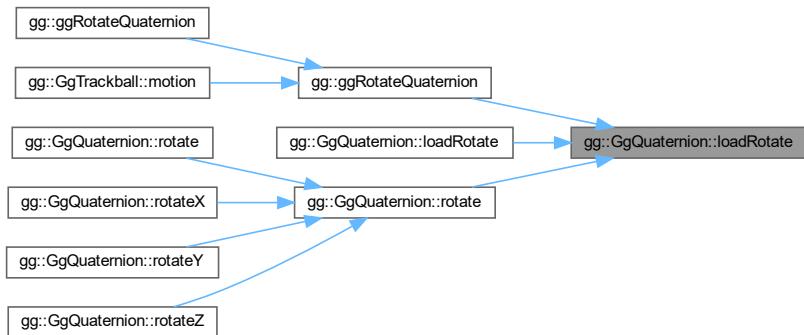
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

格納された回転を表す四元数.

gg.cpp の 3229 行目に定義があります。

被呼び出し関係図:



### 8.18.3.50 loadRotateX()

```
gg::GgQuaternion & gg::GgQuaternion::loadRotateX (
    GLfloat a )
```

x 軸中心に角度 *a* 回転する四元数を格納する.

引数

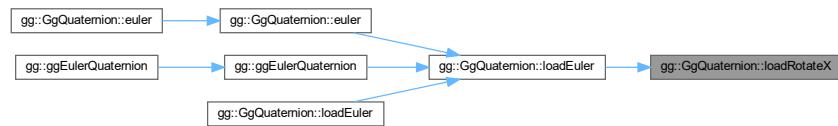
<i>a</i>	回転角.
----------	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3253 行目に定義があります。

被呼び出し関係図:



### 8.18.3.51 loadRotateY()

```
gg::GgQuaternion & gg::GgQuaternion::loadRotateY (
    GLfloat a )
```

y 軸中心に角度 a 回転する四元数を格納する.

引数

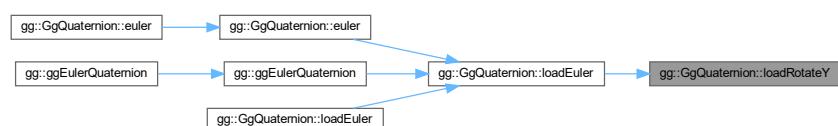
<b>a</b>	回転角.
----------	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3267 行目に定義があります。

被呼び出し関係図:



### 8.18.3.52 loadRotateZ()

```
gg::GgQuaternion & gg::GgQuaternion::loadRotateZ (
    GLfloat a )
```

z 軸中心に角度 a 回転する四元数を格納する.

引数

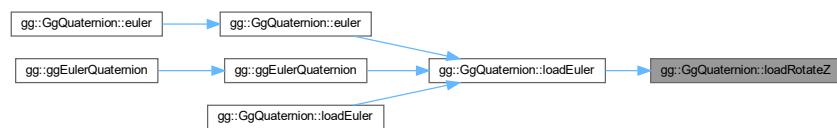
a	回転角.
---	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3281 行目に定義があります。

被呼び出し関係図:



### 8.18.3.53 loadSlerp() [1/4]

```
GgQuaternion & gg::GgQuaternion::loadSlerp (
    const GgQuaternion & q,
    const GgQuaternion & r,
    GLfloat t ) [inline]
```

球面線形補間の結果を格納する.

引数

q	GgQuaternion 型の四元数.
r	GgQuaternion 型の四元数.
t	補間パラメータ.

戻り値

格納した q, r を t で内分した四元数.

gg.h の 4284 行目に定義があります。

呼び出し関係図:



#### 8.18.3.54 loadSlerp() [2/4]

```
GgQuaternion & gg::GgQuaternion::loadSlerp (
    const GgQuaternion & q,
    const GLfloat * a,
    GLfloat t ) [inline]
```

球面線形補間の結果を格納する.

引数

<i>q</i>	GgQuaternion 型の四元数.
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した *q*, *a* を *t* で内分した四元数.

gg.h の 4297 行目に定義があります。

呼び出し関係図:



### 8.18.3.55 loadSlerp() [3/4]

```
GgQuaternion & gg::GgQuaternion::loadSlerp (
    const GLfloat * a,
    const GgQuaternion & q,
    GLfloat t ) [inline]
```

球面線形補間の結果を格納する。

引数

<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>q</i>	GgQuaternion 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

格納した *a*, *q* を *t* で内分した四元数.

gg.h の 4310 行目に定義があります。

呼び出し関係図:



### 8.18.3.56 loadSlerp() [4/4]

```
GgQuaternion & gg::GgQuaternion::loadSlerp (
    const GLfloat * a,
    const GLfloat * b,
    GLfloat t ) [inline]
```

球面線形補間の結果を格納する。

引数

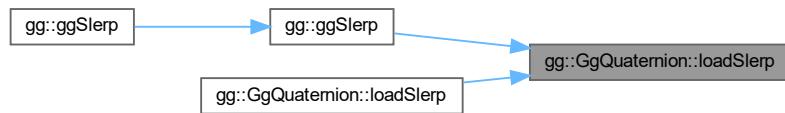
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した  $a, b$  を  $t$  で内分した四元数.

gg.h の 4270 行目に定義があります。

被呼び出し関係図:



#### 8.18.3.57 loadSubtract() [1/4]

```
GgQuaternion & gg::GgQuaternion::loadSubtract (const GgQuaternion & q) [inline]
```

四元数から別の四元数を減算した結果を格納する.

引数

<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

$q$  を引いた四元数.

gg.h の 3650 行目に定義があります。

呼び出し関係図:



### 8.18.3.58 loadSubtract() [2/4]

```
GgQuaternion & gg::GgQuaternion::loadSubtract (
    const GgVector & v ) [inline]
```

四元数から別の四元数を減算した結果を格納する。

引数

<b>v</b>	四元数を格納した <code>GgVector</code> 型の変数。
----------	--------------------------------------

戻り値

`v` を引いた四元数。

`gg.h` の 3639 行目に定義があります。

呼び出し関係図:



### 8.18.3.59 loadSubtract() [3/4]

```
GgQuaternion & gg::GgQuaternion::loadSubtract (
    const GLfloat * a ) [inline]
```

四元数から別の四元数を減算した結果を格納する。

引数

<b>a</b>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
----------	---

戻り値

`a` を引いた四元数。

`gg.h` の 3628 行目に定義があります。

呼び出し関係図:



### 8.18.3.60 loadSubtract() [4/4]

```
GgQuaternion & gg::GgQuaternion::loadSubtract (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) [inline]
```

四元数から別の四元数を減算した結果を格納する。

引数

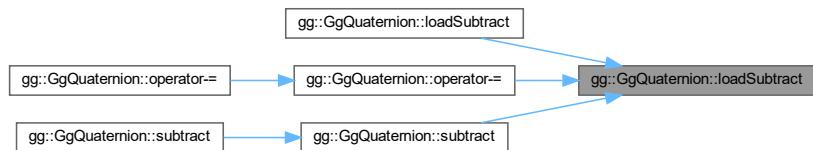
<i>x</i>	引く四元数の x 要素.
<i>y</i>	引く四元数の y 要素.
<i>z</i>	引く四元数の z 要素.
<i>w</i>	引く四元数の w 要素.

戻り値

(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 3613 行目に定義があります。

被呼び出し関係図:



### 8.18.3.61 multiply() [1/4]

```
GgQuaternion gg::GgQuaternion::multiply (
    const GgQuaternion & q ) const [inline]
```

四元数に別の四元数を乗算した結果を返す.

引数

<i>q</i>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

*q* を掛けた四元数.

gg.h の 3892 行目に定義があります。

### 8.18.3.62 multiply() [2/4]

```
GgQuaternion gg::GgQuaternion::multiply (
    const GgVector & v ) const [inline]
```

四元数に別の四元数を乗算した結果を返す.

引数

<i>v</i>	四元数を格納した GgVector 型の変数.
----------	-------------------------

戻り値

*v* を掛けた四元数.

gg.h の 3881 行目に定義があります。

### 8.18.3.63 multiply() [3/4]

```
GgQuaternion gg::GgQuaternion::multiply (
    const GLfloat * a ) const [inline]
```

四元数に別の四元数を乗算した結果を返す.

引数

<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------	--------------------------------

戻り値

a を掛けた四元数.

gg.h の 3868 行目に定義がります。

#### 8.18.3.64 multiply() [4/4]

```
GgQuaternion gg::GgQuaternion::multiply (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) const [inline]
```

四元数に別の四元数を乗算した結果を返す.

引数

x	掛ける四元数の x 要素.
y	掛ける四元数の y 要素.
z	掛ける四元数の z 要素.
w	掛ける四元数の w 要素.

戻り値

(x, y, z, w) を掛けた四元数.

gg.h の 3856 行目に定義がります。

#### 8.18.3.65 norm()

```
GLfloat gg::GgQuaternion::norm () const [inline]
```

四元数のノルムを求める.

戻り値

四元数のノルム.

gg.h の 3495 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.66 normalize()

`GgQuaternion gg::GgQuaternion::normalize () const [inline]`

正規化する.

戻り値

正規化された四元数.

`gg.h` の 4405 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



**8.18.3.67 operator\*() [1/3]**

```
GgQuaternion gg::GgQuaternion::operator* (
    const GgQuaternion & q ) const [inline]
```

gg.h の 4037 行目に定義があります。

呼び出し関係図:

**8.18.3.68 operator\*() [2/3]**

```
GgQuaternion gg::GgQuaternion::operator* (
    const GgVector & v ) const [inline]
```

gg.h の 4033 行目に定義があります。

**8.18.3.69 operator\*() [3/3]**

```
GgQuaternion gg::GgQuaternion::operator* (
    const GLfloat * a ) const [inline]
```

gg.h の 4029 行目に定義があります。

被呼び出し関係図:

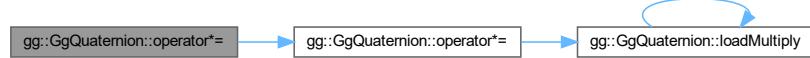


### 8.18.3.70 operator\*=( ) [1/3]

```
GgQuaternion & gg::GgQuaternion::operator*=
( const GgQuaternion & q ) [inline]
```

gg.h の 3989 行目に定義があります。

呼び出し関係図:



### 8.18.3.71 operator\*=( ) [2/3]

```
GgQuaternion & gg::GgQuaternion::operator*=
( const GgVector & v ) [inline]
```

gg.h の 3985 行目に定義があります。

呼び出し関係図:

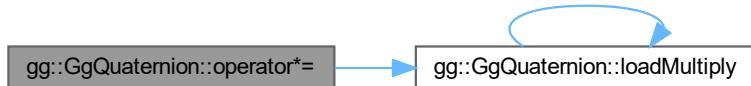


### 8.18.3.72 operator\*=( ) [3/3]

```
GgQuaternion & gg::GgQuaternion::operator*=
( const GLfloat * a ) [inline]
```

gg.h の 3981 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

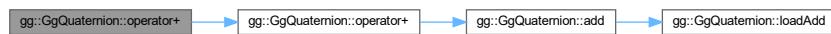


### 8.18.3.73 operator+() [1/3]

```
GgQuaternion gg::GgQuaternion::operator+ (
    const GgQuaternion & q ) const [inline]
```

gg.h の 4013 行目に定義があります。

呼び出し関係図:



### 8.18.3.74 operator+() [2/3]

```
GgQuaternion gg::GgQuaternion::operator+ (
    const GgVector & v ) const [inline]
```

gg.h の 4009 行目に定義があります。

呼び出し関係図:

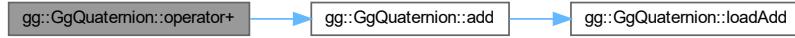


### 8.18.3.75 operator+() [3/3]

```
GgQuaternion gg::GgQuaternion::operator+ (
    const GLfloat * a ) const [inline]
```

gg.h の 4005 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.76 operator+=() [1/3]

```
GgQuaternion & gg::GgQuaternion::operator+= (
    const GgQuaternion & q ) [inline]
```

gg.h の 3965 行目に定義があります。

呼び出し関係図:



**8.18.3.77 operator+=( ) [2/3]**

```
GgQuaternion & gg::GgQuaternion::operator+= ( const GgVector & v ) [inline]
```

gg.h の 3961 行目に定義があります。

呼び出し関係図:

**8.18.3.78 operator+=( ) [3/3]**

```
GgQuaternion & gg::GgQuaternion::operator+= ( const GLfloat * a ) [inline]
```

gg.h の 3957 行目に定義があります。

呼び出し関係図:

**8.18.3.79 operator-() [1/3]**

```
GgQuaternion gg::GgQuaternion::operator- ( const GgQuaternion & q ) const [inline]
```

gg.h の 4025 行目に定義があります。

呼び出し関係図:



### 8.18.3.80 operator-() [2/3]

```
GgQuaternion gg::GgQuaternion::operator- (
    const GgVector & v ) const [inline]
```

gg.h の 4021 行目に定義があります。

呼び出し関係図:

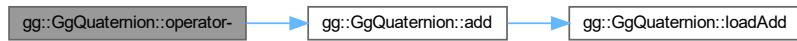


### 8.18.3.81 operator-() [3/3]

```
GgQuaternion gg::GgQuaternion::operator- (
    const GLfloat * a ) const [inline]
```

gg.h の 4017 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



**8.18.3.82 operator-() [1/3]**

```
GgQuaternion & gg::GgQuaternion::operator-= ( const GgQuaternion & q ) [inline]
```

gg.h の 3977 行目に定義がります。

呼び出し関係図:

**8.18.3.83 operator-() [2/3]**

```
GgQuaternion & gg::GgQuaternion::operator-= ( const GgVector & v ) [inline]
```

gg.h の 3973 行目に定義がります。

呼び出し関係図:

**8.18.3.84 operator-() [3/3]**

```
GgQuaternion & gg::GgQuaternion::operator-= ( const GLfloat * a ) [inline]
```

gg.h の 3969 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:

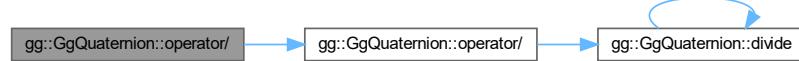


### 8.18.3.85 operator() [1/3]

```
GgQuaternion gg::GgQuaternion::operator/ (
    const GgQuaternion & q ) const [inline]
```

`gg.h` の 4049 行目に定義があります。

呼び出し関係図:



### 8.18.3.86 operator() [2/3]

```
GgQuaternion gg::GgQuaternion::operator/ (
    const GgVector & v ) const [inline]
```

`gg.h` の 4045 行目に定義があります。

呼び出し関係図:

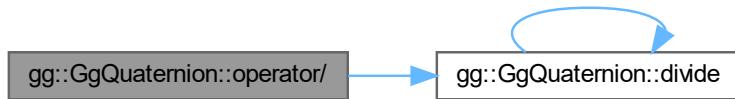


**8.18.3.87 operator/() [3/3]**

```
GgQuaternion gg::GgQuaternion::operator/ (
    const GLfloat * a ) const [inline]
```

gg.h の 4041 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

**8.18.3.88 operator/=(()) [1/3]**

```
GgQuaternion & gg::GgQuaternion::operator/= (
    const GgQuaternion & q ) [inline]
```

gg.h の 4001 行目に定義があります。

呼び出し関係図:

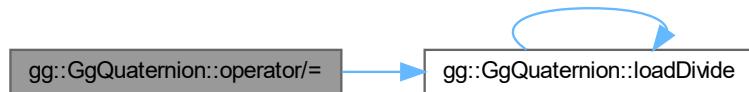


### 8.18.3.89 operator/() [2/3]

```
GgQuaternion & gg::GgQuaternion::operator/= (  
    const GgVector & v ) [inline]
```

gg.h の 3997 行目に定義があります。

呼び出し関係図:

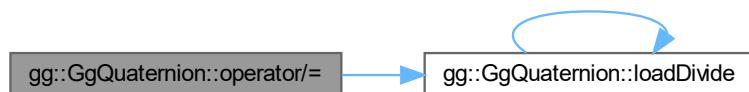


### 8.18.3.90 operator/() [3/3]

```
GgQuaternion & gg::GgQuaternion::operator/= (  
    const GLfloat * a ) [inline]
```

gg.h の 3993 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



**8.18.3.91 operator=() [1/2]**

```
GgQuaternion & gg::GgQuaternion::operator= (
    const GgVector & v ) [inline]
```

gg.h の 3953 行目に定義があります。

呼び出し関係図:

**8.18.3.92 operator=() [2/2]**

```
GgQuaternion & gg::GgQuaternion::operator= (
    const GLfloat * a ) [inline]
```

gg.h の 3949 行目に定義があります。

呼び出し関係図:

**8.18.3.93 rotate() [1/3]**

```
GgQuaternion gg::GgQuaternion::rotate (
    const GLfloat * v ) const [inline]
```

四元数を ( $v[0], v[1], v[2]$ ) を軸として角度  $v[3]$  回転した四元数を返す。

引数

<code>v</code>	軸ベクトルを表す <code>GLfloat</code> 型の 4 要素の配列変数。
----------------	---

戻り値

回転した四元数.

[gg.h の 4178 行目](#)に定義があります。

呼び出し関係図:



#### 8.18.3.94 `rotate()` [2/3]

```
GgQuaternion gg::GgQuaternion::rotate (
    const GLfloat * v,
    GLfloat a ) const [inline]
```

四元数を ( $v[0], v[1], v[2]$ ) を軸として角度  $a$  回転した四元数を返す.

引数

$v$	軸ベクトルを表す <code>GLfloat</code> 型の 3 要素の配列変数.
$a$	回転角.

戻り値

回転した四元数.

[gg.h の 4167 行目](#)に定義があります。

呼び出し関係図:



8.18.3.95 `rotate()` [3/3]

```
GgQuaternion gg::GgQuaternion::rotate (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat a ) const [inline]
```

四元数を  $(x, y, z)$  を軸として角度  $a$  回転した四元数を返す。

引数

$x$	軸ベクトルの $x$ 成分.
$y$	軸ベクトルの $y$ 成分.
$z$	軸ベクトルの $z$ 成分.
$a$	回転角.

戻り値

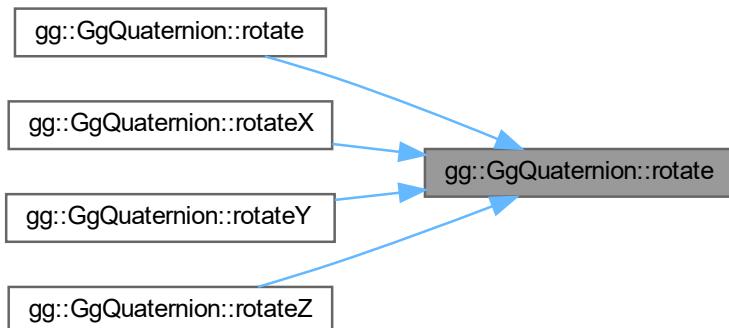
回転した四元数.

gg.h の 4154 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.18.3.96 rotateX()

```
GgQuaternion gg::GgQuaternion::rotateX (
    GLfloat a ) const [inline]
```

四元数を x 軸中心に角度 a 回転した四元数を返す.

引数

a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4189 行目に定義があります。

呼び出し関係図:



### 8.18.3.97 rotateY()

```
GgQuaternion gg::GgQuaternion::rotateY (
    GLfloat a ) const [inline]
```

四元数を y 軸中心に角度 a 回転した四元数を返す.

引数

a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4200 行目に定義があります。

呼び出し関係図:



#### 8.18.3.98 rotateZ()

```
GgQuaternion gg::GgQuaternion::rotateZ (
    GLfloat a ) const [inline]
```

四元数を z 軸中心に角度 a 回転した四元数を返す.

引数

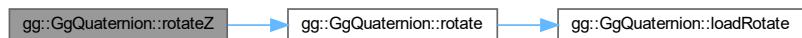
a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4211 行目に定義があります。

呼び出し関係図:



#### 8.18.3.99 slerp() [1/2]

```
GgQuaternion gg::GgQuaternion::slerp (
    const GgQuaternion & q,
    GLfloat t ) const [inline]
```

球面線形補間の結果を返す.

引数

<i>q</i>	<code>GgQuaternion</code> 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

四元数を *q* に対して *t* で内分した結果.

`gg.h` の [4393](#) 行目に定義があります。

#### 8.18.3.100 `slerp()` [2/2]

```
GgQuaternion gg::GgQuaternion::slerp (
    GLfloat * a,
    GLfloat t ) const [inline]
```

球面線形補間の結果を返す.

引数

<i>a</i>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

四元数を *a* に対して *t* で内分した結果.

`gg.h` の [4379](#) 行目に定義があります。

#### 8.18.3.101 `subtract()` [1/4]

```
GgQuaternion gg::GgQuaternion::subtract (
    const GgQuaternion & q ) const [inline]
```

四元数から別の四元数を減算した結果を返す.

引数

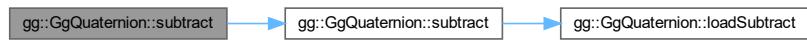
<i>q</i>	<code>GgQuaternion</code> 型の四元数.
----------	----------------------------------

戻り値

v を引いた四元数.

gg.h の 3842 行目に定義があります。

呼び出し関係図:



### 8.18.3.102 subtract() [2/4]

```
GgQuaternion gg::GgQuaternion::subtract (
    const GgVector & v ) const [inline]
```

四元数から別の四元数を減算した結果を返す.

引数

v	四元数を格納した GgVector 型の変数.
---	-------------------------

戻り値

v を引いた四元数.

gg.h の 3831 行目に定義があります。

呼び出し関係図:



### 8.18.3.103 subtract() [3/4]

```
GgQuaternion gg::GgQuaternion::subtract (
    const GLfloat * a ) const [inline]
```

四元数から別の四元数を減算した結果を返す.

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a を引いた四元数.

gg.h の 3820 行目に定義がります。

呼び出し関係図:



#### 8.18.3.104 subtract() [4/4]

```
GgQuaternion gg::GgQuaternion::subtract (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w ) const [inline]
```

四元数から別の四元数を減算した結果を返す.

引数

x	引く四元数の x 要素.
y	引く四元数の y 要素.
z	引く四元数の z 要素.
w	引く四元数の w 要素.

戻り値

(x, y, z, w) を引いた四元数.

gg.h の 3808 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.19 gg::GgShader クラス

```
#include <gg.h>
```

### 公開メンバ関数

- [GgShader](#) (const std::string &vert, const std::string &frag="", const std::string &geom="", int nvarying=0, const char \*const \*varyings=nullptr)
- [GgShader](#) (const std::array< std::string, 3 > &files, int nvarying=0, const char \*const \*varyings=nullptr)
- [GgShader](#) (const GgShader &o)=delete
- virtual [~GgShader](#) ()
- [GgShader & operator=](#) (const GgShader &o)=delete
- void [use](#) () const
- void [unuse](#) () const
- GLuint [get](#) () const

### 8.19.1 詳解

シェーダの基底クラス.

覚え書き

シェーダのクラスはこのクラスを派生して作る.

[gg.h](#) の 6797 行目に定義があります。

## 8.19.2 構築子と解体子

### 8.19.2.1 GgShader() [1/3]

```
gg::GgShader::GgShader (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    int nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

[gg.h の 6813 行目](#)に定義があります。

呼び出し関係図:



### 8.19.2.2 GgShader() [2/3]

```
gg::GgShader::GgShader (
    const std::array< std::string, 3 > & files,
    int nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列.
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト(nullptrなら不使用).

gg.h の 6831 行目に定義がります。

### 8.19.2.3 GgShader() [3/3]

```
gg::GgShader::GgShader (
    const GgShader & o ) [delete]
```

コピー コンストラクタは使用禁止。

### 8.19.2.4 ~GgShader()

```
virtual gg::GgShader::~GgShader () [inline], [virtual]
```

デストラクタ。

gg.h の 6848 行目に定義がります。

## 8.19.3 関数詳解

### 8.19.3.1 get()

```
GLuint gg::GgShader::get () const [inline]
```

シェーダのプログラム名を得る。

戻り値

シェーダのプログラム名。

gg.h の 6881 行目に定義がります。

### 8.19.3.2 operator=()

```
GgShader & gg::GgShader::operator= (
    const GgShader & o ) [delete]
```

代入演算子は使用禁止。

### 8.19.3.3 unuse()

```
void gg::GgShader::unuse ( ) const [inline]
```

シェーダプログラムの使用を終了する。

[gg.h](#) の 6871 行目に定義があります。

### 8.19.3.4 use()

```
void gg::GgShader::use ( ) const [inline]
```

シェーダプログラムの使用を開始する。

[gg.h](#) の 6863 行目に定義があります。

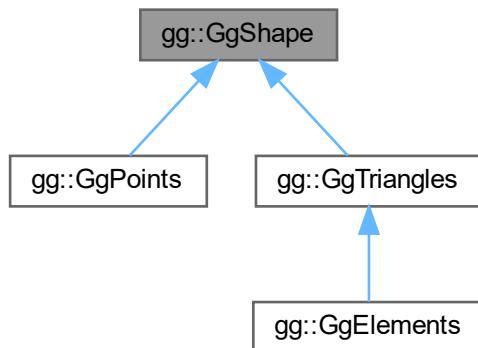
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

## 8.20 gg::GgShape クラス

```
#include <gg.h>
```

gg::GgShape の継承関係図



## 公開メンバ関数

- `GgShape (GLenum mode=0)`
- `virtual ~GgShape ()`
- `virtual operator bool () const noexcept`
- `virtual bool operator! () const noexcept`
- `const GLuint & get () const`
- `void setMode (GLenum mode)`
- `const GLenum & getMode () const`
- `virtual void draw (GLint first=0, GLsizei count=0) const`

### 8.20.1 詳解

形状データの基底クラス。

覚え書き

形状データのクラスはこのクラスを派生して作る。基本図形の種類と頂点配列オブジェクトを保持する。

`gg.h` の 6173 行目に定義があります。

### 8.20.2 構築子と解体子

#### 8.20.2.1 GgShape()

```
gg::GgShape::GgShape (
    GLenum mode = 0 ) [inline]
```

コンストラクタ。

引数

<code>mode</code>	基本図形の種類。
-------------------	----------

`gg.h` の 6188 行目に定義があります。

#### 8.20.2.2 ~GgShape()

```
virtual gg::GgShape::~GgShape () [inline], [virtual]
```

デストラクタ。

`gg.h` の 6197 行目に定義があります。

### 8.20.3 関数詳解

#### 8.20.3.1 draw()

```
virtual void gg::GgShape::draw (
    GLint first = 0,
    GLsizei count = 0 ) const [inline], [virtual]
```

図形の描画, 派生クラスでこの手続きをオーバーライドする.

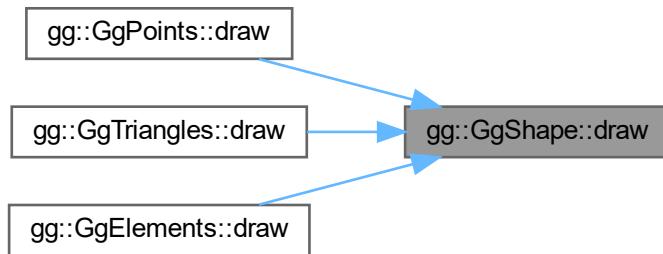
引数

<i>first</i>	描画する最初のアイテム.
<i>count</i>	描画するアイテムの数, 0 なら全部のアイテムを描画する.

[gg::GgPoints](#), [gg::GgTriangles](#), [gg::GgElements](#)で再実装されています。

gg.h の 6257 行目に定義があります。

被呼び出し関係図:



#### 8.20.3.2 get()

```
const GLuint & gg::GgShape::get () const [inline]
```

頂点配列オブジェクト名を取り出す.

戻り値

頂点配列オブジェクト名.

gg.h の 6226 行目に定義があります。

被呼び出し関係図:



#### 8.20.3.3 getMode()

```
const GLenum & gg::GgShape::getMode ( ) const [inline]
```

基本図形の検査.

戻り値

この頂点配列オブジェクトの基本図形の種類.

gg.h の 6246 行目に定義があります。

#### 8.20.3.4 operator bool()

```
virtual gg::GgShape::operator bool ( ) const [inline], [explicit], [virtual], [noexcept]
```

頂点配列オブジェクトが有効かどうか調べる.

戻り値

頂点配列オブジェクトが有効なら true

gg::GgPoints で再実装されています。

gg.h の 6206 行目に定義があります。

### 8.20.3.5 operator”!()

```
virtual bool gg::GgShape::operator! ( ) const [inline], [virtual], [noexcept]
```

頂点配列オブジェクトが有効かどうかの結果を反転する。

戻り値

頂点配列オブジェクトが有効なら `false`, 無効なら `true`.

`gg::GgPoints`で再実装されています。

`gg.h` の 6216 行目に定義があります。

### 8.20.3.6 setMode()

```
void gg::GgShape::setMode (
    GLenum mode ) [inline]
```

基本図形の設定。

引数

<code>mode</code>	基本図形の種類.
-------------------	----------

`gg.h` の 6236 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- `gg.h`

## 8.21 gg::GgSimpleObj クラス

```
#include <gg.h>
```

公開メンバ関数

- `GgSimpleObj` (const std::string &name, bool normalize=false)
- virtual ~`GgSimpleObj` ()  
デストラクタ.
- `operator bool` () const noexcept
- bool `operator!` () const noexcept
- const `GgTriangles` \* `get` () const
- virtual void `draw` (GLint first=0, GLsizei count=0) const

### 8.21.1 詳解

Wavefront OBJ 形式のファイル (Arrays 形式).

gg.h の 8122 行目に定義があります。

### 8.21.2 構築子と解体子

#### 8.21.2.1 GgSimpleObj()

```
gg::GgSimpleObj::GgSimpleObj (
    const std::string & name,
    bool normalize = false )
```

コンストラクタ.

引数

<i>name</i>	三角形分割された Alias OBJ 形式のファイルのファイル名.
<i>normalize</i>	true なら図形のサイズを [-1, 1] に正規化する.

gg.cpp の 5969 行目に定義があります。

呼び出し関係図:



#### 8.21.2.2 ~GgSimpleObj()

```
virtual gg::GgSimpleObj::~GgSimpleObj ( ) [inline], [virtual]
```

デストラクタ.

gg.h の 8145 行目に定義があります。

### 8.21.3 関数詳解

#### 8.21.3.1 draw()

```
void gg::GgSimpleObj::draw (
    GLint first = 0,
    GLsizei count = 0 ) const [virtual]
```

Wavefront OBJ 形式のデータを描画する手続き.

引数

<i>first</i>	描画する最初のverts番号.
<i>count</i>	描画するvertsの数, 0なら全部のvertsを描く.

[gg.cpp](#) の 5997 行目に定義があります。

#### 8.21.3.2 get()

```
const GgTriangles * gg::GgSimpleObj::get () const [inline]
```

形状データの取り出し.

戻り値

[GgTriangles](#) 型の形状データのポインタ.

[gg.h](#) の 8174 行目に定義があります。

呼び出し関係図:



### 8.21.3.3 operator bool()

```
gg::GgSimpleObj::operator bool () const [inline], [explicit], [noexcept]
```

オブジェクトが有効かどうか調べる。

戻り値

オブジェクトが有効なら true

gg.h の 8154 行目に定義があります。

### 8.21.3.4 operator”!”()

```
bool gg::GgSimpleObj::operator! () const [inline], [noexcept]
```

オブジェクトが有効かどうかの結果を反転する。

戻り値

オブジェクトが有効なら false, 無効なら true.

gg.h の 8164 行目に定義があります。

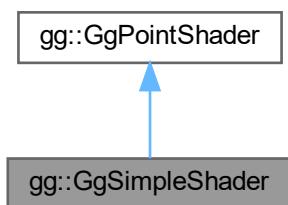
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

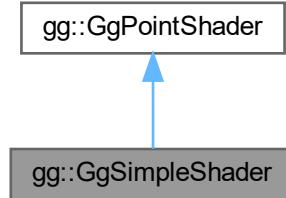
## 8.22 gg::GgSimpleShader クラス

```
#include <gg.h>
```

gg::GgSimpleShader の継承関係図



gg::GgSimpleShader 連携図



## クラス

- struct [Light](#)
- class [LightBuffer](#)
- struct [Material](#)
- class [MaterialBuffer](#)

## 公開メンバ関数

- [GgSimpleShader \(\)](#)
- [GgSimpleShader \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- [GgSimpleShader \(const std::array< std::string, 3 > &files, GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- [GgSimpleShader \(const GgSimpleShader &o\)](#)
- virtual ~[GgSimpleShader \(\)](#)
- [GgSimpleShader & operator= \(const GgSimpleShader &o\)](#)
- bool [load \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- bool [load \(const std::array< std::string, 3 > &files, GLint nvarying=0, const char \\*const \\*varyings=nullptr\)](#)
- virtual void [loadModelviewMatrix \(const GLfloat \\*mv, const GLfloat \\*mn\) const](#)
- virtual void [loadModelviewMatrix \(const GgMatrix &mv, const GgMatrix &mn\) const](#)
- virtual void [loadModelviewMatrix \(const GLfloat \\*mv\) const](#)
- virtual void [loadModelviewMatrix \(const GgMatrix &mv\) const](#)
- virtual void [loadMatrix \(const GLfloat \\*mp, const GLfloat \\*mv, const GLfloat \\*mn\) const](#)
- virtual void [loadMatrix \(const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn\) const](#)
- virtual void [loadMatrix \(const GLfloat \\*mp, const GLfloat \\*mv\) const](#)
- virtual void [loadMatrix \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- void [use \(\) const](#)
- void [use \(const GLfloat \\*mp, const GLfloat \\*mv, const GLfloat \\*mn\) const](#)
- void [use \(const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn\) const](#)
- void [use \(const GLfloat \\*mp, const GLfloat \\*mv\) const](#)
- void [use \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- void [use \(const LightBuffer \\*light, GLint i=0\) const](#)
- void [use \(const LightBuffer &light, GLint i=0\) const](#)
- void [use \(const GLfloat \\*mp, const GLfloat \\*mv, const GLfloat \\*mn, const LightBuffer \\*light, GLint i=0\) const](#)

- void **use** (const **GgMatrix** &mp, const **GgMatrix** &mv, const **GgMatrix** &mn, const **LightBuffer** &light, GLint i=0) const
- void **use** (const GLfloat \*mp, const GLfloat \*mv, const **LightBuffer** \*light, GLint i=0) const
- void **use** (const **GgMatrix** &mp, const **GgMatrix** &mv, const **LightBuffer** &light, GLint i=0) const
- void **use** (const GLfloat \*mp, const **LightBuffer** \*light, GLint i=0) const
- void **use** (const **GgMatrix** &mp, const **LightBuffer** &light, GLint i=0) const

### 8.22.1 詳解

三角形に単純な陰影付けを行うシェーダ.

[gg.h](#) の 7145 行目に定義があります。

### 8.22.2 構築子と解体子

#### 8.22.2.1 GgSimpleShader() [1/4]

```
gg::GgSimpleShader::GgSimpleShader ( ) [inline]
```

コンストラクタ.

[gg.h](#) の 7162 行目に定義があります。

#### 8.22.2.2 GgSimpleShader() [2/4]

```
gg::GgSimpleShader::GgSimpleShader (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

[gg.h](#) の 7179 行目に定義があります。

### 8.22.2.3 GgSimpleShader() [3/4]

```
gg::GgSimpleShader::GgSimpleShader (
    const std::array< std::string, 3 > & files,
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

コンストラクタ。

引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列。
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用)。

gg.h の 7197 行目に定義があります。

### 8.22.2.4 GgSimpleShader() [4/4]

```
gg::GgSimpleShader::GgSimpleShader (
    const GgSimpleShader & o ) [inline]
```

コピー コンストラクタ。

gg.h の 7209 行目に定義があります。

### 8.22.2.5 ~GgSimpleShader()

```
virtual gg::GgSimpleShader::~GgSimpleShader () [inline], [virtual]
```

デストラクタ。

gg.h の 7220 行目に定義があります。

## 8.22.3 関数詳解

### 8.22.3.1 load() [1/2]

```
bool gg::GgSimpleShader::load (
    const std::array< std::string, 3 > & files,
    GLint nvarying = 0,
    const char *const * varyings = nullptr ) [inline]
```

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

## 引数

<i>files</i>	バーテックスシェーダのソースファイル名の配列.
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).

## 戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 7266 行目に定義があります。

## 8.22.3.2 load() [2/2]

```
bool gg::GgSimpleShader::load (
    const std::string & vert,
    const std::string & frag = "",
    const std::string & geom = "",
    GLint nvarying = 0,
    const char *const * varyings = nullptr )
```

シェーダのソースプログラムの文字列からプログラムオブジェクトを作成する.

## 引数

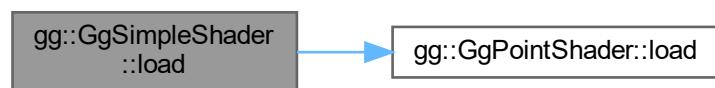
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列.
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).

## 戻り値

プログラムオブジェクトの作成に成功したら true.

gg.cpp の 5952 行目に定義があります。

呼び出し関係図:



### 8.22.3.3 loadMatrix() [1/4]

```
virtual void gg::GgSimpleShader::loadMatrix (
    const GgMatrix & mp,
    const GgMatrix & mv ) const [inline], [virtual]
```

投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列。
<i>mv</i>	GgMatrix 型のモデルビュー変換行列。

gg::GgPointShaderを再実装しています。

gg.h の 7360 行目に定義があります。

呼び出し関係図:



### 8.22.3.4 loadMatrix() [2/4]

```
virtual void gg::GgSimpleShader::loadMatrix (
    const GgMatrix & mp,
    const GgMatrix & mv,
    const GgMatrix & mn ) const [inline], [virtual]
```

投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列。
<i>mv</i>	GgMatrix 型のモデルビュー変換行列。
<i>mn</i>	GgMatrix 型のモデルビュー変換行列の法線変換行列。

gg.h の 7338 行目に定義があります。

呼び出し関係図:



#### 8.22.3.5 loadMatrix() [3/4]

```

virtual void gg::GgSimpleShader::loadMatrix (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline], [virtual]
  
```

投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.

[gg::GgPointShader](#)を再実装しています。

[gg.h](#) の 7349 行目に定義があります。

#### 8.22.3.6 loadMatrix() [4/4]

```

virtual void gg::GgSimpleShader::loadMatrix (
    const GLfloat * mp,
    const GLfloat * mv,
    const GLfloat * mn ) const [inline], [virtual]
  
```

投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.

[gg.h](#) の 7325 行目に定義がります。

### 8.22.3.7 `loadModelviewMatrix()` [1/4]

```
virtual void gg::GgSimpleShader::loadModelviewMatrix (
    const GgMatrix & mv ) const [inline], [virtual]
```

モデルビュー変換行列とそれから求めた法線変換行列を設定する。

引数

<i>mv</i>	<a href="#">GgMatrix</a> 型のモデルビュー変換行列.
-----------	--

[gg::GgPointShader](#)を再実装しています。

[gg.h](#) の 7313 行目に定義がります。

呼び出し関係図:



### 8.22.3.8 `loadModelviewMatrix()` [2/4]

```
virtual void gg::GgSimpleShader::loadModelviewMatrix (
    const GgMatrix & mv,
    const GgMatrix & mn ) const [inline], [virtual]
```

モデルビュー変換行列と法線変換行列を設定する。

引数

<i>mv</i>	<a href="#">GgMatrix</a> 型のモデルビュー変換行列.
<i>mn</i>	<a href="#">GgMatrix</a> 型のモデルビュー変換行列の法線変換行列.

[gg.h](#) の 7293 行目に定義がります。

呼び出し関係図:



#### 8.22.3.9 loadModelviewMatrix() [3/4]

```
virtual void gg::GgSimpleShader::loadModelviewMatrix ( const GLfloat * mv ) const [inline], [virtual]
```

モデルビュー変換行列とそれから求めた法線変換行列を設定する。

引数

<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
-----------	--

gg::GgPointShaderを再実装しています。

gg.h の 7303 行目に定義があります。

#### 8.22.3.10 loadModelviewMatrix() [4/4]

```
virtual void gg::GgSimpleShader::loadModelviewMatrix ( const GLfloat * mv, const GLfloat * mn ) const [inline], [virtual]
```

モデルビュー変換行列と法線変換行列を設定する。

引数

<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.

gg.h の 7281 行目に定義があります。

### 8.22.3.11 operator=( )

```
GgSimpleShader & gg::GgSimpleShader::operator= (
    const GgSimpleShader & o ) [inline]
```

代入演算子.

[gg.h](#) の 7227 行目に定義があります。

### 8.22.3.12 use() [1/13]

```
void gg::GgSimpleShader::use ( ) const [inline], [virtual]
```

シェーダプログラムの使用を開始する.

[gg::GgPointShader](#)を再実装しています。

[gg.h](#) の 7889 行目に定義があります。

### 8.22.3.13 use() [2/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv ) const [inline]
```

投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する.

引数

<i>mp</i>	<a href="#">GgMatrix</a> 型の投影変換行列.
<i>mv</i>	<a href="#">GgMatrix</a> 型のモデルビュー変換行列.

[gg.h](#) の 7940 行目に定義があります。

呼び出し関係図:



## 8.22.3.14 use() [3/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv,
    const GgMatrix & mn ) const [inline]
```

投影変換行列とモデルビュー変換行列と法線変換行列を指定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
<i>mn</i>	GgMatrix 型のモデルビュー変換行列の法線変換行列.

gg.h の 7918 行目に定義があります。

呼び出し関係図:



## 8.22.3.15 use() [4/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv,
    const GgMatrix & mn,
    const LightBuffer & light,
    GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列とモデルビュー変換行列と法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
<i>mn</i>	GgMatrix 型のモデルビュー変換行列の法線変換行列.
<i>light</i>	光源の特性の gg::LightBuffer 構造体.
<i>i</i>	光源データの uniform block のインデックス.

`gg.h` の 8004 行目に定義がります。

呼び出し関係図:



#### 8.22.3.16 `use()` [5/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv,
    const LightBuffer & light,
    GLint i = 0 ) const [inline]
```

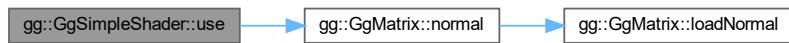
光源を指定し投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	<code>GgMatrix</code> 型の投影変換行列.
<i>mv</i>	<code>GgMatrix</code> 型のモデルビュー変換行列.
<i>light</i>	光源の特性の <code>gg::LightBuffer</code> 構造体.
<i>i</i>	光源データの uniform block のインデックス.

`gg.h` の 8041 行目に定義がります。

呼び出し関係図:



#### 8.22.3.17 `use()` [6/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
```

```
const LightBuffer & light,
GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列。
<i>light</i>	光源の特性の gg::LightBuffer 構造体。
<i>i</i>	光源データの uniform block のインデックス。

gg.h の 8074 行目に定義があります。

呼び出し関係図:



#### 8.22.3.18 use() [7/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline]
```

投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列。
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列。

gg.h の 7929 行目に定義があります。

#### 8.22.3.19 use() [8/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
```

```
const GLfloat * mp,
const GLfloat * mv ) const [inline]
```

投影変換行列とモデルビュー変換行列と法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.

[gg.h](#) の 7902 行目に定義があります。

#### 8.22.3.20 use() [9/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv,
    const GLfloat * mn,
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列とモデルビュー変換行列と法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.
<i>light</i>	光源の特性の <code>gg::LightBuffer</code> 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

[gg.h](#) の 7980 行目に定義があります。

#### 8.22.3.21 use() [10/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv,
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>light</i>	光源の特性の gg::LightBuffer 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 8023 行目に定義がります。

### 8.22.3.22 use() [11/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列を設定してシェーダプログラムの使用を開始する.

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>light</i>	光源の特性の gg::LightBuffer 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 8058 行目に定義がります。

### 8.22.3.23 use() [12/13]

```
void gg::GgSimpleShader::use (
    const LightBuffer & light,
    GLint i = 0 ) const [inline]
```

光源を指定してシェーダプログラムの使用を開始する.

引数

<i>light</i>	光源の特性の gg::LightBuffer 構造体.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 7966 行目に定義がります。

### 8.22.3.24 use() [13/13]

```
void gg::GgSimpleShader::use (
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定してシェーダプログラムの使用を開始する。

引数

<i>light</i>	光源の特性の gg::LightBuffer 構造体のポインタ。
<i>i</i>	光源データの uniform block のインデックス。

gg.h の 7951 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.23 gg::GgTexture クラス

```
#include <gg.h>
```

公開メンバ関数

- [GgTexture \(const GLvoid \\*image, GLsizei width, GLsizei height, GLenum format=GL\\_RGB, GLenum type=GL\\_UNSIGNED\\_BYTE, GLenum internal=GL\\_RGBA, GLenum wrap=GL\\_CLAMP\\_TO\\_EDGE, bool swizzle=true\)](#)
- [virtual ~GgTexture \(\)](#)
- [GgTexture \(const GgTexture &o\)=delete](#)
- [GgTexture & operator= \(const GgTexture &o\)=delete](#)
- [void bind \(\) const](#)
- [void unbind \(\) const](#)
- [void swapRandB \(bool swizzle\) const](#)
- [const GLsizei & getWidth \(\) const](#)
- [const GLsizei & getHeight \(\) const](#)
- [void getSize \(GLsizei \\*size\) const](#)
- [const GLsizei \\* getSize \(\) const](#)
- [const GLuint & getTexture \(\) const](#)

### 8.23.1 詳解

テクスチャ.

覚え書き

画像データを読み込んでテクスチャマップを作成する.

gg.h の 5186 行目に定義があります。

### 8.23.2 構築子と解体子

#### 8.23.2.1 GgTexture() [1/2]

```
gg::GgTexture::GgTexture (
    const GLvoid * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RGB,
    GLenum type = GL_UNSIGNED_BYTE,
    GLenum internal = GL_RGBA,
    GLenum wrap = GL_CLAMP_TO_EDGE,
    bool swizzle = true ) [inline]
```

メモリ上のデータからテクスチャを作成するコンストラクタ.

引数

<i>image</i>	テクスチャとして用いる画像データ, nullptr ならデータを読み込まない.
<i>width</i>	テクスチャの横の画素数.
<i>height</i>	テクスチャの縦の画素数.
<i>format</i>	読み込む画像のフォーマット.
<i>type</i>	画像のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.
<i>swizzle</i>	true ならテクスチャの赤と青を入れ替える, デフォルトは true.

gg.h の 5208 行目に定義があります。

#### 8.23.2.2 ~GgTexture()

```
virtual gg::GgTexture::~GgTexture () [inline], [virtual]
```

デストラクタ.

gg.h の 5226 行目に定義があります。

### 8.23.2.3 GgTexture() [2/2]

```
gg::GgTexture::GgTexture (
    const GgTexture & o ) [delete]
```

コピー構造関数は使用禁止.

## 8.23.3 関数詳解

### 8.23.3.1 bind()

```
void gg::GgTexture::bind ( ) const [inline]
```

テクスチャの使用開始(このテクスチャを使用する際に呼び出す).

gg.h の 5245 行目に定義があります。

### 8.23.3.2 getHeight()

```
const GLsizei & gg::GgTexture::getHeight ( ) const [inline]
```

使用しているテクスチャの縦の画素数を取り出す.

戻り値

テクスチャの縦の画素数.

gg.h の 5280 行目に定義があります。

被呼び出し関係図:



### 8.23.3.3 getSize() [1/2]

```
const GLsizei * gg::GgTexture::getSize ( ) const [inline]
```

使用しているテクスチャのサイズを取り出す。

戻り値

テクスチャのサイズを格納した配列へのポインタ。

gg.h の 5301 行目に定義があります。

### 8.23.3.4 getSize() [2/2]

```
void gg::GgTexture::getSize (
    GLsizei * size ) const [inline]
```

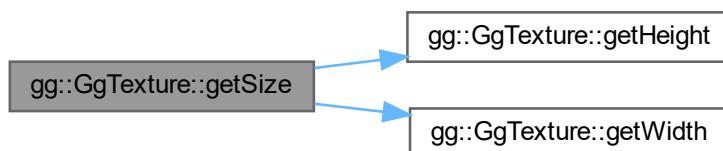
使用しているテクスチャのサイズを取り出す。

引数

<b>size</b>	テクスチャのサイズを格納する GLsizei 型の 2 要素の配列変数。
-------------	--------------------------------------

gg.h の 5290 行目に定義があります。

呼び出し関係図:



### 8.23.3.5 getTexture()

```
const GLuint & gg::GgTexture::getTexture ( ) const [inline]
```

使用しているテクスチャのテクスチャ名を得る。

戻り値

テクスチャ名.

[gg.h](#) の 5311 行目に定義があります。

#### 8.23.3.6 getWidth()

```
const GLsizei & gg::GgTexture::getWidth ( ) const [inline]
```

使用しているテクスチャの横の画素数を取り出す.

戻り値

テクスチャの横の画素数.

[gg.h](#) の 5270 行目に定義があります。

被呼び出し関係図:



#### 8.23.3.7 operator=()

```
GgTexture & gg::GgTexture::operator= (
    const GgTexture & o ) [delete]
```

代入演算子は使用禁止.

#### 8.23.3.8 swapRandB()

```
void gg::GgTexture::swapRandB (
    bool swizzle ) const
```

テクスチャの赤と青を交換する

引数

swizzle	赤と青を交換するなら true
---------	-----------------

gg.cpp の 3971 行目に定義があります。

### 8.23.3.9 unbind()

```
void gg::GgTexture::unbind ( ) const [inline]
```

テクスチャの使用終了(このテクスチャを使用しなくなったら呼び出す)。

gg.h の 5253 行目に定義があります。

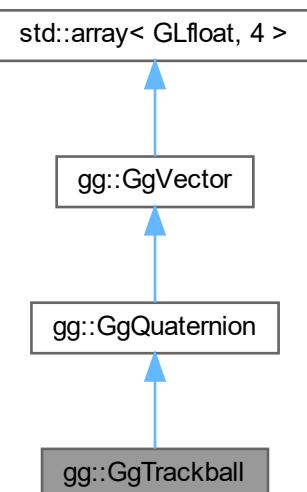
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

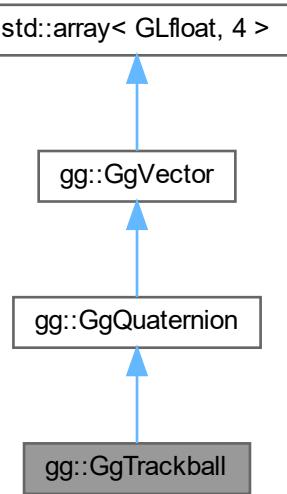
## 8.24 gg::GgTrackball クラス

```
#include <gg.h>
```

gg::GgTrackball の継承関係図



## gg::GgTrackball 連携図



## 公開メンバ関数

- `GgTrackball (const GgQuaternion &q=ggIdentityQuaternion())`
- `virtual ~GgTrackball ()`
- `GgTrackball & operator= (const GgQuaternion &q)`
- `void region (GLfloat w, GLfloat h)`
- `void region (int w, int h)`
- `void begin (GLfloat x, GLfloat y)`
- `void motion (GLfloat x, GLfloat y)`
- `void rotate (const GgQuaternion &q)`
- `void end (GLfloat x, GLfloat y)`
- `void reset (const GgQuaternion &q=ggIdentityQuaternion())`
- `const GLfloat * getStart () const`
- `const GLfloat & getStart (int direction) const`
- `void getStart (GLfloat *position) const`
- `const GLfloat * getScale () const`
- `const GLfloat getScale (int direction) const`
- `void getScale (GLfloat *factor) const`
- `const GgQuaternion & getQuaternion () const`
- `const GgMatrix & getMatrix () const`
- `const GLfloat * get () const`

## 8.24.1 詳解

簡易トラックボール処理。

`gg.h` の 4766 行目に定義があります。

## 8.24.2 構築子と解体子

### 8.24.2.1 GgTrackball()

```
gg::GgTrackball::GgTrackball (
    const GgQuaternion & q = ggIdentityQuaternion() ) [inline]
```

コンストラクタ.

引数

<code>q</code>	トラックボールの回転の初期値の四元数.
----------------	---------------------

`gg.h` の 4781 行目に定義がります。

呼び出し関係図:



### 8.24.2.2 ~GgTrackball()

```
virtual gg::GgTrackball::~GgTrackball () [inline], [virtual]
```

デストラクタ.

`gg.h` の 4789 行目に定義がります。

## 8.24.3 関数詳解

### 8.24.3.1 begin()

```
void gg::GgTrackball::begin (
    GLfloat x,
    GLfloat y )
```

トラックボール処理を開始する.

引数

<i>x</i>	現在のマウスの x 座標.
<i>y</i>	現在のマウスの y 座標.

覚え書き

マウスのドラッグ開始時 (マウスボタンを押したとき) に呼び出す。

[gg.cpp](#) の 3395 行目に定義があります。

#### 8.24.3.2 end()

```
void gg::GgTrackball::end (
    GLfloat x,
    GLfloat y )
```

トラックボール処理を停止する。

引数

<i>x</i>	現在のマウスの x 座標.
<i>y</i>	現在のマウスの y 座標.

覚え書き

マウスのドラッグ終了時 (マウスボタンを離したとき) に呼び出す。

[gg.cpp](#) の 3460 行目に定義があります。

#### 8.24.3.3 get()

```
const GLfloat * gg::GgTrackball::get ( ) const [inline]
```

現在の回転の変換行列を取り出す。

戻り値

回転の変換を表す GLfloat 型の 16 要素の配列.

gg.h の 4961 行目に定義があります。

呼び出し関係図:



#### 8.24.3.4 getMatrix()

```
const GgMatrix & gg::GgTrackball::getMatrix() const [inline]
```

現在の回転の変換行列を取り出す.

戻り値

回転の変換を表す GgMatrix 型の変換行列.

gg.h の 4951 行目に定義があります。

#### 8.24.3.5 getQuaternion()

```
const GgQuaternion & gg::GgTrackball::getQuaternion() const [inline]
```

現在の回転の四元数を取り出す.

戻り値

回転の変換を表す Quaternion 型の四元数.

gg.h の 4941 行目に定義があります。

### 8.24.3.6 `getScale()` [1/3]

```
const GLfloat * gg::GgTrackball::getScale ( ) const [inline]
```

トラックボール処理の換算係数を取り出す。

戻り値

トラックボールの換算係数のポインタ。

[gg.h](#) の 4910 行目に定義があります。

### 8.24.3.7 `getScale()` [2/3]

```
void gg::GgTrackball::getScale (
    GLfloat * factor ) const [inline]
```

トラックボール処理の換算係数を取り出す。

引数

<i>factor</i>	トラックボールの換算係数を格納する 2 要素の配列。
---------------	----------------------------

[gg.h](#) の 4930 行目に定義があります。

### 8.24.3.8 `getScale()` [3/3]

```
const GLfloat gg::GgTrackball::getScale (
    int direction ) const [inline]
```

トラックボール処理の換算係数を取り出す。

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向。
------------------	-----------------------

[gg.h](#) の 4920 行目に定義があります。

### 8.24.3.9 `getStart()` [1/3]

```
const GLfloat * gg::GgTrackball::getStart ( ) const [inline]
```

トラックボール処理の開始位置を取り出す。

戻り値

トラックボールの開始位置のポインタ.

gg.h の 4881 行目に定義があります。

#### 8.24.3.10 getStart() [2/3]

```
void gg::GgTrackball::getStart (
    GLfloat * position ) const [inline]
```

トラックボール処理の開始位置を取り出す.

引数

<i>position</i>	トラックボールの開始位置を格納する 2 要素の配列.
-----------------	----------------------------

gg.h の 4899 行目に定義があります。

#### 8.24.3.11 getStart() [3/3]

```
const GLfloat & gg::GgTrackball::getStart (
    int direction ) const [inline]
```

トラックボール処理の開始位置を取り出す.

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向.
------------------	-----------------------

gg.h の 4889 行目に定義があります。

#### 8.24.3.12 motion()

```
void gg::GgTrackball::motion (
    GLfloat x,
    GLfloat y )
```

回転の変換行列を計算する.

引数

<i>x</i>	現在のマウスの x 座標.
<i>y</i>	現在のマウスの y 座標.

覚え書き

マウスのドラッグ中に呼び出す.

[gg.cpp](#) の 3411 行目に定義があります。

呼び出し関係図:



### 8.24.3.13 operator=( )

```
GgTrackball & gg::GgTrackball::operator= (
    const GgQuaternion & q ) [inline]
```

代入.

引数

<i>q</i>	トラックボールの回転の初期値の四元数.
----------	---------------------

[gg.h](#) の 4798 行目に定義があります。

呼び出し関係図:



## 8.24.3.14 region() [1/2]

```
void gg::GgTrackball::region (
    GLfloat w,
    GLfloat h )
```

トラックボール処理するマウスの移動範囲を指定する。

引数

w	領域の横幅.
h	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す。

gg.cpp の 3382 行目に定義がります。

被呼び出し関係図:



## 8.24.3.15 region() [2/2]

```
void gg::GgTrackball::region (
    int w,
    int h ) [inline]
```

トラックボール処理するマウスの移動範囲を指定する。

引数

w	領域の横幅.
h	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す。

`gg.h` の 4824 行目に定義があります。

呼び出し関係図:



#### 8.24.3.16 reset()

```
void gg::GgTrackball::reset (
    const GgQuaternion & q = ggIdentityQuaternion() )
```

トラックボールをリセットする。

引数

<code>q</code>	トラックボールの回転の初期値の四元数。
----------------	---------------------

`gg.cpp` の 3364 行目に定義があります。

被呼び出し関係図:



#### 8.24.3.17 rotate()

```
void gg::GgTrackball::rotate (
    const GgQuaternion & q )
```

トラックボールの回転角を修正する。

引数

$q$	修正分の回転角の四元数。
-----	--------------

gg.cpp の 3439 行目に定義があります。

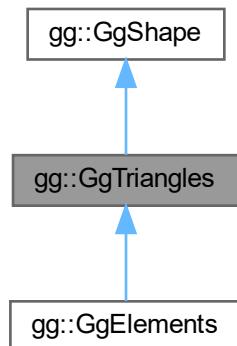
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

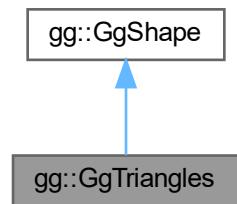
## 8.25 gg::GgTriangles クラス

```
#include <gg.h>
```

gg::GgTriangles の継承関係図



gg::GgTriangles 連携図



## 公開メンバ関数

- `GgTriangles` (GLenum mode=GL\_TRIANGLES)
- `GgTriangles` (const `GgVertex` \*vert, GLsizei count, GLenum mode=GL\_TRIANGLES, GLenum usage=GL\_STATIC\_DRAW)
- virtual ~`GgTriangles` ()
- const GLsizei & `getCount` () const
- const GLuint & `getBuffer` () const
- void `send` (const `GgVertex` \*vert, GLint first=0, GLsizei count=0) const
- void `load` (const `GgVertex` \*vert, GLsizei count, GLenum usage=GL\_STATIC\_DRAW)
- virtual void `draw` (GLint first=0, GLsizei count=0) const

### 8.25.1 詳解

三角形で表した形状データ (Arrays 形式).

`gg.h` の 6442 行目に定義があります。

### 8.25.2 構築子と解体子

#### 8.25.2.1 `GgTriangles()` [1/2]

```
gg::GgTriangles::GgTriangles (
    GLenum mode = GL_TRIANGLES ) [inline]
```

コンストラクタ.

引数

<code>mode</code>	描画する基本図形の種類.
-------------------	--------------

`gg.h` の 6455 行目に定義があります。

#### 8.25.2.2 `GgTriangles()` [2/2]

```
gg::GgTriangles::GgTriangles (
    const GgVertex * vert,
    GLsizei count,
    GLenum mode = GL_TRIANGLES,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

コンストラクタ.

引数

<i>vert</i>	この図形の頂点属性の配列 (nullptr ならデータを転送しない).
<i>count</i>	頂点数.
<i>mode</i>	描画する基本図形の種類.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6468 行目に定義があります。

### 8.25.2.3 ~GgTriangles()

```
virtual gg::GgTriangles::~GgTriangles () [inline], [virtual]
```

デストラクタ.

gg.h の 6482 行目に定義があります。

## 8.25.3 関数詳解

### 8.25.3.1 draw()

```
void gg::GgTriangles::draw (
    GLint first = 0,
    GLsizei count = 0 ) const [virtual]
```

三角形の描画.

引数

<i>first</i>	描画を開始する最初の三角形番号.
<i>count</i>	描画する三角形数, 0 なら全部の三角形を描く.

gg::GgShapeを再実装しています。

gg::GgElementsで再実装されています。

gg.cpp の 5136 行目に定義があります。

呼び出し関係図:



### 8.25.3.2 getBuffer()

```
const GLuint & gg::GgTriangles::getBuffer() const [inline]
```

頂点属性を格納した頂点バッファオブジェクト名を取り出す。

戻り値

この図形の頂点属性を格納した頂点バッファオブジェクト名。

[gg.h](#) の 6501 行目に定義があります。

### 8.25.3.3 getCount()

```
const GLsizei & gg::GgTriangles::getCount() const [inline]
```

データの数を取り出す。

戻り値

この図形の頂点属性の数(頂点数)。

[gg.h](#) の 6491 行目に定義があります。

### 8.25.3.4 load()

```
void gg::GgTriangles::load(
    const GgVertex * vert,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW)
```

バッファオブジェクトを確保して頂点属性を格納する。

引数

<i>vert</i>	頂点属性が格納されている領域の先頭のポインタ.
<i>count</i>	頂点のデータの数(頂点数).
<i>usage</i>	バッファオブジェクトの使い方.

gg.cpp の 5117 行目に定義があります。

### 8.25.3.5 send()

```
void gg::GgTriangles::send (
    const GgVertex * vert,
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

既存のバッファオブジェクトに頂点属性を転送する。

引数

<i>vert</i>	転送元の頂点属性が格納されている領域の先頭のポインタ.
<i>first</i>	転送先のバッファオブジェクトの先頭の要素番号.
<i>count</i>	転送する頂点の位置データの数(0ならバッファオブジェクト全体).

gg.h の 6513 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.26 gg::GgUniformBuffer< T > クラステンプレート

```
#include <gg.h>
```

### 公開メンバ関数

- [GgUniformBuffer \(\)](#)
- [GgUniformBuffer \(const T \\*data, GLsizei count, GLenum usage=GL\\_STATIC\\_DRAW\)](#)
- [GgUniformBuffer \(const T &data, GLsizei count, GLenum usage=GL\\_STATIC\\_DRAW\)](#)
- virtual [~GgUniformBuffer \(\)](#)
- const GLuint & [getTarget \(\) const](#)
- GLsizeiptr [getStride \(\) const](#)
- const GLsizei & [getCount \(\) const](#)
- const GLuint & [getBuffer \(\) const](#)

- void `bind` () const
- void `unbind` () const
- void \* `map` () const
- void \* `map` (GLint first, GLsizei count) const
- void `unmap` () const
- void `load` (const T \*data, GLsizei count, GLenum usage=GL\_STATIC\_DRAW)
- void `load` (const T &data, GLsizei count, GLenum usage=GL\_STATIC\_DRAW)
- void `send` (const GLvoid \*data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
- void `fill` (const GLvoid \*data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
- void `read` (GLvoid \*data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
- void `copy` (GLuint src\_buffer, GLint src\_first=0, GLint dst\_first=0, GLsizei count=0) const

## 8.26.1 詳解

```
template<typename T>
class gg::GgUniformBuffer< T >
```

ユニフォームバッファオブジェクト。

覚え書き

ユニフォーム変数を格納するバッファオブジェクトの基底クラス。

`gg.h` の 5780 行目に定義があります。

## 8.26.2 構築子と解体子

### 8.26.2.1 GgUniformBuffer() [1/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer ( ) [inline]
```

コンストラクタ。

`gg.h` の 5783 行目に定義があります。

### 8.26.2.2 GgUniformBuffer() [2/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer (
    const T * data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトのブロックごとにデータを転送するコンストラクタ。

引数

<i>data</i>	データが格納されている領域の先頭のポインタ ( <code>nullptr</code> ならデータを転送しない).
<i>count</i>	データの数.
<i>usage</i>	バッファオブジェクトの使い方.

[gg.h の 5783 行](#)目に定義があります。

### 8.26.2.3 GgUniformBuffer() [3/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer (
    const T & data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトの全ブロックに同じデータを格納するコンストラクタ.

引数

<i>data</i>	格納するデータ.
<i>count</i>	格納する数.
<i>usage</i>	バッファオブジェクトの使い方.

[gg.h の 5783 行](#)目に定義があります。

### 8.26.2.4 ~GgUniformBuffer()

```
template<typename T >
virtual gg::GgUniformBuffer< T >::~GgUniformBuffer ( ) [inline], [virtual]
```

デストラクタ.

[gg.h の 5783 行](#)目に定義があります。

## 8.26.3 関数詳解

### 8.26.3.1 bind()

```
template<typename T >
void gg::GgUniformBuffer< T >::bind ( ) const [inline]
```

ユニフォームバッファオブジェクトを結合する。

[gg.h](#) の 5868 行目に定義があります。

### 8.26.3.2 copy()

```
template<typename T >
void gg::GgUniformBuffer< T >::copy (
    GLuint src_buffer,
    GLint src_first = 0,
    GLint dst_first = 0,
    GLsizei count = 0 ) const [inline]
```

別のバッファオブジェクトからデータを複写する。

引数

<i>src_buffer</i>	複写元のバッファオブジェクト名。
<i>src_first</i>	複写元 ( <i>buffer</i> ) の先頭のデータの位置。
<i>dst_first</i>	複写先 ( <a href="#">getBuffer()</a> ) の先頭のデータの位置。
<i>count</i>	複写するデータの数 (0 ならバッファオブジェクト全体)。

[gg.h](#) の 6079 行目に定義があります。

### 8.26.3.3 fill()

```
template<typename T >
void gg::GgUniformBuffer< T >::fill (
    const GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトの全ブロックのメンバーを同じデータを格納する。

引数

<i>data</i>	格納するデータ。
<i>offset</i>	格納先のメンバのブロックの先頭からのバイトオフセット。
<i>size</i>	格納するデータの一個あたりのバイト数。
<i>first</i>	格納先のバッファオブジェクトのブロックの先頭の番号。
<i>count</i>	格納するデータの数。

gg.h の 5996 行目に定義があります。

#### 8.26.3.4 getBuffer()

```
template<typename T >
const GLuint & gg::GgUniformBuffer< T >::getBuffer ( ) const [inline]
```

ユニフォームバッファオブジェクト名を取り出す。

戻り値

このユニフォームバッファオブジェクト名。

gg.h の 5860 行目に定義があります。

#### 8.26.3.5 getCount()

```
template<typename T >
const GLsizei & gg::GgUniformBuffer< T >::getCount ( ) const [inline]
```

データの数を取り出す。

戻り値

このユニフォームバッファオブジェクトのデータの数。

gg.h の 5850 行目に定義があります。

#### 8.26.3.6 getStride()

```
template<typename T >
GLsizeiptr gg::GgUniformBuffer< T >::getStride ( ) const [inline]
```

ユニフォームバッファオブジェクトのアライメントを考慮したデータの間隔を取り出す。

戻り値

このユニフォームバッファオブジェクトのデータの間隔。

gg.h の 5840 行目に定義があります。

### 8.26.3.7 `getTarget()`

```
template<typename T >
const GLuint & gg::GgUniformBuffer< T >::getTarget ( ) const [inline]
```

ユニフォームバッファオブジェクトのターゲットを取り出す。

戻り値

このユニフォームバッファオブジェクトのターゲット。

`gg.h` の 5830 行目に定義があります。

### 8.26.3.8 `load()` [1/2]

```
template<typename T >
void gg::GgUniformBuffer< T >::load (
    const T & data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトを確保して全てのブロックに同じデータを格納する。

引数

<code>data</code>	格納するデータ。
<code>count</code>	格納する数。
<code>usage</code>	バッファオブジェクトの使い方。

`gg.h` の 5937 行目に定義があります。

### 8.26.3.9 `load()` [2/2]

```
template<typename T >
void gg::GgUniformBuffer< T >::load (
    const T * data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトを確保してブロックごとにデータを転送する。

引数

<code>data</code>	データが格納されている領域の先頭のポインタ ( <code>nullptr</code> ならデータを転送しない)。
<code>count</code>	データの数。
<code>usage</code>	バッファオブジェクトの使い方。

gg.h の 5918 行目に定義があります。

### 8.26.3.10 map() [1/2]

```
template<typename T >
void * gg::GgUniformBuffer< T >::map ( ) const [inline]
```

ユニフォームバッファオブジェクトをマップする。

戻り値

マップしたメモリの先頭のポインタ。

gg.h の 5886 行目に定義があります。

### 8.26.3.11 map() [2/2]

```
template<typename T >
void * gg::GgUniformBuffer< T >::map (
    GLint first,
    GLsizei count ) const [inline]
```

ユニフォームバッファオブジェクトの指定した範囲をマップする。

引数

<i>first</i>	マップする範囲のバッファオブジェクトの先頭からの位置。
<i>count</i>	マップするデータの数(0ならバッファオブジェクト全体)。

戻り値

マップしたメモリの先頭のポインタ。

gg.h の 5898 行目に定義があります。

### 8.26.3.12 read()

```
template<typename T >
void gg::GgUniformBuffer< T >::read (
    GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトからデータを抽出する。

引数

<i>data</i>	抽出先の領域の先頭のポインタ.
<i>offset</i>	抽出元のユニフォームバッファオブジェクトのメンバのブロックの先頭からのバイトオフセット.
<i>size</i>	抽出するデータの一個あたりのバイト数.
<i>first</i>	抽出元のユニフォームバッファオブジェクトのブロックの先頭の番号.
<i>count</i>	抽出するデータの数 (0ならユニフォームバッファオブジェクト全体).

gg.h の 6031 行目に定義がります。

### 8.26.3.13 send()

```
template<typename T >
void gg::GgUniformBuffer< T >::send (
    const GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトを確保してユニフォームバッファオブジェクトのブロックごとのメンバを同じデータで埋める。

引数

<i>data</i>	データが格納されている領域の先頭のポインタ.
<i>offset</i>	格納先のメンバのブロックの先頭からのバイトオフセット.
<i>size</i>	格納するデータの一個あたりのバイト数.
<i>first</i>	格納先のバッファオブジェクトのブロックの先頭の番号.
<i>count</i>	格納するデータの数.

gg.h の 5958 行目に定義がります。

### 8.26.3.14 unbind()

```
template<typename T >
void gg::GgUniformBuffer< T >::unbind ( ) const [inline]
```

ユニフォームバッファオブジェクトを解放する。

gg.h の 5876 行目に定義がります。

### 8.26.3.15 unmap()

```
template<typename T >
void gg::GgUniformBuffer< T >::unmap ( ) const [inline]
```

バッファオブジェクトをアンマップする。

[gg.h](#) の 5906 行目に定義があります。

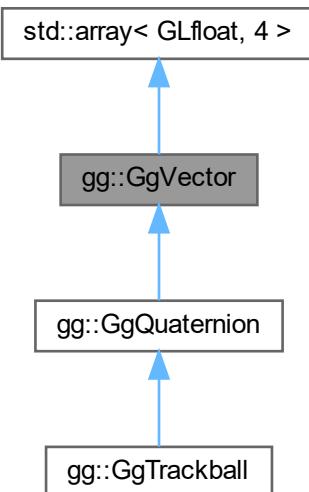
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

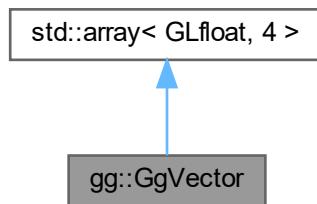
## 8.27 gg::GgVector クラス

#include <gg.h>

gg::GgVector の継承関係図



gg::GgVector 連携図



## 公開メンバ関数

- `GgVector ()`
- `constexpr GgVector (GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3)`
- `constexpr GgVector (GLfloat c)`
- `constexpr GgVector (const GLfloat *a)`
- `GgVector operator+ (const GgVector &v) const`
- `GgVector operator+ (GLfloat c) const`
- `GgVector & operator+= (const GgVector &v)`
- `GgVector & operator+= (GLfloat c)`
- `GgVector operator- (const GgVector &v) const`
- `GgVector operator- (GLfloat c) const`
- `GgVector & operator-= (const GgVector &v)`
- `GgVector & operator-= (GLfloat c)`
- `GgVector operator* (const GgVector &v) const`
- `GgVector operator* (GLfloat c) const`
- `GgVector & operator*= (const GgVector &v)`
- `GgVector & operator*= (GLfloat c)`
- `GgVector operator/ (const GgVector &v) const`
- `GgVector & operator/= (GgVector &v)`
- `GgVector operator/ (GLfloat c) const`
- `GgVector & operator/= (GLfloat c)`
- `GLfloat dot3 (const GgVector &v) const`
- `GLfloat length3 () const`
- `GLfloat distance3 (const GgVector &v) const`
- `GgVector normalize3 () const`
- `GLfloat dot4 (const GgVector &v) const`
- `GLfloat length4 () const`
- `GLfloat distance4 (const GgVector &v) const`
- `GgVector normalize4 () const`

### 8.27.1 詳解

4要素の単精度実数の配列。

`gg.h` の 1571 行目に定義がります。

### 8.27.2 構築子と解体子

#### 8.27.2.1 `GgVector()` [1/4]

`gg::GgVector::GgVector () [inline]`

コンストラクタ。

`gg.h` の 1578 行目に定義がります。

### 8.27.2.2 GgVector() [2/4]

```
constexpr gg::GgVector::GgVector (
    GLfloat v0,
    GLfloat v1,
    GLfloat v2,
    GLfloat v3 ) [inline], [constexpr]
```

コンストラクタ.

引数

<i>v0</i>	GLfloat 型の値.
<i>v1</i>	GLfloat 型の値.
<i>v2</i>	GLfloat 型の値.
<i>v3</i>	GLfloat 型の値.

gg.h の 1590 行目に定義がります。

### 8.27.2.3 GgVector() [3/4]

```
constexpr gg::GgVector::GgVector (
    GLfloat c ) [inline], [constexpr]
```

コンストラクタ.

引数

<i>c</i>	GLfloat 型の値.
----------	--------------

gg.h の 1600 行目に定義がります。

### 8.27.2.4 GgVector() [4/4]

```
constexpr gg::GgVector::GgVector (
    const GLfloat * a ) [inline], [constexpr]
```

コンストラクタ.

引数

<i>a</i>	GLfloat 型の 4 要素の配列変数.
----------	-----------------------

gg.h の 1610 行目に定義がります。

## 8.27.3 関数詳解

### 8.27.3.1 distance3()

```
GLfloat gg::GgVector::distance3 (
    const GgVector & v ) const [inline]
```

GgVector 型の 3 要素の距離.

引数

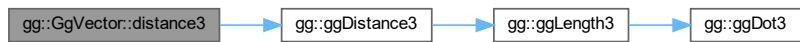
v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v の 3 要素の距離.

gg.h の 1849 行目に定義がります。

呼び出し関係図:



### 8.27.3.2 distance4()

```
GLfloat gg::GgVector::distance4 (
    const GgVector & v ) const [inline]
```

GgVector 型の 4 要素の距離.

引数

v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v の 4 要素の距離.

gg.h の 1893 行目に定義がります。

呼び出し関係図:



### 8.27.3.3 dot3()

```
GLfloat gg::GgVector::dot3 (
    const GgVector & v ) const [inline]
```

**GgVector** 型の 3 要素の内積.

引数

v	<b>GgVector</b> 型のベクトル.
---	-------------------------

戻り値

オブジェクトと v のそれぞれの 3 要素の内積.

**gg.h** の 1828 行目に定義があります。

呼び出し関係図:



### 8.27.3.4 dot4()

```
GLfloat gg::GgVector::dot4 (
    const GgVector & v ) const [inline]
```

**GgVector** 型の 4 要素の内積.

引数

v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v のそれぞれの 4 要素の内積.

gg.h の 1872 行目に定義があります。

呼び出し関係図:



#### 8.27.3.5 length3()

```
GLfloat gg::GgVector::length3 ( ) const [inline]
```

GgVector 型の 3 要素の長さ.

戻り値

オブジェクトの 4 要素の長さ.

gg.h の 1838 行目に定義があります。

呼び出し関係図:



### 8.27.3.6 length4()

```
GLfloat gg::GgVector::length4 ( ) const [inline]
```

**GgVector** 型の 4 要素の長さ.

戻り値

オブジェクトの 4 要素の長さ.

**gg.h** の 1882 行目に定義があります。

呼び出し関係図:



### 8.27.3.7 normalize3()

```
GgVector gg::GgVector::normalize3 ( ) const [inline]
```

**GgVector** 型の 4 要素の正規化.

戻り値

GLfloat 型の 4 要素の配列変数.

**gg.h** の 1859 行目に定義があります。

呼び出し関係図:



### 8.27.3.8 normalize4()

```
GgVector gg::GgVector::normalize4 () const [inline]
```

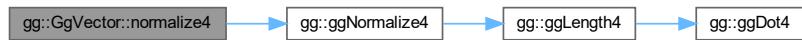
**GgVector** 型の 4 要素の正規化.

戻り値

GLfloat 型の 4 要素の配列変数.

gg.h の 1903 行目に定義があります。

呼び出し関係図:



### 8.27.3.9 operator\*() [1/2]

```
GgVector gg::GgVector::operator* (
    const GgVector & v ) const [inline]
```

**GgVector** 型の積を返す.

引数

v	<b>GgVector</b> 型の変数.
---	-----------------------

戻り値

オブジェクトの各要素と v の各要素の要素ごとの積のオブジェクト.

gg.h の 1725 行目に定義があります。

### 8.27.3.10 operator\*() [2/2]

```
GgVector gg::GgVector::operator* (
    GLfloat c ) const [inline]
```

**GgVector** 型の各要素にスカラーを乗じた積を返す.

引数

<code>c</code>	GLfloat 型の変数.
----------------	---------------

戻り値

オブジェクトの各要素に `c` を乗じたオブジェクト.

`gg.h` の 1736 行目に定義がります。

### 8.27.3.11 operator\*=( ) [1/2]

```
GgVector & gg::GgVector::operator*=
    ( const GgVector & v ) [inline]
```

`GgVector` 型を乗算する.

引数

<code>v</code>	GgVector 型の変数.
----------------	----------------

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ乗算したオブジェクトの参照.

`gg.h` の 1746 行目に定義がります。

### 8.27.3.12 operator\*=( ) [2/2]

```
GgVector & gg::GgVector::operator*=
    ( GLfloat c ) [inline]
```

`GgVector` 型の各要素にスカラーを乗じる.

引数

<code>c</code>	GgVector 型のベクトル.
----------------	------------------

戻り値

オブジェクトの各要素に `c` を乗じたオブジェクトの参照.

`gg.h` の 1761 行目に定義がります。

### 8.27.3.13 operator+() [1/2]

```
GgVector gg::GgVector::operator+ (
    const GgVector & v ) const [inline]
```

GgVector 型の和を返す.

引数

v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトの各要素と v の各要素の要素ごとの和のオブジェクト.

gg.h の 1621 行目に定義があります。

### 8.27.3.14 operator+() [2/2]

```
GgVector gg::GgVector::operator+ (
    GLfloat c ) const [inline]
```

GgVector 型の各要素にスカラーを足した和を返す.

引数

c	GLfloat 型の値.
---	--------------

戻り値

a の各要素に b を足した和のオブジェクト.

gg.h の 1632 行目に定義があります。

### 8.27.3.15 operator+=() [1/2]

```
GgVector & gg::GgVector::operator+= (
    const GgVector & v ) [inline]
```

GgVector 型を加算する.

引数

<code>v</code>	<code>GgVector</code> 型の変数.
----------------	-----------------------------

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ加算したオブジェクトの参照.

`gg.h` の 1643 行目に定義があります。

### 8.27.3.16 operator+=() [2/2]

```
GgVector & gg::GgVector::operator+= (
    GLfloat c ) [inline]
```

`GgVector` 型の各要素にスカラーを加算する.

引数

<code>c</code>	<code>GLfloat</code> 型の値.
----------------	---------------------------

戻り値

オブジェクトの各要素に `c` を足したオブジェクトの参照.

`gg.h` の 1658 行目に定義があります。

### 8.27.3.17 operator-() [1/2]

```
GgVector gg::GgVector::operator- (
    const GgVector & v ) const [inline]
```

`GgVector` 型の差を返す.

引数

<code>v</code>	<code>GgVector</code> 型の変数.
----------------	-----------------------------

戻り値

オブジェクトの各要素と `v` の各要素の要素ごとの差のオブジェクト.

`gg.h` の 1673 行目に定義があります。

### 8.27.3.18 operator-() [2/2]

```
GgVector gg::GgVector::operator- (
    GLfloat c ) const [inline]
```

**GgVector** 型の各要素からスカラーを引いた差を返す.

引数

c	GLfloat 型の変数.
---	---------------

戻り値

オブジェクトの各要素から c を引いたオブジェクト.

gg.h の 1684 行目に定義があります。

### 8.27.3.19 operator-=( ) [1/2]

```
GgVector & gg::GgVector::operator-= (
    const GgVector & v ) [inline]
```

**GgVector** 型を減算する.

引数

v	GgVector 型の変数.
---	----------------

戻り値

オブジェクトの各要素に v の各要素をそれぞれ減算したオブジェクトの参照.

gg.h の 1695 行目に定義があります。

### 8.27.3.20 operator-=( ) [2/2]

```
GgVector & gg::GgVector::operator-= (
    GLfloat c ) [inline]
```

**GgVector** 型の各要素からスカラーを引く.

引数

<code>c</code>	GLfloat 型の変数.
----------------	---------------

戻り値

オブジェクトの各要素から `c` を引いたオブジェクトの参照.

`gg.h` の 1710 行目に定義があります。

### 8.27.3.21 operator/() [1/2]

```
GgVector gg::GgVector::operator/ (
    const GgVector & v ) const [inline]
```

`GgVector` 型の商を返す.

引数

<code>v</code>	GgVector 型の変数.
----------------	----------------

戻り値

オブジェクトの各要素を `v` の各要素で要素ごとに割った結果のオブジェクト.

`gg.h` の 1776 行目に定義があります。

### 8.27.3.22 operator/() [2/2]

```
GgVector gg::GgVector::operator/ (
    GLfloat c ) const [inline]
```

`GgVector` 型の各要素をスカラーで割った商を返す.

引数

<code>c</code>	GLfloat 型の値.
----------------	--------------

戻り値

オブジェクトの各要素を `c` で割った商のオブジェクト.

`gg.h` の 1802 行目に定義があります。

### 8.27.3.23 operator/() [1/2]

```
GgVector & gg::GgVector::operator/= (  
    GgVector & v ) [inline]
```

GgVector 型を除算する.

引数

v	GgVector 型の変数.
---	----------------

戻り値

オブジェクトの各要素に v の各要素をそれぞれ乗算したオブジェクトの参照.

gg.h の 1787 行目に定義があります。

### 8.27.3.24 operator/() [2/2]

```
GgVector & gg::GgVector::operator/= (  
    GLfloat c ) [inline]
```

GgVector 型の各要素をスカラーで割る.

引数

c	GLfloat 型の変数.
---	---------------

戻り値

オブジェクトの各要素を c で割ったオブジェクトの参照.

gg.h の 1813 行目に定義があります。

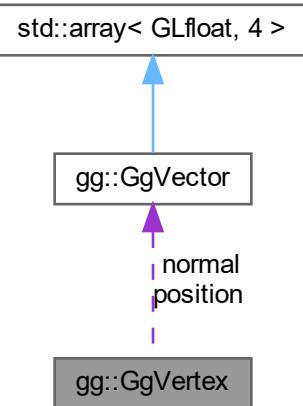
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

## 8.28 gg::GgVertex 構造体

```
#include <gg.h>
```

`gg::GgVertex` 連携図



## 公開 メンバ関数

- `GgVertex ()`
- `GgVertex (const GgVector &pos, const GgVector &norm)`
- `GgVertex (GLfloat px, GLfloat py, GLfloat pz, GLfloat nx, GLfloat ny, GLfloat nz)`
- `GgVertex (const GLfloat *pos, const GLfloat *norm)`

## 公開変数類

- `GgVector position`  
位置.
- `GgVector normal`  
法線.

### 8.28.1 詳解

三角形の頂点データ。

`gg.h` の 6381 行目に定義があります。

### 8.28.2 構築子と解体子

### 8.28.2.1 GgVertex() [1/4]

```
gg::GgVertex::GgVertex () [inline]
```

コンストラクタ.

gg.h の 6392 行目に定義があります。

### 8.28.2.2 GgVertex() [2/4]

```
gg::GgVertex::GgVertex (
    const GgVector & pos,
    const GgVector & norm ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	GgVector 型の位置データ.
<i>norm</i>	GgVector 型の法線データ.

gg.h の 6402 行目に定義があります。

### 8.28.2.3 GgVertex() [3/4]

```
gg::GgVertex::GgVertex (
    GLfloat px,
    GLfloat py,
    GLfloat pz,
    GLfloat nx,
    GLfloat ny,
    GLfloat nz ) [inline]
```

コンストラクタ.

引数

<i>px</i>	GgVector 型の位置データの x 成分.
<i>py</i>	GgVector 型の位置データの y 成分.
<i>pz</i>	GgVector 型の位置データの z 成分.
<i>nx</i>	GgVector 型の法線データの x 成分.
<i>ny</i>	GgVector 型の法線データの y 成分.
<i>nz</i>	GgVector 型の法線データの z 成分.

[gg.h](#) の 6418 行目に定義がります。

#### 8.28.2.4 GgVertex() [4/4]

```
gg::GgVertex::GgVertex (
    const GLfloat * pos,
    const GLfloat * norm ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	3 要素の <code>GLfloat</code> 型の位置データのポインタ.
<i>norm</i>	3 要素の <code>GLfloat</code> 型の法線データのポインタ.

[gg.h](#) の 6433 行目に定義がります。

### 8.28.3 メンバ詳解

#### 8.28.3.1 normal

`GgVector gg::GgVertex::normal`

法線.

[gg.h](#) の 6387 行目に定義がります。

#### 8.28.3.2 position

`GgVector gg::GgVertex::position`

位置.

[gg.h](#) の 6384 行目に定義がります。

この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

## 8.29 gg::GgVertexArray クラス

```
#include <gg.h>
```

### 公開メンバ関数

- `GgVertexArray (GLenum mode=0)`
- `GgVertexArray (const GgVertexArray &o)=delete`
- `virtual ~GgVertexArray ()`
- `GgVertexArray & operator= (const GgVertexArray &o)=delete`
- `const GLuint & get () const`
- `void bind () const`

### 8.29.1 詳解

頂点配列クラス。

gg.h の 6110 行目に定義があります。

### 8.29.2 構築子と解体子

#### 8.29.2.1 GgVertexArray() [1/2]

```
gg::GgVertexArray::GgVertexArray (
    GLenum mode = 0 ) [inline]
```

コンストラクタ。

引数

<code>mode</code>	基本図形の種類。
-------------------	----------

gg.h の 6122 行目に定義があります。

#### 8.29.2.2 GgVertexArray() [2/2]

```
gg::GgVertexArray::GgVertexArray (
    const GgVertexArray & o ) [delete]
```

コピーコンストラクタは使用禁止。

### 8.29.2.3 ~GgVertexArray()

```
virtual gg::GgVertexArray::~GgVertexArray() [inline], [virtual]
```

デストラクタ。

[gg.h](#) の 6136 行目に定義がります。

## 8.29.3 関数詳解

### 8.29.3.1 bind()

```
void gg::GgVertexArray::bind() const [inline]
```

頂点配列オブジェクトを結合する。

[gg.h](#) の 6160 行目に定義がります。

### 8.29.3.2 get()

```
const GLuint & gg::GgVertexArray::get() const [inline]
```

頂点配列オブジェクト名を取り出す。

戻り値

頂点配列オブジェクト名。

[gg.h](#) の 6152 行目に定義がります。

### 8.29.3.3 operator=()

```
GgVertexArray & gg::GgVertexArray::operator= (
    const GgVertexArray & o) [delete]
```

代入演算子は使用禁止。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

## 8.30 Intrinsic 構造体

```
#include <Intrinsics.h>
```

### 公開メンバ関数

- `Intrinsic()`
- `Intrinsic(const std::array<float, 2> &fov, const std::array<float, 2> &center, const std::array<int, 2> size, double fps)`
- `Intrinsic(const picojson::object &object)`
- `void setFov(float focal)`
- `void setFov(float fovx, float fovy)`
- `void setCenter(float x, float y)`
- `void setSize(int width, int height)`
- `void setFps(double frequency)`

### 公開変数類

- `std::array<float, 2> fov`  
キャプチャデバイスのレンズの縦横の画角
- `std::array<float, 2> center`  
キャプチャデバイスのレンズの中心(主点)の位置
- `std::array<int, 2> size`  
キャプチャデバイスの解像度
- `double fps`  
キャプチャデバイスのフレームレート

### 静的公開変数類

- `static constexpr auto sensorSize { 35.0f }`  
展開時のセンサーサイズ

#### 8.30.1 詳解

キャプチャデバイス固有のパラメータ

`Intrinsics.h` の 20 行目に定義があります。

#### 8.30.2 構築子と解体子

### 8.30.2.1 `Intrinsics()` [1/3]

```
Intrinsics::Intrinsics () [inline]
```

キャプチャデバイス固有のパラメータの構造体のデフォルトコンストラクタ

覚え書き

構成ファイルが読みなったときにしか使わない

[Intrinsics.h](#) の 39 行目に定義があります。

### 8.30.2.2 `Intrinsics()` [2/3]

```
Intrinsics::Intrinsics (
    const std::array< float, 2 > & fov,
    const std::array< float, 2 > & center,
    const std::array< int, 2 > size,
    double fps ) [inline]
```

パラメータを指定するときに使う キャプチャデバイス固有のパラメータの構造体のコンストラクタ

引数

<code>fov</code>	キャプチャデバイスのレンズの縦横の画角
<code>center</code>	キャプチャデバイスのレンズの中心(主点)の位置
<code>resolution</code>	キャプチャデバイスの解像度
<code>fps</code>	キャプチャデバイスのフレームレート

[Intrinsics.h](#) の 53 行目に定義があります。

### 8.30.2.3 `Intrinsics()` [3/3]

```
Intrinsics::Intrinsics (
    const picojson::object & object )
```

構成ファイルを読み込むときに使う キャプチャデバイス固有のパラメータの構造体のコンストラクタ

[Intrinsics.cpp](#) の 14 行目に定義があります。

呼び出し関係図:



### 8.30.3 関数詳解

#### 8.30.3.1 setCenter()

```
void Intrinsic::setCenter (
    float x,
    float y ) [inline]
```

キャプチャデバイスのレンズの中心の位置を変更する

引数

x	キャプチャデバイスのレンズの中心の位置の x 座標
y	キャプチャデバイスのレンズの中心の位置の y 座標

Intrinsic.h の 97 行目に定義があります。

被呼び出し関係図:



#### 8.30.3.2 setFov() [1/2]

```
void Intrinsic::setFov (
    float focal )
```

スクリーンの高さをもとにキャプチャデバイスのレンズの縦横の画角を変更する

引数

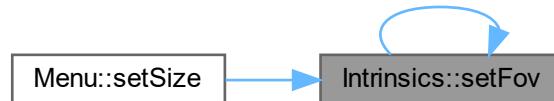
<i>focal</i>	撮像面の対角線長が <code>sensorSize</code> のときの焦点距離
--------------	--

[Intrinsics.cpp](#) の 32 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.30.3.3 `setFov()` [2/2]

```
void Intrinsics::setFov (
    float fovx,
    float fovy ) [inline]
```

キャプチャデバイスのレンズの縦横の画角を変更する

引数

<i>fovx</i>	キャプチャデバイスのレンズの <code>fovx</code> 方向の画角
<i>fovy</i>	キャプチャデバイスのレンズの <code>fovy</code> 方向の画角

[Intrinsics.h](#) の 84 行目に定義があります。

### 8.30.3.4 setFps()

```
void Intrinsics::setFps (
    double frequency ) [inline]
```

キャプチャデバイスのフレームレートを変更する

引数

<i>frequency</i>	フレームレート
------------------	---------

[Intrinsics.h](#) の 124 行目に定義があります。

### 8.30.3.5 setSize()

```
void Intrinsics::setSize (
    int width,
    int height ) [inline]
```

キャプチャデバイスの解像度を変更する

引数

<i>width</i>	キャプチャデバイスのフレームの横の画素数
<i>height</i>	キャプチャデバイスのフレームの縦の画素数

[Intrinsics.h](#) の 111 行目に定義があります。

## 8.30.4 メンバ詳解

### 8.30.4.1 center

```
std::array<float, 2> Intrinsics::center
```

キャプチャデバイスのレンズの中心(主点)の位置

[Intrinsics.h](#) の 26 行目に定義があります。

#### 8.30.4.2 fov

```
std::array<float, 2> Intrinsics::fov
```

キャプチャデバイスのレンズの縦横の画角

[Intrinsics.h](#) の 23 行目に定義があります。

#### 8.30.4.3 fps

```
double Intrinsics::fps
```

キャプチャデバイスのフレームレート

[Intrinsics.h](#) の 32 行目に定義があります。

#### 8.30.4.4 sensorSize

```
constexpr auto Intrinsics::sensorSize { 35.0f } [static], [constexpr]
```

展開時のセンサーサイズ

[Intrinsics.h](#) の 69 行目に定義があります。

#### 8.30.4.5 size

```
std::array<int, 2> Intrinsics::size
```

キャプチャデバイスの解像度

[Intrinsics.h](#) の 29 行目に定義があります。

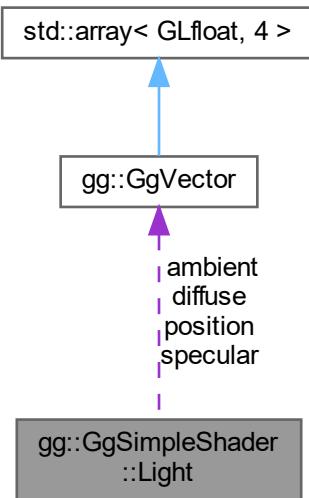
この構造体詳解は次のファイルから抽出されました:

- [Intrinsics.h](#)
- [Intrinsics.cpp](#)

## 8.31 gg::GgSimpleShader::Light 構造体

```
#include <gg.h>
```

gg::GgSimpleShader::Light 連携図



### 公開変数類

- [GgVector ambient](#)  
光源強度の環境光成分.
- [GgVector diffuse](#)  
光源強度の拡散反射光成分.
- [GgVector specular](#)  
光源強度の鏡面反射光成分.
- [GgVector position](#)  
光源の位置.

#### 8.31.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する光源データ.

gg.h の 7368 行目に定義があります。

#### 8.31.2 メンバ詳解

### 8.31.2.1 ambient

`GgVector gg::GgSimpleShader::Light::ambient`

光源強度の環境光成分.

`gg.h` の 7370 行目に定義があります。

### 8.31.2.2 diffuse

`GgVector gg::GgSimpleShader::Light::diffuse`

光源強度の拡散反射光成分.

`gg.h` の 7371 行目に定義があります。

### 8.31.2.3 position

`GgVector gg::GgSimpleShader::Light::position`

光源の位置.

`gg.h` の 7373 行目に定義があります。

### 8.31.2.4 specular

`GgVector gg::GgSimpleShader::Light::specular`

光源強度の鏡面反射光成分.

`gg.h` の 7372 行目に定義があります。

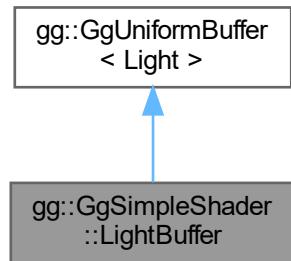
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

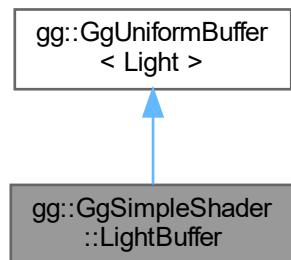
## 8.32 gg::GgSimpleShader::LightBuffer クラス

```
#include <gg.h>
```

gg::GgSimpleShader::LightBuffer の継承関係図



gg::GgSimpleShader::LightBuffer 連携図



### 公開 メンバ関数

- `LightBuffer (const Light *light=nullptr, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `LightBuffer (const Light &light, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~LightBuffer ()`
- `void loadAmbient (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GgVector &ambient, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GLfloat *ambient, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GgVector &diffuse, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GLfloat *diffuse, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (const GgVector &specular, GLint first=0, GLsizei count=1) const`

- void **loadSpecular** (const GLfloat \*specular, GLint first=0, GLsizei count=1) const
- void **loadColor** (const **Light** &color, GLint first=0, GLsizei count=1) const
- void **loadPosition** (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const **GgVector** &position, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const GLfloat \*position, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const **GgVector** \*position, GLint first=0, GLsizei count=1) const
- void **load** (const **Light** \*light, GLint first=0, GLsizei count=1) const
- void **load** (const **Light** &light, GLint first=0, GLsizei count=1) const
- void **select** (GLint i=0) const

### 8.32.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する光源データのユニフォームバッファオブジェクト。

[gg.h](#) の 7379 行目に定義があります。

### 8.32.2 構築子と解体子

#### 8.32.2.1 LightBuffer() [1/2]

```
gg::GgSimpleShader::LightBuffer::LightBuffer (
    const Light * light = nullptr,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

デフォルトコンストラクタ。

引数

<i>light</i>	<a href="#">GgSimpleShader::Light</a> 型の光源データのポインタ。
<i>count</i>	バッファ中の <a href="#">GgSimpleShader::Light</a> 型の光源データの数。
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される。

[gg.h](#) の 7391 行目に定義があります。

#### 8.32.2.2 LightBuffer() [2/2]

```
gg::GgSimpleShader::LightBuffer::LightBuffer (
    const Light & light,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じデータで埋めるコンストラクタ。

引数

<i>light</i>	GgSimpleShader::Light 型の光源データ.
<i>count</i>	バッファ中の GgSimpleShader::Light 型の光源データの数.
<i>usage</i>	バッファの使い方のパターン, glBufferData() の第 4 引数の usage に渡される.

gg.h の 7407 行目に定義があります。

### 8.32.2.3 ~LightBuffer()

```
virtual gg::GgSimpleShader::LightBuffer::~LightBuffer () [inline], [virtual]
```

デストラクタ.

gg.h の 7419 行目に定義があります。

## 8.32.3 関数詳解

### 8.32.3.1 load() [1/2]

```
void gg::GgSimpleShader::LightBuffer::load (
    const Light & light,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の色と位置を設定する.

引数

<i>light</i>	光源の特性の GgSimpleShader::Light 構造体.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7611 行目に定義があります。

### 8.32.3.2 load() [2/2]

```
void gg::GgSimpleShader::LightBuffer::load (
    const Light * light,
```

```
GLint first = 0,
GLsizei count = 1 ) const [inline]
```

光源の色と位置を設定する。

引数

<i>light</i>	光源の特性の <a href="#">GgSimpleShader::Light</a> 構造体のポインタ.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.h](#) の 7599 行目に定義があります。

### 8.32.3.3 [loadAmbient\(\)](#) [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    const GgVector & ambient,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の環境光成分を設定する。

引数

<i>ambient</i>	光源の強度の環境光成分を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5462 行目に定義があります。

### 8.32.3.4 [loadAmbient\(\)](#) [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    const GLfloat * ambient,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の環境光成分を設定する。

引数

<i>ambient</i>	光源の強度の環境光成分を格納した <a href="#">GLfloat</a> 型の 4 要素の配列変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7454 行目に定義がります。

### 8.32.3.5 loadAmbient() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の環境光成分を設定する。

引数

<i>r</i>	光源の強度の環境光成分の赤成分.
<i>g</i>	光源の強度の環境光成分の緑成分.
<i>b</i>	光源の強度の環境光成分の青成分.
<i>a</i>	光源の強度の拡散反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5436 行目に定義がります。

### 8.32.3.6 loadColor()

```
void gg::GgSimpleShader::LightBuffer::loadColor (
    const Light & color,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の色を設定するが位置は変更しない。

引数

<i>color</i>	光源の特性の GgSimpleShader::Light 構造体.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5589 行目に定義がります。

### 8.32.3.7 loadDiffuse() [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    const GgVector & diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の拡散反射光成分を設定する。

引数

<i>diffuse</i>	光源の強度の拡散反射光成分を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5514 行目に定義があります。

### 8.32.3.8 loadDiffuse() [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    const GLfloat * diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の拡散反射光成分を設定する。

引数

<i>diffuse</i>	光源の強度の拡散反射光成分を格納した <a href="#">GLfloat</a> 型の 4 要素の配列変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.h](#) の 7491 行目に定義があります。

### 8.32.3.9 loadDiffuse() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の拡散反射光成分を設定する。

## 引数

<i>r</i>	光源の強度の拡散反射光成分の赤成分.
<i>g</i>	光源の強度の拡散反射光成分の緑成分.
<i>b</i>	光源の強度の拡散反射光成分の青成分.
<i>a</i>	光源の強度の拡散反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5488 行目に定義があります。

## 8.32.3.10 loadPosition() [1/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GgVector & position,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の位置を設定する.

## 引数

<i>position</i>	光源の位置の同次座標を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5643 行目に定義があります。

## 8.32.3.11 loadPosition() [2/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GgVector * position,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の位置を設定する.

## 引数

<i>position</i>	光源の位置の同次座標を格納した <a href="#">GgVector</a> 型の配列.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.h](#) の 7587 行目に定義がります。

### 8.32.3.12 loadPosition() [3/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GLfloat * position,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の位置を設定する。

引数

<i>position</i>	光源の位置の同次座標を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.h](#) の 7574 行目に定義がります。

### 8.32.3.13 loadPosition() [4/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の位置を設定する。

引数

<i>x</i>	光源の位置の x 座標。
<i>y</i>	光源の位置の y 座標。
<i>z</i>	光源の位置の z 座標。
<i>w</i>	光源の位置の w 座標, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5617 行目に定義がります。

## 8.32.3.14 loadSpecular() [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    const GgVector & specular,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>specular</i>	光源の強度の鏡面反射光成分を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5566 行目に定義があります。

## 8.32.3.15 loadSpecular() [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    const GLfloat * specular,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>specular</i>	光源の強度の鏡面反射光成分を格納した <a href="#">GLfloat</a> 型の 4 要素の配列変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.h](#) の 7528 行目に定義があります。

## 8.32.3.16 loadSpecular() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>r</i>	光源の強度の鏡面反射光成分の赤成分.
<i>g</i>	光源の強度の鏡面反射光成分の緑成分.
<i>b</i>	光源の強度の鏡面反射光成分の青成分.
<i>a</i>	光源の強度の鏡面反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5540 行目に定義があります。

### 8.32.3.17 select()

```
void gg::GgSimpleShader::LightBuffer::select (
    GLint i = 0 ) const [inline]
```

光源を選択する.

引数

<i>i</i>	光源データの uniform block のインデックス.
----------	-------------------------------

gg.h の 7621 行目に定義があります。

被呼び出し関係図:



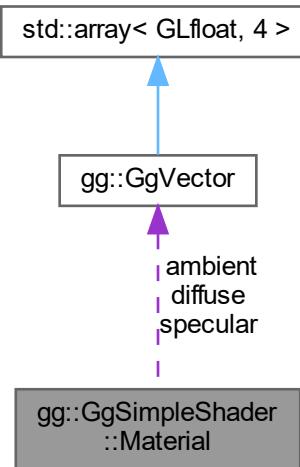
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.33 gg::GgSimpleShader::Material 構造体

```
#include <gg.h>
```

gg::GgSimpleShader::Material 連携図



## 公開変数類

- **GgVector ambient**  
環境光に対する反射係数.
- **GgVector diffuse**  
拡散反射係数.
- **GgVector specular**  
鏡面反射係数.
- **GLfloat shininess**  
輝き係数.

### 8.33.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する材質データ.

[gg.h](#) の 7632 行目に定義があります。

### 8.33.2 メンバ詳解

#### 8.33.2.1 ambient

[GgVector gg::GgSimpleShader::Material::ambient](#)

環境光に対する反射係数.

[gg.h](#) の 7634 行目に定義があります。

### 8.33.2.2 diffuse

`GgVector gg::GgSimpleShader::Material::diffuse`

拡散反射係数.

`gg.h` の 7635 行目に定義があります。

### 8.33.2.3 shininess

`GLfloat gg::GgSimpleShader::Material::shininess`

輝き係数.

`gg.h` の 7637 行目に定義があります。

### 8.33.2.4 specular

`GgVector gg::GgSimpleShader::Material::specular`

鏡面反射係数.

`gg.h` の 7636 行目に定義があります。

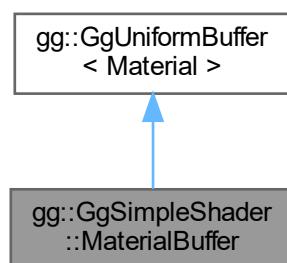
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

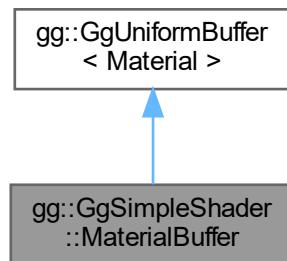
## 8.34 gg::GgSimpleShader::MaterialBuffer クラス

#include <gg.h>

gg::GgSimpleShader::MaterialBuffer の継承関係図



gg::GgSimpleShader::MaterialBuffer 連携図



## 公開 メンバ関数

- `MaterialBuffer (const Material *material=nullptr, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `MaterialBuffer (const Material &material, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~MaterialBuffer ()`
- `void loadAmbient (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GgVector &ambient, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GLfloat *ambient, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GgVector &diffuse, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GLfloat *diffuse, GLint first=0, GLsizei count=1) const`
- `void loadAmbientAndDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbientAndDiffuse (const GgVector &color, GLint first=0, GLsizei count=1) const`
- `void loadAmbientAndDiffuse (const GLfloat *color, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (const GgVector &specular, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (const GLfloat *specular, GLint first=0, GLsizei count=1) const`
- `void loadShininess (GLfloat shininess, GLint first=0, GLsizei count=1) const`
- `void loadShininess (const GLfloat *shininess, GLint first=0, GLsizei count=1) const`
- `void load (const Material *material, GLint first=0, GLsizei count=1) const`
- `void load (const Material &material, GLint first=0, GLsizei count=1) const`
- `void select (GLint i=0) const`

### 8.34.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する材質データのユニフォームバッファオブジェクト。

[gg.h](#) の 7643 行目に定義があります。

### 8.34.2 構築子と解体子

### 8.34.2.1 MaterialBuffer() [1/2]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material * material = nullptr,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

デフォルトコンストラクタ.

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データのポインタ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

`gg.h` の 7655 行目に定義がります。

### 8.34.2.2 MaterialBuffer() [2/2]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material & material,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じデータで埋めるコンストラクタ.

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

`gg.h` の 7671 行目に定義がります。

### 8.34.2.3 ~MaterialBuffer()

```
virtual gg::GgSimpleShader::MaterialBuffer::~MaterialBuffer () [inline], [virtual]
```

デストラクタ.

`gg.h` の 7683 行目に定義がります。

## 8.34.3 関数詳解

**8.34.3.1 load() [1/2]**

```
void gg::GgSimpleShader::MaterialBuffer::load (
    const Material & material,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

材質を設定する。

引数

<i>material</i>	光源の特性の GgSimpleShader::Material 構造体.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7868 行目に定義があります。

**8.34.3.2 load() [2/2]**

```
void gg::GgSimpleShader::MaterialBuffer::load (
    const Material * material,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

材質を設定する。

引数

<i>material</i>	光源の特性の GgSimpleShader::Material 構造体のポインタ.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7856 行目に定義があります。

**8.34.3.3 loadAmbient() [1/3]**

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbient (
    const GgVector & ambient,
    GLint first = 0,
    GLsizei count = 1 ) const
```

三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の環境光成分を設定する。

引数

<i>ambient</i>	光源の強度の環境光成分を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5695 行目に定義がります。

#### 8.34.3.4 `loadAmbient()` [2/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbient (
    const GLfloat * ambient,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

環境光に対する反射係数を設定する.

引数

<i>ambient</i>	環境光に対する反射係数を格納した <a href="#">GLfloat</a> 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

[gg.h](#) の 7718 行目に定義がります。

#### 8.34.3.5 `loadAmbient()` [3/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbient (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

環境光に対する反射係数を設定する.

引数

<i>r</i>	環境光に対する反射係数の赤成分.
<i>g</i>	環境光に対する反射係数の緑成分.
<i>b</i>	環境光に対する反射係数の青成分.
<i>a</i>	環境光に対する反射係数の不透明度, デフォルトは 1.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5669 行目に定義がります。

#### 8.34.3.6 loadAmbientAndDiffuse() [1/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse (
    const GgVector & color,
    GLint first = 0,
    GLsizei count = 1 ) const
```

三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する。

引数

<i>color</i>	環境光に対する反射係数と拡散反射係数を格納した <a href="#">GgVector</a> 型の変数。
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5799 行目に定義がります。

#### 8.34.3.7 loadAmbientAndDiffuse() [2/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse (
    const GLfloat * color,
    GLint first = 0,
    GLsizei count = 1 ) const
```

環境光に対する反射係数と拡散反射係数を設定する。

引数

<i>color</i>	環境光に対する反射係数と拡散反射係数を格納した <a href="#">GLfloat</a> 型の 4 要素の配列変数。
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5822 行目に定義がります。

#### 8.34.3.8 loadAmbientAndDiffuse() [3/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse (
    GLfloat r,
```

```
GLfloat g,
GLfloat b,
GLfloat a = 1.0f,
GLint first = 0,
GLsizei count = 1 ) const
```

環境光に対する反射係数と拡散反射係数を設定する。

引数

<i>r</i>	環境光に対する反射係数と拡散反射係数の赤成分.
<i>g</i>	環境光に対する反射係数と拡散反射係数の緑成分.
<i>b</i>	環境光に対する反射係数と拡散反射係数の青成分.
<i>a</i>	環境光に対する反射係数と拡散反射係数の不透明度, デフォルトは 1.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

[gg.cpp](#) の 5773 行目に定義がります。

### 8.34.3.9 loadDiffuse() [1/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadDiffuse (
    const GgVector & diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const
```

三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の拡散反射光成分を設定する。

引数

<i>ambient</i>	光源の強度の拡散反射光成分を格納した <a href="#">GgVector</a> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5747 行目に定義がります。

### 8.34.3.10 loadDiffuse() [2/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadDiffuse (
    const GLfloat * diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

拡散反射係数を設定する。

引数

<i>diffuse</i>	拡散反射係数を格納した GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7755 行目に定義があります。

#### 8.34.3.11 loadDiffuse() [3/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadDiffuse (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

拡散反射係数を設定する.

引数

<i>r</i>	拡散反射係数の赤成分.
<i>g</i>	拡散反射係数の緑成分.
<i>b</i>	拡散反射係数の青成分.
<i>a</i>	拡散反射係数の不透明度, デフォルトは 1.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5721 行目に定義があります。

#### 8.34.3.12 loadShininess() [1/2]

```
void gg::GgSimpleShader::MaterialBuffer::loadShininess (
    const GLfloat * shininess,
    GLint first = 0,
    GLsizei count = 1 ) const
```

輝き係数を設定する.

引数

<i>shininess</i>	輝き係数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

[gg.cpp](#) の 5935 行目に定義がります。

### 8.34.3.13 loadShininess() [2/2]

```
void gg::GgSimpleShader::MaterialBuffer::loadShininess (
    GLfloat shininess,
    GLint first = 0,
    GLsizei count = 1 ) const
```

輝き係数を設定する。

引数

<i>shininess</i>	輝き係数。
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

[gg.cpp](#) の 5914 行目に定義がります。

### 8.34.3.14 loadSpecular() [1/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadSpecular (
    const GgVector & specular,
    GLint first = 0,
    GLsizei count = 1 ) const
```

三角形に単純な陰影付けを行うシェーダが参照する材質データ：鏡面反射係数を設定する。

引数

<i>ambient</i>	鏡面反射係数を格納した <a href="#">GgVector</a> 型の変数。
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5891 行目に定義がります。

### 8.34.3.15 loadSpecular() [2/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadSpecular (
    const GLfloat * specular,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

鏡面反射係数を設定する。

引数

<i>specular</i>	鏡面反射係数を格納した GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7825 行目に定義があります。

#### 8.34.3.16 loadSpecular() [3/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadSpecular (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

鏡面反射係数を設定する.

引数

<i>r</i>	鏡面反射係数の赤成分.
<i>g</i>	鏡面反射係数の緑成分.
<i>b</i>	鏡面反射係数の青成分.
<i>a</i>	鏡面反射係数の不透明度, デフォルトは 1.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5865 行目に定義があります。

#### 8.34.3.17 select()

```
void gg::GgSimpleShader::MaterialBuffer::select (
    GLint i = 0 ) const [inline]
```

材質を選択する.

引数

<i>i</i>	材質データの uniform block のインデックス.
----------	-------------------------------

gg.h の 7878 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

## 8.35 Menu クラス

```
#include <Menu.h>
```

### 公開メンバ関数

- `Menu (const Config &config, Capture &capture, Calibration &calibration)`
- `Menu (const Menu &menu)=delete`
- `virtual ~Menu ()`
- `Menu & operator= (const Menu &menu)=delete`
- `operator bool () const`
- `const auto & getPose () const`
- `auto getMenubarHeight () const`
- `void setSize (const std::array< int, 2 > &size)`
- `std::array< GLsizei, 2 > setup (GLfloat aspect) const`
- `void draw ()`
- `void saveImage (const cv::Mat &image, const std::string &filename="*.jpg")`

### 公開変数類

- `bool detectMarker`  
*ArUco Marker を検出するなら true*
- `bool detectBoard`  
*ChArUco Board を検出するなら true*

#### 8.35.1 詳解

##### メニューの描画

[Menu.h](#) の 23 行目に定義がります。

#### 8.35.2 構築子と解体子

##### 8.35.2.1 Menu() [1/2]

```
Menu::Menu (
    const Config & config,
    Capture & capture,
    Calibration & calibration )
```

コンストラクタ

引数

<i>config</i>	構成データ
<i>capture</i>	入力フレームを取得するキャプチャデバイス
<i>calibration</i>	較正オブジェクト

[Menu.cpp](#) の 314 行目に定義がります。

### 8.35.2.2 Menu() [2/2]

```
Menu::Menu (   
    const Menu & menu ) [delete]
```

コピー構造子は使用しない

引数

<i>menu</i>	コピー元
-------------	------

### 8.35.2.3 ~Menu()

```
Menu::~Menu ( ) [virtual]
```

デストラクタ

[Menu.cpp](#) の 355 行目に定義がります。

## 8.35.3 関数詳解

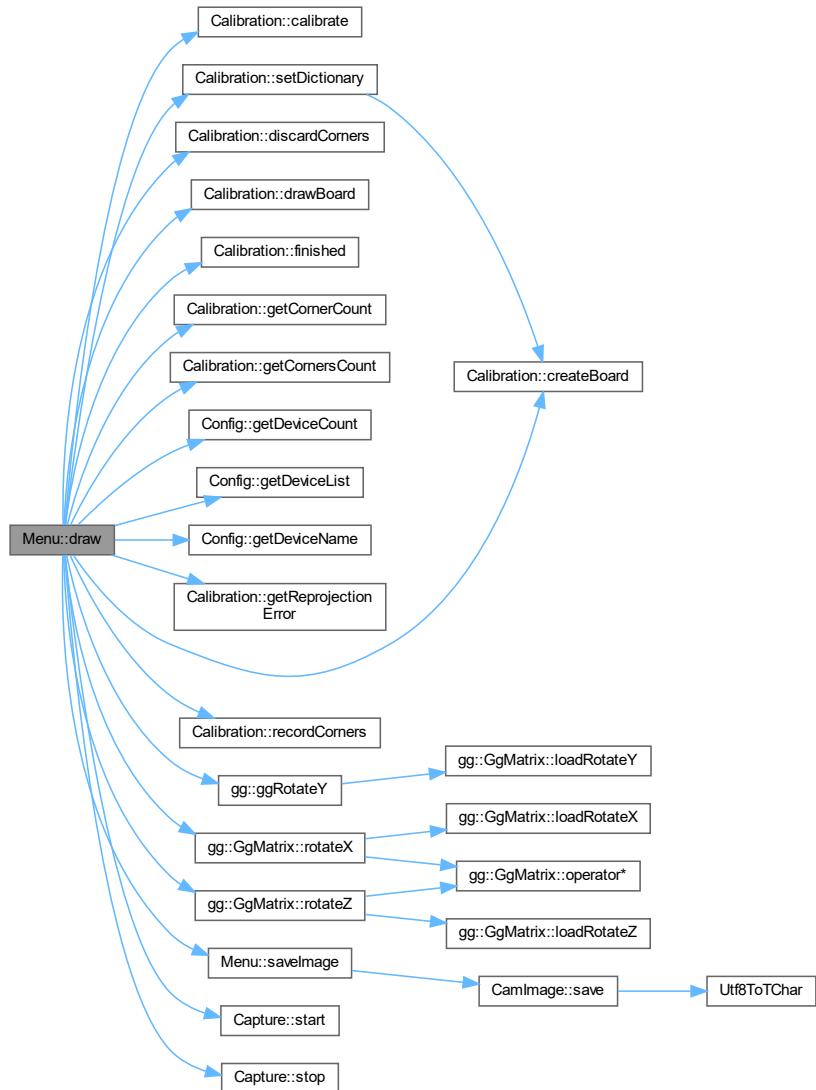
### 8.35.3.1 draw()

```
void Menu::draw ( )
```

メニューを描画する

[Menu.cpp](#) の 387 行目に定義がります。

呼び出し関係図:



### 8.35.3.2 getMenubarHeight()

```
auto Menu::getMenubarHeight ( ) const [inline]
```

メニューバーの高さを得る

戻り値

メニューバーの高さ

[Menu.h](#) の 189 行目に定義があります。

### 8.35.3.3 `getPose()`

```
const auto & Menu::getPose () const [inline]
```

正規化デバイス座標系における焦点距離を求める

戻り値

図形の姿勢

[Menu.h](#) の 179 行目に定義があります。

### 8.35.3.4 `operator bool()`

```
Menu::operator bool () const [inline], [explicit]
```

処理を継続するかどうか調べる

戻り値

処理を継続するなら true

[Menu.h](#) の 169 行目に定義があります。

### 8.35.3.5 `operator=()`

```
Menu & Menu::operator= (
    const Menu & menu ) [delete]
```

代入演算子は使用しない

引数

<i>menu</i>	代入元
-------------	-----

### 8.35.3.6 `saveImage()`

```
void Menu::saveImage (
    const cv::Mat & image,
    const std::string & filename = "*.jpg" )
```

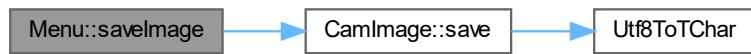
画像の保存

引数

<i>image</i>	保存する画像データ
<i>filename</i>	保存する画像ファイル名のテンプレート

[Menu.cpp](#) の 762 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.35.3.7 setSize()

```
void Menu::setSize (
    const std::array< int, 2 > & size )
```

解像度初期値を設定する

引数

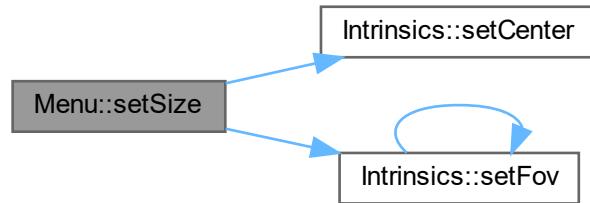
<i>size</i>	キャプチャされるフレームの解像度
-------------	------------------

覚え書き

解像度と現在の焦点距離 *focal* をもとに、撮像系の縦横の画角の初期値も設定する。投影面の中心位置も設定する。

[Menu.cpp](#) の 364 行目に定義があります。

呼び出し関係図:



### 8.35.3.8 setup()

```
std::array< GLsizei, 2 > Menu::setup (
    GLfloat aspect ) const
```

シェーダを設定する

引数

<code>aspect</code>	表示領域の縦横比
---------------------	----------

戻り値

描画すべきメッシュの横と縦の格子点数

覚え書き

格子点数は画角 `aspect` と展開用 メッシュのサンプル点数 `samples` から求める。

[Menu.cpp](#) の 377 行目に定義があります。

呼び出し関係図:



## 8.35.4 メンバ詳解

### 8.35.4.1 detectBoard

```
bool Menu::detectBoard
```

ChArUco Board を検出するなら true

[Menu.h](#) の 134 行目に定義があります。

### 8.35.4.2 detectMarker

```
bool Menu::detectMarker
```

ArUco Marker を検出するなら true

[Menu.h](#) の 131 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [Menu.h](#)
- [Menu.cpp](#)

## 8.36 Mesh クラス

```
#include <Mesh.h>
```

### 公開メンバ関数

- [Mesh \(\)](#)
- [Mesh \(const Mesh &mesh\)=delete](#)
- [virtual ~Mesh \(\)](#)
- [Mesh & operator= \(const Mesh &mesh\)=delete](#)
- [void draw \(const std::array< GLsizei, 2 > &size\) const](#)

### 8.36.1 詳解

メッシュの描画クラス

[Mesh.h](#) の 17 行目に定義があります。

## 8.36.2 構築子と解体子

### 8.36.2.1 Mesh() [1/2]

```
Mesh::Mesh ( ) [inline]
```

コンストラクタ

[Mesh.h](#) の 27 行目に定義がります。

### 8.36.2.2 Mesh() [2/2]

```
Mesh::Mesh ( const Mesh & mesh ) [delete]
```

コピーコンストラクタは使用しない

### 8.36.2.3 ~Mesh()

```
virtual Mesh::~Mesh ( ) [inline], [virtual]
```

デストラクタ

[Mesh.h](#) の 42 行目に定義がります。

## 8.36.3 関数詳解

### 8.36.3.1 draw()

```
void Mesh::draw ( const std::array< GLsizei, 2 > & size ) const [inline]
```

描画

引数

<code>size</code>	描画するメッシュの横と縦の格子点数
-------------------	-------------------

[Mesh.h](#) の 57 行目に定義がります。

被呼び出し関係図:



### 8.36.3.2 operator=()

```
Mesh & Mesh::operator= (
    const Mesh & mesh ) [delete]
```

代入演算子は使用しない

このクラス詳解は次のファイルから抽出されました:

- [Mesh.h](#)

## 8.37 Preference クラス

```
#include <Preference.h>
```

公開メンバ関数

- [Preference \(\)](#)
- [Preference \(const std::string &description, const std::string &vert, const std::string &frag, const Intrinsics &intrinsics=Intrinsics{}\)](#)
- [Preference \(const picojson::object &object\)](#)
- [virtual ~Preference \(\)](#)
- [void buildShader \(\)](#)
- [const auto & getDescription \(\) const](#)
- [const auto & getIntrinsics \(\) const](#)
- [const auto & getShader \(\) const](#)
- [void setPreference \(picojson::object &object\) const](#)

### 8.37.1 詳解

キャプチャデバイスの構成データ

[Preference.h](#) の 20 行目に定義があります。

## 8.37.2 構築子と解体子

### 8.37.2.1 Preference() [1/3]

```
Preference::Preference ( )
```

キャプチャデバイスの構成データのデフォルトコンストラクタ

覚え書き

構成ファイルが読みなったときにしか使わない

[Preference.cpp](#) の 13 行目に定義があります。

### 8.37.2.2 Preference() [2/3]

```
Preference::Preference (
    const std::string & description,
    const std::string & vert,
    const std::string & frag,
    const Intrinsics & intrinsics = Intrinsics{} )
```

シェーダのソースファイルを指定するときに使う キャプチャデバイスの構成データのコンストラクタ

引数

<i>description</i>	説明
<i>vert</i>	バーテックスシェーダのソースファイル名
<i>frag</i>	フレグメントシェーダのソースファイル名
<i>intrinsics</i>	キャプチャデバイス固有のパラメータ

[Preference.cpp](#) の 22 行目に定義があります。

### 8.37.2.3 Preference() [3/3]

```
Preference::Preference (
    const picojson::object & object )
```

構成ファイルを読み込むときに使う キャプチャデバイスの構成データのコンストラクタ

[Preference.cpp](#) の 36 行目に定義があります。

呼び出し関係図:



#### 8.37.2.4 ~Preference()

```
Preference::~Preference () [virtual]
```

デストラクタ

[Preference.cpp](#) の 50 行目に定義がります。

### 8.37.3 関数詳解

#### 8.37.3.1 buildShader()

```
void Preference::buildShader ()
```

シェーダをビルドする

[Preference.cpp](#) の 59 行目に定義がります。

#### 8.37.3.2 getDescription()

```
const auto & Preference::getDescription () const [inline]
```

説明を取り出す

戻り値

構成の説明文

[Preference.h](#) の 80 行目に定義がります。

### 8.37.3.3 getIntrinsics()

```
const auto & Preference::getIntrinsics() const [inline]
```

キャプチャデバイス固有のパラメータを取り出す

戻り値

キャプチャデバイス固有のパラメータ

Preference.h の 90 行目に定義があります。

### 8.37.3.4 getShader()

```
const auto & Preference::getShader() const [inline]
```

シェーダを取り出す

戻り値

この構成のシェーダの参照

Preference.h の 100 行目に定義があります。

### 8.37.3.5 setPreference()

```
void Preference::setPreference(  
    picojson::object & object) const
```

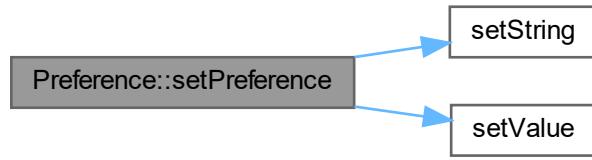
JSON オブジェクトに構成を格納する

引数

<i>object</i>	格納先の JSON オブジェクト
---------------	------------------

Preference.cpp の 71 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [Preference.h](#)
- [Preference.cpp](#)

## 8.38 Settings 構造体

```
#include <Config.h>
```

公開メンバ関数

- `Settings (const std::string &dictionaryName)`
- auto `getFocal () const`

公開変数類

- int `samples`  
展開に用いるメッシュのサンプル数
- std::array< float, 3 > `euler`  
キャプチャデバイスの姿勢のオイラー角
- float `focal`  
描画時の焦点距離
- std::array< float, 2 > `focalRange`  
描画時の焦点距離の範囲
- std::string `dictionaryName`  
使用中の *ArUco Marker* 辞書名
- std::array< float, 2 > `checkerLength`  
*CharUco Board* のマス目一辺の長さと *ArUco Marker* の一辺の長さ (単位 cm)

## 静的公開変数類

- static constexpr decltype(euler) **defaultEuler** { 0.0f, 0.0f, 0.0f }
- キャプチャデバイスの姿勢のデフォルト値
- static constexpr decltype(focal) **defaultFocal** { 50.0f }
- 展開時の焦点距離のデフォルト値
- static constexpr decltype(focalRange) **defaultFocalRange** { 10.0f, 200.0f }
- 描画時の焦点距離の範囲のデフォルト値

### 8.38.1 詳解

表示関連の設定データ

[Config.h](#) の 20 行目に定義があります。

### 8.38.2 構築子と解体子

#### 8.38.2.1 **Settings()**

```
Settings::Settings (
    const std::string & dictionaryName ) [inline]
```

コンストラクタ

[Config.h](#) の 52 行目に定義があります。

### 8.38.3 関数詳解

#### 8.38.3.1 **getFocal()**

```
auto Settings::getFocal ( ) const [inline]
```

正規化デバイス座標系における焦点距離を求める

[Config.h](#) の 64 行目に定義があります。

被呼び出し関係図:



## 8.38.4 メンバ詳解

### 8.38.4.1 checkerLength

```
std::array<float, 2> Settings::checkerLength
```

ChArUco Board のマス目一辺の長さと ArUco Marker の一辺の長さ (単位 cm)

[Config.h](#) の 47 行目に定義があります。

### 8.38.4.2 defaultEuler

```
constexpr decltype(euler) Settings::defaultEuler { 0.0f, 0.0f, 0.0f } [static], [constexpr]
```

キャプチャデバイスの姿勢のデフォルト値

[Config.h](#) の 29 行目に定義があります。

### 8.38.4.3 defaultFocal

```
constexpr decltype(focal) Settings::defaultFocal { 50.0f } [static], [constexpr]
```

展開時の焦点距離のデフォルト値

[Config.h](#) の 35 行目に定義があります。

### 8.38.4.4 defaultFocalRange

```
constexpr decltype(focalRange) Settings::defaultFocalRange { 10.0f, 200.0f } [static], [constexpr]
```

描画時の焦点距離の範囲のデフォルト値

[Config.h](#) の 41 行目に定義があります。

### 8.38.4.5 dictionaryName

```
std::string Settings::dictionaryName
```

使用中の ArUco Marker 辞書名

[Config.h](#) の 44 行目に定義があります。

#### 8.38.4.6 euler

```
std::array<float, 3> Settings::euler
```

キャプチャデバイスの姿勢のオイラー角

[Config.h](#) の 26 行目に定義があります。

#### 8.38.4.7 focal

```
float Settings::focal
```

描画時の焦点距離

[Config.h](#) の 32 行目に定義があります。

#### 8.38.4.8 focalRange

```
std::array<float, 2> Settings::focalRange
```

描画時の焦点距離の範囲

[Config.h](#) の 38 行目に定義があります。

#### 8.38.4.9 samples

```
int Settings::samples
```

展開に用いるメッシュのサンプル数

[Config.h](#) の 23 行目に定義があります。

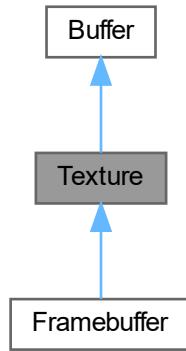
この構造体詳解は次のファイルから抽出されました:

- [Config.h](#)

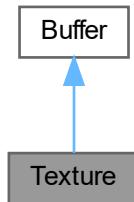
### 8.39 Texture クラス

```
#include <Texture.h>
```

Texture の継承関係図



Texture 連携図



### 公開メンバ関数

- `Texture ()`
- `Texture (GLsizei width, GLsizei height, int channels)`
- `Texture (const Texture &texture)`
- `Texture (Texture &&texture) noexcept`
- `virtual ~Texture ()`
- `Texture & operator= (const Texture &texture)`
- `Texture & operator= (Texture &&texture) noexcept`
- `virtual void create (GLsizei width, GLsizei height, int channels)`
- `virtual void copy (const Buffer &texture) noexcept`
- `virtual void discard ()`

- auto `getTextureName () const`
- virtual const std::array< GLsizei, 2 > & `getSize () const`
- virtual int `getChannels () const`
- void `bindTexture (int unit=0) const`
- void `unbindTexture () const`
- void `readPixels (GLuint buffer) const`
- void `readPixels (Buffer &buffer) const`
- void `readPixels () const`
- void `drawPixels (GLuint buffer) const`
- void `drawPixels (const Texture &texture)`
- void `drawPixels (const Buffer &buffer)`
- void `drawPixels () const`

その他の継承メンバ

### 8.39.1 詳解

テクスチャクラス

`Texture.h` の 17 行目に定義があります。

### 8.39.2 構築子と解体子

#### 8.39.2.1 `Texture()` [1/4]

`Texture::Texture ( ) [inline]`

テクスチャのデフォルトコンストラクタ

`Texture.h` の 33 行目に定義があります。

#### 8.39.2.2 `Texture()` [2/4]

```
Texture::Texture (
    GLsizei width,
    GLsizei height,
    int channels ) [inline]
```

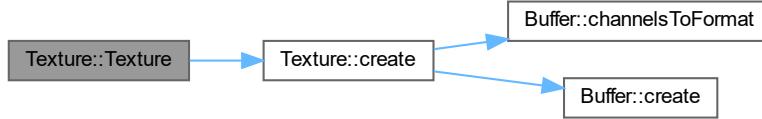
テクスチャを作成するコンストラクタ

引数

<code>width</code>	作成するテクスチャの横の画素数
<code>height</code>	作成するテクスチャの縦の画素数
<code>channels</code>	作成するテクスチャのチャネル数

`Texture.h` の 48 行目に定義があります。

呼び出し関係図:



### 8.39.2.3 `Texture()` [3/4]

```
Texture::Texture (
    const Texture & texture ) [inline]
```

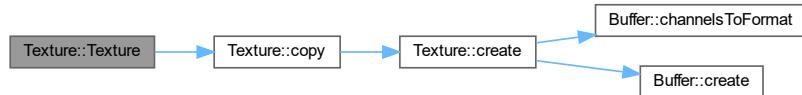
コピー構造

引数

<code>texture</code>	コピー元のテクスチャ
----------------------	------------

`Texture.h` の 58 行目に定義があります。

呼び出し関係図:



### 8.39.2.4 `Texture()` [4/4]

```
Texture::Texture (
    Texture && texture ) [inline], [noexcept]
```

ムーブ構造

引数

<i>texture</i>	ムーブ元のテクスチャ
----------------	------------

Texture.h の 68 行目に定義があります。

### 8.39.2.5 ~Texture()

```
virtual Texture::~Texture () [inline], [virtual]
```

デストラクタ

Texture.h の 76 行目に定義があります。

呼び出し関係図:



## 8.39.3 関数詳解

### 8.39.3.1 bindTexture()

```
void Texture::bindTexture (
    int unit = 0 ) const [inline]
```

テクスチャユニットを指定してテクスチャを結合する

引数

<i>unit</i>	テクスチャユニット番号
-------------	-------------

覚え書き

*unit* にはシェーダにおいてテクスチャのサンプラーの uniform 変数に設定する GL\_TEXTURE*i* の *i* と一致させること。

[Texture.h](#) の 164 行目に定義があります。

呼び出し関係図:



### 8.39.3.2 copy()

```
void Texture::copy (
    const Buffer & texture ) [virtual], [noexcept]
```

テクスチャをコピーする

引数

<code>texture</code>	コピー元のテクスチャ
----------------------	------------

覚え書き

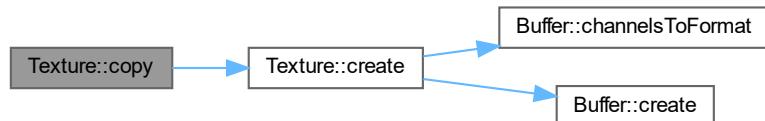
このテクスチャのサイズがコピー元と異なれば、このテクスチャを削除して新しいテクスチャを作り直してコピーする。引数の型が [Buffer](#) なのは、このオブジェクトの型が [Buffer](#) のとき [Buffer::copy\(\)](#)、[Texture](#) のとき [Texture::copy\(\)](#)、[Framebuffer](#) のとき [Framebuffer::copy\(\)](#) を呼ぶようにするため。

[Buffer](#)を再実装しています。

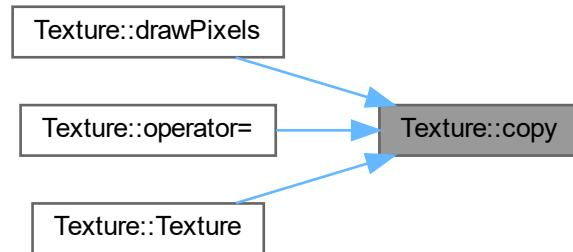
[Framebuffer](#)で再実装されています。

[Texture.cpp](#) の 46 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.39.3.3 create()

```
void Texture::create (
    GLsizei width,
    GLsizei height,
    int channels ) [virtual]
```

テクスチャを作成する

引数

<i>width</i>	作成するテクスチャの横の画素数
<i>height</i>	作成するテクスチャの縦の画素数
<i>channels</i>	作成するテクスチャのチャネル数

覚え書き

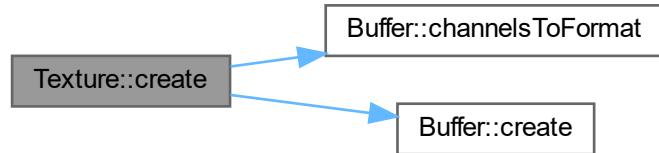
このテクスチャのサイズが引数で指定したサイズと異なれば、このテクスチャを削除して新しいテクスチャを作り直す。

Bufferを再実装しています。

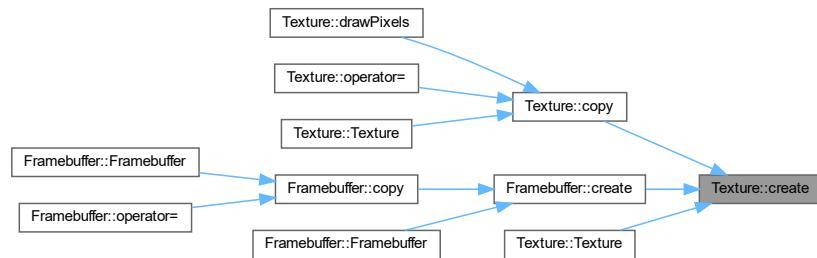
Framebufferで再実装されています。

Texture.cpp の 13 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.39.3.4 discard()

`void Texture::discard ( ) [virtual]`

テクスチャを破棄する

[Buffer](#)を再実装しています。

[Framebuffer](#)で再実装されています。

[Texture.cpp](#) の 97 行目に定義があります。

被呼び出し関係図:



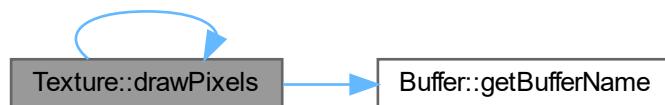
**8.39.3.5 drawPixels() [1/4]**

```
void Texture::drawPixels ( ) const [inline]
```

ピクセルバッファオブジェクトからテクスチャにデータをコピーする

[Texture.h](#) の 286 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

**8.39.3.6 drawPixels() [2/4]**

```
void Texture::drawPixels ( const Buffer & buffer ) [inline]
```

指定したオブジェクトからテクスチャにデータをコピーする

引数

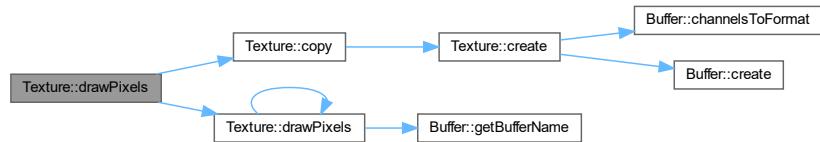
<i>buffer</i>	テクスチャをコピーする先のオブジェクト
---------------	---------------------

覚え書き

テクスチャのサイズをコピー元のオブジェクトのバッファに合わせる。

[Texture.h](#) の 274 行目に定義があります。

呼び出し関係図:



### 8.39.3.7 drawPixels() [3/4]

```
void Texture::drawPixels (
    const Texture & texture ) [inline]
```

指定したテクスチャからデータをコピーする

引数

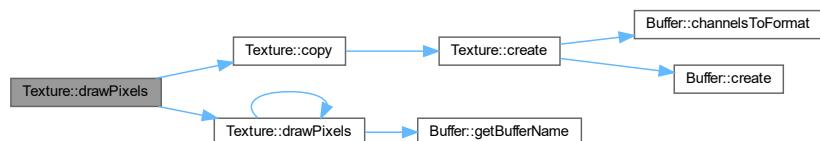
<code>texture</code>	コピー元のテクスチャ
----------------------	------------

覚え書き

テクスチャのサイズをコピー元のテクスチャに合わせる。

`Texture.h` の 257 行目に定義があります。

呼び出し関係図:



### 8.39.3.8 drawPixels() [4/4]

```
void Texture::drawPixels (
    GLuint buffer ) const
```

指定したピクセルバッファオブジェクトからテクスチャにデータをコピーする

引数

<code>buffer</code>	コピーするテクスチャを格納したピクセルバッファオブジェクト名
---------------------	--------------------------------

覚え書き

コピーするデータのサイズが一致している必要がある。

[Texture.cpp](#) の 148 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.39.3.9 getChannels()

```
virtual int Texture::getChannels() const [inline], [virtual]
```

テクスチャのチャネル数を得る

[Buffer](#)を再実装しています。

[Framebuffer](#)で再実装されています。

[Texture.h](#) の 150 行目に定義があります。

### 8.39.3.10 getSize()

```
virtual const std::array< GLsizei, 2 > & Texture::getSize ( ) const [inline], [virtual]
```

テクスチャのサイズを得る

[Buffer](#)を再実装しています。

[Framebuffer](#)で再実装されています。

[Texture.h](#) の 142 行目に定義があります。

### 8.39.3.11 getTextureName()

```
auto Texture::getTextureName ( ) const [inline]
```

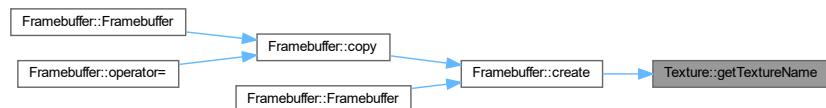
テクスチャ名を得る

戻り値

テクスチャ名

[Texture.h](#) の 134 行目に定義があります。

被呼び出し関係図:



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

```
Texture & Texture::operator= (
    const Texture & texture )
```

代入演算子

引数

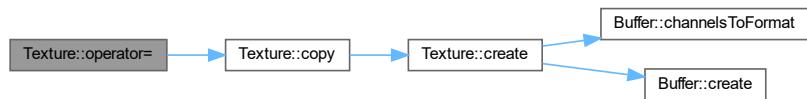
<i>texture</i>	代入元のテクスチャ
----------------	-----------

戻り値

代入結果のテクスチャ

Texture.cpp の 59 行目に定義がります。

呼び出し関係図:



### 8.39.3.13 operator=(Texture&) [2/2]

```
Texture & Texture::operator= (
    Texture && texture ) [noexcept]
```

ムーブ代入演算子

引数

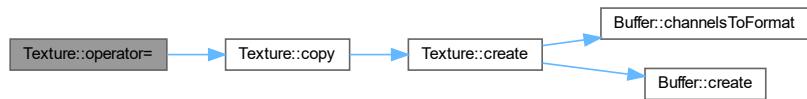
texture	ムーブ代入元のテクスチャ
---------	--------------

戻り値

ムーブ代入結果のテクスチャ

Texture.cpp の 75 行目に定義がります。

呼び出し関係図:



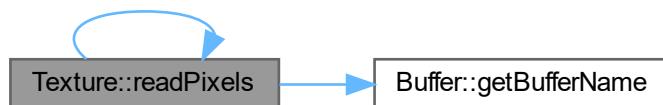
### 8.39.3.14 `readPixels()` [1/3]

```
void Texture::readPixels ( ) const [inline]
```

テクスチャからピクセルバッファオブジェクトにデータをコピーする

[Texture.h](#) の 224 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.39.3.15 `readPixels()` [2/3]

```
void Texture::readPixels (
    Buffer & buffer ) const [inline]
```

テクスチャから指定したオブジェクトのバッファにデータをコピーする

引数

<code>buffer</code>	テクスチャをコピーする先のオブジェクト
---------------------	---------------------

覚え書き

コピーする先のオブジェクトのサイズをテクスチャに合わせる。

[Texture.h](#) の 209 行目に定義があります。

呼び出し関係図:



#### 8.39.3.16 `readPixels()` [3/3]

```
void Texture::readPixels (
    GLuint buffer ) const
```

テクスチャから指定したピクセルバッファオブジェクトにデータをコピーする

引数

<code>buffer</code>	テクスチャをコピーする先のピクセルバッファオブジェクト名
---------------------	------------------------------

覚え書き

コピーするデータのサイズが一致している必要がある。

[Texture.cpp](#) の 114 行目に定義があります。

#### 8.39.3.17 `unbindTexture()`

```
void Texture::unbindTexture ( ) const [inline]
```

テクスチャの結合を解除する

[Texture.h](#) の 176 行目に定義があります。

被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

- [Texture.h](#)
- [Texture.cpp](#)

## 8.40 GgApp::Window クラス

```
#include <GgApp.h>
```

### 公開メンバ関数

- `Window (const std::string &title="GLFW Window", int width=640, int height=480, int fullscreen=0, GLFWwindow *share=nullptr)`
- `Window (const Window &w)=delete`
- `virtual ~Window ()`
- `Window & operator= (const Window &w)=delete`
- `auto * get () const`
- `void setClose (int flag=GLFW_TRUE) const`
- `bool shouldClose () const`
- `operator bool ()`
- `void swapBuffers () const`
- `void restoreViewport () const`
- `void updateViewport ()`
- `auto getWidth () const`
- `auto getHeight () const`
- `auto getFboWidth () const`
- `auto getFboHeight () const`
- `const auto & getSize () const`
- `const auto & getFboSize () const`
- `auto getAspect () const`
- `bool getKey (int key) const`
- `void selectInterface (int no)`
- `void setVelocity (GLfloat vx, GLfloat vy, GLfloat vz=0.1f)`
- `int getLastKey ()`
- `auto getArrow (int direction=0, int mods=0) const`
- `auto getArrowX (int mods=0) const`
- `auto getArrowY (int mods=0) const`
- `void getArrow (GLfloat *arrow, int mods=0) const`
- `auto getShiftArrowX () const`
- `auto getShiftArrowY () const`
- `void getShiftArrow (GLfloat *shift_arrow) const`
- `auto getControlArrowX () const`
- `auto getControlArrowY () const`
- `void getControlArrow (GLfloat *control_arrow) const`
- `auto getAltArrowX () const`
- `auto getAltArrowY () const`
- `void getAltArrow (GLfloat *alt_arrow) const`
- `const auto * getMouse () const`
- `void getMouse (GLfloat *position) const`
- `auto getMouse (int direction) const`
- `auto getX () const`
- `auto getY () const`
- `const auto * getWheel () const`
- `void getWheel (GLfloat *rotation) const`
- `auto getWheel (int direction) const`
- `auto getWheelX () const`
- `auto getWheelY () const`
- `const auto & getTranslation (int button=GLFW_MOUSE_BUTTON_1) const`

- auto `getTranslationMatrix` (int button=GLFW\_MOUSE\_BUTTON\_1) const
- auto `getScrollMatrix` (int button=GLFW\_MOUSE\_BUTTON\_1) const
- auto `getRotation` (int button=GLFW\_MOUSE\_BUTTON\_1) const
- auto `getRotationMatrix` (int button=GLFW\_MOUSE\_BUTTON\_1) const
- void `resetRotation` ()
- void `resetTranslation` ()
- void `reset` ()
- void \* `getUserPointer` () const
- void `setUserPointer` (void \*pointer)
- void `setResizeFunc` (void(\*func)(const Window \*window, int width, int height))
- void `setKeyboardFunc` (void(\*func)(const Window \*window, int key, int scancode, int action, int mods))
- void `setMouseFunc` (void(\*func)(const Window \*window, int button, int action, int mods))
- void `setWheelFunc` (void(\*func)(const Window \*window, double x, double y))

#### 8.40.1 詳解

ウィンドウ関連の処理.

覚え書き

GLFW を使って OpenGL のウィンドウを操作するラッパークラス.

GgApp.h の 102 行目に定義があります。

#### 8.40.2 構築子と解体子

##### 8.40.2.1 Window() [1/2]

```
GgApp::Window::Window (
    const std::string & title = "GLFW Window",
    int width = 640,
    int height = 480,
    int fullscreen = 0,
    GLFWwindow * share = nullptr )
```

コンストラクタ.

引数

<code>title</code>	ウィンドウタイトルの文字列.
<code>width</code>	開くウィンドウの幅, フルスクリーン時は無視され実際のディスプレイの幅が使われる.
<code>height</code>	開くウィンドウの高さ, フルスクリーン時は無視され実際のディスプレイの高さが使われる.
<code>fullscreen</code>	フルスクリーン表示を行うディスプレイ番号, 0ならフルスクリーン表示を行わない.
<code>share</code>	共有するコンテキスト, <code>nullptr</code> ならコンテキストを共有しない.

[GgApp.cpp](#) の 324 行目に定義がります。

呼び出し関係図:



#### 8.40.2.2 Window() [2/2]

```
GgApp::Window::Window (
    const Window & w ) [delete]
```

コピー構造子は使用しない

#### 8.40.2.3 ~Window()

```
virtual GgApp::Window::~Window ( ) [inline], [virtual]
```

デストラクタ.

[GgApp.h](#) の 227 行目に定義がります。

### 8.40.3 関数詳解

#### 8.40.3.1 get()

```
auto * GgApp::Window::get ( ) const [inline]
```

ウィンドウの識別子のポインタを取得する。

戻り値

GLFWwindow 型のウィンドウ識別子のポインタ.

[GgApp.h](#) の 246 行目に定義がります。

### 8.40.3.2 getAltArrowX()

```
auto GgApp::Window::getAltArrowX ( ) const [inline]
```

ALT キーを押しながら矢印キーを押したときの現在の X 値を得る.

戻り値

ALT キーを押しながら矢印キーを押したときの現在の X 値.

[GgApp.h](#) の 548 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.40.3.3 getAltArrowY()

```
auto GgApp::Window::getAltArrowY ( ) const [inline]
```

ALT キーを押しながら矢印キーを押したときの現在の Y 値を得る.

戻り値

ALT キーを押しながら矢印キーを押したときの現在の Y 値.

GgApp.h の 558 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.40.3.4 getAltArrow()

```
void GgApp::Window::getAltArrow (
    GLfloat * alt_arrow ) const [inline]
```

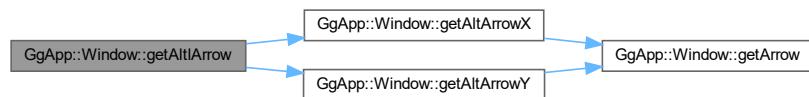
ALT キーを押しながら矢印キーを押したときの現在の値を得る.

引数

<i>alt_arrow</i>	ALT キーを押しながら矢印キーを押したときの値を格納する GLfloat 型の 2 要素の配列.
------------------	---

GgApp.h の 568 行目に定義があります。

呼び出し関係図:



### 8.40.3.5 getArrow() [1/2]

```
void GgApp::Window::getArrow (
    GLfloat * arrow,
    int mods = 0 ) const [inline]
```

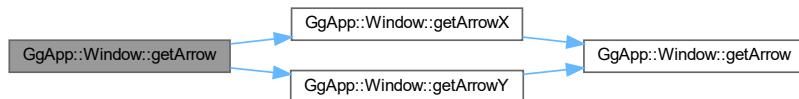
矢印キーの現在の値を得る.

引数

<i>arrow</i>	矢印キーの値を格納する GLfloat[2] の配列.
<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).

GgApp.h の 475 行目に定義があります。

呼び出し関係図:



### 8.40.3.6 getArrow() [2/2]

```
auto GgApp::Window::getArrow (
    int direction = 0,
    int mods = 0 ) const [inline]
```

矢印キーの現在の値を得る.

引数

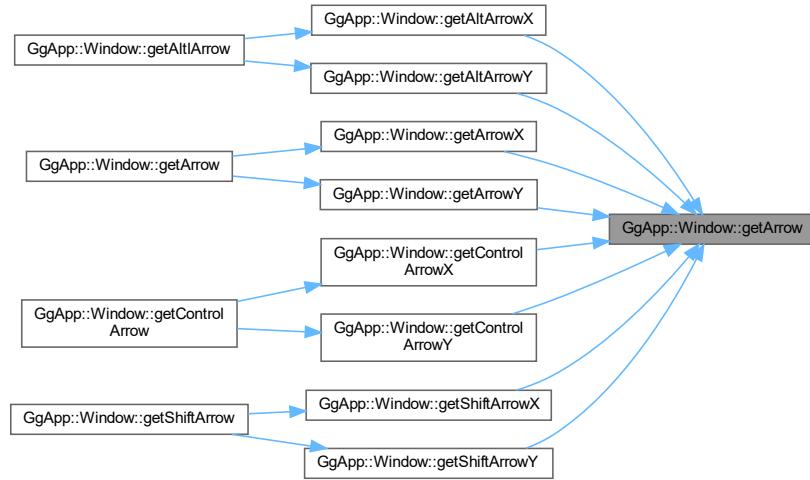
<i>direction</i>	方向 (0: X, 1:Y).
<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).

戻り値

矢印キーの値.

GgApp.h の 441 行目に定義があります。

被呼び出し関係図:



#### 8.40.3.7 getArrowX()

```
auto GgApp::Window::getArrowX (
    int mods = 0 ) const [inline]
```

矢印キーの現在の X 値を得る.

引数

<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).
-------------	--

戻り値

矢印キーの X 値.

GgApp.h の 453 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.40.3.8 getArrowY()

```
auto GgApp::Window::getArrowY ( int mods = 0 ) const [inline]
```

矢印キーの現在の Y 値を得る.

引数

<i>mods</i>	修飾キーの状態 (0:なし, 1: SHIFT, 2: CTRL, 3: ALT).
-------------	--

戻り値

矢印キーの Y 値.

GgApp.h の 464 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.40.3.9 `getAspect()`

```
auto GgApp::Window::getAspect ( ) const [inline]
```

ウィンドウの縦横比を得る.

戻り値

ウィンドウの縦横比.

[GgApp.h](#) の 378 行目に定義があります。

### 8.40.3.10 `getControlArrow()`

```
void GgApp::Window::getControlArrow (
    GLfloat * control_arrow ) const [inline]
```

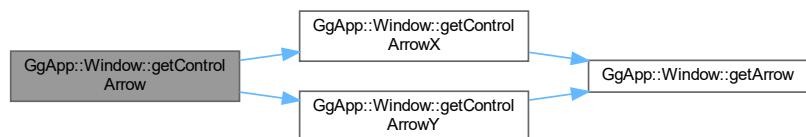
CTRL キーを押しながら矢印キーを押したときの現在の値を得る.

引数

<code>control_arrow</code>	CTRL キーを押しながら矢印キーを押したときの値を格納する GLfloat 型の 2 要素の配列.
----------------------------	--

[GgApp.h](#) の 537 行目に定義があります。

呼び出し関係図:



### 8.40.3.11 `getControlArrowX()`

```
auto GgApp::Window::getControlArrowX ( ) const [inline]
```

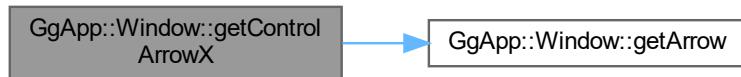
CTRL キーを押しながら矢印キーを押したときの現在の X 値を得る.

戻り値

CTRL キーを押しながら矢印キーを押したときの現在の X 値.

GgApp.h の 517 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.40.3.12 getControlArrowY()

```
auto GgApp::Window::getControlArrowY ( ) const [inline]
```

CTRL キーを押しながら矢印キーを押したときの現在の Y 値を得る.

戻り値

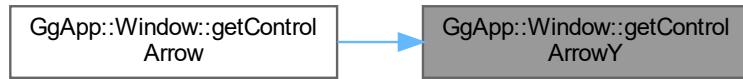
CTRL キーを押しながら矢印キーを押したときの現在の Y 値.

GgApp.h の 527 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.40.3.13 getFboHeight()

`auto GgApp::Window::getFboHeight () const [inline]`

FBO の高さを得る.

戻り値

FBO の高さ.

[GgApp.h](#) の 348 行目に定義があります。

#### 8.40.3.14 getFboSize()

`const auto & GgApp::Window::getFboSize () const [inline]`

FBO のサイズを得る.

戻り値

FBO の幅と高さを格納した `GLsizei` 型の 2 要素の配列.

[GgApp.h](#) の 368 行目に定義があります。

#### 8.40.3.15 getFboWidth()

`auto GgApp::Window::getFboWidth () const [inline]`

FBO の横幅を得る.

戻り値

FBO の横幅.

[GgApp.h](#) の 338 行目に定義があります。

#### 8.40.3.16 getHeight()

```
auto GgApp::Window::getHeight ( ) const [inline]
```

ウィンドウの高さを得る.

戻り値

  ウィンドウの高さ.

[GgApp.h](#) の 328 行目に定義があります。

#### 8.40.3.17 getKey()

```
bool GgApp::Window::getKey (
    int key ) const [inline]
```

キーが押されているかどうかを判定する.

戻り値

  キーが押されていれば true.

[GgApp.h](#) の 388 行目に定義があります。

#### 8.40.3.18 getLastKey()

```
int GgApp::Window::getLastKey ( ) [inline]
```

最後にタイプしたキーを得る.

戻り値

  最後にタイプしたキーの文字.

[GgApp.h](#) の 426 行目に定義があります。

#### 8.40.3.19 getMouse() [1/3]

```
const auto * GgApp::Window::getMouse ( ) const [inline]
```

マウスカーソルの現在位置を得る.

戻り値

  マウスカーソルの現在位置を格納した GLfloat 型の 2 要素の配列.

[GgApp.h](#) の 579 行目に定義があります。

#### 8.40.3.20 getMouse() [2/3]

```
void GgApp::Window::getMouse (
    GLfloat * position ) const [inline]
```

マウスカーソルの現在位置を得る.

引数

<i>position</i>	マウスカーソルの現在位置を格納した GLfloat 型の 2 要素の配列.
-----------------	---------------------------------------

GgApp.h の 590 行目に定義がります。

#### 8.40.3.21 getMouse() [3/3]

```
auto GgApp::Window::getMouse (
    int direction ) const [inline]
```

マウスカーソルの現在位置を得る.

引数

<i>direction</i>	方向 (0:X, 1:Y).
------------------	----------------

戻り値

*direction* 方向のマウスカーソルの現在位置.

GgApp.h の 603 行目に定義がります。

#### 8.40.3.22 getMouseX()

```
auto GgApp::Window::getMouseX ( ) const [inline]
```

マウスカーソルの現在位置の X 座標を得る.

戻り値

*direction* 方向のマウスカーソルの X 方向の現在位置.

GgApp.h の 614 行目に定義がります。

#### 8.40.3.23 getMouseY()

```
auto GgApp::Window::getMouseY ( ) const [inline]
```

マウスカーソルの現在位置の Y 座標を得る.

戻り値

*direction* 方向のマウスカーソルの Y 方向の現在位置.

GgApp.h の 625 行目に定義がります。

#### 8.40.3.24 getRotation()

```
auto GgApp::Window::getRotation (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボールの回転変換行列を得る。

引数

<i>button</i>	回転変換行列を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	--

戻り値

回転を行う GgQuaternion 型の四元数。

GgApp.h の 747 行目に定義があります。

#### 8.40.3.25 getRotationMatrix()

```
auto GgApp::Window::getRotationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボールの回転変換行列を得る。

引数

<i>button</i>	回転変換行列を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	--

戻り値

回転を行う GgMatrix 型の変換行列。

GgApp.h の 760 行目に定義があります。

#### 8.40.3.26 getScrollMatrix()

```
auto GgApp::Window::getScrollMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによってオブジェクトの平行移動の変換行列を得る。

引数

<i>button</i>	平行移動量を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	---

戻り値

クリッピング座標系で平行移動を行う GgMatrix 型の変換行列.

GgApp.h の 727 行目に定義があります。

#### 8.40.3.27 getShiftArrow()

```
void GgApp::Window::getShiftArrow (
    GLfloat * shift_arrow ) const [inline]
```

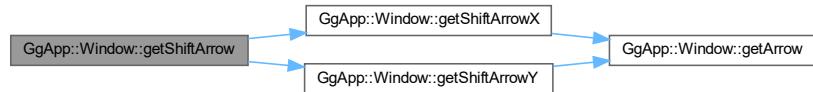
SHIFT キーを押しながら矢印キーを押したときの現在の値を得る.

引数

<i>shift_arrow</i>	SHIFT キーを押しながら矢印キーを押したときの値を格納する GLfloat 型の 2 要素の配列.
--------------------	---

GgApp.h の 506 行目に定義があります。

呼び出し関係図:



#### 8.40.3.28 getShiftArrowX()

```
auto GgApp::Window::getShiftArrowX ( ) const [inline]
```

SHIFT キーを押しながら矢印キーを押したときの現在の X 値を得る.

戻り値

SHIFT キーを押しながら矢印キーを押したときの現在の X 値.

GgApp.h の 486 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



#### 8.40.3.29 getShiftArrowY()

```
auto GgApp::Window::getShiftArrowY ( ) const [inline]
```

SHIFT キーを押しながら矢印キーを押したときの現在の Y 値を得る.

戻り値

SHIFT キーを押しながら矢印キーを押したときの現在の Y 値.

GgApp.h の 496 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



### 8.40.3.30 getSize()

```
const auto & GgApp::Window::getSize () const [inline]
```

ウィンドウのサイズを得る.

戻り値

ウィンドウの幅と高さを格納した `GLsizei` 型の 2 要素の配列の参照.

[GgApp.h](#) の 358 行目に定義があります。

### 8.40.3.31 getTranslation()

```
const auto & GgApp::Window::getTranslation (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボール処理を考慮したマウスによるスクロールの変換行列を得る.

引数

<code>button</code>	平行移動量を取得するマウスボタン ( <code>GLFW_MOUSE_BUTTON_[1,2]</code> ).
---------------------	--

戻り値

平行移動量を格納した `GLfloat[3]` の配列のポインタ.

[GgApp.h](#) の 694 行目に定義があります。

### 8.40.3.32 getTranslationMatrix()

```
auto GgApp::Window::getTranslationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによって視点の平行移動の変換行列を得る.

引数

<code>button</code>	平行移動量を取得するマウスボタン ( <code>GLFW_MOUSE_BUTTON_[1,2]</code> ).
---------------------	--

戻り値

視点座標系で平行移動を行う `GgMatrix` 型の変換行列.

GgApp.h の 707 行目に定義がります。

#### 8.40.3.33 getUserPointer()

```
void * GgApp::Window::getUserPointer ( ) const [inline]
```

ユーザー ポインタを取り出す.

戻り値

保存されているユーザ ポインタ.

GgApp.h の 802 行目に定義がります。

#### 8.40.3.34 getWheel() [1/3]

```
const auto * GgApp::Window::getWheel ( ) const [inline]
```

マウスホイールの回転量を得る.

戻り値

マウスホイールの回転量を格納した GLfloat 型の 2 要素の配列.

GgApp.h の 636 行目に定義がります。

#### 8.40.3.35 getWheel() [2/3]

```
void GgApp::Window::getWheel (   
    GLfloat * rotation ) const [inline]
```

マウスホイールの回転量を得る.

引数

<i>rotation</i>	マウスホイールの回転量を格納した GLfloat 型の 2 要素の配列.
-----------------	--------------------------------------

GgApp.h の 647 行目に定義がります。

#### 8.40.3.36 `getWheel()` [3/3]

```
auto GgApp::Window::getWheel ( int direction ) const [inline]
```

マウスホイールの回転量を得る。

引数

<i>direction</i>	方向 (0:X, 1:Y).
------------------	----------------

戻り値

*direction* 方向のマウスホイールの回転量。

[GgApp.h](#) の 660 行目に定義があります。

#### 8.40.3.37 `getWheelX()`

```
auto GgApp::Window::getWheelX ( ) const [inline]
```

マウスホイールの X 方向の回転量を得る。

戻り値

マウスホイールの X 方向の回転量。

[GgApp.h](#) の 671 行目に定義があります。

#### 8.40.3.38 `getWheelY()`

```
auto GgApp::Window::getWheelY ( ) const [inline]
```

マウスホイールの Y 方向の回転量を得る。

戻り値

マウスホイールの Y 方向の回転量。

[GgApp.h](#) の 682 行目に定義があります。

### 8.40.3.39 getWidth()

```
auto GgApp::Window::getWidth ( ) const [inline]
```

ウィンドウの横幅を得る.

戻り値

ウィンドウの横幅.

GgApp.h の 318 行目に定義があります。

被呼び出し関係図:



### 8.40.3.40 operator bool()

```
GgApp::Window::operator bool ( ) [explicit]
```

イベントを取得してループを継続すべきかどうか調べる.

戻り値

ループを継続すべきなら true.

GgApp.cpp の 417 行目に定義があります。

### 8.40.3.41 operator=( )

```
Window & GgApp::Window::operator= (
    const Window & w) [delete]
```

代入演算子は使用しない

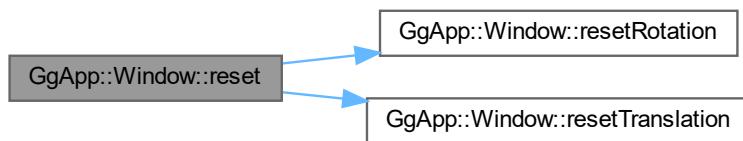
#### 8.40.3.42 reset()

```
void GgApp::Window::reset ( ) [inline]
```

トラックボール・マウスホイール・矢印キーの値を初期化する。

[GgApp.h](#) の 788 行目に定義があります。

呼び出し関係図:



#### 8.40.3.43 resetRotation()

```
void GgApp::Window::resetRotation ( ) [inline]
```

トラックボール処理を初期化する。

[GgApp.h](#) の 770 行目に定義があります。

被呼び出し関係図:



#### 8.40.3.44 resetTranslation()

```
void GgApp::Window::resetTranslation ( ) [inline]
```

平行移動量を初期化する。

GgApp.h の 779 行目に定義があります。

被呼び出し関係図:



#### 8.40.3.45 restoreViewport()

```
void GgApp::Window::restoreViewport ( ) const [inline]
```

ビューポートを元のサイズに復帰する。

GgApp.h の 287 行目に定義があります。

#### 8.40.3.46 selectInterface()

```
void GgApp::Window::selectInterface ( int no ) [inline]
```

インターフェースを選択する。

引数

no	インターフェース番号.
----	-------------

GgApp.h の 403 行目に定義があります。

#### 8.40.3.47 setClose()

```
void GgApp::Window::setClose ( int flag = GLFW_TRUE ) const [inline]
```

ウィンドウのクローズフラグを設定する。

引数

<code>flag</code>	クローズフラグ, 0 (GLFW_FALSE) 以外ならウィンドウを閉じる.
-------------------	--

GgApp.h の 256 行目に定義があります。

#### 8.40.3.48 setKeyboardFunc()

```
void GgApp::Window::setKeyboardFunc (
    void(*)(const Window *window, int key, int scancode, int action, int mods) func )
[inline]
```

ユーザ定義の `keyboard` 関数を設定する。

引数

<code>func</code>	ユーザ定義の <code>keyboard</code> 関数, キーボードの操作時に呼び出される.
-------------------	--

GgApp.h の 832 行目に定義があります。

#### 8.40.3.49 setMouseFunc()

```
void GgApp::Window::setMouseFunc (
    void(*)(const Window *window, int button, int action, int mods) func ) [inline]
```

ユーザ定義の `mouse` 関数を設定する。

引数

<code>func</code>	ユーザ定義の <code>mouse</code> 関数, マウスボタンの操作時に呼び出される.
-------------------	--

GgApp.h の 842 行目に定義があります。

#### 8.40.3.50 setResizeFunc()

```
void GgApp::Window::setResizeFunc (
    void(*)(const Window *window, int width, int height) func ) [inline]
```

ユーザ定義の `resize` 関数を設定する。

引数

<i>func</i>	ユーザ定義の <code>resize</code> 関数、ウィンドウのサイズ変更時に呼び出される。
-------------	--

`GgApp.h` の 822 行目に定義があります。

#### 8.40.3.51 `setUserPointer()`

```
void GgApp::Window::setUserPointer (
    void * pointer ) [inline]
```

任意のユーザポインタを保存する。

引数

<i>pointer</i>	保存するユーザポインタ。
----------------	--------------

`GgApp.h` の 812 行目に定義があります。

#### 8.40.3.52 `setVelocity()`

```
void GgApp::Window::setVelocity (
    GLfloat vx,
    GLfloat vy,
    GLfloat vz = 0.1f ) [inline]
```

マウスの移動速度を設定する。

引数

<i>vx</i>	x 方向の移動速度。
<i>vy</i>	y 方向の移動速度。
<i>vz</i>	z 方向の移動速度。

`GgApp.h` の 416 行目に定義があります。

#### 8.40.3.53 `setWheelFunc()`

```
void GgApp::Window::setWheelFunc (
    void(*)(const Window *window, double x, double y) func ) [inline]
```

ユーザ定義の `wheel` 関数を設定する。

引数

<code>func</code>	ユーザ定義の <code>wheel</code> 関数、マウスホイールの操作時に呼び出される。
-------------------	--

GgApp.h の 852 行目に定義があります。

#### 8.40.3.54 shouldClose()

```
bool GgApp::Window::shouldClose ( ) const [inline]
```

ウィンドウを閉じるべきかどうか調べる。

戻り値

ウィンドウを閉じるべきなら `true`。

GgApp.h の 266 行目に定義があります。

#### 8.40.3.55 swapBuffers()

```
void GgApp::Window::swapBuffers ( ) const
```

カラーバッファを入れ替える。

GgApp.cpp の 467 行目に定義があります。

#### 8.40.3.56 updateViewport()

```
void GgApp::Window::updateViewport ( )
```

ビューポートのサイズを更新する。

GgApp.cpp の 485 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [GgApp.h](#)
- [GgApp.cpp](#)



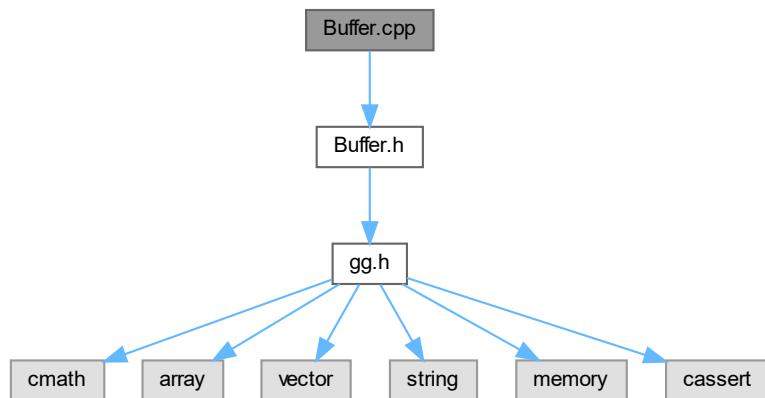
# Chapter 9

## ファイル詳解

### 9.1 Buffer.cpp ファイル

```
#include "Buffer.h"
```

Buffer.cpp の依存先関係図:



#### 9.1.1 詳解

バッファクラスの実装

著者

Kohe Tokoi

日付

February 20, 2024

[Buffer.cpp](#) に定義があります。

## 9.2 Buffer.cpp

### [詳解]

```

00001
00008 #include "Buffer.h"
00009
00010 //
00011 // フォーマットからチャネル数を求める
00012 //
00013 int Buffer::formatToChannels(GLenum format) const
00014 {
00015     switch (format)
00016     {
00017         case GL_RED:
00018             return 1;
00019         case GL_RG:
00020             return 2;
00021         case GL_RGB:
00022             return 3;
00023         case GL_RGBA:
00024             return 4;
00025         default:
00026             assert(false);
00027             break;
00028     }
00029
00030     return 0;
00031 };
00032
00033 //
00034 // 既存のバッファにコピーする
00035 //
00036 void Buffer::copyBuffer(const Buffer& buffer) noexcept
00037 {
00038 #if defined(USE_PIXEL_BUFFER_OBJECT)
00039     // コピー元のピクセルバッファオブジェクト
00040     glBindBuffer(GL_COPY_READ_BUFFER, buffer.bufferName);
00041
00042     // コピー元のピクセルバッファオブジェクトのサイズを調べる
00043     GLint readSize;
00044     glGetBufferParameteriv(GL_COPY_READ_BUFFER, GL_BUFFER_SIZE, &readSize);
00045
00046     // コピー先のピクセルバッファオブジェクト
00047     glBindBuffer(GL_COPY_WRITE_BUFFER, bufferName);
00048
00049     // コピー先のピクセルバッファオブジェクトのサイズを調べる
00050     GLint writeSize;
00051     glGetBufferParameteriv(GL_COPY_WRITE_BUFFER, GL_BUFFER_SIZE, &writeSize);
00052
00053     // バッファサイズが一致しているか確認する
00054     assert(readSize == writeSize);
00055
00056     // バッファオブジェクトをコピーする
00057     glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
00058         0, 0, std::min(readSize, writeSize));
00059 #else
00060     // フレームのデータをコピーする
00061     memcpy(bufferName.data(), buffer.bufferName.data(), bufferName.size());
00062 #endif
00063 }
00064
00065 //
00066 // バッファを作成する
00067 //
00068 void Buffer::create(GLsizei width, GLsizei height, int channels)
00069 {
00070     // 指定したサイズが保持しているフレームのサイズと同じなら何もしない
00071     if (width == bufferSize[0] && height == bufferSize[1]
00072         && channels == bufferChannels) return;
00073
00074     // フレームのサイズとチャンネル数を記録する
00075     bufferSize = std::array<int, 2>{ width, height };
00076     bufferChannels = channels;
00077
00078     // フレームの保存に必要なメモリ量を求める
00079     const auto size{ width * height * channels };
00080
00081 #if defined(USE_PIXEL_BUFFER_OBJECT)
00082     // 以前のピクセルバッファオブジェクトを破棄する
00083     glDeleteBuffers(1, &bufferName);
00084
00085     // 新しいピクセルバッファオブジェクトを作成する
00086     glGenBuffers(1, &bufferName);
00087
00088     // ピクセルバッファオブジェクトのメモリを確保してデータを転送する

```

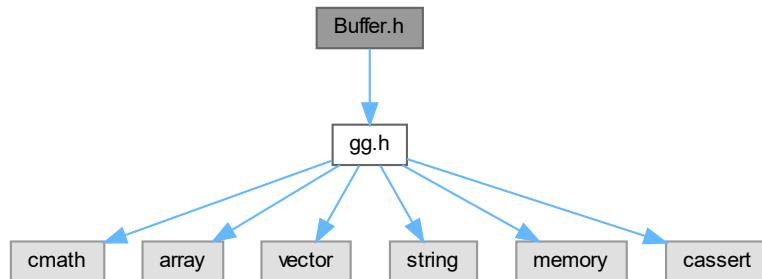
```
00089 glBindBuffer(GL_PIXEL_PACK_BUFFER, bufferName);
00090 glBufferData(GL_PIXEL_PACK_BUFFER, size, nullptr, GL_DYNAMIC_COPY);
00091 glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00092 #else
00093 // メモリを確保する
00094 bufferName.resize(size);
00095 #endif
00096 }
00097 //
00098 //
00099 // バッファをコピーする
00100 //
00101 void Buffer::copy(const Buffer& buffer) noexcept
00102 {
00103 // コピー元と同じサイズの空のバッファを作る
00104 Buffer::create(buffer.bufferSize[0], buffer.bufferSize[1],
00105 buffer.bufferChannels);
00106 //
00107 // 作成されたバッファにコピーする
00108 copyBuffer(buffer);
00109 }
00110 //
00111 //
00112 // 代入演算子
00113 //
00114 Buffer& Buffer::operator=(const Buffer& buffer)
00115 {
00116 // 代入元と代入先が同じでなければ
00117 if (&buffer != this)
00118 {
00119 // 引数のバッファをバッファをこのバッファにコピーする
00120 Buffer::copy(buffer);
00121 }
00122 //
00123 // このバッファを返す
00124 return *this;
00125 }
00126 //
00127 //
00128 // ムーブ代入演算子
00129 //
00130 Buffer& Buffer::operator=(Buffer&& buffer) noexcept
00131 {
00132 // 代入元と代入先が同じでなければ
00133 if (&buffer != this)
00134 {
00135 // 引数のバッファをバッファをこのバッファにコピーする
00136 Buffer::copy(buffer);
00137 //
00138 // 引数のバッファを破棄する
00139 buffer.Buffer::discard();
00140 }
00141 //
00142 // このバッファを返す
00143 return *this;
00144 }
00145 //
00146 //
00147 // ピクセルバッファオブジェクトをマップする
00148 //
00149 GLvoid* Buffer::map()
00150 #if defined(USE_PIXEL_BUFFER_OBJECT)
00151 const
00152 #endif
00153 {
00154 #if defined(USE_PIXEL_BUFFER_OBJECT)
00155 // ピクセルバッファオブジェクトを参照する
00156 glBindBuffer(GL_PIXEL_PACK_BUFFER, bufferName);
00157 //
00158 // ピクセルバッファオブジェクトをマップする
00159 return glMapBuffer(GL_PIXEL_PACK_BUFFER, GL_READ_WRITE);
00160 #
00161 else
00162 // データの格納場所を返す
00163 return bufferName.data();
00164 #
00165 //
00166 //
00167 // ピクセルバッファオブジェクトをアンマップする
00168 //
00169 void Buffer::unmap() const
00170 {
00171 #if defined(USE_PIXEL_BUFFER_OBJECT)
00172 // ピクセルバッファオブジェクトのマップを解除する
00173 glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
00174 // アップロード先のピクセルバッファオブジェクトの結合を解除する
```

```
00176 glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00177 #endif
00178 }
00179
00180 //
00181 // ピクセルバッファオブジェクトのデータを読み出す
00182 //
00183 void Buffer::getData(GLsizei size, GLvoid* data) const
00184 {
00185 #if defined(USE_PIXEL_BUFFER_OBJECT)
00186 // コピー元のピクセルバッファオブジェクトを結合する
00187 glBindBuffer(GL_PIXEL_UNPACK_BUFFER, bufferName);
00188
00189 // コピー元のピクセルバッファオブジェクトのサイズを調べる
00190 GLint readSize;
00191 glGetBufferParameteriv(GL_PIXEL_UNPACK_BUFFER, GL_BUFFER_SIZE, &readSize);
00192
00193 // コピー元とコピー先の小さい方のサイズをコピーする
00194 glGetBufferSubData(GL_PIXEL_UNPACK_BUFFER, 0, std::min(size, readSize), data);
00195
00196 // コピー元のピクセルバッファオブジェクトの結合を解除する
00197 glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
00198 #else
00199 // コピー元とコピー先の小さい方のサイズ
00200 const auto bufferSize{ static_cast<GLsizei>(bufferName.size()) };
00201 if (size > bufferSize) size = bufferSize;
00202
00203 // コピー元のメモリから引数に指定したコピー先の配列にコピーする
00204 memcpy(data, bufferName.data(), size);
00205 #endif
00206 }
00207
00208 //
00209 // ピクセルバッファオブジェクトにデータを転送する
00210 //
00211 void Buffer::setData(GLsizei size, const GLvoid* data)
00212 #if defined(USE_PIXEL_BUFFER_OBJECT)
00213 const
00214 #endif
00215 {
00216 #if defined(USE_PIXEL_BUFFER_OBJECT)
00217 // コピー先のピクセルバッファオブジェクト
00218 glBindBuffer(GL_PIXEL_PACK_BUFFER, bufferName);
00219
00220 // コピー先のピクセルバッファオブジェクトのサイズを調べる
00221 GLint writeSize;
00222 glGetBufferParameteriv(GL_PIXEL_PACK_BUFFER, GL_BUFFER_SIZE, &writeSize);
00223
00224 // コピー元とコピー先の小さい方のサイズをコピーする
00225 glGetBufferSubData(GL_PIXEL_PACK_BUFFER, 0, std::min(size, writeSize), data);
00226
00227 // コピー先のピクセルバッファオブジェクトの結合を解除する
00228 glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00229 #else
00230 // コピー元とコピー先の小さい方のサイズ
00231 const auto bufferSize{ static_cast<GLsizei>(bufferName.size()) };
00232 if (size > bufferSize) size = bufferSize;
00233
00234 // 引数に指定したコピー元の配列からコピー先のメモリにコピーする
00235 memcpy(bufferName.data(), data, size);
00236 #endif
00237 }
00238
00239 //
00240 // バッファを破棄する
00241 //
00242 void Buffer::discard()
00243 {
00244 #if defined(USE_PIXEL_BUFFER_OBJECT)
00245 // ピクセルバッファオブジェクトの結合を解除する
00246 glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00247 glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
00248
00249 // ピクセルバッファオブジェクトを削除する
00250 glDeleteBuffers(1, &bufferName);
00251 bufferName = 0;
00252 #else
00253 // メモリを消去する
00254 bufferName.clear();
00255 #endif
00256
00257 // バッファのサイズを 0 にする
00258 bufferSize = std::array<int, 2>{ 0, 0 };
00259 bufferChannels = 0;
00260 }
```

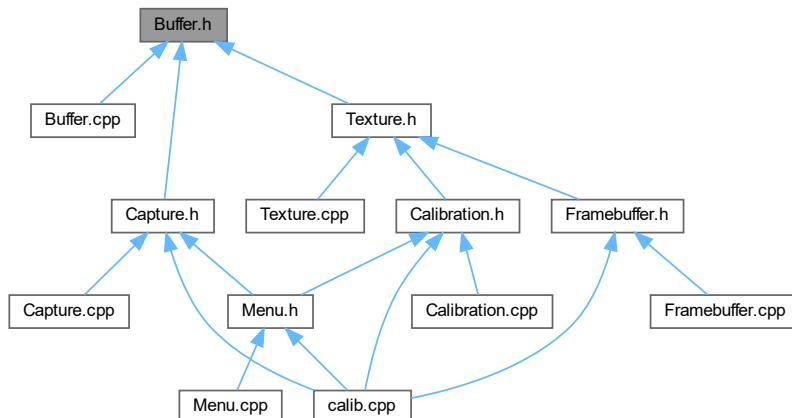
### 9.3 Buffer.h ファイル

```
#include "gg.h"
```

Buffer.h の依存先関係図:



被依存関係図:



#### クラス

- class **Buffer**

#### マクロ定義

- `#define USE_PIXEL_BUFFER_OBJECT`

### 9.3.1 詳解

バッファクラスの定義

著者

Kohe Tokoi

日付

April 3, 2023

[Buffer.h](#) に定義があります。

### 9.3.2 マクロ定義詳解

#### 9.3.2.1 USE\_PIXEL\_BUFFER\_OBJECT

```
#define USE_PIXEL_BUFFER_OBJECT
```

[Buffer.h](#) の 16 行目に定義があります。

## 9.4 Buffer.h

### [詳解]

```
00001 #pragma once
00002
00010
00011 // 補助プログラム
00012 #include "gg.h"
00013 using namespace gg;
00014
00015 // ピクセルバッファオブジェクトを使うとき
00016 #define USE_PIXEL_BUFFER_OBJECT
00017
00024 class Buffer
00025 {
00027     std::array<int, 2> bufferSize;
00028     int bufferChannels;
00031
00032 #if defined(USE_PIXEL_BUFFER_OBJECT)
00034     GLuint
00035 #else
00037     std::vector<GLubyte>
00038 #endif
00039     bufferName;
00040
00041 protected:
00042
00049     auto channelsToFormat(int channels) const
00050     {
00051         // OpenGL のテクスチャフォーマットのリスト
00052         static constexpr GLenum toFormat[] { GL_RED, GL_RG, GL_RGB, GL_RGBA };
00053
00054         // フォーマットを返す
00055         return toFormat[(channels - 1) & 3];
00056     }
00057 }
```

```
00064     int formatToChannels(GLenum format) const;
00065
00071     void copyBuffer(const Buffer& buffer) noexcept;
00072
00073 public:
00074
00078     Buffer()
00079     : bufferSize{ 0, 0 }
00080     , bufferChannels{ 0 }
00081 #if defined(USE_PIXEL_BUFFER_OBJECT)
00082     , bufferName{ 0 }
00083 #endif
00084     {
00085     }
00086
00098     Buffer(GLsizei width, GLsizei height, int channels)
00099     {
00100         Buffer::create(width, height, channels);
00101     }
00102
00108     Buffer(const Buffer& buffer)
00109     {
00110         Buffer::copy(buffer);
00111     }
00112
00118     Buffer(Buffer&& buffer) noexcept
00119     {
00120         *this = std::move(buffer);
00121     }
00122
00126     virtual ~Buffer()
00127     {
00128         Buffer::discard();
00129     }
00130
00137     Buffer& operator=(const Buffer& buffer);
00138
00145     Buffer& operator=(Buffer&& buffer) noexcept;
00146
00158     virtual void create(GLsizei width, GLsizei height, int channels);
00159
00171     void resize(GLsizei width, GLsizei height, int channels)
00172     {
00173         create(width, height, channels);
00174     }
00175
00186     void resize(const Buffer& buffer)
00187     {
00188         create(buffer.getWidth(), buffer.getHeight(), buffer.getChannels());
00189     }
00190
00200     virtual void copy(const Buffer& buffer) noexcept;
00201
00205     virtual void discard();
00206
00210 #if defined(USE_PIXEL_BUFFER_OBJECT)
00211     auto getBufferName() const
00212 #else
00213     auto& getBufferName()
00214 #endif
00215     {
00216         return bufferName;
00217     }
00218
00222     virtual const std::array<GLsizei, 2>& getSize() const
00223     {
00224         return bufferSize;
00225     }
00226
00230     GLsizei getWidth() const
00231     {
00232         return getSize()[0];
00233     }
00234
00238     GLsizei getHeight() const
00239     {
00240         return getSize()[1];
00241     }
00242
00246     virtual int getChannels() const
00247     {
00248         return bufferChannels;
00249     }
00250
00254     auto getFormat() const
00255     {
00256         return channelsToFormat(getChannels());
```

```

00257 }
00258
00262 auto getAspect() const
00263 {
00264     return static_cast<GLfloat>(getWidth()) / static_cast<GLfloat>(getHeight());
00265 }
00266
00270 void bindBuffer(GLenum target = GL_PIXEL_PACK_BUFFER) const
00271 {
00272 #if defined(USE_PIXEL_BUFFER_OBJECT)
00273     glBindBuffer(target, bufferName);
00274 #endif
00275 }
00276
00280 void unbindBuffer(GLenum target = GL_PIXEL_PACK_BUFFER) const
00281 {
00282 #if defined(USE_PIXEL_BUFFER_OBJECT)
00283     glBindBuffer(target, 0);
00284 #endif
00285 }
00286
00292 GLvoid* map()
00293 #if defined(USE_PIXEL_BUFFER_OBJECT)
00294     const
00295 #endif
00296 ;
00297
00301 void unmap() const;
00302
00309 void getData(GLsizei size, GLvoid* data) const;
00310
00317 void setData(GLsizei size, const GLvoid* data)
00318 #if defined(USE_PIXEL_BUFFER_OBJECT)
00319     const
00320 #endif
00321 ;
00322 };

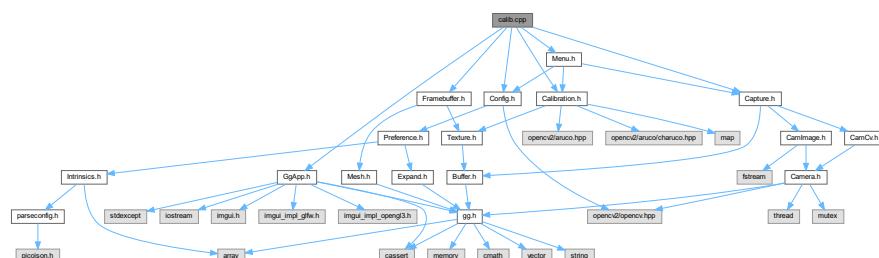
```

## 9.5 calib.cpp ファイル

```

#include "GgApp.h"
#include "Config.h"
#include "Capture.h"
#include "Calibration.h"
#include "Menu.h"
#include "Framebuffer.h"
calib.cpp の依存先関係図:

```



## マクロ定義

- `#define CONFIG_FILE PROJECT_NAME ".config.json"`

### 9.5.1 詳解

ChArUco Board によるカメラキャリレーション

著者

Kohe Tokoi

日付

March 6, 2024

calib.cpp に定義があります。

### 9.5.2 マクロ定義詳解

#### 9.5.2.1 CONFIG\_FILE

```
#define CONFIG_FILE PROJECT_NAME ".config.json"
```

calib.cpp の 28 行目に定義があります。

## 9.6 calib.cpp

### [詳解]

```
00001
00008
00009 // ウィンドウ関連の処理
00010 #include "GgApp.h"
00011
00012 // 構成データ
00013 #include "Config.h"
00014
00015 // キャプチャデバイス
00016 #include "Capture.h"
00017
00018 // 較正
00019 #include "Calibration.h"
00020
00021 // メニュー
00022 #include "Menu.h"
00023
00024 // フレームバッファオブジェクト
00025 #include "Framebuffer.h"
00026
00027 // 構成ファイル名
00028 #define CONFIG_FILE PROJECT_NAME ".config.json"
00029
00030 //
00031 // アプリケーション本体
00032 //
00033 int GgApp::main(int argc, const char* const* argv)
00034 {
00035     // 構成ファイルを読み込む
00036     Config config{ CONFIG_FILE };
00037
00038     // 構成にもとづいてウィンドウを作成する
00039     GgApp::Window window{ config.getTitle(), config.getWidth(), config.getHeight() };
00040 }
```

```

00041 // 開いたウィンドウに対して初期化処理を実行する
00042 config.initialize();
00043
00044 // キャプチャデバイスを作る
00045 Capture capture;
00046
00047 // 軸正オブジェクトを作成する
00048 Calibration calibration{ config.getDictionaryName(), config.getCheckerLength() };
00049
00050 // メニューを作る
00051 Menu menu{ config, capture, calibration };
00052
00053 // キャプチャデバイスで初期画像を開く
00054 capture.openImage(config.getInitialImage());
00055
00056 // 解像度と画角の調整値の初期値を初期画像に合わせる
00057 menu.setSize(capture.getSize());
00058
00059 // キャプチャしたフレームを保持するテクスチャ
00060 Texture frame;
00061
00062 // 画像の展開に用いるフレームバッファオブジェクトのサイズを初期ウィンドウに合わせる
00063 Framebuffer framebuffer{ config.getWidth(), config.getHeight() };
00064
00065 // ウィンドウが開いている間繰り返す
00066 while (window && menu)
00067 {
00068     // メニューを表示して設定を更新する
00069     menu.draw();
00070
00071     // 選択しているキャプチャデバイスから1フレーム取得する
00072     capture.retrieve(frame);
00073
00074     // ピクセルバッファオブジェクトの内容をテクスチャに転送する
00075     frame.drawPixels();
00076
00077     // フレームバッファオブジェクトのサイズをキャプチャしたフレームに合わせる
00078     //framebuffer.resize(frame);
00079
00080     // シェーダの設定を行う
00081     const auto&& size{ menu.setup(framebuffer.getAspect()) };
00082
00083     // フレームバッファオブジェクトにフレームを展開する
00084     framebuffer.update(size, frame);
00085
00086     // ArUco Marker を検出するなら
00087     if (menu.detectMarker)
00088     {
00089         // フレームバッファオブジェクトの内容をピクセルバッファオブジェクトに転送する
00090         framebuffer.readPixels();
00091
00092         // ArUco Marker と ArUco Board を検出する
00093         calibration.detect(framebuffer, menu.detectBoard);
00094
00095         // ピクセルバッファオブジェクトの内容をフレームバッファオブジェクトに書き戻す
00096         framebuffer.drawPixels();
00097     }
00098
00099     // フレームバッファオブジェクトの内容を表示する
00100     framebuffer.draw(window.getWidth(), window.getHeight() - menu.getMenubarHeight());
00101
00102     // カラーバッファを入れ替えてイベントを取り出す
00103     window.swapBuffers();
00104 }
00105
00106 return 0;
00107 }

```

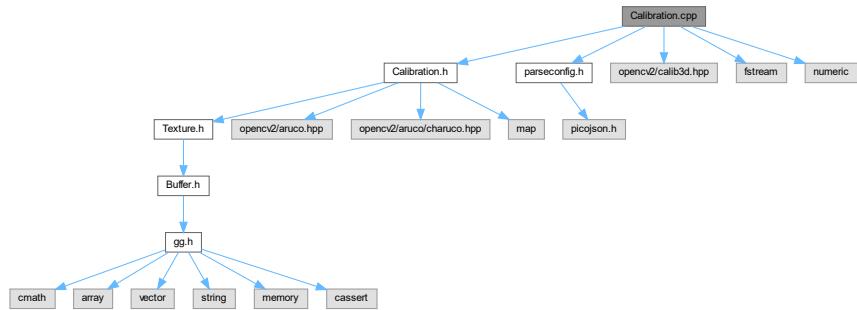
## 9.7 Calibration.cpp ファイル

```

#include "Calibration.h"
#include "parseconfig.h"
#include <opencv2/calib3d.hpp>
#include <fstream>
#include <numeric>

```

Calibration.cpp の依存先関係図:



### 9.7.1 詳解

較正用フレームバッファオブジェクトクラスの実装

著者

Kohe Tokoi

日付

February 20, 2024

Calibration.cpp に定義があります。

## 9.8 Calibration.cpp

### [詳解]

```

00001
00002 #include "Calibration.h"
00003
00004 // 構成ファイルの読み取り補助
00005 #include "parseconfig.h"
00006
00007 // OpenCV
00008 #include <opencv2/calib3d.hpp>
00009
00010 // 標準ライブラリ
00011 #include <fstream>
00012 #include <numeric>
00013
00014 // コンストラクタ
00015
00016 // コンストラクタ
00017
00018 // デストラクタ
00019
00020 // デストラクタ
00021
00022 // デストラクタ
00023 Calibration::Calibration(const std::string& dictionaryName, const std::array<float, 2>& length)
00024     : size{ 0, 0 }
00025     , repError{ 0.0 }
00026 {
00027     // ArUco Marker の辞書を選択する
00028     setDictionary(dictionaryName, length);
00029 }
00030
00031 // デストラクタ
00032 // デストラクタ
00033 // デストラクタ
00034 Calibration::~Calibration()
  
```

```
00035 {  
00036 }  
00037  
00038 //  
00039 // ChArUco Board を作成する  
00040 //  
00041 void Calibration::createBoard(const std::array<float, 2>& length)  
00042 {  
00043     // キャリブレーション用の ChArUco Board を作成する  
00044     board = new cv::aruco::CharucoBoard(cv::Size{ 10, 7 },  
00045         length[0] * 0.01f, length[1] * 0.01f, dictionary);  
00046  
00047     // キャリブレーション用の ChArUco Board の検出器を作成する  
00048     boardDetector = new cv::aruco::CharucoDetector(*board);  
00049 }  
00050  
00051 //  
00052 // ArUco Marker の辞書と検出器を設定する  
00053 //  
00054 void Calibration::setDictionary(const std::string& dictionaryName, const std::array<float, 2>& length)  
00055 {  
00056     // ArUco Marker の辞書を検索する  
00057     auto dictionaryItem{ dictionaryList.find(dictionaryName) };  
00058  
00059     // ArUco Marker の辞書が見つからなかったら辞書リストの最初の辞書を使う  
00060     if (dictionaryItem == dictionaryList.end()) dictionaryItem = dictionaryList.begin();  
00061  
00062     // ArUco Marker の辞書を設定する  
00063     dictionary = cv::aruco::getPredefinedDictionary(dictionaryItem->second);  
00064  
00065     // ArUco Marker の検出器を作成する  
00066     cv::aruco::DetectorParameters detectorParams = cv::aruco::DetectorParameters();  
00067     detector = new cv::aruco::ArucoDetector(dictionary, detectorParams);  
00068  
00069     // キャリブレーション用の ChArUco Board を作成する  
00070     createBoard(length);  
00071 }  
00072  
00073 //  
00074 // ChArUco Board を描く  
00075 //  
00076 void Calibration::drawBoard(cv::Mat& boardImage, int width, int height)  
00077 {  
00078     boardDetector->getBoard().generateImage(cv::Size{ width, height }, boardImage, 10, 1);  
00079 }  
00080  
00081 //  
00082 // ArUco Marker を検出する  
00083 //  
00084 bool Calibration::detect(Buffer& buffer, bool detectBoard)  
00085 {  
00086     // 入力画像のサイズを記録しておく  
00087     size = cv::Size{ buffer.getWidth(), buffer.getHeight() };  
00088  
00089     // ピクセルバッファオブジェクトを CPU のメモリ空間にマップする  
00090     cv::Mat image{ size, CV_8UC(buffer.getChannels()), buffer.map() };  
00091  
00092     // ArUco Marker のコーナーを検出する  
00093     detector->detectMarkers(image, corners, ids, rejected);  
00094  
00095     // 調整が完了していれば  
00096     if (finished())  
00097     {  
00098         std::vector<cv::Vec3d> rvecs, tvecs;  
00099  
00100         // ChArUco Board 上の ArUco Marker の一辺の長さ  
00101         const auto markerLength{ board->getMarkerLength() };  
00102  
00103         // 全てのマーカの姿勢を推定して  
00104         cv::aruco::estimatePoseSingleMarkers(corners, markerLength,  
00105             cameraMatrix, distCoeffs, rvecs, tvecs);  
00106  
00107         // 各々のマーカについて  
00108         for (size_t i = 0; i < rvecs.size(); ++i)  
00109         {  
00110             // 座標軸を描く  
00111             cv::drawFrameAxes(image, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], markerLength);  
00112         }  
00113     }  
00114     else  
00115     {  
00116         // 検出結果をピクセルバッファオブジェクトに書き込む  
00117         cv::aruco::drawDetectedMarkers(image, corners, ids);  
00118     }  
00119  
00120     // ChArUco Board の検出を行っているのなら  
00121     if (detectBoard)
```

```

00123 {
00124     // 検出された ArUco Marker のコーナーと ChArUco Board のレイアウトを使って残りのコーナーを再検出する
00125     detector->refineDetectedMarkers(image, *board, corners, ids, rejected);
00126
00127     // ArUco Marker が検出されたら
00128     if (!corners.empty())
00129     {
00130         // ChArUco Board のコーナーを検出する
00131         cv::aruco::interpolateCornersCharuco(corners, ids, image, board, charucoCorners, charucoIds);
00132         //boardDetector->detectBoard(image, currentCharucoCorners, currentCharucoIds);
00133         //boardDetector->getBoard().matchImagePoints(
00134         //    currentCharucoCorners, currentCharucoIds,
00135         //    currentObjectPoints, currentImagePoints
00136         //);
00137
00138         // ChArUco Board の角の位置を表示に描き込む
00139         cv::aruco::drawDetectedCornersCharuco(image, charucoCorners, charucoIds, cv::Scalar(0, 0, 255));
00140     }
00141 }
00142
00143     // ピクセルバッファオブジェクトのマップを解除する
00144 buffer.unmap();
00145
00146     // マーカが見つかれば true を返す
00147     return !corners.empty();
00148 }
00149
00150 // 
00151 // 標本を取得する
00152 //
00153 void Calibration::recordCorners()
00154 {
00155     // ChArUco Board の角が4つ以上見つかれば
00156     if (charucoCorners.total() >= 4)
00157     {
00158         // ChArUco Board の角を記録する
00159         allCorners.push_back(charucoCorners);
00160         allIds.push_back(charucoIds);
00161     }
00162 #if defined(DEBUG)
00163     std::cerr << "charucoCorners = " << charucoCorners.total()
00164     << ", allCorners = " << allCorners.size() << "\n";
00165 #endif
00166
00167     // ChArUco Board の角を破棄する
00168     charucoCorners.release();
00169     charucoIds.release();
00170 }
00171
00172 //
00173 // 標本を消去する
00174 //
00175 void Calibration::discardCorners()
00176 {
00177     // 記録した標本を消去する
00178     allCorners.clear();
00179     allIds.clear();
00180
00181     // 較正の計算結果を消去する
00182     cameraMatrix.release();
00183     distCoeffs.release();
00184 }
00185
00186 //
00187 // 較正する
00188 //
00189 void Calibration::calibrate()
00190 {
00191     // 再投影誤差はとりあえず 0にしておく
00192     repError = 0.0f;
00193
00194     // 交点を合計6つ以上検出できていれば
00195     if (allCorners.size() >= 6) try
00196     {
00197         // ChArUco Board の姿勢
00198         std::vector<cv::Mat> boardRvecs, boardTvecs;
00199
00200         // 取得した全ての交点からカメラパラメータを推定する
00201         repError = cv::aruco::calibrateCameraCharuco(allCorners, allIds, board, size,
00202             cameraMatrix, distCoeffs, boardRvecs, boardTvecs);
00203
00204         //int calibrationFlags = 0
00205         // //| cv::CALIB_USE_INTRINSIC_GUESS // cameraMatrix contains valid initial values of fx, fy, cx,
00206         // cy that are optimized further.Otherwise, (cx, cy) is initially set to the image center(imageSize is
00207         // used), and focal distances are computed in a least - squares fashion.Note, that if intrinsic
00208         // parameters are known, there is no need to use this function just to estimate extrinsic parameters.Use
00209         // solvePnP instead.

```

```

00206 // /// cv::CALIB_FIX_PRINCIPAL_POINT // The principal point is not changed during the global
00207 optimization. It stays at the center or at a different location specified when
00208 CALIB_USE_INTRINSIC_GUESS is set too.
00209 // /// cv::CALIB_FIX_ASPECT_RATIO // The functions consider only fy as a free parameter. The
ratio fx / fy stays the same as in the input cameraMatrix. When CALIB_USE_INTRINSIC_GUESS is not set,
the actual input values of fx and fy are ignored, only their ratio is computed and used further.
00210 // /// cv::CALIB_ZERO_TANGENT_DIST // Tangential distortion coefficients(p1, p2) are set to
zeros and stay zero.
00211 // /// cv::CALIB_FIX_FOCAL_LENGTH // The focal length is not changed during the global
optimization if CALIB_USE_INTRINSIC_GUESS is set.
00212 // /// cv::CALIB_FIX_K1 // , ..., CALIB_FIX_K6 The corresponding radial distortion
coefficient is not changed during the optimization. If CALIB_USE_INTRINSIC_GUESS is set, the
coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
00213 // /// cv::CALIB_RATIONAL_MODEL // Coefficients k4, k5, and k6 are enabled. To provide the
backward compatibility, this extra flag should be explicitly specified to make the calibration
function use the rational model and return 8 coefficients or more.
00214 // /// cv::CALIB_THIN_PRISM_MODEL // Coefficients s1, s2, s3 and s4 are enabled. To provide the
backward compatibility, this extra flag should be explicitly specified to make the calibration
function use the thin prism model and return 12 coefficients or more.
00215 // /// cv::CALIB_FIX_TAUX_TAUY // The coefficients of the tilted sensor model are not
changed during the optimization. If CALIB_USE_INTRINSIC_GUESS is set, the coefficient from the supplied
distCoeffs matrix is used. Otherwise, it is set to 0.
00216 //
00217 //repError = cv::calibrateCamera(
00218 // allObjectPoints, allImagePoints, getSize(),
00219 // cameraMatrix, distCoeffs, boardRvecs, boardRvecs, cv::noArray(),
00220 // cv::noArray(), cv::noArray(), calibrationFlags);
00221 }
00222 catch (const cv::Exception&)
00223 {
00224 // 収束しなかった場合は計算結果を捨てる
00225 cameraMatrix.release();
00226 distCoeffs.release();
00227 }
00228 }
00229
00230 //
00231 // ArUco Marker の3次元姿勢の変換行列を求める
00232 //
00233 void Calibration::getAllMarkerPoses(std::map<int, GgMatrix>& poses, float markerLength)
00234 {
00235 std::vector<cv::Vec3d> rvecs, tvecs;
00236
00237 // 全てのマーカの姿勢を推定して
00238 cv::aruco::estimatePoseSingleMarkers(corners, markerLength,
00239 cameraMatrix, distCoeffs, rvecs, tvecs);
00240
00241 // 各々のマーカについて
00242 for (size_t i = 0; i < rvecs.size(); ++i)
00243 {
00244 // 回転角を求める
00245 const auto r{ cv::norm(rvecs[i]) };
00246
00247 // 回転軸ベクトルを正規化する
00248 rvecs[i] /= r;
00249
00250 // 回転軸ベクトル
00251 const auto rx{ static_cast<GLfloat>(rvecs[i][0]) };
00252 const auto ry{ static_cast<GLfloat>(rvecs[i][1]) };
00253 const auto rz{ static_cast<GLfloat>(rvecs[i][2]) };
00254
00255 // 平行移動量
00256 const auto tx{ static_cast<GLfloat>(tvecs[i][0]) };
00257 const auto ty{ static_cast<GLfloat>(tvecs[i][1]) };
00258 const auto tz{ static_cast<GLfloat>(tvecs[i][2]) };
00259
00260 // 各マーカの姿勢を求める
00261 poses[ids[i]] = ggTranslate(tx, ty, tz).rotate(rx, ry, rz, static_cast<GLfloat>(r));
00262
00263 }
00264 }
00265
00266 //
00267 // カメラパラメータの JSON オブジェクトから数値の配列を取得する
00268 //
00269 static bool getMatrix(const picojson::object& object,
00270 const std::string& key, cv::Mat& mat, int cols, int rows)
00271 {
00272 // key に一致するオブジェクトを探す
00273 const auto& value{ object.find(key) };
00274
00275 // オブジェクトが無いか配列でなかったら戻る

```

```
00276 if (value == object.end() || !value->second.is<picojson::array>()) return false;
00277 // 配列を取り出す
00278 const auto& array{ value->second.get<picojson::array>() };
00279 // 配列の要素数とデータの格納先の行列の要素数が一致していなければ戻る
00280 if (static_cast<size_t>(cols) * rows != array.size()) return false;
00281 // カメラ行列の要素を確保する
00282 mat = cv::Mat::zeros(rows, cols, CV_64F);
00283 // 配列の要素について
00284 for (size_t i = 0; i < array.size(); ++i)
00285 {
00286     // 行列の要素の位置
00287     const auto x{ static_cast<int>(i % cols) };
00288     const auto y{ static_cast<int>(i / cols) };
00289     // 要素が数値なら格納する
00290     if (array[i].is<double>()) mat.at<double>(y, x) = array[i].get<double>();
00291 }
00292 return true;
00293 }
00300
00301 //
00302 // カメラパラメータの JSON オブジェクトから数値の配列を取得する
00303 //
00304 static void setMatrix(picojson::object& object,
00305 const std::string& key, const cv::Mat& mat)
00306 {
00307     // picojson の配列
00308     picojson::array array;
00309     // 配列の要素について
00310     for (size_t i = 0; i < mat.total(); ++i)
00311     {
00312         // 行列の要素の位置
00313         const auto x{ static_cast<int>(i % mat.cols) };
00314         const auto y{ static_cast<int>(i / mat.cols) };
00315         // 要素を picojson::array に追加する
00316         array.emplace_back(picojson::value(mat.at<double>(y, x)));
00317     }
00318     // オブジェクトに追加する
00319     object.emplace(key, array);
00320 }
00321
00322 // ファイルからキャリブレーションパラメータを読み込む
00323 //
00324 00325 bool Calibration::loadParameters(const std::string& filename)
00326 {
00327     // パラメタファイルの読み込み
00328     std::ifstream json{ filename };
00329     if (!json) return false;
00330     // JSON の読み込み
00331     picojson::value value;
00332     json >> value;
00333     json.close();
00334     // 構成内容の取り出し
00335     const auto& object{ value.get<picojson::object>() };
00336     // オブジェクトが空だったらエラー
00337     if (object.empty()) return false;
00338     // カメラ行列
00339     if (!getMatrix(object, "camera matrix", cameraMatrix, 3, 3)) return false;
00340     // 歪み定数
00341     if (!getMatrix(object, "distortion", distCoeffs, 5, 1)) return false;
00342     // 再投影誤差
00343     getValue(object, "error", repError);
00344     // キャリブレーションパラメータの読み込み
00345     return true;
00346 }
00347
00348 // キャリブレーションパラメータをファイルに保存する
00349 //
00350 00351 bool Calibration::saveParameters(const std::string& filename) const
00352 {
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362 }
```

```

00363 // 設定値を保存する
00364 std::ofstream config{ filename };
00365 if (!config) return false;
00366
00367 // オブジェクト
00368 picojson::object object;
00369
00370 // カメラ行列
00371 setMatrix(object, "camera matrix", cameraMatrix);
00372
00373 // 歪み定数
00374 setMatrix(object, "distortion", distCoeffs);
00375
00376 // 再投影誤差
00377 setValue(object, "error", repError);
00378
00379 // 構成をシリализして保存
00380 picojson::value v{ object };
00381 config << v.serialize(true);
00382 config.close();
00383
00384 // キャリブレーションパラメータの書き込み
00385 return true;
00386 }
00387
00388 // ArUco Marker 辞書のリスト
00389 const std::map<const std::string, const cv::aruco::PredefinedDictionaryType>
    Calibration::dictionaryList
00390 {
00391     { "DICT_4X4_50", cv::aruco::DICT_4X4_50 },
00392     { "DICT_4X4_100", cv::aruco::DICT_4X4_100 },
00393     { "DICT_4X4_250", cv::aruco::DICT_4X4_250 },
00394     { "DICT_4X4_1000", cv::aruco::DICT_4X4_1000 },
00395     { "DICT_5X5_50", cv::aruco::DICT_5X5_50 },
00396     { "DICT_5X5_100", cv::aruco::DICT_5X5_100 },
00397     { "DICT_5X5_250", cv::aruco::DICT_5X5_250 },
00398     { "DICT_5X5_1000", cv::aruco::DICT_5X5_1000 },
00399     { "DICT_6X6_50", cv::aruco::DICT_6X6_50 },
00400     { "DICT_6X6_100", cv::aruco::DICT_6X6_100 },
00401     { "DICT_6X6_250", cv::aruco::DICT_6X6_250 },
00402     { "DICT_6X6_1000", cv::aruco::DICT_6X6_1000 },
00403     { "DICT_7X7_50", cv::aruco::DICT_7X7_50 },
00404     { "DICT_7X7_100", cv::aruco::DICT_7X7_100 },
00405     { "DICT_7X7_250", cv::aruco::DICT_7X7_250 },
00406     { "DICT_7X7_1000", cv::aruco::DICT_7X7_1000 },
00407     { "DICT_ARUCO_ORIGINAL", cv::aruco::DICT_ARUCO_ORIGINAL },
00408     { "DICT_APRILTAG_16h5", cv::aruco::DICT_APRILTAG_16h5 },
00409     { "DICT_APRILTAG_25h9", cv::aruco::DICT_APRILTAG_25h9 },
00410     { "DICT_APRILTAG_36h10", cv::aruco::DICT_APRILTAG_36h10 },
00411     { "DICT_APRILTAG_36h11", cv::aruco::DICT_APRILTAG_36h11 }
00412 };

```

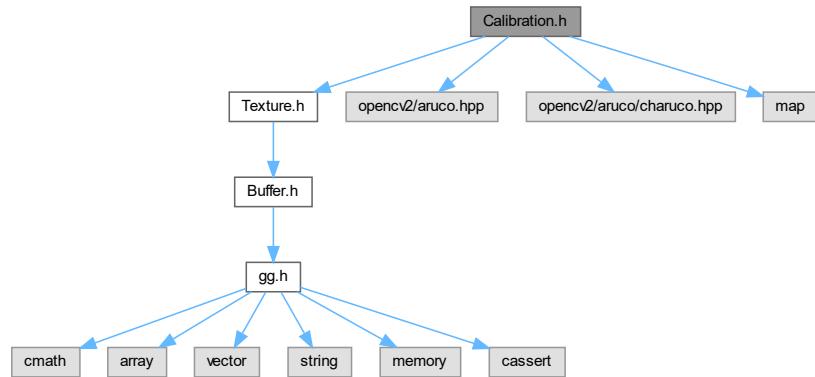
## 9.9 Calibration.h ファイル

```

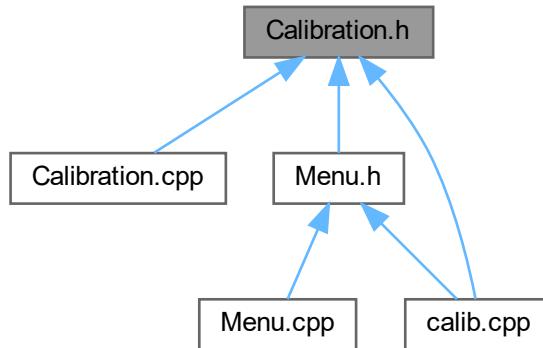
#include "Texture.h"
#include <opencv2/aruco.hpp>
#include <opencv2/aruco/charuco.hpp>
#include <map>

```

Calibration.h の依存先関係図:



被依存関係図:



クラス

- class [Calibration](#)

### 9.9.1 詳解

較正クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Calibration.h](#) に定義があります。

## 9.10 Calibration.h

### [詳解]

```

00001 #pragma once
00002
00010
00011 // テクスチャ
00012 #include "Texture.h"
00013
00014 // ArUco Maker
00015 #include <opencv2/aruco.hpp>
00016
00017 // ChArUco Board
00018 #include <opencv2/aruco/charuco.hpp>
00019
00020 // 標準ライブラリ
00021 #include <map>
00022
00026 class Calibration
00027 {
00029     cv::Size size;
00030
00032     cv::aruco::Dictionary dictionary;
00033
00035     cv::Ptr<cv::aruco::ArucoDetector> detector;
00036
00038     cv::Ptr<cv::aruco::CharucoBoard> board;
00039
00041     cv::Ptr<cv::aruco::CharucoDetector> boardDetector;
00042
00044     std::vector<std::vector<cv::Point2f>> corners, rejected;
00045     std::vector<int> ids;
00046
00048     cv::Mat charucoCorners, charucoIds;
00049     //std::vector<cv::Point2f> currentCharucoCorners;
00050     //std::vector<int> currentCharucoIds;
00051     //std::vector<cv::Point3f> currentObjectPoints;
00052     //std::vector<cv::Point2f> currentImagePoints;
00053
00055     std::vector<cv::Mat> allCorners, allIds;
00056     //std::vector<std::vector<cv::Point2f>> allCharucoCorners;
00057     //std::vector<std::vector<int>> allCharucoIds;
00058     //std::vector<std::vector<cv::Point2f>> allImagePoints;
00059     //std::vector<std::vector<cv::Point3f>> allObjectPoints;
00060
00062     cv::Mat cameraMatrix;
00063
00065     cv::Mat distCoeffs;
00066
00068     double repError;
00069
00070 public:
00071
00078     Calibration(const std::string& dictionaryName, const std::array<float, 2>& length);
00079
00085     Calibration(const Calibration& calibration) = delete;
00086
00090     virtual ~Calibration();
00091
00097     Calibration& operator=(const Calibration& calibration) = delete;
00098
00104     void createBoard(const std::array<float, 2>& length);
00105
00112     void setDictionary(const std::string& dictionaryName, const std::array<float, 2>& length);
00113
00121     void drawBoard(cv::Mat& boardImage, int width, int height);
00122
00130     bool detect(Buffer& buffer, bool detectBoard);
00131
00137     const auto get CornersCount() const
00138     {
00139         return static_cast<int>(corners.size());
00140     }
00141
00145     void recordCorners();
00146
00152     const auto get CornerCount() const
00153     {
00154         return static_cast<int>(allCorners.size());
00155     }
00156
00160     void discardCorners();
00161
00165     void calibrate();
00166

```

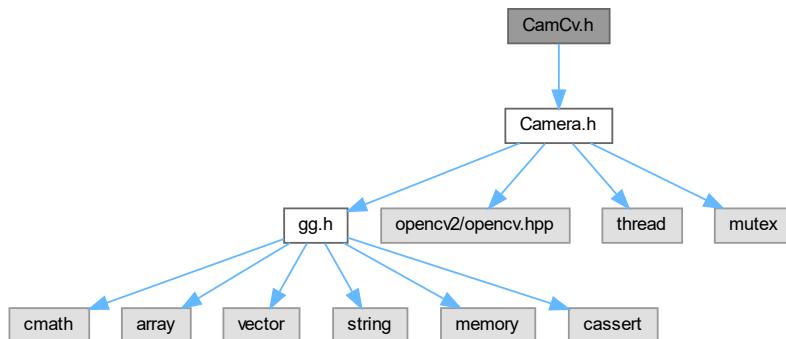
```

00172     const auto& getCameraMatrix() const
00173     {
00174         return cameraMatrix;
00175     }
00176
00182     const auto& getDistortionCoefficients() const
00183     {
00184         return distCoeffs;
00185     }
00186
00192     auto getReprojectionError() const
00193     {
00194         return repError;
00195     }
00196
00202     auto finished()
00203     {
00204         return cameraMatrix.total() == 9 && distCoeffs.total() == 5;
00205     }
00206
00213     void getAllMarkerPoses(std::map<int, GgMatrix>& poses, float markerLength);
00214
00221     bool loadParameters(const std::string& filename);
00222
00229     bool saveParameters(const std::string& filename) const;
00230
00232     static const std::map<const std::string, const cv::aruco::PredefinedDictionaryType> dictionaryList;
00233 };

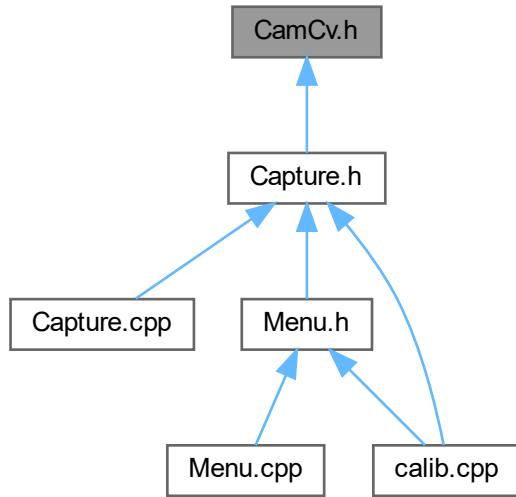
```

## 9.11 CamCv.h ファイル

#include "Camera.h"  
CamCv.h の依存先関係図:



被依存関係図:



クラス

- class [CamCv](#)

### 9.11.1 詳解

OpenCV を使ったビデオキャプチャクラスの定義

著者

Kohe Tokoi

日付

December 27, 2022

[CamCv.h](#) に定義がります。

## 9.12 CamCv.h

### [詳解]

```

00001 #pragma once
00002
00010
00011 // カメラ関連の処理
00012 #include "Camera.h"
00013
00017 class CamCv : public Camera
00018 {
00019     cv::VideoCapture camera;
00020
00023     double elapsedTime;
00024
00026     int exposure, gain;
00027
00037     bool init(int initialWidth, int initialHeight, double initialFPS, const char* fourcc = "")
00038     {
00039         // カメラのコーデック・解像度・フレームレートを設定する
00040         if (fourcc[0] != '\0') camera.set(cv::CAP_PROP_FOURCC,
00041             cv::VideoWriter::fourcc(fourcc[0], fourcc[1], fourcc[2], fourcc[3]));
00042         if (initialWidth > 0) camera.set(cv::CAP_PROP_FRAME_WIDTH, initialWidth);
00043         if (initialHeight > 0) camera.set(cv::CAP_PROP_FRAME_HEIGHT, initialHeight);
00044         if (initialFPS > 0.0) camera.set(cv::CAP_PROP_FPS, initialFPS);
00045
00046         // FPS が 0 なら逆数をカメラの遅延に使う
00047         const auto fps{ camera.get(cv::CAP_PROP_FPS) };
00048         if (fps > 0.0) interval = 1000.0 / fps;
00049
00050         // ムービーファイルのインポント・アウトポイントの初期値とフレーム数
00051         in = camera.get(cv::CAP_PROP_POS_FRAMES);
00052         out = total = camera.get(cv::CAP_PROP_FRAME_COUNT);
00053
00054         // 経過時間
00055         elapsedTime = 0.0;
00056
00057         // カメラから最初のフレームをキャプチャできなかったらカメラは使えない
00058         if (!camera.grab()) return false;
00059
00060         // カメラの利得と露出を取得する
00061         gain = static_cast<GLsizei>(camera.get(cv::CAP_PROP_GAIN));
00062         exposure = static_cast<GLsizei>(camera.get(cv::CAP_PROP_EXPOSURE) * 10.0);
00063
00064         // フレームを取り出してキャプチャ用のメモリを確保する
00065         camera.retrieve(frame);
00066
00067 #if defined(DEBUG)
00068     char codec[5]{ 0, 0, 0, 0, 0 };
00069     getCodec(codec);
00070     std::cerr << "in:" << in << ", out:" << out
00071     << ", width:" << frame.cols << ", height:" << frame.rows
00072     << ", fourcc: " << codec << "\n";
00073 #endif
00074
00075         // 取り出した転送用の一時メモリにデータを格納する
00076         copyFrame();
00077
00078         // フレームがキャプチャされたことを記録する
00079         captured = true;
00080
00081         // カメラが使える
00082         return true;
00083     }
00084
00088     void capture()
00089     {
00090         // 再生開始時刻
00091         auto startTime{ glfwGetTime() };
00092
00093         // スレッドが実行可の間
00094         while (running)
00095         {
00096             // フレームを取り出せたら true
00097             auto status{ (total <= 0.0 || camera.get(cv::CAP_PROP_POS_FRAMES) < out) && camera.grab() };
00098
00099             // ムービーファイルでないかムービーファイルの終端でなければ次のフレームを取り出して
00100             if (status && camera.retrieve(frame))
00101             {
00102                 // ピクセルバッファオブジェクトをロックしてから
00103                 std::lock_guard lock{ mtx };
00104
00105                 // 転送用の一時メモリにデータを格納したら
00106                 copyFrame();
00107

```

```

00108     // 新しいフレームがキャプチャされたことを通知する
00109     captured = true;
00110 }
00111
00112 // 遅延時間
00113 auto deferred{ 0.0 };
00114
00115 // ムービーファイルから入力しているとき
00116 if (total > 0.0)
00117 {
00118     // ムービーファイルの終端に到達していたら
00119     if (!status)
00120     {
00121         // インポイントまで巻き戻す
00122         camera.set(cv::CAP_PROP_POS_FRAMES, in);
00123
00124         // 経過時間を戻す
00125         elapsedTime = 0.0;
00126
00127         // 開始時間を更新する
00128         startTime = glfwGetTime();
00129     }
00130     else
00131     {
00132         // 現在のフレームのインポイントからの経過時間
00133         const auto pos{ camera.get(cv::CAP_PROP_POS_MSEC) - in };
00134
00135         // 再生位置の次のフレームの時刻に対する経過時間
00136         const auto now{ (elapsedTime + glfwGetTime() - startTime) * 1000.0 + interval };
00137
00138         // 遅延時間はフレームの経過時間と現在の経過時間の差
00139         deferred = pos - now;
00140     }
00141 }
00142
00143 // 遅延時間あれば
00144 if (deferred > 0.0)
00145 {
00146     // 待つ
00147     std::this_thread::sleep_for(std::chrono::milliseconds(static_cast<int>(deferred)));
00148 }
00149 }
00150
00151 // 再生停止までの経過時間を積算する
00152 elapsedTime += glfwGetTime() - startTime;
00153 }
00154
00155 public:
00156
00160 CamCv()
00161     : elapsedTime{ 0.0 }
00162     , exposure{ 0 }
00163     , gain{ 0 }
00164 {}
00165
00169 virtual ~CamCv()
00170 {
00171 }
00172
00183 auto open(int device, int width = 0, int height = 0, double fps = 0.0, const char* fourcc = "", int
pref = cv::CAP_ANY)
00184 {
00185     // カメラを開いて初期化する
00186     return camera.open(device, pref) && init(width, height, fps, fourcc);
00187 }
00188
00192 void close()
00193 {
00194     // キャプチャデバイスを開放する
00195     camera.release();
00196
00197     // キャプチャデバイスを閉じる
00198     Camera::close();
00199 }
00200
00211 auto open(const std::string& file, int width = 0, int height = 0, double fps = 0.0, const char*
fourcc = "", int pref = cv::CAP_ANY)
00212 {
00213     // ファイル／ネットワークを開いて初期化する
00214     return camera.open(file, pref) && init(width, height, fps, fourcc);
00215 }
00216
00222 virtual double getFps() const
00223 {
00224     return camera.get(cv::CAP_PROP_FPS);
00225 }
00226

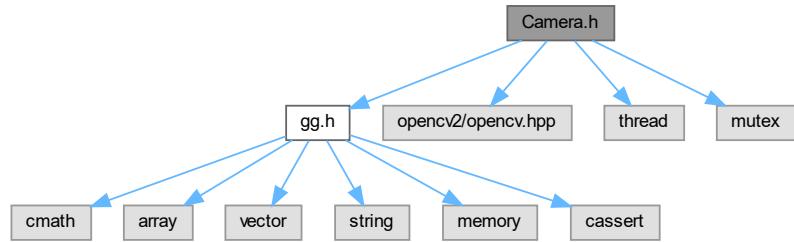
```

```
00232     auto getCodec() const
00233     {
00234         return static_cast<int>(camera.get(cv::CAP_PROP_FOURCC));
00235     }
00236
00242     void getCodec(char* fourcc) const
00243     {
00244         int cc{ getCodec() };
00245         for (int i = 0; i < 4; ++i)
00246         {
00247             fourcc[i] = static_cast<char>(cc & 0x7f);
00248             if (!isalnum(fourcc[i])) fourcc[i] = '?';
00249             cc >>= 8;
00250         }
00251     }
00252
00258     auto getPosition() const
00259     {
00260         return camera.get(cv::CAP_PROP_POS_FRAMES);
00261     }
00262
00268     void setPosition(double frame)
00269     {
00270         camera.set(cv::CAP_PROP_POS_FRAMES, frame);
00271     }
00272
00278     void setExposure(double exposure)
00279     {
00280         if (camera.isOpened()) camera.set(cv::CAP_PROP_EXPOSURE, exposure);
00281     }
00282
00286     void increaseExposure()
00287     {
00288         setExposure(++exposure * 0.1);
00289     }
00290
00294     void decreaseExposure()
00295     {
00296         setExposure(--exposure * 0.1);
00297     }
00298
00304     void setGain(double gain)
00305     {
00306         if (camera.isOpened()) camera.set(cv::CAP_PROP_GAIN, gain);
00307     }
00308
00312     void increaseGain()
00313     {
00314         setGain(++gain);
00315     }
00316
00320     void decreaseGain()
00321     {
00322         setGain(--gain);
00323     }
00324 },
```

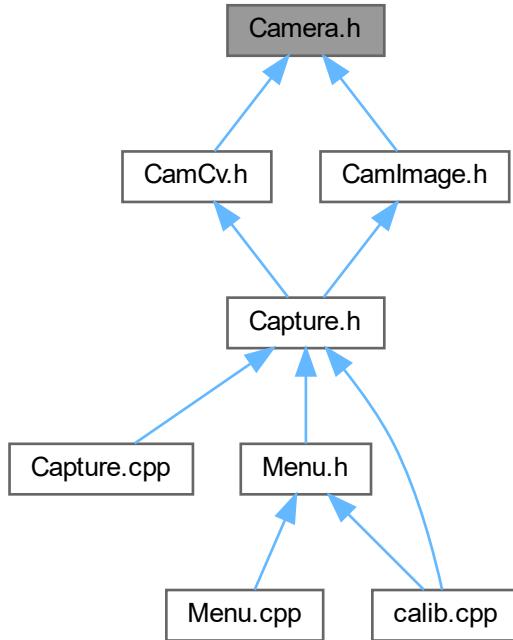
## 9.13 Camera.h ファイル

```
#include "gg.h"
#include <opencv2/opencv.hpp>
#include <thread>
#include <mutex>
```

Camera.h の依存先関係図:



被依存関係図:



## クラス

- class [Camera](#)

### 9.13.1 詳解

キャプチャデバイス関連の基底クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

Camera.h に定義があります。

## 9.14 Camera.h

### [詳解]

```
00001 #pragma once
00002
00010
00011 // 補助プログラム
00012 #include "gg.h"
00013
00014 // OpenCV
00015 #include <opencv2/opencv.hpp>
00016 #if defined(_MSC_VER)
00017 # define CV_VERSION_STR CVAUX_STR(CV_MAJOR_VERSION) CVAUX_STR(CV_MINOR_VERSION)
        CVAUX_STR(CV_SUBMINOR_VERSION)
00018 # if defined(_DEBUG)
00019 #     define CV_EXT_STR "d.lib"
00020 # else
00021 #     define CV_EXT_STR ".lib"
00022 # endif
00023 # pragma comment(lib, "opencv_core" CV_VERSION_STR CV_EXT_STR)
00024 # pragma comment(lib, "opencv_imgproc" CV_VERSION_STR CV_EXT_STR)
00025 # pragma comment(lib, "opencv_imgcodecs" CV_VERSION_STR CV_EXT_STR)
00026 # pragma comment(lib, "opencv_videoio" CV_VERSION_STR CV_EXT_STR)
00027 # pragma comment(lib, "opencv_calib3d" CV_VERSION_STR CV_EXT_STR)
00028 # pragma comment(lib, "opencv_aruco" CV_VERSION_STR CV_EXT_STR)
00029 # pragma comment(lib, "opencv_objdetect" CV_VERSION_STR CV_EXT_STR)
00030 #endif
00031
00032 // 非同期処理
00033 #include <thread>
00034 #include <mutex>
00035
00039 class Camera
00040 {
00041 protected:
00042
00044     double total;
00045
00047     double interval;
00048
00050     int channels;
00051
00053     cv::Mat frame;
00054
00056     std::vector<GLubyte> pixels;
00057
00059     bool captured;
00060
00062     std::thread thr;
00063
00065     std::mutex mtx;
00066
00068     bool running;
00069
00075     void copyFrame()
00076     {
00077         // 取り出したフレームのチャネル数を保存する
00078         channels = frame.channels();
00079
00080         // フレームの大きさを求める
00081         const auto length{ frame.cols * frame.rows * channels };
00082
00083         // 転送用に必要なメモリサイズが以前と違ったらメモリを確保しなおす
00084         if (static_cast<int>(pixels.size()) != length) pixels.resize(length);
00085 }
```

```

00086 // R と B を入れ替えてコピーする
00087 if (channels > 0)
00088 {
00089     if (channels < 3)
00090         std::copy(frame.data, frame.data + length, pixels.data());
00091     else
00092     {
00093         for (int i = 0; i < length; i += channels)
00094         {
00095             pixels.data()[i + 0] = frame.data[i + 2];
00096             pixels.data()[i + 1] = frame.data[i + 1];
00097             pixels.data()[i + 2] = frame.data[i + 0];
00098             if (channels >= 3) pixels.data()[i + 3] = 1;
00099         }
00100     }
00101 }
00102 }
00103
00107 virtual void capture()
00108 {
00109     // 繙承されていなければスレッドを起動しない
00110     stop();
00111 }
00112
00113
00114 public:
00115
00117     double in;
00118
00120     double out;
00121
00125     Camera()
00126         : total{ -1.0 }
00127         , interval{ 10.0 }
00128         , channels{ 3 }
00129         , captured{ false }
00130         , running{ false }
00131         , in{ -1.0 }
00132         , out{ -1.0 }
00133     {
00134     }
00135
00141     Camera(const Camera& camera) = delete;
00142
00146     virtual ~Camera()
00147     {
00148         // キャプチャスレッドを停止する
00149         stop();
00150
00151         // キャプチャデバイスの使用を終了する
00152         close();
00153
00154         // 一時メモリを空にする
00155         pixels.clear();
00156     }
00157
00163     Camera& operator=(const Camera& camera) = delete;
00164
00168     void start()
00169     {
00170         // スレッドが起動状態であることを記録しておく
00171         running = true;
00172
00173         // スレッドを起動する
00174         thr = std::thread([&] { this->capture(); });
00175     }
00176
00180     void stop()
00181     {
00182         // キャプチャスレッドが実行中なら
00183         if (running)
00184         {
00185             // キャプチャスレッドのループを止めて
00186             running = false;
00187
00188             // 合流する
00189             thr.join();
00190         }
00191     }
00198     void transmit(GLuint buffer)
00199     {
00200         // 新しいフレームが取得されているときカメラのロックが成功したら
00201         if (captured && mtx.try_lock())
00202         {
00203             // フレームをピクセルバッファオブジェクトに転送して
00204             glBindBuffer(GL_PIXEL_PACK_BUFFER, buffer);

```

```

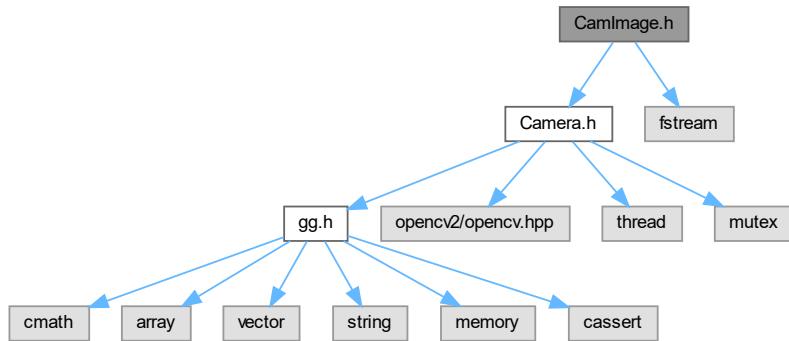
00205     glBindBuffer(GL_PIXEL_PACK_BUFFER, 0, pixels.size(), pixels.data());
00206     glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00207
00208     // 次のフレームの取得を待つ
00209     captured = false;
00210
00211     // キャプチャデバイスのロックを解除する
00212     mtx.unlock();
00213 }
00214
00215 void transmit(decltype(pixels)& buffer)
00216 {
00217     // 新しいフレームが取得されているときカメラのロックが成功したら
00218     if (captured && mtx.try_lock())
00219     {
00220         // フレームをメモリに転送して
00221         const auto size{ std::max(pixels.size(), buffer.size()) };
00222         memcpy(buffer.data(), pixels.data(), size);
00223
00224         // 次のフレームの取得を待つ
00225         captured = false;
00226
00227         // キャプチャデバイスのロックを解除する
00228         mtx.unlock();
00229     }
00230 }
00231
00232 virtual void close()
00233 {
00234     // フレームを取得していないことにする
00235     captured = false;
00236 }
00237
00238 auto isRunning() const
00239 {
00240     return running;
00241 }
00242
00243 std::array<int, 2> getSize() const
00244 {
00245     return std::array<int, 2>{ frame.cols, frame.rows };
00246 }
00247
00248 auto getWidth() const
00249 {
00250     return frame.cols;
00251 }
00252
00253 auto getHeight() const
00254 {
00255     return frame.rows;
00256 }
00257
00258 auto getChannels() const
00259 {
00260     return frame.channels();
00261 }
00262
00263 auto getFrames() const
00264 {
00265     return total;
00266 }
00267
00268 virtual double getFps() const
00269 {
00270     return 1000.0 / interval;
00271 }
00272
00273 virtual void increaseExposure(){}
00274
00275 virtual void decreaseExposure(){}
00276
00277 virtual void increaseGain(){}
00278
00279 virtual void decreaseGain(){}
00280 };

```

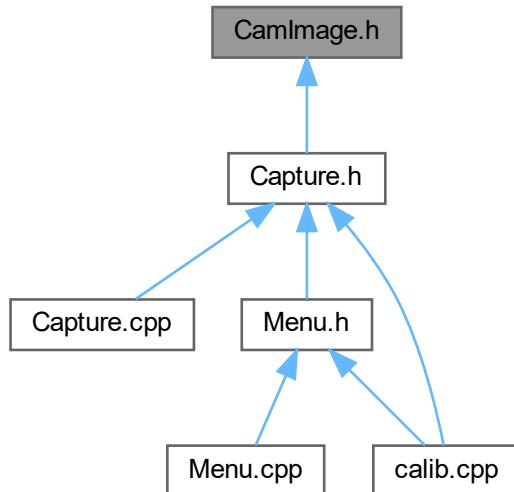
## 9.15 CamImage.h ファイル

```
#include "Camera.h"
#include "fstream"
```

CamImage.h の依存先関係図:



被依存関係図:



クラス

- class [CamImage](#)

### 9.15.1 詳解

OpenCV を使って画像ファイルを読み込むクラス

著者

Kohe Tokoi

日付

November 15, 2022

[CamImage.h](#) に定義があります。

## 9.16 CamImage.h

[詳解]

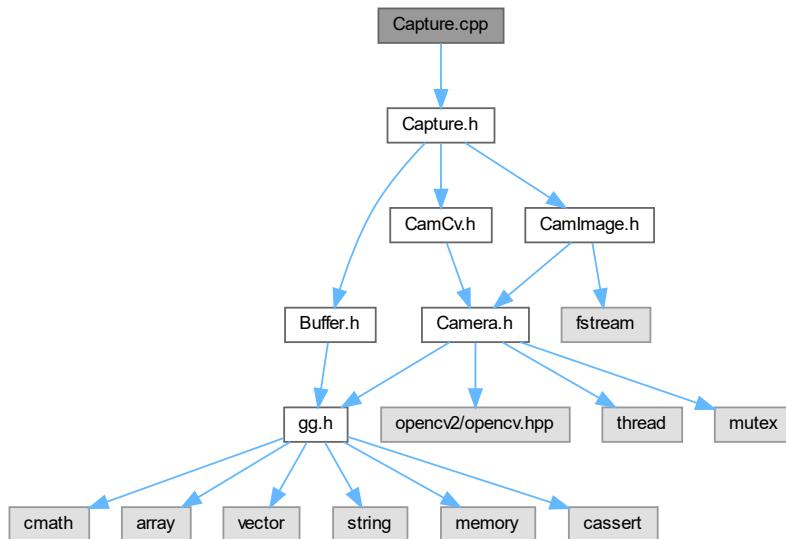
```
00001 #pragma once
00002
00010
00011 // カメラ関連の処理
00012 #include "Camera.h"
00013
00014 // ファイル入出力
00015 #include "fstream"
00016
00020 class CamImage : public Camera
00021 {
00022 public:
00023
00027     CamImage()
00028     {
00029     }
00030
00034     CamImage(std::string& filename, bool flip = false)
00035     {
00036         open(filename, flip);
00037     }
00038
00042     virtual ~CamImage()
00043     {
00044     }
00045
00052     bool open(const std::string& filename, bool flip = false)
00053     {
00054         // 画像ファイルを読み込む
00055         frame = load(filename);
00056
00057         // 画像データが読み込めなかつたら戻る
00058         if (frame.empty()) return false;
00059
00060         // 必要なら上下を反転する
00061         if (flip) cv::flip(frame, frame, 1);
00062
00063         // 転送用の一時メモリにデータを格納する
00064         copyFrame();
00065
00066         // 画像が読み込まれたことを記録する
00067         captured = true;
00068
00069         // 画像ファイルが開けた
00070         return true;
00071     }
00072
00078     bool isOpened() const
00079     {
00080         // 画像が読み込んでいたら true
00081         return !pixels.empty();
00082     }
00083
00090     static cv::Mat load(const std::string& filename)
00091     {
00092         // 画像ファイルをバイナリモードで開いて読み込み位置を最後に移動する
00093         std::ifstream file(Utf8ToTChar(filename),
00094             std::ifstream::in |
00095             std::ifstream::binary |
00096             std::ifstream::ate);
00097
00098         // 画像ファイルが開けたら
```

```
00099 if (file.good())
00100 {
00101     // 画像ファイルを読み込むメモリを確保する
00102     std::vector<char> buffer(static_cast<std::vector<char>::size_type>(file.tellg()));
00103
00104     // 画像ファイルを先頭から全部読み込む
00105     file.seekg(0, std::ifstream::beg);
00106     file.read(buffer.data(), buffer.size());
00107
00108     // 画像ファイルを閉じる
00109     file.close();
00110
00111     // 画像ファイルが読み込めたら
00112     if (file.good())
00113     {
00114         // 読み込んだ画像データを復号して返す
00115         return cv::imdecode(buffer, cv::IMREAD_COLOR);
00116     }
00117 }
00118
00119     // 読み込めなかった
00120     return cv::Mat{};
00121 }
00122
00123 static bool save(const std::string& filename, const cv::Mat& image)
00124 {
00125     // ファイル名の拡張子の場所を取り出す
00126     const auto pos{filename.find_last_of('.')};
00127
00128     // 拡張子があればそれを取り出し、無ければ ".png" にする
00129     const std::string ext{ pos != std::string::npos ? filename.substr(pos) : ".png" };
00130
00131     // 画像の符号化に失敗したら戻る
00132     std::vector<uchar> buffer;
00133     if (!cv::imencode(ext, image, buffer)) return false;
00134
00135     // 画像ファイルをバイナリモードで開く
00136     std::ofstream file(Utf8ToTChar(filename),
00137                         std::ofstream::out |
00138                         std::ofstream::binary);
00139
00140     // 画像ファイルが開けなかつたら戻る
00141     if (file.fail()) return false;
00142
00143     // 画像データをファイルに書き込む
00144     const auto ptr{ reinterpret_cast<char*>(buffer.data()) };
00145     file.write(ptr, buffer.size());
00146
00147     // 画像ファイルを閉じる
00148     file.close();
00149
00150     // 読み始めたかどうかを返す
00151     return file.good();
00152 }
```

## 9.17 Capture.cpp ファイル

```
#include "Capture.h"
```

Capture.cpp の依存先関係図:



### 9.17.1 詳解

キャプチャクラスの実装

著者

Kohe Tokoi

日付

Apilil 3, 2023

[Capture.cpp](#) に定義があります。

## 9.18 Capture.cpp

### [詳解]

```

00001
00002 #include "Capture.h"
00003
00004 // 
00005 // 画像ファイルを開く
00006 //
00007 bool Capture::openImage(const std::string& filename)
00008 {
  
```

```

00015 // 新しいキャプチャデバイスを作成したら
00016 auto camImage{ std::make_unique<CamImage>() };
00017
00018 // キャプチャデバイスを開く
00019 if (camImage->open(filename))
00020 {
00021     // このキャプチャデバイスを使うことにする
00022     camera = std::move(camImage);
00023     return true;
00024 }
00025
00026 // 開けなかった
00027 return false;
00028 }
00029
00030 //
00031 // 動画ファイルを開く
00032 //
00033 bool Capture::openMovie(const std::string& filename,
00034     cv::VideoCaptureAPIs backend)
00035 {
00036     // 新しいキャプチャデバイスを作成したら
00037     auto camCv{ std::make_unique<CamCv>() };
00038
00039     // キャプチャデバイスを開く
00040     if (camCv->open(filename, 0, 0, 0.0, "", backend))
00041     {
00042         // このキャプチャデバイスを使うことにする
00043         camera = std::move(camCv);
00044         return true;
00045     }
00046
00047 // 開けなかった
00048 return false;
00049 }
00050
00051 //
00052 // デバイスを開く
00053 //
00054 bool Capture::openDevice(int deviceNumber, std::array<int, 2>& size, double fps,
00055     cv::VideoCaptureAPIs backend, const char* fourcc)
00056 {
00057     // 既にカメラが有効なら一旦閉じる
00058     if (camera) camera->close();
00059
00060     // 新しいキャプチャデバイスを作成したら
00061     auto camCv{ std::make_unique<CamCv>() };
00062
00063     // このデバイスをデバイス番号で開いて
00064     if (camCv->open(deviceNumber, size[0], size[1], fps, fourcc, backend))
00065     {
00066         // このキャプチャデバイスを使うことにする
00067         camera = std::move(camCv);
00068         return true;
00069     }
00070
00071 // 開けなかった
00072 return false;
00073 }
00074
00075 //
00076 // キャプチャ開始
00077 //
00078 void Capture::start()
00079 {
00080     // キャプチャデバイスが有効ならキャプチャスレッドを起動する
00081     if (camera) camera->start();
00082 }
00083
00084 //
00085 // キャプチャ終了
00086 //
00087 void Capture::stop()
00088 {
00089     // キャプチャデバイスが有効ならキャプチャスレッドを停止する
00090     if (camera) camera->stop();
00091 }
00092
00093 //
00094 // キャプチャデバイスを閉じる
00095 //
00096 void Capture::close()
00097 {
00098     // キャプチャデバイスが有効ならキャプチャスレッドを停止する
00099     if (camera)
00100     {
00101         camera->close();
00102     }
00103 }

```

```

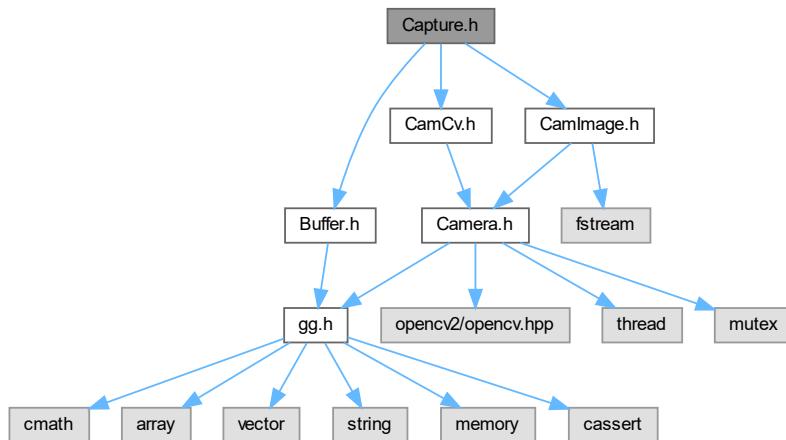
00102     camera.reset();
00103 }
00104 }
00105
00106 /**
00107 // キャプチャデバイスの解像度とフレームレートを取得する
00108 //
00109 std::array<int, 2> Capture::getSize() const
00110 {
00111     // キャプチャデバイスが有効ならその解像度を返す
00112     return camera ? camera->getSize() : std::array<int, 2>{ 0, 0 };
00113 }
00114
00115 /**
00116 // キャプチャデバイスのフレームレートを得る
00117 //
00118 double Capture::getFps() const
00119 {
00120     // キャプチャデバイスが有効ならそのフレームレートを返す
00121     return camera ? camera->getFps() : 0.0;
00122 }
00123
00124 /**
00125 // フレームを取得する
00126 //
00127 void Capture::retrieve(Buffer& buffer)
00128 {
00129     // キャプチャデバイスが有効なら
00130     if (camera)
00131     {
00132         // バッファのサイズを取得したフレームのサイズに合わせて
00133         buffer.create(camera->getWidth(), camera->getHeight(), camera->getChannels());
00134
00135         // バッファのピクセルバッファオブジェクトにフレームを転送する
00136         camera->transmit(buffer.getBufferName());
00137     }
00138 }

```

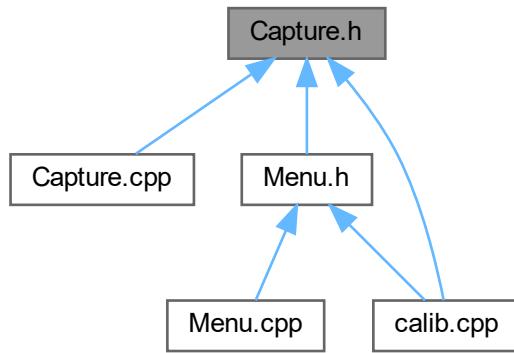
## 9.19 Capture.h ファイル

```
#include "Buffer.h"
#include "CamImage.h"
#include "CamCv.h"
```

Capture.h の依存先関係図:



被依存関係図:



クラス

- class [Capture](#)

### 9.19.1 詳解

キャプチャクラスの定義

著者

Kohe Tokoi

日付

Apilil 3, 2023

[Capture.h](#) に定義があります。

## 9.20 Capture.h

### [詳解]

```

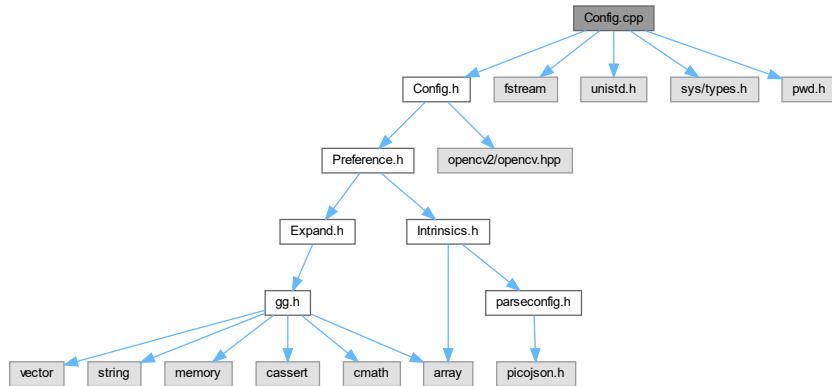
00001 #pragma once
00002
00010
00011 // バッファクラス
00012 #include "Buffer.h"
00013
00014 // OpenCV による画像ファイルの入力
00015 #include "CamImage.h"
00016
00017 // OpenCV による動画の入力
00018 #include "CamCv.h"
00019
  
```

```
00023 class Capture
00024 {
00026     std::unique_ptr<Camera> camera;
00027
00028 public:
00029
00033     Capture()
00034     {
00035     }
00036
00042     Capture(const std::string& filename)
00043     {
00044         openImage(filename);
00045     }
00046
00050     virtual ~Capture()
00051     {
00052         close();
00053     }
00054
00061     bool openImage(const std::string& filename);
00062
00070     bool openMovie(const std::string& filename,
00071                 cv::VideoCaptureAPIs backend = cv::CAP_FFMPEG);
00072
00083     bool openDevice(int deviceNumber,
00084                     std::array<int, 2>& size, double fps,
00085                     cv::VideoCaptureAPIs backend = cv::CAP_FFMPEG,
00086                     const char* fourcc = "");
00087
00091     void start();
00092
00096     void stop();
00097
00101     void close();
00102
00108     explicit operator bool() const
00109     {
00110         return camera && camera->isRunning();
00111     }
00112
00116     bool isOpened() const
00117     {
00118         return bool(camera);
00119     }
00120
00126     std::array<int, 2> getSize() const;
00127
00133     double getFps() const;
00134
00140     void retrieve(Buffer& buffer);
00141 },
```

## 9.21 Config.cpp ファイル

```
#include "Config.h"
#include <fstream>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
```

Config.cpp の依存先関係図:



## 関数

- void [getAnyList](#) (std::vector< std::string > &list)

### 9.21.1 詳解

構成データクラスの実装

著者

Kohe Tokoi

日付

February 20, 2024

[Config.cpp](#) に定義があります。

### 9.21.2 関数詳解

#### 9.21.2.1 [getAnyList\(\)](#)

```
void getAnyList (
    std::vector< std::string > & list )
```

[Config.cpp](#) の 16 行目に定義があります。

被呼び出し関係図:



## 9.22 Config.cpp

### [詳解]

```

00001
00008 #include "Config.h"
00009
00010 // 標準ライブラリ
00011 #include <fstream>
00012
00013 //
00014 // デフォルトのビデオデバイスの一覧を作る
00015 //
00016 void getAnyList(std::vector<std::string>& list)
00017 {
00018     list.emplace_back("any");
00019     list.emplace_back("Device 1");
00020     list.emplace_back("Device 2");
00021     list.emplace_back("Device 3");
00022     list.emplace_back("Device 4");
00023     list.emplace_back("Device 5");
00024     list.emplace_back("Device 6");
00025     list.emplace_back("Device 7");
00026 }
00027
00028 #if defined(_MSC_VER)
00029 //
00030 // Direct Show のビデオデバイスの一覧を作る
00031 //
00032 //    https://docs.microsoft.com/en-us/windows/win32/directshow/selecting-a-capture-device
00033 //    https://www.geekpage.jp/programming/directshow/list-capture-device.php
00034 //    http://www.antillia.com/sol9.2.0/4.25.html
00035 //
00036 #include <dshow.h>
00037 #pragma comment(lib, "strmiids")
00038
00039 void getDirectShowList(std::vector<std::string>& list)
00040 {
00041     // COM を開く
00042     HRESULT hr{ CoInitialize(nullptr) };
00043     if (FAILED(hr)) return;
00044
00045     // デバイスの列挙子を作成する
00046     ICreateDevEnum* pDevEnum{ nullptr };
00047     hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL, CLSCTX_INPROC_SERVER, IID_ICreateDevEnum,
00048     reinterpret_cast<PVOID*>(&pDevEnum));
00049
00050     // 列挙子が作れなかったら戻る
00051     if (FAILED(hr)) return;
00052
00053     // デバイスの列挙子の異名を作成する
00054     IEnumMoniker* pEnumMoniker{ nullptr };
00055     hr = pDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategory, &pEnumMoniker, 0);
00056
00057     // デバイスの列挙子はもういらないので開放する
00058     pDevEnum->Release();
00059
00060     // 列挙子の異名が作れなかったら戻る
00061     if (FAILED(hr)) return;
00062
00063     // 列挙子の異名が一つもなければ戻る
00064     if (!pEnumMoniker) return;
00065
00066     // 列挙子の異名の取り出し先
00067     IMoniker* pMoniker{ nullptr };
00068
00069     // 列挙子の異名を一つずつ取り出す
00070     while (pEnumMoniker->Next(1, &pMoniker, nullptr) == S_OK)
00071     {
00072         // プロパティバッグの場所を取り出す
00073         IPropertyBag* pPropertyBag;
00074         hr = pMoniker->BindToStorage(0, 0, IID_IPropertyBag, reinterpret_cast<void**>(&pPropertyBag));
00075
00076         // プロパティバッグの場所が取り出せなかった次に行く
00077         if (FAILED(hr))
00078         {
00079             pMoniker->Release();
00080             continue;
00081         }
00082
00083         // FriendlyName の格納場所
00084         VARIANT friendlyName;
00085         VariantInit(&friendlyName);
00086
00087         // FriendlyName を取得する
00088         hr = pPropertyBag->Read(L"friendlyName", &friendlyName, 0);

```

```

00089
00090 // Friendly Name を保存する
00091 list.emplace_back(FAILED(hr) ? "Unknown" : TCharToUtf8(friendlyName.bstrVal));
00092
00093 // FriendlyName の格納場所を消去する
00094 VariantClear(&friendlyName);
00095
00096 // プロパティバッグを解放する
00097 pMoniker->Release();
00098 pPropertyBag->Release();
00099 }
00100
00101 // デバイスの列挙子の異名を開放する
00102 pEnumMoniker->Release();
00103
00104 // COM を閉じる
00105 CoUninitialize();
00106 }
00107
00108 //
00109 // Media Foundation のビデオデバイスの一覧を作る
00110 //
00111 // https://docs.microsoft.com/ja-jp/windows/win32/medfound/audio-video-capture-in-media-foundation
00112 //
00113 #include <Mfidl.h>
00114 #include <Mfapi.h>
00115 #include <Mferror.h>
00116 #pragma comment(lib, "mf.lib")
00117 #pragma comment(lib, "mfplat.lib")
00118
00119 void getMediaFoundationList(std::vector<std::string>& list)
00120 {
00121 // Create an attribute store to hold the search criteria.
00122 IMFAttributes* pConfig{ NULL };
00123 HRESULT hr{ MFCreateAttributes(&pConfig, 1) };
00124
00125 // Request video capture devices.
00126 if (SUCCEEDED(hr))
00127 {
00128     hr = pConfig->SetGUID(
00129         MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
00130         MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
00131     );
00132 }
00133
00134 // Enumerate the devices,
00135 IMFActivate** ppDevices{ NULL };
00136 UINT32 count{ 0 };
00137 if (SUCCEEDED(hr))
00138 {
00139     hr = MFEnumDeviceSources(pConfig, &ppDevices, &count);
00140 }
00141
00142 for (DWORD i = 0; i < count; i++)
00143 {
00144     // Try to get the display name.
00145     WCHAR* szFriendlyName{ NULL };
00146     UINT32 cchName{ 0 };
00147     HRESULT hr{
00148         ppDevices[i]->GetAllocatedString(MF_DEVSOURCE_ATTRIBUTE_FRIENDLY_NAME,
00149             &szFriendlyName, &cchName)
00150     };
00151
00152     if (SUCCEEDED(hr))
00153     {
00154         list.emplace_back(TCharToUtf8(szFriendlyName));
00155     }
00156     CoTaskMemFree(szFriendlyName);
00157 }
00158
00159 for (DWORD i = 0; i < count; i++)
00160 {
00161     ppDevices[i]->Release();
00162 }
00163 CoTaskMemFree(ppDevices);
00164 }
00165
00166 // appData のバスを得るときに使う
00167 #include <shlobj.core.h>
00168
00169 #else
00170 # if defined(_APPLE_)
00171 //
00172 // macOS のビデオデバイスの一覧を作る
00173 //
00174 //
00175 void getAvFoundationList(std::vector<std::string>& list)

```

```
00176 {
00177     // TODO: macOS の AV Foundation のビデオデバイスの一覧を得る方法に書き換える
00178     getAnyList(list);
00179 }
00180 # endif
00181
00182 // パスワードのエントリからホームディレクトリの場所を得るときに使う
00183 #include <unistd.h>
00184 #include <sys/types.h>
00185 #include <pwd.h>
00186
00187 #endif
00188
00189 //
00190 // コンストラクタ
00191 //
00192 Config::Config(const std::string& filename)
00193     : title{ PROJECT_NAME }
00194     , windowSize{ 1280, 720 }
00195     , background{ 0.2f, 0.3f, 0.4f, 1.0f }
00196     , settings{ "DICTIONARY" }
00197     , menuFont{ "Mplus1-Regular.ttf" }
00198     , menuFontSize{ 20.0f }
00199 {
00200     // バックエンドごとのキャプチャデバイスの一覧を初期化する
00201     for (auto& [api, name] : backendList)
00202     {
00203         // バックエンドごとに空のリストを追加する
00204         deviceList.emplace(api, std::vector<std::string>());
00205     }
00206
00207     // キャプチャデバイスの一覧を作る
00208     getAnyList(deviceList.at(cv::CAP_ANY));
00209 #if defined(_MSC_VER)
00210     getDirectShowList(deviceList.at(cv::CAP_DSHOW));
00211     getMediaFoundationList(deviceList.at(cv::CAP_MSMF));
00212 #elif defined(_APPLE_)
00213     getAvFoundationList(deviceList.at(cv::CAP_AVFOUNDATION));
00214 #endif
00215
00216     // 構成ファイルの保存場所を決定する
00217 #if defined(_DEBUG)
00218     const auto path{ Utf8ToTChar(filename) };
00219 #else
00220 # if defined(_MSC_VER)
00221     // 構成ファイルの保存先のパス
00222     wchar_t appDataPath[MAX_PATH];
00223
00224     // AppData のパスを取得する
00225     SHGetSpecialFolderPathW(NULL, appDataPath, CSIDL_APPDATA, 0);
00226
00227     // AppData のパスに構成ファイル名を連結する
00228     const auto path{ CString(appDataPath) + TEXT("\\") + Utf8ToTChar(filename) };
00229 # else
00230     // パスワードファイルのエントリを取得する
00231     const passwd* pw{ getpwuid(getuid()) };
00232
00233     // ホームディレクトリのパスに構成ファイル名を連結する
00234     const auto path{ std::string(pw->pw_dir) + "/" + filename };
00235 # endif
00236 #endif
00237
00238     // 構成ファイルを読み込む
00239     if (!load(path))
00240     {
00241         // デフォルトの設定を追加する
00242         if (!load(Utf8ToTChar(filename))) preferenceList.emplace_back();
00243
00244         // デフォルトの設定を入れた構成ファイルを作成する
00245         save(path);
00246     }
00247 }
00248
00249 //
00250 // デストラクタ
00251 //
00252 Config::~Config()
00253 {
00254 }
00255
00256 //
00257 // 構成データを初期化する
00258 //
00259 void Config::initialize()
00260 {
00261     // 構成リストのすべて構成についてシェーダをビルドする
00262     for (auto& preference : preferenceList) preference.buildShader();
```

```

00263 // 背景色を設定する
00264     glClearColor(background[0], background[1], background[2], background[3]);
00265 }
00266 }
00267 //
00268 //
00269 // 構成ファイルを読み込む
00270 //
00271 bool Config::load(const pathString& filename)
00272 {
00273     // 構成ファイルの読み込み
00274     std::ifstream json{filename};
00275     if (!json) return false;
00276
00277     // JSON の読み込み
00278     picojson::value value;
00279     json >> value;
00280     json.close();
00281
00282     // 構成内容の取り出し
00283     const auto& object{ value.get<picojson::object>() };
00284
00285     // オブジェクトが空だったらエラー
00286     if (object.empty()) return false;
00287
00288     // ウィンドウのサイズ
00289     getValue(object, "size", windowSize);
00290
00291     // ウィンドウの背景色
00292     getValue(object, "background", background);
00293
00294     // メッシュのサンプル数
00295     getValue(object, "samples", settings.samples);
00296
00297     // キャプチャデバイスの姿勢
00298     getValue(object, "pose", settings.euler);
00299
00300     // 描画時の焦点距離
00301     getValue(object, "focal", settings.focal);
00302
00303     // 描画時の焦点距離の範囲
00304     getValue(object, "range", settings.focalRange);
00305
00306     // ArUco Marker の辞書名
00307     getString(object, "dictionary", settings.dictionaryName);
00308
00309     // 初期表示画像
00310     getString(object, "initial", initialImage);
00311
00312     // FFmpeg のリスト
00313     getString(object, "ffmpeg", deviceList[cv::CAP_FFMPEG]);
00314
00315     // Gstreamer のリスト
00316     getString(object, "gstreamer", deviceList[cv::CAP_GSTREAMER]);
00317
00318     // キャプチャデバイスの構成を探す
00319     const auto& camera{ object.find("camera") };
00320
00321     // キャプチャデバイスの構成の配列が見つからなければエラー
00322     if (camera == object.end() || !camera->second.is<picojson::array>()) return false;
00323
00324     // 配列の個々の要素について
00325     for (const auto& value : camera->second.get<picojson::array>())
00326     {
00327         // キャプチャデバイスの構成のオブジェクトを取り出す
00328         const auto& preference{ value.get<picojson::object>() };
00329
00330         // キャプチャデバイスの構成をリストに追加
00331         preferenceList.emplace_back(preference);
00332     }
00333
00334     // キャプチャデバイスの構成が一つも読み取れなければエラー
00335     return !preferenceList.empty();
00336 }
00337
00338 //
00339 // 構成ファイルを保存する
00340 //
00341 bool Config::save(const pathString& filename) const
00342 {
00343     // 設定値を保存する
00344     std::ofstream config{filename};
00345     if (!config) return false;
00346
00347     // オブジェクト
00348     picojson::object object;
00349

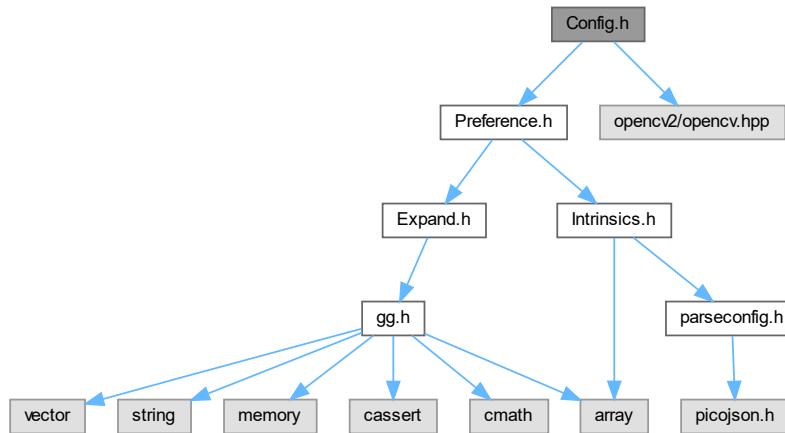
```

```
00350 // ウィンドウのサイズ
00351 setValue(object, "size", windowSize);
00352
00353 // ウィンドウの背景色
00354 setValue(object, "background", background);
00355
00356 // メッシュのサンプル数
00357 setValue(object, "samples", settings.samples);
00358
00359 // キャプチャデバイスの姿勢
00360 setValue(object, "pose", settings.euler);
00361
00362 // 描画時の焦点距離
00363 setValue(object, "focal", settings.focal);
00364
00365 // 描画時の焦点距離の範囲
00366 setValue(object, "range", settings.focalRange);
00367
00368 // ArUco Marker 辞書名
00369 setString(object, "dictionary", settings.dictionaryName);
00370
00371 // 初期表示画像
00372 setString(object, "initial", initialImage);
00373
00374 // FFmpeg のリスト
00375 setString(object, "ffmpeg", deviceList.at(cv::CAP_FFMPEG));
00376
00377 // Gstreamer のリスト
00378 setString(object, "gstreamer", deviceList.at(cv::CAP_GSTREAMER));
00379
00380 // 配列
00381 picojson::array array;
00382
00383 // 構成リストのすべて構成について
00384 for (const auto& preference : preferenceList)
00385 {
00386     // 構成の要素
00387     picojson::object camera;
00388
00389     // 構成を JSON オブジェクトに格納する
00390     preference.setPreference(camera);
00391
00392     // 要素を picojson::array に追加する
00393     array.emplace_back(picojson::value(camera));
00394 }
00395
00396 // オブジェクトに追加する
00397 object.emplace("camera", array);
00398
00399 // 構成をシリализして保存
00400 picojson::value v{ object };
00401 config << v.serialize(true);
00402 config.close();
00403
00404 // 構成データの書き込み成功
00405 return true;
00406 }
00407
00408 // バックエンドのリスト
00409 const std::map<cv::VideoCaptureAPIs, const char*> Config::backendList
00410 {
00411 #if defined(_MSC_VER)
00412     { cv::CAP_MSMF, "Media Foundation" },
00413     { cv::CAP_DSHOW, "Direct Show" },
00414 #elif defined(_APPLE_)
00415     { cv::CAP_AVFOUNDATION, "AV Foundation" },
00416 #endif
00417     { cv::CAP_GSTREAMER, "GStreamer" },
00418     { cv::CAP_ANY, "(any)" },
00419     { cv::CAP_FFMPEG, u8"動画ファイル履歴" }
00420 };
00421
00422 // コーデックのリスト
00423 const std::vector<const char*> Config::codecList
00424 {
00425     "(any)",
00426     "MJPG",
00427     "H264",
00428     "BGR3",
00429     "YUY2",
00430     "I420",
00431     "NV12"
00432 };
00433
00434 // キャプチャデバイスのリスト
00435 std::map<cv::VideoCaptureAPIs, std::vector<std::string>> Config::deviceList;
00436
```

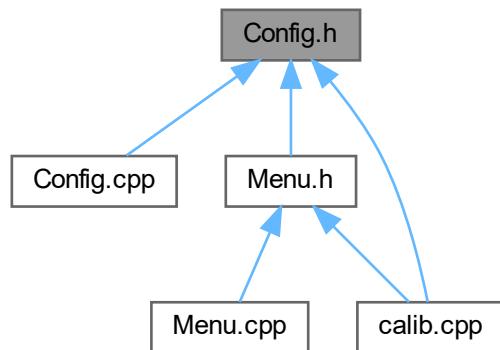
```
00437 // 初期表示の画像ファイル名
00438 std::string Config::initialImage{ "initial.jpg" };
```

## 9.23 Config.h ファイル

```
#include "Preference.h"
#include <opencv2/opencv.hpp>
Config.h の依存先関係図:
```



被依存関係図:



クラス

- struct **Settings**
- class **Config**

### 9.23.1 詳解

構成データクラスの定義

著者

Kohe Tokoi

日付

March 6, 2024

Config.h に定義があります。

## 9.24 Config.h

### [詳解]

```
00001 #pragma once
00002
00010
00011 // キャプチャデバイスの構成
00012 #include "Preference.h"
00013
00014 // OpenCV
00015 #include <opencv2/opencv.hpp>
00016
00020 struct Settings
00021 {
00023     int samples;
00024
00026     std::array<float, 3> euler;
00027
00029     static constexpr decltype(euler) defaultEuler{ 0.0f, 0.0f, 0.0f };
00030
00032     float focal;
00033
00035     static constexpr decltype(focal) defaultFocal{ 50.0f };
00036
00038     std::array<float, 2> focalRange;
00039
00041     static constexpr decltype(focalRange) defaultFocalRange{ 10.0f, 200.0f };
00042
00044     std::string dictionaryName;
00045
00047     std::array<float, 2> checkerLength;
00048
00052     Settings(const std::string& dictionaryName)
00053         : samples{ 57600 }
00054         , euler{ defaultEuler }
00055         , focal{ defaultFocal }
00056         , focalRange{ defaultFocalRange }
00057         , dictionaryName{ dictionaryName }
00058         , checkerLength{ 4.0f, 2.0f }
00059     {}
00060
00064     auto getFocal() const
00065     {
00066         // 投影面の対角線長は 35mm (17.5mm × 2) とする
00067         return focal / 17.5f;
00068     }
00069 };
00070
00074 class Config
00075 {
00077     friend class Menu;
00078
00080     std::string title;
00081
00083     std::array<GLsizei, 2> windowSize;
00084
00086     std::array<GLfloat, 4> background;
```

```

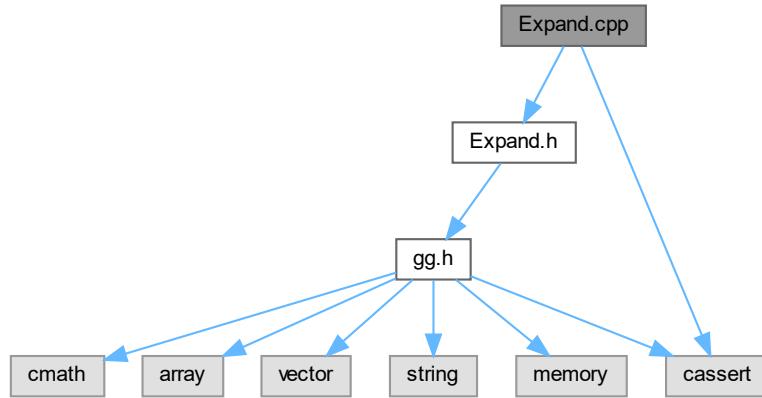
00087     std::string menuFont;
00090
00092     float menuFontSize;
00093
00095     Settings settings;
00096
00098     static const std::map<cv::VideoCaptureAPIs, const char*> backendList;
00099
00101     static const std::vector<const char*> codecList;
00102
00104     static std::map <cv::VideoCaptureAPIs, std::vector<std::string>> deviceList;
00105
00107     static std::string initialImage;
00108
00110     std::vector<Preference> preferenceList;
00111
00112 public:
00113
00119     Config(const std::string& filename);
00120
00124     virtual ~Config();
00125
00131     void initialize();
00132
00139     bool load(const pathString& filename);
00140
00147     bool save(const pathString& filename) const;
00148
00154     const auto& getTitle() const
00155     {
00156         return title;
00157     }
00158
00164     auto getWidth() const
00165     {
00166         return windowSize[0];
00167     }
00168
00174     auto getHeight() const
00175     {
00176         return windowSize[1];
00177     }
00178
00182     const auto& getInitialImage() const
00183     {
00184         return initialImage;
00185     }
00186
00190     const auto& getDictionaryName() const
00191     {
00192         return settings.dictionaryName;
00193     }
00194
00198     const auto& getCheckerLength() const
00199     {
00200         return settings.checkerLength;
00201     }
00202
00209     const auto& getDeviceList(cv::VideoCaptureAPIs api) const
00210     {
00211         return deviceList.at(api);
00212     }
00213
00220     auto getDeviceCount(cv::VideoCaptureAPIs api) const
00221     {
00222         return static_cast<int>(deviceList.at(api).size());
00223     }
00224
00232     const auto& getDeviceName(cv::VideoCaptureAPIs api, int number) const
00233     {
00234         static const std::string empty{};
00235         const auto& list{ deviceList.at(api) };
00236         return list.empty() ? empty : list[number];
00237     }
00238 };

```

## 9.25 Expand.cpp ファイル

```
#include "Expand.h"
#include <cassert>
```

Expand.cpp の依存先関係図:



## 9.26 Expand.cpp

### [詳解]

```

00001 // 展開用シェーダ
00002 // 展開用シェーダ
00003 //
00004 #include "Expand.h"
00005
00006 // 標準ライブラリ
00007 #include <cassert>
00008
00009 //
00010 // コンストラクタ
00011 //
00012 Expand::Expand(const std::string& vert, const std::string& frag)
00013   : program{ gg::ggLoadShader(vert, frag) }
00014   , imageLoc{ glGetUniformLocation(program, "image") }
00015   , screenLoc{ glGetUniformLocation(program, "screen") }
00016   , focalLoc{ glGetUniformLocation(program, "focal") }
00017   , rotationLoc{ glGetUniformLocation(program, "rotation") }
00018   , circleLoc{ glGetUniformLocation(program, "circle") }
00019   , borderLoc{ glGetUniformLocation(program, "border") }
00020   , gapLoc{ glGetUniformLocation(program, "gap") }
00021 {
00022   // プログラムオブジェクトが作れなかったら落とす
00023   assert(program);
00024 }
00025
00026 //
00027 // デストラクタ
00028 //
00029 Expand::~Expand()
00030 {
00031   glDeleteProgram(program);
00032 }
00033
00037 std::array<int, 2> Expand::setup(int samples, GLfloat aspect, const gg::GgMatrix& pose,
00038   const std::array<GLfloat, 2>& fov, const std::array<GLfloat, 2>& center, GLfloat focal,
00039   const std::array<GLfloat, 4>& border, int unit) const
00040 {
00041   // プログラムオブジェクトの指定
00042   glUseProgram(program);
00043
00044   // テクスチャユニットの指定
00045   glUniform1i(imageLoc, unit);
00046
00047   // 境界色
00048   glUniform4fv(borderLoc, 1, border.data());
00049
00050   // 投影像の画角（度）と中心位置
  
```

```

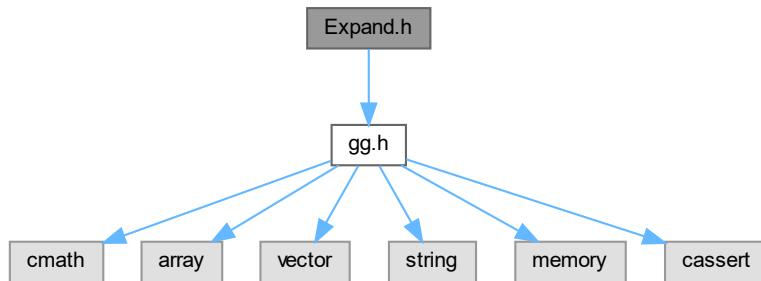
00051 glUniform4f(circleLoc, fov[0], fov[1], center[0], center[1]);
00052
00053 // スクリーンのサイズと中心位置
00054 // screen[0] = (right - left) / 2
00055 // screen[1] = (top - bottom) / 2
00056 // screen[2] = (right + left) / 2
00057 // screen[3] = (top + bottom) / 2
00058 const GLfloat screen[] { aspect, 1.0f, 0.0f, 0.0f };
00059 glUniform4fv(screenLoc, 1, screen);
00060
00061 // レンズの主点とスクリーンの距離
00062 glUniform1f(focalLoc, focal);
00063
00064 // 背景に対する視線の回転行列
00065 glUniformMatrix4fv(rotationLoc, 1, GL_FALSE, pose.get());
00066
00067 // メッシュの横の格子点数
00068 // 標本点の数 (頂点数) samples = w * h とするとき、これに縦横比
00069 // aspect = w / h をかければ aspect * samples = w * w となるから、
00070 // w = sqrt(aspect * samples), h = samples / w で求められる
00071 const auto w{ static_cast<int>(sqrt(aspect * samples)) };
00072 const auto h{ samples / w };
00073
00074 // 格子間隔
00075 // クリッピング空間全体を埋める四角形は [-1, 1] の範囲すなわち
00076 // 縦横 2 の大きさだから、それを縦横の (格子数 - 1) で割る
00077 std::array<GLfloat, 2> gap{ 2.0f / (w - 1), 2.0f / (h - 1) };
00078 glUniform2fv(gapLoc, 1, gap.data());
00079
00080 // 描画するメッシュの横と縦の格子点数を返す
00081 return std::array<int, 2>{ w, h };
00082 }

```

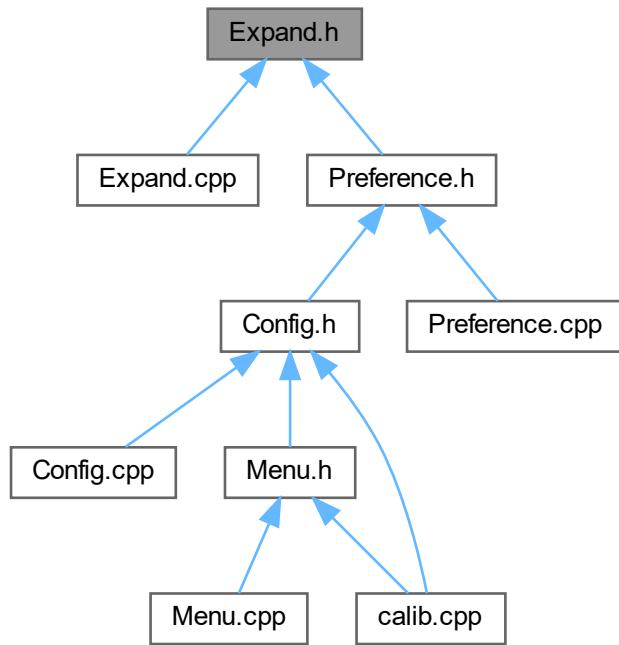
## 9.27 Expand.h ファイル

#include "gg.h"

Expand.h の依存先関係図:



被依存関係図:



クラス

- class [Expand](#)

### 9.27.1 詳解

展開用シェーダクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Expand.h](#) に定義があります。

## 9.28 Expand.h

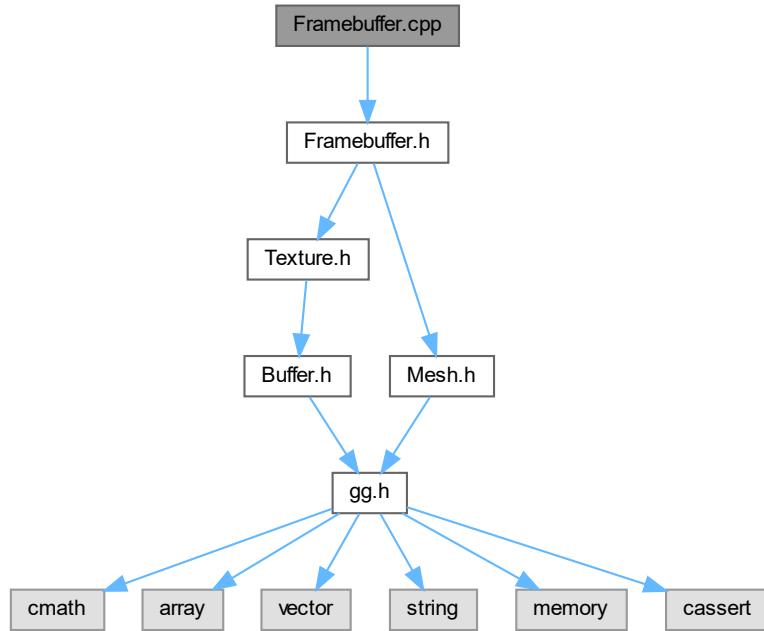
### [詳解]

```
00001 #pragma once
00002
00010
00011 // 補助プログラム
00012 #include "gg.h"
00013
00017 class Expand
00018 {
00020     const GLuint program;
00021
00023     const GLint imageLoc;
00024
00026     const GLint screenLoc;
00027
00029     const GLint focalLoc;
00030
00032     const GLint rotationLoc;
00033
00035     const GLint circleLoc;
00036
00038     const GLint borderLoc;
00039
00041     const GLint gapLoc;
00042
00043 public:
00044
00051     Expand(const std::string& vert, const std::string& frag);
00052
00058     Expand(const Expand& shader) = delete;
00059
00063     virtual ~Expand();
00064
00070     Expand& operator=(const Expand& shader) = delete;
00071
00077     auto getProgram() const
00078     {
00079         return program;
00080     }
00081
00099     std::array<GLsizei, 2> setup(int samples, GLfloat aspect, const gg::GgMatrix& pose,
00100         const std::array<GLfloat, 2>& fov, const std::array<GLfloat, 2>& center, GLfloat focal,
00101         const std::array<GLfloat, 4>& border, int unit = 0) const;
00102 };
```

## 9.29 Framebuffer.cpp ファイル

```
#include "Framebuffer.h"
```

Framebuffer.cpp の依存先関係図:



### 9.29.1 詳解

フレームバッファオブジェクトクラスの実装

著者

Kohe Tokoi

日付

February 20, 2024

[Framebuffer.cpp](#) に定義があります。

## 9.30 Framebuffer.cpp

### [詳解]

```

00001
00008 #include "Framebuffer.h"
00009
00010 //
00011 // フレームバッファオブジェクトを作成する
00012 //
00013 GLuint Framebuffer::createFramebuffer(GLuint textureName, GLenum attachment)
00014 {
00015     GLuint framebufferName;
00016     glGenFramebuffers(1, &framebufferName);
00017     glBindFramebuffer(GL_FRAMEBUFFER, framebufferName);
00018     glFramebufferTexture(GL_FRAMEBUFFER, attachment, textureName, 0);
00019     glBindBuffer(GL_FRAMEBUFFER, attachment);
00020     glReadBuffer(GL_FRAMEBUFFER, 0);
00021     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00022     return framebufferName;
00023 }
00024
00025 //
00026 // フレームバッファオブジェクトを作成するコンストラクタ
00027 //
00028 Framebuffer::Framebuffer(GLsizei width, GLsizei height, int channels,
00029     GLenum attachment)
00030     : attachment{ attachment }
00031     , viewport{ 0, 0, 0, 0 }
00032 {
00033     Framebuffer::create(width, height, channels);
00034 }
00035
00041 Framebuffer::Framebuffer(const Framebuffer& framebuffer)
00042     : attachment{ framebuffer.attachment }
00043     , viewport{ framebuffer.viewport }
00044 {
00045     Framebuffer::copy(framebuffer);
00046 }
00047
00053 Framebuffer::Framebuffer(Framebuffer&& framebuffer) noexcept
00054 {
00055     *this = std::move(framebuffer);
00056 }
00057
00058 //
00059 // デストラクタ
00060 //
00061 Framebuffer::~Framebuffer()
00062 {
00063     discard();
00064 }
00065
00066 //
00067 // 代入演算子
00068 //
00069 Framebuffer& Framebuffer::operator=(const Framebuffer& framebuffer)
00070 {
00071     // 代入元と代入先が同じでなければ
00072     if (&framebuffer != this)
00073     {
00074         // 引数のフレームバッファオブジェクトをコピーする
00075         Framebuffer::copy(framebuffer);
00076     }
00077
00078     // このフレームバッファオブジェクトを返す
00079     return *this;
00080 }
00081
00082 //
00083 // ムーブ代入演算子
00084 //
00085 Framebuffer& Framebuffer::operator=(Framebuffer&& framebuffer) noexcept
00086 {
00087     // 代入元と代入先が同じでなければ
00088     if (&framebuffer != this)
00089     {
00090         // ムーブ元のフレームバッファオブジェクトをコピーする
00091         Framebuffer::copy(framebuffer);
00092
00093         // ムーブ元のバッファを破棄する
00094         framebuffer.Buffer::discard();
00095
00096         // ムーブ元のテクスチャを破棄する
00097         framebuffer.Texture::discard();
00098

```

```
00099     // ムーブ元のフレームバッファオブジェクトを破棄する
00100     framebuffer.Framebuffer::discard();
00101 }
00102
00103 // このフレームバッファオブジェクトを返す
00104 return *this;
00105 }
00106
00107 //
00108 // フレームバッファオブジェクトを破棄する
00109 //
00110 void Framebuffer::discard()
00111 {
00112     // デフォルトのフレームバッファオブジェクトに戻す
00113     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00114
00115     // フレームバッファオブジェクトを削除する
00116     glDeleteFramebuffers(1, &framebufferName);
00117     framebufferName = 0;
00118
00119     // バッファオブジェクトのメンバを初期化する
00120     framebufferSize = std::array<int, 2>{ 0, 0 };
00121     framebufferChannels = 0;
00122     attachment = GL_COLOR_ATTACHMENT0;
00123     viewport = std::array<GLint, 4>{ 0, 0, 0, 0 };
00124 }
00125
00126 //
00127 // フレームバッファオブジェクトを作成する
00128 //
00129 void Framebuffer::create(GLsizei width, GLsizei height, int channels)
00130 {
00131     // 既存のテクスチャを破棄して新しいテクスチャを作成する
00132     Texture::create(width, height, channels);
00133
00134     // 指定したサイズがフレームバッファオブジェクトのサイズと同じなら何もしない
00135     if (width == framebufferSize[0] && height == framebufferSize[1]
00136         && channels == framebufferChannels) return;
00137
00138     // フレームバッファオブジェクトのサイズとチャンネル数を記録する
00139     framebufferSize = std::array<int, 2>{ width, height };
00140     framebufferChannels = channels;
00141
00142     // 以前のフレームバッファオブジェクトを削除する
00143     glDeleteFramebuffers(1, &framebufferName);
00144
00145     // フレームバッファオブジェクトを作成する
00146     glGenFramebuffers(1, &framebufferName);
00147     glBindFramebuffer(GL_FRAMEBUFFER, framebufferName);
00148     glFramebufferTexture(GL_FRAMEBUFFER, attachment, getTextureName(), 0);
00149     glDrawBuffers(1, &attachment);
00150     glReadBuffer(attachment);
00151     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00152 }
00153
00154 //
00155 // フレームバッファオブジェクトをコピーする
00156 //
00157 void Framebuffer::copy(const Buffer& framebuffer) noexcept
00158 {
00159     // コピー元と同じサイズの空のフレームバッファオブジェクトを作る
00160     Framebuffer::create(framebuffer.getWidth(), framebuffer.getHeight(),
00161                         framebuffer.getChannels());
00162
00163     // 作成したフレームバッファオブジェクトのバッファにコピーする
00164     copyBuffer(framebuffer);
00165 }
00166
00167 //
00168 // レンダリング先をフレームバッファオブジェクトに切り替える
00169 //
00170 void Framebuffer::bindFramebuffer()
00171 {
00172     // 現在のビューポートを保存する
00173     glGetIntegerv(GL_VIEWPORT, viewport.data());
00174
00175     // 描画先をフレームバッファオブジェクトに切り替える
00176     glBindFramebuffer(GL_FRAMEBUFFER, framebufferName);
00177
00178     // ビューポートをフレームバッファオブジェクトに合わせる
00179     glViewport(0, 0, framebufferSize[0], framebufferSize[1]);
00180 }
00181
00182 //
00183 // レンダリング先を通常のフレームバッファに戻す
00184 //
00185 void Framebuffer::unbindFramebuffer() const
```

```

00186 {
00187     // 描画先を通常のフレームバッファに戻す
00188     glBindFramebuffer(GL_FRAMEBUFFER, 0);
00189
00190     // 読み書きを通常のフレームバッファのバックバッファに対して行う
00191     glDrawBuffer(GL_BACK);
00192     glReadBuffer(GL_BACK);
00193
00194     // ビューポートを復帰する
00195     glViewport(viewport[0], viewport[1], viewport[2], viewport[3]);
00196 }
00197
00198 //
00199 // テクスチャを展開してフレームバッファオブジェクトを更新する
00200 //
00201 void Framebuffer::update(const std::array<int, 2>& size)
00202 {
00203     // 描画先をフレームバッファオブジェクトに切り替える
00204     bindFramebuffer();
00205
00206     // テクスチャをフレームバッファオブジェクトに展開する
00207     mesh.draw(size);
00208
00209     // 描画先を通常のフレームバッファに戻す
00210     unbindFramebuffer();
00211 }
00212
00213 //
00214 // テクスチャを展開してフレームバッファオブジェクトを更新する
00215 //
00216 void Framebuffer::update(const std::array<int, 2>& size, const Texture& frame, int unit)
00217 {
00218     // 展開するするテクスチャを指定する
00219     frame.bindTexture(unit);
00220
00221     // テクスチャをフレームバッファオブジェクトに展開する
00222     update(size);
00223
00224     // 展開するするテクスチャの指定を解除する
00225     frame.unbindTexture();
00226 }
00227
00228 //
00229 // フレームバッファオブジェクトの表示
00230 //
00231 void Framebuffer::draw(GLsizei width, GLsizei height) const
00232 {
00233     // フレームバッファオブジェクトの縦横比
00234     const auto f{ static_cast<float>(getWidth() * height) };
00235
00236     // ウィンドウの表示領域の縦横比
00237     const auto d{ static_cast<float>(getHeight() * width) };
00238
00239     // 描画する領域
00240     GLint dx0, dy0, dx1, dy1;
00241
00242     // 表示領域の右上端の位置を求める
00243     --width;
00244     --height;
00245
00246     // フレームバッファオブジェクトの縦横比が大きかったら
00247     if (f > d)
00248     {
00249         // ディスプレイ上の描画する領域の高さを求める
00250         const auto h{ static_cast<GLint>(d / getWidth() + 0.5f) };
00251
00252         // 表示が横長なので描画する領域の横幅いっぱいに表示する
00253         dx0 = 0;
00254         dx1 = width;
00255
00256         // 高さは縦横比を維持して描画する領域の中央に描く
00257         dy0 = (height - h) / 2;
00258         dy1 = dy0 + h;
00259     }
00260     else
00261     {
00262         // ディスプレイ上の描画する領域の幅を求める
00263         const auto w{ static_cast<GLint>(f / getHeight() + 0.5f) };
00264
00265         // 表示が縦長なので描画する領域の高さいっぱいに表示する
00266         dy0 = 0;
00267         dy1 = height;
00268
00269         // 横幅は縦横比を維持して描画する領域の中央に描く
00270         dx0 = (width - w) / 2;
00271         dx1 = dx0 + w;
00272     }
}

```

```

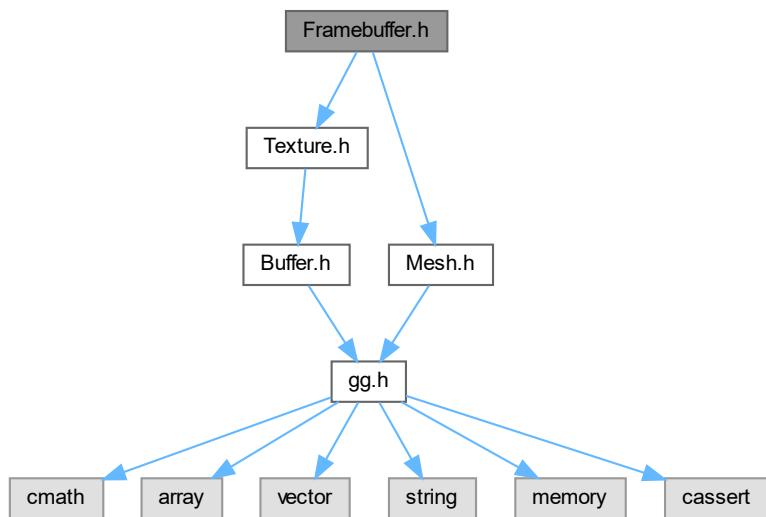
00273 // フレームバッファオブジェクトを読み込み元にする
00274 glBindFramebuffer(GL_READ_FRAMEBUFFER, framebufferName);
00275 glReadBuffer(attachment);
00276
00277 // 現在のフレームバッファを消去する
00278 glClear(GL_COLOR_BUFFER_BIT);
00279
00280 // フレームバッファオブジェクトの内容を通常のフレームバッファに書き込む
00281 glBlitFramebuffer(0, 0, getWidth() - 1, getHeight() - 1,
00282 dx0, dy1, dx1, dy0, GL_COLOR_BUFFER_BIT, GL_NEAREST);
00283
00284 // 読み込み元を通常のフレームバッファに戻す
00285 glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00286 glReadBuffer(GL_BACK);
00287
00288 }
```

## 9.31 Framebuffer.h ファイル

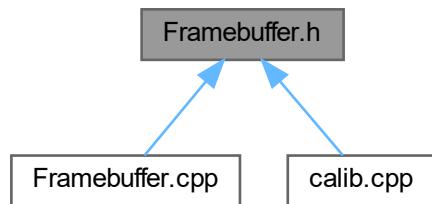
```
#include "Texture.h"
```

```
#include "Mesh.h"
```

Framebuffer.h の依存先関係図:



被依存関係図:



## クラス

- class [Framebuffer](#)

### 9.31.1 詳解

フレームバッファオブジェクトクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Framebuffer.h](#) に定義があります。

## 9.32 Framebuffer.h

### [詳解]

```

00001 #pragma once
00002
00010
00011 // テクスチャ
00012 #include "Texture.h"
00013
00014 // 展開に用いるメッシュ
00015 #include "Mesh.h"
00016
00020 class Framebuffer : public Texture
00021 {
00023     std::array<int, 2> framebufferSize;
00024
00026     int framebufferChannels;
00027
00029     GLuint framebufferName;
00030
00032     GLenum attachment;
00033
00035     std::array<GLuint, 4> viewport;
00036
00037 // 展開に用いるメッシュ
00038     Mesh mesh;
00039
00047     static GLuint createFramebuffer(GLuint textureName, GLenum attachment);
00048
00049 public:
00050
00054     Framebuffer()
00055         : Texture{}
00056         , framebufferSize{ 0, 0 }
00057         , framebufferChannels{ 0 }
00058         , framebufferName{ 0 }
00059         , attachment{ GL_COLOR_ATTACHMENT0 }
00060         , viewport{ 0, 0, 0, 0 }
00061     {
00062     }
00063
00072     Framebuffer(GLsizei width, GLsizei height, int channels = 3,
00073                 GLenum attachment = GL_COLOR_ATTACHMENT0);
00074
00080     Framebuffer(const Framebuffer& framebuffer);
00081
00087     Framebuffer(Framebuffer&& framebuffer) noexcept;
00088
00092     virtual ~Framebuffer();

```

```

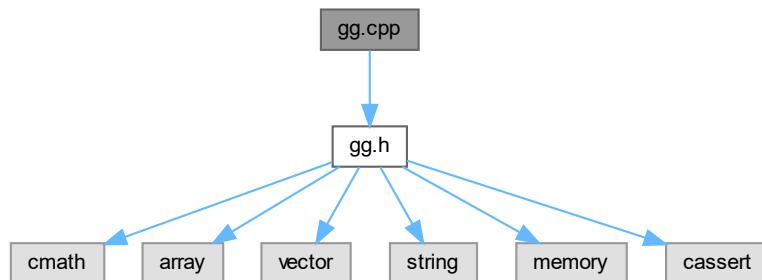
00093     Framebuffer& operator=(const Framebuffer& framebuffer);
00100
00101     Framebuffer& operator=(Framebuffer&& framebuffer) noexcept;
00108
00109     virtual void create(GLsizei width, GLsizei height, int channels);
00112
00113     virtual void copy(const Buffer& framebuffer) noexcept;
00116
00117     virtual void discard();
00120
00123     auto getFramebufferName() const
00124     {
00125         return framebufferName;
00126     }
00129
00130     virtual const std::array<GLsizei, 2>& getSize() const
00131     {
00132         return framebufferSize;
00133     }
00136
00137     virtual int getChannels() const
00138     {
00139         return framebufferChannels;
00140     }
00143
00144     void bindFramebuffer();
00147
00148     void unbindFramebuffer() const;
00149
00150     void update(const std::array<int, 2>& size);
00151
00152     void update(const std::array<int, 2>& size, const Texture& frame, int unit = 0);
00153
00154     void draw(GLsizei width, GLsizei height) const;
00157
00158 };
00159

```

## 9.33 gg.cpp ファイル

#include "gg.h"

gg.cpp の依存先関係図:



### 9.33.1 詳解

ゲームグラフィックス特論の宿題用補助プログラム GLFW3 版の定義。

著者

Kohe Tokoi

日付

January 5, 2024

gg.cpp に定義があります。

## 9.34 gg.cpp

### [詳解]

```

00001 /*
00002 ゲームグラフィックス特論用補助プログラム GLFW3 版
00004
00005 Copyright (c) 2011-2024 Kohe Tokoi. All Rights Reserved.
00006
00007 Permission is hereby granted, free of charge, to any person obtaining a copy
00008 of this software and associated documentation files (the "Software"), to deal
00009 in the Software without restriction, including without limitation the rights
0010 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
0011 copies or substantial portions of the Software.
0012
0013 The above copyright notice and this permission notice shall be included in
0014 all copies or substantial portions of the Software.
0015
0016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
0017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
0018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
0019 KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
0020 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
0021 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
0022
0023 */
0024
0032 #include "gg.h"
0033
0035
0036 // 標準ライブラリ
0037 #include <cfloat>
0038 #include <cstdlib>
0039 #include <iostream>
0040 #include <fstream>
0041 #include <sstream>
0042 #include <map>
0043
0045 #define READ_TEXTURE_COORDINATE_FROMOBJ 0
0046
0047 // Windows (Visual Studio) のとき
0048 #if defined(_MSC_VER)
0049 // デバッグビルドかどうか調べて
0050 # if defined(_DEBUG)
0051 // デバッグビルドならそのことを示す記号定数を別に定義して
0052 # define DEBUG
0053 // デバッグビルド用のライブラリをリンクする
0054 # pragma comment(lib, "glfw3d.lib")
0055 # else
0056 // リリースビルドならコンソールにメッセージを出さないようにして
0057 # pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
0058 // リリースビルド用のライブラリをリンクする
0059 # pragma comment(lib, "glfw3.lib")
0060 # endif
0061
0062 //
0063 // For VC++ MFC Convert UTF-8 to TCHAR, or Convert TCHAR to UTF-8. VC++ MFC用 UTF-8⇒TCHARの変換処理
0064 //
0065 // Original author: mt-u
0066 // Copyright (c) 2013 mt-u
0067 // https://gist.github.com/mt-u/6878251
0068 //
0069 // Modified by: Kohe Tokoi
0070 //
0071
0072 //
0073 // UTF-8 文字列を CString に変換する
0074 //
0075 pathString Utf8ToTChar(const std::string& string)
0076 {
0077     // UTF-8 文字列を UTF-16 に変換した後の文字列の長さを求める
0078     const INT length{ MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, NULL, 0) };
0079

```

```
00080 // 変換結果の格納に必要な長さのメモリを確保する
00081 std::vector<WCHAR> utf16(length);
00082
00083 // UTF-8 文字列を UTF-16 に変換する
00084 MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, utf16.data(), length);
00085
00086 // 変換した文字列を CString にして返す
00087 return CString{ utf16.data(), static_cast<int>(wcslen(utf16.data())) };
00088 }
00089
00090 //
00091 // Cstring を UTF-8 文字列に変換する
00092 //
00093 std::string TCHARToUtf8(const pathString& cstring)
00094 {
00095 // 与えられた文字列を一旦 UTF-16 に変換する
00096 CStringW cstringw{ cstring };
00097
00098 // UTF-16 を UTF-8 に変換した後の文字列の長さを求める
00099 const INT length{ WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, NULL, 0, NULL, NULL) };
00100
00101 // 変換結果の格納に必要な長さのメモリを確保する
00102 std::vector<CHAR> utf8(length);
00103
00104 // UTF-16 を UTF-8 に変換する
00105 WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, utf8.data(), length, NULL, NULL);
00106
00107 // 変換した文字列を std::string にして返す
00108 return std::string{ utf8.data(), strlen(utf8.data()) };
00109 }
00110 #endif
00111
00112 // OpenGL 3.2 の API のエントリポイント
00113 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
00114 PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
00115 PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
00116 PFNGLACTIVETEXTUREPROC glActiveTexture;
00117 PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
00118 PFNGLATTACHSHADERPROC glAttachShader;
00119 PFNGLBEGINCNDITONALRENDERNVPROC glBeginConditionalRenderNV;
00120 PFNGLBEGINCNDITONALRENDERPROC glBeginConditionalRender;
00121 PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00122 PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00123 PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00124 PFNGLBEGINQUERYPROC glBeginQuery;
00125 PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
00126 PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
00127 PFNGLBINDBUFFERBASEPROC glBindBufferBase;
00128 PFNGLBINDBUFFERPROC glBindBuffer;
00129 PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
00130 PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
00131 PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
00132 PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
00133 PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
00134 PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
00135 PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
00136 PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
00137 PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00138 PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00139 PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
00140 PFNGLBINDSAMPLERPROC glBindSampler;
00141 PFNGLBINDSAMPLERSPROC glBindSamplers;
00142 PFNGLBINDTEXTUREPROC glBindTexture;
00143 PFNGLBINDTEXTURESPROC glBindTextures;
00144 PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
00145 PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
00146 PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
00147 PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
00148 PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
00149 PFNGLBLENDBARRIERKHRPROC glBindBlendBarrierKHR;
00150 PFNGLBLENDBARRIERNVPROC glBindBlendBarrierNV;
00151 PFNGLBLENDCOLORPROC glBindColor;
00152 PFNGLBLENDEQUATIONIARBPROC glBindEquationiARB;
00153 PFNGLBLENDEQUATIONIPROC glBindEquationi;
00154 PFNGLBLENDEQUATIONPROC glBindEquation;
00155 PFNGLBLENDEQUATIONSSEPARATEIARBPROC glBindEquationSeparateiARB;
00156 PFNGLBLENDEQUATIONSEPARATEIPROC glBindEquationSeparatei;
00157 PFNGLBLENDEQUATIONSEPARATEPROC glBindEquationSeparate;
00158 PFNGLBLENDFUNCIARBPROC glBindFunciARB;
00159 PFNGLBLENDFUNCIPROC glBindFunci;
00160 PFNGLBLENDFUNCPROC glBindFunc;
00161 PFNGLBLENDFUNCSEPARATEIARBPROC glBindFuncSeparateiARB;
00162 PFNGLBLENDFUNCSEPARATEIPROC glBindFuncSeparatei;
00163 PFNGLBLENDFUNCSEPARATEPROC glBindFuncSeparate;
00164 PFNGLBLENDPARAMETERINVPROC glBindParameteriNV;
00165 PFNGLBLITFRAMEBUFFERPROC glBindBlitFramebuffer;
00166 PFNGLBLITNAMEDFRAMEBUFFERPROC glBindNamedFramebuffer;
```

```
00167 PFNGLBUFFERADDRESSRANGENVPROC glBufferAddressRangeNV;
00168 PFNGLBUFFERDATAPROC glBufferData;
00169 PFNGLBUFFERPAGECOMMITMENTARBPROC glBufferPageCommitmentARB;
00170 PFNGLBUFFERSTORAGEPROC glBufferStorage;
00171 PFNGLBUFFERSUBDATAPROC glBufferSubData;
00172 PFNGLCALLCOMMANDLISTNVPROC glCallCommandListNV;
00173 PFNGLCHECKFRAMEBUFFERSTATUSPROC glCheckFramebufferStatus;
00174 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glCheckNamedFramebufferStatusEXT;
00175 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glCheckNamedFramebufferStatus;
00176 PFNGLCLAMPCOLORPROC glClampColor;
00177 PFNGLCLEARBUFFERDATAPROC glClearBufferData;
00178 PFNGLCLEARBUFFERFIPROC glClearBufferfi;
00179 PFNGLCLEARBUFFERFVPROC glClearBufferfv;
00180 PFNGLCLEARBUFFERIVPROC glClearBufferiv;
00181 PFNGLCLEARBUFFERSUBDATAPROC glClearBufferSubData;
00182 PFNGLCLEARBUFFERUIVPROC glClearBufferuiv;
00183 PFNGLCLEARCOLORPROC glClearColor;
00184 PFNGLCLEARDEPTHFPROC glClearDepthf;
00185 PFNGLCLEARDEPTHPROC glClearDepth;
00186 PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
00187 PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
00188 PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
00189 PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
00190 PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
00191 PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
00192 PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
00193 PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferuiv;
00194 PFNGLCLEARPROC glClear;
00195 PFNGLCLEARSTENCILPROC glClearStencil;
00196 PFNGLCLEARTEXIMAGEPROC glClearTexImage;
00197 PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
00198 PFNGLCLIENTATTRIBDEFAULTEXTPROC glClientAttribDefaultEXT;
00199 PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
00200 PFNGLCLIPCONTROLPROC glClipControl;
00201 PFNGLCOLORFORMATNVPROC glColorFormatNV;
00202 PFNGLCOLORMASKIPROC glColorMaski;
00203 PFNGLCOLORMASKPROC glColorMask;
00204 PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
00205 PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
00206 PFNGLCOMPILEL_SHADERINCLUDEARBPROC glCompileShaderIncludeARB;
00207 PFNGLCOMPILESHADERPROC glCompileShader;
00208 PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
00209 PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
00210 PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
00211 PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
00212 PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
00213 PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
00214 PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
00215 PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
00216 PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
00217 PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
00218 PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
00219 PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
00220 PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
00221 PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
00222 PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
00223 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
00224 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
00225 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00226 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00227 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00228 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00229 PFNGLCONSERVATIVE_RASTER_PARAMETERFNVPROC glConservativeRasterParameterfnv;
00230 PFNGLCONSERVATIVE_RASTER_PARAMETERINVPROC glConservativeRasterParameterinv;
00231 PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
00232 PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
00233 PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
00234 PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
00235 PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
00236 PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
00237 PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
00238 PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
00239 PFNGLCOPYPATHNVPROC glCopyPathNV;
00240 PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
00241 PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
00242 PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
00243 PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
00244 PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
00245 PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
00246 PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
00247 PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
00248 PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
00249 PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
00250 PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
00251 PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
00252 PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
00253 PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationNV;
```

```
00254 PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTableNV;
00255 PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancedNV;
00256 PFNGLCOVERFILLPATHNVPROC glCoverFillPathNV;
00257 PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancedNV;
00258 PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathNV;
00259 PFNGLCREATEBUFFERSPROC glCreateBuffers;
00260 PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
00261 PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
00262 PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
00263 PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
00264 PFNGLCREATEPROGRAMMPROC glCreateProgram;
00265 PFNGLCREATEQUERIESPROC glCreateQueries;
00266 PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
00267 PFNGLCREATESAMPLERSPROC glCreateSamplers;
00268 PFNGLCREATESHADERPROC glCreateShader;
00269 PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
00270 PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
00271 PFNGLCREATESTATESNVPROC glCreateStatesNV;
00272 PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCLeventARB;
00273 PFNGLCREATETEXTURESPROC glCreateTextures;
00274 PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
00275 PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
00276 PFNGLCULLFACEPROC glCullFace;
00277 PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
00278 PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
00279 PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
00280 PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
00281 PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
00282 PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
00283 PFNGLDELETEBUFFERSPROC glDeleteBuffers;
00284 PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
00285 PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
00286 PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
00287 PFNGLDELETEPATHSNVPROC glDeletePathsNV;
00288 PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
00289 PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
00290 PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
00291 PFNGLDELETEPROGRAMPROC glDeleteProgram;
00292 PFNGLDELETEQUERIESPROC glDeleteQueries;
00293 PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
00294 PFNGLDELETESAMPLERSPROC glDeleteSamplers;
00295 PFNGLDELETESHADERPROC glDeleteShader;
00296 PFNGLDELETESTATESNVPROC glDeleteStatesNV;
00297 PFNGLDELETESYNCPROC glDeleteSync;
00298 PFNGLDELETETEXTURESPROC glDeleteTextures;
00299 PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
00300 PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
00301 PFNGLDEPTHFUNCPROC glDepthFunc;
00302 PFNGLDEPTHMASKPROC glDepthMask;
00303 PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
00304 PFNGLDEPTHRANGEFPROC glDepthRangef;
00305 PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
00306 PFNGLDEPTHRANGEPROC glDepthRange;
00307 PFNGLDETACHSHADERPROC glDetachShader;
00308 PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
00309 PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
00310 PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
00311 PFNGLDISABLEIPROC glDisablei;
00312 PFNGLDISABLEPROC glDisable;
00313 PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexAttribArrayAttribEXT;
00314 PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexAttribArrayAttrib;
00315 PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexAttribArrayEXT;
00316 PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArrayAttrib;
00317 PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
00318 PFNGLDISPATCHCOMPUTEINDIRECTPROC glDispatchComputeIndirect;
00319 PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
00320 PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
00321 PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
00322 PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
00323 PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
00324 PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
00325 PFNGLDRAWARRAYSPROC glDrawArrays;
00326 PFNGLDRAWBUFFERPROC glDrawBuffer;
00327 PFNGLDRAWBUFFERSPROC glDrawBuffers;
00328 PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
00329 PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsNV;
00330 PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
00331 PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
00332 PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
00333 PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
00334 PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
00335 PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
00336 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC glDrawElementsInstancedBaseVertexBaseInstance;
00337 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
00338 PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
00339 PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
00340 PFNGLDRAWELEMENTSPROC glDrawElements;
```

```

00341 PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00342 PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00343 PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
00344 PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
00345 PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
00346 PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00347 PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00348 PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00349 PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00350 PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00351 PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00352 PFNGLENABLEIIPROC glEnablei;
00353 PFNGLENABLEPROC glEnable;
00354 PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexAttribArrayAttribEXT;
00355 PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00356 PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00357 PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArrayArray;
00358 PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
00359 PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
00360 PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
00361 PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
00362 PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
00363 PFNGLENDQUERYPROC glEndQuery;
00364 PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
00365 PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
00366 PFNGLFENCESYNCNPROC glFenceSync;
00367 PFNGLFINISHPROC glFinish;
00368 PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00369 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00370 PFNGLFLUSHMAPPEDNAMEDBUFFERPROC glFlushMappedNamedBufferRange;
00371 PFNGLFLUSHPROC glFlush;
00372 PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00373 PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00374 PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00375 PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00376 PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00377 PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00378 PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
00379 PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC glFramebufferSampleLocationsfvARB;
00380 PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
00381 PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00382 PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00383 PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00384 PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
00385 PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00386 PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
00387 PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00388 PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00389 PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
00390 PFNGLFRONTFACEPROC glFrontFace;
00391 PFNGLGENBUFFERSPROC glGenBuffers;
00392 PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
00393 PFNGLGENERATEMULTITEMPMAPEXTPROC glGenerateMultiTexMipmapEXT;
00394 PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
00395 PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
00396 PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
00397 PFNGLGENPATHSNVPROC glGenPathsNV;
00398 PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
00399 PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
00400 PFNGLGENQUERIESPROC glGenQueries;
00401 PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
00402 PFNGLGENSAMPLERSPROC glGenSamplers;
00403 PFNGLGENTEXTURESPROC glGenTextures;
00404 PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
00405 PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
00406 PFNGLGETACTIVEATOMCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
00407 PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
00408 PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
00409 PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
00410 PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
00411 PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
00412 PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
00413 PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
00414 PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
00415 PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
00416 PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
00417 PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
00418 PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
00419 PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
00420 PFNGLGETBOOLEANVPROC glGetBooleanv;
00421 PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00422 PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00423 PFNGLGETBUFFERPARAMETERUI64VNPROC glGetBufferParameterui64vNV;
00424 PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00425 PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00426 PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00427 PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;

```

```
00428 PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
00429 PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
00430 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00431 PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00432 PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00433 PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00434 PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00435 PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00436 PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
00437 PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00438 PFNGLGETDOUBLEVPROC glGetDoublev;
00439 PFNGLGETERRORPROC glGetError;
00440 PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
00441 PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
00442 PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
00443 PFNGLGETFLOATI_VPROC glGetFloati_v;
00444 PFNGLGETFLOATVPROC glGetFloatv;
00445 PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
00446 PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
00447 PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
00448 PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
00449 PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
00450 PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
00451 PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
00452 PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
00453 PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
00454 PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
00455 PFNGLGETINTEGER64VPROC glGetInteger64v;
00456 PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00457 PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00458 PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00459 PFNGLGETINTEGERUI64VNVPROC glGetIntegerui64vNV;
00460 PFNGLGETINTEGERVPROC glGetInterv;
00461 PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
00462 PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
00463 PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
00464 PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
00465 PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
00466 PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
00467 PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGenfvEXT;
00468 PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
00469 PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
00470 PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
00471 PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
00472 PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
00473 PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
00474 PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIiivEXT;
00475 PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
00476 PFNGLGETMULTITEXPARAMETERIVEXTPROC glGetMultiTexParameterivEXT;
00477 PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
00478 PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
00479 PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
00480 PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC glGetNamedBufferParameterui64vNV;
00481 PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
00482 PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
00483 PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
00484 PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
00485 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC glGetNamedFramebufferAttachmentParameterivEXT;
00486 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
00487 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
00488 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
00489 PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
00490 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00491 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00492 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIiivEXT;
00493 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00494 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00495 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00496 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00497 PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00498 PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00499 PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
00500 PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
00501 PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00502 PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00503 PFNGLGETNTEXIMAGEPROC glGetnTexImage;
00504 PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
00505 PFNGLGETNUNIFORMMDVPROC glGetnUniformmdv;
00506 PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
00507 PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00508 PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00509 PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
00510 PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
00511 PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00512 PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
00513 PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
00514 PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
```

```
00515 PFNGLGETOBJECTLABELPROC glGetObjectLabel;
00516 PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
00517 PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
00518 PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
00519 PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
00520 PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
00521 PFNGLGETPATHMETRICRANGENVPROC glGetPathMetricRangeNV;
00522 PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
00523 PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
00524 PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
00525 PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
00526 PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
00527 PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
00528 PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
00529 PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
00530 PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
00531 PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
00532 PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
00533 PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
00534 PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
00535 PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
00536 PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
00537 PFNGLGETPOINTERI_VEXTPROC glGetPointeri_vEXT;
00538 PFNGLGETPOINTERVPROC glGetPointerv;
00539 PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
00540 PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
00541 PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
00542 PFNGLGETPROGRAMIVPROC glGetProgramiv;
00543 PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
00544 PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
00545 PFNGLGETPROGRAMRESOURCEFVNVPROC glGetProgramResourcefvNV;
00546 PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00547 PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00548 PFNGLGETPROGRAMRESOURCELICATIONINDEXPROC glGetProgramResourceLocationIndex;
00549 PFNGLGETPROGRAMRESOURCELATIONPROC glGetProgramResourceLocation;
00550 PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00551 PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00552 PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
00553 PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00554 PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00555 PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00556 PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00557 PFNGLGETQUERYIVPROC glGetQueryiv;
00558 PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
00559 PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
00560 PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00561 PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00562 PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00563 PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00564 PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00565 PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
00566 PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
00567 PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
00568 PFNGLGETSHADERIVPROC glGetShaderiv;
00569 PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
00570 PFNGLGETSHADERSOURCEPROC glGetShaderSource;
00571 PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
00572 PFNGLGETSTRINGIPROC glGetStringi;
00573 PFNGLGETSTRINGPROC glGetString;
00574 PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
00575 PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
00576 PFNGLGETSYNCIVPROC glGetSynciv;
00577 PFNGLGETTEXIMAGEPROC glGetTexImage;
00578 PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00579 PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00580 PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00581 PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00582 PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00583 PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
00584 PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00585 PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
00586 PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00587 PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00588 PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00589 PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
00590 PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
00591 PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
00592 PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
00593 PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
00594 PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
00595 PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
00596 PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
00597 PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
00598 PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
00599 PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00600 PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00601 PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
```

```
00602 PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00603 PFNGLGETTRANSFORMFEEDBACKI64VPROC glGetTransformFeedbacki64v;
00604 PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00605 PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
00606 PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00607 PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00608 PFNGLGETUNIFORMDVPROC glGetUniformdv;
00609 PFNGLGETUNIFORMFVPROC glGetUniformfv;
00610 PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00611 PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
00612 PFNGLGETUNIFORMINDICESPROC glGetUniformLocationIndices;
00613 PFNGLGETUNIFORMIVPROC glGetUniformLocation;
00614 PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocationLocation;
00615 PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineui;
00616 PFNGLGETUNIFORMUI64VARBPROC glGetUniformLocationui64vARB;
00617 PFNGLGETUNIFORMUI64VNVPROC glGetUniformLocationui64vNV;
00618 PFNGLGETUNIFORMUIVPROC glGetUniformLocationui;
00619 PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00620 PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
00621 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00622 PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00623 PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00624 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00625 PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00626 PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribArraybdv;
00627 PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribArraybfv;
00628 PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribArrayIiv;
00629 PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribArrayIuiv;
00630 PFNGLGETVERTEXATTRIBLIVPROC glGetVertexAttribArraylivel;
00631 PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribArrayldv;
00632 PFNGLGETVERTEXATTRIBL64VNVPROC glGetVertexAttribArrayLi64vNV;
00633 PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribArrayLui64vARB;
00634 PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribArrayLui64vNV;
00635 PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribArrayPointerv;
00636 PFNGLGETVKPROCAADDRNVPROC glGetVkProcAddrNV;
00637 PFNGLHINTPROC glHint;
00638 PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00639 PFNGLINSETEVENTMARKEREXTPROC glInsertEventMarkerEXT;
00640 PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00641 PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
00642 PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
00643 PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
00644 PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
00645 PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
00646 PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebufer;
00647 PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00648 PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00649 PFNGLISBUFFERPROC glIsBuffer;
00650 PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
00651 PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
00652 PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00653 PFNGLISENABLEDIPROC glIsEnabledi;
00654 PFNGLISENABLEDPROC glIsEnabled;
00655 PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00656 PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
00657 PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
00658 PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00659 PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00660 PFNGLISPATHNVPROC glIsPathNV;
00661 PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00662 PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00663 PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00664 PFNGLISPROGRAMPROC glIsProgram;
00665 PFNGLISQUERYPROC glIsQuery;
00666 PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00667 PFNGLISSAMPLERPROC glIsSampler;
00668 PFNGLISSHADERPROC glIsShader;
00669 PFNGLISSTATENVPROC glIsStateNV;
00670 PFNGLISSYNCNCPROC glIsSync;
00671 PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00672 PFNGLISTTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00673 PFNGLISTTEXTUREPROC glIsTexture;
00674 PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00675 PFNGLISVERTEXARRAYPROC glIsVertexArray;
00676 PFNGLLABELOBJECTTEXTPROC glLabelObjectEXT;
00677 PFNGLLINKNEWWIDTHPROC glLineWidth;
00678 PFNGLLINKPROGRAMPROC glLinkProgram;
00679 PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
00680 PFNGLLOGICOPPROC glLogicOp;
00681 PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
00682 PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
00683 PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
00684 PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
00685 PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
00686 PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
00687 PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
00688 PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
```

```

00689 PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
00690 PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
00691 PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
00692 PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
00693 PFNGLMAPBUFFERPROC glMapBuffer;
00694 PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
00695 PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
00696 PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
00697 PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
00698 PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
00699 PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00700 PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
00701 PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
00702 PFNGLMATRIXLOADDEXTPROC glMatrixLoadDEXT;
00703 PFNGLMATRIXLOADFEXTPROC glMatrixLoadFEXT;
00704 PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00705 PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
00706 PFNGLMATRIXLOADTRANPOSEDEXTPROC glMatrixLoadTransposedEXT;
00707 PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
00708 PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
00709 PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
00710 PFNGLMATRIXMULTDEXTPROC glMatrixMultDEXT;
00711 PFNGLMATRIXMULTFEXTPROC glMatrixMultFEXT;
00712 PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
00713 PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
00714 PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefEXT;
00715 PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
00716 PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
00717 PFNGLMATRIXPUSHEXTPROC glMatrixPushEXT;
00718 PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedEXT;
00719 PFNGLMATRIXROTATEFEXTPROC glMatrixRotatefEXT;
00720 PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
00721 PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
00722 PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
00723 PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
00724 PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
00725 PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
00726 PFNGLMEMORYBARRIERPROC glMemoryBarrier;
00727 PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
00728 PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
00729 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
00730 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
00731 PFNGLMULTIDRAWARRAYSINDIRECTCOUNTRARBPROC glMultiDrawArraysIndirectCountARB;
00732 PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00733 PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
00734 PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00735 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00736 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00737 PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTRARBPROC glMultiDrawElementsIndirectCountARB;
00738 PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00739 PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00740 PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00741 PFNGLMULTITEXCOORDPOINTEREXTPROC glMultiTexCoordPointerEXT;
00742 PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00743 PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00744 PFNGLMULTITEXENVIEVTEXTPROC glMultiTexEnvieEXT;
00745 PFNGLMULTITEXENVIVEVTEXTPROC glMultiTexEnvivEXT;
00746 PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00747 PFNGLMULTITEXGENDVEXTPROC glMultiTexGendvEXT;
00748 PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
00749 PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
00750 PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
00751 PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
00752 PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
00753 PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
00754 PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
00755 PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
00756 PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
00757 PFNGLMULTITEXPARAMETERIEVTEXTPROC glMultiTexParameteriEXT;
00758 PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
00759 PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
00760 PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterivEXT;
00761 PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
00762 PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
00763 PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
00764 PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
00765 PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
00766 PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
00767 PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
00768 PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
00769 PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
00770 PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
00771 PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubDataEXT;
00772 PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
00773 PFNGLNAMEDCOPYSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
00774 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
00775 PFNGLNAMEDFRAMEBUFFERBUFFERSPROC glNamedFramebufferDrawBuffers;

```

```

00776 PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
00777 PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC glNamedFramebufferParameteri;
00778 PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
00779 PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00780 PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00781 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARPROC glNamedFramebufferSampleLocationsfvARB;
00782 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
00783 PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00784 PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00785 PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00786 PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00787 PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00788 PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00789 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00790 PFNGLNAMEDFRAMEBUFFERTEXTUREPROC glNamedFramebufferTexture;
00791 PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00792 PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00793 PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
00794 PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00795 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
00796 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00797 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00798 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uvEXT;
00799 PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00800 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4VEXTPROC glNamedProgramLocalParametersI4iveEXT;
00801 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
00802 PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00803 PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00804 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00805 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00806 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00807 PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00808 PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
00809 PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00810 PFNGLOBJECTLABELPROC glObjectLabel;
00811 PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
00812 PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
00813 PFNGLPATCHPARAMETERIPROC glPatchParameteri;
00814 PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00815 PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00816 PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00817 PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00818 PFNGLPATHGLYPHINDEXEXARRAYNVPROC glPathGlyphIndexArrayNV;
00819 PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
00820 PFNGLPATHGLYPHRANGEENVPROC glPathGlyphRangeNV;
00821 PFNGLPATHGLYPHSNV;
00822 PFNGLPATHMEMORYGLYPHINDEXNVPROC glPathMemoryGlyphIndexArrayNV;
00823 PFNGLPATHPARAMETERFNVPROC glPathParameterfvNV;
00824 PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00825 PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
00826 PFNGLPATHPARAMETERIRINVPROC glPathParameterivNV;
00827 PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
00828 PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00829 PFNGLPATHSTRINGNVPROC glPathStringNV;
00830 PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00831 PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00832 PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00833 PFNGLPIXELSTOREFPROC glPixelStoref;
00834 PFNGLPIXELSTOREIPROC glPixelStorei;
00835 PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
00836 PFNGLPOINTPARAMETERERFPROC glPointParameterf;
00837 PFNGLPOINTPARAMETERERFVPROC glPointParameterfv;
00838 PFNGLPOINTPARAMETERIPROC glPointParameteri;
00839 PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00840 PFNGLPOINTSIZEPROC glPointSize;
00841 PFNGLPOLYGONMODEPROC glPolygonMode;
00842 PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00843 PFNGLPOLYGONOFFSETPROC glPolygonOffset;
00844 PFNGLPOPODDEBUGGROUPPROC glPopDebugGroup;
00845 PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00846 PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
00847 PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
00848 PFNGLPROGRAMBINARYPROC glProgramBinary;
00849 PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
00850 PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
00851 PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
00852 PFNGLPROGRAMUNIFORM1DINDEXTPROC glProgramUniform1dEXT;
00853 PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
00854 PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
00855 PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1d;
00856 PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
00857 PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
00858 PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
00859 PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
00860 PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
00861 PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;

```

```
00862 PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
00863 PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
00864 PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniform1iEXT;
00865 PFNGLPROGRAMUNIFORM1IPROC glProgramUniform1i;
00866 PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
00867 PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
00868 PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
00869 PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64NV;
00870 PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
00871 PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
00872 PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
00873 PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
00874 PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
00875 PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
00876 PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
00877 PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
00878 PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
00879 PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
00880 PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
00881 PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
00882 PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
00883 PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
00884 PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
00885 PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64NV;
00886 PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
00887 PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
00888 PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
00889 PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
00890 PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
00891 PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
00892 PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
00893 PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64NV;
00894 PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
00895 PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
00896 PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
00897 PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
00898 PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
00899 PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
00900 PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
00901 PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
00902 PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
00903 PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
00904 PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
00905 PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
00906 PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
00907 PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
00908 PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
00909 PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64NV;
00910 PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
00911 PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
00912 PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
00913 PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
00914 PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
00915 PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
00916 PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
00917 PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64NV;
00918 PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
00919 PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
00920 PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
00921 PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
00922 PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
00923 PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
00924 PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
00925 PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
00926 PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
00927 PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
00928 PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
00929 PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
00930 PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
00931 PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
00932 PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
00933 PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64NV;
00934 PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
00935 PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
00936 PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
00937 PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
00938 PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
00939 PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
00940 PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
00941 PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64NV;
00942 PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
00943 PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64NV;
00944 PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
00945 PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
00946 PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
00947 PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
00948 PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
```

```
00949 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
00950 PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
00951 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64vNV;
00952 PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dEXT;
00953 PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
00954 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
00955 PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
00956 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
00957 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
00958 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00959 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
00960 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
00961 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
00962 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
00963 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
00964 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
00965 PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dv;
00966 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
00967 PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
00968 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
00969 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dv;
00970 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
00971 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
00972 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
00973 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
00974 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
00975 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
00976 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dvEXT;
00977 PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dv;
00978 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
00979 PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
00980 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
00981 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dv;
00982 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
00983 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
00984 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
00985 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dv;
00986 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
00987 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
00988 PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
00989 PFNGLPROGRAMUNIFORMUI64VNVPROC glProgramUniformui64vNV;
00990 PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
00991 PFNGLPUSHCLIENTATTRIBDEFAULTTEXTPROC glPushClientAttribDefaultEXT;
00992 PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
00993 PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
00994 PFNGLQUERYCOUNTERPROC glQueryCounter;
00995 PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
00996 PFNGLREADBUFFERPROC glReadBuffer;
00997 PFNGLREADNPPIXELSARBPROC glReadnPixelsARB;
00998 PFNGLREADNPPIXELSPROC glReadnPixels;
00999 PFNGLREADPIXELSPROC glReadPixels;
01000 PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
01001 PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
01002 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
01003 PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
01004 PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
01005 PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
01006 PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
01007 PFNGLSAMPLEMASKIPROC glSampleMaski;
01008 PFNGLSAMPLERPARAMETERFPROC glSamplerParameterf;
01009 PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
01010 PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIiv;
01011 PFNGLSAMPLERPARAMETERIPROC glSamplerParameteri;
01012 PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
01013 PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameteriv;
01014 PFNGLSCISSORARRAYVPROC glScissorArrayv;
01015 PFNGLSCISSORINDEXEDPROC glScissorIndexed;
01016 PFNGLSCISSORINDEXEDVPROC glScissorIndexeddv;
01017 PFNGLSCISSORPROC glScissor;
01018 PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
01019 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
01020 PFNGLSHADERBINARYPROC glShaderBinary;
01021 PFNGLSHADERSOURCEPROC glShaderSource;
01022 PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
01023 PFNGLSIGNALKVFENCENVPROC glSignalVkFenceNV;
01024 PFNGLSIGNALKSEMAPHORENVPROC glSignalVkSemaphoreNV;
01025 PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
01026 PFNGLSTATECAPTURENVPROC glStateCaptureNV;
01027 PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
01028 PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
01029 PFNGLSTENCILFUNCPROC glStencilFunc;
01030 PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
01031 PFNGLSTENCILMASKPROC glStencilMask;
01032 PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
01033 PFNGLSTENCILLOPPROC glStencilOp;
01034 PFNGLSTENCILLOPSEPARATEPROC glStencilOpSeparate;
01035 PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
```

```
01036 PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
01037 PFNGLSTENCILLTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01038 PFNGLSTENCILLTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01039 PFNGLSTENCILLTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01040 PFNGLSTENCILLTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
01041 PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
01042 PFNGLTEXBUFFERARBPROC glTexBufferARB;
01043 PFNGLTEXBUFFERPROC glTexBuffer;
01044 PFNGLTEXBUFFERRANGEPROC glTexBufferSize;
01045 PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
01046 PFNGLTEXIMAGE1DPROC glTexImage1D;
01047 PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
01048 PFNGLTEXIMAGE2DPROC glTexImage2D;
01049 PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
01050 PFNGLTEXIMAGE3DPROC glTexImage3D;
01051 PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
01052 PFNGLTEXPARAMETERFPROC glTexParameterf;
01053 PFNGLTEXPARAMETERFVPROC glTexParameterfv;
01054 PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
01055 PFNGLTEXPARAMETERIIPROC glTexParameterIi;
01056 PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01057 PFNGLTEXPARAMETERIIVPROC glTexParameteriv;
01058 PFNGLTEXSTORAGE1DPROC glTexStorage1D;
01059 PFNGLTEXSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
01060 PFNGLTEXSTORAGE2DPROC glTexStorage2D;
01061 PFNGLTEXSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
01062 PFNGLTEXSTORAGE3DPROC glTexStorage3D;
01063 PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01064 PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01065 PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01066 PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
01067 PFNGLTEXTUREBARRIERPROC glTextureBarrier;
01068 PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
01069 PFNGLTEXTUREBUFFERPROC glTextureBuffer;
01070 PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01071 PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01072 PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
01073 PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
01074 PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
01075 PFNGLTEXTUREPAGECOMMITMENTEXTPROC glTexturePageCommitmentEXT;
01076 PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
01077 PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
01078 PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
01079 PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
01080 PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
01081 PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIivEXT;
01082 PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiv;
01083 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01084 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01085 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01086 PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterivEXT;
01087 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01088 PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
01089 PFNGLTEXTURESTORAGE1DEXTPROC glTextureStorage1DEXT;
01090 PFNGLTEXTURESTORAGE1DPROC glTextureStorage1D;
01091 PFNGLTEXTURESTORAGE2DEXTPROC glTextureStorage2DEXT;
01092 PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
01093 PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
01094 PFNGLTEXTURESTORAGE2DPROC glTextureStorage2D;
01095 PFNGLTEXTURESTORAGE3DEXTPROC glTextureStorage3DEXT;
01096 PFNGLTEXTURESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
01097 PFNGLTEXTURESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
01098 PFNGLTEXTURESTORAGE3DPROC glTextureStorage3D;
01099 PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
01100 PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
01101 PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
01102 PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
01103 PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
01104 PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
01105 PFNGLTEXTUREVIEWPROC glTextureView;
01106 PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
01107 PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
01108 PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
01109 PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
01110 PFNGLUNIFORM1DPROC glUniform1d;
01111 PFNGLUNIFORM1DVPROC glUniform1dv;
01112 PFNGLUNIFORM1FPROC glUniform1f;
01113 PFNGLUNIFORM1FVPROC glUniform1fv;
01114 PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
01115 PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
01116 PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
01117 PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
01118 PFNGLUNIFORM1IPROC glUniform1i;
01119 PFNGLUNIFORM1IVPROC glUniform1iv;
01120 PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01121 PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01122 PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
```

```
01123 PFNGLUNIFORM1UI64NVPROC glUniformui64vNV;
01124 PFNGLUNIFORM1UIPROC glUniformui;
01125 PFNGLUNIFORM1UIVPROC glUniformuiv;
01126 PFNGLUNIFORM2DPROC glUniform2d;
01127 PFNGLUNIFORM2DVPROC glUniform2dv;
01128 PFNGLUNIFORM2FPROC glUniform2f;
01129 PFNGLUNIFORM2FVPROC glUniform2fv;
01130 PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
01131 PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
01132 PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
01133 PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
01134 PFNGLUNIFORM2IPROC glUniform2i;
01135 PFNGLUNIFORM2IVPROC glUniform2iv;
01136 PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01137 PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01138 PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
01139 PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
01140 PFNGLUNIFORM2UIPROC glUniform2ui;
01141 PFNGLUNIFORM2UIVPROC glUniform2uiv;
01142 PFNGLUNIFORM3DPROC glUniform3d;
01143 PFNGLUNIFORM3DVPROC glUniform3dv;
01144 PFNGLUNIFORM3FPROC glUniform3f;
01145 PFNGLUNIFORM3FVPROC glUniform3fv;
01146 PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
01147 PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
01148 PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
01149 PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
01150 PFNGLUNIFORM3IPROC glUniform3i;
01151 PFNGLUNIFORM3IVPROC glUniform3iv;
01152 PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01153 PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01154 PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
01155 PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
01156 PFNGLUNIFORM3UIPROC glUniform3ui;
01157 PFNGLUNIFORM3UIVPROC glUniform3uiv;
01158 PFNGLUNIFORM4DPROC glUniform4d;
01159 PFNGLUNIFORM4DVPROC glUniform4dv;
01160 PFNGLUNIFORM4FPROC glUniform4f;
01161 PFNGLUNIFORM4FVPROC glUniform4fv;
01162 PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
01163 PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
01164 PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
01165 PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
01166 PFNGLUNIFORM4IPROC glUniform4i;
01167 PFNGLUNIFORM4IVPROC glUniform4iv;
01168 PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01169 PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01170 PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
01171 PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
01172 PFNGLUNIFORM4UIPROC glUniform4ui;
01173 PFNGLUNIFORM4UIVPROC glUniform4uiv;
01174 PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01175 PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01176 PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01177 PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
01178 PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
01179 PFNGLUNIFORMMATRIX2DPROC glUniformMatrix2dv;
01180 PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
01181 PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
01182 PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
01183 PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
01184 PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
01185 PFNGLUNIFORMMATRIX3DPROC glUniformMatrix3dv;
01186 PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
01187 PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
01188 PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
01189 PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
01190 PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
01191 PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
01192 PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
01193 PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
01194 PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
01195 PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
01196 PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
01197 PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
01198 PFNGLUNIFORMUI64NVPROC glUniformui64NV;
01199 PFNGLUNIFORMUI64VNVPROC glUniformui64vNV;
01200 PFNGLUNMAPBUFFERPROC glUnmapBuffer;
01201 PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
01202 PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
01203 PFNGLUSEPROGRAMPROC glUseProgram;
01204 PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
01205 PFNGLUSESHEADERPROGRAMEXTPROC glUseShaderProgramEXT;
01206 PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
01207 PFNGLVALIDATEPROGRAMPROC glValidateProgram;
01208 PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
01209 PFNGLVERTEXARRAYFORMATPROC glVertexArrayAttribFormat;
```

```

01210 PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribIFormat;
01211 PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribIFormat;
01212 PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01213 PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01214 PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
01215 PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01216 PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01217 PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01218 PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
01219 PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01220 PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
01221 PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01222 PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
01223 PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
01224 PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01225 PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01226 PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01227 PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01228 PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
01229 PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
01230 PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01231 PFNGLVERTEXARRAYVERTEXATTRIBVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01232 PFNGLVERTEXARRAYVERTEXBUFFERRPROC glVertexArrayVertexBuffer;
01233 PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01234 PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
01235 PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
01236 PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
01237 PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
01238 PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
01239 PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
01240 PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
01241 PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
01242 PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
01243 PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
01244 PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
01245 PFNGLVERTEXATTRIB2SPROC glVertexAttrib2s;
01246 PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
01247 PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
01248 PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
01249 PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
01250 PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
01251 PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
01252 PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
01253 PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
01254 PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
01255 PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
01256 PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
01257 PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
01258 PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
01259 PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4nbv;
01260 PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4niv;
01261 PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4nsv;
01262 PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4nub;
01263 PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4nubv;
01264 PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4nui;
01265 PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4nusv;
01266 PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
01267 PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
01268 PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
01269 PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
01270 PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
01271 PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01272 PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01273 PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01274 PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01275 PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01276 PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01277 PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
01278 PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
01279 PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
01280 PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01281 PFNGLVERTEXATTRIBI2IVPROC glVertexAttribI2iv;
01282 PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
01283 PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
01284 PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
01285 PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
01286 PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01287 PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
01288 PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01289 PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
01290 PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
01291 PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
01292 PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
01293 PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
01294 PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
01295 PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
01296 PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;

```

```

01297 PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
01298 PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
01299 PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
01300 PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
01301 PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
01302 PFNGLVERTEXATTRIBL1I64VNPROC glVertexAttribL1i64vNV;
01303 PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
01304 PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
01305 PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
01306 PFNGLVERTEXATTRIBL1UI64VNPROC glVertexAttribL1ui64vNV;
01307 PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
01308 PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
01309 PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
01310 PFNGLVERTEXATTRIBL2I64VNPROC glVertexAttribL2i64vNV;
01311 PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
01312 PFNGLVERTEXATTRIBL2UI64VNPROC glVertexAttribL2ui64vNV;
01313 PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01314 PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
01315 PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
01316 PFNGLVERTEXATTRIBL3I64VNPROC glVertexAttribL3i64vNV;
01317 PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
01318 PFNGLVERTEXATTRIBL3UI64VNPROC glVertexAttribL3ui64vNV;
01319 PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01320 PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
01321 PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
01322 PFNGLVERTEXATTRIBL4I64VNPROC glVertexAttribL4i64vNV;
01323 PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
01324 PFNGLVERTEXATTRIBL4UI64VNPROC glVertexAttribL4ui64vNV;
01325 PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
01326 PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
01327 PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
01328 PFNGLVERTEXATTRIBBP1UIPROC glVertexAttribPlui;
01329 PFNGLVERTEXATTRIBBP1UIVPROC glVertexAttribPluiv;
01330 PFNGLVERTEXATTRIBBP2UIPROC glVertexAttribP2ui;
01331 PFNGLVERTEXATTRIBBP2UIVPROC glVertexAttribP2uiv;
01332 PFNGLVERTEXATTRIBBP3UIPROC glVertexAttribP3ui;
01333 PFNGLVERTEXATTRIBBP3UIVPROC glVertexAttribP3uiv;
01334 PFNGLVERTEXATTRIBBP4UIPROC glVertexAttribP4ui;
01335 PFNGLVERTEXATTRIBBP4UIVPROC glVertexAttribP4uiv;
01336 PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
01337 PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
01338 PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
01339 PFNGLVIEWPORTARRAYVPROC glVertexportArrayv;
01340 PFNGLVIEWPORTINDEXEDFPROC glVertexportIndexedf;
01341 PFNGLVIEWPORTINDEXEDFVPROC glVertexportIndexedfv;
01342 PFNGLVIEWPORTPOSITIONWSCALENVPROC glVertexportPositionWScaleNV;
01343 PFNGLVIEWPORTPROC glVertexport;
01344 PFNGLVIEWPORTSWIZZLENVPROC glVertexportSwizzleNV;
01345 PFNGLWAITSYNCPROC glWaitSync;
01346 PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01347 PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01348 PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01349 #endif
01350
01352
01354 GLint gg::ggBufferAlignment(0);
01355
01356 //
01357 // ゲームグラフィックス特論の都合にもとづく初期化
01358 //
01359 void gg::ggInit()
01360 {
01361     // すでにこの関数が実行されていたら以降の処理を行わない
01362     if (ggBufferAlignment) return;
01363
01364     // macOS 以外で OpenGL 3.2 以降の API を取得する
01365 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
01366     glActiveProgramEXT = PFNGLACTIVEPROGRAMEXTPROC(glfwGetProcAddress("glActiveProgramEXT"));
01367     glActiveShaderProgram = PFNGLACTIVESHADERPROGRAMPROC(glfwGetProcAddress("glActiveShaderProgram"));
01368     glActiveTexture = PFNGLACTIVETEXTUREPROC(glfwGetProcAddress("glActiveTexture"));
01369     glApplyFramebufferAttachmentCMAAINTEL =
01370         PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC(glfwGetProcAddress("glApplyFramebufferAttachmentCMAAINTEL"));
01371     glAttachShader = PFNGLATTACHSHADERPROC(glfwGetProcAddress("glAttachShader"));
01372     glBeginConditionalRender =
01373         PFNGLBEGINCONDITIONALRENDERPROC(glfwGetProcAddress("glBeginConditionalRender"));
01374     glBeginConditionalRenderNV =
01375         PFNGLBEGINCONDITIONALRENDERNVPROC(glfwGetProcAddress("glBeginConditionalRenderNV"));
01376     glBeginPerfMonitorAMD = PFNGLBEGINPERFMONITORAMDPROC(glfwGetProcAddress("glBeginPerfMonitorAMD"));
01377     glBeginPerfQueryINTEL = PFNGLBEGINPERFQUERYINTELPROC(glfwGetProcAddress("glBeginPerfQueryINTEL"));
01378     glBeginQuery = PFNGLBEGINQUERYPROC(glfwGetProcAddress("glBeginQuery"));
01379     glBeginQueryIndexed =
01380         PFNGLBEGINQUERYINDEXEDPROC(glfwGetProcAddress("glBeginQueryIndexed"));
01381     glBeginTransformFeedback =
01382         PFNGLBEGINTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBeginTransformFeedback"));
01383     glBindAttribLocation = PFNGLBINDATTRIBLOCATIONPROC(glfwGetProcAddress("glBindAttribLocation"));
01384     glBindBuffer = PFNGLBINDBUFFERPROC(glfwGetProcAddress("glBindBuffer"));
01385     glBindBufferBase = PFNGLBINDBUFFERBASEPROC(glfwGetProcAddress("glBindBufferBase"));
01386     glBindBufferRange = PFNGLBINDBUFFERRANGEPROC(glfwGetProcAddress("glBindBufferRange"));

```

```

01382 glBindBuffersBase = PFNGLBINDBUFFERSBASEPROC(glfwGetProcAddress("glBindBuffersBase"));
01383 glBindBuffersRange = PFNGLBINDBUFFERSRANGEPROC(glfwGetProcAddress("glBindBuffersRange"));
01384 glBindFragDataLocation =
PFNGLBINDFRAGDATALOCATIONPROC(glfwGetProcAddress("glBindFragDataLocation"));
01385 glBindFragDataLocationIndexed =
PFNGLBINDFRAGDATALOCATIONINDEXEDPROC(glfwGetProcAddress("glBindFragDataLocationIndexed"));
01386 glBindFramebuffer = PFNGLBINDFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
01387 glBindImageTexture = PFNGLBINDIMAGETEXTUREPROC(glfwGetProcAddress("glBindImageTexture"));
01388 glBindImageTextures = PFNGLBINDIMAGETEXTURESPROC(glfwGetProcAddress("glBindImageTextures"));
01389 glBindMultiTextureEXT = PFNGLBINDMULTITEXTUREEXTPROC(glfwGetProcAddress("glBindMultiTextureEXT"));
01390 glBindProgramPipeline = PFNGLBINDPROGRAMPIPELINEPROC(glfwGetProcAddress("glBindProgramPipeline"));
01391 glBindRenderbuffer = PFNGLBINDRENDERBUFFERPROC(glfwGetProcAddress("glBindRenderbuffer"));
01392 glBindSampler = PFNGLBINDSAMPLERPROC(glfwGetProcAddress("glBindSampler"));
01393 glBindSamplers = PFNGLBINDSAMPLERSPROC(glfwGetProcAddress("glBindSamplers"));
01394 glBindTexture = PFNGLBINDTEXTUREPROC(glfwGetProcAddress("glBindTexture"));
01395 glBindTextureUnit = PFNGLBINDTEXTUREUNITPROC(glfwGetProcAddress("glBindTextureUnit"));
01396 glBindTextures = PFNGLBINDTEXTURESPROC(glfwGetProcAddress("glBindTextures"));
01397 glBindTransformFeedback =
PFNGLBINDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBindTransformFeedback"));
01398 glBindVertexArray = PFNGLBINDVERTEXARRAYPROC(glfwGetProcAddress("glBindVertexArray"));
01399 glBindVertexBuffer = PFNGLBINDVERTEXBUFFERPROC(glfwGetProcAddress("glBindVertexBuffer"));
01400 glBindVertexBuffers = PFNGLBINDVERTEXBUFFERSPROC(glfwGetProcAddress("glBindVertexBuffers"));
01401 glBindBarrierKHR = PFNGLBLENDBARRIERKHRPROC(glfwGetProcAddress("glBlendBarrierKHR"));
01402 glBindBarrierNV = PFNGLBLENDBARRIERNVPROC(glfwGetProcAddress("glBlendBarrierNV"));
01403 glBindColor = PFNGLBLENDCOLORPROC(glfwGetProcAddress("glBlendColor"));
01404 glBindEquation = PFNGLBLENDEQUATIONPROC(glfwGetProcAddress("glBlendEquation"));
01405 glBindEquationSeparate =
PFNGLBLENDEQUATIONSEPARATEPROC(glfwGetProcAddress("glBlendEquationSeparate"));
01406 glBindEquationSeparatei =
PFNGLBLENDEQUATIONSEPARATEIPROC(glfwGetProcAddress("glBlendEquationSeparatei"));
01407 glBindEquationSeparateiARB =
PFNGLBLENDEQUATIONSEPARATEIARBPROC(glfwGetProcAddress("glBlendEquationSeparateiARB"));
01408 glBindEquationi = PFNGLBLENDEQUATIONIPROC(glfwGetProcAddress("glBlendEquationi"));
01409 glBindEquationiARB = PFNGLBLENDEQUATIONIARBPROC(glfwGetProcAddress("glBlendEquationiARB"));
01410 glBindFunc = PFNGLBLENDFUNCPROC(glfwGetProcAddress("glBlendFunc"));
01411 glBindFuncSeparate = PFNGLBLENDFUNCSeparatePROC(glfwGetProcAddress("glBlendFuncSeparate"));
01412 glBindFuncSeparatei = PFNGLBLENDFUNCSeparateIPROC(glfwGetProcAddress("glBlendFuncSeparatei"));
01413 glBindFuncSeparateiARB =
PFNGLBLENDFUNCSEPARATEIARBPROC(glfwGetProcAddress("glBlendFuncSeparateiARB"));
01414 glBindFunci = PFNGLBLENDFUNCIPROC(glfwGetProcAddress("glBlendFunci"));
01415 glBindFunciARB = PFNGLBLENDFUNCIARBPROC(glfwGetProcAddress("glBlendFunciARB"));
01416 glBindParameteriNV = PFNGLBLENDPARAMETERINVPROC(glfwGetProcAddress("glBlendParameteriNV"));
01417 glBindFramebuffer = PFNGLBLITFRAMEBUFFERPROC(glfwGetProcAddress("glBlitFramebuffer"));
01418 glBindNamedFramebuffer =
PFNGLBLITNAMEDFRAMEBUFFERPROC(glfwGetProcAddress("glBlitNamedFramebuffer"));
01419 glBindBufferAddressRangeNV =
PFNGLBUFFERADDRESSRANGENVPROC(glfwGetProcAddress("glBufferAddressRangeNV"));
01420 glBindBufferData = PFNGLBUFFERDATAPROC(glfwGetProcAddress("glBufferData"));
01421 glBindBufferPageCommitmentARB =
PFNGLBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glBufferPageCommitmentARB"));
01422 glBindBufferStorage = PFNGLBUFFERSTORAGEPROC(glfwGetProcAddress("glBufferStorage"));
01423 glBindBufferSubData = PFNGLBUFFERSUBDATAPROC(glfwGetProcAddress("glBufferSubData"));
01424 glBindCommandListNV = PFNGLCALLCOMMANDLISTNVPROC(glfwGetProcAddress("glCallCommandListNV"));
01425 glBindCheckFramebufferStatus =
PFNGLCHECKFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckFramebufferStatus"));
01426 glBindCheckNamedFramebufferStatus =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckNamedFramebufferStatus"));
01427 glBindCheckNamedFramebufferStatusEXT =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC(glfwGetProcAddress("glCheckNamedFramebufferStatusEXT"));
01428 glBindColor = PFNGLCLAMPCOLORPROC(glfwGetProcAddress("glClampColor"));
01429 glBindClear = PFNGLCLEARPROC(glfwGetProcAddress("glClear"));
01430 glBindClearBufferData = PFNGLCLEARBUFFERDATAPROC(glfwGetProcAddress("glClearBufferData"));
01431 glBindClearBufferSubData = PFNGLCLEARBUFFERSUBDATAPROC(glfwGetProcAddress("glClearBufferSubData"));
01432 glBindClearBufferfi = PFNGLCLEARBUFFERFIPROC(glfwGetProcAddress("glClearBufferfi"));
01433 glBindClearBufferfv = PFNGLCLEARBUFFERFVPROC(glfwGetProcAddress("glClearBufferfv"));
01434 glBindClearBufferiv = PFNGLCLEARBUFFERIVPROC(glfwGetProcAddress("glClearBufferiv"));
01435 glBindClearBufferuiv = PFNGLCLEARBUFFERUIVPROC(glfwGetProcAddress("glClearBufferuiv"));
01436 glBindClearColor = PFNGLCLEARCOLORPROC(glfwGetProcAddress("glClearColor"));
01437 glBindClearDepth = PFNGLCLEARDEPTHPROC(glfwGetProcAddress("glClearDepth"));
01438 glBindClearDepthf = PFNGLCLEARDEPTHFPROC(glfwGetProcAddress("glClearDepthf"));
01439 glBindClearNamedBufferData =
PFNGLCLEARNAMEDBUFFERDATAPROC(glfwGetProcAddress("glClearNamedBufferData"));
01440 glBindClearNamedBufferDataEXT =
PFNGLCLEARNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataEXT"));
01441 glBindClearNamedBufferSubData =
PFNGLCLEARNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glClearNamedBufferSubData"));
01442 glBindClearNamedBufferSubDataEXT =
PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferSubDataEXT"));
01443 glBindClearNamedFramebufferfi =
PFNGLCLEARNAMEDFRAMEBUFFERFIPROC(glfwGetProcAddress("glClearNamedFramebufferfi"));
01444 glBindClearNamedFramebufferfv =
PFNGLCLEARNAMEDFRAMEBUFFERFVPROC(glfwGetProcAddress("glClearNamedFramebufferfv"));
01445 glBindClearNamedFramebufferiv =
PFNGLCLEARNAMEDFRAMEBUFFERIVPROC(glfwGetProcAddress("glClearNamedFramebufferiv"));
01446 glBindClearNamedFramebufferuiv =
PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC(glfwGetProcAddress("glClearNamedFramebufferuiv"));
01447 glBindClearStencil = PFNGLCLEARSTENCILPROC(glfwGetProcAddress("glClearStencil"));

```

```

01448 glClearTexImage = PFNGLCLEARTEXIMAGEPROC(glfwGetProcAddress("glClearTexImage"));
01449 glClearTexSubImage = PFNGLCLEARTEXSUBIMAGEPROC(glfwGetProcAddress("glClearTexSubImage"));
01450 glClientAttribDefaultEXT =
01451 PFNGLCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glClientAttribDefaultEXT"));
01452 glClientWaitSync = PFNGLCLIENTWAITSYNCPROC(glfwGetProcAddress("glClientWaitSync"));
01453 glClipControl = PFNGLCLIPCONTROLPROC(glfwGetProcAddress("glClipControl"));
01454 glColorFormatNV = PFNGLCOLORFORMATNVPROC(glfwGetProcAddress("glColorFormatNV"));
01455 glColorMask = PFNGLCOLORMASKPROC(glfwGetProcAddress("glColorMask"));
01456 glColorMaski = PFNGLCOLORMASKIPROC(glfwGetProcAddress("glColorMaski"));
01457 glCommandListSegmentsNV =
01458 PFNGLCOMMANDLISTSEGMENTSNVPROC(glfwGetProcAddress("glCommandListSegmentsNV"));
01459 glCompileCommandListNV =
01460 PFNGLCOMPILECOMMANDLISTNVPROC(glfwGetProcAddress("glCompileCommandListNV"));
01461 glCompileShader = PFNGLCOMPILESHADERPROC(glfwGetProcAddress("glCompileShader"));
01462 glCompileShaderIncludeARB =
01463 PFNGLCOMPILESHADERINCLUDEARBPROC(glfwGetProcAddress("glCompileShaderIncludeARB"));
01464 glCompressedMultiTexImage1DEXT =
01465 PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage1DEXT"));
01466 glCompressedMultiTexImage2DEXT =
01467 PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage2DEXT"));
01468 glCompressedMultiTexImage3DEXT =
01469 PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage3DEXT"));
01470 glCompressedMultiTexSubImage1DEXT =
01471 PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage1DEXT"));
01472 glCompressedMultiTexSubImage2DEXT =
01473 PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage2DEXT"));
01474 glCompressedMultiTexSubImage3DEXT =
01475 PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage3DEXT"));
01476 glCompressedTexImage1D =
01477 PFNGLCOMPRESSEDTEXIMAGE1DPROC(glfwGetProcAddress("glCompressedTexImage1D"));
01478 glCompressedTexImage2D =
01479 PFNGLCOMPRESSEDTEXIMAGE2DPROC(glfwGetProcAddress("glCompressedTexImage2D"));
01480 glCompressedTexImage3D =
01481 PFNGLCOMPRESSEDTEXIMAGE3DPROC(glfwGetProcAddress("glCompressedTexImage3D"));
01482 glCompressedTexSubImage1D =
01483 PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTexSubImage1D"));
01484 glCompressedTexSubImage2D =
01485 PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTexSubImage2D"));
01486 glCompressedTexSubImage3D =
01487 PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTexSubImage3D"));
01488 glCompressedTextureImage1DEXT =
01489 PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureImage1DEXT"));
01490 glCompressedTextureImage2DEXT =
01491 PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureImage2DEXT"));
01492 glCompressedTextureImage3DEXT =
01493 PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureImage3DEXT"));
01494 glCompressedTextureSubImage1D =
01495 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTextureSubImage1D"));
01496 glCompressedTextureSubImage2D =
01497 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTextureSubImage2D"));
01498 glCompressedTextureSubImage3D =
01499 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTextureSubImage3D"));
01500 glConservativeRasterParameterfNV =
01501 PFNGLCONSERVATIVERASTERPARAMETERFNVPROC(glfwGetProcAddress("glConservativeRasterParameterfNV"));
01502 glConservativeRasterParameteriNV =
01503 PFNGLCONSERVATIVERASTERPARAMETERINVPROC(glfwGetProcAddress("glConservativeRasterParameteriNV"));
01504 glCopyBufferSubData = PFNGLCOPYBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyBufferSubData"));
01505 glCopyImageSubData = PFNGLCOPYIMAGESUBDATAPROC(glfwGetProcAddress("glCopyImageSubData"));
01506 glCopyMultiTexImage1DEXT =
01507 PFNGLCOPYMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage1DEXT"));
01508 glCopyMultiTexImage2DEXT =
01509 PFNGLCOPYMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage2DEXT"));
01510 glCopyMultiTexSubImage1DEXT =
01511 PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage1DEXT"));
01512 glCopyMultiTexSubImage2DEXT =
01513 PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage2DEXT"));
01514 glCopyMultiTexSubImage3DEXT =
01515 PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage3DEXT"));
01516 glCopyNamedBufferSubData =
01517 PFNGLCOPYNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyNamedBufferSubData"));
01518 glCopyPathNV = PFNGLCOPYPATHNVPROC(glfwGetProcAddress("glCopyPathNV"));
01519 glCopyTexImage1D = PFNGLCOPYTEXIMAGE1DPROC(glfwGetProcAddress("glCopyTexImage1D"));
01520 glCopyTexImage2D = PFNGLCOPYTEXIMAGE2DPROC(glfwGetProcAddress("glCopyTexImage2D"));
01521 glCopyTexSubImage1D = PFNGLCOPYTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCopyTexSubImage1D"));
01522 glCopyTexSubImage2D = PFNGLCOPYTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCopyTexSubImage2D"));
01523 glCopyTexSubImage3D = PFNGLCOPYTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCopyTexSubImage3D"));
01524 glCopyTextureImage1DEXT =
01525 PFNGLCOPYTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureImage1DEXT"));
01526 glCopyTextureImage2DEXT =
01527 PFNGLCOPYTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureImage2DEXT"));
01528 glCopyTextureSubImage1D =

```

```

01500 PFNGLCOPYTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCopyTextureSubImage1D"));
01500     glCopyTextureSubImage1DEXT =
01501 PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage1DEXT"));
01501     glCopyTextureSubImage2D =
01502 PFNGLCOPYTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCopyTextureSubImage2D"));
01502     glCopyTextureSubImage2DEXT =
01503 PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage2DEXT"));
01503     glCopyTextureSubImage3D =
01504 PFNGLCOPYTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCopyTextureSubImage3D"));
01504     glCopyTextureSubImage3DEXT =
01505 PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage3DEXT"));
01505     glCoverFillPathInstancedNV =
01506 PFNGLCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverFillPathInstancedNV"));
01506     glCoverFillPathNV = PFNGLCOVERFILLPATHNVPROC(glfwGetProcAddress("glCoverFillPathNV"));
01507     glCoverStrokePathInstancedNV =
01508 PFNGLCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverStrokePathInstancedNV"));
01508     glCoverStrokePathNV = PFNGLCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glCoverStrokePathNV"));
01509     glCoverageModulationNV =
01510 PFNGLCOVERAGEMODULATIONNNVPROC(glfwGetProcAddress("glCoverageModulationNV"));
01510     glCoverageModulationTableNV =
01511 PFNGLCOVERAGEMODULATIONTABLENNVPROC(glfwGetProcAddress("glCoverageModulationTableNV"));
01511     glCreateBuffers = PFNGLCREATEBUFFERSPROC(glfwGetProcAddress("glCreateBuffers"));
01512     glCreateCommandListsNV =
01513 PFNGLCREATECOMMANDLISTSNVPROC(glfwGetProcAddress("glCreateCommandListsNV"));
01513     glCreateFramebuffers = PFNGLCREATEFRAMEBUFFERSPROC(glfwGetProcAddress("glCreateFramebuffers"));
01514     glCreatePerfQueryINTEL =
01515 PFNGLCREATEPERFQUERYINTELPROC(glfwGetProcAddress("glCreatePerfQueryINTEL"));
01515     glCreateProgram = PFNGLCREATEPROGRAMPROC(glfwGetProcAddress("glCreateProgram"));
01516     glCreateProgramPipelines =
01517 PFNGLCREATEPROGRAMPIPELINESPROC(glfwGetProcAddress("glCreateProgramPipelines"));
01517     glCreateQueries = PFNGLCREATEQUERIESPROC(glfwGetProcAddress("glCreateQueries"));
01518     glCreateRenderbuffers = PFNGLCREATE RENDERBUFFERSPROC(glfwGetProcAddress("glCreateRenderbuffers"));
01519     glCreateSamplers = PFNGLCREATEAMPLERSPROC(glfwGetProcAddress("glCreateSamplers"));
01520     glCreateShader = PFNGLCREATESHADERPROC(glfwGetProcAddress("glCreateShader"));
01521     glCreateShaderProgramEXT =
01522 PFNGLCREATESHADERPROGRAMEXTPROC(glfwGetProcAddress("glCreateShaderProgramEXT"));
01522     glCreateShaderProgramv =
01523 PFNGLCREATESHADERPROGRAMVPROC(glfwGetProcAddress("glCreateShaderProgramv"));
01523     glCreateStatesNV = PFNGLCREATESTATESNVPROC(glfwGetProcAddress("glCreateStatesNV"));
01524     glCreateSyncFromCLeventARB =
01525 PFNGLCREATESYNCFROMCLEVENTARBPROC(glfwGetProcAddress("glCreateSyncFromCLeventARB"));
01525     glCreateTextures = PFNGLCREATETEXTURESPROC(glfwGetProcAddress("glCreateTextures"));
01526     glCreateTransformFeedbacks =
01527 PFNGLCREATETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glCreateTransformFeedbacks"));
01527     glCreateVertexArrays = PFNGLCREATEVERTEXARRAYSPROC(glfwGetProcAddress("glCreateVertexArrays"));
01528     glCullFace = PFNGLCULLFACEPROC(glfwGetProcAddress("glCullFace"));
01529     glDebugMessageCallback =
01530 PFNGLDEBUGMESSAGECALLBACKPROC(glfwGetProcAddress("glDebugMessageCallback"));
01530     glDebugMessageCallbackARB =
01531 PFNGLDEBUGMESSAGECALLBACKARBPROC(glfwGetProcAddress("glDebugMessageCallbackARB"));
01531     glDebugMessageControl = PFNGLDEBUGMESSAGECONTROLPROC(glfwGetProcAddress("glDebugMessageControl"));
01532     glDebugMessageControlARB =
01533 PFNGLDEBUGMESSAGECONTROLARBPROC(glfwGetProcAddress("glDebugMessageControlARB"));
01533     glDebugMessageInsert = PFNGLDEBUGMESSAGEINSERTPROC(glfwGetProcAddress("glDebugMessageInsert"));
01534     glDebugMessageInsertARB =
01535 PFNGLDEBUGMESSAGEINSERTARBPROC(glfwGetProcAddress("glDebugMessageInsertARB"));
01535     glDeleteBuffers = PFNGLDELETEBUFFERSPROC(glfwGetProcAddress("glDeleteBuffers"));
01536     glDeleteCommandListsNV =
01537 PFNGLDELETECOMMANDLISTSNVPROC(glfwGetProcAddress("glDeleteCommandListsNV"));
01537     glDeleteFramebuffers = PFNGLDELETEFRAMEBUFFERSPROC(glfwGetProcAddress("glDeleteFramebuffers"));
01538     glDeleteNamedStringARB =
01539 PFNGLDELETENAMEDSTRINGARBPROC(glfwGetProcAddress("glDeleteNamedStringARB"));
01539     glDeletePathsNV = PFNGLDELETEPATHSNVPROC(glfwGetProcAddress("glDeletePathsNV"));
01540     glDeletePerfMonitorsAMD =
01541 PFNGLDELETEPERFMONITORSAMDPROC(glfwGetProcAddress("glDeletePerfMonitorsAMD"));
01541     glDeletePerfQueryINTEL =
01542 PFNGLDELETEPERFQUERYINTELPROC(glfwGetProcAddress("glDeletePerfQueryINTEL"));
01542     glDeleteProgram = PFNGLDELETEPROGRAMPROC(glfwGetProcAddress("glDeleteProgram"));
01543     glDeleteProgramPipelines =
01544 PFNGLDELETEPROGRAMPIPELINESPROC(glfwGetProcAddress("glDeleteProgramPipelines"));
01544     glDeleteQueries = PFNGLDELETEQUERIESPROC(glfwGetProcAddress("glDeleteQueries"));
01545     glDeleteRenderbuffers = PFNGLDELETERENDERBUFFERSPROC(glfwGetProcAddress("glDeleteRenderbuffers"));
01546     glDeleteSamplers = PFNGLDELETESAMPLERSPROC(glfwGetProcAddress("glDeleteSamplers"));
01547     glDeleteShader = PFNGLDELETESHADERPROC(glfwGetProcAddress("glDeleteShader"));
01548     glDeleteStatesNV = PFNGLDELETESTATESNVPROC(glfwGetProcAddress("glDeleteStatesNV"));
01549     glDeleteSync = PFNGLDELETESYNCPROC(glfwGetProcAddress("glDeleteSync"));
01550     glDeleteTextures = PFNGLDELETETEXTURESPROC(glfwGetProcAddress("glDeleteTextures"));
01551     glDeleteTransformFeedbacks =
01552 PFNGLDELETETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glDeleteTransformFeedbacks"));
01552     glDeleteVertexArrays = PFNGLDELETEVERTEXARRAYSPROC(glfwGetProcAddress("glDeleteVertexArrays"));
01553     glDepthFunc = PFNGLDEPTHFUNCPROC(glfwGetProcAddress("glDepthFunc"));
01554     glDepthMask = PFNGLDEPTHMASKPROC(glfwGetProcAddress("glDepthMask"));
01555     glDepthRange = PFNGLDEPTH RANGEPROC(glfwGetProcAddress("glDepthRange"));
01556     glDepthRangeArrayv = PFNGLDEPTH RANGEARRAYVPROC(glfwGetProcAddress("glDepthRangeArrayv"));
01557     glDepthRangeIndexed = PFNGLDEPTH RANGEINDEXEDPROC(glfwGetProcAddress("glDepthRangeIndexed"));
01558     glDepthRangef = PFNGLDEPTH RANGEFPROC(glfwGetProcAddress("glDepthRangef"));
01559     glDetachShader = PFNGLDETACHSHADERPROC(glfwGetProcAddress("glDetachShader"));

```

```

01560     glDisable = PFNGLDISABLEPROC(glfwGetProcAddress("glDisable"));
01561     glDisableClientStateIndexedEXT =
01562         PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glDisableClientStateIndexedEXT"));
01563     glDisableClientStateiEXT =
01564         PFNGLDISABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glDisableClientStateiEXT"));
01565     glDisableIndexedEXT = PFNGLDISABLEINDEXEDEXTPROC(glfwGetProcAddress("glDisableIndexedEXT"));
01566     glDisableVertexArrayAttrib =
01567         PFNGLDISABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glDisableVertexArrayAttrib"));
01568     glDisableVertexArrayAttribEXT =
01569         PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glDisableVertexArrayAttribEXT"));
01570     glDisableVertexArrayEXT =
01571         PFNGLDISABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glDisableVertexArrayEXT"));
01572     glDisableVertexAttribArray =
01573         PFNGLDISABLEVERTEXATTRIBPROC(glfwGetProcAddress("glDisableVertexAttribArray"));
01574     glDisablei = PFNGLDISABLEIPROC(glfwGetProcAddress("glDisablei"));
01575     glDispatchCompute = PFNGLDISPATCHCOMPUTEPROC(glfwGetProcAddress("glDispatchCompute"));
01576     glDispatchComputeGroupSizeARB =
01577         PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC(glfwGetProcAddress("glDispatchComputeGroupSizeARB"));
01578     glDispatchComputeIndirect =
01579         PFNGLDISPATCHCOMPUTEINDIRECTPROC(glfwGetProcAddress("glDispatchComputeIndirect"));
01580     glDrawArrays = PFNGLDRAWARRAYSPROC(glfwGetProcAddress("glDrawArrays"));
01581     glDrawArraysIndirect = PFNGLDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glDrawArraysIndirect"));
01582     glDrawArraysInstanced = PFNGLDRAWARRAYSINSTANCEDPROC(glfwGetProcAddress("glDrawArraysInstanced"));
01583     glDrawArraysInstancedARB =
01584         PFNGLDRAWARRAYSINSTANCEDARBPROC(glfwGetProcAddress("glDrawArraysInstancedARB"));
01585     glDrawArraysInstancedBaseInstance =
01586         PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawArraysInstancedBaseInstance"));
01587     glDrawArraysInstancedEXT =
01588         PFNGLDRAWARRAYSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawArraysInstancedEXT"));
01589     glDrawBuffer = PFNGLDRAWBUFFERPROC(glfwGetProcAddress("glDrawBuffer"));
01590     glDrawBuffers = PFNGLDRAWBUFFERSPROC(glfwGetProcAddress("glDrawBuffers"));
01591     glDrawCommandsAddressNV =
01592         PFNGLDRAWCOMMANDSADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsAddressNV"));
01593     glDrawCommandsNV = PFNGLDRAWCOMMANDSNVPROC(glfwGetProcAddress("glDrawCommandsNV"));
01594     glDrawCommandsStatesAddressNV =
01595         PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsStatesAddressNV"));
01596     glDrawCommandsStatesNV =
01597         PFNGLDRAWCOMMANDSSTATESNVPROC(glfwGetProcAddress("glDrawCommandsStatesNV"));
01598     glDrawElements = PFNGLDRAWELEMENTSPROC(glfwGetProcAddress("glDrawElements"));
01599     glDrawElementsBaseVertex =
01600         PFNGLDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsBaseVertex"));
01601     glDrawElementsIndirect =
01602         PFNGLDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glDrawElementsIndirect"));
01603     glDrawElementsInstanced =
01604         PFNGLDRAWELEMENTSINSTANCEDPROC(glfwGetProcAddress("glDrawElementsInstanced"));
01605     glDrawElementsInstancedARB =
01606         PFNGLDRAWELEMENTSINSTANCEDARBPROC(glfwGetProcAddress("glDrawElementsInstancedARB"));
01607     glDrawElementsInstancedBaseInstance =
01608         PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseInstance"));
01609     glDrawElementsInstancedBaseVertex =
01610         PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertex"));
01611     glDrawElementsInstancedEXT =
01612         PFNGLDRAWELEMENTSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawElementsInstancedEXT"));
01613     glDrawRangeElements = PFNGLDRAWRANGEELEMENTSPROC(glfwGetProcAddress("glDrawRangeElements"));
01614     glDrawRangeElementsBaseVertex =
01615         PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawRangeElementsBaseVertex"));
01616     glDrawTransformFeedback =
01617         PFNGLDRAWTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glDrawTransformFeedback"));
01618     glDrawTransformFeedbackInstanced =
01619         PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackInstanced"));
01620     glDrawTransformFeedbackStream =
01621         PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC(glfwGetProcAddress("glDrawTransformFeedbackStream"));
01622     glDrawTransformFeedbackStreamInstanced =
01623         PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackStreamInstanced"));
01624     glDrawVkImageNV = PFNGLDRAWVKIMAGENVPROC(glfwGetProcAddress("glDrawVkImageNV"));
01625     glEdgeFlagFormatNV = PFNGLEDGEFLAGFORMATNVPROC(glfwGetProcAddress("glEdgeFlagFormatNV"));
01626     glEnable = PFNGLENABLEPROC(glfwGetProcAddress("glEnable"));
01627     glEnableClientStateIndexedEXT =
01628         PFNGLENABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glEnableClientStateIndexedEXT"));
01629     glEnableClientStateiEXT =
01630         PFNGLENABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glEnableClientStateiEXT"));
01631     glEnableIndexedEXT = PFNGLENABLEINDEXEDEXTPROC(glfwGetProcAddress("glEnableIndexedEXT"));
01632     glEnableVertexAttribArray =
01633         PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArray"));
01634     glEnableVertexAttribArrayEXT =
01635         PFNGLENABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayEXT"));
01636     glEnableVertexArrayEXT =
01637         PFNGLENABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glEnableVertexArrayEXT"));
01638     glEnableVertexAttribArray =
01639         PFNGLENABLEVERTEXATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArray"));
01640     glEnablei = PFNGLENABLEIPROC(glfwGetProcAddress("glEnablei"));
01641     glEndConditionalRender =
01642         PFNGLENDCONDITIONALRENDERPROC(glfwGetProcAddress("glEndConditionalRender"));
01643     glEndConditionalRenderNV =
01644         PFNGLENDCONDITIONALRENDERNVPROC(glfwGetProcAddress("glEndConditionalRenderNV"));

```

```

01612     glEndPerfMonitorAMD = PFNGLENDPERFMONITORAMDPROC(glfwGetProcAddress("glEndPerfMonitorAMD"));
01613     glEndPerfQueryINTEL = PFNGLENDPERFQUERYINTELPROC(glfwGetProcAddress("glEndPerfQueryINTEL"));
01614     glEndQuery = PFNGLENDQUERYPROC(glfwGetProcAddress("glEndQuery"));
01615     glEndQueryIndexed = PFNGLENDQUERYINDEXEDPROC(glfwGetProcAddress("glEndQueryIndexed"));
01616     glEndTransformFeedback =
01617         PFNGLENDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glEndTransformFeedback"));
01618     glEvaluateDepthValuesARB =
01619         PFNGLEVALUATEDEPTHVALUESARBPROC(glfwGetProcAddress("glEvaluateDepthValuesARB"));
01620     glFinishSync = PFNGLFENCESYNCPROC(glfwGetProcAddress("glFinishSync"));
01621     glFlush = PFNGLFLUSHPROC(glfwGetProcAddress("glFlush"));
01622     glFlushMappedBufferRange =
01623         PFNGLFLUSHMAPPEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedBufferRange"));
01624     glFlushMappedNamedBufferRange =
01625         PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedNamedBufferRange"));
01626     glFlushMappedNamedBufferRangeEXT =
01627         PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glFlushMappedNamedBufferRangeEXT"));
01628     glFogCoordFormatNV = PFNGLFOGCOORDFORMATNVPROC(glfwGetProcAddress("glFogCoordFormatNV"));
01629     glFragmentCoverageColorNV =
01630         PFNGLFRAGMENTCOVERAGECOLORNVPROC(glfwGetProcAddress("glFragmentCoverageColorNV"));
01631     glFramebufferDrawBufferEXT =
01632         PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC(glfwGetProcAddress("glFramebufferDrawBufferEXT"));
01633     glFramebufferDrawBuffersEXT =
01634         PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC(glfwGetProcAddress("glFramebufferDrawBuffersEXT"));
01635     glFramebufferParameteri =
01636         PFNGLFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glFramebufferParameteri"));
01637     glFramebufferReadBufferEXT =
01638         PFNGLFRAMEBUFFERREADBUFFEREXTPROC(glfwGetProcAddress("glFramebufferReadBufferEXT"));
01639     glFramebufferRenderbuffer =
01640         PFNGLFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glFramebufferRenderbuffer"));
01641     glFramebufferSampleLocationsfvARB =
01642         PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvARB"));
01643     glFramebufferSampleLocationsfvNV =
01644         PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvNV"));
01645     glFramebufferTexture = PFNGLFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glFramebufferTexture"));
01646     glFramebufferTexture1D =
01647         PFNGLFRAMEBUFFERTEXTURE1DPROC(glfwGetProcAddress("glFramebufferTexture1D"));
01648     glFramebufferTexture2D =
01649         PFNGLFRAMEBUFFERTEXTURE2DPROC(glfwGetProcAddress("glFramebufferTexture2D"));
01650     glFramebufferTexture3D =
01651         PFNGLFRAMEBUFFERTEXTURE3DPROC(glfwGetProcAddress("glFramebufferTexture3D"));
01652     glFramebufferTextureARB =
01653         PFNGLFRAMEBUFFERTEXTUREARBPROC(glfwGetProcAddress("glFramebufferTextureARB"));
01654     glFramebufferTextureFaceARB =
01655         PFNGLFRAMEBUFFERTEXTUREFACEARBPROC(glfwGetProcAddress("glFramebufferTextureFaceARB"));
01656     glFramebufferTextureLayer =
01657         PFNGLFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glFramebufferTextureLayer"));
01658     glFramebufferTextureLayerARB =
01659         PFNGLFRAMEBUFFERTEXTURELAYERARBPROC(glfwGetProcAddress("glFramebufferTextureLayerARB"));
01660     glFramebufferTextureMultiviewOVR =
01661         PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC(glfwGetProcAddress("glFramebufferTextureMultiviewOVR"));
01662     glFrontFace = PFNGLFRONTFACEPROC(glfwGetProcAddress("glFrontFace"));
01663     glGenBuffers = PFNGLGENBUFFERSPROC(glfwGetProcAddress("glGenBuffers"));
01664     glGenFramebuffers = PFNGLGENFRAMEBUFFERSPROC(glfwGetProcAddress("glGenFramebuffers"));
01665     glGenPathsNV = PFNGLGENPATHSNVPROC(glfwGetProcAddress("glGenPathsNV"));
01666     glGenPerfMonitorsAMD = PFNGLGENPERFMONITORSAMDPROC(glfwGetProcAddress("glGenPerfMonitorsAMD"));
01667     glGenProgramPipelines = PFNGLGENPROGRAMPIPELINESPROC(glfwGetProcAddress("glGenProgramPipelines"));
01668     glGenQueries = PFNGLGENQUERIESPROC(glfwGetProcAddress("glGenQueries"));
01669     glGenRenderbuffers = PFNGLGENRENDERBUFFERSPROC(glfwGetProcAddress("glGenRenderbuffers"));
01670     glGenSamplers = PFNGLGENSAMPLERSPROC(glfwGetProcAddress("glGenSamplers"));
01671     glGenTextures = PFNGLGENTEXTURESPROC(glfwGetProcAddress("glGenTextures"));
01672     glGenTransformFeedbacks =
01673         PFNGLGENTRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glGenTransformFeedbacks"));
01674     glGenVertexArrays = PFNGLGENVERTEXARRAYSPROC(glfwGetProcAddress("glGenVertexArrays"));
01675     glGenerateMipmap = PFNGLGENERATEMIPMAPPROC(glfwGetProcAddress("glGenerateMipmap"));
01676     glGenerateMultiTexMipmapEXT =
01677         PFNGLGENERATEMULTITEXMIPMAPEXTPROC(glfwGetProcAddress("glGenerateMultiTexMipmapEXT"));
01678     glGenerateTextureMipmap =
01679         PFNGLGENERATETEXTUREMIPMAPPROC(glfwGetProcAddress("glGenerateTextureMipmap"));
01680     glGenerateTextureMipmapEXT =
01681         PFNGLGENERATETEXTUREMIPMAPEXTPROC(glfwGetProcAddress("glGenerateTextureMipmapEXT"));
01682     glGetActiveAtomicCounterBufferiv =
01683         PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC(glfwGetProcAddress("glGetActiveAtomicCounterBufferiv"));
01684     glGetActiveAttrib = PFNGLGETACTIVEATTRIBPROC(glfwGetProcAddress("glGetActiveAttrib"));
01685     glGetActiveSubroutineName =
01686         PFNGLGETACTIVESUBROUTINENAMEPROC(glfwGetProcAddress("glGetActiveSubroutineName"));
01687     glGetActiveSubroutineUniformName =
01688         PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveSubroutineUniformName"));
01689     glGetActiveSubroutineUniformiv =
01690         PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC(glfwGetProcAddress("glGetActiveSubroutineUniformiv"));
01691     glGetActiveUniform = PFNGLGETACTIVEUNIFORMPROC(glfwGetProcAddress("glGetActiveUniform"));
01692     glGetActiveUniformBlockName =
01693         PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC(glfwGetProcAddress("glGetActiveUniformBlockName"));
01694     glGetActiveUniformBlockiv =
01695         PFNGLGETACTIVEUNIFORMBLOCKIVPROC(glfwGetProcAddress("glGetActiveUniformBlockiv"));
01696     glGetActiveUniformName =
01697         PFNGLGETACTIVEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveUniformName"));

```

```

01667 glGetActiveUniformsiv = PFNGLGETACTIVEUNIFORMSIVPROC(glfwGetProcAddress("glGetActiveUniformsiv"));
01668 glGetAttachedShaders = PFNGLGETATTACHEDSHADERSPROC(glfwGetProcAddress("glGetAttachedShaders"));
01669 glGetAttribLocation = PFNGLGETATTRIBLOCATIONPROC(glfwGetProcAddress("glGetAttribLocation"));
01670 glGetBooleanIndexedvEXT =
PFNGLGETBOOLEANINDEXEDVEXTPROC(glfwGetProcAddress("glGetBooleanIndexedvEXT"));
01671 glGetBooleani_v = PFNGLGETBOOLEANI_VPROC(glfwGetProcAddress("glGetBooleani_v"));
01672 glGetBooleanv = PFNGLGETBOOLEANVPROC(glfwGetProcAddress("glGetBooleanv"));
01673 glGetBufferParameteri64v =
PFNGLGETBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetBufferParameteri64v"));
01674 glGetBufferParameteriv =
PFNGLGETBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetBufferParameteriv"));
01675 glGetBufferParameterui64NV =
PFNGLGETBUFFERPARAMETERUI64NVPROC(glfwGetProcAddress("glGetBufferParameterui64vNV"));
01676 glGetBufferPointerv = PFNGLGETBUFFERPOINTERVPROC(glfwGetProcAddress("glGetBufferPointerv"));
01677 glGetBufferSubData = PFNGLGETBUFFERSUBDATAPROC(glfwGetProcAddress("glGetBufferSubData"));
01678 glGetCommandHeaderNV = PFNGLGETCOMMANDHEADERNVPROC(glfwGetProcAddress("glGetCommandHeaderNV"));
01679 glGetCompressedMultiTexImageEXT =
PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedMultiTexImageEXT"));
01680 glGetCompressedTexImage =
PFNGLGETCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetCompressedTexImage"));
01681 glGetCompressedTextureImage =
PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureImage"));
01682 glGetCompressedTextureImageEXT =
PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedTextureImageEXT"));
01683 glGetCompressedTextureSubImage =
PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureSubImage"));
01684 glGetCoverageModulationTableNV =
PFNGLGETCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glGetCoverageModulationTableNV"));
01685 glGetDebugMessageLog = PFNGLGETDEBUGMESSAGELOGPROC(glfwGetProcAddress("glGetDebugMessageLog"));
01686 glGetDebugMessageLogARB =
PFNGLGETDEBUGMESSAGELOGARBPROC(glfwGetProcAddress("glGetDebugMessageLogARB"));
01687 glGetDoubleIndexedvEXT =
PFNGLGETDOUBLEINDEXEDVEXTPROC(glfwGetProcAddress("glGetDoubleIndexedvEXT"));
01688 glGetDoublei_v = PFNGLGETDOUBLEI_VPROC(glfwGetProcAddress("glGetDoublei_v"));
01689 glGetDoublei_vEXT = PFNGLGETDOUBLEI_VEXTPROC(glfwGetProcAddress("glGetDoublei_vEXT"));
01690 glGetDoublev = PFNGLGETDOUBLEVPROC(glfwGetProcAddress("glGetDoublev"));
01691 glGetError = PFNGLGETERRORPROC(glfwGetProcAddress("glGetError"));
01692 glGetFirstPerfQueryIdINTEL =
PFNGLGETFIRSTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetFirstPerfQueryIdINTEL"));
01693 glGetFloatIndexedvEXT = PFNGLGETFLOATINDEXEDVEXTPROC(glfwGetProcAddress("glGetFloatIndexedvEXT"));
01694 glGetFloati_v = PFNGLGETFLOATI_VPROC(glfwGetProcAddress("glGetFloati_v"));
01695 glGetFloati_vEXT = PFNGLGETFLOATI_VEXTPROC(glfwGetProcAddress("glGetFloati_vEXT"));
01696 glGetFloatv = PFNGLGETFLOATVPROC(glfwGetProcAddress("glGetFloatv"));
01697 glGetFragDataIndex = PFNGLGETFRAGDATAINDEXPROC(glfwGetProcAddress("glGetFragDataIndex"));
01698 glGetFragDataLocation = PFNGLGETFRAGDATALOCATIONPROC(glfwGetProcAddress("glGetFragDataLocation"));
01699 glGetFramebufferAttachmentParameteriv =
PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferAttachmentParameteriv"));
01700 glGetFramebufferParameteri =
PFNGLGETFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferParameteriv"));
01701 glGetFramebufferParameterivEXT =
PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetFramebufferParameterivEXT"));
01702 glGetGraphicsResetStatus =
PFNGLGETGRAPHICSRESETSTATUSPROC(glfwGetProcAddress("glGetGraphicsResetStatus"));
01703 glGetGraphicsResetStatusARB =
PFNGLGETGRAPHICSRESETSTATUSARBPROC(glfwGetProcAddress("glGetGraphicsResetStatusARB"));
01704 glGetImageHandleARB = PFNGLGETIMAGEHANDLEARBPROC(glfwGetProcAddress("glGetImageHandleARB"));
01705 glGetImageHandleNV = PFNGLGETIMAGEHANDLEENVPROC(glfwGetProcAddress("glGetImageHandleNV"));
01706 glGetInteger64i_v = PFNGLGETINTEGER64I_VPROC(glfwGetProcAddress("glGetInteger64i_v"));
01707 glGetInteger64v = PFNGLGETINTEGER64VPROC(glfwGetProcAddress("glGetInteger64v"));
01708 glGetIntegerIndexedvEXT =
PFNGLGETINTEGERINDEXEDVEXTPROC(glfwGetProcAddress("glGetIntegerIndexedvEXT"));
01709 glGetIntegeri_v = PFNGLGETINTEGERI_VPROC(glfwGetProcAddress("glGetIntegeri_v"));
01710 glGetIntegerui64i_vNV = PFNGLGETINTEGERUI64I_VNVPROC(glfwGetProcAddress("glGetIntegerui64i_vNV"));
01711 glGetIntegerui64vNV = PFNGLGETINTEGERUI64VNVPROC(glfwGetProcAddress("glGetIntegerui64vNV"));
01712 glGetIntegerv = PFNGLGETINTEGERVPROC(glfwGetProcAddress("glGetIntegerv"));
01713 glGetInternalformatSampleivNV =
PFNGLGETINTERNALFORMATSAMPLEIVNVPROC(glfwGetProcAddress("glGetInternalformatSampleivNV"));
01714 glGetInternalformati64v =
PFNGLGETINTERNALFORMATI64VPROC(glfwGetProcAddress("glGetInternalformati64v"));
01715 glGetInternalformativ = PFNGLGETINTERNALFORMATIVPROC(glfwGetProcAddress("glGetInternalformativ"));
01716 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01717 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01718 glGetMultiTexGendvEXT = PFNGLGETMULTITEXGENDVEXTPROC(glfwGetProcAddress("glGetMultiTexGendvEXT"));
01719 glGetMultiTexGenfvEXT = PFNGLGETMULTITEXGENFVEXTPROC(glfwGetProcAddress("glGetMultiTexGenfvEXT"));
01720 glGetMultiTexGenivEXT = PFNGLGETMULTITEXGENIVEXTPROC(glfwGetProcAddress("glGetMultiTexGenivEXT"));
01721 glGetMultiTexImageEXT = PFNGLGETMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetMultiTexImageEXT"));
01722 glGetMultiTexLevelParameterfvEXT =
PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterfvEXT"));
01723 glGetMultiTexLevelParameterivEXT =
PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterivEXT"));
01724 glGetMultiTexParameteriivEXT =
PFNGLGETMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameteriivEXT"));
01725 glGetMultiTexParameterIiivEXT =
PFNGLGETMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIiivEXT"));
01726 glGetMultiTexParameterfvEXT =
PFNGLGETMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterfvEXT"));
01727 glGetMultiTexParameterivEXT =

```

```

01728 PFNGLGETMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterivEXT"));
01729 glGetMultisamplefv = PFNGLGETMULTISAMPLEFVPROC(glfwGetProcAddress("glGetMultisamplefv"));
01730 PFNGLGETNAMEDBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetNamedBufferParameteri64v"));
01731 glGetNamedBufferParameteriv =
01732 PFNGLGETNAMEDBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedBufferParameteriv"));
01733 glGetNamedBufferParameterivEXT =
01734 PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedBufferParameterivEXT"));
01735 glGetNamedBufferParameterui64NV =
01736 PFNGLGETNAMEDBUFFERPARAMETERUI64NVPROC(glfwGetProcAddress("glGetNamedBufferParameterui64NV"));
01737 glGetNamedBufferPointerv =
01738 PFNGLGETNAMEDBUFFERPOINTERVPROC(glfwGetProcAddress("glGetNamedBufferPointerv"));
01739 glGetNamedBufferPointervEXT =
01740 PFNGLGETNAMEDBUFFERPOINTERVEXTPROC(glfwGetProcAddress("glGetNamedBufferPointervEXT"));
01741 glGetNamedBufferData =
01742 PFNGLGETNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glGetNamedBufferData"));
01743 glGetNamedBufferDataEXT =
01744 PFNGLGETNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glGetNamedBufferDataEXT"));
01745 glGetNamedFramebufferAttachmentParameteriv =
01746 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameteriv"));
01747 glGetNamedFramebufferAttachmentParameterivEXT =
01748 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameterivEXT"));
01749 glGetNamedFramebufferParameteriv =
01750 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIuvEXT"));
01751 glGetNamedProgramLocalParameterIuvEXT =
01752 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterDuvEXT"));
01753 glGetNamedProgramLocalParameterdvEXT =
01754 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterDvEXT"));
01755 glGetNamedProgramLocalParameterfvEXT =
01756 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterFvEXT"));
01757 glGetNamedProgramStringEXT =
01758 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glGetNamedProgramStringEXT"));
01759 glGetNamedProgramivEXT =
01760 PFNGLGETNAMEDPROGRAMIVEXTPROC(glfwGetProcAddress("glGetNamedProgramivEXT"));
01761 glGetNamedProgramvEXT =
01762 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedRenderbufferParameteriv"));
01763 glGetNamedRenderbufferParameterivEXT =
01764 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedRenderbufferParameterivEXT"));
01765 glGetStringARB = PFNGLGETNAMEDSTRINGARBPROC(glfwGetProcAddress("glGetStringARB"));
01766 glGetStringivARB = PFNGLGETNAMEDSTRINGIVARBPROC(glfwGetProcAddress("glGetStringivARB"));
01767 glGetNextPerfQueryIdINTEL =
01768 PFNGLGETNEXTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetNextPerfQueryIdINTEL"));
01769 glGetObjectLabel = PFNGLGETOBJECTLABELPROC(glfwGetProcAddress("glGetObjectLabel"));
01770 glGetObjectLabelEXT = PFNGLGETOBJECTLABELEXTPROC(glfwGetProcAddress("glGetObjectLabelEXT"));
01771 glGetObjectPtrLabel = PFNGLGETOBJECTPTRLABELPROC(glfwGetProcAddress("glGetObjectPtrLabel"));
01772 glGetPathCommandsNV = PFNGLGETPATHCOMMANDSNVPROC(glfwGetProcAddress("glGetPathCommandsNV"));
01773 glGetPathCoordsNV = PFNGLGETPATHCOORDSNVPROC(glfwGetProcAddress("glGetPathCoordsNV"));
01774 glGetPathDashArrayNV = PFNGLGETPATHDASHARRAYNVPROC(glfwGetProcAddress("glGetPathDashArrayNV"));
01775 glGetPathLengthNV = PFNGLGETPATHLENGTHNVPROC(glfwGetProcAddress("glGetPathLengthNV"));
01776 glGetPathMetricRangeNV =
01777 PFNGLGETPATHMETRICRANGEVPROC(glfwGetProcAddress("glGetPathMetricRangeNV"));
01778 glGetPathMetricsNV = PFNGLGETPATHMETRICSNVPROC(glfwGetProcAddress("glGetPathMetricsNV"));
01779 glGetPathParameterfvNV =
01780 PFNGLGETPATHPARAMETERFVNVPROC(glfwGetProcAddress("glGetPathParameterfvNV"));
01781 glGetPathParameterivNV =
01782 PFNGLGETPATHPARAMETERIVNVPROC(glfwGetProcAddress("glGetPathParameterivNV"));
01783 glGetPathSpacingNV = PFNGLGETPATHSPACINGNVPROC(glfwGetProcAddress("glGetPathSpacingNV"));
01784 glGetPerfCounterInfoINTEL =
01785 PFNGLGETPERFCOUNTERINFOINTELPROC(glfwGetProcAddress("glGetPerfCounterInfoINTEL"));
01786 glGetPerfMonitorCounterDataAMD =
01787 PFNGLGETPERFMONITORCOUNTERDATAAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterDataAMD"));
01788 glGetPerfMonitorCounterInfoAMD =
01789 PFNGLGETPERFMONITORCOUNTERINFOAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterInfoAMD"));
01790 glGetPerfMonitorCounterStringAMD =
01791 PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterStringAMD"));
01792 glGetPerfMonitorCountersAMD =
01793 PFNGLGETPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glGetPerfMonitorCountersAMD"));
01794 glGetPerfMonitorGroupStringAMD =
01795 PFNGLGETPERFMONITORGROUPSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupStringAMD"));
01796 glGetPerfMonitorGroupsAMD =
01797 PFNGLGETPERFMONITORGROUPSAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupsAMD"));
01798 glGetPerfQueryDataINTEL =
01799 PFNGLGETPERFQUERYDATAINTELPROC(glfwGetProcAddress("glGetPerfQueryDataINTEL"));
01800 glGetPerfQueryIdByNameINTEL =
01801 PFNGLGETPERFQUERYIDBYNAMEINTELPROC(glfwGetProcAddress("glGetPerfQueryIdByNameINTEL"));
01802 glGetPerfQueryInfoINTEL =
01803 PFNGLGETPERFQUERYINFOINTELPROC(glfwGetProcAddress("glGetPerfQueryInfoINTEL"));
01804 glGetPointerIndexedvEXT =
01805 PFNGLGETPOINTERINDEXEDVEXTPROC(glfwGetProcAddress("glGetPointerIndexedvEXT"));
01806 glGetPointeri_vEXT = PFNGLGETPOINTERI_VEXTPROC(glfwGetProcAddress("glGetPointeri_vEXT"));
01807 glGetPointerv = PFNGLGETPOINTERVPROC(glfwGetProcAddress("glGetPointerv"));
01808 glGetProgramBinary = PFNGLGETPROGRAMBINARYPROC(glfwGetProcAddress("glGetProgramBinary"));
01809 glGetProgramInfoLog = PFNGLGETPROGRAMINFOLOGPROC(glfwGetProcAddress("glGetProgramInfoLog"));

```

```

01779    glGetProgramInterfaceiv =
01780    PFNGLGETPROGRAMINTERFACEIVPROC(glfwGetProcAddress("glGetProgramInterfaceiv"));
01781    glGetProgramPipelineInfoLog =
01782    PFNGLGETPROGRAMPIPELINEINFOLOGPROC(glfwGetProcAddress("glGetProgramPipelineInfoLog"));
01783    glGetProgramPipelineiv =
01784    PFNGLGETPROGRAMPIPELINEIVPROC(glfwGetProcAddress("glGetProgramPipelineiv"));
01785    glGetProgramResourceIndex =
01786    PFNGLGETPROGRAMRESOURCEINDEXPROC(glfwGetProcAddress("glGetProgramResourceIndex"));
01787    glGetProgramResourceLocation =
01788    PFNGLGETPROGRAMRESOURCELOCATIONPROC(glfwGetProcAddress("glGetProgramResourceLocation"));
01789    glGetProgramResourceLocationIndex =
01790    PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC(glfwGetProcAddress("glGetProgramResourceLocationIndex"));
01791    glGetProgramResourceName =
01792    PFNGLGETPROGRAMRESOURCENAMEPROC(glfwGetProcAddress("glGetProgramResourceName"));
01793    glGetProgramResourcefvNV =
01794    PFNGLGETPROGRAMRESOURCEFNVPROC(glfwGetProcAddress("glGetProgramResourcefvNV"));
01795    glGetProgramResourceiv =
01796    PFNGLGETPROGRAMRESOURCEIVPROC(glfwGetProcAddress("glGetProgramResourceiv"));
01797    glGetProgramStageiv = PFNGLGETPROGRAMSTAGEIVPROC(glfwGetProcAddress("glGetProgramStageiv"));
01798    glGetProgramiv = PFNGLGETPROGRAMIVPROC(glfwGetProcAddress("glGetProgramiv"));
01799    glGetQueryBufferObjecti64v =
01800    PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
01801    glGetQueryBufferObjectiv =
01802    PFNGLGETQUERYBUFFEROBJECTIVPROC(glfwGetProcAddress("glGetQueryBufferObjectiv"));
01803    glGetQueryBufferObjectui64v =
01804    PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
01805    glGetQueryBufferObjectuiv =
01806    PFNGLGETQUERYBUFFEROBJECTUIVPROC(glfwGetProcAddress("glGetQueryBufferObjectuiv"));
01807    glGetQueryIndexediv = PFNGLGETQUERYINDEXEDIVPROC(glfwGetProcAddress("glGetQueryIndexediv"));
01808    glGetQueryObjecti64v = PFNGLGETQUERYOBJECTI64VPROC(glfwGetProcAddress("glGetQueryObjecti64v"));
01809    glGetQueryObjectiv = PFNGLGETQUERYOBJECTIVPROC(glfwGetProcAddress("glGetQueryObjectiv"));
01810    glGetQueryObjectui64v = PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
01811    glGetQueryObjectuiv = PFNGLGETQUERYOBJECTUIVPROC(glfwGetProcAddress("glGetQueryObjectuiv"));
01812    glGetQueryiv = PFNGLGETQUERYIVPROC(glfwGetProcAddress("glGetQueryiv"));
01813    glGetRenderbufferParameteriv =
01814    PFNGLGETRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetRenderbufferParameteriv"));
01815    glGetSamplerParameterIiv =
01816    PFNGLGETSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glGetSamplerParameterIiv"));
01817    glGetSamplerParameterIuiv =
01818    PFNGLGETSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glGetSamplerParameterIuiv"));
01819    glGetSamplerParameterfv =
01820    PFNGLGETSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glGetSamplerParameterfv"));
01821    glGetSamplerParameteriv =
01822    PFNGLGETSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glGetSamplerParameteriv"));
01823    glGetShaderInfoLog = PFNGLGETSHADERINFOLOGPROC(glfwGetProcAddress("glGetShaderInfoLog"));
01824    glGetShaderPrecisionFormat =
01825    PFNGLGETSHADERPRECISIONFORMATPROC(glfwGetProcAddress("glGetShaderPrecisionFormat"));
01826    glGetShaderSource = PFNGLGETSHADERSOURCEPROC(glfwGetProcAddress("glGetShaderSource"));
01827    glGetShaderiv = PFNGLGETSHADERIVPROC(glfwGetProcAddress("glGetShaderiv"));
01828    glGetStageIndexNV = PFNGLGETSTAGEINDEXNVPROC(glfwGetProcAddress("glGetStageIndexNV"));
01829    glGetString = PFNGLGETSTRINGPROC(glfwGetProcAddress("glGetString"));
01830    glGetStringi = PFNGLGETSTRINGIPROC(glfwGetProcAddress("glGetStringi"));
01831    glGetSubroutineIndex = PFNGLGETSUBROUTINEINDEXPROC(glfwGetProcAddress("glGetSubroutineIndex"));
01832    glGetSubroutineUniformLocation =
01833    PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetSubroutineUniformLocation"));
01834    glGetSynciv = PFNGLGETSYNCIVPROC(glfwGetProcAddress("glGetSynciv"));
01835    glGetTexImage = PFNGLGETTEXIMAGEPROC(glfwGetProcAddress("glGetTexImage"));
01836    glGetTexLevelParameterfv =
01837    PFNGLGETTEXLEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTexLevelParameterfv"));
01838    glGetTexLevelParameteriv =
01839    PFNGLGETTEXLEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTexLevelParameteriv"));
01840    glGetTexParameterIiv = PFNGLGETTEXPARAMETERIIVPROC(glfwGetProcAddress("glGetTexParameterIiv"));
01841    glGetTexParameterIuiv = PFNGLGETTEXPARAMETERIUIVPROC(glfwGetProcAddress("glGetTexParameterIuiv"));
01842    glGetTexParameterfv = PFNGLGETTEXPARAMETERFVPROC(glfwGetProcAddress("glGetTexParameterfv"));
01843    glGetTexParameteriv = PFNGLGETTEXPARAMETERIVPROC(glfwGetProcAddress("glGetTexParameteriv"));
01844    glGetTextureHandleARB = PFNGLGETTEXTUREHANDLEARBPROC(glfwGetProcAddress("glGetTextureHandleARB"));
01845    glGetTextureHandleNV = PFNGLGETTEXTUREHANDLENVPROC(glfwGetProcAddress("glGetTextureHandleNV"));
01846    glGetTexImageEXT = PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTexImageEXT"));
01847    glGetTextureImageEXT = PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTextureImageEXT"));
01848    glGetTextureLevelParameterfv =
01849    PFNGLGETTEXTURELEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTextureLevelParameterfv"));
01850    glGetTextureLevelParameterfvEXT =
01851    PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterfvEXT"));
01852    glGetTextureLevelParameteriv =
01853    PFNGLGETTEXTURELEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTextureLevelParameteriv"));
01854    glGetTextureLevelParameterivEXT =
01855    PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterivEXT"));
01856    glGetTextureParameterIiv =
01857    PFNGLGETTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glGetTextureParameterIiv"));
01858    glGetTextureParameterIuivEXT =
01859    PFNGLGETTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glGetTextureParameterIuiv"));
01860    glGetTextureParameterfvEXT =
01861    PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
01862    glGetTextureParameterfv =
01863    PFNGLGETTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glGetTextureParameterfv"));

```

```

01835     glGetTextureParameterfvEXT =
01836     PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
01837     glGetTextureParameteriv =
01838     PFNGLGETTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glGetTextureParameteriv"));
01839     glGetTextureParameterivEXT =
01840     PFNGLGETTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureParameterivEXT"));
01841     glGetTextureSamplerHandleARB =
01842     PFNGLGETTEXTURESAMPLERHANDLEARBPROC(glfwGetProcAddress("glGetTextureSamplerHandleARB"));
01843     glGetTextureSamplerHandleNV =
01844     PFNGLGETTEXTURESAMPLERHANDLENVPROC(glfwGetProcAddress("glGetTextureSamplerHandleNV"));
01845     glGetTextureSubImage =
01846     PFNGLGETTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetTextureSubImage"));
01847     glGetTransformFeedbackVarying =
01848     PFNGLGETTRANSFORMFEEDBACKVARYINGPROC(glfwGetProcAddress("glGetTransformFeedbackVarying"));
01849     glGetTransformFeedbacki64_v =
01850     PFNGLGETTRANSFORMFEEDBACKI64_VPROC(glfwGetProcAddress("glGetTransformFeedbacki64_v"));
01851     glGetTransformFeedbacki_v =
01852     PFNGLGETTRANSFORMFEEDBACKI_VPROC(glfwGetProcAddress("glGetTransformFeedbacki_v"));
01853     glGetTransformFeedbackiv =
01854     PFNGLGETTRANSFORMFEEDBACKIVPROC(glfwGetProcAddress("glGetTransformFeedbackiv"));
01855     glGetUniformLocation =
01856     PFNGLGETUNIFORMBLOCKINDEXPROC(glfwGetProcAddress("glGetUniformBlockIndex"));
01857     glGetUniformLocation =
01858     PFNGLGETUNIFORMINDICESPROC(glfwGetProcAddress("glGetUniformIndices"));
01859     glGetUniformLocation =
01860     PFNGLGETUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetUniformLocation"));
01861     glGetUniformLocation =
01862     PFNGLGETUNIFORMSUBROUTINEUIVPROC(glfwGetProcAddress("glGetUniformSubroutineuiv"));
01863     glGetUniformLocation =
01864     PFNGLGETUNIFORMMDVPROC(glfwGetProcAddress("glGetUniformmdv"));
01865     glGetUniformLocation =
01866     PFNGLGETUNIFORMFVPROC(glfwGetProcAddress("glGetUniformfv"));
01867     glGetUniformLocation =
01868     PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64varb"));
01869     glGetUniformLocation =
01870     PFNGLGETUNIFORMI64NVPROC(glfwGetProcAddress("glGetUniformi64nv"));
01871     glGetUniformLocation =
01872     PFNGLGETUNIFORMIVPROC(glfwGetProcAddress("glGetUniformiv"));
01873     glGetUniformLocation =
01874     PFNGLGETUNIFORMLIVPROC(glfwGetProcAddress("glGetUniformliv"));
01875     glGetUniformLocation =
01876     PFNGLGETUNIFORMMIVPROC(glfwGetProcAddress("glGetUniformmiv"));
01877     glGetUniformLocation =
01878     PFNGLGETUNIFORMMDVPROC(glfwGetProcAddress("glGetUniformmdv"));
01879     glGetUniformLocation =
01880     PFNGLGETUNIFORMDVARBPROC(glfwGetProcAddress("glGetUniformdvARB"));
01881     glGetUniformLocation =
01882     PFNGLGETUNIFORMFVARBPROC(glfwGetProcAddress("glGetUniformfvARB"));
01883     glGetUniformLocation =
01884     PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64varb"));
01885     glGetUniformLocation =
01886     PFNGLGETUNIFORMIVARBPROC(glfwGetProcAddress("glGetUniformivARB"));
01887     glGetUniformLocation =
01888     PFNGLGETUNIFORMMUIVARBPROC(glfwGetProcAddress("glGetUniformmivARB"));
01889     glGetUniformLocation =
01890     PFNGLINDEXFORMATNVPROC(glfwGetProcAddress("glIndexFormatNV"));
01891     glGetUniformLocation =
01892     PFNGLINSERTEVENTMARKEREXTPROC(glfwGetProcAddress("glInsertEventMarkerEXT"));
01893     glGetUniformLocation =
01894     PFNGLINTERPOLATEPATHSNVPROC(glfwGetProcAddress("glInterpolatePathsNV"));
01895     glGetUniformLocation =
01896     PFNGLINVALIDATEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
01897     glGetUniformLocation =
01898     PFNGLINVALIDATEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateBufferSubData"));
01899     glGetUniformLocation =

```

```

0186 PFNGLINVALIDATEFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateFramebuffer"));
0187     glInvalidateNamedFramebufferData =
0188 PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferData"));
0189     glInvalidateNamedFramebufferSubData =
0190 PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferSubData"));
0191     glInvalidateSubFramebuffer =
0192 PFNGLINVALIDATESUBFRAMEBUFERPROC(glfwGetProcAddress("glInvalidateSubFramebuffer"));
0193     glInvalidateTexImage = PFNGLINVALIDATETEXIMAGEPROC(glfwGetProcAddress("glInvalidateTexImage"));
0194     glInvalidateTexSubImage =
0195 PFNGLINVALIDATETEXSUBIMAGEPROC(glfwGetProcAddress("glInvalidateTexSubImage"));
0196     glIsBuffer = PFNGLISBUFFERPROC(glfwGetProcAddress("glIsBuffer"));
0197     glIsBufferResidentNV = PFNGLISBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsBufferResidentNV"));
0198     glIsCommandListNV = PFNGLCOMMANDLISTNVPROC(glfwGetProcAddress("glIsCommandListNV"));
0199     glIsEnabled = PFNGLISENABLEDPROC(glfwGetProcAddress("glIsEnabled"));
0200     glIsEnabledIndexedEXT = PFNGLISENABLEDINDEXEDEXTPROC(glfwGetProcAddress("glIsEnabledIndexedEXT"));
0201     glIsEnabledi = PFNGLISENABLEDIPROC(glfwGetProcAddress("glIsEnabledi"));
0202     glIsFramebuffer = PFNGLISFRAMEBUFFERPROC(glfwGetProcAddress("glIsFramebuffer"));
0203     glIsImageHandleResidentARB =
0204 PFNGLISIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsImageHandleResidentARB"));
0205     glIsImageHandleResidentNV =
0206 PFNGLISIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsImageHandleResidentNV"));
0207     glIsNamedBufferResidentNV =
0208 PFNGLISNAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsNamedBufferResidentNV"));
0209     glIsNamedStringARB = PFNGLISNAMEDSTRINGARBPROC(glfwGetProcAddress("glIsNamedStringARB"));
0210     glIsPathNV = PFNGLISPATHNVPROC(glfwGetProcAddress("glIsPathNV"));
0211     glIsPointInFillPathNV = PFNGLISPOINTINFILLPATHNVPROC(glfwGetProcAddress("glIsPointInFillPathNV"));
0212     glIsPointInStrokePathNV =
0213 PFNGLISPOINTINSTROKEPATHNVPROC(glfwGetProcAddress("glIsPointInStrokePathNV"));
0214     glIsProgram = PFNGLISPROGRAMPROC(glfwGetProcAddress("glIsProgram"));
0215     glIsProgramPipeline = PFNGLISPROGRAMPIPELINEPROC(glfwGetProcAddress("glIsProgramPipeline"));
0216     glIsQuery = PFNGLISQUERYPROC(glfwGetProcAddress("glIsQuery"));
0217     glIsRenderbuffer = PFNGLISRENDERBUFFERPROC(glfwGetProcAddress("glIsRenderbuffer"));
0218     glIsSampler = PFNGLISSAMPLERPROC(glfwGetProcAddress("glIsSampler"));
0219     glIsShader = PFNGLISSHADERPROC(glfwGetProcAddress("glIsShader"));
0220     glIsStateNV = PFNGLISSTATENVPROC(glfwGetProcAddress("glIsStateNV"));
0221     glIsSync = PFNGLISSYNCPROC(glfwGetProcAddress("glIsSync"));
0222     glIsTexture = PFNGLISTEXTUREPROC(glfwGetProcAddress("glIsTexture"));
0223     glIsTextureHandleResidentARB =
0224 PFNGLISTEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsTextureHandleResidentARB"));
0225     glIsTextureHandleResidentNV =
0226 PFNGLISTEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsTextureHandleResidentNV"));
0227     glIsTransformFeedback = PFNGLISTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glIsTransformFeedback"));
0228     glIsVertexArray = PFNGLISVERTEXARRAYPROC(glfwGetProcAddress("glIsVertexArray"));
0229     glLabelObjectEXT = PFNGLLABELOBJECTEXTPROC(glfwGetProcAddress("glLabelObjectEXT"));
0230     glLineWidth = PFNGLLINEWIDTHPROC(glfwGetProcAddress("glLineWidth"));
0231     glLinkProgram = PFNGLLINKPROGRAMPROC(glfwGetProcAddress("glLinkProgram"));
0232     glListDrawCommandsStatesClientNV =
0233 PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC(glfwGetProcAddress("glListDrawCommandsStatesClientNV"));
0234     glLogicOp = PFNGLLOGICOPPROC(glfwGetProcAddress("glLogicOp"));
0235     glMakeBufferNonResidentNV =
0236 PFNGLMAKEBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferNonResidentNV"));
0237     glMakeBufferResidentNV =
0238 PFNGLMAKEBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferResidentNV"));
0239     glMakeImageHandleNonResidentARB =
0240 PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleNonResidentARB"));
0241     glMakeImageHandleNonResidentNV =
0242 PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleNonResidentNV"));
0243     glMakeImageHandleResidentARB =
0244 PFNGLMAKEIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleResidentARB"));
0245     glMakeImageHandleResidentNV =
0246 PFNGLMAKEIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleResidentNV"));
0247     glMakeNamedBufferNonResidentNV =
0248 PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferNonResidentNV"));
0249     glMakeNamedBufferResidentNV =
0250 PFNGLMAKENAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferResidentNV"));
0251     glMakeTextureHandleNonResidentARB =
0252 PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentARB"));
0253     glMakeTextureHandleNonResidentNV =
0254 PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentNV"));
0255     glMakeTextureHandleResidentARB =
0256 PFNGLMAKETEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleResidentARB"));
0257     glMakeTextureHandleResidentNV =
0258 PFNGLMAKETEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleResidentNV"));
0259     glMapBuffer = PFNGLMAPBUFFERPROC(glfwGetProcAddress("glMapBuffer"));
0260     glMapBufferRange = PFNGLMAPBUFFERRANGEPROC(glfwGetProcAddress("glMapBufferRange"));
0261     glMapNamedBuffer = PFNGLMAPNAMEDBUFFERPROC(glfwGetProcAddress("glMapNamedBuffer"));
0262     glMapNamedBufferEXT = PFNGLMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glMapNamedBufferEXT"));
0263     glMapNamedBufferRange = PFNGLMAPNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glMapNamedBufferRange"));
0264     glMapNamedBufferRangeEXT =
0265 PFNGLMAPNAMEDBUFERRANGEEXTPROC(glfwGetProcAddress("glMapNamedBufferRangeEXT"));
0266     glMatrixFrustumEXT = PFNGLMATRIXFRUSTUMEXTPROC(glfwGetProcAddress("glMatrixFrustumEXT"));
0267     glMatrixLoad3x2fNV = PFNGLMATRIXLOAD3X2FNVPROC(glfwGetProcAddress("glMatrixLoad3x2fNV"));
0268     glMatrixLoad3x3fNV = PFNGLMATRIXLOAD3X3FNVPROC(glfwGetProcAddress("glMatrixLoad3x3fNV"));
0269     glMatrixLoadIdentityEXT =
0270 PFNGLMATRIXLOADIDENTITYEXTPROC(glfwGetProcAddress("glMatrixLoadIdentityEXT"));
0271     glMatrixLoadTranspose3x3fNV =
0272 PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixLoadTranspose3x3fNV"));

```

```

01956     glMatrixLoadTransposedEXT =
01957     PFNGLMATRIXLOADTRANPOSEDEXTPROC(glfwGetProcAddress("glMatrixLoadTransposedEXT"));
01958     glMatrixLoadTransposefEXT =
01959     PFNGLMATRIXLOADTRANPOSEFEXTPROC(glfwGetProcAddress("glMatrixLoadTransposefEXT"));
01960     glMatrixLoaddEXT = PFNGLMATRIXLOADDEXTPROC(glfwGetProcAddress("glMatrixLoaddEXT"));
01961     glMatrixLoadfEXT = PFNGLMATRIXLOADFEXTPROC(glfwGetProcAddress("glMatrixLoadfEXT"));
01962     glMatrixMult3x2fNV = PFNGLMATRIXMULT3X2FNVPROC(glfwGetProcAddress("glMatrixMult3x2fNV"));
01963     glMatrixMult3x3fNV = PFNGLMATRIXMULT3X3FNVPROC(glfwGetProcAddress("glMatrixMult3x3fNV"));
01964     glMatrixMultTranspose3x3fNV =
01965     PFNGLMATRIXMULTTRANPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixMultTranspose3x3fNV"));
01966     glMatrixMultTransposedEXT =
01967     PFNGLMATRIXMULTTRANPOSEDEXTPROC(glfwGetProcAddress("glMatrixMultTransposedEXT"));
01968     glMatrixMultTransposefEXT =
01969     PFNGLMATRIXMULTTRANPOSEFEXTPROC(glfwGetProcAddress("glMatrixMultTransposefEXT"));
01970     glMatrixMultdEXT = PFNGLMATRIXMULTDEXTPROC(glfwGetProcAddress("glMatrixMultdEXT"));
01971     glMatrixMultfEXT = PFNGLMATRIXMULTFEXTPROC(glfwGetProcAddress("glMatrixMultfEXT"));
01972     glMatrixOrthoEXT = PFNGLMATRIXORTHOEXTPROC(glfwGetProcAddress("glMatrixOrthoEXT"));
01973     glMatrixPopEXT = PFNGLMATRIXPOPEXTPROC(glfwGetProcAddress("glMatrixPopEXT"));
01974     glMatrixPushEXT = PFNGLMATRIXPUSHEXTPROC(glfwGetProcAddress("glMatrixPushEXT"));
01975     glMatrixRotatedEXT = PFNGLMATRIXROTATEDEXTPROC(glfwGetProcAddress("glMatrixRotatedEXT"));
01976     glMatrixRotatefEXT = PFNGLMATRIXROTATEFEXTPROC(glfwGetProcAddress("glMatrixRotatefEXT"));
01977     glMatrixScaledEXT = PFNGLMATRIXSCALEDEXTPROC(glfwGetProcAddress("glMatrixScaledEXT"));
01978     glMatrixTranslatedEXT = PFNGLMATRIXTRANSLATEDEXTPROC(glfwGetProcAddress("glMatrixTranslatedEXT"));
01979     glMatrixTranslatedNV = PFNGLMATRIXTRANSLATEDNVPROC(glfwGetProcAddress("glMatrixTranslatedNV"));
01980     glMemoryBarrier = PFNGLMEMORYBARRIERPROC(glfwGetProcAddress("glMemoryBarrier"));
01981     glMemoryBarrierByRegion =
01982     PFNGLMEMORYBARRIERBYREGIONPROC(glfwGetProcAddress("glMemoryBarrierByRegion"));
01983     glMinSampleShading = PFNGLMINSAMPLESHADINGPROC(glfwGetProcAddress("glMinSampleShading"));
01984     glMinSampleShadingARB = PFNGLMINSAMPLESHADINGARBPROC(glfwGetProcAddress("glMinSampleShadingARB"));
01985     glMultiDrawArrays = PFNGLMULTIDRAWARRAYSPROC(glfwGetProcAddress("glMultiDrawArrays"));
01986     glMultiDrawArraysIndirect =
01987     PFNGLMULTIDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glMultiDrawArraysIndirect"));
01988     glMultiDrawArraysIndirectBindlessCountNV =
01989     PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessCountNV"));
01990     glMultiDrawArraysIndirectBindlessNV =
01991     PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessNV"));
01992     glMultiDrawElements = PFNGLMULTIDRAWELEMENTSPROC(glfwGetProcAddress("glMultiDrawElements"));
01993     glMultiDrawElementsBaseVertex =
01994     PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glMultiDrawElementsBaseVertex"));
01995     glMultiDrawElementsIndirect =
01996     PFNGLMULTIDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glMultiDrawElementsIndirect"));
01997     glMultiDrawElementsIndirectBindlessCountNV =
01998     PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessCountNV"));
01999     glMultiDrawElementsIndirectBindlessNV =
02000     PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessNV"));
02001     glMultiDrawElementsIndirectCountARB =
02002     PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawElementsIndirectCountARB"));
02003     glMultiTexBufferEXT = PFNGLMULTITEXBUFFEREXTPROC(glfwGetProcAddress("glMultiTexBufferEXT"));
02004     glMultiTexCoordPointerEXT =
02005     PFNGLMULTITEXCOORDPOINTERTEXPROC(glfwGetProcAddress("glMultiTexCoordPointerEXT"));
02006     glMultiTexEnvfEXT = PFNGLMULTITEXENVFEXTPROC(glfwGetProcAddress("glMultiTexEnvfEXT"));
02007     glMultiTexEnvfvEXT =
02008     PFNGLMULTITEXENVFVEXTPROC(glfwGetProcAddress("glMultiTexEnvfvEXT"));
02009     glMultiTexEnvivEXT =
02010     PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
02011     glMultiTexEnvivEXT =
02012     PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
02013     glMultiTexEnvivEXT =
02014     PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
02015     glMultiTexImage1DEXT =
02016     PFNGLMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexImage1DEXT"));
02017     glMultiTexImage2DEXT =
02018     PFNGLMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexImage2DEXT"));
02019     glMultiTexImage3DEXT =
02020     PFNGLMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexImage3DEXT"));
02021     glMultiTexParameterIiEXT =
02022     PFNGLMULTITEXPARAMETERIIEXTPROC(glfwGetProcAddress("glMultiTexParameterIiEXT"));
02023     glMultiTexParameterIiivEXT =
02024     PFNGLMULTITEXPARAMETERIIIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIiivEXT"));
02025     glMultiTexParameterfEXT =
02026     PFNGLMULTITEXPARAMETERFEXTPROC(glfwGetProcAddress("glMultiTexParameterfEXT"));
02027     glMultiTexParameterfvEXT =
02028     PFNGLMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glMultiTexParameterfvEXT"));
02029     glMultiTexParameteriEXT =
02030     PFNGLMULTITEXPARAMETERIEXTPROC(glfwGetProcAddress("glMultiTexParameteriEXT"));
02031     glMultiTexParameterivEXT =
02032     PFNGLMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glMultiTexParameterivEXT"));
02033     glMultiTexRenderbufferEXT =
02034     PFNGLMULTITEXRENDERBUFFEREXTPROC(glfwGetProcAddress("glMultiTexRenderbufferEXT"));
02035     glMultiTexSubImage1DEXT =
02036     PFNGLMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexSubImage1DEXT"));
02037     glMultiTexSubImage2DEXT =
02038     PFNGLMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexSubImage2DEXT"));
02039     glMultiTexSubImage3DEXT =

```

```

PFNGLMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexSubImage3DEXT"));
02017 gNamedBufferData = PFNGLNAMEDBUFFERDATAPROC(glfwGetProcAddress("gNamedBufferData"));
02018 gNamedBufferDataEXT = PFNGLNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("gNamedBufferDataEXT"));
02019 gNamedBufferPageCommitmentARB =
PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("gNamedBufferPageCommitmentARB"));
02020 gNamedBufferPageCommitmentEXT =
PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC(glfwGetProcAddress("gNamedBufferPageCommitmentEXT"));
02021 gNamedBufferStorage = PFNGLNAMEDBUFFERSTORAGEPROC(glfwGetProcAddress("gNamedBufferStorage"));
02022 gNamedBufferStorageEXT =
PFNGLNAMEDBUFFERSTORAGEEXTPROC(glfwGetProcAddress("gNamedBufferStorageEXT"));
02023 gNamedBufferSubData = PFNGLNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("gNamedBufferSubData"));
02024 gNamedBufferSubDataEXT =
PFNGLNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("gNamedBufferSubDataEXT"));
02025 gNamedCopyBufferSubDataEXT =
PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC(glfwGetProcAddress("gNamedCopyBufferSubDataEXT"));
02026 gNamedFramebufferDrawBuffer =
PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC(glfwGetProcAddress("gNamedFramebufferDrawBuffer"));
02027 gNamedFramebufferDrawBuffers =
PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC(glfwGetProcAddress("gNamedFramebufferDrawBuffers"));
02028 gNamedFramebufferParameteri =
PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("gNamedFramebufferParameteri"));
02029 gNamedFramebufferParameteriEXT =
PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC(glfwGetProcAddress("gNamedFramebufferParameteriEXT"));
02030 gNamedFramebufferReadBuffer =
PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC(glfwGetProcAddress("gNamedFramebufferReadBuffer"));
02031 gNamedFramebufferRenderbuffer =
PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("gNamedFramebufferRenderbuffer"));
02032 gNamedFramebufferRenderbufferEXT =
PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC(glfwGetProcAddress("gNamedFramebufferRenderbufferEXT"));
02033 gNamedFramebufferSampleLocationsfvARB =
PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSVARBPROC(glfwGetProcAddress("gNamedFramebufferSampleLocationsfvARB"));
02034 gNamedFramebufferSampleLocationsfvNV =
PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("gNamedFramebufferSampleLocationsfvNV"));
02035 gNamedFramebufferTexture =
PFNGLNAMEDFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("gNamedFramebufferTexture"));
02036 gNamedFramebufferTexture1DEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC(glfwGetProcAddress("gNamedFramebufferTexture1DEXT"));
02037 gNamedFramebufferTexture2DEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC(glfwGetProcAddress("gNamedFramebufferTexture2DEXT"));
02038 gNamedFramebufferTexture3DEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC(glfwGetProcAddress("gNamedFramebufferTexture3DEXT"));
02039 gNamedFramebufferTextureEXT =
PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC(glfwGetProcAddress("gNamedFramebufferTextureEXT"));
02040 gNamedFramebufferTextureFaceEXT =
PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC(glfwGetProcAddress("gNamedFramebufferTextureFaceEXT"));
02041 gNamedFramebufferTextureLayer =
PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("gNamedFramebufferTextureLayer"));
02042 gNamedFramebufferTextureLayerEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC(glfwGetProcAddress("gNamedFramebufferTextureLayerEXT"));
02043 gNamedProgramLocalParameter4dEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameter4dEXT"));
02044 gNamedProgramLocalParameter4dvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameter4dvEXT"));
02045 gNamedProgramLocalParameter4fEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameter4fEXT"));
02046 gNamedProgramLocalParameter4fvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameter4fvEXT"));
02047 gNamedProgramLocalParameterI4iEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameterI4iEXT"));
02048 gNamedProgramLocalParameterI4ivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameterI4ivEXT"));
02049 gNamedProgramLocalParameterI4uiEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameterI4uiEXT"));
02050 gNamedProgramLocalParameterI4uivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameterI4uivEXT"));
02051 gNamedProgramLocalParameters4fvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParameters4fvEXT"));
02052 gNamedProgramLocalParametersI4ivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IEXTPROC(glfwGetProcAddress("gNamedProgramLocalParametersI4ivEXT"));
02053 gNamedProgramLocalParametersI4uivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC(glfwGetProcAddress("gNamedProgramLocalParametersI4uivEXT"));
02054 gNamedProgramStringEXT =
PFNGLNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("gNamedProgramStringEXT"));
02055 gNamedRenderbufferStorage =
PFNGLNAMEDRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("gNamedRenderbufferStorage"));
02056 gNamedRenderbufferStorageEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC(glfwGetProcAddress("gNamedRenderbufferStorageEXT"));
02057 gNamedRenderbufferStorageMultisample =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("gNamedRenderbufferStorageMultisample"));
02058 gNamedRenderbufferStorageMultisampleCoverageEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC(glfwGetProcAddress("gNamedRenderbufferStorageMultisampleCoverage"));
02059 gNamedRenderbufferStorageMultisampleEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC(glfwGetProcAddress("gNamedRenderbufferStorageMultisampleEXT"));
02060 gNamedStringARB = PFNGLNAMEDSTRINGARBPROC(glfwGetProcAddress("gNamedStringARB"));
02061 gNormalFormatNV = PFNGLNORMFORMATNVPROC(glfwGetProcAddress("gNormalFormatNV"));
02062 gObjectLabel = PFNGLOBJECTLABELPROC(glfwGetProcAddress("gObjectLabel"));
02063 gObjectPtrLabel = PFNGLOBJECTPTRLABELPROC(glfwGetProcAddress("gObjectPtrLabel"));

```

```

02064 glPatchParameterfv = PFNGLPATCHPARAMETERFVPROC(glfwGetProcAddress("glPatchParameterfv"));
02065 glPatchParameteri = PFNGLPATCHPARAMETERIPROC(glfwGetProcAddress("glPatchParameteri"));
02066 glPathCommandsNV = PFNGLPATHCOMMANDSNVPROC(glfwGetProcAddress("glPathCommandsNV"));
02067 glPathCoordsNV = PFNGLPATHCOORDSNVPROC(glfwGetProcAddress("glPathCoordsNV"));
02068 glPathCoverDepthFuncNV =
PFNGLPATHCOVERDEPTHFUNCNVPROC(glfwGetProcAddress("glPathCoverDepthFuncNV"));
02069 glPathDashArrayNV = PFNGLPATHDASHARRAYNVPROC(glfwGetProcAddress("glPathDashArrayNV"));
02070 glPathGlyphIndexArrayNV =
PFNGLPATHGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathGlyphIndexArrayNV"));
02071 glPathGlyphIndexRangeNV =
PFNGLPATHGLYPHINDEXRANGENVPROC(glfwGetProcAddress("glPathGlyphIndexRangeNV"));
02072 glPathGlyphRangeNV = PFNGLPATHGLYPHRANGEPROC(glfwGetProcAddress("glPathGlyphRangeNV"));
02073 glPathGlyphsNV = PFNGLPATHGLYPHSNVPROC(glfwGetProcAddress("glPathGlyphsNV"));
02074 glPathMemoryGlyphIndexArrayNV =
PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathMemoryGlyphIndexArrayNV"));
02075 glPathParameterfNV = PFNGLPATHPARAMETERFNVPROC(glfwGetProcAddress("glPathParameterfNV"));
02076 glPathParameterfvNV = PFNGLPATHPARAMETERFVNVPROC(glfwGetProcAddress("glPathParameterfvNV"));
02077 glPathParameteriNV = PFNGLPATHPARAMETERINVPROC(glfwGetProcAddress("glPathParameteriNV"));
02078 glPathParameterivNV = PFNGLPATHPARAMETERIVNVPROC(glfwGetProcAddress("glPathParameterivNV"));
02079 glPathStencilDepthOffsetNV =
PFNGLPATHSTENCILDEPTHOFFSETNVPROC(glfwGetProcAddress("glPathStencilDepthOffsetNV"));
02080 glPathStencilFuncNV = PFNGLPATHSTENCILFUNCNVPROC(glfwGetProcAddress("glPathStencilFuncNV"));
02081 glPathStringNV = PFNGLPATHSTRINGNVPROC(glfwGetProcAddress("glPathStringNV"));
02082 glPathSubCommandsNV = PFNGLPATHSUBCOMMANDSNVPROC(glfwGetProcAddress("glPathSubCommandsNV"));
02083 glPathSubCoordsNV = PFNGLPATHSUBCOORDSNVPROC(glfwGetProcAddress("glPathSubCoordsNV"));
02084 glPauseTransformFeedback =
PFNGLPAUSETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glPauseTransformFeedback"));
02085 glPixelStore = PFNGLPIXELSTOREFPROC(glfwGetProcAddress("glPixelStoref"));
02086 glPixelStorei = PFNGLPIXELSTOREIPROC(glfwGetProcAddress("glPixelStorei"));
02087 glPointAlongPathNV = PFNGLPOINTALONGPATHNVPROC(glfwGetProcAddress("glPointAlongPathNV"));
02088 glPointParameterf = PFNGLPOINTPARAMETERFPROC(glfwGetProcAddress("glPointParameterf"));
02089 glPointParameterfv = PFNGLPOINTPARAMETERFVPROC(glfwGetProcAddress("glPointParameterfv"));
02090 glPointParameteri = PFNGLPOINTPARAMETERIPROC(glfwGetProcAddress("glPointParameteri"));
02091 glPointParameteriv = PFNGLPOINTPARAMETERIVPROC(glfwGetProcAddress("glPointParameteriv"));
02092 glPointSize = PFNGLPOINTSIZEPROC(glfwGetProcAddress("glPointSize"));
02093 glPolygonMode = PFNGLPOLYGONMODEPROC(glfwGetProcAddress("glPolygonMode"));
02094 glPolygonOffset = PFNGLPOLYGOFFSETPROC(glfwGetProcAddress("glPolygonOffset"));
02095 glPolygonOffsetClampEXT =
PFNGLPOLYGONOFFSETCLAMPEXTPROC(glfwGetProcAddress("glPolygonOffsetClampEXT"));
02096 glPopDebugGroup = PFNGLPOPODEBUGGROUPPROC(glfwGetProcAddress("glPopDebugGroup"));
02097 glPopGroupMarkerEXT = PFNGLPOPGROUPMARKEREXTPROC(glfwGetProcAddress("glPopGroupMarkerEXT"));
02098 glPrimitiveBoundingBoxARB =
PFNGLPRIMITIVEBOXARBPROC(glfwGetProcAddress("glPrimitiveBoundingBoxARB"));
02099 glPrimitiveRestartIndex =
PFNGLPRIMITIVERESTARTINDEXPROC(glfwGetProcAddress("glPrimitiveRestartIndex"));
02100 glProgramBinary = PFNGLPROGRAMBINARYPROC(glfwGetProcAddress("glProgramBinary"));
02101 glProgramParameteri = PFNGLPROGRAMPARAMETERIPROC(glfwGetProcAddress("glProgramParameteri"));
02102 glProgramParameteriARB =
PFNGLPROGRAMPARAMETERIARBPROC(glfwGetProcAddress("glProgramParameteriARB"));
02103 glProgramPathFragmentInputGenNV =
PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC(glfwGetProcAddress("glProgramPathFragmentInputGenNV"));
02104 glProgramUniform1d = PFNGLPROGRAMUNIFORM1DPROC(glfwGetProcAddress("glProgramUniform1d"));
02105 glProgramUniform1dEXT = PFNGLPROGRAMUNIFORM1DEXTPROC(glfwGetProcAddress("glProgramUniform1dEXT"));
02106 glProgramUniform1dv = PFNGLPROGRAMUNIFORM1DVPROC(glfwGetProcAddress("glProgramUniform1dv"));
02107 glProgramUniform1dvEXT =
PFNGLPROGRAMUNIFORM1DVEXTPROC(glfwGetProcAddress("glProgramUniform1dvEXT"));
02108 glProgramUniform1f = PFNGLPROGRAMUNIFORM1FPROC(glfwGetProcAddress("glProgramUniform1f"));
02109 glProgramUniform1fEXT = PFNGLPROGRAMUNIFORM1FEXTPROC(glfwGetProcAddress("glProgramUniform1fEXT"));
02110 glProgramUniform1fv = PFNGLPROGRAMUNIFORM1FVPROC(glfwGetProcAddress("glProgramUniform1fv"));
02111 glProgramUniform1fvEXT =
PFNGLPROGRAMUNIFORM1FVEXTPROC(glfwGetProcAddress("glProgramUniform1fvEXT"));
02112 glProgramUniform1i = PFNGLPROGRAMUNIFORM1IPROC(glfwGetProcAddress("glProgramUniform1i"));
02113 glProgramUniform1i64ARB =
PFNGLPROGRAMUNIFORM1I64ARBPROC(glfwGetProcAddress("glProgramUniform1i64ARB"));
02114 glProgramUniform1i64NV =
PFNGLPROGRAMUNIFORM1I64NVPROC(glfwGetProcAddress("glProgramUniform1i64NV"));
02115 glProgramUniform1i64vARB =
PFNGLPROGRAMUNIFORM1I64VARBPROC(glfwGetProcAddress("glProgramUniform1i64vARB"));
02116 glProgramUniform1i64vNV =
PFNGLPROGRAMUNIFORM1I64NVPROC(glfwGetProcAddress("glProgramUniform1i64vNV"));
02117 glProgramUniform1iEXT = PFNGLPROGRAMUNIFORM1IEXTPROC(glfwGetProcAddress("glProgramUniform1iEXT"));
02118 glProgramUniform1iv = PFNGLPROGRAMUNIFORM1IVPROC(glfwGetProcAddress("glProgramUniform1iv"));
02119 glProgramUniform1ivEXT =
PFNGLPROGRAMUNIFORM1IVEXTPROC(glfwGetProcAddress("glProgramUniform1ivEXT"));
02120 glProgramUniform1ui = PFNGLPROGRAMUNIFORM1UIPROC(glfwGetProcAddress("glProgramUniform1ui"));
02121 glProgramUniform1ui64ARB =
PFNGLPROGRAMUNIFORM1UI64ARBPROC(glfwGetProcAddress("glProgramUniform1ui64ARB"));
02122 glProgramUniform1ui64NV =
PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniform1ui64NV"));
02123 glProgramUniform1ui64vARB =
PFNGLPROGRAMUNIFORM1UI64VARBPROC(glfwGetProcAddress("glProgramUniform1ui64vARB"));
02124 glProgramUniform1ui64vNV =
PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniform1ui64vNV"));
02125 glProgramUniform1uiEXT =
PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniform1uiEXT"));
02126 glProgramUniform1uiv = PFNGLPROGRAMUNIFORM1UIVPROC(glfwGetProcAddress("glProgramUniform1uiv"));
02127 glProgramUniform1uivEXT =

```

```

PFNGLPROGRAMUNIFORM1UIVEXTPROC(glfwGetProcAddress("glProgramUniform1uvEXT"));
02128 glProgramUniform2d = PFNGLPROGRAMUNIFORM2DFPROC(glfwGetProcAddress("glProgramUniform2d"));
02129 glProgramUniform2dEXT = PFNGLPROGRAMUNIFORM2DEXTPROC(glfwGetProcAddress("glProgramUniform2dEXT"));
02130 glProgramUniform2dv = PFNGLPROGRAMUNIFORM2DVPROC(glfwGetProcAddress("glProgramUniform2dv"));
02131 glProgramUniform2dvEXT =
PFNGLPROGRAMUNIFORM2DVEXTPROC(glfwGetProcAddress("glProgramUniform2dvEXT"));
02132 glProgramUniform2f = PFNGLPROGRAMUNIFORM2FPROC(glfwGetProcAddress("glProgramUniform2f"));
02133 glProgramUniform2fEXT = PFNGLPROGRAMUNIFORM2FEXTPROC(glfwGetProcAddress("glProgramUniform2fEXT"));
02134 glProgramUniform2fv = PFNGLPROGRAMUNIFORM2FVPROC(glfwGetProcAddress("glProgramUniform2fv"));
02135 glProgramUniform2fvEXT =
PFNGLPROGRAMUNIFORM2FVEXTPROC(glfwGetProcAddress("glProgramUniform2fvEXT"));
02136 glProgramUniform2i = PFNGLPROGRAMUNIFORM2IPROC(glfwGetProcAddress("glProgramUniform2i"));
02137 glProgramUniform2i64ARB =
PFNGLPROGRAMUNIFORM2I64ARBPROC(glfwGetProcAddress("glProgramUniform2i64ARB"));
02138 glProgramUniform2i64NV =
PFNGLPROGRAMUNIFORM2I64NVPROC(glfwGetProcAddress("glProgramUniform2i64NV"));
02139 glProgramUniform2i64vARB =
PFNGLPROGRAMUNIFORM2I64VARBPROC(glfwGetProcAddress("glProgramUniform2i64vARB"));
02140 glProgramUniform2i64vNV =
PFNGLPROGRAMUNIFORM2I64VNPROC(glfwGetProcAddress("glProgramUniform2i64vNV"));
02141 glProgramUniform2iEXT = PFNGLPROGRAMUNIFORM2IEXTPROC(glfwGetProcAddress("glProgramUniform2iEXT"));
02142 glProgramUniform2iv = PFNGLPROGRAMUNIFORM2IVPROC(glfwGetProcAddress("glProgramUniform2iv"));
02143 glProgramUniform2ivEXT =
PFNGLPROGRAMUNIFORM2IVEXTPROC(glfwGetProcAddress("glProgramUniform2ivEXT"));
02144 glProgramUniform2ui = PFNGLPROGRAMUNIFORM2UIPROC(glfwGetProcAddress("glProgramUniform2ui"));
02145 glProgramUniform2ui64ARB =
PFNGLPROGRAMUNIFORM2UI64ARBPROC(glfwGetProcAddress("glProgramUniform2ui64ARB"));
02146 glProgramUniform2ui64NV =
PFNGLPROGRAMUNIFORM2UI64NVPROC(glfwGetProcAddress("glProgramUniform2ui64NV"));
02147 glProgramUniform2ui64vARB =
PFNGLPROGRAMUNIFORM2UI64VARBPROC(glfwGetProcAddress("glProgramUniform2ui64vARB"));
02148 glProgramUniform2ui64vNV =
PFNGLPROGRAMUNIFORM2UI64VNPROC(glfwGetProcAddress("glProgramUniform2ui64vNV"));
02149 glProgramUniform2uiEXT =
PFNGLPROGRAMUNIFORM2UIEXTPROC(glfwGetProcAddress("glProgramUniform2uiEXT"));
02150 glProgramUniform2uiv = PFNGLPROGRAMUNIFORM2UIVPROC(glfwGetProcAddress("glProgramUniform2uiv"));
02151 glProgramUniform2uivEXT =
PFNGLPROGRAMUNIFORM2UIVEXTPROC(glfwGetProcAddress("glProgramUniform2uivEXT"));
02152 glProgramUniform3d = PFNGLPROGRAMUNIFORM3DFPROC(glfwGetProcAddress("glProgramUniform3d"));
02153 glProgramUniform3dEXT = PFNGLPROGRAMUNIFORM3DDEXTPROC(glfwGetProcAddress("glProgramUniform3dEXT"));
02154 glProgramUniform3dv = PFNGLPROGRAMUNIFORM3DVPROC(glfwGetProcAddress("glProgramUniform3dv"));
02155 glProgramUniform3dvEXT =
PFNGLPROGRAMUNIFORM3DVEXTPROC(glfwGetProcAddress("glProgramUniform3dvEXT"));
02156 glProgramUniform3f = PFNGLPROGRAMUNIFORM3FPROC(glfwGetProcAddress("glProgramUniform3f"));
02157 glProgramUniform3fEXT = PFNGLPROGRAMUNIFORM3FEXTPROC(glfwGetProcAddress("glProgramUniform3fEXT"));
02158 glProgramUniform3fv = PFNGLPROGRAMUNIFORM3FVPROC(glfwGetProcAddress("glProgramUniform3fv"));
02159 glProgramUniform3fvEXT =
PFNGLPROGRAMUNIFORM3FVEXTPROC(glfwGetProcAddress("glProgramUniform3fvEXT"));
02160 glProgramUniform3i = PFNGLPROGRAMUNIFORM3IPROC(glfwGetProcAddress("glProgramUniform3i"));
02161 glProgramUniform3i64ARB =
PFNGLPROGRAMUNIFORM3I64ARBPROC(glfwGetProcAddress("glProgramUniform3i64ARB"));
02162 glProgramUniform3i64NV =
PFNGLPROGRAMUNIFORM3I64NVPROC(glfwGetProcAddress("glProgramUniform3i64NV"));
02163 glProgramUniform3i64vARB =
PFNGLPROGRAMUNIFORM3I64VARBPROC(glfwGetProcAddress("glProgramUniform3i64vARB"));
02164 glProgramUniform3i64vNV =
PFNGLPROGRAMUNIFORM3I64VNPROC(glfwGetProcAddress("glProgramUniform3i64vNV"));
02165 glProgramUniform3iEXT = PFNGLPROGRAMUNIFORM3IEXTPROC(glfwGetProcAddress("glProgramUniform3iEXT"));
02166 glProgramUniform3iv = PFNGLPROGRAMUNIFORM3IVPROC(glfwGetProcAddress("glProgramUniform3iv"));
02167 glProgramUniform3ivEXT =
PFNGLPROGRAMUNIFORM3IVEXTPROC(glfwGetProcAddress("glProgramUniform3ivEXT"));
02168 glProgramUniform3ui = PFNGLPROGRAMUNIFORM3UIPROC(glfwGetProcAddress("glProgramUniform3ui"));
02169 glProgramUniform3ui64ARB =
PFNGLPROGRAMUNIFORM3UI64ARBPROC(glfwGetProcAddress("glProgramUniform3ui64ARB"));
02170 glProgramUniform3ui64NV =
PFNGLPROGRAMUNIFORM3UI64NVPROC(glfwGetProcAddress("glProgramUniform3ui64NV"));
02171 glProgramUniform3ui64vARB =
PFNGLPROGRAMUNIFORM3UI64VARBPROC(glfwGetProcAddress("glProgramUniform3ui64vARB"));
02172 glProgramUniform3ui64vNV =
PFNGLPROGRAMUNIFORM3UI64VNPROC(glfwGetProcAddress("glProgramUniform3ui64vNV"));
02173 glProgramUniform3uiEXT =
PFNGLPROGRAMUNIFORM3UIEXTPROC(glfwGetProcAddress("glProgramUniform3uiEXT"));
02174 glProgramUniform3uiv = PFNGLPROGRAMUNIFORM3UIVPROC(glfwGetProcAddress("glProgramUniform3uiv"));
02175 glProgramUniform3uivEXT =
PFNGLPROGRAMUNIFORM3UIVEXTPROC(glfwGetProcAddress("glProgramUniform3uivEXT"));
02176 glProgramUniform4d = PFNGLPROGRAMUNIFORM4DFPROC(glfwGetProcAddress("glProgramUniform4d"));
02177 glProgramUniform4dEXT = PFNGLPROGRAMUNIFORM4DDEXTPROC(glfwGetProcAddress("glProgramUniform4dEXT"));
02178 glProgramUniform4dv = PFNGLPROGRAMUNIFORM4DVPROC(glfwGetProcAddress("glProgramUniform4dv"));
02179 glProgramUniform4dvEXT =
PFNGLPROGRAMUNIFORM4DVEXTPROC(glfwGetProcAddress("glProgramUniform4dvEXT"));
02180 glProgramUniform4f = PFNGLPROGRAMUNIFORM4FPROC(glfwGetProcAddress("glProgramUniform4f"));
02181 glProgramUniform4fEXT = PFNGLPROGRAMUNIFORM4FEXTPROC(glfwGetProcAddress("glProgramUniform4fEXT"));
02182 glProgramUniform4fv = PFNGLPROGRAMUNIFORM4FVPROC(glfwGetProcAddress("glProgramUniform4fv"));
02183 glProgramUniform4fvEXT =
PFNGLPROGRAMUNIFORM4FVEXTPROC(glfwGetProcAddress("glProgramUniform4fvEXT"));
02184 glProgramUniform4i = PFNGLPROGRAMUNIFORM4IPROC(glfwGetProcAddress("glProgramUniform4i"));
02185 glProgramUniform4i64ARB =

```

```

02186 PFNGLPROGRAMUNIFORM4I64ARBPROC(glfwGetProcAddress("glProgramUniform4i64ARB"));
02187     glProgramUniform4i64NV =
02188 PFNGLPROGRAMUNIFORM4I64NVPROC(glfwGetProcAddress("glProgramUniform4i64NV"));
02189     glProgramUniform4i64vARB =
02190 PFNGLPROGRAMUNIFORM4I64VARBPROC(glfwGetProcAddress("glProgramUniform4i64vARB"));
02191     glProgramUniform4i64vNV =
02192 PFNGLPROGRAMUNIFORM4I64NVPROC(glfwGetProcAddress("glProgramUniform4i64vNV"));
02193     glProgramUniform4ivEXT = PFNGLPROGRAMUNIFORM4IEXTPROC(glfwGetProcAddress("glProgramUniform4iEXT"));
02194     glProgramUniform4iv = PFNGLPROGRAMUNIFORM4IVPROC(glfwGetProcAddress("glProgramUniform4iv"));
02195     glProgramUniform4ivEXT =
02196 PFNGLPROGRAMUNIFORM4IVEXTPROC(glfwGetProcAddress("glProgramUniform4ivEXT"));
02197     glProgramUniform4ui = PFNGLPROGRAMUNIFORM4UIPROC(glfwGetProcAddress("glProgramUniform4ui"));
02198     glProgramUniform4ui64ARB =
02199 PFNGLPROGRAMUNIFORM4UI64ARBPROC(glfwGetProcAddress("glProgramUniform4ui64ARB"));
02200     glProgramUniform4ui64NV =
02201 PFNGLPROGRAMUNIFORM4UI64NVPROC(glfwGetProcAddress("glProgramUniform4ui64NV"));
02202     glProgramUniform4ui64vARB =
02203 PFNGLPROGRAMUNIFORM4UI64VARBPROC(glfwGetProcAddress("glProgramUniform4ui64vARB"));
02204     glProgramUniform4ui64vNV =
02205 PFNGLPROGRAMUNIFORM4UI64VNVPROC(glfwGetProcAddress("glProgramUniform4ui64vNV"));
02206     glProgramUniform4uiEXT =
02207 PFNGLPROGRAMUNIFORM4UIEXTPROC(glfwGetProcAddress("glProgramUniform4uiEXT"));
02208     glProgramUniform4uiv = PFNGLPROGRAMUNIFORM4UIVPROC(glfwGetProcAddress("glProgramUniform4uiv"));
02209     glProgramUniform4uivEXT =
02210 PFNGLPROGRAMUNIFORM4UIVEXTPROC(glfwGetProcAddress("glProgramUniform4uivEXT"));
02211     glProgramUniformHandleui64ARB =
02212 PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glProgramUniformHandleui64ARB"));
02213     glProgramUniformHandleui64NV =
02214 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glProgramUniformHandleui64NV"));
02215     glProgramUniformMatrix2dv =
02216 PFNGLPROGRAMUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glProgramUniformMatrix2dv"));
02217     glProgramUniformMatrix2dEXT =
02218 PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2dEXT"));
02219     glProgramUniformMatrix2fv =
02220 PFNGLPROGRAMUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glProgramUniformMatrix2fv"));
02221     glProgramUniformMatrix2fvEXT =
02222 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2fvEXT"));
02223     glProgramUniformMatrix2x3dv =
02224 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dv"));
02225     glProgramUniformMatrix2x3dEXT =
02226 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dEXT"));
02227     glProgramUniformMatrix2x3fv =
02228 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fv"));
02229     glProgramUniformMatrix2x3fvEXT =
02230 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fvEXT"));
0230     glProgramUniformMatrix2x4dv =
0231 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dv"));
0232     glProgramUniformMatrix2x4dEXT =
0233 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dEXT"));
0234     glProgramUniformMatrix2x4fv =
0235 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fv"));
0236     glProgramUniformMatrix2x4fvEXT =
0237 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fvEXT"));
0238     glProgramUniformMatrix3dv =
0239 PFNGLPROGRAMUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glProgramUniformMatrix3dv"));
0240     glProgramUniformMatrix3dEXT =
0241 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3dEXT"));
0242     glProgramUniformMatrix3fv =
0243 PFNGLPROGRAMUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glProgramUniformMatrix3fv"));
0244     glProgramUniformMatrix3fvEXT =
0245 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3fvEXT"));
0246     glProgramUniformMatrix3x2dv =
0247 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dv"));
0248     glProgramUniformMatrix3x2dEXT =
0249 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dEXT"));
0250     glProgramUniformMatrix3x2fv =
0251 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fv"));
0252     glProgramUniformMatrix3x2fvEXT =
0253 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fvEXT"));
0254     glProgramUniformMatrix3x4dv =
0255 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dv"));
0256     glProgramUniformMatrix3x4dEXT =
0257 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dEXT"));
0258     glProgramUniformMatrix3x4fv =
0259 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fv"));
0260     glProgramUniformMatrix3x4fvEXT =
0261 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fvEXT"));
0262     glProgramUniformMatrix4dv =
0263 PFNGLPROGRAMUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glProgramUniformMatrix4dv"));
0264     glProgramUniformMatrix4dEXT =
0265 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4dEXT"));
0266     glProgramUniformMatrix4fv =
0267 PFNGLPROGRAMUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));
0268     glProgramUniformMatrix4fvEXT =
0269 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));

```

```

02231     glProgramUniformMatrix4fvEXT =
02232     PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fvEXT"));
02233     glProgramUniformMatrix4x2dv =
02234     PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dv"));
02235     glProgramUniformMatrix4x2dvEXT =
02236     PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dvEXT"));
02237     glProgramUniformMatrix4x2fv =
02238     PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fv"));
02239     glProgramUniformMatrix4x2fvEXT =
02240     PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fvEXT"));
02241     glProgramUniformMatrix4x3dv =
02242     PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dv"));
02243     glProgramUniformMatrix4x3dvEXT =
02244     PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dvEXT"));
02245     glProgramUniformMatrix4x3fv =
02246     PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fv"));
02247     glProgramUniformMatrix4x3fvEXT =
02248     PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fvEXT"));
02249     glProgramUniformui64NV =
02250     PFNGLPROGRAMUNIFORMUI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
02251     glProgramUniformui64NVNv =
02252     PFNGLPROGRAMUNIFORMUI64VNPROC(glfwGetProcAddress("glProgramUniformui64NVNv"));
02253     glProvokingVertex = PFNGLPROVOKINGVERTEXPROC(glfwGetProcAddress("glProvokingVertex"));
02254     glPushClientAttribDefaultEXT =
02255     PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glPushClientAttribDefaultEXT"));
02256     glPushDebugGroup = PFNGLPUSHDEBUGGROUPPROC(glfwGetProcAddress("glPushDebugGroup"));
02257     glPushGroupMarkerEXT = PFNGLPUSHGROUPMARKEREXTPROC(glfwGetProcAddress("glPushGroupMarkerEXT"));
02258     glQueryCounter = PFNGLQUERYCOUNTERPROC(glfwGetProcAddress("glQueryCounter"));
02259     glRasterSamplesEXT = PFNGLRASTERSAMPLESEXTPROC(glfwGetProcAddress("glRasterSamplesEXT"));
02260     glReadBuffer = PFNGLREADBUFFERPROC(glfwGetProcAddress("glReadBuffer"));
02261     glReadPixels = PFNGLREADPIXELSPROC(glfwGetProcAddress("glReadPixels"));
02262     glReadnPixels = PFNGLREADNPPIXELSPROC(glfwGetProcAddress("glReadnPixels"));
02263     glReadnPixelsARB = PFNGLREADNPPIXELSARBPROC(glfwGetProcAddress("glReadnPixelsARB"));
02264     glReleaseShaderCompiler =
02265     PFNGLRELEASESHADERCOMPILERPROC(glfwGetProcAddress("glReleaseShaderCompiler"));
02266     glRenderbufferStorage = PFNGLRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glRenderbufferStorage"));
02267     glRenderbufferStorageMultisample =
02268     PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glRenderbufferStorageMultisample"));
02269     glRenderbufferStorageMultisampleCoverageNV =
02270     PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC(glfwGetProcAddress("glRenderbufferStorageMultisampleCoverageNV"));
02271     glResolveDepthValuesNV =
02272     PFNGLRESOLVEDEPTHVALUESNVPROC(glfwGetProcAddress("glResolveDepthValuesNV"));
02273     glResumeTransformFeedback =
02274     PFNGLRESUMETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glResumeTransformFeedback"));
02275     glSampleCoverage = PFNGLSAMPLECOVERAGEPROC(glfwGetProcAddress("glSampleCoverage"));
02276     glSampleMaski = PFNGLSAMPLEMASKIPROC(glfwGetProcAddress("glSampleMaski"));
02277     glSamplerParameterIiv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameterIiv"));
02278     glSamplerParameterIuiv =
02279     PFNGLSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glSamplerParameterIuiv"));
02280     glSamplerParameterf = PFNGLSAMPLERPARAMETERFPROC(glfwGetProcAddress("glSamplerParameterf"));
02281     glSamplerParameterfv = PFNGLSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glSamplerParameterfv"));
02282     glSamplerParameteri = PFNGLSAMPLERPARAMETERIPROC(glfwGetProcAddress("glSamplerParameteri"));
02283     glSamplerParameteriv = PFNGLSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glSamplerParameteriv"));
02284     glScissor = PFNGLSCISSORPROC(glfwGetProcAddress("glScissor"));
02285     glScissorArrayv = PFNGLSCISSORARRAYVPROC(glfwGetProcAddress("glScissorArrayv"));
02286     glScissorIndexed = PFNGLSCISSORINDEXEDPROC(glfwGetProcAddress("glScissorIndexed"));
02287     glScissorIndexeddv = PFNGLSCISSORINDEXEDVPROC(glfwGetProcAddress("glScissorIndexeddv"));
02288     glSecondaryColorFormatNV =
02289     PFNGLSECONDARYCOLORFORMATNVPROC(glfwGetProcAddress("glSecondaryColorFormatNV"));
02290     glSelectPerfMonitorCountersAMD =
02291     PFNGLSELECTPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glSelectPerfMonitorCountersAMD"));
02292     glShaderBinary = PFNGLSHADERBINARYPROC(glfwGetProcAddress("glShaderBinary"));
02293     glShaderSource = PFNGLSHADERSOURCEPROC(glfwGetProcAddress("glShaderSource"));
02294     glShaderStorageBlockBinding =
02295     PFNGLSHADERSTORAGEBLOCKBINDINGPROC(glfwGetProcAddress("glShaderStorageBlockBinding"));
02296     glSignalVkFenceNV = PFNGLSIGNALKVFENCENVPROC(glfwGetProcAddress("glSignalVkFenceNV"));
02297     glSignalVkSemaphoreNV = PFNGLSIGNALKSEMAPHORENVPROC(glfwGetProcAddress("glSignalVkSemaphoreNV"));
02298     glSpecializeShaderARB = PFNGLSPECIALIZESHADERARBPROC(glfwGetProcAddress("glSpecializeShaderARB"));
02299     glStateCaptureNV = PFNGLSTATECAPTURENVPROC(glfwGetProcAddress("glStateCaptureNV"));
02300     glStencilFillPathInstancedNV =
02301     PFNGLSTENCILFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilFillPathInstancedNV"));
02302     glStencilFillPathNV = PFNGLSTENCILFILLPATHNVPROC(glfwGetProcAddress("glStencilFillPathNV"));
02303     glStencilFunc = PFNGLSTENCILFUNCPROC(glfwGetProcAddress("glStencilFunc"));
02304     glStencilFuncSeparate = PFNGLSTENCILFUNCSEPARATEPROC(glfwGetProcAddress("glStencilFuncSeparate"));
02305     glStencilMask = PFNGLSTENCILMASKPROC(glfwGetProcAddress("glStencilMask"));
02306     glStencilMaskSeparate = PFNGLSTENCILMASKSEPARATEPROC(glfwGetProcAddress("glStencilMaskSeparate"));
02307     glStencilOp = PFNGLSTENCILOPPROC(glfwGetProcAddress("glStencilOp"));
02308     glStencilOpSeparate = PFNGLSTENCILOPSEPARATEPROC(glfwGetProcAddress("glStencilOpSeparate"));
02309     glStencilStrokePathInstancedNV =
02310     PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilStrokePathInstancedNV"));
02311     glStencilStrokePathNV = PFNGLSTENCILSTROKEPATHNVPROC(glfwGetProcAddress("glStencilStrokePathNV"));
02312     glStencilThenCoverFillPathInstancedNV =
02313     PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathInstancedNV"));
02314     glStencilThenCoverFillPathNV =
02315     PFNGLSTENCILTHENCOVERFILLPATHNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathNV"));
02316     glStencilThenCoverStrokePathInstancedNV =
02317     PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathInstancedNV"));

```

```

02292     glStencilThenCoverStrokePathNV =
02293     PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathNV"));
02294     glSubpixelPrecisionBiasNV =
02295     PFNGLSUBPIXELPRECISIONBIASNVPROC(glfwGetProcAddress("glSubpixelPrecisionBiasNV"));
02296     glTexBuffer = PFNGLTEXBUFFERPROC(glfwGetProcAddress("glTexBufferARB"));
02297     glTexBufferARB = PFNGLTEXBUFFERARBPROC(glfwGetProcAddress("glTexBufferARB"));
02298     glTexCoordFormatNV = PFNGLTEXCOORDFORMATNVPROC(glfwGetProcAddress("glTexCoordFormatNV"));
02299     glTexImage1D = PFNGLTEXIMAGE1DPROC(glfwGetProcAddress("glTexImage1D"));
02300     glTexImage2D = PFNGLTEXIMAGE2DPROC(glfwGetProcAddress("glTexImage2D"));
02301     glTexImage2DMultisample =
02302     PFNGLTEXIMAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage2DMultisample"));
02303     glTexImage3D =
02304     PFNGLTEXIMAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage3DMultisample"));
02305     glTexImage3DMultisample =
02306     PFNGLTEXPAGECOMMITMENTARBPROC(glfwGetProcAddress("glTexPageCommitmentARB"));
02307     glTexParameterIiv = PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameterIiv"));
02308     glTexParameterIuiv = PFNGLTEXPARAMETERIUIVPROC(glfwGetProcAddress("glTexParameterIuiv"));
02309     glTexParameterIfv = PFNGLTEXPARAMETERIFVPROC(glfwGetProcAddress("glTexParameterIfv"));
02310     glTexParameterIi = PFNGLTEXPARAMETERIPROC(glfwGetProcAddress("glTexParameterIi"));
02311     glTexParameteriv = PFNGLTEXPARAMETERIVPROC(glfwGetProcAddress("glTexParameteriv"));
02312     glTexStorage1D = PFNGLTEXTSTORAGE1DPROC(glfwGetProcAddress("glTexStorage1D"));
02313     glTexStorage2D = PFNGLTEXTSTORAGE2DPROC(glfwGetProcAddress("glTexStorage2D"));
02314     glTexStorage2DMultisample =
02315     PFNGLTEXTSTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage2DMultisample"));
02316     glTexStorage3D = PFNGLTEXTSTORAGE3DPROC(glfwGetProcAddress("glTexStorage3D"));
02317     glTexStorage3DMultisample =
02318     PFNGLTEXTSTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage3DMultisample"));
02319     glTexSubImage1D = PFNGLTEXSUBIMAGE1DPROC(glfwGetProcAddress("glTexSubImage1D"));
02320     glTexSubImage2D = PFNGLTEXSUBIMAGE2DPROC(glfwGetProcAddress("glTexSubImage2D"));
02321     glTexSubImage3D = PFNGLTEXSUBIMAGE3DPROC(glfwGetProcAddress("glTexSubImage3D"));
02322     glTextureBarrier = PFNGLTEXTUREBARRIERPROC(glfwGetProcAddress("glTextureBarrier"));
02323     glTextureBarrierNV = PFNGLTEXTUREBARRIERNVPROC(glfwGetProcAddress("glTextureBarrierNV"));
02324     glTextureBuffer = PFNGLTEXTUREBUFFERPROC(glfwGetProcAddress("glTextureBuffer"));
02325     glTextureBufferEXT = PFNGLTEXTUREBUFFEREXTPROC(glfwGetProcAddress("glTextureBufferEXT"));
02326     glTextureBufferRange = PFNGLTEXTUREBUFFERRANGEPROC(glfwGetProcAddress("glTextureBufferRange"));
02327     glTextureBufferRangeEXT =
02328     PFNGLTEXTUREBUFFERRANGEEXTPROC(glfwGetProcAddress("glTextureBufferRangeEXT"));
02329     glTextureImage1DEXT = PFNGLTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glTextureImage1DEXT"));
02330     glTextureImage2DEXT = PFNGLTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glTextureImage2DEXT"));
02331     glTextureImage3DEXT = PFNGLTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glTextureImage3DEXT"));
02332     glTexturePageCommitmentEXT =
02333     PFNGLTEXTURECOMMITMENTEXTPROC(glfwGetProcAddress("glTexturePageCommitmentEXT"));
02334     glTextureParameterIiv = PFNGLTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glTextureParameterIiv"));
02335     glTextureParameterIiuiEXT =
02336     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIiuiEXT"));
02337     glTextureParameterIuivEXT =
02338     PFNGLTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02339     glTextureParameterIuivEXT =
02340     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02341     glTextureParameterIuivEXT =
02342     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02343     glTextureParameterIuivEXT =
02344     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02345     glTextureParameterIuivEXT =
02346     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02347     glTextureParameterIuivEXT =
02348     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02349     glTextureParameterIuivEXT =
02350     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02351     glTextureParameterIuivEXT =
02352     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02353     glTextureParameterIuivEXT =
02354     PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02355     glTextureParameterIuivEXT =

```

```

02356     glTextureSubImage3DEXT =
02357     PFNGLTEXTURESUBIMAGE3DDEXTPROC(glfwGetProcAddress("glTextureSubImage3DEXT"));
02358     glTextureView = PFNGLTEXTUREVIEWPROC(glfwGetProcAddress("glTextureView"));
02359     glTransformFeedbackBufferBase =
02360     PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC(glfwGetProcAddress("glTransformFeedbackBufferBase"));
02361     glTransformFeedbackBufferRange =
02362     PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC(glfwGetProcAddress("glTransformFeedbackBufferRange"));
02363     glTransformFeedbackVaryings =
02364     PFNGLTRANSFORMFEEDBACKVARYINGSPROC(glfwGetProcAddress("glTransformFeedbackVaryings"));
02365     glTransformPathNV = PFNGLTRANSFORMPATHNVPROC(glfwGetProcAddress("glTransformPathNV"));
02366     glUniform1d = PFNGLUNIFORM1DPROC(glfwGetProcAddress("glUniform1d"));
02367     glUniform1dv = PFNGLUNIFORM1DVPROC(glfwGetProcAddress("glUniform1dv"));
02368     glUniform1f = PFNGLUNIFORM1FPROC(glfwGetProcAddress("glUniform1f"));
02369     glUniform1fv = PFNGLUNIFORM1FVPROC(glfwGetProcAddress("glUniform1fv"));
02370     glUniform1i = PFNGLUNIFORM1IPROC(glfwGetProcAddress("glUniform1i"));
02371     glUniform1i64ARB = PFNGLUNIFORM1I64ARBPROC(glfwGetProcAddress("glUniform1i64ARB"));
02372     glUniform1i64NV = PFNGLUNIFORM1I64NVPROC(glfwGetProcAddress("glUniform1i64NV"));
02373     glUniform1i64vARB = PFNGLUNIFORM1I64VARBPROC(glfwGetProcAddress("glUniform1i64vARB"));
02374     glUniform1i64vNV = PFNGLUNIFORM1I64VNVPROC(glfwGetProcAddress("glUniform1i64vNV"));
02375     glUniform1liv = PFNGLUNIFORM1IVPROC(glfwGetProcAddress("glUniform1liv"));
02376     glUniform1lui = PFNGLUNIFORM1UIPROC(glfwGetProcAddress("glUniform1lui"));
02377     glUniform1lui64ARB = PFNGLUNIFORM1UI64ARBPROC(glfwGetProcAddress("glUniform1lui64ARB"));
02378     glUniform1lui64NV = PFNGLUNIFORM1UI64NVPROC(glfwGetProcAddress("glUniform1lui64NV"));
02379     glUniform2d = PFNGLUNIFORM2DPROC(glfwGetProcAddress("glUniform2d"));
02380     glUniform2dv = PFNGLUNIFORM2DVPROC(glfwGetProcAddress("glUniform2dv"));
02381     glUniform2f = PFNGLUNIFORM2FPROC(glfwGetProcAddress("glUniform2f"));
02382     glUniform2fv = PFNGLUNIFORM2FVPROC(glfwGetProcAddress("glUniform2fv"));
02383     glUniform2i = PFNGLUNIFORM2IPROC(glfwGetProcAddress("glUniform2i"));
02384     glUniform2i64ARB = PFNGLUNIFORM2I64ARBPROC(glfwGetProcAddress("glUniform2i64ARB"));
02385     glUniform2i64NV = PFNGLUNIFORM2I64NVPROC(glfwGetProcAddress("glUniform2i64NV"));
02386     glUniform2i64vARB = PFNGLUNIFORM2I64VARBPROC(glfwGetProcAddress("glUniform2i64vARB"));
02387     glUniform2iv = PFNGLUNIFORM2IVPROC(glfwGetProcAddress("glUniform2iv"));
02388     glUniform2ui = PFNGLUNIFORM2UIPROC(glfwGetProcAddress("glUniform2ui"));
02389     glUniform2ui64ARB = PFNGLUNIFORM2UI64ARBPROC(glfwGetProcAddress("glUniform2ui64ARB"));
02390     glUniform2ui64NV = PFNGLUNIFORM2UI64NVPROC(glfwGetProcAddress("glUniform2ui64NV"));
02391     glUniform2ui64vARB = PFNGLUNIFORM2UI64VARBPROC(glfwGetProcAddress("glUniform2ui64vARB"));
02392     glUniform2ui64vNV = PFNGLUNIFORM2UI64VNVPROC(glfwGetProcAddress("glUniform2ui64vNV"));
02393     glUniform2uiv = PFNGLUNIFORM2UIVPROC(glfwGetProcAddress("glUniform2uiv"));
02394     glUniform3d = PFNGLUNIFORM3DPROC(glfwGetProcAddress("glUniform3d"));
02395     glUniform3dv = PFNGLUNIFORM3DVPROC(glfwGetProcAddress("glUniform3dv"));
02396     glUniform3f = PFNGLUNIFORM3FPROC(glfwGetProcAddress("glUniform3f"));
02397     glUniform3fv = PFNGLUNIFORM3FVPROC(glfwGetProcAddress("glUniform3fv"));
02398     glUniform3i = PFNGLUNIFORM3IPROC(glfwGetProcAddress("glUniform3i"));
02399     glUniform3i64ARB = PFNGLUNIFORM3I64ARBPROC(glfwGetProcAddress("glUniform3i64ARB"));
02400     glUniform3i64NV = PFNGLUNIFORM3I64NVPROC(glfwGetProcAddress("glUniform3i64NV"));
02401     glUniform3i64vARB = PFNGLUNIFORM3I64VARBPROC(glfwGetProcAddress("glUniform3i64vARB"));
02402     glUniform3i64vNV = PFNGLUNIFORM3I64VNVPROC(glfwGetProcAddress("glUniform3i64vNV"));
02403     glUniform3iv = PFNGLUNIFORM3IVPROC(glfwGetProcAddress("glUniform3iv"));
02404     glUniform3ui = PFNGLUNIFORM3UIPROC(glfwGetProcAddress("glUniform3ui"));
02405     glUniform3ui64ARB = PFNGLUNIFORM3UI64ARBPROC(glfwGetProcAddress("glUniform3ui64ARB"));
02406     glUniform3ui64NV = PFNGLUNIFORM3UI64NVPROC(glfwGetProcAddress("glUniform3ui64NV"));
02407     glUniform3ui64vARB = PFNGLUNIFORM3UI64VARBPROC(glfwGetProcAddress("glUniform3ui64vARB"));
02408     glUniform3ui64vNV = PFNGLUNIFORM3UI64VNVPROC(glfwGetProcAddress("glUniform3ui64vNV"));
02409     glUniform3uiv = PFNGLUNIFORM3UIVPROC(glfwGetProcAddress("glUniform3uiv"));
02410     glUniform4d = PFNGLUNIFORM4DPROC(glfwGetProcAddress("glUniform4d"));
02411     glUniform4dv = PFNGLUNIFORM4DVPROC(glfwGetProcAddress("glUniform4dv"));
02412     glUniform4f = PFNGLUNIFORM4FPROC(glfwGetProcAddress("glUniform4f"));
02413     glUniform4fv = PFNGLUNIFORM4FVPROC(glfwGetProcAddress("glUniform4fv"));
02414     glUniform4i = PFNGLUNIFORM4IPROC(glfwGetProcAddress("glUniform4i"));
02415     glUniform4i64ARB = PFNGLUNIFORM4I64ARBPROC(glfwGetProcAddress("glUniform4i64ARB"));
02416     glUniform4i64NV = PFNGLUNIFORM4I64NVPROC(glfwGetProcAddress("glUniform4i64NV"));
02417     glUniform4i64vARB = PFNGLUNIFORM4I64VARBPROC(glfwGetProcAddress("glUniform4i64vARB"));
02418     glUniform4i64vNV = PFNGLUNIFORM4I64VNVPROC(glfwGetProcAddress("glUniform4i64vNV"));
02419     glUniform4iv = PFNGLUNIFORM4IVPROC(glfwGetProcAddress("glUniform4iv"));
02420     glUniform4ui = PFNGLUNIFORM4UIPROC(glfwGetProcAddress("glUniform4ui"));
02421     glUniform4ui64ARB = PFNGLUNIFORM4UI64ARBPROC(glfwGetProcAddress("glUniform4ui64ARB"));
02422     glUniform4ui64NV = PFNGLUNIFORM4UI64NVPROC(glfwGetProcAddress("glUniform4ui64NV"));
02423     glUniform4ui64vARB = PFNGLUNIFORM4UI64VARBPROC(glfwGetProcAddress("glUniform4ui64vARB"));
02424     glUniform4ui64vNV = PFNGLUNIFORM4UI64VNVPROC(glfwGetProcAddress("glUniform4ui64vNV"));
02425     glUniform4uiv = PFNGLUNIFORM4UIVPROC(glfwGetProcAddress("glUniform4uiv"));
02426     glUniformBlockBinding = PFNGLUNIFORMBLOCKBINDINGPROC(glfwGetProcAddress("glUniformBlockBinding"));
02427     glUniformHandleui64ARB =
02428     PFNGLUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glUniformHandleui64ARB"));
02429     glUniformHandleui64NV = PFNGLUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glUniformHandleui64NV"));
02430     glUniformHandleui64vARB =
02431     PFNGLUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glUniformHandleui64vARB"));
02432     glUniformHandleui64vNV =
02433     PFNGLUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glUniformHandleui64VNVPROC"));
02434     glUniformMatrix2x2dv = PFNGLUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glUniformMatrix2dv"));
02435     glUniformMatrix2x2fv = PFNGLUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glUniformMatrix2fv"));
02436     glUniformMatrix2x3dv = PFNGLUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glUniformMatrix2x3dv"));
02437     glUniformMatrix2x3fv = PFNGLUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glUniformMatrix2x3fv"));
02438     glUniformMatrix2x4dv = PFNGLUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glUniformMatrix2x4dv"));

```

```

02436 glUniformMatrix2x4fv = PFNGLUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glUniformMatrix2x4fv"));
02437 glUniformMatrix3dv = PFNGLUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glUniformMatrix3dv"));
02438 glUniformMatrix3fv = PFNGLUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glUniformMatrix3fv"));
02439 glUniformMatrix3x2dv = PFNGLUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glUniformMatrix3x2dv"));
02440 glUniformMatrix3x2fv = PFNGLUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glUniformMatrix3x2fv"));
02441 glUniformMatrix3x4dv = PFNGLUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glUniformMatrix3x4dv"));
02442 glUniformMatrix3x4fv = PFNGLUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glUniformMatrix3x4fv"));
02443 glUniformMatrix4dv = PFNGLUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glUniformMatrix4dv"));
02444 glUniformMatrix4fv = PFNGLUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glUniformMatrix4fv"));
02445 glUniformMatrix4x2dv = PFNGLUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glUniformMatrix4x2dv"));
02446 glUniformMatrix4x2fv = PFNGLUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glUniformMatrix4x2fv"));
02447 glUniformMatrix4x3dv = PFNGLUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glUniformMatrix4x3dv"));
02448 glUniformMatrix4x3fv = PFNGLUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glUniformMatrix4x3fv"));
02449 glUniformSubroutinesuiv =
PFNGLUNIFORMSUBROUTINESUIVPROC(glfwGetProcAddress("glUniformSubroutinesuiv"));
02450 glUniformui64NV = PFNGLUNIFORMUI64NVPROC(glfwGetProcAddress("glUniformui64NV"));
02451 glUniformui64vNV = PFNGLUNIFORMUI64VNPROC(glfwGetProcAddress("glUniformui64vNV"));
02452 glUnmapBuffer = PFNGLUNMAPBUFFERPROC(glfwGetProcAddress("glUnmapBuffer"));
02453 glUnmapNamedBuffer = PFNGLUNMAPNAMEDBUFFERPROC(glfwGetProcAddress("glUnmapNamedBuffer"));
02454 glUnmapNamedBufferEXT = PFNGLUNMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glUnmapNamedBufferEXT"));
02455 glUseProgram = PFNGLUSEPROGRAMPROC(glfwGetProcAddress("glUseProgram"));
02456 glUseProgramStages = PFNGLUSEPROGRAMSTAGESPROC(glfwGetProcAddress("glUseProgramStages"));
02457 glUseShaderProgramEXT = PFNGLUSES_SHADERPROGRAMEXTPROC(glfwGetProcAddress("glUseShaderProgramEXT"));
02458 glValidateProgram = PFNGLVALIDATEPROGRAMPROC(glfwGetProcAddress("glValidateProgram"));
02459 glValidateProgramPipeline =
PFNGLVALIDATEPROGRAMPIPELINEPROC(glfwGetProcAddress("glValidateProgramPipeline"));
02460 glVertexArrayAttribBinding =
PFNGLVERTEXARRAYATTRIBBINDINGPROC(glfwGetProcAddress("glVertexArrayAttribBinding"));
02461 glVertexArrayAttribFormat =
PFNGLVERTEXARRAYATTRIBFORMATPROC(glfwGetProcAddress("glVertexArrayAttribFormat"));
02462 glVertexArrayAttribIFormat =
PFNGLVERTEXARRAYATTRIBIFORMATPROC(glfwGetProcAddress("glVertexArrayAttribIFormat"));
02463 glVertexArrayAttribLFormat =
PFNGLVERTEXARRAYATTRIBLFORMATPROC(glfwGetProcAddress("glVertexArrayAttribLFormat"));
02464 glVertexArrayBindVertexBufferEXT =
PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC(glfwGetProcAddress("glVertexArrayBindVertexBufferEXT"));
02465 glVertexArrayBindingDivisor =
PFNGLVERTEXARRAYBINDINGDIVISORPROC(glfwGetProcAddress("glVertexArrayBindingDivisor"));
02466 glVertexArrayColorOffsetEXT =
PFNGLVERTEXARRAYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArrayColorOffsetEXT"));
02467 glVertexArrayEdgeFlagOffsetEXT =
PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayEdgeFlagOffsetEXT"));
02468 glVertexArrayElementBuffer =
PFNGLVERTEXARRAYELEMENTBUFFERPROC(glfwGetProcAddress("glVertexArrayElementBuffer"));
02469 glVertexArrayFogCoordOffsetEXT =
PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayFogCoordOffsetEXT"));
02470 glVertexArrayIndexOffsetEXT =
PFNGLVERTEXARRAYINDEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayIndexOffsetEXT"));
02471 glVertexArrayMultiTexCoordOffsetEXT =
PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayMultiTexCoordOffsetEXT"));
02472 glVertexArrayNormalOffsetEXT =
PFNGLVERTEXARRAYNORMALOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayNormalOffsetEXT"));
02473 glVertexArraySecondaryColorOffsetEXT =
PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArraySecondaryColorOffsetEXT"));
02474 glVertexArrayTexCoordOffsetEXT =
PFNGLVERTEXARRAYTEXCOORDOFSETEXTPROC(glfwGetProcAddress("glVertexArrayTexCoordOffsetEXT"));
02475 glVertexArrayVertexAttribBindingEXT =
PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribBindingEXT"));
02476 glVertexArrayVertexAttribDivisorEXT =
PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribDivisorEXT"));
02477 glVertexArrayVertexAttribFormatEXT =
PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribFormatEXT"));
02478 glVertexArrayVertexAttribIFormatEXT =
PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIFormatEXT"));
02479 glVertexArrayVertexAttribIOffsetEXT =
PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIOffsetEXT"));
02480 glVertexArrayVertexAttribLFormatEXT =
PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLFormatEXT"));
02481 glVertexArrayVertexAttribLOffsetEXT =
PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLOffsetEXT"));
02482 glVertexArrayVertexAttribOffsetEXT =
PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribOffsetEXT"));
02483 glVertexArrayVertexBindingDivisorEXT =
PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexBindingDivisorEXT"));
02484 glVertexArrayVertexBuffer =
PFNGLVERTEXARRAYVERTEXBUFFERPROC(glfwGetProcAddress("glVertexArrayVertexBuffer"));
02485 glVertexArrayVertexBuffers =
PFNGLVERTEXARRAYVERTEXBUFFERSPROC(glfwGetProcAddress("glVertexArrayVertexBuffers"));
02486 glVertexArrayVertexOffsetEXT =
PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexOffsetEXT"));
02487 glVertexAttribId = PFNGLVERTEXATTRIB1DPROC(glfwGetProcAddress("glVertexAttribId"));
02488 glVertexAttribLdv = PFNGLVERTEXATTRIB1DPROC(glfwGetProcAddress("glVertexAttribLdv"));
02489 glVertexAttribI1f = PFNGLVERTEXATTRIB1FPROC(glfwGetProcAddress("glVertexAttribI1f"));
02490 glVertexAttribI1fv = PFNGLVERTEXATTRIB1FVPROC(glfwGetProcAddress("glVertexAttribI1fv"));
02491 glVertexAttribI1s = PFNGLVERTEXATTRIB1SPROC(glfwGetProcAddress("glVertexAttribI1s"));
02492 glVertexAttribI1sv = PFNGLVERTEXATTRIB1SVPROC(glfwGetProcAddress("glVertexAttribI1sv"));
02493 glVertexAttribI2d = PFNGLVERTEXATTRIB2DPROC(glfwGetProcAddress("glVertexAttribI2d"));

```

```

02494     glVertexAttrib2dv = PFNGLVERTEXATTRIB2DVPROC(glfwGetProcAddress("glVertexAttrib2dv"));
02495     glVertexAttrib2f = PFNGLVERTEXATTRIB2FPROC(glfwGetProcAddress("glVertexAttrib2f"));
02496     glVertexAttrib2fv = PFNGLVERTEXATTRIB2FVPROC(glfwGetProcAddress("glVertexAttrib2fv"));
02497     glVertexAttrib2s = PFNGLVERTEXATTRIB2SPROC(glfwGetProcAddress("glVertexAttrib2s"));
02498     glVertexAttrib2sv = PFNGLVERTEXATTRIB2SVPROC(glfwGetProcAddress("glVertexAttrib2sv"));
02499     glVertexAttrib3d = PFNGLVERTEXATTRIB3DPROC(glfwGetProcAddress("glVertexAttrib3d"));
02500     glVertexAttrib3dv = PFNGLVERTEXATTRIB3DVPROC(glfwGetProcAddress("glVertexAttrib3dv"));
02501     glVertexAttrib3f = PFNGLVERTEXATTRIB3FPROC(glfwGetProcAddress("glVertexAttrib3f"));
02502     glVertexAttrib3fv = PFNGLVERTEXATTRIB3FVPROC(glfwGetProcAddress("glVertexAttrib3fv"));
02503     glVertexAttrib3s = PFNGLVERTEXATTRIB3SPROC(glfwGetProcAddress("glVertexAttrib3s"));
02504     glVertexAttrib3sv = PFNGLVERTEXATTRIB3SVPROC(glfwGetProcAddress("glVertexAttrib3sv"));
02505     glVertexAttrib4Nbv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4Nbv"));
02506     glVertexAttrib4Niv = PFNGLVERTEXATTRIB4NIVPROC(glfwGetProcAddress("glVertexAttrib4Niv"));
02507     glVertexAttrib4Nsv = PFNGLVERTEXATTRIB4NSVPROC(glfwGetProcAddress("glVertexAttrib4Nsv"));
02508     glVertexAttrib4Nub = PFNGLVERTEXATTRIB4NUBPROC(glfwGetProcAddress("glVertexAttrib4Nub"));
02509     glVertexAttrib4Nubv = PFNGLVERTEXATTRIB4NUBVPROC(glfwGetProcAddress("glVertexAttrib4Nubv"));
02510     glVertexAttrib4Nuiv = PFNGLVERTEXATTRIB4NUIVPROC(glfwGetProcAddress("glVertexAttrib4Nuiv"));
02511     glVertexAttrib4Nusv = PFNGLVERTEXATTRIB4NUSVPROC(glfwGetProcAddress("glVertexAttrib4Nusv"));
02512     glVertexAttrib4bv = PFNGLVERTEXATTRIB4BVPROC(glfwGetProcAddress("glVertexAttrib4bv"));
02513     glVertexAttrib4d = PFNGLVERTEXATTRIB4DPROC(glfwGetProcAddress("glVertexAttrib4d"));
02514     glVertexAttrib4dv = PFNGLVERTEXATTRIB4DVPROC(glfwGetProcAddress("glVertexAttrib4dv"));
02515     glVertexAttrib4f = PFNGLVERTEXATTRIB4FPROC(glfwGetProcAddress("glVertexAttrib4f"));
02516     glVertexAttrib4fv = PFNGLVERTEXATTRIB4FVPROC(glfwGetProcAddress("glVertexAttrib4fv"));
02517     glVertexAttrib4iv = PFNGLVERTEXATTRIB4IVPROC(glfwGetProcAddress("glVertexAttrib4iv"));
02518     glVertexAttrib4s = PFNGLVERTEXATTRIB4SPROC(glfwGetProcAddress("glVertexAttrib4s"));
02519     glVertexAttrib4sv = PFNGLVERTEXATTRIB4SVPROC(glfwGetProcAddress("glVertexAttrib4sv"));
02520     glVertexAttrib4ubv = PFNGLVERTEXATTRIB4UBVPROC(glfwGetProcAddress("glVertexAttrib4ubv"));
02521     glVertexAttrib4uiv = PFNGLVERTEXATTRIB4UIVPROC(glfwGetProcAddress("glVertexAttrib4uiv"));
02522     glVertexAttrib4usv = PFNGLVERTEXATTRIB4USVPROC(glfwGetProcAddress("glVertexAttrib4usv"));
02523     glVertexAttribBinding = PFNGLVERTEXATTRIBBINDINGPROC(glfwGetProcAddress("glVertexAttribBinding"));
02524     glVertexAttribDivisor = PFNGLVERTEXATTRIBDIVISORPROC(glfwGetProcAddress("glVertexAttribDivisor"));
02525     glVertexAttribDivisorARB =
PFNGLVERTEXATTRIBDIVISORARBPROC(glfwGetProcAddress("glVertexAttribDivisorARB"));
02526     glVertexAttribFormat = PFNGLVERTEXATTRIBFORMATPROC(glfwGetProcAddress("glVertexAttribFormat"));
02527     glVertexAttribFormatNV =
PFNGLVERTEXATTRIBFORMATNVPROC(glfwGetProcAddress("glVertexAttribFormatNV"));
02528     glVertexAttribI1i = PFNGLVERTEXATTRIBI1IPROC(glfwGetProcAddress("glVertexAttribI1i"));
02529     glVertexAttribI1iv = PFNGLVERTEXATTRIBI1IVPROC(glfwGetProcAddress("glVertexAttribI1iv"));
02530     glVertexAttribI1ui = PFNGLVERTEXATTRIBI1UIPROC(glfwGetProcAddress("glVertexAttribI1ui"));
02531     glVertexAttribI1uiv = PFNGLVERTEXATTRIBI1UIVPROC(glfwGetProcAddress("glVertexAttribI1uiv"));
02532     glVertexAttribI2i = PFNGLVERTEXATTRIBI2IPROC(glfwGetProcAddress("glVertexAttribI2i"));
02533     glVertexAttribI2iv = PFNGLVERTEXATTRIBI2IVPROC(glfwGetProcAddress("glVertexAttribI2iv"));
02534     glVertexAttribI2ui = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2ui"));
02535     glVertexAttribI2uiv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2uiv"));
02536     glVertexAttribI3i = PFNGLVERTEXATTRIBI3IPROC(glfwGetProcAddress("glVertexAttribI3i"));
02537     glVertexAttribI3iv = PFNGLVERTEXATTRIBI3IVPROC(glfwGetProcAddress("glVertexAttribI3iv"));
02538     glVertexAttribI3ui = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3ui"));
02539     glVertexAttribI3uiv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3uiv"));
02540     glVertexAttribI4bv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4bv"));
02541     glVertexAttribI4i = PFNGLVERTEXATTRIBI4IPROC(glfwGetProcAddress("glVertexAttribI4i"));
02542     glVertexAttribI4iv = PFNGLVERTEXATTRIBI4IVPROC(glfwGetProcAddress("glVertexAttribI4iv"));
02543     glVertexAttribI4sv = PFNGLVERTEXATTRIBI4SVPROC(glfwGetProcAddress("glVertexAttribI4sv"));
02544     glVertexAttribI4ubv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4ubv"));
02545     glVertexAttribI4ui = PFNGLVERTEXATTRIBI4UIPROC(glfwGetProcAddress("glVertexAttribI4ui"));
02546     glVertexAttribI4uiv = PFNGLVERTEXATTRIBI4UIVPROC(glfwGetProcAddress("glVertexAttribI4uiv"));
02547     glVertexAttribI4usv = PFNGLVERTEXATTRIBI4USVPROC(glfwGetProcAddress("glVertexAttribI4usv"));
02548     glVertexAttribIFormat = PFNGLVERTEXATTRIBIFORMATPROC(glfwGetProcAddress("glVertexAttribIFormat"));
02549     glVertexAttribIFormatNV =
PFNGLVERTEXATTRIBIFORMATNVPROC(glfwGetProcAddress("glVertexAttribIFormatNV"));
02550     glVertexAttribIPointer =
PFNGLVERTEXATTRIBIPOINTERPROC(glfwGetProcAddress("glVertexAttribIPointer"));
02551     glVertexAttribL1d = PFNGLVERTEXATTRIBL1DPROC(glfwGetProcAddress("glVertexAttribL1d"));
02552     glVertexAttribL1dv = PFNGLVERTEXATTRIBL1DVPROC(glfwGetProcAddress("glVertexAttribL1dv"));
02553     glVertexAttribL1i64NV = PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
02554     glVertexAttribL1i64NVNV =
PFNGLVERTEXATTRIBL1I64NVNPROC(glfwGetProcAddress("glVertexAttribL1i64NVNV"));
02555     glVertexAttribL1ui64ARB =
PFNGLVERTEXATTRIBL1UI64ARBPROC(glfwGetProcAddress("glVertexAttribL1ui64ARB"));
02556     glVertexAttribL1ui64NV =
PFNGLVERTEXATTRIBL1UI64NVPROC(glfwGetProcAddress("glVertexAttribL1ui64NV"));
02557     glVertexAttribL1ui64vARB =
PFNGLVERTEXATTRIBL1UI64VARBPROC(glfwGetProcAddress("glVertexAttribL1ui64vARB"));
02558     glVertexAttribL1ui64NVNV =
PFNGLVERTEXATTRIBL1UI64NVNPROC(glfwGetProcAddress("glVertexAttribL1ui64NVNV"));
02559     glVertexAttribL2d = PFNGLVERTEXATTRIBL2DPROC(glfwGetProcAddress("glVertexAttribL2d"));
02560     glVertexAttribL2dv = PFNGLVERTEXATTRIBL2DVPROC(glfwGetProcAddress("glVertexAttribL2dv"));
02561     glVertexAttribL2i64NV = PFNGLVERTEXATTRIBL2I64NVPROC(glfwGetProcAddress("glVertexAttribL2i64NV"));
02562     glVertexAttribL2i64NVNV =
PFNGLVERTEXATTRIBL2I64NVNPROC(glfwGetProcAddress("glVertexAttribL2i64NVNV"));
02563     glVertexAttribL2ui64NV =
PFNGLVERTEXATTRIBL2UI64NVPROC(glfwGetProcAddress("glVertexAttribL2ui64NV"));
02564     glVertexAttribL2ui64NVNV =
PFNGLVERTEXATTRIBL2UI64NVNPROC(glfwGetProcAddress("glVertexAttribL2ui64NVNV"));
02565     glVertexAttribL3d = PFNGLVERTEXATTRIBL3DPROC(glfwGetProcAddress("glVertexAttribL3d"));
02566     glVertexAttribL3dv = PFNGLVERTEXATTRIBL3DVPROC(glfwGetProcAddress("glVertexAttribL3dv"));
02567     glVertexAttribL3i64NV = PFNGLVERTEXATTRIBL3I64NVPROC(glfwGetProcAddress("glVertexAttribL3i64NV"));
02568     glVertexAttribL3i64NVNV =

```

```

PFNGLVERTEXATTRIBL3I64NVPROC(glfwGetProcAddress("glVertexAttribL3i64vNV"));
02569    glVertexAttribL3ui64NV =
02570    PFNGLVERTEXATTRIBL3UI64NVPROC(glfwGetProcAddress("glVertexAttribL3ui64NV"));
02571    glVertexAttribL3ui64NV =
02572    PFNGLVERTEXATTRIBL3UI64NVPROC(glfwGetProcAddress("glVertexAttribL3ui64vNV"));
02573    glVertexAttribL4d = PFNGLVERTEXATTRIBL4DPROC(glfwGetProcAddress("glVertexAttribL4d"));
02574    glVertexAttribL4d = PFNGLVERTEXATTRIBL4DPROC(glfwGetProcAddress("glVertexAttribL4d"));
02575    glVertexAttribL4i64NV = PFNGLVERTEXATTRIBL4I64NVPROC(glfwGetProcAddress("glVertexAttribL4i64NV"));
02576    glVertexAttribL4ui64NV = PFNGLVERTEXATTRIBL4UI64NVPROC(glfwGetProcAddress("glVertexAttribL4ui64NV"));
02577    glVertexAttribLFormat = PFNGLVERTEXATTRIBLFORMATPROC(glfwGetProcAddress("glVertexAttribLFormat"));
02578    glVertexAttribLFormatNV =
02579    PFNGLVERTEXATTRIBLFORMATNVPROC(glfwGetProcAddress("glVertexAttribLFormatNV"));
02580    glVertexAttribLPointer =
02581    PFNGLVERTEXATTRIBLPOINTERPROC(glfwGetProcAddress("glVertexAttribLPointer"));
02582    glVertexAttribPlui = PFNGLVERTEXATTRIBP1UIPROC(glfwGetProcAddress("glVertexAttribPlui"));
02583    glVertexAttribPluiv = PFNGLVERTEXATTRIBP1UIVPROC(glfwGetProcAddress("glVertexAttribPluiv"));
02584    glVertexAttribP2ui = PFNGLVERTEXATTRIBP2UIPROC(glfwGetProcAddress("glVertexAttribP2ui"));
02585    glVertexAttribP2uiv = PFNGLVERTEXATTRIBP2UIVPROC(glfwGetProcAddress("glVertexAttribP2uiv"));
02586    glVertexAttribP3ui = PFNGLVERTEXATTRIBP3UIPROC(glfwGetProcAddress("glVertexAttribP3ui"));
02587    glVertexAttribP3uiv = PFNGLVERTEXATTRIBP3UIVPROC(glfwGetProcAddress("glVertexAttribP3uiv"));
02588    glVertexAttribP4ui = PFNGLVERTEXATTRIBP4UIPROC(glfwGetProcAddress("glVertexAttribP4ui"));
02589    glVertexAttribP4uiv = PFNGLVERTEXATTRIBP4UIVPROC(glfwGetProcAddress("glVertexAttribP4uiv"));
02590    glVertexAttribPointer = PFNGLVERTEXATTRIBPOINTERPROC(glfwGetProcAddress("glVertexAttribPointer"));
02591    glVertexBindingDivisor =
02592    PFNGLVERTEXBINDINGDIVISORPROC(glfwGetProcAddress("glVertexBindingDivisor"));
02593    glVertexFormatNV = PFNGLVERTEXFORMATNVPROC(glfwGetProcAddress("glVertexFormatNV"));
02594    glVertexViewport = PFNGLVIEWPORTPROC(glfwGetProcAddress("glViewport"));
02595    glVertexViewportArrayv = PFNGLVIEWPORTARRAYVPROC(glfwGetProcAddress("glViewportArrayv"));
02596    glVertexViewportIndexeddf = PFNGLVIEWPORTINDEXEDFPROC(glfwGetProcAddress("glViewportIndexeddf"));
02597    glVertexViewportIndexeddfv = PFNGLVIEWPORTINDEXEDFVPROC(glfwGetProcAddress("glViewportIndexedfv"));
02598    glVertexViewportPositionWScaleNV =
02599    PFNGLVIEWPORTPOSITIONWSCALENVPROC(glfwGetProcAddress("glViewportPositionWScaleNV"));
02600    glVertexViewportSwizzleNV = PFNGLVIEWPORTSWIZZLENVPROC(glfwGetProcAddress("glViewportSwizzleNV"));
02601 #endif
02602
02603 // 使用している GPU のバッファアライメントを調べる
02604 glGetIntegerv(GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT, &ggBufferAlignment);
02605 }
02606
02607 //
02608 // OpenGL のエラーをチェックする
02609 //
02610 // OpenGL の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する
02611 //
02612 // msg エラー発生時に標準エラー出力に出力する文字列. nullptr なら何も出力しない
02613 //
02614 void gg::ggError(const std::string& name, unsigned int line)
02615 {
02616     const GLenum error{ glGetError() };
02617
02618     if (error != GL_NO_ERROR)
02619     {
02620         if (!name.empty())
02621         {
02622             std::cerr << name;
02623             if (line > 0) std::cerr << " (" << line << ")";
02624             std::cerr << ":" << std::endl;
02625         }
02626
02627         switch (error)
02628         {
02629             case GL_INVALID_ENUM:
02630                 std::cerr << "An unacceptable value is specified for an enumerated argument" << std::endl;
02631                 break;
02632             case GL_INVALID_VALUE:
02633                 std::cerr << "A numeric argument is out of range" << std::endl;
02634                 break;
02635             case GL_INVALID_OPERATION:
02636                 std::cerr << "The specified operation is not allowed in the current state" << std::endl;
02637                 break;
02638             case GL_OUT_OF_MEMORY:
02639                 std::cerr << "There is not enough memory left to execute the command" << std::endl;
02640                 break;
02641             case GL_INVALID_FRAMEBUFFER_OPERATION:
02642                 std::cerr << "The specified operation is not allowed current frame buffer" << std::endl;
02643                 break;
02644             default:
02645                 std::cerr << "An OpenGL error has occurred: " << std::hex << std::showbase << error << std::endl;

```

```
02646     break;
02647 }
02648 }
02649 }
02650
02651 //
02652 // FBO のエラーをチェックする
02653 //
02654 //   FBO の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する
02655 //
02656 //   msg エラー発生時に標準エラー出力に出来する文字列. nullptr なら何も出力しない
02657 //
02658 void gg::ggFBOError(const std::string& name, unsigned int line)
02659 {
02660     const GLenum status{ glCheckFramebufferStatus(GL_FRAMEBUFFER) };
02661
02662     if (status != GL_FRAMEBUFFER_COMPLETE)
02663     {
02664         if (!name.empty())
02665         {
02666             std::cerr << name;
02667             if (line > 0) std::cerr << " (" << line << ")";
02668             std::cerr << ":" ;
02669         }
02670
02671         switch (status)
02672         {
02673             case GL_FRAMEBUFFER_UNDEFINED:
02674                 std::cerr << "the default framebuffer does not exist" << std::endl;
02675                 break;
02676             case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
02677                 std::cerr << "Framebuffer incomplete, duplicate attachment" << std::endl;
02678                 break;
02679             case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
02680                 std::cerr << "Framebuffer incomplete, missing attachment" << std::endl;
02681                 break;
02682             case GL_FRAMEBUFFER_UNSUPPORTED:
02683                 std::cerr << "Unsupported framebuffer internal" << std::endl;
02684                 break;
02685             case GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE:
02686                 std::cerr << "The value of GL_RENDERBUFFER_SAMPLES is not"
02687                 " the same for all attached renderbuffers or,"
02688                 " if the attached images are a mix of renderbuffers and textures,"
02689                 " the value of GL_RENDERBUFFER_SAMPLES is not zero" << std::endl;
02690                 break;
02691 #if !defined(GL_GLES_PROTOTYPES)
02692             case GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS:
02693                 std::cerr << "Any framebuffer attachment is layered,"
02694                 " and any populated attachment is not layered,"
02695                 " or if all populated color attachments are not from textures"
02696                 " of the same target" << std::endl;
02697                 break;
02698             case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER:
02699                 std::cerr << "Framebuffer incomplete, missing draw buffer" << std::endl;
02700                 break;
02701             case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
02702                 std::cerr << "Framebuffer incomplete, missing read buffer" << std::endl;
02703                 break;
02704 #endif
02705             default:
02706                 std::cerr << "Programming error; will fail on all hardware: " << std::hex << std::showbase <<
02707                     status << std::endl;
02708                 break;
02709         }
02710     }
02711
02712 //
02713 // 変換行列：行列とベクトルの積 c ← a × b
02714 //
02715 void gg::GgMatrix::projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02716 {
02717     for (int i = 0; i < 4; ++i)
02718     {
02719         c[i] = a[0 + i] * b[0] + a[4 + i] * b[1] + a[8 + i] * b[2] + a[12 + i] * b[3];
02720     }
02721 }
02722
02723 //
02724 // 変換行列：行列と行列の積 c ← a × b
02725 //
02726 void gg::GgMatrix::multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02727 {
02728     for (int i = 0; i < 16; ++i)
02729     {
02730         int j = i & 3, k = i & ~3;
02731     }
02732 }
```

```

02732     c[i] = a[0 + j] * b[k + 0] + a[4 + j] * b[k + 1] + a[8 + j] * b[k + 2] + a[12 + j] * b[k + 3];
02733 }
02735
02736 // 
02737 // 変換行列：単位行列を設定する
02738 //
02739 gg::GgMatrix& gg::GgMatrix::loadIdentity()
02740 {
02741     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02742     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02743     data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02744     data()[ 0] = data()[ 5] = data()[10] = data()[15] = 1.0f;
02745
02746     return *this;
02747 }
02748
02749 //
02750 // 変換行列：平行移動変換行列を設定する
02751 //
02752 gg::GgMatrix& gg::GgMatrix::loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02753 {
02754     data()[12] = x;
02755     data()[13] = y;
02756     data()[14] = z;
02757     data()[ 0] = data()[ 5] = data()[10] = data()[15] = w;
02758     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02759     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02760     data()[11] = 0.0f;
02761
02762     return *this;
02763 }
02764
02765 //
02766 // 変換行列：拡大縮小変換行列を設定する
02767 //
02768 gg::GgMatrix& gg::GgMatrix::loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02769 {
02770     data()[ 0] = x;
02771     data()[ 5] = y;
02772     data()[10] = z;
02773     data()[15] = w;
02774     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02775     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02776     data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02777
02778     return *this;
02779 }
02780
02781 //
02782 // 変換行列：x 軸中心の回転変換行列を設定する
02783 //
02784 gg::GgMatrix& gg::GgMatrix::loadRotateX(GLfloat a)
02785 {
02786     const GLfloat c{ cosf(a) };
02787     const GLfloat s{ sinf(a) };
02788
02789     data()[ 0] = 1.0f; data()[ 1] = 0.0f; data()[ 2] = 0.0f; data()[ 3] = 0.0f;
02790     data()[ 4] = 0.0f; data()[ 5] = c;      data()[ 6] = s;      data()[ 7] = 0.0f;
02791     data()[ 8] = 0.0f; data()[ 9] = -s;    data()[10] = c;    data()[11] = 0.0f;
02792     data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02793
02794     return *this;
02795 }
02796
02797 //
02798 // 変換行列：y 軸中心の回転変換行列を設定する
02799 //
02800 gg::GgMatrix& gg::GgMatrix::loadRotateY(GLfloat a)
02801 {
02802     const GLfloat c{ cosf(a) };
02803     const GLfloat s{ sinf(a) };
02804
02805     data()[ 0] = c;      data()[ 1] = 0.0f; data()[ 2] = -s;    data()[ 3] = 0.0f;
02806     data()[ 4] = 0.0f; data()[ 5] = 1.0f; data()[ 6] = 0.0f; data()[ 7] = 0.0f;
02807     data()[ 8] = s;      data()[ 9] = 0.0f; data()[10] = c;    data()[11] = 0.0f;
02808     data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02809
02810     return *this;
02811 }
02812
02813 //
02814 // 変換行列：z 軸中心の回転変換行列を設定する
02815 //
02816 gg::GgMatrix& gg::GgMatrix::loadRotateZ(GLfloat a)
02817 {
02818     const GLfloat c{ cosf(a) };

```

```

02819 const GLfloat s{ sinf(a) };
02820
02821 data()[ 0] = c;    data()[ 1] = s;    data()[ 2] = 0.0f; data()[ 3] = 0.0f;
02822 data()[ 4] = -s;   data()[ 5] = c;    data()[ 6] = 0.0f; data()[ 7] = 0.0f;
02823 data()[ 8] = 0.0f; data()[ 9] = 0.0f; data()[10] = 1.0f; data()[11] = 0.0f;
02824 data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02825
02826 return *this;
02827 }
02828
02829 //
02830 // 変換行列：任意軸中心の回転変換行列を設定する
02831 //
02832 gg::GgMatrix& gg::GgMatrix::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
02833 {
02834 const GLfloat d{ sqrtf(x * x + y * y + z * z) };
02835
02836 if (d > 0.0f)
02837 {
02838 const GLfloat l{ x / d }, m{ y / d }, n{ z / d };
02839 const GLfloat l2{ l * l }, m2{ m * m }, n2{ n * n };
02840 const GLfloat lm{ l * m }, mn{ m * n }, nl{ n * l };
02841 const GLfloat c{ cosf(a) }, cl{ 1.0f - c };
02842 const GLfloat s{ sinf(a) };
02843
02844 data()[ 0] = (1.0f - l2) * c + l2;
02845 data()[ 1] = lm * cl + n * s;
02846 data()[ 2] = nl * cl - m * s;
02847 data()[ 3] = 0.0f;
02848
02849 data()[ 4] = lm * cl - n * s;
02850 data()[ 5] = (1.0f - m2) * c + m2;
02851 data()[ 6] = mn * cl + l * s;
02852 data()[ 7] = 0.0f;
02853
02854 data()[ 8] = nl * cl + m * s;
02855 data()[ 9] = mn * cl - l * s;
02856 data()[10] = (1.0f - n2) * c + n2;
02857 data()[11] = 0.0f;
02858
02859 data()[12] = 0.0f;
02860 data()[13] = 0.0f;
02861 data()[14] = 0.0f;
02862 data()[15] = 1.0f;
02863 }
02864
02865 return *this;
02866 }
02867
02868 //
02869 // 変換行列：転置行列を設定する
02870 //
02871 gg::GgMatrix& gg::GgMatrix::loadTranspose(const GLfloat* marray)
02872 {
02873 data()[ 0] = marray[ 0];
02874 data()[ 1] = marray[ 4];
02875 data()[ 2] = marray[ 8];
02876 data()[ 3] = marray[12];
02877 data()[ 4] = marray[ 1];
02878 data()[ 5] = marray[ 5];
02879 data()[ 6] = marray[ 9];
02880 data()[ 7] = marray[13];
02881 data()[ 8] = marray[ 2];
02882 data()[ 9] = marray[ 6];
02883 data()[10] = marray[10];
02884 data()[11] = marray[14];
02885 data()[12] = marray[ 3];
02886 data()[13] = marray[ 7];
02887 data()[14] = marray[11];
02888 data()[15] = marray[15];
02889
02890 return *this;
02891 }
02892
02893 //
02894 // 変換行列：逆行行列を設定する
02895 //
02896 gg::GgMatrix& gg::GgMatrix::loadInvert(const GLfloat* marray)
02897 {
02898 GLfloat lu[20], * plu[4];
02899
02900 // j 行の要素の値の絶対値の最大値を plu[j][4] に求める
02901 for (int j = 0; j < 4; ++j)
02902 {
02903     GLfloat max{ fabsf(*(plu[j] = lu + 5 * j) = *(marray++)) };
02904
02905     for (int i = 0; ++i < 4; )

```

```

02906    {
02907        GLfloat a{ fabsf(plu[j][i] = *(marray++)) };
02908        if (a > max) max = a;
02909    }
02910    if (max == 0.0f) return *this;
02911    plu[j][4] = 1.0f / max;
02912 }
02913
02914 // ピボットを考慮した LU 分解
02915 for (int j = 0; j < 4; ++j)
02916 {
02917     GLfloat max{ fabsf(plu[j][j] * plu[j][4]) };
02918     int i = j;
02919
02920     for (int k = j; ++k < 4;)
02921     {
02922         GLfloat a{ fabsf(plu[k][j] * plu[k][4]) };
02923         if (a > max)
02924         {
02925             max = a;
02926             i = k;
02927         }
02928     if (i > j)
02929     {
02930         GLfloat* t{ plu[j] };
02931         plu[j] = plu[i];
02932         plu[i] = t;
02933     }
02934     if (plu[j][j] == 0.0f) return *this;
02935     for (int k = j; ++k < 4;)
02936     {
02937         plu[k][j] /= plu[j][j];
02938         for (int i = j; ++i < 4;)
02939         {
02940             plu[k][i] -= plu[j][i] * plu[k][j];
02941         }
02942     }
02943 }
02944
02945 // LU 分解から逆行列を求める
02946 for (int k = 0; k < 4; ++k)
02947 {
02948     // array に単位行列を設定する
02949     for (int i = 0; i < 4; ++i)
02950     {
02951         data()[i * 4 + k] = (plu[i] == lu + k * 5) ? 1.0f : 0.0f;
02952     }
02953     // lu から逆行列を求める
02954     for (int i = 0; i < 4; ++i)
02955     {
02956         for (int j = i; ++j < 4;)
02957         {
02958             data()[j * 4 + k] -= data()[i * 4 + k] * plu[j][i];
02959         }
02960     }
02961     for (int i = 4; --i >= 0;)
02962     {
02963         for (int j = i; ++j < 4;)
02964         {
02965             data()[i * 4 + k] -= plu[i][j] * data()[j * 4 + k];
02966         }
02967         data()[i * 4 + k] /= plu[i][i];
02968     }
02969 }
02970 }
02971 return *this;
02972 }
02973 }
02974
02975 //
02976 // 変換行列：法線変換行列を設定する
02977 //
02978 gg::GgMatrix& gg::GgMatrix::loadNormal(const GLfloat* marray)
02979 {
02980     data()[ 0] = marray[ 5] * marray[10] - marray[ 6] * marray[ 9];
02981     data()[ 1] = marray[ 6] * marray[ 8] - marray[ 4] * marray[10];
02982     data()[ 2] = marray[ 4] * marray[ 9] - marray[ 5] * marray[ 8];
02983     data()[ 4] = marray[ 9] * marray[ 2] - marray[10] * marray[ 1];
02984     data()[ 5] = marray[10] * marray[ 0] - marray[ 8] * marray[ 2];
02985     data()[ 6] = marray[ 8] * marray[ 1] - marray[ 9] * marray[ 0];
02986     data()[ 8] = marray[ 1] * marray[ 6] - marray[ 2] * marray[ 5];
02987     data()[ 9] = marray[ 2] * marray[ 4] - marray[ 0] * marray[ 6];
02988     data()[10] = marray[ 0] * marray[ 5] - marray[ 1] * marray[ 4];
02989     data()[ 3] = data()[ 7] = data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02990     data()[15] = 1.0f;
02991
02992     return *this;

```

```

02993 }
02994
02995 // 
02996 // 変換行列：ビュー変換行列を設定する
02997 //
02998 gg::GgMatrix& gg::GgMatrix::loadLookat(
02999     GLfloat ex, GLfloat ey, GLfloat ez,
03000     GLfloat tx, GLfloat ty, GLfloat tz,
03001     GLfloat ux, GLfloat uy, GLfloat uz
03002 )
03003 {
03004     // z 軸 = e - t
03005     const GLfloat zx{ ex - tx };
03006     const GLfloat zy{ ey - ty };
03007     const GLfloat zz{ ez - tz };
03008
03009     // x 軸 = u × z 軸
03010     const GLfloat xx{ uy * zz - uz * zy };
03011     const GLfloat xy{ uz * zx - ux * zz };
03012     const GLfloat xz{ ux * zy - uy * zx };
03013
03014     // y 軸 = z 軸 × x 軸
03015     const GLfloat yx{ zy * xx - zz * xy };
03016     const GLfloat yy{ zz * xx - zx * xz };
03017     const GLfloat yz{ zx * xy - zy * xx };
03018
03019     // y 軸の長さをチェック
03020     GLfloat y{ yx * yx + yy * yy + yz * yz };
03021     if (y == 0.0f) return *this;
03022
03023     // x 軸の正規化
03024     const GLfloat x{ sqrtf(xx * xx + xy * xy + xz * xz) };
03025     data()[ 0] = xx / x;
03026     data()[ 4] = xy / x;
03027     data()[ 8] = xz / x;
03028
03029     // y 軸の正規化
03030     y = sqrtf(y);
03031     data()[ 1] = yx / y;
03032     data()[ 5] = yy / y;
03033     data()[ 9] = yz / y;
03034
03035     // z 軸の正規化
03036     const GLfloat z{ sqrtf(zx * zx + zy * zy + zz * zz) };
03037     data()[ 2] = zx / z;
03038     data()[ 6] = zy / z;
03039     data()[10] = zz / z;
03040
03041     // 平行移動
03042     data()[12] = -(ex * data()[ 0] + ey * data()[ 4] + ez * data()[ 8]);
03043     data()[13] = -(ex * data()[ 1] + ey * data()[ 5] + ez * data()[ 9]);
03044     data()[14] = -(ex * data()[ 2] + ey * data()[ 6] + ez * data()[10]);
03045
03046     // 残り
03047     data()[ 3] = data()[ 7] = data()[11] = 0.0f;
03048     data()[15] = 1.0f;
03049
03050     return *this;
03051 }
03052
03053 //
03054 // 変換行列：平行投影変換行列を設定する
03055 //
03056 gg::GgMatrix& gg::GgMatrix::loadOrthogonal(
03057     GLfloat left, GLfloat right,
03058     GLfloat bottom, GLfloat top,
03059     GLfloat zNear, GLfloat zFar
03060 )
03061 {
03062     const GLfloat dx{ right - left };
03063     const GLfloat dy{ top - bottom };
03064     const GLfloat dz{ zFar - zNear };
03065
03066     if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
03067     {
03068         data()[ 0] = 2.0f / dx;
03069         data()[ 5] = 2.0f / dy;
03070         data()[10] = -2.0f / dz;
03071         data()[12] = -(right + left) / dx;
03072         data()[13] = -(top + bottom) / dy;
03073         data()[14] = -(zFar + zNear) / dz;
03074         data()[15] = 1.0f;
03075         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03076         data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
03077         data()[11] = 0.0f;
03078     }
03079

```

```

03080     return *this;
03081 }
03082
03083 //
03084 // 変換行列：透視投影変換行列を設定する
03085 //
03086 gg::GgMatrix& gg::GgMatrix::loadFrustum(
03087     GLfloat left, GLfloat right,
03088     GLfloat bottom, GLfloat top,
03089     GLfloat zNear, GLfloat zFar
03090 )
03091 {
03092     const GLfloat dx{ right - left };
03093     const GLfloat dy{ top - bottom };
03094     const GLfloat dz{ zFar - zNear };
03095
03096     if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
03097     {
03098         data()[ 0] = 2.0f * zNear / dx;
03099         data()[ 5] = 2.0f * zNear / dy;
03100         data()[ 8] = (right + left) / dx;
03101         data()[ 9] = (top + bottom) / dy;
03102         data()[10] = -(zFar + zNear) / dz;
03103         data()[11] = -1.0f;
03104         data()[14] = -2.0f * zFar * zNear / dz;
03105         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03106         data()[ 6] = data()[ 7] = data()[12] = data()[13] =
03107         data()[15] = 0.0f;
03108     }
03109
03110     return *this;
03111 }
03112
03113 //
03114 // 変換行列：画角から透視投影変換行列を設定する
03115 //
03116 gg::GgMatrix& gg::GgMatrix::loadPerspective(
03117     GLfloat fovy, GLfloat aspect,
03118     GLfloat zNear, GLfloat zFar
03119 )
03120 {
03121     const GLfloat dz{ zFar - zNear };
03122
03123     if (dz != 0.0f)
03124     {
03125         data()[ 5] = 1.0f / tanf(fovy * 0.5f);
03126         data()[ 0] = data()[ 5] / aspect;
03127         data()[10] = -(zFar + zNear) / dz;
03128         data()[11] = -1.0f;
03129         data()[14] = -2.0f * zFar * zNear / dz;
03130         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03131         data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
03132         data()[12] = data()[13] = data()[15] = 0.0f;
03133     }
03134
03135     return *this;
03136 }
03137
03138 //
03139 // 四元数：GgQuaternion 型の四元数 p, q の積を r に求める
03140 //
03141 void gg::GgQuaternion::multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const
03142 {
03143     r[0] = p[1] * q[2] - p[2] * q[1] + p[0] * q[3] + p[3] * q[0];
03144     r[1] = p[2] * q[0] - p[0] * q[2] + p[1] * q[3] + p[3] * q[1];
03145     r[2] = p[0] * q[1] - p[1] * q[0] + p[2] * q[3] + p[3] * q[2];
03146     r[3] = p[3] * q[3] - p[0] * q[0] - p[1] * q[1] - p[2] * q[2];
03147 }
03148
03149 //
03150 // 四元数：GgQuaternion 型の四元数 q が表す変換行列を m に求める
03151 //
03152 void gg::GgQuaternion::toMatrix(GLfloat* m, const GLfloat* q) const
03153 {
03154     const GLfloat xx{ q[0] * q[0] * 2.0f };
03155     const GLfloat yy{ q[1] * q[1] * 2.0f };
03156     const GLfloat zz{ q[2] * q[2] * 2.0f };
03157     const GLfloat xy{ q[0] * q[1] * 2.0f };
03158     const GLfloat yz{ q[1] * q[2] * 2.0f };
03159     const GLfloat zx{ q[2] * q[0] * 2.0f };
03160     const GLfloat xw{ q[0] * q[3] * 2.0f };
03161     const GLfloat yw{ q[1] * q[3] * 2.0f };
03162     const GLfloat zw{ q[2] * q[3] * 2.0f };
03163
03164     m[ 0] = 1.0f - yy - zz;
03165     m[ 1] = xy + zw;
03166     m[ 2] = zx - yw;

```

```

03167     m[ 4] = xy - zw;
03168     m[ 5] = 1.0f - zz - xx;
03169     m[ 6] = yz + xw;
03170     m[ 8] = zx + yw;
03171     m[ 9] = yz - xw;
03172     m[10] = 1.0f - xx - yy;
03173     m[ 3] = m[ 7] = m[11] = m[12] = m[13] = m[14] = 0.0f;
03174     m[15] = 1.0f;
03175 }
03176
03177 /**
03178 // 四元数：回転変換行列 a が表す四元数を q に求める
03179 /**
03180 void gg::GgQuaternion::toQuaternion(GLfloat* q, const GLfloat* a) const
03181 {
03182     const GLfloat tr{ a[0] + a[5] + a[10] + a[15] };
03183
03184     if (tr > 0.0f)
03185     {
03186         q[3] = sqrtf(tr) * 0.5f;
03187         q[0] = (a[6] - a[9]) * 0.25f / q[3];
03188         q[1] = (a[8] - a[2]) * 0.25f / q[3];
03189         q[2] = (a[1] - a[4]) * 0.25f / q[3];
03190     }
03191 }
03192
03193 /**
03194 // 四元数：球面線形補間 p に q と r を t で補間した四元数を求める
03195 /**
03196 void gg::GgQuaternion::slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const
03197 {
03198     const GLfloat qr{ ggDot3(q, r) };
03199     const GLfloat ss{ 1.0f - qr * qr };
03200
03201     if (ss == 0.0f)
03202     {
03203         if (p != q)
03204         {
03205             p[0] = q[0];
03206             p[1] = q[1];
03207             p[2] = q[2];
03208             p[3] = q[3];
03209         }
03210     }
03211     else
03212     {
03213         const GLfloat sp{ sqrtf(ss) };
03214         const GLfloat ph{ acosf(qr) };
03215         const GLfloat pt{ ph * t };
03216         const GLfloat t1{ sinf(pt) / sp };
03217         const GLfloat t0{ sinf(ph - pt) / sp };
03218
03219         p[0] = q[0] * t0 + r[0] * t1;
03220         p[1] = q[1] * t0 + r[1] * t1;
03221         p[2] = q[2] * t0 + r[2] * t1;
03222         p[3] = q[3] * t0 + r[3] * t1;
03223     }
03224 }
03225
03226 /**
03227 // 四元数：(x, y, z) を軸とし角度 a 回転する四元数を求める
03228 /**
03229 gg::GgQuaternion& gg::GgQuaternion::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03230 {
03231     const GLfloat l(x * x + y * y + z * z);
03232
03233     if (l != 0.0)
03234     {
03235         GLfloat s{ sinf(a *= 0.5f) / sqrtf(l) };
03236
03237         data()[0] = x * s;
03238         data()[1] = y * s;
03239         data()[2] = z * s;
03240     }
03241     else
03242     {
03243         data()[0] = data()[1] = data()[2] = 0.0f;
03244     }
03245     data()[3] = cosf(a);
03246
03247     return *this;
03248 }
03249
03250 /**
03251 // x 軸を中心に角度 a 回転する四元数を格納する
03252 /**
03253 gg::GgQuaternion& gg::GgQuaternion::loadRotateX(GLfloat a)

```

```

03254 {
03255     const GLfloat t{ a * 0.5f };
03256
03257     data()[0] = sinf(t);
03258     data()[3] = cosf(t);
03259     data()[1] = data()[2] = 0.0f;
03260
03261     return *this;
03262 }
03263
03264 /**
03265 // y 軸を中心に角度 a 回転する四元数を格納する
03266 /**
03267 gg::GgQuaternion& gg::GgQuaternion::loadRotateY(GLfloat a)
03268 {
03269     const GLfloat t{ a * 0.5f };
03270
03271     data()[1] = sinf(t);
03272     data()[3] = cosf(t);
03273     data()[0] = data()[2] = 0.0f;
03274
03275     return *this;
03276 }
03277
03278 /**
03279 // z 軸を中心に角度 a 回転する四元数を格納する
03280 /**
03281 gg::GgQuaternion& gg::GgQuaternion::loadRotateZ(GLfloat a)
03282 {
03283     const GLfloat t{ a * 0.5f };
03284
03285     data()[2] = sinf(t);
03286     data()[3] = cosf(t);
03287     data()[0] = data()[1] = 0.0f;
03288
03289     return *this;
03290 }
03291
03292 /**
03293 // 四元数：オイラー角 (heading, pitch, roll) にもとづいて四元数を求める
03294 /**
03295 gg::GgQuaternion& gg::GgQuaternion::loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll)
03296 {
03297     GgQuaternion h, p, r;
03298
03299     h.loadRotateY(heading);
03300     p.loadRotateX(pitch);
03301     r.loadRotateZ(roll);
03302
03303     *this = h * p * r;
03304
03305     return *this;
03306 }
03307
03308 /**
03309 // 四元数：正規化して格納する
03310 /**
03311 gg::GgQuaternion& gg::GgQuaternion::loadNormalize(const GLfloat* a)
03312 {
03313     data()[0] = a[0];
03314     data()[1] = a[1];
03315     data()[2] = a[2];
03316     data()[3] = a[3];
03317
03318     ggNormalize4(data());
03319
03320     return *this;
03321 }
03322
03323 /**
03324 // 四元数：共役四元数を格納する
03325 /**
03326 gg::GgQuaternion& gg::GgQuaternion::loadConjugate(const GLfloat* a)
03327 {
03328     // w 要素を反転する
03329     data()[0] = a[0];
03330     data()[1] = a[1];
03331     data()[2] = a[2];
03332     data()[3] = -a[3];
03333
03334     return *this;
03335 }
03336
03337 /**
03338 // 四元数：逆元を格納する
03339 /**
03340 gg::GgQuaternion& gg::GgQuaternion::loadInvert(const GLfloat* a)

```

```
03341 {
03342     // ノルムの二乗を求める
03343     const GLfloat l(ggDot4(a, a));
03344
03345     if (l > 0.0f)
03346     {
03347         // 共役四元数を求める
03348         GgQuaternion r;
03349         r.loadConjugate(a);
03350
03351         // ノルムの二乗で割る
03352         data()[0] = r[0] / l;
03353         data()[1] = r[1] / l;
03354         data()[2] = r[2] / l;
03355         data()[3] = r[3] / l;
03356     }
03357
03358     return *this;
03359 }
03360
03361 //
03362 // 簡易トラックボール処理：リセット
03363 //
03364 void gg::GgTrackball::reset(const GgQuaternion& q)
03365 {
03366     // ドラッグ中ではない
03367     drag = false;
03368
03369     // 単位クオーターニオンに初期値を与える
03370     this->load(cq = q);
03371
03372     // 回転行列を初期化する
03373     static_cast<GgQuaternion*>(this)->getMatrix(rt);
03374 }
03375
03376 //
03377 // 簡易トラックボール処理：トラックボールする領域の設定
03378 //
03379 // Reshape コールバック (resize) の中に実行する
03380 // (w, h) ウィンドウサイズ
03381 //
03382 void gg::GgTrackball::region(GLfloat w, GLfloat h)
03383 {
03384     // マウスボインタ位置のウィンドウ内の相対的位置への換算用
03385     scale[0] = 2.0f / w;
03386     scale[1] = 2.0f / h;
03387 }
03388
03389 //
03390 // 簡易トラックボール処理：ドラッグ開始時の処理
03391 //
03392 // マウスボタンを押したときに実行する
03393 // (x, y) 現在のマウス位置
03394 //
03395 void gg::GgTrackball::begin(GLfloat x, GLfloat y)
03396 {
03397     // ドラッグ開始
03398     drag = true;
03399
03400     // ドラッグ開始点を記録する
03401     start[0] = x;
03402     start[1] = y;
03403 }
03404
03405 //
03406 // 簡易トラックボール処理：ドラッグ中の処理
03407 //
03408 // マウスのドラッグ中に実行する
03409 // (x, y) 現在のマウス位置
03410 //
03411 void gg::GgTrackball::motion(GLfloat x, GLfloat y)
03412 {
03413     // ドラッグ中でなければ何もしない
03414     if (!drag) return;
03415
03416     // マウスボインタの位置のドラッグ開始位置からの変位
03417     const GLfloat d[] { (x - start[0]) * scale[0], (y - start[1]) * scale[1] };
03418
03419     // マウスボインタの位置のドラッグ開始位置からの距離
03420     const GLfloat a { hypotf(d[0], d[1]) };
03421
03422     // マウスボインタの位置がドラッグ開始位置から移動していれば
03423     if (a > 0.0)
03424     {
03425         // 現在の回転の四元数に作った四元数を掛けて合成する
03426         this->load(ggRotateQuaternion(d[1], d[0], 0.0f, a * 3.1415926536f) * cq);
03427 }
```

```

03428     // 合成した四元数から回転の変換行列を求める
03429     static_cast<GgQuaternion*>(this)->getMatrix(rt);
03430 }
03431 }
03432
03433 //
03434 // 簡易トラックボール処理：回転角の修正
03435 //
03436 // 現在の回転角を修正する
03437 // q 修正分の回転角を表す四元数
03438 //
03439 void gg::GgTrackball::rotate(const GgQuaternion& q)
03440 {
03441     // ドラッグ中なら何もしない
03442     if (drag) return;
03443
03444     // 保存されている四元数に修正分の四元数を掛けて合成する
03445     this->load(q * cq);
03446
03447     // 合成した四元数から回転の変換行列を求める
03448     static_cast<GgQuaternion*>(this)->getMatrix(rt);
03449
03450     // 誤差を吸収するために正規化して保存する
03451     cq = normalize();
03452 }
03453
03454 //
03455 // 簡易トラックボール処理：停止時の処理
03456 //
03457 // マウスボタンを離したときに実行する
03458 // (x, y) 現在のマウス位置
03459 //
03460 void gg::GgTrackball::end(GLfloat x, GLfloat y)
03461 {
03462     // ドラッグ終了点における回転を求める
03463     motion(x, y);
03464
03465     // 誤差を吸収するために正規化して保存する
03466     cq = normalize();
03467
03468     // ドラッグ終了
03469     drag = false;
03470 }
03471
03472 //
03473 // 配列に格納された画像の内容を TGA ファイルに保存する
03474 //
03475 // name ファイル名
03476 // buffer 画像データ
03477 // width 画像の横の画素数
03478 // height 画像の縦の画素数
03479 // depth 画像の 1 画素のバイト数
03480 // 戻り値 保存に成功すれば true, 失敗すれば false
03481 //
03482 bool gg::ggSaveTga(
03483     const std::string& name,
03484     const void* buffer,
03485     unsigned int width,
03486     unsigned int height,
03487     unsigned int depth
03488 )
03489 {
03490     // ファイルを開く
03491     std::ofstream file{ Utf8ToTChar(name), std::ios::binary };
03492
03493     // ファイルが開けなかったら戻る
03494     if (file.fail()) return false;
03495
03496     // 画像のヘッダ
03497     const unsigned char type{ static_cast<unsigned char>(depth == 0 ? 0 : depth < 3 ? 3 : 2) };
03498     const unsigned char alpha{ static_cast<unsigned char>(depth == 2 || depth == 4 ? 8 : 0) };
03499     const unsigned char header[18]
03500     {
03501         0,           // ID length
03502         0,           // Color map type (none)
03503         type,        // Image Type (2:RGB, 3:Grayscale)
03504         0, 0,        // Offset into the color map table
03505         0, 0,        // Number of color map entries
03506         0,           // Number of a color map entry bits per pixel
03507         0, 0,        // Horizontal image position
03508         0, 0,        // Vertical image position
03509         static_cast<unsigned char>(width & 0xff),
03510         static_cast<unsigned char>(width >> 8),
03511         static_cast<unsigned char>(height & 0xff),
03512         static_cast<unsigned char>(height >> 8),
03513         static_cast<unsigned char>(depth * 8),
03514         alpha        // Image descriptor

```

```
03515    };
03516
03517    // ヘッダを書き込む
03518    file.write(reinterpret_cast<const char*>(header), sizeof header);
03519
03520    // ヘッダの書き込みに失敗したら戻る
03521    if (file.bad())
03522    {
03523        file.close();
03524        return false;
03525    }
03526
03527    // データを書き込む
03528    const unsigned int size{ width * height * depth };
03529    if (type == 2)
03530    {
03531        // フルカラー
03532        std::vector<char> temp(size);
03533        for (GLuint i = 0; i < size; i += depth)
03534        {
03535            temp[i + 2] = static_cast<const char*>(buffer)[i + 0];
03536            temp[i + 1] = static_cast<const char*>(buffer)[i + 1];
03537            temp[i + 0] = static_cast<const char*>(buffer)[i + 2];
03538            if (depth == 4) temp[i + 3] = static_cast<const char*>(buffer)[i + 3];
03539        }
03540        file.write(temp.data(), size);
03541    }
03542    else if (type == 3)
03543    {
03544        // グレースケール
03545        file.write(static_cast<const char*>(buffer), size);
03546    }
03547
03548    // フッタを書き込む
03549    constexpr char footer[] = "\0\0\0\0\0\0\0\0\0TRUEVISION-XFILE.";
03550    file.write(footer, sizeof footer);
03551
03552    // データの書き込みに失敗したら戻る
03553    if (file.bad())
03554    {
03555        file.close();
03556        return false;
03557    }
03558
03559    // ファイルを閉じる
03560    file.close();
03561
03562    // データの書き込みに成功した
03563    return true;
03564 }
03565
03566 //
03567 // カラーバッファの内容を TGA ファイルに保存する
03568 //
03569 //   name 保存するファイル名
03570 //   戻り値 保存に成功すれば true, 失敗すれば false
03571 //
03572 bool gg::ggSaveColor(const std::string& name)
03573 {
03574     // 現在のビューポートのサイズを得る
03575     GLint viewport[4];
03576     glGetIntegerv(GL_VIEWPORT, viewport);
03577
03578     // ビューポートのサイズ分のメモリを確保する
03579     std::vector<GLubyte> buffer(viewport[2] * viewport[3] * 3);
03580
03581     // 画面表示の完了を待つ
03582     glFinish();
03583
03584     // カラーバッファを読み込む
03585     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03586         GL_RGB, GL_UNSIGNED_BYTE, buffer.data());
03587
03588     // 読み込んだデータをファイルに書き込む
03589     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 3);
03590 }
03591
03592 //
03593 // デプスバッファの内容を TGA ファイルに保存する
03594 //
03595 //   name 保存するファイル名
03596 //   戻り値 保存に成功すれば true, 失敗すれば false
03597 //
03598 bool gg::ggSaveDepth(const std::string& name)
03599 {
03600     // 現在のビューポートのサイズを得る
03601     GLint viewport[4];
```

```

03602     glGetIntegerv(GL_VIEWPORT, viewport);
03603
03604     // ビューポートのサイズ分のメモリを確保する
03605     std::vector<GLubyte> buffer(viewport[2] * viewport[3]);
03606
03607     // 画面表示の完了を待つ
03608     glFinish();
03609
03610     // デブスマッファを読み込む
03611     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03612         GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, buffer.data());
03613
03614     // 読み込んだデータをファイルに書き込む
03615     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 1);
03616 }
03617
03618 //
03619 // TGA ファイル (8/16/24/32bit) を読み込む
03620 //
03621 // name 読み込むファイル名
03622 // pWidth 読み込んだファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)
03623 // pHeight 読み込んだファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)
03624 // pFormat 読み込んだファイルのフォーマットの格納先のポインタ (nullptr なら格納しない)
03625 // image 読み込んだ画像を格納する vector
03626 // 戻り値 読み込みに成功すれば true, 失敗すれば false
03627 //
03628 bool gg::ggReadImage(
03629     const std::string& name,
03630     std::vector<GLubyte>& image,
03631     GLsizei* pWidth,
03632     GLsizei* pHeight,
03633     GLenum* pFormat
03634 )
03635 {
03636     // ファイルを開く
03637     std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
03638
03639     // ファイルが開けなかったら戻る
03640     if (file.fail()) return false;
03641
03642     // ヘッダを読み込む
03643     unsigned char header[18];
03644     file.read(reinterpret_cast<char*>(header), sizeof(header));
03645
03646     // ヘッダの読み込みに失敗したら戻る
03647     if (file.bad())
03648     {
03649         file.close();
03650         return false;
03651     }
03652
03653     // 深度
03654     const int depth{ header[16] / 8 };
03655     switch (depth)
03656     {
03657         case 1:
03658             *pFormat = GL_RED;
03659             break;
03660         case 2:
03661             *pFormat = GL_RG;
03662             break;
03663         case 3:
03664             *pFormat = GL_RGB;
03665             break;
03666         case 4:
03667             *pFormat = GL_RGBA;
03668             break;
03669         default:
03670             // 取り扱えないフォーマットだったら戻る
03671             file.close();
03672             return false;
03673     }
03674
03675     // 画像の縦横の画素数
03676     *pWidth = header[13] << 8 | header[12];
03677     *pHeight = header[15] << 8 | header[14];
03678
03679     // データサイズ
03680     const int size{ *pWidth * *pHeight * depth };
03681     if (size < 2) return false;
03682
03683     // 読み込みに使うメモリを確保する
03684     image.resize(size);
03685
03686     // データを読み込む
03687     if (header[2] & 8)
03688     {

```

```
03689 // RLE
03690 int p{ 0 };
03691 char c;
03692 while (file.get(c))
03693 {
03694     if (c & 0x80)
03695     {
03696         // run-length packet
03697         const int count{ (c & 0x7f) + 1 };
03698         if (p + depth * count > size) break;
03699         char temp[4];
03700         file.read(temp, depth);
03701         for (int i = 0; i < count; ++i)
03702         {
03703             for (int j = 0; j < depth;) image[p++] = temp[j++];
03704         }
03705     }
03706     else
03707     {
03708         // raw packet
03709         const int count{ (c + 1) * depth };
03710         if (p + count > size) break;
03711         file.read(reinterpret_cast<char*>(image.data() + p), count);
03712         p += count;
03713     }
03714 }
03715 }
03716 else
03717 {
03718     // 非圧縮
03719     file.read(reinterpret_cast<char*>(image.data()), size);
03720 }
03721
03722 // 読み込みに失敗したら戻る
03723 if (file.bad())
03724 {
03725     file.close();
03726     return false;
03727 }
03728
03729 // ファイルを閉じる
03730 file.close();
03731
03732 // ファイルの読み込みに成功した
03733 return true;
03734 }
03735
03736 //
03737 // テクスチャを作成して画像を読み込む
03738 //
03739 // image 画像データ, nullptr ならメモリの確保だけを行う
03740 // width 画像の横の画素数
03741 // height 画像の縦の画素数
03742 // format 画像データのフォーマット
03743 // type 画像のデータ型
03744 // internal テクスチャの内部フォーマット
03745 // wrap テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE
03746 // swizzle true ならテクスチャの赤と青を入れ替える, デフォルトは true
03747 // 戻り値 テクスチャ名
03748 //
03749 GLuint gg::ggLoadTexture(
03750     const GLvoid* image,
03751     GLsizei width,
03752     GLsizei height,
03753     GLenum format,
03754     GLenum type,
03755     GLenum internal,
03756     GLenum wrap,
03757     bool swizzle
03758 )
03759 {
03760     // テクスチャオブジェクト
03761     const GLuint tex{ [] { glGenTextures(1, &tex); return tex; } () };
03762     glBindTexture(GL_TEXTURE_2D, tex);
03763
03764     // アルファチャンネルがついていれば 4 バイト境界に設定する
03765     glPixelStorei(GL_UNPACK_ALIGNMENT, format == GL_RGBA ? 4 : 1);
03766
03767     // テクスチャを割り当てる
03768     glTexImage2D(GL_TEXTURE_2D, 0, internal, width, height, 0, format, type, image);
03769
03770     // バイナリ (ミップマップなし), エッジでクランプ
03771     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
03772     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
03773     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap);
03774     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);
03775 }
```

```

03776 if (swizzle)
03777 {
03778     // テクスチャのサンプリング時に赤と青を入れ替える
03779     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, GL_BLUE);
03780     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, GL_RED);
03781 }
03782
03783 // テクスチャ名を返す
03784 return tex;
03785 }
03786
03787 //
03788 // テクスチャを作成して TGA フォーマットの画像ファイルを読み込む
03789 //
03790 // name TGA ファイル名
03791 // pWidth 読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)
03792 // pHeight 読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)
03793 // internal テクスチャの内部フォーマット、0 なら外部フォーマットに合わせる。
03794 // wrap テクスチャのラッピングモード、デフォルトは GL_CLAMP_TO_EDGE
03795 // 戻り値 テクスチャ名
03796 //
03797 GLuint gg::ggLoadImage(
03798     const std::string& name,
03799     GLsizei* pWidth,
03800     GLsizei* pHeight,
03801     GLenum internal,
03802     GLenum wrap
03803 )
03804 {
03805     // 画像データ
03806     std::vector<GLubyte> image;
03807
03808     // 画像サイズ
03809     GLsizei width, height;
03810
03811     // 画像フォーマット
03812     GLenum format;
03813
03814     // 画像を読み込む
03815     ggReadImage(name, image, &width, &height, &format);
03816
03817     // 画像が読み込めなかつたら戻る
03818     if (image.empty()) return 0;
03819
03820     // internal == 0 なら内部フォーマットを読み込んだファイルに合わせる
03821     if (internal == 0) internal = format;
03822
03823     // テクスチャに読み込む
03824     const GLuint tex(ggLoadTexture(image.data(), width, height,
03825         format, GL_UNSIGNED_BYTE, internal, wrap, true));
03826
03827     // 画像サイズを返す
03828     if (pWidth) *pWidth = width;
03829     if (pHeight) *pHeight = height;
03830
03831     // テクスチャ名を返す
03832     return tex;
03833 }
03834
03835 //
03836 // ゲレースケール画像 (8bit) から法線マップのデータを作成する
03837 //
03838 // width 高さマップのゲレースケール画像 hmap の横の画素数
03839 // height 高さマップのゲレースケール画像のデータ hmap の縦の画素数
03840 // stride データの間隔
03841 // hmap ゲレースケール画像のデータ
03842 // nz 法線の z 成分の割合
03843 // internal テクスチャの内部フォーマット
03844 // nmap 法線マップを格納する vector
03845 //
03846 void gg::ggCreateNormalMap(
03847     const GLubyte* hmap,
03848     GLsizei width,
03849     GLsizei height,
03850     GLenum format,
03851     GLfloat nz,
03852     GLenum internal,
03853     std::vector<GgVector>& nmap
03854 )
03855 {
03856     // メモリサイズ
03857     const GLsizei size{ width * height };
03858
03859     // 法線マップのメモリを確保する
03860     nmap.resize(size);
03861
03862     // 画素のバイト数

```

```

03863     GLint stride;
03864     switch (format)
03865     {
03866         case GL_RED:
03867             stride = 1;
03868             break;
03869         case GL_RGB:
03870             stride = 2;
03871             break;
03872         case GL_RGBA:
03873             stride = 3;
03874             break;
03875         case GL_RGBA:
03876             stride = 4;
03877             break;
03878     default:
03879         stride = 1;
03880         break;
03881     }
03882
03883 // 法線マップの作成
03884 for (GLsizei i = 0; i < size; ++i)
03885 {
03886     const int x{ i % width };
03887     const int y{ i - x };
03888     const int u0{ (y + (x - 1 + width) % width) * stride };
03889     const int u1{ (y + (x + 1) % width) * stride };
03890     const int v0{ ((y - width + size) % size + x) * stride };
03891     const int v1{ ((y + width) % size + x) * stride };
03892
03893     // 隣接する画素との値の差を法線の成分に用いる
03894     nmap[i][0] = static_cast<GLfloat>(hmap[u1] - hmap[u0]);
03895     nmap[i][1] = static_cast<GLfloat>(hmap[v1] - hmap[v0]);
03896     nmap[i][2] = nz;
03897     nmap[i][3] = hmap[i * stride];
03898
03899     // 法線ベクトルを正規化する
03900     ggNormalize3(nmap[i]);
03901 }
03902
03903 // 内部フォーマットが浮動小数点テクスチャでなければ [0,1] に正規化する
03904 if (
03905     internal != GL_RGB16F &&
03906     internal != GL_RGBA16F &&
03907     internal != GL_RGB32F &&
03908     internal != GL_RGBA32F
03909 )
03910 {
03911     for (GLsizei i = 0; i < size; ++i)
03912     {
03913         nmap[i][0] = nmap[i][0] * 0.5f + 0.5f;
03914         nmap[i][1] = nmap[i][1] * 0.5f + 0.5f;
03915         nmap[i][2] = nmap[i][2] * 0.5f + 0.5f;
03916         nmap[i][3] *= 0.0039215686f; // == 1/255
03917     }
03918 }
03919 }
03920
03921 //
03922 // TGA 画像ファイルの高さマップ読み込んで法線マップのテクスチャを作成する
03923 //
03924 //   name TGA ファイル名
03925 //   nz 作成した法線の z 成分の割合
03926 //   pWidth 読み込んだ画像の横の画素数の格納先のポインタ (nullptr なら格納しない)
03927 //   pHeight 読み込んだ画像の縦の画素数の格納先のポインタ (nullptr なら格納しない)
03928 //   internal テクスチャの内部フォーマット
03929 //   戻り値 テクスチャ名
03930 //
03931 GLuint gg::ggLoadHeight(
03932     const std::string& name,
03933     GLfloat nz,
03934     GLsizei* pWidth,
03935     GLsizei* pHeight,
03936     GLenum internal
03937 )
03938 {
03939     // 画像データ
03940     std::vector<GLubyte> hmap;
03941
03942     // 画像サイズ
03943     GLsizei width, height;
03944
03945     // 画像フォーマット
03946     GLenum format;
03947
03948     // 高さマップの画像を読み込む
03949     ggReadImage(name, hmap, &width, &height, &format);

```

```

03950
03951 // 画像が読み込めなかったら戻る
03952 if (hmap.empty()) return 0;
03953
03954 // 法線マップ
03955 std::vector<GgVector> nmap;
03956
03957 // 法線マップを作成する
03958 ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
03959
03960 // 画像サイズを返す
03961 if (pWidth) *pWidth = width;
03962 if (pHeight) *pHeight = height;
03963
03964 // テクスチャを作成して返す
03965 return ggLoadTexture(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal, GL_REPEAT);
03966 }
03967
03968 //
03969 // テクスチャの赤と青を交換する
03970 //
03971 void gg::GgTexture::swapRnB(bool swap) const
03972 {
03973     const auto swizzle
03974     {
03975         swap ?
03976             std::pair<GLint, GLint>{ GL_BLUE, GL_RED } :
03977             std::pair<GLint, GLint>{ GL_RED, GL_BLUE }
03978     };
03979
03980 glBindTexture(GL_TEXTURE_2D, texture);
03981 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, swizzle.first);
03982 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, swizzle.second);
03983 glBindTexture(GL_TEXTURE_2D, 0);
03984 }
03985
03986 //
03987 // TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成する
03988 //
03989 // name 読み込むファイル名
03990 // internal glTexImage2D() に指定するテクスチャの内部フォーマット。0なら外部フォーマットに合わせる
03991 // 戻り値 テクスチャの作成に成功すれば true, 失敗すれば false
03992 //
03993 void gg::GgColorTexture::load(
03994     const std::string& name,
03995     GLenum internal,
03996     GLenum wrap
03997 )
03998 {
03999     // 画像データ
04000     std::vector<GLubyte> image;
04001
04002     // 画像サイズ
04003     GLsizei width, height;
04004
04005     // 画像フォーマット
04006     GLenum format;
04007
04008     // 画像を読み込む
04009     ggReadImage(name, image, &width, &height, &format);
04010
04011     // internal == 0なら内部フォーマットを読み込んだファイルに合わせる
04012     if (internal == 0) internal = format;
04013
04014     // テクスチャを作成する
04015     texture = std::make_shared<GgTexture>(image.data(), width, height,
04016         format, GL_UNSIGNED_BYTE, internal, wrap, true);
04017 }
04018
04019 //
04020 // TGA フォーマットの画像ファイルから高さマップ読み込んで法線マップのテクスチャを作成する
04021 //
04022 // name 画像ファイル名
04023 // width テクスチャとして用いる画像データの横幅
04024 // height テクスチャとして用いる画像データの高さ
04025 // format テクスチャとして用いる画像データのフォーマット (GL_RED, GL_RG, GL_RGB, GL_RGBA)
04026 // nz 法線マップの z 成分の値
04027 // internal テクスチャの内部フォーマット
04028 //
04029 void gg::GgNormalTexture::load(
04030     const std::string& name,
04031     GLfloat nz,
04032     GLenum internal
04033 )
04034 {
04035     // 高さマップ
04036     std::vector<GLubyte> hmap;

```

```
04037
04038 // 画像サイズ
04039 GLsizei width, height;
04040
04041 // 画像フォーマット
04042 GLenum format;
04043
04044 // 高さマップの画像を読み込む
04045 ggReadImage(name, hmap, &width, &height, &format);
04046
04047 // 法線マップ
04048 std::vector<GgVector> nmap;
04049
04050 // 法線マップを作成する
04051 ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
04052 }
04053
04054
04055 // OBJ ファイルの読み込みに使うデータ型と関数
04056 //
04057 // OBJ ファイルの読み込みに使うデータ型と関数
04058 //
04059 namespace gg
04060 {
04061 // GLfloat 型の 2 要素のベクトル
04062 using vec2 = std::array<GLfloat, 2>;
04063
04064 // GLfloat 型の 3 要素のベクトル
04065 using vec3 = std::array<GLfloat, 3>;
04066
04067 // 三角形データ
04068 struct fidx
04069 {
04070     GLuint p[3];           // 頂点座標番号
04071     GLuint n[3];           // 頂点法線番号
04072     GLuint t[3];           // テクスチャ座標番号
04073     bool smooth;          // スムーズシェーディングの有無
04074 };
04075
04076 // ポリゴングループ
04077 struct fgrp
04078 {
04079     GLuint nextgroup;      // 次のポリゴングループの最初の三角形番号
04080     GLuint mtlno;          // このポリゴングループの材質番号
04081
04082     // コンストラクタ
04083     fgrp(GLuint nextgroup, GLuint mtlno) :
04084         nextgroup(nextgroup),
04085         mtlno(mtlno)
04086     {
04087     }
04088 };
04089
04090 // デフォルトの材質
04091 constexpr GgSimpleShader::Material defaultMaterial
04092 {
04093     { 0.1f, 0.1f, 0.1f, 1.0f },
04094     { 0.6f, 0.6f, 0.6f, 0.0f },
04095     { 0.3f, 0.3f, 0.3f, 0.0f },
04096     60.0f
04097 };
04098
04099 // デフォルトの材質名
04100 constexpr char defaultMaterialName[] = ".default.";
04101
04102 //
04103 // Alias OBJ 形式の MTL ファイルを読み込む
04104 //
04105 // mtlpath MTL ファイルのパス名
04106 // mtl 読み込んだ材質名をキー、材質番号を値にした map
04107 // material 材質データ
04108 //
04109 static bool ggLoadMtl(const std::string& mtlpath,
04110     std::map<std::string, GLuint>& mtl,
04111     std::vector<GgSimpleShader::Material>& material)
04112 {
04113     // MTL ファイルが無ければ戻る
04114     std::ifstream mtlfile{ Utf8ToTChar(mtlpath), std::ios::binary };
04115     if (!mtlfile)
04116     {
04117 #if defined(DEBUG)
04118         std::cerr << "Warning: Can't open MTL file: " << mtlpath << std::endl;
04119 #endif
04120         return false;
04121     }
04122
04123 // 一行読み込み用のバッファ
04124 std::string mtlline;
```

```

04125 // 材質名（ループの外に置く）
04126 std::string mtlname{ defaultMaterialName };
04128
04129 // 現在の材質番号を登録する
04130 mtl[mtlname] = static_cast<GLuint>(material.size());
04131
04132 // 現在の材質にデフォルトの材質を設定する
04133 material.emplace_back(defaultMaterial);
04134
04135 // 材質データを読み込む
04136 while (std::getline(mtlfile, mtlline))
04137 {
04138     // 空行は読み飛ばす
04139     if (mtlline == "") continue;
04140
04141     // 最後の文字が '\r' なら
04142     if (*(mtlline.end() - 1) == '\r')
04143     {
04144         // 最後の文字を削除する
04145         mtlline.erase(mtlline.end() - 1, mtlline.end());
04146
04147         // 空行になら読み飛ばす
04148         if (mtlline == "") continue;
04149     }
04150
04151     // 読み込んだ行を文字列ストリームにする
04152     std::istringstream mtlstr{ mtlline };
04153
04154     // オペレータ
04155     std::string mtlop;
04156
04157     // 文字列ストリームから材質パラメータの種類を取り出す
04158     mtlstr >> mtlop;
04159
04160     // '#' で始まる場合はコメントとして行末まで読み飛ばす
04161     if (mtlop[0] == '#') continue;
04162
04163     if (mtlop == "newmtl")
04164     {
04165         // 新規作成する材質名を取り出す
04166         mtlstr >> mtlname;
04167
04168         // 材質名が既に存在するかどうか調べる
04169         const auto m{ mtl.find(mtlname) };
04170         if (m == mtl.end())
04171         {
04172             // 存在しないので新規作成する材質の番号をその材質名に割り当てる
04173             mtl[mtlname] = static_cast<GLuint>(material.size());
04174
04175             // 新規作成する材質にデフォルトの材質を設定しておく
04176             material.emplace_back(defaultMaterial);
04177         }
04178
04179 #if defined(DEBUG)
04180     std::cerr << "newmtl: " << mtlname << std::endl;
04181 #endif
04182 }
04183 else if (mtlop == "Ka")
04184 {
04185     // 環境光の反射係数を登録する
04186     mtlstr
04187         >> material.back().ambient[0]
04188         >> material.back().ambient[1]
04189         >> material.back().ambient[2];
04190
04191 else if (mtlop == "Kd")
04192 {
04193     // 拡散反射係数を登録する
04194     mtlstr
04195         >> material.back().diffuse[0]
04196         >> material.back().diffuse[1]
04197         >> material.back().diffuse[2];
04198
04199 else if (mtlop == "Ks")
04200 {
04201     // 鏡面反射係数を登録する
04202     mtlstr
04203         >> material.back().specular[0]
04204         >> material.back().specular[1]
04205         >> material.back().specular[2];
04206
04207 else if (mtlop == "Ns")
04208 {
04209     // 輝き係数を登録する
04210     GLfloat shininess;
04211     mtlstr >> shininess;

```

```
04212     material.back().shininess = shininess * 0.1f;
04213 }
04214 else if (mtlpath == "d")
04215 {
04216     // 不透明度を登録する
04217     mtlstr >> material.back().ambient[3];
04218 }
04219 }
04220
04221 // MTL ファイルの読み込みに失敗したら戻る
04222 if (mtlfile.bad())
04223 {
04224 #if defined(DEBUG)
04225     std::cerr << "Warning: Can't read MTL file: " << mtlpath << std::endl;
04226 #endif
04227
04228 // MTL ファイルを閉じる
04229 mtlfile.close();
04230
04231 // MTL ファイルが読み込みに失敗したので戻る
04232 return false;
04233 }
04234
04235 // MTL ファイルを閉じる
04236 mtlfile.close();
04237
04238 // MTL ファイルの読み込みに成功した
04239 return true;
04240 }
04241
04242 //
04243 // Alias OBJ 形式のファイルを解析する
04244 //
04245 // name Alias OBJ 形式のファイル名
04246 // group 同じ材質を割り当てるポリゴングループ
04247 // mtl 読み込んだ材質名をキーにした map
04248 // pos 頂点の位置
04249 // norm 頂点の法線
04250 // tex 頂点のテクスチャ座標
04251 // face 三角形のデータ
04252 //
04253 static bool ggParseObj(
04254     const std::string& name,
04255     std::vector<figrp>& group,
04256     std::vector<GgSimpleShader::Material>& material,
04257     std::vector<vec3>& pos,
04258     std::vector<vec3>& norm,
04259     std::vector<vec2>& tex,
04260     std::vector<fidx>& face,
04261     bool normalize
04262 )
04263 {
04264     // ファイルパスからディレクトリ名を取り出す
04265     const std::string path{ name };
04266     const size_t base{ path.find_last_of("/") };
04267     const std::string dirname{ (base == std::string::npos) ? "" : path.substr(0, base + 1) };
04268
04269     // OBJ ファイルを読み込む
04270     std::ifstream file{ Utf8ToTChar(path) };
04271
04272     // 読み込みに失敗したら戻る
04273     if (file.fail())
04274     {
04275 #if defined(DEBUG)
04276         std::cerr << "Error: Can't open OBJ file: " << path << std::endl;
04277 #endif
04278     return false;
04279     }
04280
04281     // ポリゴングループの最初の三角形番号
04282     GLsizei startgroup(static_cast<GLsizei>(group.size()));
04283
04284     // スムーズシェーディングのスイッチ
04285     bool smooth{ false };
04286
04287     // 材質のテーブル
04288     std::map<std::string, GLuint> mtl;
04289
04290     // 現在の材質名 (ループの外で宣言する)
04291     std::string mtlname;
04292
04293     // 座標値の最小値・最大値
04294     vec3 bmin{ FLT_MAX }, bmax{ -FLT_MAX };
04295
04296     // 一行読み込み用のバッファ
04297     std::string line;
04298 }
```

```

04299 // データの読み込み
04300 while (std::getline(file, line))
04301 {
04302     // 空行は読み飛ばす
04303     if (line == "") continue;
04304
04305     // 最後の文字が '\r' なら
04306     if ((*line.end() - 1) == '\r')
04307     {
04308         // 最後の文字を削除する
04309         line.erase(line.end() - 1, line.end());
04310
04311         // 空行になったら読み飛ばす
04312         if (line == "") continue;
04313     }
04314
04315     // 一行を文字列ストリームに入れる
04316     std::istringstream str(line);
04317
04318     // 最初のトークンを命令 (op) とみなす
04319     std::string op;
04320     str >> op;
04321
04322     if (op[0] == '#') continue;
04323
04324     if (op == "v")
04325     {
04326         // 頂点位置
04327         vec3 v;
04328
04329         // 頂点位置はスペースで区切られている
04330         str >> v[0] >> v[1] >> v[2];
04331
04332         // 頂点位置を記録する
04333         pos.emplace_back(v);
04334
04335         // 頂点位置の最小値と最大値を求める (AABB)
04336         for (int i = 0; i < 3; ++i)
04337         {
04338             bmin[i] = std::min(bmin[i], v[i]);
04339             bmax[i] = std::max(bmax[i], v[i]);
04340         }
04341     }
04342     else if (op == "vt")
04343     {
04344         // テクスチャ座標
04345         vec2 t;
04346
04347         // 頂点位置はスペースで区切られている
04348         str >> t[0] >> t[1];
04349
04350         // テクスチャ座標を記録する
04351         tex.emplace_back(t);
04352
04353     else if (op == "vn")
04354     {
04355         // 頂点法線
04356         vec3 n;
04357
04358         // 頂点法線はスペースで区切られている
04359         str >> n[0] >> n[1] >> n[2];
04360
04361         // 頂点法線を記録する
04362         norm.emplace_back(n);
04363     }
04364     else if (op == "f")
04365     {
04366         // 三角形データ
04367         fidx f;
04368
04369         // スムースシェーディング
04370         f.smooth = smooth;
04371
04372         // 三頂点のそれぞれについて
04373         for (int i = 0; i < 3; ++i)
04374         {
04375             // 1項目取り出す
04376             std::string s;
04377             str >> s;
04378
04379             // 文字の位置
04380             auto c{ s.begin() };
04381
04382             // テクスチャ座標と法線の番号は未定義を表す 0 にしておく
04383             f.p[i] = f.t[i] = f.n[i] = 0;
04384
04385             // 項目の最初の要素は頂点座標番号

```

```

04386     while (c != s.end() && isdigit(*c)) f.p[i] = f.p[i] * 10 + *c++ - '0';
04387     if (c == s.end() || *c++ != '/') continue;
04388
04389     // 二つ目の項目はテクスチャ座標
04390     while (c != s.end() && isdigit(*c)) f.t[i] = f.t[i] * 10 + *c++ - '0';
04391     if (c == s.end() || *c++ != '/') continue;
04392
04393     // 三つ目の項目は法線番号
04394     while (c != s.end() && isdigit(*c)) f.n[i] = f.n[i] * 10 + *c++ - '0';
04395 }
04396
04397     // 三角形データを登録する
04398     face.emplace_back(f);
04399 }
04400 else if (op == "s")
04401 {
04402     // '1' だったらスムーズシェーディング有効
04403     std::string s;
04404     str >> s;
04405     smooth = s == "1";
04406 }
04407 else if (op == "usemtl")
04408 {
04409     // 次のポリゴングループの最初の三角形番号
04410     const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04411
04412     // ポリゴングループに三角形が存在すれば
04413     if (nextgroup > startgroup)
04414     {
04415         // ポリゴングループの三角形数と材質番号を記録する
04416         group.emplace_back(nextgroup, mtl[mtlname]);
04417
04418         // 次のポリゴングループの開始番号を保存しておく
04419         startgroup = nextgroup;
04420     }
04421
04422     // 次に usemtl が来るまで材質名を保持する
04423     str >> mtlname;
04424
04425     // 材質の存在チェック
04426     if (mtl.find(mtlname) == mtl.end())
04427     {
04428 #if defined(DEBUG)
04429         std::cerr << "Warning: Undefined material: " << mtlname << std::endl;
04430 #endif
04431
04432         // デフォルトの材質を割り当てておく
04433         mtlname = defaultMaterialName;
04434     }
04435 #if defined(DEBUG)
04436     else std::cerr << "usemtl: " << mtlname << std::endl;
04437 #endif
04438 }
04439 else if (op == "mtllib")
04440 {
04441     // MTL ファイルのパス名を作る
04442     str >> std::ws;
04443     std::string mtllibpath;
04444     std::getline(str, mtllibpath);
04445
04446     // MTL ファイルを読み込む
04447     ggLoadMtl(dirname + mtllibpath, mtl, material);
04448 }
04449 }
04450
04451     // OBJ ファイルの読み込みに失敗したら戻る
04452     if (file.bad())
04453     {
04454 #if defined(DEBUG)
04455         std::cerr << "Error: Can't read OBJ file: " << path << std::endl;
04456 #endif
04457         file.close();
04458         return false;
04459     }
04460
04461     // ファイルを閉じる
04462     file.close();
04463
04464     // 最後のポリゴングループの次の三角形番号
04465     const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04466     if (nextgroup > startgroup)
04467     {
04468         // 最後のポリゴングループの三角形数と材質を記録する
04469         group.emplace_back(nextgroup, static_cast<GLuint>(mtl[mtlname]));
04470     }
04471
04472     // スムーズシェーディングしない三角形の頂点を追加する

```

```

04473     for (auto& f : face)
04474     {
04475         if (!f.smooth)
04476         {
04477             // 三頂点のそれぞれについて
04478             for (int i = 0; i < 3; ++i)
04479             {
04480                 // 新しい頂点座標を生成する (std::array の要素は emplace_back できない)
04481                 pos.emplace_back(pos[f.p[i] - 1]);
04482                 f.p[i] = static_cast<GLuint>(pos.size());
04483
04484                 if (f.t[i] > 0)
04485                 {
04486                     // 新しいテクスチャ座標を生成する
04487                     tex.emplace_back(tex[f.t[i] - 1]);
04488                     f.t[i] = static_cast<GLuint>(tex.size());
04489                 }
04490
04491                 if (f.n[i] > 0)
04492                 {
04493                     // 新しい法線を生成する
04494                     norm.emplace_back(norm[f.n[i] - 1]);
04495                     f.n[i] = static_cast<GLuint>(norm.size());
04496                 }
04497             }
04498         }
04499     }
04500
04501     // 法線データがなければ算出しておく
04502     if (norm.empty())
04503     {
04504         // 法線データ数の初期値は頂点数と同じでスムーズシェーディングのために初期値は 0
04505         norm.resize(pos.size(), { 0.0f, 0.0f, 0.0f });
04506
04507         // 面の法線の算出と頂点法線の算出
04508         for (auto& f : face)
04509         {
04510             // 頂点座標番号
04511             const GLuint v0{ f.p[0] - 1 };
04512             const GLuint v1{ f.p[1] - 1 };
04513             const GLuint v2{ f.p[2] - 1 };
04514
04515             // v1 - v0, v2 - v0 を求める
04516             const GLfloat d1[] { pos[v1][0] - pos[v0][0], pos[v1][1] - pos[v0][1], pos[v1][2] - pos[v0][2] };
04517         };
04518         const GLfloat d2[] { pos[v2][0] - pos[v0][0], pos[v2][1] - pos[v0][1], pos[v2][2] - pos[v0][2] };
04519
04520         // 外積により面法線を求める
04521         vec3 n;
04522         ggCross(n.data(), d1, d2);
04523
04524         if (f.smooth)
04525         {
04526             // スムースシェーディングを行うときは
04527             for (int i = 0; i < 3; ++i)
04528             {
04529                 // 面法線を頂点法線に積算する
04530                 norm[v0][i] += n[i];
04531                 norm[v1][i] += n[i];
04532                 norm[v2][i] += n[i];
04533
04534                 // 面の各頂点の法線番号は頂点番号と同じにする
04535                 f.n[i] = f.p[i];
04536             }
04537         }
04538         else
04539         {
04540             // 面法線を最初の頂点に保存する
04541             norm[v0] = n;
04542             f.n[0] = f.p[0];
04543
04544             // 2 頂点追加
04545             for (int i = 1; i < 3; ++i)
04546             {
04547                 norm.emplace_back(n);
04548                 f.n[i] = static_cast<GLuint>(norm.size());
04549             }
04550         }
04551
04552         // 頂点の法線ベクトルを正規化する
04553         for (auto& n : norm) ggNormalize3(n.data());
04554
04555         // 図形の正規化
04556         if (normalize)
04557     }

```

```

04558 {
04559     // 図形の大きさ
04560     const GLfloat sx{ bmax[0] - bmin[0] };
04561     const GLfloat sy{ bmax[1] - bmin[1] };
04562     const GLfloat sz{ bmax[2] - bmin[2] };
04563
04564     // 図形のスケール
04565     GLfloat s{ sx };
04566     if (sy > s) s = sy;
04567     if (sz > s) s = sz;
04568     const GLfloat scale{ s != 0.0f ? 2.0f / s : 1.0f };
04569
04570     // 図形の中心位置
04571     const GLfloat cx{ (bmax[0] + bmin[0]) * 0.5f };
04572     const GLfloat cy{ (bmax[1] + bmin[1]) * 0.5f };
04573     const GLfloat cz{ (bmax[2] + bmin[2]) * 0.5f };
04574
04575     // 図形の大きさと位置を正規化する
04576     for (auto& p : pos)
04577     {
04578         p[0] = (p[0] - cx) * scale;
04579         p[1] = (p[1] - cy) * scale;
04580         p[2] = (p[2] - cz) * scale;
04581     }
04582 }
04583
04584 #if defined(DEBUG)
04585     std::cerr
04586     << "[" << name << "] \n(Parsed) Group: " << group.size() << ", Material: " << mtl.size()
04587     << ", Pos: " << pos.size() << ", Norm: " << norm.size() << ", Tex: " << tex.size()
04588     << ", Face: " << face.size() << "\n";
04589 #endif
04590
04591     // OBJ ファイルの読み込み成功
04592     return true;
04593 }
04594 }
04595
04596
04597 //
04598 // 三角形分割された Alias OBJ 形式のファイルと MTL ファイルを読み込む (Arrays 形式)
04599 //
04600 // name 読み込むOBJ ファイル名
04601 // group 読み込んだデータの各ポリゴングループの最初の三角形番号と三角形数
04602 // material 読み込んだデータのポリゴングループごとの材質
04603 // vert 読み込んだデータの頂点属性
04604 // normalize true ならサイズを正規化する
04605 // 戻り値 読み込みに成功したら true
04606 //
04607 bool gg::ggLoadSimpleObj(const std::string& name,
04608     std::vector<std::array<GLuint, 3>>& group,
04609     std::vector<GgSimpleShader::Material>& material,
04610     std::vector<GgVertex>& vert,
04611     bool normalize)
04612 {
04613     // 読み込み用の一時記憶領域
04614     std::vector<fggrp> tgroup;
04615     std::vector<vec3> tpos;
04616     std::vector<vec3> tnorm;
04617     std::vector<vec2> ttex;
04618     std::vector<fidx> tface;
04619
04620     // OBJ ファイルを解析する
04621     if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04622
04623     // 頂点属性データのメモリを確保する
04624     vert.reserve(vert.size() + tface.size() * 3);
04625
04626     // ポリゴン グループデータのメモリを確保する
04627     group.reserve(group.size() + tgroup.size());
04628     material.reserve(material.size() + tgroup.size());
04629
04630     // ポリゴン グループの最初の三角形番号
04631     GLuint startgroup{ 0 };
04632
04633     // ポリゴン グループデータの作成
04634     for (auto& g : tgroup)
04635     {
04636         // このポリゴン グループの最初の頂点番号と頂点数・材質番号
04637         std::array<GLuint, 3> v;
04638
04639         // このポリゴン グループの最初の頂点番号を保存する
04640         v[0] = static_cast<GLuint>(vert.size());
04641
04642         // 三角形ごとの頂点データの作成
04643         for (GLuint j = startgroup; j < g.nextgroup; ++j)
04644         {
04645             // 処理対象の三角形

```

```

04646     auto& f = tface[j];
04647
04648     // 三頂点のそれぞれについて
04649     for (int i = 0; i < 3; ++i)
04650     {
04651         // テクスチャ座標
04652         vec2 tex{ 0.0f, 0.0f };
04653         if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04654
04655         // 頂点法線
04656         vec3 norm{ 0.0f, 0.0f, 0.0f };
04657         if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04658
04659         // 頂点属性の追加
04660         vert.emplace_back(tpos[f.p[i] - 1].data(), norm.data());
04661     }
04662 }
04663
04664     // このポリゴン グループの頂点数を保存する
04665     v[1] = static_cast<GLuint>(vert.size()) - v[0];
04666     v[2] = g.mtlno;
04667
04668     // このポリゴン グループの最初の頂点番号と頂点数・材質番号を登録する
04669     group.emplace_back(v);
04670
04671     // 次のポリゴン グループの最初の三角形番号を求める
04672     startgroup = g.nextgroup;
04673 }
04674
04675 #if defined(DEBUG)
04676     std::cerr
04677         << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04678         << ", Vertex: " << vert.size() << "\n";
04679 #endif
04680
04681     // OBJ ファイルの読み込み成功
04682     return true;
04683 }
04684
04685 //
04686 // 三角形分割された Alias OBJ 形式のファイルと MTL ファイルを読み込む (Elements 形式)
04687 //
04688 //    name 読み込むOBJ ファイル名
04689 //    group 読み込んだデータの各ポリゴン グループの最初の三角形番号と三角形数
04690 //    material 読み込んだデータのポリゴン グループごとの材質
04691 //    vert 読み込んだデータの頂点属性
04692 //    face 読み込んだデータの三角形の頂点インデックス
04693 //    normalize true ならサイズを正規化する
04694 //    戻り値 読み込みに成功したら true
04695 //
04696 bool gg::ggLoadSimpleObj(const std::string& name,
04697     std::vector<std::array<GLuint, 3>>& group,
04698     std::vector<GgSimpleShader::Material>& material,
04699     std::vector<GgVertex>& vert,
04700     std::vector<GLuint>& face,
04701     bool normalize)
04702 {
04703     // 読み込み用の一時記憶領域
04704     std::vector<fgrp> tgroup;
04705     std::vector<vec3> tpos;
04706     std::vector<vec3> tnrm;
04707     std::vector<vec2> ttex;
04708     std::vector<fidx> tface;
04709
04710     // OBJ ファイルを解析する
04711     if (!ggParseObj(name, tgroup, material, tpos, tnrm, ttex, tface, normalize)) return false;
04712
04713     // 頂点属性データの最初の頂点番号
04714     const GLuint vertbase{ static_cast<GLuint>(vert.size()) };
04715
04716     // 頂点属性データのメモリを確保する
04717     vert.resize(vertbase + tpos.size());
04718
04719     // 三角形データのメモリを確保する
04720     face.reserve(face.size() + tface.size());
04721
04722     // ポリゴン グループデータのメモリを確保する
04723     group.reserve(group.size() + tgroup.size());
04724     material.reserve(material.size() + tgroup.size());
04725
04726     // ポリゴン グループの最初の三角形番号
04727     GLuint startgroup{ 0 };
04728
04729     // ポリゴン グループデータの作成
04730     for (auto& g : tgroup)
04731     {
04732         // このポリゴン グループの最初の頂点番号

```

```

04733     const GLuint first{ static_cast<GLuint>(face.size()) };
04734
04735     // 三角形ごとの頂点データの作成
04736     for (GLuint j = startgroup; j < g.nextgroup; ++j)
04737     {
04738         // 処理対象の三角形
04739         auto& f{ tface[j] };
04740
04741         // 三頂点のそれぞれについて
04742         for (int i = 0; i < 3; ++i)
04743         {
04744             // 追加する三角形データの頂点番号
04745             const GLuint q{ f.p[i] - 1 + vertbase };
04746
04747             // 三角形データの追加
04748             face.emplace_back(q);
04749
04750             // テクスチャ座標番号
04751             vec2 tex{ 0.0f, 0.0f };
04752             if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04753
04754             // 頂点法線番号
04755             vec3 norm{ 0.0f, 0.0f, 0.0f };
04756             if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04757
04758             // 頂点の格納
04759             vert[q] = GgVertex(tpos[f.p[i] - 1].data(), norm.data());
04760         }
04761     }
04762
04763     // このポリゴン グループの最初の三角形番号と三角形数・材質番号を登録する
04764     group.emplace_back(std::array<GLuint, 3>{ first, static_cast<GLuint>(face.size()) - first, g.mtlno });
04765
04766     // 次のポリゴン グループの最初の三角形番号を求める
04767     startgroup = g.nextgroup;
04768 }
04769
04770 #if defined(DEBUG)
04771     std::cerr
04772         << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04773         << ", Vertex: " << vert.size() << ", Face: " << face.size() << "\n";
04774 #endif
04775
04776 // OBJ ファイルの読み込み成功
04777 return true;
04778 }
04779
04780 //
04781 // シェーダオブジェクトのコンパイル結果を表示する
04782 //
04783 static GLboolean printShaderInfoLog(GLuint shader, const std::string& str)
04784 {
04785     // コンパイル結果を取得する
04786     GLint status;
04787     glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
04788 #if defined(DEBUG)
04789     if (status == GL_FALSE) std::cerr << "Compile Error in " << str << std::endl;
04790 #endif
04791
04792     // シェーダのコンパイル時のログの長さを取得する
04793     GLsizei bufSize;
04794     glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &bufSize);
04795
04796     if (bufSize > 1)
04797     {
04798         // シェーダのコンパイル時のログの内容を取得する
04799         std::vector<GLchar> infoLog(bufSize);
04800         GLsizei length;
04801         glGetShaderInfoLog(shader, bufSize, &length, infoLog.data());
04802 #if defined(DEBUG)
04803         std::cerr << infoLog.data();
04804 #endif
04805     }
04806
04807     // コンパイル結果を返す
04808     return static_cast<GLboolean>(status);
04809 }
04810
04811 //
04812 // プログラムオブジェクトのリンク結果を表示する
04813 //
04814 static GLboolean printProgramInfoLog(GLuint program)
04815 {
04816     // リンク結果を取得する
04817     GLint status;
04818     glGetProgramiv(program, GL_LINK_STATUS, &status);

```

```

04819 #if defined(DEBUG)
04820   if (status == GL_FALSE) std::cerr << "Link Error." << std::endl;
04821 #endif
04822
04823 // シェーダのリンク時のログの長さを取得する
04824 GLint bufSize;
04825 glGetProgramiv(program, GL_INFO_LOG_LENGTH, &bufSize);
04826
04827 // シェーダのリンク時のログの内容を取得する
04828 if (bufSize > 1)
04829 {
04830   std::vector<GLchar> infoLog(bufSize);
04831   GLsizei length;
04832   glGetProgramInfoLog(program, bufSize, &length, infoLog.data());
04833 #if defined(DEBUG)
04834   std::cerr << infoLog.data() << std::endl;
04835 #endif
04836 }
04837
04838 // リンク結果を返す
04839 return static_cast<GLboolean>(status);
04840 }
04841
04842 //
04843 // シェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する
04844 //
04845 // vsrc パーテックスシェーダのソースプログラムの文字列
04846 // fsrc フラグメントシェーダのソースプログラムの文字列 (nullptr なら不使用)
04847 // gsrc ジオメトリシェーダのソースプログラムの文字列 (nullptr なら不使用)
04848 // nvarying フィードバックする varying 変数の数 (0 なら不使用)
04849 // varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04850 // vtext パーテックスシェーダのコンパイル時のメッセージに追加する文字列
04851 // ftext フラグメントシェーダのコンパイル時のメッセージに追加する文字列
04852 // gtext ジオメトリシェーダのコンパイル時のメッセージに追加する文字列
04853 // 戻り値 プログラムオブジェクトのプログラム名 (作成できなければ 0)
04854 //
04855 GLuint gg::ggCreateShader(
04856   const std::string& vsrc,
04857   const std::string& fsrc,
04858   const std::string& gsrc,
04859   GLint nvarying,
04860   const char* const* varyings,
04861   const std::string& vtext,
04862   const std::string& ftext,
04863   const std::string& gtext)
04864 {
04865   // シェーダプログラムの作成
04866   const GLuint program{ glCreateProgram() };
04867
04868   if (program > 0)
04869   {
04870     bool status = true;
04871
04872     if (!vsrc.empty())
04873     {
04874       // パーテックスシェーダのシェーダオブジェクトを作成する
04875       const GLuint vertShader{ glCreateShader(GL_VERTEX_SHADER) };
04876       const GLchar* vsrcp{ vsrc.c_str() };
04877       glShaderSource(vertShader, 1, &vsrcp, nullptr);
04878       glCompileShader(vertShader);
04879
04880       // パーテックスシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04881       if (printShaderInfoLog(vertShader, vtext))
04882         glAttachShader(program, vertShader);
04883     else
04884       status = false;
04885     glDeleteShader(vertShader);
04886   }
04887
04888   if (!fsrc.empty())
04889   {
04890     // フラグメントシェーダのシェーダオブジェクトを作成する
04891     const GLuint fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
04892     const GLchar* fsrcc{ fsrc.c_str() };
04893     glShaderSource(fragShader, 1, &fsrcc, nullptr);
04894     glCompileShader(fragShader);
04895
04896     // フラグメントシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04897     if (printShaderInfoLog(fragShader, ftext))
04898       glAttachShader(program, fragShader);
04899     else
04900       status = false;
04901     glDeleteShader(fragShader);
04902   }
04903
04904   if (!gsrc.empty())
04905   {

```

```
04906 #if defined(GL_GLES_PROTOTYPES)
04907 # if defined(DEBUG)
04908     std::cerr << gtext << ": The geometry is not supported." << std::endl;
04909     status = false;
04910 # endif
04911 #else
04912     // ジオメトリシェーダのシェーダオブジェクトを作成する
04913     const GLuint geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
04914     const GLchar* gsrcp{ gsrc.c_str() };
04915     glShaderSource(geomShader, 1, &gsrcp, nullptr);
04916     glCompileShader(geomShader);
04917
04918     // ジオメトリシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04919     if (printShaderInfoLog(geomShader, gtext))
04920         glAttachShader(program, geomShader);
04921     else
04922         status = false;
04923     glDeleteShader(geomShader);
04924 #endif
04925 }
04926
04927 // 全てのシェーダオブジェクトのコンパイルに成功したら
04928 if (status)
04929 {
04930     // feedback に使う varying 変数を指定する
04931     if (nvarying > 0)
04932         glTransformFeedbackVaryings(program, nvarying, varyings, GL_SEPARATE_ATTRIBS);
04933
04934     // シェーダプログラムをリンクする
04935     glLinkProgram(program);
04936
04937     // リンクに成功したらプログラムオブジェクト名を返す
04938     if (printProgramInfoLog(program) != GL_FALSE) return program;
04939 }
04940 }
04941
04942 // プログラムオブジェクトが作成できなかった
04943 glDeleteProgram(program);
04944 return 0;
04945 }
04946
04947 //
04948 // シェーダのソースファイルを読み込んだ vector を返す
04949 //
04950 // name ソースファイル名
04951 // src 読み込んだソースファイルの文字列
04952 // 戻り値 読み込みの成功すれば true, 失敗したら false
04953 //
04954 static bool readShaderSource(const std::string& name, std::string& src)
04955 {
04956     // ファイル名が nullptr ならそのまま戻る
04957     if (name.empty()) return true;
04958
04959     // ソースファイルを開く
04960     std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
04961     if (file.fail())
04962     {
04963         // ファイルが開けなければエラーで戻る
04964 #if defined(DEBUG)
04965         std::cerr << "Error: Can't open source file: " << name << std::endl;
04966 #endif
04967         return false;
04968     }
04969
04970     // ファイル全体を文字列として読み込む
04971     std::istreambuf_iterator<char> it{ file };
04972     std::istreambuf_iterator<char> last;
04973     src = std::string(it, last);
04974
04975     // ファイルがうまく読み込めなければ戻る
04976     if (file.bad())
04977     {
04978 #if defined(DEBUG)
04979         std::cerr << "Error: Could not read source file: " << name << std::endl;
04980 #endif
04981         file.close();
04982         return false;
04983     }
04984
04985     // ファイルを閉じて戻る
04986     file.close();
04987     return true;
04988 }
04989
04990 //
04991 // シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する
04992 //
```

```

04993 // vert バーテックスシェーダのソースファイル名
04994 // frag フラグメントシェーダのソースファイル名 (nullptr なら不使用)
04995 // geom ジオメトリシェーダのソースファイル名 (nullptr なら不使用)
04996 // nvarying フィードバックする varying 変数の数 (0 なら不使用)
04997 // varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04998 // 戻り値 シェーダプログラムのプログラム名 (作成できなければ 0)
04999 //
05000 GLuint gg::ggLoadShader(
05001     const std::string& vert,
05002     const std::string& frag,
05003     const std::string& geom,
05004     GLint nvarying,
05005     const char* const* varyings
05006 )
05007 {
05008     // 読み込んだシェーダのソースプログラム
05009     std::string vsrc, fsrc, gsrc;
05010
05011     // 読み込んだ結果
05012     bool status;
05013
05014     // シェーダのソースファイルを読み込む
05015     status = readShaderSource(vert, vsrc);
05016     status = readShaderSource(frag, fsrc) && status;
05017     status = readShaderSource(geom, gsrc) && status;
05018
05019     // 全てのソースファイルが読み込めていなかったらエラー
05020     if (!status) return 0;
05021
05022     // プログラムオブジェクトを作成する
05023     return ggCreateShader(vsrc, fsrc, gsrc, nvarying, varyings, vert, frag, geom);
05024 }
05025
05026 #if !defined(__APPLE__) && !defined(GL_GLES_PROTOTYPES)
05027 //
05028 // コンピュートシェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する
05029 //
05030 //    csrc コンピュートシェーダのソースプログラムの文字列
05031 //    戻り値 プログラムオブジェクトのプログラム名 (作成できなければ 0)
05032 //
05033 GLuint gg::ggCreateComputeShader(const std::string& csrc, const std::string& ctext)
05034 {
05035     // シェーダプログラムの作成
05036     const GLuint program{ glCreateProgram() };
05037
05038     if (program > 0)
05039     {
05040         // ソースプログラムの文字列が空だったら 0 を返す
05041         if (csrc.empty()) return 0;
05042
05043         // コンピュートシェーダのシェーダオブジェクトを作成する
05044         const GLuint compShader{ glCreateShader(GL_COMPUTE_SHADER) };
05045         const GLchar* csrcp{ csrc.c_str() };
05046         glShaderSource(compShader, 1, &csrcp, nullptr);
05047         glCompileShader(compShader);
05048
05049         // コンピュートシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
05050         if (printShaderInfoLog(compShader, ctext)) glAttachShader(program, compShader);
05051         glDeleteShader(compShader);
05052
05053         // シェーダプログラムをリンクする
05054         glLinkProgram(program);
05055
05056         // プログラムオブジェクトが作成できなければ 0 を返す
05057         if (printProgramInfoLog(program) == GL_FALSE)
05058         {
05059             glDeleteProgram(program);
05060             return 0;
05061         }
05062     }
05063
05064     // プログラムオブジェクトを返す
05065     return program;
05066 }
05067
05068 //
05069 // コンピュートシェーダのソースファイルを読み込んでプログラムオブジェクトを作成する
05070 //
05071 //    comp コンピュートシェーダのソースファイル名
05072 //    戻り値 プログラムオブジェクトのプログラム名 (作成できなければ 0)
05073 //
05074 GLuint gg::ggLoadComputeShader(const std::string& comp)
05075 {
05076     // シェーダのソースファイルを読み込む
05077     std::string csrc;
05078     if (readShaderSource(comp, csrc))
05079     {

```

```
05080     // プログラムオブジェクトを作成する
05081     return ggCreateComputeShader(csrc.data(), comp);
05082 }
05083
05084 // プログラムオブジェクト作成失敗
05085 return 0;
05086 }
05087 #endif
05088
05089 //
05090 // 点: データ作成
05091 //
05092 void gg::GgPoints::load(const GgVector* pos, GLsizei count, GLenum usage)
05093 {
05094     // 頂点バッファオブジェクトを作成する
05095     position = std::make_shared<GgBuffer<GgVector>>(GL_ARRAY_BUFFER, pos,
05096     static_cast<GLsizei>(sizeof(GgVector)), count, usage);
05097     // このバッファオブジェクトは index == 0 の in 変数から入力する
05098     glVertexAttribPointer(0, static_cast<GLint>(pos->size()), GL_FLOAT, GL_FALSE, 0, 0);
05099     glEnableVertexAttribArray(0);
05100 }
05101
05102 //
05103 // 点: 描画
05104 //
05105 void gg::GgPoints::draw(GLint first, GLsizei count) const
05106 {
05107     // 頂点配列オブジェクトを指定する
05108     GgShape::draw(first, count);
05109
05110     // 図形を描画する
05111     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05112 }
05113
05114 //
05115 // 三角形: データ作成
05116 //
05117 void gg::GgTriangles::load(const GgVertex* vert, GLsizei count, GLenum usage)
05118 {
05119     // 頂点バッファオブジェクトを作成する
05120     vertex = std::make_shared<GgBuffer<GgVertex>>(GL_ARRAY_BUFFER, vert,
05121     static_cast<GLsizei>(sizeof(GgVertex)), count, usage);
05122     // 頂点の位置は index == 0 の in 変数から入力する
05123     glVertexAttribPointer(0, static_cast<GLint>(vert->position.size()), GL_FLOAT, GL_FALSE,
05124     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, position));
05125     glEnableVertexAttribArray(0);
05126
05127     // 頂点の法線は index == 1 の in 変数から入力する
05128     glVertexAttribPointer(1, static_cast<GLint>(vert->normal.size()), GL_FLOAT, GL_FALSE,
05129     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, normal));
05130     glEnableVertexAttribArray(1);
05131 }
05132
05133 //
05134 // 三角形: 描画
05135 //
05136 void gg::GgTriangles::draw(GLint first, GLsizei count) const
05137 {
05138     // 頂点配列オブジェクトを指定する
05139     GgShape::draw(first, count);
05140
05141     // 図形を描画する
05142     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05143 }
05144
05145 //
05146 // オブジェクト: 描画
05147 //
05148 void gg::GgElements::draw(GLint first, GLsizei count) const
05149 {
05150     // 頂点配列オブジェクトを指定する
05151     GgShape::draw(first, count);
05152
05153     // 図形を描画する
05154     glDrawElements(getMode(), count > 0 ? count : getIndexCount() - first,
05155     GL_UNSIGNED_INT, static_cast<GLuint*>(0) + first);
05156 }
05157
05158 //
05159 // 点群を立方体状に生成する
05160 //
05161 std::shared_ptr<gg::GgPoints> gg::ggPointsCube(GLsizei count, GLfloat length, GLfloat cx, GLfloat cy,
05162     GLfloat cz)
05163 {
05164     // メモリを確保する
```

```

05164     std::vector<GgVector> pos(count);
05165
05166     // 点を生成する
05167     for (GLsizei v = 0; v < count; ++v)
05168     {
05169         const GgVector p
05170         {
05171             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cx,
05172             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cy,
05173             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cz,
05174             1.0f
05175         };
05176
05177         pos.emplace_back(p);
05178     }
05179
05180     // 点データの GgPoints オブジェクトを作成して返す
05181     return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05182 }
05183
05184 //
05185 // 点群を球状に生成する
05186 //
05187 std::shared_ptr<gg::GgPoints> gg::ggPointsSphere(GLsizei count, GLfloat radius,
05188     GLfloat cx, GLfloat cy, GLfloat cz)
05189 {
05190     // メモリを確保する
05191     std::vector<GgVector> pos(count);
05192
05193     // 点を生成する
05194     for (GLsizei v = 0; v < count; ++v)
05195     {
05196         const GLfloat r{ radius * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) };
05197         const GLfloat t{ 6.28318530717f * static_cast<GLfloat>(rand()) / (static_cast<GLfloat>(RAND_MAX) +
05198             1.0f) };
05199         const GLfloat cp{ 2.0f * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 1.0f };
05200         const GLfloat sp{ sqrtf(1.0f - cp * cp) };
05201         const GLfloat ct{ cosf(t) };
05202         const GLfloat st{ sinf(t) };
05203         const GgVector p{ r * sp * ct + cx, r * sp * st + cy, r * cp + cz, 1.0f };
05204
05205         pos.emplace_back(p);
05206     }
05207
05208     // 点データの GgPoints オブジェクトを作成して返す
05209     return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05210 }
05211
05212 // 矩形状に 2 枚の三角形を生成する
05213 //
05214 std::shared_ptr<gg::GgTriangles> gg::ggRectangle(GLfloat width, GLfloat height)
05215 {
05216     // 頂点属性
05217     std::array<gg::GgVertex, 4> vert;
05218
05219     // 頂点位置と法線を求める
05220     for (int v = 0; v < 4; ++v)
05221     {
05222         const GLfloat x{ ((v & 1) * 2 - 1) * width };
05223         const GLfloat y{ ((v & 2) - 1) * height };
05224
05225         vert[v] = gg::GgVertex{ x, y, 0.0f, 0.0f, 0.0f, 1.0f };
05226     }
05227
05228     // 矩形の GgTriangles オブジェクトを作成する
05229     return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05230     GL_TRIANGLE_STRIP);
05231 }
05232
05233 // 楕円状に三角形を生成する
05234 //
05235 std::shared_ptr<gg::GgTriangles> gg::ggEllipse(GLfloat width, GLfloat height, GLuint slices)
05236 {
05237     // 楕円のスケール
05238     constexpr GLfloat scale{ 0.5f };
05239
05240     // 作業用のメモリ
05241     std::vector<gg::GgVertex> vert;
05242
05243     // 頂点位置と法線を求める
05244     for (GLuint v = 0; v < slices; ++v)
05245     {
05246         const GLfloat t{ 6.28318530717f * static_cast<GLfloat>(v) / static_cast<GLfloat>(slices) };
05247         const GLfloat x{ cosf(t) * width * scale };
05248         const GLfloat y{ sinf(t) * height * scale };
05249     }
}

```

```
05249
05250     vert.emplace_back(x, y, 0.0f, 0.0f, 0.0f, 1.0f);
05251 }
05252
05253 // GgTriangles オブジェクトを作成する
05254 return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLES_FAN);
05255 }
05256
05257 //
05258 // Wavefront OBJ ファイルを読み込む (Arrays 形式)
05259 //
05260 std::shared_ptr<gg::GgTriangles> gg::ggArraysObj(const std::string& name, bool normalize)
05261 {
05262     std::vector<std::array<GLuint, 3>> group;
05263     std::vector<gg::GgSimpleShader::Material> material;
05264     std::vector<gg::GgVertex> vert;
05265
05266 // ファイルを読み込む
05267 if (!ggLoadSimpleObj(name, group, material, vert, normalize)) return nullptr;
05268
05269 // GgTriangles オブジェクトを作成する
05270 return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLES),
05271 GL_TRIANGLES);
05272
05273 //
05274 // Wavefront OBJ ファイル を読み込む (Elements 形式)
05275 //
05276 std::shared_ptr<gg::GgElements> gg::ggElementsObj(const std::string& name, bool normalize)
05277 {
05278     std::vector<std::array<GLuint, 3>> group;
05279     std::vector<gg::GgSimpleShader::Material> material;
05280     std::vector<gg::GgVertex> vert;
05281     std::vector<GLuint> face;
05282
05283 // ファイルを読み込む
05284 if (!ggLoadSimpleObj(name, group, material, vert, face, normalize)) return nullptr;
05285
05286 // GgElements オブジェクトを作成する
05287 return std::make_shared<gg::GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05288 face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05289 }
05290
05291 //
05292 // メッシュ形状を作成する (Elements 形式)
05293 //
05294 std::shared_ptr<gg::GgElements> gg::ggElementsMesh(GLuint slices, GLuint stacks, const
05295 GLfloat *pos) [3], const GLfloat (*norm) [3])
05296 {
05297 // 頂点属性
05298     std::vector<gg::GgVertex> vert((slices + 1) * (stacks + 1));
05299
05300 // 頂点の法線を求める
05301 for (GLuint j = 0; j <= stacks; ++j)
05302 {
05303     for (GLuint i = 0; i <= slices; ++i)
05304     {
05305         // 処理対象の頂点番号
05306         const GLuint k{ j * (slices + 1) + i };
05307
05308         // 頂点の法線
05309         gg::GgVector tnorm;
05310         tnorm[3] = 0.0f;
05311
05312         if (norm)
05313         {
05314             tnorm[0] = norm[k][0];
05315             tnorm[1] = norm[k][1];
05316             tnorm[2] = norm[k][2];
05317         }
05318         else
05319         {
05320             // 処理対象の頂点の周囲の頂点番号
05321             const GLuint kim{ i > 0 ? k - 1 : k };
05322             const GLuint kip{ i < slices ? k + 1 : k };
05323             const GLuint kjm{ j > 0 ? k - slices - 1 : k };
05324             const GLuint kjp{ j < stacks ? k + slices + 1 : k };
05325
05326             // 接線ベクトル
05327             const std::array<GLfloat, 3> t
05328             {
05329                 pos[kip][0] - pos[kim][0],
05330                 pos[kip][1] - pos[kim][1],
05331                 pos[kip][2] - pos[kim][2]
05332             };
05333 }
```

```

05333 // 従接線ベクトル
05334 const std::array<GLfloat, 3> b
05335 {
05336     pos[kjp][0] - pos[kjm][0],
05337     pos[kjp][1] - pos[kjm][1],
05338     pos[kjp][2] - pos[kjm][2]
05339 };
05340
05341 // 法線
05342 tnorm[0] = t[1] * b[2] - t[2] * b[1];
05343 tnorm[1] = t[2] * b[0] - t[0] * b[2];
05344 tnorm[2] = t[0] * b[1] - t[1] * b[0];
05345
05346 // 法線の正規化
05347 ggNormalize3(tnorm);
05348 }
05349
05350 // 頂点の位置
05351 const gg::GgVector tpos{ pos[k][0], pos[k][1], pos[k][2], 1.0f };
05352
05353 // 頂点属性の保存
05354 vert.emplace_back(tpos, tnorm);
05355 }
05356 }
05357
05358 // 頂点のインデックス (三角形データ)
05359 std::vector<GLuint> face;
05360
05361 // 頂点のインデックスを求める
05362 for (GLuint j = 0; j < stacks; ++j)
05363 {
05364     for (GLuint i = 0; i < slices; ++i)
05365     {
05366         // 処理対象のマス
05367         const GLuint k{ (slices + 1) * j + i };
05368
05369         // マスの上半分の三角形
05370         face.emplace_back(k);
05371         face.emplace_back(k + slices + 2);
05372         face.emplace_back(k + 1);
05373
05374         // マスのお下半分の三角形
05375         face.emplace_back(k);
05376         face.emplace_back(k + slices + 1);
05377         face.emplace_back(k + slices + 2);
05378     }
05379 }
05380
05381 // GgElements オブジェクトを作成する
05382 return std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05383     face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05384 }
05385
05386 //
05387 // 球状に三角形データを生成する (Elements 形式)
05388 //
05389 std::shared_ptr<gg::GgElements> gg::ggElementsSphere(GLfloat radius, int slices, int stacks)
05390 {
05391     // 頂点の位置と法線
05392     std::vector<GLfloat> p, n;
05393
05394     // 頂点の位置と法線を求める
05395     for (int j = 0; j <= stacks; ++j)
05396     {
05397         const GLfloat t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };
05398         const GLfloat ph{ 3.1415926536f * t };
05399         const GLfloat y{ cosf(ph) };
05400         const GLfloat r{ sinf(ph) };
05401
05402         for (int i = 0; i <= slices; ++i)
05403         {
05404             const GLfloat s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };
05405             const GLfloat th{ -2.0f * 3.1415926536f * s };
05406             const GLfloat x{ r * cosf(th) };
05407             const GLfloat z{ r * sinf(th) };
05408
05409             // 頂点の座標値
05410             p.emplace_back(x * radius);
05411             p.emplace_back(y * radius);
05412             p.emplace_back(z * radius);
05413
05414             // 頂点の法線
05415             n.emplace_back(x);
05416             n.emplace_back(y);
05417             n.emplace_back(z);
05418         }
05419     }

```

```

05420
05421 // GgElements オブジェクトを作成する
05422 return ggElementsMesh(slices, stacks, reinterpret_cast<GLfloat(*)[3]>(p.data()),
05423     reinterpret_cast<GLfloat(*)[3]>(p.data())));
05424 }
05425
05426 //
05427 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の環境光成分を設定する
05428 //
05429 // r 光源の強度の環境光成分の赤成分
05430 // g 光源の強度の環境光成分の緑成分
05431 // b 光源の強度の環境光成分の青成分
05432 // a 光源の強度の環境光成分の不透明度、デフォルトは 1
05433 // first 値を設定する光源データの最初の番号、デフォルトは 0
05434 // count 値を設定する光源データの数、デフォルトは 1
05435 //
05436 void gg::GgSimpleShader::LightBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05437 GLint first, GLsizei count) const
05438 {
05439 // データを格納するバッファオブジェクトの先頭のポインタ
05440 char* const start{ static_cast<char*>(map(first, count)) };
05441 for (GLsizei i = 0; i < count; ++i)
05442 {
05443 // バッファオブジェクトの i 番目のブロックのポインタ
05444 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05445
05446 // 光源の環境光成分を設定する
05447 light->ambient[0] = r;
05448 light->ambient[1] = g;
05449 light->ambient[2] = b;
05450 light->ambient[3] = a;
05451 }
05452 unmap();
05453 }
05454
05455 //
05456 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の環境光成分を設定する
05457 //
05458 // ambient 光源の強度の環境光成分
05459 // first 値を設定する光源データの最初の番号、デフォルトは 0
05460 // count 値を設定する光源データの数、デフォルトは 1
05461 //
05462 void gg::GgSimpleShader::LightBuffer::loadAmbient(const GgVector& ambient,
05463 GLint first, GLsizei count) const
05464 {
05465 // データを格納するバッファオブジェクトの先頭のポインタ
05466 char* const start{ static_cast<char*>(map(first, count)) };
05467 for (GLsizei i = 0; i < count; ++i)
05468 {
05469 // バッファオブジェクトの i 番目のブロックのポインタ
05470 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05471
05472 // 光源の強度の環境光成分を設定する
05473 light->ambient = ambient;
05474 }
05475 unmap();
05476 }
05477
05478 //
05479 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の拡散反射光成分を設定する
05480 //
05481 // r 光源の強度の拡散反射光成分の赤成分
05482 // g 光源の強度の拡散反射光成分の緑成分
05483 // b 光源の強度の拡散反射光成分の青成分
05484 // a 光源の強度の拡散反射光成分の不透明度、デフォルトは 1
05485 // first 値を設定する光源データの最初の番号、デフォルトは 0
05486 // count 値を設定する光源データの数、デフォルトは 1
05487 //
05488 void gg::GgSimpleShader::LightBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05489 GLint first, GLsizei count) const
05490 {
05491 // データを格納するバッファオブジェクトの先頭のポインタ
05492 char* const start{ static_cast<char*>(map(first, count)) };
05493 for (GLsizei i = 0; i < count; ++i)
05494 {
05495 // バッファオブジェクトの i 番目のブロックのポインタ
05496 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05497
05498 // 光源の拡散反射光成分を設定する
05499 light->diffuse[0] = r;
05500 light->diffuse[1] = g;
05501 light->diffuse[2] = b;
05502 light->diffuse[3] = a;
05503 }
05504 unmap();
05505 }
05506

```

```

05507 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の拡散反射光成分を設定する
05508 // ambient 光源の強度の拡散反射光成分
05509 //
05510 // first 値を設定する光源データの最初の番号, デフォルトは 0
05512 // count 値を設定する光源データの数, デフォルトは 1
05513 //
05514 void gg::GgSimpleShader::LightBuffer::loadDiffuse(const GgVector& diffuse,
05515 GLint first, GLsizei count) const
05516 {
05517 // データを格納するバッファオブジェクトの先頭のポインタ
05518 char* const start{ static_cast<char*>(map(first, count)) };
05519 for (GLsizei i = 0; i < count; ++i)
05520 {
05521 // バッファオブジェクトの i 番目のブロックのポインタ
05522 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05523
05524 // 光源の強度の拡散反射光成分を設定する
05525 light->diffuse = diffuse;
05526 }
05527 unmmap();
05528 }
05529
05530 //
05531 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の鏡面反射光成分を設定する
05532 //
05533 // r 光源の強度の鏡面反射光成分の赤成分
05534 // g 光源の強度の鏡面反射光成分の緑成分
05535 // b 光源の強度の鏡面反射光成分の青成分
05536 // a 光源の強度の鏡面反射光成分の不透明度, デフォルトは 1
05537 // first 値を設定する光源データの最初の番号, デフォルトは 0
05538 // count 値を設定する光源データの数, デフォルトは 1
05539 //
05540 void gg::GgSimpleShader::LightBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05541 GLint first, GLsizei count) const
05542 {
05543 // データを格納するバッファオブジェクトの先頭のポインタ
05544 char* const start{ static_cast<char*>(map(first, count)) };
05545 for (GLsizei i = 0; i < count; ++i)
05546 {
05547 // バッファオブジェクトの i 番目のブロックのポインタ
05548 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05549
05550 // 光源の鏡面反射光成分を設定する
05551 light->specular[0] = r;
05552 light->specular[1] = g;
05553 light->specular[2] = b;
05554 light->specular[3] = a;
05555 }
05556 unmmap();
05557 }
05558
05559 //
05560 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の鏡面反射光成分を設定する
05561 //
05562 // ambient 光源の強度の鏡面反射光成分
05563 // first 値を設定する光源データの最初の番号, デフォルトは 0
05564 // count 値を設定する光源データの数, デフォルトは 1
05565 //
05566 void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& specular,
05567 GLint first, GLsizei count) const
05568 {
05569 // データを格納するバッファオブジェクトの先頭のポインタ
05570 char* const start{ static_cast<char*>(map(first, count)) };
05571 for (GLsizei i = 0; i < count; ++i)
05572 {
05573 // バッファオブジェクトの i 番目のブロックのポインタ
05574 Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05575
05576 // 光源の強度の鏡面反射光成分を設定する
05577 light->specular = specular;
05578 }
05579 unmmap();
05580 }
05581
05582 //
05583 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の色を設定するか位置は変更しない
05584 //
05585 // material 光源の特性の GgSimpleShader::Light 構造体
05586 // first 値を設定する光源データの最初の番号, デフォルトは 0
05587 // count 値を設定する光源データの数, デフォルトは 1
05588 //
05589 void gg::GgSimpleShader::LightBuffer::loadColor(const Light& color,
05590 GLint first, GLsizei count) const
05591 {
05592 // データを格納するバッファオブジェクトの先頭のポインタ
05593 char* const start{ static_cast<char*>(map(first, count)) };

```

```

05594     for (GLsizei i = 0; i < count; ++i)
05595     {
05596         // バッファオブジェクトの i 番目のブロックのポインタ
05597         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05598
05599         // 光源の色を設定する
05600         light->ambient = color.ambient;
05601         light->diffuse = color.diffuse;
05602         light->specular = color.specular;
05603     }
05604     unmap();
05605 }
05606
05607 //
05608 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の位置を設定する
05609 //
05610 //    x 光源の位置の x 座標
05611 //    y 光源の位置の y 座標
05612 //    z 光源の位置の z 座標
05613 //    w 光源の位置の w 座標, デフォルトは 1
05614 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05615 //    count 値を設定する光源データの数, デフォルトは 1
05616 //
05617 void gg::GgSimpleShader::LightBuffer::loadPosition(GLfloat x, GLfloat y, GLfloat z, GLfloat w,
05618 GLint first, GLsizei count) const
05619 {
05620     // データを格納するバッファオブジェクトの先頭のポインタ
05621     char* const start{ static_cast<char*>(map(first, count)) };
05622     for (GLsizei i = 0; i < count; ++i)
05623     {
05624         // バッファオブジェクトの i 番目のブロックのポインタ
05625         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05626
05627         // 光源の位置を設定する
05628         light->position[0] = x;
05629         light->position[1] = y;
05630         light->position[2] = z;
05631         light->position[3] = w;
05632     }
05633     unmap();
05634 }
05635
05636 //
05637 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の位置を設定する
05638 //
05639 //    position 光源の位置
05640 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05641 //    count 値を設定する光源データの数, デフォルトは 1
05642 //
05643 void gg::GgSimpleShader::LightBuffer::loadPosition(const GgVector& position,
05644 GLint first, GLsizei count) const
05645 {
05646     // データを格納するバッファオブジェクトの先頭のポインタ
05647     char* const start{ static_cast<char*>(map(first, count)) };
05648     for (GLsizei i = 0; i < count; ++i)
05649     {
05650         // バッファオブジェクトの i 番目のブロックのポインタ
05651         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05652
05653         // 光源の位置を設定する
05654         light->position = position;
05655     }
05656     unmap();
05657 }
05658
05659 //
05660 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数を設定する
05661 //
05662 //    r 環境光に対する反射係数の赤成分
05663 //    g 環境光に対する反射係数の緑成分
05664 //    b 環境光に対する反射係数の青成分
05665 //    a 環境光に対する反射係数の不透明度, デフォルトは 1
05666 //    first 値を設定する材質データの最初の番号, デフォルトは 0
05667 //    count 値を設定する材質データの数, デフォルトは 1
05668 //
05669 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05670 GLint first, GLsizei count) const
05671 {
05672     // データを格納するバッファオブジェクトの先頭のポインタ
05673     char* const start{ static_cast<char*>(map(first, count)) };
05674     for (GLsizei i = 0; i < count; ++i)
05675     {
05676         // バッファオブジェクトの i 番目のブロックのポインタ
05677         Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05678
05679         // 環境光に対する反射係数を設定する
05680         material->ambient[0] = r;

```

```

05681     material->ambient[1] = g;
05682     material->ambient[2] = b;
05683     material->ambient[3] = a;
05684 }
05685 unmap();
05686 }
05687
05688 //
05689 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数を設定する
05690 //
05691 // ambient 環境光に対する反射係数
05692 // first 値を設定する光源データの最初の番号, デフォルトは 0
05693 // count 値を設定する光源データの数, デフォルトは 1
05694 //
05695 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(const GgVector& ambient,
05696   GLint first, GLsizei count) const
05697 {
05698     // データを格納するバッファオブジェクトの先頭のポインタ
05699     char* const start{ static_cast<char*>(map(first, count)) };
05700     for (GLsizei i = 0; i < count; ++i)
05701     {
05702       // バッファオブジェクトの i 番目のブロックのポインタ
05703       Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05704
05705       // 光源の強度の環境光成分を設定する
05706       material->ambient = ambient;
05707     }
05708     unmap();
05709 }
05710
05711 //
05712 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：拡散反射係数を設定する
05713 //
05714 // r 拡散反射係数の赤成分
05715 // g 拡散反射係数の緑成分
05716 // b 拡散反射係数の青成分
05717 // a 拡散反射係数の不透明度, デフォルトは 1
05718 // first 値を設定する材質データの最初の番号, デフォルトは 0
05719 // count 値を設定する材質データの数, デフォルトは 1
05720 //
05721 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05722   GLint first, GLsizei count) const
05723 {
05724     // データを格納するバッファオブジェクトの先頭のポインタ
05725     char* const start{ static_cast<char*>(map(first, count)) };
05726     for (GLsizei i = 0; i < count; ++i)
05727     {
05728       // バッファオブジェクトの i 番目のブロックのポインタ
05729       Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05730
05731       // 拡散反射係数を設定する
05732       material->diffuse[0] = r;
05733       material->diffuse[1] = g;
05734       material->diffuse[2] = b;
05735       material->diffuse[3] = a;
05736     }
05737     unmap();
05738 }
05739
05740 //
05741 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：拡散反射係数を設定する
05742 //
05743 // diffuse 拡散反射係数
05744 // first 値を設定する光源データの最初の番号, デフォルトは 0
05745 // count 値を設定する光源データの数, デフォルトは 1
05746 //
05747 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(const GgVector& diffuse,
05748   GLint first, GLsizei count) const
05749 {
05750     // データを格納するバッファオブジェクトの先頭のポインタ
05751     char* const start{ static_cast<char*>(map(first, count)) };
05752     for (GLsizei i = 0; i < count; ++i)
05753     {
05754       // バッファオブジェクトの i 番目のブロックのポインタ
05755       Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05756
05757       // 光源の強度の環境光成分を設定する
05758       material->diffuse = diffuse;
05759     }
05760     unmap();
05761 }
05762
05763 //
05764 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する
05765 //
05766 // r 環境光に対する反射係数と拡散反射係数の赤成分
05767 // g 環境光に対する反射係数と拡散反射係数の緑成分

```

```

05768 // b 環境光に対する反射係数と拡散反射係数の青成分
05769 // a 環境光に対する反射係数と拡散反射係数の不透明度, デフォルトは 1
05770 // first 値を設定する材質データの最初の番号, デフォルトは 0
05771 // count 値を設定する材質データの数, デフォルトは 1
05772 //
05773 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(GLfloat r, GLfloat g, GLfloat b,
05774     GLfloat a,
05775     GLint first, GLsizei count) const
05776 {
05777     // データを格納するバッファオブジェクトの先頭のポインタ
05778     char* const start{ static_cast<char*>(map(first, count)) };
05779     for (GLsizei i = 0; i < count; ++i)
05780     {
05781         // バッファオブジェクトの i 番目のブロックのポインタ
05782         Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05783         // 環境光に対する反射係数と拡散反射係数を設定する
05784         material->ambient[0] = material->diffuse[0] = r;
05785         material->ambient[1] = material->diffuse[1] = g;
05786         material->ambient[2] = material->diffuse[2] = b;
05787         material->ambient[3] = material->diffuse[3] = a;
05788     }
05789     unmap();
05790 }
05791
05792 //
05793 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する
05794 //
05795 // color 環境光に対する反射係数と拡散反射係数
05796 // first 値を設定する光源データの最初の番号, デフォルトは 0
05797 // count 値を設定する光源データの数, デフォルトは 1
05798 //
05799 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GgVector& color,
05800     GLint first, GLsizei count) const
05801 {
05802     // データを格納するバッファオブジェクトの先頭のポインタ
05803     char* const start{ static_cast<char*>(map(first, count)) };
05804     for (GLsizei i = 0; i < count; ++i)
05805     {
05806         // バッファオブジェクトの i 番目のブロックのポインタ
05807         Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05808         // 光源の強度の環境光成分を設定する
05809         material->ambient = material->diffuse = color;
05810     }
05811     unmap();
05812 }
05813
05814
05815 //
05816 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する
05817 //
05818 // color 環境光に対する反射係数と拡散反射係数を格納した GLfloat 型の 4 要素の配列
05819 // first 値を設定する材質データの最初の番号, デフォルトは 0
05820 // count 値を設定する材質データの数, デフォルトは 1
05821 //
05822 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GLfloat* color,
05823     GLint first, GLsizei count) const
05824 {
05825     // ambient 要素のバイトオフセット
05826     constexpr GLint ambientOffset{ offsetof(Material, ambient) };
05827
05828     // ambient 要素のバイト数
05829     constexpr size_t ambientSize{ sizeof(Material::diffuse) };
05830
05831     // diffuse 要素のバイトオフセット
05832     constexpr GLint diffuseOffset{ offsetof(Material, diffuse) };
05833
05834     // diffuse 要素のバイト数
05835     constexpr size_t diffuseSize{ sizeof(Material::diffuse) };
05836
05837     // 元のデータの先頭
05838     const char* source{ reinterpret_cast<const char*>(color) };
05839
05840     // first 番目のブロックから count 個の ambient 要素と diffuse 要素に値を設定する
05841     bind();
05842     for (GLsizei i = 0; i < count; ++i)
05843     {
05844         // 格納先
05845         const GLsizeiptr destination{ getStride() * (first + i) };
05846
05847         // first + i 番目のブロックの ambient 要素に値を設定する
05848         glBufferSubData(getTarget(), destination + ambientOffset, ambientSize, source + i * ambientSize);
05849
05850         // first + i 番目のブロックの diffuse 要素に値を設定する
05851         glBufferSubData(getTarget(), destination + diffuseOffset, diffuseSize, source + i * diffuseSize);
05852     }
05853 }

```

```

05854 //
05855 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：鏡面反射係数を設定する
05856 //
05857 //
05858 // r 鏡面反射係数の赤成分
05859 // g 鏡面反射係数の緑成分
05860 // b 鏡面反射係数の青成分
05861 // a 鏡面反射係数の不透明度, デフォルトは 1
05862 // first 値を設定する材質データの最初の番号, デフォルトは 0
05863 // count 値を設定する材質データの数, デフォルトは 1
05864 //
05865 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05866 GLint first, GLsizei count) const
05867 {
05868 // データを格納するバッファオブジェクトの先頭のポインタ
05869 char* const start{ static_cast<char*>(map(first, count)) };
05870 for (GLsizei i = 0; i < count; ++i)
05871 {
05872 // バッファオブジェクトの i 番目のブロックのポインタ
05873 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05874
05875 // 鏡面反射係数を設定する
05876 material->specular[0] = r;
05877 material->specular[1] = g;
05878 material->specular[2] = b;
05879 material->specular[3] = a;
05880 }
05881 unmap();
05882 }
05883
05884 //
05885 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：鏡面反射係数を設定する
05886 //
05887 // specular 鏡面反射係数
05888 // first 値を設定する光源データの最初の番号, デフォルトは 0
05889 // count 値を設定する光源データの数, デフォルトは 1
05890 //
05891 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(const GgVector& specular,
05892 GLint first, GLsizei count) const
05893 {
05894 // データを格納するバッファオブジェクトの先頭のポインタ
05895 char* const start{ static_cast<char*>(map(first, count)) };
05896 for (GLsizei i = 0; i < count; ++i)
05897 {
05898 // バッファオブジェクトの i 番目のブロックのポインタ
05899 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05900
05901 // 光源の強度の環境光成分を設定する
05902 material->specular = specular;
05903 }
05904 unmap();
05905 }
05906
05907 //
05908 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：輝き係数を設定する
05909 //
05910 // shininess 輝き係数
05911 // first 値を設定する材質データの最初の番号, デフォルトは 0
05912 // count 値を設定する材質データの数, デフォルトは 1
05913 //
05914 void gg::GgSimpleShader::MaterialBuffer::loadShininess(GLfloat shininess,
05915 GLint first, GLsizei count) const
05916 {
05917 // データを格納するバッファオブジェクトの先頭のポインタ
05918 char* const start{ static_cast<char*>(map(first, count)) };
05919 for (GLsizei i = 0; i < count; ++i)
05920 {
05921 // バッファオブジェクトの i 番目のブロックのポインタ
05922 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05923 material->shininess = shininess;
05924 }
05925 unmap();
05926 }
05927
05928 //
05929 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：輝き係数を設定する
05930 //
05931 // shininess 輝き係数
05932 // first 値を設定する材質データの最初の番号, デフォルトは 0
05933 // count 値を設定する材質データの数, デフォルトは 1
05934 //
05935 void gg::GgSimpleShader::MaterialBuffer::loadShininess(const GLfloat* shininess,
05936 GLint first, GLsizei count) const
05937 {
05938 // データを格納するバッファオブジェクトの先頭のポインタ
05939 char* const start{ static_cast<char*>(map(first, count)) };
05940 for (GLsizei i = 0; i < count; ++i)

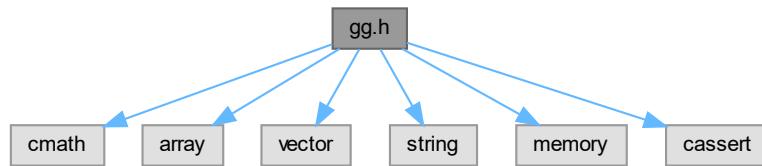
```

```
05941  {
05942      // バッファオブジェクトの i 番目のブロックのポインタ
05943      Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05944      material->shininess = shininess[i];
05945  }
05946  unmap();
05947 }
05948
05949 //
05950 // 三角形に単純な陰影付けを行うシェーダ：シェーダのソースファイルの読み込み
05951 //
05952 bool gg::GgSimpleShader::load(const std::string& vert, const std::string& frag, const std::string&
05953     geom,
05954     GLint nvarying, const char* const* varyings)
05955 {
05956     if (!GgPointShader::load(vert, frag, geom, nvarying, varyings)) return false;
05957     mnLoc = glGetUniformLocation(get(), "mn");
05958     lightIndex = glGetUniformLocation(get(), "Light");
05959     glUniformBlockBinding(get(), lightIndex, 0);
05960     materialIndex = glGetUniformLocation(get(), "Material");
05961     glUniformBlockBinding(get(), materialIndex, 1);
05962
05963     return true;
05964 }
05965
05966 //
05967 // Wavefront OBJ 形式のデータ：コンストラクタ
05968 //
05969 gg::GgSimpleObj::GgSimpleObj(const std::string& name, bool normalize)
05970 {
05971     // 作業用のメモリ
05972     std::vector<GgSimpleShader::Material> mat;
05973     std::vector<GgVertex> vert;
05974     std::vector<GLuint> face;
05975
05976     // グループのデータのメモリを確保する
05977     group = std::make_shared<std::vector<std::array<GLuint, 3>>>();
05978
05979     // ファイルを読み込む
05980     if (ggLoadSimpleObj(name, *group, mat, vert, face, normalize))
05981     {
05982         // 頂点バッファオブジェクトを作成する
05983         data = std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05984             face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05985
05986         // 描画するオブジェクトを切り替えるために頂点配列オブジェクトを閉じておく
05987         glBindVertexArray(0);
05988
05989         // 材質データを設定する
05990         material = std::make_shared<GgSimpleShader::MaterialBuffer>(mat.data(),
05991             static_cast<GLsizei>(mat.size()));
05992     }
05993
05994 //
05995 // Wavefront OBJ 形式のデータ：図形の描画
05996 //
05997 void gg::GgSimpleObj::draw(GLint first, GLsizei count) const
05998 {
05999     // 保持しているグループの数
06000     const GLsizei ng{ static_cast<GLsizei>(group->size()) };
06001
06002     // 描画する最後のグループの次
06003     GLsizei last(count <= 0 ? ng : first + count);
06004     if (last > ng) last = ng;
06005
06006     for (GLsizei i = first; i < last; ++i)
06007     {
06008         // グループのデータ
06009         const auto& g{ (*group)[i] };
06010
06011         // 材質を設定する
06012         material->select(g[2]);
06013
06014         // 図形を描画する
06015         data->draw(g[0], g[1]);
06016     }
06017 }
```

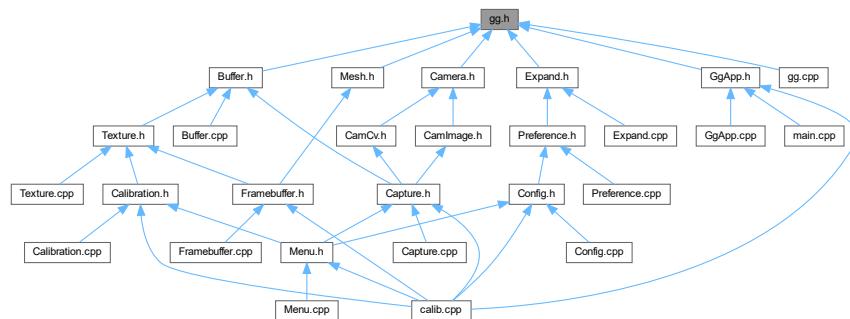
## 9.35 gg.h ファイル

```
#include <cmath>
#include <array>
#include <vector>
#include <string>
#include <memory>
#include <cassert>
```

gg.h の依存先関係図:



被依存関係図:



## クラス

- class [gg::GgVector](#)
- class [gg::GgMatrix](#)
- class [gg::GgQuaternion](#)
- class [gg::GgTrackball](#)
- class [gg::GgTexture](#)
- class [gg::GgColorTexture](#)
- class [gg::GgNormalTexture](#)
- class [gg::GgBuffer< T >](#)
- class [gg::GgUniformBuffer< T >](#)
- class [gg::GgVertexArray](#)
- class [gg::GgShape](#)
- class [gg::GgPoints](#)
- struct [gg::GgVertex](#)

- class `gg::GgTriangles`
- class `gg::GgElements`
- class `gg::GgShader`
- class `gg::GgPointShader`
- class `gg::GgSimpleShader`
- struct `gg::GgSimpleShader::Light`
- class `gg::GgSimpleShader::LightBuffer`
- struct `gg::GgSimpleShader::Material`
- class `gg::GgSimpleShader::MaterialBuffer`
- class `gg::GgSimpleObj`

## 名前空間

- namespace `gg`

## マクロ定義

- `#define ggError()`
- `#define ggFBOError()`

## 型定義

- using `pathString` = std::string
- using `pathChar` = char

## 列挙型

- enum `gg::BindingPoints` { `gg::LightBindingPoint` = 0 , `gg::MaterialBindingPoint` }

## 関数

- `pathString Utf8ToTChar (const std::string &string)`
- `std::string TCharToUtf8 (const pathString &cstring)`
- void `gg::gglInit ()`
- void `gg::ggError (const std::string &name="", unsigned int line=0)`
- void `gg::ggFBOError (const std::string &name="", unsigned int line=0)`
- void `gg::ggCross (GLfloat *c, const GLfloat *a, const GLfloat *b)`
- `GLfloat gg::ggDot3 (const GLfloat *a, const GLfloat *b)`
- `GLfloat gg::ggLength3 (const GLfloat *a)`
- `GLfloat gg::ggDistance3 (const GLfloat *a, const GLfloat *b)`
- void `gg::ggNormalize3 (const GLfloat *a, GLfloat *b)`
- void `gg::ggNormalize3 (GLfloat *a)`
- `GLfloat gg::ggDot4 (const GLfloat *a, const GLfloat *b)`
- `GLfloat gg::ggLength4 (const GLfloat *a)`
- `GLfloat gg::ggDistance4 (const GLfloat *a, const GLfloat *b)`
- void `gg::ggNormalize4 (const GLfloat *a, GLfloat *b)`
- void `gg::ggNormalize4 (GLfloat *a)`
- const `GgVector & gg::operator+ (const GgVector &v)`
- `GgVector gg::operator+ (GLfloat a, const GgVector &b)`
- const `GgVector gg::operator- (const GgVector &v)`

- GgVector [gg::operator-](#) (GLfloat a, const GgVector &b)
- GgVector [gg::operator\\*](#) (GLfloat a, const GgVector &b)
- GgVector [gg::operator/](#) (GLfloat a, const GgVector &b)
- GgVector [gg::ggCross](#) (const GgVector &a, const GgVector &b)
- GLfloat [gg::ggDot3](#) (const GgVector &a, const GgVector &b)
- GLfloat [gg::ggLength3](#) (const GgVector &a)
- GLfloat [gg::ggDistance3](#) (const GgVector &a, const GgVector &b)
- GgVector [gg::ggNormalize3](#) (const GgVector &a)
- void [gg::ggNormalize3](#) (GgVector \*a)
- GLfloat [gg::ggDot4](#) (const GgVector &a, const GgVector &b)
- GLfloat [gg::ggLength4](#) (const GgVector &a)
- GLfloat [gg::ggDistance4](#) (const GgVector &a, const GgVector &b)
- GgVector [gg::ggNormalize4](#) (const GgVector &a)
- void [gg::ggNormalize4](#) (GgVector \*a)
- GgMatrix [gg::ggIdentity](#) ()
- GgMatrix [gg::ggTranslate](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
- GgMatrix [gg::ggTranslate](#) (const GLfloat \*t)
- GgMatrix [gg::ggTranslate](#) (const GgVector &t)
- GgMatrix [gg::ggScale](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
- GgMatrix [gg::ggScale](#) (const GLfloat \*s)
- GgMatrix [gg::ggScale](#) (const GgVector &s)
- GgMatrix [gg::ggRotateX](#) (GLfloat a)
- GgMatrix [gg::ggRotateY](#) (GLfloat a)
- GgMatrix [gg::ggRotateZ](#) (GLfloat a)
- GgMatrix [gg::ggRotate](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat a)
- GgMatrix [gg::ggRotate](#) (const GLfloat \*r, GLfloat a)
- GgMatrix [gg::ggRotate](#) (const GgVector &r, GLfloat a)
- GgMatrix [gg::ggRotate](#) (const GLfloat \*r)
- GgMatrix [gg::ggRotate](#) (const GgVector &r)
- GgMatrix [gg::ggLookat](#) (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)
- GgMatrix [gg::ggLookat](#) (const GLfloat \*e, const GLfloat \*t, const GLfloat \*u)
- GgMatrix [gg::ggLookat](#) (const GgVector &e, const GgVector &t, const GgVector &u)
- GgMatrix [gg::ggOrthogonal](#) (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
- GgMatrix [gg::ggFrustum](#) (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
- GgMatrix [gg::ggPerspective](#) (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)
- GgMatrix [gg::ggTranspose](#) (const GgMatrix &m)
- GgMatrix [gg::ggInvert](#) (const GgMatrix &m)
- GgMatrix [gg::ggNormal](#) (const GgMatrix &m)
- GgQuaternion [gg::ggQuaternion](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- GgQuaternion [gg::ggQuaternion](#) (const GLfloat \*a)
- GgQuaternion [gg::ggIdentityQuaternion](#) ()
- GgQuaternion [gg::ggMatrixQuaternion](#) (const GLfloat \*a)
- GgQuaternion [gg::ggMatrixQuaternion](#) (const GgMatrix &m)
- GgMatrix [gg::ggQuaternionMatrix](#) (const GgQuaternion &q)
- GgMatrix [gg::ggQuaternionTransposeMatrix](#) (const GgQuaternion &q)
- GgQuaternion [gg::ggRotateQuaternion](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat a)
- GgQuaternion [gg::ggRotateQuaternion](#) (const GLfloat \*v, GLfloat a)
- GgQuaternion [gg::ggRotateQuaternion](#) (const GLfloat \*v)
- GgQuaternion [gg::ggEulerQuaternion](#) (GLfloat heading, GLfloat pitch, GLfloat roll)
- GgQuaternion [gg::ggEulerQuaternion](#) (const GLfloat \*e)
- GgQuaternion [gg::ggSlerp](#) (const GLfloat \*a, const GLfloat \*b, GLfloat t)
- GgQuaternion [gg::ggSlerp](#) (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)
- GgQuaternion [gg::ggSlerp](#) (const GgQuaternion &q, const GLfloat \*a, GLfloat t)

- GgQuaternion `gg::ggSlerp` (const GLfloat \*a, const GgQuaternion &q, GLfloat t)
- GLfloat `gg::ggNorm` (const GgQuaternion &q)
- GgQuaternion `gg::ggNormalize` (const GgQuaternion &q)
- GgQuaternion `gg::ggConjugate` (const GgQuaternion &q)
- GgQuaternion `gg::ggInvert` (const GgQuaternion &q)
- bool `gg::ggSaveTga` (const std::string &name, const void \*buffer, unsigned int width, unsigned int height, unsigned int depth)
- bool `gg::ggSaveColor` (const std::string &name)
- bool `gg::ggSaveDepth` (const std::string &name)
- bool `gg::ggReadImage` (const std::string &name, std::vector< GLubyte > &image, GLsizei \*pWidth, GLsizei \*pHeight, GLenum \*pFormat)
- GLuint `gg::ggLoadTexture` (const GLvoid \*image, GLsizei width, GLsizei height, GLenum format=GL\_RGB, GLenum type=GL\_UNSIGNED\_BYTE, GLenum internal=GL\_RGB, GLenum wrap=GL\_CLAMP\_TO\_EDGE, bool swizzle=true)
- GLuint `gg::ggLoadImage` (const std::string &name, GLsizei \*pWidth=nullptr, GLsizei \*pHeight=nullptr, GLenum internal=0, GLenum wrap=GL\_CLAMP\_TO\_EDGE)
- void `gg::ggCreateNormalMap` (const GLubyte \*hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)
- GLuint `gg::ggLoadHeight` (const std::string &name, GLfloat nz, GLsizei \*pWidth=nullptr, GLsizei \*pHeight=nullptr, GLenum internal=GL\_RGBA)
- GLuint `gg::ggCreateShader` (const std::string &vsr, const std::string &fsr="", const std::string &gsr="", GLint nvarying=0, const char \*const \*varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string &gtext="geometry shader")
- GLuint `gg::ggLoadShader` (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char \*const \*varyings=nullptr)
- GLuint `gg::ggLoadShader` (const std::array< std::string, 3 > &files, GLint nvarying=0, const char \*const \*varyings=nullptr)
- GLuint `gg::ggCreateComputeShader` (const std::string &csr, const std::string &ctext="compute shader")
- GLuint `gg::ggLoadComputeShader` (const std::string &comp)
- std::shared\_ptr< GgPoints > `gg::ggPointsCube` (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared\_ptr< GgPoints > `gg::ggPointsSphere` (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared\_ptr< GgTriangles > `gg::ggRectangle` (GLfloat width=1.0f, GLfloat height=1.0f)
- std::shared\_ptr< GgTriangles > `gg::ggEllipse` (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)
- std::shared\_ptr< GgTriangles > `gg::ggArraysObj` (const std::string &name, bool normalize=false)
- std::shared\_ptr< GgElements > `gg::ggElementsObj` (const std::string &name, bool normalize=false)
- std::shared\_ptr< GgElements > `gg::ggElementsMesh` (GLuint slices, GLuint stacks, const GLfloat(\*norm)[3], const GLfloat(\*pos)[3]=nullptr)
- std::shared\_ptr< GgElements > `gg::ggElementsSphere` (GLfloat radius=1.0f, int slices=16, int stacks=8)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, bool normalize=false)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)

## 変数

- GLint `gg::ggBufferAlignment`  
使用している GPU のバッファアライメント。

### 9.35.1 詳解

ゲームグラフィックス特論の宿題用補助プログラム GLFW3 版の宣言。

著者

Kohe Tokoi

日付

January 5, 2024

[gg.h](#) に定義があります。

### 9.35.2 マクロ定義詳解

#### 9.35.2.1 ggError

```
#define ggError( )
```

OpenGL のエラーの発生を検知したときにソースファイル名と行番号を表示する。

覚え書き

このマクロを置いた場所（より前）で OpenGL のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

[gg.h](#) の 1392 行目に定義があります。

#### 9.35.2.2 ggFBOError

```
#define ggFBOError( )
```

FBO のエラーの発生を検知したときにソースファイル名と行番号を表示する。

覚え書き

このマクロを置いた場所（より前）で FBO のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

[gg.h](#) の 1419 行目に定義があります。

### 9.35.3 型定義詳解

#### 9.35.3.1 pathChar

```
using pathChar = char
gg.h の 78 行目に定義があります。
```

#### 9.35.3.2 pathString

```
using pathString = std::string
gg.h の 77 行目に定義があります。
```

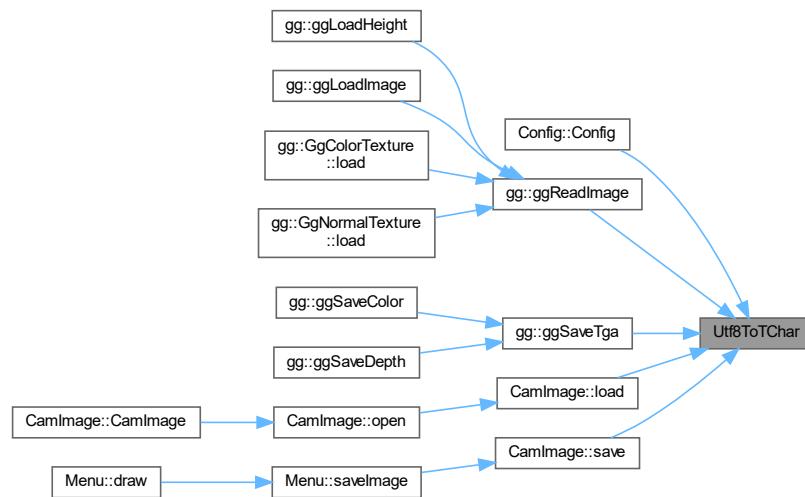
### 9.35.4 関数詳解

#### 9.35.4.1 TCharToUtf8()

```
std::string TCHARToUtf8 (
    const pathString & cstring ) [inline]
gg.h の 80 行目に定義があります。
```

#### 9.35.4.2 Utf8ToTChar()

```
pathString Utf8ToTChar (
    const std::string & string ) [inline]
gg.h の 79 行目に定義があります。
被呼び出し関係図:
```



## 9.36 gg.h

### [詳解]

```

00001 #pragma once
00002
00034
00035 // 標準ライブラリ
00036 #include <cmath>
00037 #include <array>
00038 #include <vector>
00039 #include <string>
00040 #include <memory>
00041 #include <cassert>
00042
00043 // Windows (Visual Studio) のとき
00044 #if defined(_MSC_VER)
00045 // 非推奨の警告を出さない
00046 # pragma warning(disable:4996)
00047 // 数学ライブラリの定数を使う
00048 # define _USE_MATH_DEFINES
00049 // MIN() / MAX マクロは使わない
00050 # define NOMINMAX
00051 // ファイルパスの文字コード
00052 #include <atlstr.h>
00053 using pathString = CString;
00054 using pathChar = wchar_t;
00055 extern pathString Utf8ToTChar(const std::string& string);
00056 extern std::string TCharToUtf8(const pathString& cstring);
00057 // デバッグビルドかどうか調べる
00058 # if defined(DEBUG)
00059 #   if !defined(DEBUG)
00060 #     define DEBUG
00061 #   endif
00062 #   define GLFW3_CONFIGURATION "Debug"
00063 # else
00064 #   if !defined(NDEBUG)
00065 #     define NDEBUG
00066 #   endif
00067 #   define GLFW3_CONFIGURATION "Release"
00068 # endif
00069 // プラットフォームを調べる
00070 # if defined(_WIN64)
00071 #   define GLFW3_PLATFORM "x64"
00072 # else
00073 #   define GLFW3_PLATFORM "Win32"
00074 # endif
00075 #else
00076 // ファイルパスの文字コード
00077 using pathString = std::string;
00078 using pathChar = char;
00079 inline pathString Utf8ToTChar(const std::string& string) { return string; }
00080 inline std::string TCharToUtf8(const pathString& cstring) { return cstring; }
00081 #endif
00082
00084
00085 // macOS で "OpenGL deprecated の警告を出さない
00086 #if defined(__APPLE__)
00087 # define GL_SILENCE_DEPRECATED
00088 #endif
00089
00090 // フレームワークに GLFW 3 を使う
00091 #if defined(IMGUI_IMPL_OPENGL_ES2)
00092 # define GLFW_INCLUDE_ES2
00093 #elif defined(IMGUI_IMPL_OPENGL_ES3)
00094 # define GLFW_INCLUDE_ES3
00095 #else
00096 # define GLFW_INCLUDE_GLCOREARB
00097 #endif
00098 #include <GLFW/glfw3.h>
00099
00100 // OpenGL 3.2 の API のエントリポイント
00101 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
00102 extern PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
00103 extern PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
00104 extern PFNGLACTIVETEXTUREPROC glActiveTexture;
00105 extern PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
00106 extern PFNGLATTACHSHADERPROC glAttachShader;
00107 extern PFNGLBEGINCNDITRNLRENDERNVPROC glBeginConditionalRenderNV;
00108 extern PFNGLBEGINCNDITRNLRENDERPROC glBeginConditionalRender;
00109 extern PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00110 extern PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00111 extern PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00112 extern PFNGLBEGINQUERYPROC glBeginQuery;
00113 extern PFNGLBEGINTTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
00114 extern PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;

```

```
00115 extern PFNGLBINDBUFFERBASEPROC glBindBufferBase;
00116 extern PFNGLBINDBUFFERPROC glBindBuffer;
00117 extern PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
00118 extern PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
00119 extern PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
00120 extern PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
00121 extern PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
00122 extern PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
00123 extern PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
00124 extern PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
00125 extern PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00126 extern PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00127 extern PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
00128 extern PFNGLBINDSAMPLERPROC glBindSampler;
00129 extern PFNGLBINDSAMPLERSPROC glBindSamplers;
00130 extern PFNGLBINDTEXTUREPROC glBindTexture;
00131 extern PFNGLBINDTEXTURESPROC glBindTextures;
00132 extern PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
00133 extern PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
00134 extern PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
00135 extern PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
00136 extern PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
00137 extern PFNGLBLENDBARRIERKHRPROC glBindBarrierKHR;
00138 extern PFNGLBLENDBARRIERNVPROC glBindBarrierNV;
00139 extern PFNGLBLENDCOLORPROC glBindColor;
00140 extern PFNGLBLENDEQUATIONIARBPROC glBindEquationiARB;
00141 extern PFNGLBLENDEQUATIONIPROC glBindEquationi;
00142 extern PFNGLBLENDEQUATIONPROC glBindEquation;
00143 extern PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00144 extern PFNGLBLENDEQUATIONSEPARATEIPROC glBindEquationSeparatei;
00145 extern PFNGLBLENDEQUATIONSEPARATEPROC glBindEquationSeparate;
00146 extern PFNGLBLENDFUNCIARBPROC glBindFunciARB;
00147 extern PFNGLBLENDFUNCIPROC glBindFunci;
00148 extern PFNGLBLENDFUNCPROC glBindFunc;
00149 extern PFNGLBLENDFUNCSEPARATEIARBPROC glBindFuncSeparateiARB;
00150 extern PFNGLBLENDFUNCSEPARATEIPROC glBindFuncSeparatei;
00151 extern PFNGLBLENDFUNCSEPARATEPROC glBindFuncSeparate;
00152 extern PFNGLBLENDPARAMETERINVPROC glBindParameteriNV;
00153 extern PFNGLBLITFRAMEBUFFERPROC glBlitFramebuffer;
00154 extern PFNGLBLITNAMEDFRAMEBUFFERPROC glBindNamedFramebuffer;
00155 extern PFNGLBUFFERADDRESSRANGEVNPROC glBindBufferAddressRangeNV;
00156 extern PFNGLBUFFERDATAPROC glBindBufferData;
00157 extern PFNGLBUFFERPAGECOMMITMENTARBPROC glBindBufferPageCommitmentARB;
00158 extern PFNGLBUFFERSTORAGEPROC glBindBufferStorage;
00159 extern PFNGLBUFFERSUBDATAPROC glBindBufferSubData;
00160 extern PFNGLCALLCOMMANDLISTNVPROC glCallCommandListNV;
00161 extern PFNGLCHECKFRAMEBUFFERSTATUSPROC glCheckFramebufferStatus;
00162 extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glCheckNamedFramebufferStatusEXT;
00163 extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glCheckNamedFramebufferStatus;
00164 extern PFNGLCLAMPCOLORPROC glClampColor;
00165 extern PFNGLCLEARBUFFERDATAPROC glClearBufferData;
00166 extern PFNGLCLEARBUFFERFIPROC glClearBufferfi;
00167 extern PFNGLCLEARBUFFERFVPROC glClearBufferfv;
00168 extern PFNGLCLEARBUFFERIVPROC glClearBufferiv;
00169 extern PFNGLCLEARBUFFERSUBDATAPROC glClearBufferSubData;
00170 extern PFNGLCLEARBUFFERUIVPROC glClearBufferuiv;
00171 extern PFNGLCLEARCOLORPROC glClearColor;
00172 extern PFNGLCLEARDEPTHFPROC glClearDepthf;
00173 extern PFNGLCLEARDEPTHPROC glClearDepth;
00174 extern PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
00175 extern PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
00176 extern PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
00177 extern PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
00178 extern PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
00179 extern PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
00180 extern PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
00181 extern PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferuiv;
00182 extern PFNGLCLEARPROC glClear;
00183 extern PFNGLCLEARSTENCILPROC glClearStencil;
00184 extern PFNGLCLEARTEXIMAGEPROC glClearTexImage;
00185 extern PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
00186 extern PFNGLCLIENTATTRIBDEFAULTTEXTPROC glClientAttribDefaultEXT;
00187 extern PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
00188 extern PFNGLCLIPCONTROLPROC glClipControl;
00189 extern PFNGLCOLORFORMATNVPROC glColorFormatNV;
00190 extern PFNGLCOLORMASKIPROC glColorMaski;
00191 extern PFNGLCOLORMASKPROC glColorMask;
00192 extern PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
00193 extern PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
00194 extern PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
00195 extern PFNGLCOMPILESHADERPROC glCompileShader;
00196 extern PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
00197 extern PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
00198 extern PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
00199 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
00200 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
00201 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
```

```

00202 extern PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
00203 extern PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
00204 extern PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
00205 extern PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
00206 extern PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
00207 extern PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
00208 extern PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
00209 extern PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
00210 extern PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
00211 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
00212 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00213 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00214 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00215 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00216 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00217 extern PFNGLCONSERVATERASTERPARAMETERFNVPROC glConservativeRasterParameterfnv;
00218 extern PFNGLCONSERVATIVERASTERPARAMETERINVPROC glConservativeRasterParameterinv;
00219 extern PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
00220 extern PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
00221 extern PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
00222 extern PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
00223 extern PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
00224 extern PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
00225 extern PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
00226 extern PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
00227 extern PFNGLCOPYPATHNVPROC glCopyPathnv;
00228 extern PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
00229 extern PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
00230 extern PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
00231 extern PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
00232 extern PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
00233 extern PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
00234 extern PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
00235 extern PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
00236 extern PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
00237 extern PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
00238 extern PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
00239 extern PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
00240 extern PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
00241 extern PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationnv;
00242 extern PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTablenv;
00243 extern PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancednv;
00244 extern PFNGLCOVERFILLPATHNVPROC glCoverFillPathnv;
00245 extern PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancednv;
00246 extern PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathnv;
00247 extern PFNGLCREATEBUFFERSPROC glCreateBuffers;
00248 extern PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
00249 extern PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
00250 extern PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
00251 extern PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
00252 extern PFNGLCREATEPROGRAMPROC glCreateProgram;
00253 extern PFNGLCREATEQUERIESPROC glCreateQueries;
00254 extern PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
00255 extern PFNGLCREATESAMPLERSPROC glCreateSamplers;
00256 extern PFNGLCREATESHADERPROC glCreateShader;
00257 extern PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
00258 extern PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
00259 extern PFNGLCREATESTATESNVPROC glCreateStatesNV;
00260 extern PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCleventARB;
00261 extern PFNGLCREATETEXTURESPROC glCreateTextures;
00262 extern PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
00263 extern PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
00264 extern PFNGLCULLFACEPROC glCullFace;
00265 extern PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
00266 extern PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
00267 extern PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
00268 extern PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
00269 extern PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
00270 extern PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
00271 extern PFNGLDELETEBUFFERSPROC glDeleteBuffers;
00272 extern PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
00273 extern PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
00274 extern PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
00275 extern PFNGLDELETEPATHSNVPROC glDeletePathsNV;
00276 extern PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
00277 extern PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
00278 extern PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
00279 extern PFNGLDELETEPROGRAMPROC glDeleteProgram;
00280 extern PFNGLDELETEQUERIESPROC glDeleteQueries;
00281 extern PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
00282 extern PFNGLDELETESAMPLERSPROC glDeleteSamplers;
00283 extern PFNGLDELETESHADERPROC glDeleteShader;
00284 extern PFNGLDELETESTATESNVPROC glDeleteStatesNV;
00285 extern PFNGLDELETESYNCPROC glDeleteSync;
00286 extern PFNGLDELETETEXTURESPROC glDeleteTextures;
00287 extern PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
00288 extern PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;

```

```

00289 extern PFNGLDEPTHFUNCPROC glDepthFunc;
00290 extern PFNGLDEPTHMASKPROC glDepthMask;
00291 extern PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
00292 extern PFNGLDEPTHRANGEFPROC glDepthRangef;
00293 extern PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
00294 extern PFNGLDEPTHRANGEPROC glDepthRange;
00295 extern PFNGLDETACHSHADERPROC glDetachShader;
00296 extern PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
00297 extern PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
00298 extern PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
00299 extern PFNGLDISABLEIPROC glDisablei;
00300 extern PFNGLDISABLEPROC glDisable;
00301 extern PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexAttribArrayAttribEXT;
00302 extern PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexAttribArrayAttrib;
00303 extern PFNGLDISABLEVERTEXARRAYATTRAYEXTPROC glDisableVertexAttribArrayEXT;
00304 extern PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArrayArray;
00305 extern PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
00306 extern PFNGLDISPATCHCOMPUTEINDIRECTPROC glDispatchComputeIndirect;
00307 extern PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
00308 extern PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
00309 extern PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
00310 extern PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
00311 extern PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
00312 extern PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
00313 extern PFNGLDRAWARRAYSPROC glDrawArrays;
00314 extern PFNGLDRAWBUFFERPROC glDrawBuffer;
00315 extern PFNGLDRAWBUFFERSPROC glDrawBuffers;
00316 extern PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
00317 extern PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
00318 extern PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
00319 extern PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
00320 extern PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
00321 extern PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
00322 extern PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
00323 extern PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
00324 extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC
    glDrawElementsInstancedBaseVertexBaseInstance;
00325 extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
00326 extern PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
00327 extern PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
00328 extern PFNGLDRAWELEMENTSPROC glDrawElements;
00329 extern PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00330 extern PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00331 extern PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
00332 extern PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
00333 extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
00334 extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00335 extern PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00336 extern PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00337 extern PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00338 extern PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00339 extern PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00340 extern PFNGLENABLEIPROC glEnablei;
00341 extern PFNGLENABLEPROC glEnable;
00342 extern PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexAttribArrayAttribEXT;
00343 extern PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00344 extern PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00345 extern PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArrayArray;
00346 extern PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
00347 extern PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
00348 extern PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
00349 extern PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
00350 extern PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
00351 extern PFNGLENDQUERYPROC glEndQuery;
00352 extern PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
00353 extern PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
00354 extern PFNGLFENCESYNCNPROC glFenceSync;
00355 extern PFNGLFINISHPROC glFinish;
00356 extern PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00357 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00358 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
00359 extern PFNGLFLUSHPROC glFlush;
00360 extern PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00361 extern PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00362 extern PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00363 extern PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00364 extern PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00365 extern PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00366 extern PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
00367 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSfvARBPROC glFramebufferSampleLocationsfvARB;
00368 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSfvNVPROC glFramebufferSampleLocationsfvNV;
00369 extern PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00370 extern PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00371 extern PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00372 extern PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
00373 extern PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00374 extern PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;

```

```

00375 extern PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00376 extern PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00377 extern PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
00378 extern PFNGLFRONTPFACEPROC glFrontFace;
00379 extern PFNGLGENBUFFERSPROC glGenBuffers;
00380 extern PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
00381 extern PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
00382 extern PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
00383 extern PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
00384 extern PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
00385 extern PFNGLGENPATHSNVPROC glGenPathsNV;
00386 extern PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
00387 extern PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
00388 extern PFNGLGENQUERIESPROC glGenQueries;
00389 extern PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
00390 extern PFNGLGENSAMPLERSPROC glGenSamplers;
00391 extern PFNGLGENTEXTURESPROC glGenTextures;
00392 extern PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
00393 extern PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
00394 extern PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
00395 extern PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
00396 extern PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
00397 extern PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
00398 extern PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
00399 extern PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
00400 extern PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
00401 extern PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
00402 extern PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
00403 extern PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
00404 extern PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
00405 extern PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
00406 extern PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
00407 extern PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
00408 extern PFNGLGETBOOLEANVPROC glGetBooleanv;
00409 extern PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00410 extern PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00411 extern PFNGLGETBUFFERPARAMETERUI64VNVPROC glGetBufferParameterui64vNV;
00412 extern PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00413 extern PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00414 extern PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00415 extern PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
00416 extern PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
00417 extern PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
00418 extern PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00419 extern PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00420 extern PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00421 extern PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00422 extern PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00423 extern PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00424 extern PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
00425 extern PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00426 extern PFNGLGETDOUBLEVPROC glGetDoublev;
00427 extern PFNGLGETERRORORPROC glGetError;
00428 extern PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
00429 extern PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
00430 extern PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
00431 extern PFNGLGETFLOATI_VPROC glGetFloati_v;
00432 extern PFNGLGETFLOATVPROC glGetFloatv;
00433 extern PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
00434 extern PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
00435 extern PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
00436 extern PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
00437 extern PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
00438 extern PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
00439 extern PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
00440 extern PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
00441 extern PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
00442 extern PFNGLGETINTEGER64IVPROC glGetInteger64i_v;
00443 extern PFNGLGETINTEGER64VPROC glGetInteger64v;
00444 extern PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00445 extern PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00446 extern PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00447 extern PFNGLGETINTEGERUI64VNVPROC glGetIntegerui64vNV;
00448 extern PFNGLGETINTEGERVPROC glGetIntegerv;
00449 extern PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
00450 extern PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
00451 extern PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
00452 extern PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
00453 extern PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
00454 extern PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
00455 extern PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGenfvEXT;
00456 extern PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
00457 extern PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
00458 extern PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
00459 extern PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
00460 extern PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
00461 extern PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;

```

```
00462 extern PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIivEXT;
00463 extern PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuiVEXT;
00464 extern PFNGLGETMULTITEXPARAMETERIVEXTPROC glGetMultiTexParameterivEXT;
00465 extern PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
00466 extern PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
00467 extern PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
00468 extern PFNGLGETNAMEDBUFFERPARAMETERUI64NVPROC glGetNamedBufferParameterui64vNV;
00469 extern PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
00470 extern PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
00471 extern PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
00472 extern PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
00473 extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC
    glGetNamedFramebufferAttachmentParameterivEXT;
00474 extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
00475 extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
00476 extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
00477 extern PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
00478 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00479 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00480 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIivEXT;
00481 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERUIVEXTPROC glGetNamedProgramLocalParameterUiVEXT;
00482 extern PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00483 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00484 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00485 extern PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00486 extern PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00487 extern PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetCompressedTexImageARB;
00488 extern PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
00489 extern PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00490 extern PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00491 extern PFNGLGETNTEXIMAGEPROC glGetnTexImage;
00492 extern PFNGLGETNUNIFORMDVARBPROC glGetnUniformdARB;
00493 extern PFNGLGETNUNIFORMDMDVARBPROC glGetnUniformmdARB;
00494 extern PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
00495 extern PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00496 extern PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00497 extern PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
00498 extern PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
00499 extern PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00500 extern PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
00501 extern PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
00502 extern PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
00503 extern PFNGLGETOBJECTLABELPROC glGetObjectLabel;
00504 extern PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
00505 extern PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
00506 extern PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
00507 extern PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
00508 extern PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
00509 extern PFNGLGETPATHMETRICRANGEENVPROC glGetPathMetricRangeEnv;
00510 extern PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
00511 extern PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
00512 extern PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
00513 extern PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
00514 extern PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
00515 extern PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
00516 extern PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
00517 extern PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
00518 extern PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
00519 extern PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
00520 extern PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
00521 extern PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
00522 extern PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
00523 extern PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
00524 extern PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
00525 extern PFNGLGETPOINTERIIVEXTPROC glGetPointeriivEXT;
00526 extern PFNGLGETPOINTERVERPROC glGetPointerv;
00527 extern PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
00528 extern PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
00529 extern PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
00530 extern PFNGLGETPROGRAMIVPROC glGetProgramiv;
00531 extern PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
00532 extern PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
00533 extern PFNGLGETPROGRAMRESOURCEFVNVPROC glGetProgramResourcefvNV;
00534 extern PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00535 extern PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00536 extern PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00537 extern PFNGLGETPROGRAMRESOURCELOCATIONPROC glGetProgramResourceLocation;
00538 extern PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00539 extern PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00540 extern PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
00541 extern PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00542 extern PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00543 extern PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00544 extern PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00545 extern PFNGLGETQUERYIVPROC glGetQueryiv;
00546 extern PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
00547 extern PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
```

```

00548 extern PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00549 extern PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00550 extern PFNGLGETRENDERBUFFERPARAMETERIIVPROC glGetRenderbufferParameteriv;
00551 extern PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00552 extern PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00553 extern PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
00554 extern PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameteriv;
00555 extern PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
00556 extern PFNGLGETSHADERIVPROC glGetShaderiv;
00557 extern PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
00558 extern PFNGLGETSHADERSOURCEPROC glGetShaderSource;
00559 extern PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
00560 extern PFNGLGETSTRINGIPROC glGetStringi;
00561 extern PFNGLGETSTRINGPROC glGetString;
00562 extern PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
00563 extern PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
00564 extern PFNGLGETSYNCIVPROC glGetSynciv;
00565 extern PFNGLGETTEXIMAGEPROC glGetTexImage;
00566 extern PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00567 extern PFNGLGETTEXLEVELPARAMETERIIVPROC glGetTexLevelParameteriv;
00568 extern PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00569 extern PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00570 extern PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00571 extern PFNGLGETTEXPARAMETERIIVPROC glGetTexParameteriv;
00572 extern PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00573 extern PFNGLGETTEXTUREHANDLEENVPROC glGetTextureHandleNV;
00574 extern PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00575 extern PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00576 extern PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00577 extern PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
00578 extern PFNGLGETTEXTURELEVELPARAMETERIIVEXTPROC glGetTextureLevelParameterivEXT;
00579 extern PFNGLGETTEXTURELEVELPARAMETERIIVPROC glGetTextureLevelParameteriv;
00580 extern PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
00581 extern PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
00582 extern PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
00583 extern PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
00584 extern PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
00585 extern PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
00586 extern PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterivEXT;
00587 extern PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameteriv;
00588 extern PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00589 extern PFNGLGETTEXTURESAMPLERHANDLEENVPROC glGetTextureSamplerHandleNV;
00590 extern PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00591 extern PFNGLGETTRANSFORMFEEDBACKI64VPROC glGetTransformFeedbacki64_v;
00592 extern PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00593 extern PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
00594 extern PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00595 extern PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00596 extern PFNGLGETUNIFORMDVMPROC glGetUniformDv;
00597 extern PFNGLGETUNIFORMFVPROC glGetUniformfv;
00598 extern PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00599 extern PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
00600 extern PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
00601 extern PFNGLGETUNIFORMIVPROC glGetUniformiv;
00602 extern PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
00603 extern PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
00604 extern PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
00605 extern PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
00606 extern PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
00607 extern PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00608 extern PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
00609 extern PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00610 extern PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00611 extern PFNGLGETVERTEXARRAYVPROC glGetVertexArrayiv;
00612 extern PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00613 extern PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00614 extern PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribArrayAttribdv;
00615 extern PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribArrayAttribfv;
00616 extern PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribArrayAttribIiv;
00617 extern PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribArrayAttribIuiv;
00618 extern PFNGLGETVERTEXATTRIBVPROC glGetVertexAttribArrayAttribiv;
00619 extern PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribArrayAttribLdv;
00620 extern PFNGLGETVERTEXATTRIBLII64VNVPROC glGetVertexAttribArrayAttribLi64vNV;
00621 extern PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribArrayAttribLui64vARB;
00622 extern PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribArrayAttribLui64vNV;
00623 extern PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribArrayAttribPointerv;
00624 extern PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrNV;
00625 extern PFNGLHINTPROC glHint;
00626 extern PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00627 extern PFNGLINSERTEVENTMARKEREXTPROC glInsertEventMarkerEXT;
00628 extern PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00629 extern PFNGLINVALIDATEDATABUFFERPROC glInvalidateBufferData;
00630 extern PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferDataSubData;
00631 extern PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFrameBuffer;
00632 extern PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFrameBufferData;
00633 extern PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFrameBufferDataSubData;
00634 extern PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFrameBuffer;

```

```

00635 extern PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00636 extern PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00637 extern PFNGLISBUFFERPROC glIsBuffer;
00638 extern PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
00639 extern PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
00640 extern PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00641 extern PFNGLISENABLEDIPROC glIsEnabledi;
00642 extern PFNGLISENABLEDPROC glIsEnabled;
00643 extern PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00644 extern PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
00645 extern PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
00646 extern PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00647 extern PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00648 extern PFNGLISPATHNVPROC glIsPathNV;
00649 extern PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00650 extern PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00651 extern PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00652 extern PFNGLISPROGRAMPROC glIsProgram;
00653 extern PFNGLISQUERYPROC glIsQuery;
00654 extern PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00655 extern PFNGLISSAMPLERPROC glIsSampler;
00656 extern PFNGLISSHADERPROC glIsShader;
00657 extern PFNGLISSTATENVPROC glIsStateNV;
00658 extern PFNGLISSYNCPROC glIsSync;
00659 extern PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00660 extern PFNGLISTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00661 extern PFNGLISTEXTUREREPROC glIsTexture;
00662 extern PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00663 extern PFNGLISVERTEXARRAYPROC glIsVertexArray;
00664 extern PFNGLLABELOBJECTTEXTPROC glLabelObjectEXT;
00665 extern PFNGLLINKPROGRAMPROC glLinkProgram;
00667 extern PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
00668 extern PFNGLLOGICOPPROC glLogicOp;
00669 extern PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
00670 extern PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
00671 extern PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
00672 extern PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
00673 extern PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
00674 extern PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
00675 extern PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
00676 extern PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
00677 extern PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
00678 extern PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
00679 extern PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
00680 extern PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
00681 extern PFNGLMAPBUFFERPROC glMapBuffer;
00682 extern PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
00683 extern PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
00684 extern PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
00685 extern PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
00686 extern PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
00687 extern PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00688 extern PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fnv;
00689 extern PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fnv;
00690 extern PFNGLMATRIXLOADDEXTPROC glMatrixLoaddext;
00691 extern PFNGLMATRIXLOADFEXTPROC glMatrixLoadfext;
00692 extern PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00693 extern PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fnv;
00694 extern PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedext;
00695 extern PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefext;
00696 extern PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fnv;
00697 extern PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fnv;
00698 extern PFNGLMATRIXMULTDEXTPROC glMatrixMultdext;
00699 extern PFNGLMATRIXMULTFEXTPROC glMatrixMultfext;
00700 extern PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fnv;
00701 extern PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedext;
00702 extern PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefext;
00703 extern PFNGLMATRIXORTHOEXTPROC glMatrixOrthoext;
00704 extern PFNGLMATRIXPOPEXTPROC glMatrixPopext;
00705 extern PFNGLMATRIXPUSHEXTPROC glMatrixPushext;
00706 extern PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedext;
00707 extern PFNGLMATRIXTRANSLATEFEXTPROC glMatrixRotatefext;
00708 extern PFNGLMATRIXSCALEDEXTPROC glMatrixScaledext;
00709 extern PFNGLMATRIXSCALEFEXTPROC glMatrixScalefext;
00710 extern PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedext;
00711 extern PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefext;
00712 extern PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
00713 extern PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
00714 extern PFNGLMEMORYBARRIERPROC glMemoryBarrier;
00715 extern PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
00716 extern PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
00717 extern PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
00718 extern PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
00719 extern PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC glMultiDrawArraysIndirectCountARB;
00720 extern PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00721 extern PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays

```

```

00722 extern PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00723 extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00724 extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00725 extern PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC glMultiDrawElementsIndirectCountARB;
00726 extern PFNGLMULTIDRAWELEMENTSINDIRECTCPROC glMultiDrawElementsIndirect;
00727 extern PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00728 extern PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00729 extern PFNGLMULTITEXCOORDPOINTEREXTPROC glMultiTexCoordPointerEXT;
00730 extern PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00731 extern PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00732 extern PFNGLMULTITEXENVIEXTPROC glMultiTexEnviEXT;
00733 extern PFNGLMULTITEXENVIVEXTPROC glMultiTexEnvivEXT;
00734 extern PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00735 extern PFNGLMULTITEXGENDVEXTPROC glMultiTexGendvEXT;
00736 extern PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
00737 extern PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
00738 extern PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
00739 extern PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
00740 extern PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
00741 extern PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
00742 extern PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
00743 extern PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
00744 extern PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
00745 extern PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameteriEXT;
00746 extern PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
00747 extern PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
00748 extern PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
00749 extern PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
00750 extern PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
00751 extern PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
00752 extern PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
00753 extern PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
00754 extern PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
00755 extern PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
00756 extern PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
00757 extern PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
00758 extern PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
00759 extern PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubDataEXT;
00760 extern PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
00761 extern PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
00762 extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
00763 extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
00764 extern PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
00765 extern PFNGLNAMEDFRAMEBUFFERPARAMETERIIPROC glNamedFramebufferParameteri;
00766 extern PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
00767 extern PFNGLNAMEDFRAMEBUFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00768 extern PFNGLNAMEDFRAMEBUFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00769 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glNamedFramebufferSampleLocationsfvARB;
00770 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
00771 extern PFNGLNAMEDFRAMEBUFTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00772 extern PFNGLNAMEDFRAMEBUFTTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00773 extern PFNGLNAMEDFRAMEBUFTTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00774 extern PFNGLNAMEDFRAMEBUFTTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00775 extern PFNGLNAMEDFRAMEBUFTTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00776 extern PFNGLNAMEDFRAMEBUFTTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00777 extern PFNGLNAMEDFRAMEBUFTTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00778 extern PFNGLNAMEDFRAMEBUFTTEXTUREPROC glNamedFramebufferTexture;
00779 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00780 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00781 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
00782 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00783 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
00784 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00785 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00786 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uvEXT;
00787 extern PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00788 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
00789 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uvEXT;
00790 extern PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00791 extern PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00792 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00793 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00794 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00795 extern PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00796 extern PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
00797 extern PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00798 extern PFNGLOBJECTLABELPROC glObjectLabel;
00799 extern PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
00800 extern PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
00801 extern PFNGLPATCHPARAMETERIIPROC glPatchParameteri;
00802 extern PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00803 extern PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00804 extern PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00805 extern PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00806 extern PFNGLPATHGLYPHINDEXARRAYNVPROC glPathGlyphIndexArrayNV;
00807 extern PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;

```

```
00808 extern PFNGLPATHGLYPHRANGENVPROC glPathGlyphRangeNV;
00809 extern PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
00810 extern PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
00811 extern PFNGLPATHPARAMETERFNVPROC glPathParameterfvNV;
00812 extern PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00813 extern PFNGLPATHPARAMETERINVPROC glPathParameterivNV;
00814 extern PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
00815 extern PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
00816 extern PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00817 extern PFNGLPATHSTRINGNVPROC glPathStringNV;
00818 extern PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00819 extern PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00820 extern PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00821 extern PFNGLPIXELSTOREREFPROC glPixelStoref;
00822 extern PFNGLPIXELSTOREIPROC glPixelStorei;
00823 extern PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
00824 extern PFNGLPOINTPARAMETERFPROC glPointParameterf;
00825 extern PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
00826 extern PFNGLPOINTPARAMETERIPROC glPointParameteri;
00827 extern PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00828 extern PFNGLPOINTSIZESPROC glPointSize;
00829 extern PFNGLPOLYGONMODEPROC glPolygonMode;
00830 extern PFNGLPOLYGOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00831 extern PFNGLPOLYGOFFSETPROC glPolygonOffset;
00832 extern PFNGLPOPDEBUGGROUPPROC glPopDebugGroup;
00833 extern PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00834 extern PFNGLPRIMITIVEBOUNDBOXARBPROC glPrimitiveBoundingBoxARB;
00835 extern PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
00836 extern PFNGLPROGRAMBINARYPROC glProgramBinary;
00837 extern PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
00838 extern PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
00839 extern PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
00840 extern PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
00841 extern PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
00842 extern PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
00843 extern PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dv;
00844 extern PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
00845 extern PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
00846 extern PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
00847 extern PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
00848 extern PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
00849 extern PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;
00850 extern PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
00851 extern PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
00852 extern PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniform1iEXT;
00853 extern PFNGLPROGRAMUNIFORM1IPROC glProgramUniform1i;
00854 extern PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
00855 extern PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
00856 extern PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
00857 extern PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniform1ui64NV;
00858 extern PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
00859 extern PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
00860 extern PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
00861 extern PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
00862 extern PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
00863 extern PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
00864 extern PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
00865 extern PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
00866 extern PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
00867 extern PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
00868 extern PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
00869 extern PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
00870 extern PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
00871 extern PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
00872 extern PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
00873 extern PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
00874 extern PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
00875 extern PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
00876 extern PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
00877 extern PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
00878 extern PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
00879 extern PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
00880 extern PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
00881 extern PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
00882 extern PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
00883 extern PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
00884 extern PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
00885 extern PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
00886 extern PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
00887 extern PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
00888 extern PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
00889 extern PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
00890 extern PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
00891 extern PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
00892 extern PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
00893 extern PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
00894 extern PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
```

```

00895 extern PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
00896 extern PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
00897 extern PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
00898 extern PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
00899 extern PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
00900 extern PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
00901 extern PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
00902 extern PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
00903 extern PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
00904 extern PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
00905 extern PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
00906 extern PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
00907 extern PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
00908 extern PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
00909 extern PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
00910 extern PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
00911 extern PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
00912 extern PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
00913 extern PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
00914 extern PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
00915 extern PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
00916 extern PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
00917 extern PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
00918 extern PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
00919 extern PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
00920 extern PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
00921 extern PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
00922 extern PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
00923 extern PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
00924 extern PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
00925 extern PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
00926 extern PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
00927 extern PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
00928 extern PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
00929 extern PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
00930 extern PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
00931 extern PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
00932 extern PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
00933 extern PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
00934 extern PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
00935 extern PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
00936 extern PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
00937 extern PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
00938 extern PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
00939 extern PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
00940 extern PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dvEXT;
00941 extern PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
00942 extern PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
00943 extern PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
00944 extern PFNGLPROGRAMUNIFORMMATRIX2X2DVEXTPROC glProgramUniformMatrix2x2dvEXT;
00945 extern PFNGLPROGRAMUNIFORMMATRIX2X2DVPROC glProgramUniformMatrix2x2dv;
00946 extern PFNGLPROGRAMUNIFORMMATRIX2X2FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00947 extern PFNGLPROGRAMUNIFORMMATRIX2X2FVPROC glProgramUniformMatrix2x3fv;
00948 extern PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
00949 extern PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
00950 extern PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
00951 extern PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
00952 extern PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
00953 extern PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dv;
00954 extern PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
00955 extern PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
00956 extern PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
00957 extern PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dv;
00958 extern PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
00959 extern PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
00960 extern PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
00961 extern PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
00962 extern PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
00963 extern PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
00964 extern PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dvEXT;
00965 extern PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dv;
00966 extern PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
00967 extern PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
00968 extern PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
00969 extern PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dv;
00970 extern PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
00971 extern PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
00972 extern PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
00973 extern PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dv;
00974 extern PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
00975 extern PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
00976 extern PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
00977 extern PFNGLPROGRAMUNIFORMUI64VNVPROC glProgramUniformui64vNV;
00978 extern PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
00979 extern PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC glPushClientAttribDefaultEXT;
00980 extern PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
00981 extern PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;

```

```
00982 extern PFNGLQUERYCOUNTERPROC glQueryCounter;
00983 extern PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
00984 extern PFNGLREADBUFFERPROC glReadBuffer;
00985 extern PFNGLREADNPixelsARBPROC glReadnPixelsARB;
00986 extern PFNGLREADNPixelsPROC glReadnPixels;
00987 extern PFNGLREADPIXELSPROC glReadPixels;
00988 extern PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
00989 extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
00990 extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
00991 extern PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
00992 extern PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
00993 extern PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
00994 extern PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
00995 extern PFNGLSAMPLEMASKIPROC glSampleMaski;
00996 extern PFNGLSAMPLERPARAMETERRFPROC glSamplerParameterf;
00997 extern PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
00998 extern PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIiv;
00999 extern PFNGLSAMPLERPARAMETERIPROC glSamplerParameteri;
01000 extern PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
01001 extern PFNGLSAMPLERPARAMETERIVPROC glSamplerParameteriv;
01002 extern PFNGLSCISSORARRAYVPROC glScissorArrayv;
01003 extern PFNGLSCISSORINDEXEDPROC glScissorIndexed;
01004 extern PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
01005 extern PFNGLSCISSORPROC glScissor;
01006 extern PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
01007 extern PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
01008 extern PFNGLSHADERBINARYPROC glShaderBinary;
01009 extern PFNGLSHADERSOURCEPROC glShaderSource;
01010 extern PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
01011 extern PFNGLSIGNALKVFENCENVPROC glSignalVkFenceNV;
01012 extern PFNGLSIGNALKSEMAPHORENVPROC glSignalVkSemaphoreNV;
01013 extern PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
01014 extern PFNGLSTATECAPTURENVPROC glStateCaptureNV;
01015 extern PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
01016 extern PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
01017 extern PFNGLSTENCILFUNCPROC glStencilFunc;
01018 extern PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
01019 extern PFNGLSTENCILMASKPROC glStencilMask;
01020 extern PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
01021 extern PFNGLSTENCILLOPPROC glStencilOp;
01022 extern PFNGLSTENCILLOPSEPARATEPROC glStencilOpSeparate;
01023 extern PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
01024 extern PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
01025 extern PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01026 extern PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01027 extern PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01028 extern PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
01029 extern PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
01030 extern PFNGLTEXBUFFERARBPROC glTexBufferARB;
01031 extern PFNGLTEXBUFFERPROC glTexBuffer;
01032 extern PFNGLTEXBUFFERRANGEPROC glTexBufferSize;
01033 extern PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
01034 extern PFNGLTEXIMAGE1DPROC glTexImage1D;
01035 extern PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
01036 extern PFNGLTEXIMAGE2DPROC glTexImage2D;
01037 extern PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
01038 extern PFNGLTEXIMAGE3DPROC glTexImage3D;
01039 extern PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
01040 extern PFNGLTEXPARAMETERFPROC glTexParameterf;
01041 extern PFNGLTEXPARAMETERFVPROC glTexParameterfv;
01042 extern PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
01043 extern PFNGLTEXPARAMETERIIPROC glTexParameterIi;
01044 extern PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01045 extern PFNGLTEXPARAMETERIVPROC glTexParameteriv;
01046 extern PFNGLTEXTSTORAGE1DPROC glTexStorage1D;
01047 extern PFNGLTEXTSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
01048 extern PFNGLTEXTSTORAGE2DPROC glTexStorage2D;
01049 extern PFNGLTEXTSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
01050 extern PFNGLTEXTSTORAGE3DPROC glTexStorage3D;
01051 extern PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01052 extern PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01053 extern PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01054 extern PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
01055 extern PFNGLTEXTUREBARRIERPROC glTextureBarrier;
01056 extern PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
01057 extern PFNGLTEXTUREBUFFERPROC glTextureBuffer;
01058 extern PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01059 extern PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01060 extern PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
01061 extern PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
01062 extern PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
01063 extern PFNGLTEXTUREPAGECOMMITMENTEXTPROC glTexturePageCommitmentEXT;
01064 extern PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
01065 extern PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
01066 extern PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
01067 extern PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
01068 extern PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
```

```
01069 extern PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIiivEXT;
01070 extern PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiiv;
01071 extern PFNGLTEXTUREPARAMETERIIPROC glTextureParameterIi;
01072 extern PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01073 extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01074 extern PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterIiivEXT;
01075 extern PFNGLTEXTUREPARAMETERIVPROC glTextureParameterIiiv;
01076 extern PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
01077 extern PFNGLTEXTURERESTORAGE1DEXTPROC glTextureStorage1DEXT;
01078 extern PFNGLTEXTURERESTORAGE1DPROC glTextureStorage1D;
01079 extern PFNGLTEXTURERESTORAGE2DEXTPROC glTextureStorage2DEXT;
01080 extern PFNGLTEXTURERESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
01081 extern PFNGLTEXTURERESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
01082 extern PFNGLTEXTURERESTORAGE2DPROC glTextureStorage2D;
01083 extern PFNGLTEXTURERESTORAGE3DEXTPROC glTextureStorage3DEXT;
01084 extern PFNGLTEXTURERESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
01085 extern PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
01086 extern PFNGLTEXTURERESTORAGE3DPROC glTextureStorage3D;
01087 extern PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
01088 extern PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
01089 extern PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
01090 extern PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
01091 extern PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
01092 extern PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
01093 extern PFNGLTEXTUREVIEWPROC glTextureView;
01094 extern PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
01095 extern PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
01096 extern PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
01097 extern PFNGLTRANSFORMPATHNVPROC gltransformPathNV;
01098 extern PFNGLUNIFORM1DPROC glUniform1d;
01099 extern PFNGLUNIFORM1DVPROC glUniform1dv;
01100 extern PFNGLUNIFORM1FPROC glUniform1f;
01101 extern PFNGLUNIFORM1FVPROC glUniform1fv;
01102 extern PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
01103 extern PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
01104 extern PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
01105 extern PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
01106 extern PFNGLUNIFORM1IPROC glUniform1i;
01107 extern PFNGLUNIFORM1IVPROC glUniform1iv;
01108 extern PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01109 extern PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01110 extern PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
01111 extern PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
01112 extern PFNGLUNIFORM1UIPROC glUniform1ui;
01113 extern PFNGLUNIFORM1UIVPROC glUniform1uiv;
01114 extern PFNGLUNIFORM2DPROC glUniform2d;
01115 extern PFNGLUNIFORM2DVPROC glUniform2dv;
01116 extern PFNGLUNIFORM2FPROC glUniform2f;
01117 extern PFNGLUNIFORM2FVPROC glUniform2fv;
01118 extern PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
01119 extern PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
01120 extern PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
01121 extern PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
01122 extern PFNGLUNIFORM2IPROC glUniform2i;
01123 extern PFNGLUNIFORM2IVPROC glUniform2iv;
01124 extern PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01125 extern PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01126 extern PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
01127 extern PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
01128 extern PFNGLUNIFORM2UIPROC glUniform2ui;
01129 extern PFNGLUNIFORM2UIVPROC glUniform2uiv;
01130 extern PFNGLUNIFORM3DPROC glUniform3d;
01131 extern PFNGLUNIFORM3DVPROC glUniform3dv;
01132 extern PFNGLUNIFORM3FPROC glUniform3f;
01133 extern PFNGLUNIFORM3FVPROC glUniform3fv;
01134 extern PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
01135 extern PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
01136 extern PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
01137 extern PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
01138 extern PFNGLUNIFORM3IPROC glUniform3i;
01139 extern PFNGLUNIFORM3IVPROC glUniform3iv;
01140 extern PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01141 extern PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01142 extern PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
01143 extern PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
01144 extern PFNGLUNIFORM3UIPROC glUniform3ui;
01145 extern PFNGLUNIFORM3UIVPROC glUniform3uiv;
01146 extern PFNGLUNIFORM4DPROC glUniform4d;
01147 extern PFNGLUNIFORM4DVPROC glUniform4dv;
01148 extern PFNGLUNIFORM4FPROC glUniform4f;
01149 extern PFNGLUNIFORM4FVPROC glUniform4fv;
01150 extern PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
01151 extern PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
01152 extern PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
01153 extern PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
01154 extern PFNGLUNIFORM4IPROC glUniform4i;
01155 extern PFNGLUNIFORM4IVPROC glUniform4iv;
```

```

01156 extern PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01157 extern PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01158 extern PFNGLUNIFORM4UI64VARBPROC glUniform4ui64VARB;
01159 extern PFNGLUNIFORM4UI64VNVPROC glUniform4ui64VN;
01160 extern PFNGLUNIFORM4UIPROC glUniform4ui;
01161 extern PFNGLUNIFORM4UIVPROC glUniform4uiv;
01162 extern PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01163 extern PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01164 extern PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01165 extern PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64VARB;
01166 extern PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64VN;
01167 extern PFNGLUNIFORMMATRIX2DVPROC glUniformMatrix2dv;
01168 extern PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
01169 extern PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
01170 extern PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
01171 extern PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
01172 extern PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
01173 extern PFNGLUNIFORMMATRIX3DVPROC glUniformMatrix3dv;
01174 extern PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
01175 extern PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
01176 extern PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
01177 extern PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
01178 extern PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
01179 extern PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
01180 extern PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
01181 extern PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
01182 extern PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
01183 extern PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
01184 extern PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
01185 extern PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
01186 extern PFNGLUNIFORMUI64NVPROC glUniformui64NV;
01187 extern PFNGLUNIFORMUI64VNVPROC glUniformui64NV;
01188 extern PFNGLUNMAPBUFFERPROC glUnmapBuffer;
01189 extern PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
01190 extern PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
01191 extern PFNGLUSEPROGRAMPROC glUseProgram;
01192 extern PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
01193 extern PFNGLUSESHEADERPROGRAMEXTPROC glUseShaderProgramEXT;
01194 extern PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
01195 extern PFNGLVALIDATEPROGRAMPROC glValidateProgram;
01196 extern PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
01197 extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
01198 extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribIFormat;
01199 extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribLFormat;
01200 extern PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01201 extern PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01202 extern PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
01203 extern PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01204 extern PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01205 extern PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01206 extern PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
01207 extern PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01208 extern PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
01209 extern PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01210 extern PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
01211 extern PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribArrayAttribBindingEXT;
01212 extern PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribArrayAttribDivisorEXT;
01213 extern PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribArrayAttribFormatEXT;
01214 extern PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribArrayAttribIFormatEXT;
01215 extern PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribArrayAttribIOffsetEXT;
01216 extern PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribArrayAttribLFormatEXT;
01217 extern PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribArrayAttribLOffsetEXT;
01218 extern PFNGLVERTEXARRAYVERTEXATTRIBOFSETEXTPROC glVertexArrayVertexAttribArrayAttribOffsetEXT;
01219 extern PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexAttribArrayBindingDivisorEXT;
01220 extern PFNGLVERTEXARRAYVERTEXBUFERPROC glVertexArrayVertexBuffer;
01221 extern PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01222 extern PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
01223 extern PFNGLVERTEXATTRIBIDPROC glVertexArrayAttribId;
01224 extern PFNGLVERTEXATTRIB1DPROC glVertexArrayAttrib1d;
01225 extern PFNGLVERTEXATTRIB1FPROC glVertexArrayAttrib1f;
01226 extern PFNGLVERTEXATTRIB1FVPROC glVertexArrayAttrib1fv;
01227 extern PFNGLVERTEXATTRIB1ISPROC glVertexArrayAttrib1s;
01228 extern PFNGLVERTEXATTRIB1SVPROC glVertexArrayAttrib1sv;
01229 extern PFNGLVERTEXATTRIB2DPROC glVertexArrayAttrib2d;
01230 extern PFNGLVERTEXATTRIB2DVPROC glVertexArrayAttrib2dv;
01231 extern PFNGLVERTEXATTRIB2FPROC glVertexArrayAttrib2f;
01232 extern PFNGLVERTEXATTRIB2FVPROC glVertexArrayAttrib2fv;
01233 extern PFNGLVERTEXATTRIB2SPROC glVertexArrayAttrib2s;
01234 extern PFNGLVERTEXATTRIB2SVPROC glVertexArrayAttrib2sv;
01235 extern PFNGLVERTEXATTRIB3DPROC glVertexArrayAttrib3d;
01236 extern PFNGLVERTEXATTRIB3DVPROC glVertexArrayAttrib3dv;
01237 extern PFNGLVERTEXATTRIB3FPROC glVertexArrayAttrib3f;
01238 extern PFNGLVERTEXATTRIB3FVPROC glVertexArrayAttrib3fv;
01239 extern PFNGLVERTEXATTRIB3SPROC glVertexArrayAttrib3s;
01240 extern PFNGLVERTEXATTRIB3SVPROC glVertexArrayAttrib3sv;
01241 extern PFNGLVERTEXATTRIB4BVPROC glVertexArrayAttrib4bv;
01242 extern PFNGLVERTEXATTRIB4DPROC glVertexArrayAttrib4d;

```

```

01243 extern PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
01244 extern PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
01245 extern PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
01246 extern PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
01247 extern PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4Nbv;
01248 extern PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4Niv;
01249 extern PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4Nsv;
01250 extern PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4Nub;
01251 extern PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4Nubv;
01252 extern PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4Nuiv;
01253 extern PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4Nusv;
01254 extern PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
01255 extern PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
01256 extern PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
01257 extern PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
01258 extern PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
01259 extern PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01260 extern PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01261 extern PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01262 extern PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01263 extern PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01264 extern PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01265 extern PFNGLVERTEXATTRIBI1IIVPROC glVertexAttribI1iv;
01266 extern PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
01267 extern PFNGLVERTEXATTRIBI1UIIVPROC glVertexAttribI1uiv;
01268 extern PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01269 extern PFNGLVERTEXATTRIBI2IIVPROC glVertexAttribI2iv;
01270 extern PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
01271 extern PFNGLVERTEXATTRIBI2UIIVPROC glVertexAttribI2uiv;
01272 extern PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
01273 extern PFNGLVERTEXATTRIBI3IIVPROC glVertexAttribI3iv;
01274 extern PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01275 extern PFNGLVERTEXATTRIBI3UIIVPROC glVertexAttribI3uiv;
01276 extern PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01277 extern PFNGLVERTEXATTRIBI4IIPROC glVertexAttribI4i;
01278 extern PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
01279 extern PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
01280 extern PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
01281 extern PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
01282 extern PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
01283 extern PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
01284 extern PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
01285 extern PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
01286 extern PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
01287 extern PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
01288 extern PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
01289 extern PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
01290 extern PFNGLVERTEXATTRIBL1I64VNVPROC glVertexAttribL1i64vNV;
01291 extern PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
01292 extern PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribLlui64NV;
01293 extern PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribLlui64vARB;
01294 extern PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribLlui64vNV;
01295 extern PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
01296 extern PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
01297 extern PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
01298 extern PFNGLVERTEXATTRIBL2I64VNVPROC glVertexAttribL2i64vNV;
01299 extern PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
01300 extern PFNGLVERTEXATTRIBL2UI64VNVPROC glVertexAttribL2ui64vNV;
01301 extern PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01302 extern PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
01303 extern PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
01304 extern PFNGLVERTEXATTRIBL3I64VNVPROC glVertexAttribL3i64vNV;
01305 extern PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
01306 extern PFNGLVERTEXATTRIBL3UI64VNVPROC glVertexAttribL3ui64vNV;
01307 extern PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01308 extern PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
01309 extern PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
01310 extern PFNGLVERTEXATTRIBL4I64VNVPROC glVertexAttribL4i64vNV;
01311 extern PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
01312 extern PFNGLVERTEXATTRIBL4UI64VNVPROC glVertexAttribL4ui64vNV;
01313 extern PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
01314 extern PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
01315 extern PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
01316 extern PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
01317 extern PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
01318 extern PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
01319 extern PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
01320 extern PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
01321 extern PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
01322 extern PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
01323 extern PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
01324 extern PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
01325 extern PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
01326 extern PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
01327 extern PFNGLVIEWPORTARRAYVPROC glViewportArrayv;
01328 extern PFNGLVIEWPORTINDEXEDDFPROC glViewportIndexeddf;
01329 extern PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;

```

```
01330 extern PFNGLVIEWPORTPOSITIONSCALENVPROC glViewportPositionWScaleNV;
01331 extern PFNGLVIEWPORTPROC glViewport;
01332 extern PFNGLVIEWPORTSWIZZLENVPROC glViewportSwizzleNV;
01333 extern PFNGLWAITSYNCPROC glWaitSync;
01334 extern PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01335 extern PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01336 extern PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01337 #endif
01338
01340
01344 namespace gg
01345 {
01349   enum BindingPoints
01350   {
01351     LightBindingPoint = 0,
01352     MaterialBindingPoint,
01353   };
01354
01358   extern GLint ggBufferAlignment;
01359
01366   extern void ggInit();
01367
01377   extern void _ggError(const std::string& name = "", unsigned int line = 0);
01378
01389 #if defined(DEBUG)
01390 # define ggError() gg::_ggError(__FILE__, __LINE__)
01391 #else
01392 # define ggError()
01393 #endif
01394
01404   extern void _ggFBOError(const std::string& name = "", unsigned int line = 0);
01405
01416 #if defined(DEBUG)
01417 # define ggFBOError() gg::_ggFBOError(__FILE__, __LINE__)
01418 #else
01419 # define ggFBOError()
01420 #endif
01421
01429   inline void ggCross(GLfloat* c, const GLfloat* a, const GLfloat* b)
01430 {
01431   c[0] = a[1] * b[2] - a[2] * b[1];
01432   c[1] = a[2] * b[0] - a[0] * b[2];
01433   c[2] = a[0] * b[1] - a[1] * b[0];
01434 }
01435
01443   inline GLfloat ggDot3(const GLfloat* a, const GLfloat* b)
01444 {
01445   return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
01446 }
01447
01454   inline GLfloat ggLength3(const GLfloat* a)
01455 {
01456   return sqrtf(ggDot3(a, a));
01457 }
01458
01466   inline GLfloat ggDistance3(const GLfloat* a, const GLfloat* b)
01467 {
01468   const GLfloat c[] { a[0] - b[0], a[1] - b[1], a[2] - b[2], 0.0f };
01469   return ggLength3(c);
01470 }
01471
01478   inline void ggNormalize3(const GLfloat* a, GLfloat* b)
01479 {
01480   const GLfloat l { ggLength3(a) };
01481   assert(l > 0.0f);
01482   b[0] = a[0] / l;
01483   b[1] = a[1] / l;
01484   b[2] = a[2] / l;
01485 }
01486
01492   inline void ggNormalize3(GLfloat* a)
01493 {
01494   const GLfloat l { ggLength3(a) };
01495   assert(l > 0.0f);
01496   a[0] /= l;
01497   a[1] /= l;
01498   a[2] /= l;
01499 }
01500
01508   inline GLfloat ggDot4(const GLfloat* a, const GLfloat* b)
01509 {
01510   return a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + a[3] * b[3];
01511 }
01512
01519   inline GLfloat ggLength4(const GLfloat* a)
01520 {
01521   return sqrtf(ggDot4(a, a));
```

```

01522 }
01523
01524 inline GLfloat ggDistance4(const GLfloat* a, const GLfloat* b)
01525 {
01526     const GLfloat c[] { a[0] - b[0], a[1] - b[1], a[2] - b[2], a[3] - b[3] };
01527     return ggLength4(c);
01528 }
01529
01530
01531 inline void ggNormalize4(const GLfloat* a, GLfloat* b)
01532 {
01533     const GLfloat l { ggLength4(a) };
01534     assert(l > 0.0f);
01535     b[0] = a[0] / l;
01536     b[1] = a[1] / l;
01537     b[2] = a[2] / l;
01538     b[3] = a[3] / l;
01539 }
01540
01541
01542 inline void ggNormalize4(GLfloat* a)
01543 {
01544     const GLfloat l { ggLength4(a) };
01545     assert(l > 0.0f);
01546     a[0] /= l;
01547     a[1] /= l;
01548     a[2] /= l;
01549     a[3] /= l;
01550 }
01551
01552
01553 class GgVector : public std::array<GLfloat, 4>
01554 {
01555     public:
01556
01557     GgVector()
01558     {
01559     }
01560
01561     constexpr GgVector(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3) :
01562         std::array<GLfloat, 4>{ v0, v1, v2, v3 }
01563     {
01564     }
01565
01566     constexpr GgVector(GLfloat c) :
01567         GgVector{ c, c, c, c }
01568     {
01569     }
01570
01571     constexpr GgVector(const GLfloat* a) :
01572         GgVector{ a[0], a[1], a[2], a[3] }
01573     {
01574     }
01575
01576     GgVector operator+(const GgVector& v) const
01577     {
01578         return GgVector{ data()[0] + v[0], data()[1] + v[1], data()[2] + v[2], data()[3] + v[3] };
01579     }
01580
01581     GgVector operator+(GLfloat c) const
01582     {
01583         return GgVector{ data()[0] + c, data()[1] + c, data()[2] + c, data()[3] + c };
01584     }
01585
01586     GgVector& operator+=(const GgVector& v)
01587     {
01588         data()[0] += v[0];
01589         data()[1] += v[1];
01590         data()[2] += v[2];
01591         data()[3] += v[3];
01592         return *this;
01593     }
01594
01595     GgVector& operator+=(GLfloat c)
01596     {
01597         data()[0] += c;
01598         data()[1] += c;
01599         data()[2] += c;
01600         data()[3] += c;
01601         return *this;
01602     }
01603
01604     GgVector operator-(<b>const</b> GgVector& v) const
01605     {
01606         return GgVector{ data()[0] - v[0], data()[1] - v[1], data()[2] - v[2], data()[3] - v[3] };
01607     }
01608
01609     GgVector operator-(<b>GLfloat</b> c) const
01610     {
01611         return GgVector{ data()[0] - c, data()[1] - c, data()[2] - c, data()[3] - c };
01612     }

```

```
01687     }
01688
01695     GgVector& operator+=(const GgVector& v)
01696     {
01697         data()[0] += v[0];
01698         data()[1] += v[1];
01699         data()[2] += v[2];
01700         data()[3] += v[3];
01701         return *this;
01702     }
01703
01710     GgVector& operator-=(const GgVector& v)
01711     {
01712         data()[0] -= v[0];
01713         data()[1] -= v[1];
01714         data()[2] -= v[2];
01715         data()[3] -= v[3];
01716         return *this;
01717     }
01718
01725     GgVector operator*(const GgVector& v) const
01726     {
01727         return GgVector{ data()[0] * v[0], data()[1] * v[1], data()[2] * v[2], data()[3] * v[3] };
01728     }
01729
01736     GgVector operator*(GLfloat c) const
01737     {
01738         return GgVector{ data()[0] * c, data()[1] * c, data()[2] * c, data()[3] * c };
01739     }
01740
01746     GgVector& operator*=(const GgVector& v)
01747     {
01748         data()[0] *= v[0];
01749         data()[1] *= v[1];
01750         data()[2] *= v[2];
01751         data()[3] *= v[3];
01752         return *this;
01753     }
01754
01761     GgVector& operator*=(GLfloat c)
01762     {
01763         data()[0] *= c;
01764         data()[1] *= c;
01765         data()[2] *= c;
01766         data()[3] *= c;
01767         return *this;
01768     }
01769
01776     GgVector operator/(const GgVector& v) const
01777     {
01778         return GgVector{ data()[0] / v[0], data()[1] / v[1], data()[2] / v[2], data()[3] / v[3] };
01779     }
01780
01787     GgVector& operator/=(GgVector& v)
01788     {
01789         data()[0] /= v[0];
01790         data()[1] /= v[1];
01791         data()[2] /= v[2];
01792         data()[3] /= v[3];
01793         return *this;
01794     }
01795
01802     GgVector operator/(GLfloat c) const
01803     {
01804         return GgVector{ data()[0] / c, data()[1] / c, data()[2] / c, data()[3] / c };
01805     }
01806
01813     GgVector& operator/=(GLfloat c)
01814     {
01815         data()[0] /= c;
01816         data()[1] /= c;
01817         data()[2] /= c;
01818         data()[3] /= c;
01819         return *this;
01820     }
01821
01828     GLfloat dot3(const GgVector& v) const
01829     {
01830         return ggDot3(data(), v.data());
01831     }
01832
01838     GLfloat length3() const
01839     {
01840         return ggLength3(data());
01841     }
01842
01849     GLfloat distance3(const GgVector& v) const
```

```
01850  {
01851     return ggDistance3(data(), v.data());
01852 }
01853
01854 GgVector normalize3() const
01855 {
01856     GgVector b;
01857     ggNormalize3(data(), b.data());
01858     return b;
01859 }
01860
01861 GLfloat dot4(const GgVector& v) const
01862 {
01863     return ggDot4(data(), v.data());
01864 }
01865
01866 GLfloat length4() const
01867 {
01868     return ggLength4(data());
01869 }
01870
01871 GLfloat distance4(const GgVector& v) const
01872 {
01873     return ggDistance4(data(), v.data());
01874 }
01875
01876 GgVector normalize4() const
01877 {
01878     GgVector b;
01879     ggNormalize4(data(), b.data());
01880     return b;
01881 }
01882
01883 inline const GgVector& operator+(const GgVector& v)
01884 {
01885     return v;
01886 }
01887
01888 inline GgVector operator+(GLfloat a, const GgVector& b)
01889 {
01890     return GgVector{ a + b[0], a + b[1], a + b[2], a + b[3] };
01891 }
01892
01893 inline const GgVector operator-(const GgVector& v)
01894 {
01895     return GgVector{ -v[0], -v[1], -v[2], -v[3] };
01896 }
01897
01898 inline GgVector operator-(GLfloat a, const GgVector& b)
01899 {
01900     return GgVector{ a - b[0], a - b[1], a - b[2], a - b[3] };
01901 }
01902
01903 inline GgVector operator*(GLfloat a, const GgVector& b)
01904 {
01905     return GgVector{ a * b[0], a * b[1], a * b[2], a * b[3] };
01906 }
01907
01908 inline GgVector operator/(GLfloat a, const GgVector& b)
01909 {
01910     return GgVector{ a / b[0], a / b[1], a / b[2], a / b[3] };
01911 }
01912
01913 inline GgVector ggCross(const GgVector& a, const GgVector& b)
01914 {
01915     GgVector c;
01916     ggCross(c.data(), a.data(), b.data());
01917     return c;
01918 }
01919
02005 inline GLfloat ggDot3(const GgVector& a, const GgVector& b)
02006 {
02007     return ggDot3(a.data(), b.data());
02008 }
02009
02016 inline GLfloat ggLength3(const GgVector& a)
02017 {
02018     return ggLength3(a.data());
02019 }
02020
02028 inline GLfloat ggDistance3(const GgVector& a, const GgVector& b)
02029 {
02030     return ggDistance3(a.data(), b.data());
02031 }
02032
02042 inline GgVector ggNormalize3(const GgVector& a)
```

```
02043 {
02044     GgVector b;
02045     ggNormalize3(a.data(), b.data());
02046     b.data()[3] = 0.0f;
02047     return b;
02048 }
02049
02058 inline void ggNormalize3(GgVector* a)
02059 {
02060     ggNormalize3(a->data());
02061     a->data()[3] = 0.0f;
02062 }
02063
02071 inline GLfloat ggDot4(const GgVector& a, const GgVector& b)
02072 {
02073     return ggDot4(a.data(), b.data());
02074 }
02075
02082 inline GLfloat ggLength4(const GgVector& a)
02083 {
02084     return ggLength4(a.data());
02085 }
02086
02094 inline GLfloat ggDistance4(const GgVector& a, const GgVector& b)
02095 {
02096     return ggDistance4(a.data(), b.data());
02097 }
02098
02105 inline GgVector ggNormalize4(const GgVector& a)
02106 {
02107     GgVector b;
02108     ggNormalize4(a.data(), b.data());
02109     return b;
02110 }
02111
02117 inline void ggNormalize4(GgVector* a)
02118 {
02119     ggNormalize4(a->data());
02120 }
02121
02125 class GgMatrix : public std::array<GLfloat, 16>
02126 {
02127     // 行列 a とベクトル b の積をベクトル c に代入する
02128     void projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02129
02130     // 行列 a と行列 b の積を行列 c に代入する
02131     void multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02132
02133 public:
02134
02138     GgMatrix()
02139     {
02140     }
02141
02162     constexpr GgMatrix(
02163         GLfloat m00, GLfloat m02, GLfloat m03,
02164         GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13,
02165         GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23,
02166         GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33
02167     ) :
02168         std::array<GLfloat, 16>{ m00, m10, m20, m30, m01, m11, m21, m31, m02, m12, m22, m32, m03, m13,
02169         m23, m33 }
02170     {
02171     }
02171
02177     constexpr GgMatrix(GLfloat c) :
02178         GgMatrix{ c, c }
02179     {
02180     }
02181
02187     GgMatrix(const GLfloat* a)
02188     {
02189         operator=(a);
02190     }
02191
02198     GgMatrix& operator=(const GLfloat* a)
02199     {
02200         std::copy(a, a + size(), data());
02201         return *this;
02202     }
02203
02210     GgMatrix operator+(const GLfloat* a) const
02211     {
02212         GgMatrix t;
02213         for (size_t i = 0; i < size(); ++i) t[i] = data()[i] + a[i];
02214         return t;
02215     }
```

```
02216
02217
02218
02219
02220
02221
02222
02223     GgMatrix operator+(const GgMatrix& m) const
02224     {
02225         return operator+(m.data());
02226     }
02227
02228
02229
02230
02231     GgMatrix& operator+=(const GLfloat* a)
02232     {
02233         for (size_t i = 0; i < size(); ++i) data()[i] += a[i];
02234         return *this;
02235     }
02236
02237
02238
02239
02240     GgMatrix& operator+=(const GgMatrix& m)
02241     {
02242         return operator+=(m.data());
02243     }
02244
02245
02246     GgMatrix operator-(const GLfloat* a) const
02247     {
02248         GgMatrix t;
02249         for (size_t i = 0; i < size(); ++i) t[i] = data()[i] - a[i];
02250         return t;
02251     }
02252
02253
02254
02255
02256     GgMatrix operator-(const GgMatrix& m) const
02257     {
02258         return operator-(m.data());
02259     }
02260
02261
02262
02263
02264     GgMatrix& operator-=(const GLfloat* a)
02265     {
02266         for (size_t i = 0; i < size(); ++i) data()[i] -= a[i];
02267         return *this;
02268     }
02269
02270
02271
02272
02273     GgMatrix& operator-=(const GgMatrix& m)
02274     {
02275         return operator-=(m.data());
02276     }
02277
02278
02279
02280
02281     GgMatrix operator*(const GLfloat* a) const
02282     {
02283         GgMatrix t;
02284         multiply(t.data(), data(), a);
02285         return t;
02286     }
02287
02288
02289
02290
02291     GgMatrix operator*(const GgMatrix& m) const
02292     {
02293         return operator*(m.data());
02294     }
02295
02296
02297
02298
02299     GgMatrix& operator*=(const GLfloat* a)
02300     {
02301         *this = operator*(a);
02302         return *this;
02303     }
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317     GgMatrix operator*(const GgMatrix& m) const
02318     {
02319         return operator*(m.data());
02320     }
02321
02322
02323
02324
02325
02326
02327
02328     GgMatrix& operator*=(const GLf32* a)
02329     {
02330         *this = operator*(a);
02331         return *this;
02332     }
02333
02334
02335
02336
02337
02338
02339
02340     GgMatrix& operator*=(const GgMatrix& m)
02341     {
02342         return operator*=(m.data());
02343     }
02344
02345
02346
02347
02348
02349
02350
02351     GgMatrix operator/(const GLfloat* a) const
02352     {
02353         GgMatrix i, t;
02354         i.loadInvert(a);
02355         multiply(t.data(), data(), i.data());
02356         return t;
02357     }
02358
02359
02360
02361
02362
02363
02364
02365     GgMatrix operator/(const GgMatrix& m) const
02366     {
02367         return operator/(m.data());
02368     }
02369
02370
02371
02372
02373
02374
02375
02376     GgMatrix& operator/=(const GLfloat* a)
02377     {
02378         *this = operator/(a);
02379         return *this;
02380     }
02381
02382
02383
02384
02385
02386
02387
02388     GgMatrix& operator/=(const GgMatrix& m)
02389     {
02390         return operator/=(m.data());
02391     }
02392
```

```
02396     GgMatrix& loadIdentity();
02397
02407     GgMatrix& loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02408
02415     GgMatrix& loadTranslate(const GLfloat* t)
02416     {
02417         return loadTranslate(t[0], t[1], t[2]);
02418     }
02419
02426     GgMatrix& loadTranslate(const GgVector& t)
02427     {
02428         return loadTranslate(t[0], t[1], t[2], t[3]);
02429     }
02430
02440     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02441
02448     GgMatrix& loadScale(const GLfloat* s)
02449     {
02450         return loadScale(s[0], s[1], s[2]);
02451     }
02452
02459     GgMatrix& loadScale(const GgVector& s)
02460     {
02461         return loadScale(s[0], s[1], s[2], s[3]);
02462     }
02463
02470     GgMatrix& loadRotateX(GLfloat a);
02471
02478     GgMatrix& loadRotateY(GLfloat a);
02479
02486     GgMatrix& loadRotateZ(GLfloat a);
02487
02497     GgMatrix& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
02498
02506     GgMatrix& loadRotate(const GLfloat* r, GLfloat a)
02507     {
02508         return loadRotate(r[0], r[1], r[2], a);
02509     }
02510
02518     GgMatrix& loadRotate(const GgVector& r, GLfloat a)
02519     {
02520         return loadRotate(r[0], r[1], r[2], a);
02521     }
02522
02529     GgMatrix& loadRotate(const GLfloat* r)
02530     {
02531         return loadRotate(r[0], r[1], r[2], r[3]);
02532     }
02533
02540     GgMatrix& loadRotate(const GgVector& r)
02541     {
02542         return loadRotate(r[0], r[1], r[2], r[3]);
02543     }
02544
02559     GgMatrix& loadLookat(GLfloat ex, GLfloat ey, GLfloat ez,
02560                         GLfloat tx, GLfloat ty, GLfloat tz,
02561                         GLfloat ux, GLfloat uy, GLfloat uz);
02562
02571     GgMatrix& loadLookat(const GLfloat* e, const GLfloat* t, const GLfloat* u)
02572     {
02573         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02574     }
02575
02584     GgMatrix& loadLookat(const GgVector& e, const GgVector& t, const GgVector& u)
02585     {
02586         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02587     }
02588
02600     GgMatrix& loadOrthogonal(GLfloat left, GLfloat right,
02601                           GLfloat bottom, GLfloat top,
02602                           GLfloat zNear, GLfloat zFar);
02603
02615     GgMatrix& loadFrustum(GLfloat left, GLfloat right,
02616                           GLfloat bottom, GLfloat top,
02617                           GLfloat zNear, GLfloat zFar);
02618
02628     GgMatrix& loadPerspective(GLfloat fovy, GLfloat aspect,
02629                               GLfloat zNear, GLfloat zFar);
02630
02637     GgMatrix& loadTranspose(const GLfloat* a);
02638
02645     GgMatrix& loadTranspose(const GgMatrix& m)
02646     {
02647         return loadTranspose(m.data());
02648     }
02649
02656     GgMatrix& loadInvert(const GLfloat* a);
```

```
02657
02664 GgMatrix& loadInvert(const GgMatrix& m)
02665 {
02666     return loadInvert(m.data());
02667 }
02668
02675 GgMatrix& loadNormal(const GLfloat* a);
02676
02683 GgMatrix& loadNormal(const GgMatrix& m)
02684 {
02685     return loadNormal(m.data());
02686 }
02687
02697 GgMatrix translate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02698 {
02699     GgMatrix m;
02700     return operator*(m.loadTranslate(x, y, z, w));
02701 }
02702
02709 GgMatrix translate(const GLfloat* t) const
02710 {
02711     return translate(t[0], t[1], t[2]);
02712 }
02713
02720 GgMatrix translate(const GgVector& t) const
02721 {
02722     return translate(t[0], t[1], t[2], t[3]);
02723 }
02724
02734 GgMatrix scale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02735 {
02736     GgMatrix m;
02737     return operator*(m.loadScale(x, y, z, w));
02738 }
02739
02746 GgMatrix scale(const GLfloat* s) const
02747 {
02748     return scale(s[0], s[1], s[2]);
02749 }
02750
02757 GgMatrix scale(const GgVector& s) const
02758 {
02759     return scale(s[0], s[1], s[2], s[3]);
02760 }
02761
02768 GgMatrix rotateX(GLfloat a) const
02769 {
02770     GgMatrix m;
02771     return operator*(m.loadRotateX(a));
02772 }
02773
02780 GgMatrix rotateY(GLfloat a) const
02781 {
02782     GgMatrix m;
02783     return operator*(m.loadRotateY(a));
02784 }
02785
02792 GgMatrix rotateZ(GLfloat a) const
02793 {
02794     GgMatrix m;
02795     return operator*(m.loadRotateZ(a));
02796 }
02797
02807 GgMatrix rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
02808 {
02809     GgMatrix m;
02810     return operator*(m.loadRotate(x, y, z, a));
02811 }
02812
02820 GgMatrix rotate(const GLfloat* r, GLfloat a) const
02821 {
02822     return rotate(r[0], r[1], r[2], a);
02823 }
02824
02832 GgMatrix rotate(const GgVector& r, GLfloat a) const
02833 {
02834     return rotate(r[0], r[1], r[2], a);
02835 }
02836
02843 GgMatrix rotate(const GLfloat* r) const
02844 {
02845     return rotate(r[0], r[1], r[2], r[3]);
02846 }
02847
02854 GgMatrix rotate(const GgVector& r) const
02855 {
02856     return rotate(r[0], r[1], r[2], r[3]);
```

```
02857     }
02858
02859     GgMatrix lookat(
02860         GLfloat ex, GLfloat ey, GLfloat ez,
02861         GLfloat tx, GLfloat ty, GLfloat tz,
02862         GLfloat ux, GLfloat uy, GLfloat uz
02863     ) const
02864     {
02865         GgMatrix m;
02866         return operator*(m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz));
02867     }
02868
02869     GgMatrix lookat(const GLfloat* e, const GLfloat* t, const GLfloat* u) const
02870     {
02871         return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02872     }
02873
02874     GgMatrix lookat(const GgVector& e, const GgVector& t, const GgVector& u) const
02875     {
02876         return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02877     }
02878
02879     GgMatrix orthogonal(
02880         GLfloat left, GLfloat right,
02881         GLfloat bottom, GLfloat top,
02882         GLfloat zNear, GLfloat zFar
02883     ) const
02884     {
02885         GgMatrix m;
02886         return operator*(m.loadOrthogonal(left, right, bottom, top, zNear, zFar));
02887     }
02888
02889     GgMatrix frustum(
02890         GLfloat left, GLfloat right,
02891         GLfloat bottom, GLfloat top,
02892         GLfloat zNear, GLfloat zFar
02893     ) const
02894     {
02895         GgMatrix m;
02896         return operator*(m.loadFrustum(left, right, bottom, top, zNear, zFar));
02897     }
02898
02899     GgMatrix perspective(
02900         GLfloat fovy, GLfloat aspect,
02901         GLfloat zNear, GLfloat zFar
02902     ) const
02903     {
02904         GgMatrix m;
02905         return operator*(m.loadPerspective(fovy, aspect, zNear, zFar));
02906     }
02907
02908     GgMatrix transpose() const
02909     {
02910         GgMatrix t;
02911         return t.loadTranspose(*this);
02912     }
02913
02914     GgMatrix invert() const
02915     {
02916         GgMatrix t;
02917         return t.loadInvert(*this);
02918     }
02919
02920     GgMatrix normal() const
02921     {
02922         GgMatrix t;
02923         return t.loadNormal(*this);
02924     }
02925
02926     void projection(GLfloat* c, const GLfloat* v) const
02927     {
02928         projection(c, data(), v);
02929     }
02930
02931     void projection(GLfloat* c, const GgVector& v) const
02932     {
02933         projection(c, v.data());
02934     }
02935
02936     void projection(GgVector& c, const GLfloat* v) const
02937     {
02938         projection(c.data(), v);
02939     }
02940
02941     void projection(GgVector& c, const GgVector& v) const
02942     {
02943         projection(c.data(), v.data());
02944     }
```

```
03044     }
03045
03052     GgVector operator*(const GgVector& v) const
03053     {
03054         GgVector c;
03055         projection(c, v);
03056         return c;
03057     }
03058
03064     const GLfloat* get() const
03065     {
03066         return data();
03067     }
03068
03074     void get(GLfloat* a) const
03075     {
03076         std::copy(data(), data() + size(), a);
03077     }
03078 };
03079
03085     inline GgMatrix ggIdentity()
03086     {
03087         GgMatrix t;
03088         return t.loadIdentity();
03089     };
03090
03100    inline GgMatrix ggTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03101    {
03102        GgMatrix m;
03103        return m.loadTranslate(x, y, z, w);
03104    }
03105
03112    inline GgMatrix ggTranslate(const GLfloat* t)
03113    {
03114        GgMatrix m;
03115        return m.loadTranslate(t[0], t[1], t[2]);
03116    }
03117
03124    inline GgMatrix ggTranslate(const GgVector& t)
03125    {
03126        GgMatrix m;
03127        return m.loadTranslate(t[0], t[1], t[2], t[3]);
03128    }
03129
03139    inline GgMatrix ggScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03140    {
03141        GgMatrix m;
03142        return m.loadScale(x, y, z, w);
03143    }
03144
03151    inline GgMatrix ggScale(const GLfloat* s)
03152    {
03153        GgMatrix m;
03154        return m.loadScale(s[0], s[1], s[2]);
03155    }
03156
03163    inline GgMatrix ggScale(const GgVector& s)
03164    {
03165        GgMatrix m;
03166        return m.loadScale(s[0], s[1], s[2], s[3]);
03167    }
03168
03175    inline GgMatrix ggRotateX(GLfloat a)
03176    {
03177        GgMatrix m;
03178        return m.loadRotateX(a);
03179    }
03180
03187    inline GgMatrix ggRotateY(GLfloat a)
03188    {
03189        GgMatrix m;
03190        return m.loadRotateY(a);
03191    }
03192
03199    inline GgMatrix ggRotateZ(GLfloat a)
03200    {
03201        GgMatrix m;
03202        return m.loadRotateZ(a);
03203    }
03204
03214    inline GgMatrix ggRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03215    {
03216        GgMatrix m;
03217        return m.loadRotate(x, y, z, a);
03218    }
03219
03227    inline GgMatrix ggRotate(const GLfloat* r, GLfloat a)
```

```
03228 {
03229     GgMatrix m;
03230     return m.loadRotate(r[0], r[1], r[2], a);
03231 }
03232
03240 inline GgMatrix ggRotate(const GgVector& r, GLfloat a)
03241 {
03242     GgMatrix m;
03243     return m.loadRotate(r[0], r[1], r[2], a);
03244 }
03245
03252 inline GgMatrix ggRotate(const GLfloat* r)
03253 {
03254     GgMatrix m;
03255     return m.loadRotate(r[0], r[1], r[2], r[3]);
03256 }
03257
03264 inline GgMatrix ggRotate(const GgVector& r)
03265 {
03266     GgMatrix m;
03267     return m.loadRotate(r[0], r[1], r[2], r[3]);
03268 }
03269
03284 inline GgMatrix ggLookat(
03285     GLfloat ex, GLfloat ey, GLfloat ez,           // 視点の位置
03286     GLfloat tx, GLfloat ty, GLfloat tz,           // 目標点の位置
03287     GLfloat ux, GLfloat uy, GLfloat uz           // 上方向のベクトル
03288 )
03289 {
03290     GgMatrix m;
03291     return m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz);
03292 }
03293
03302 inline GgMatrix ggLookat(
03303     const GLfloat* e,                           // 視点の位置
03304     const GLfloat* t,                           // 目標点の位置
03305     const GLfloat* u                           // 上方向のベクトル
03306 )
03307 {
03308     GgMatrix m;
03309     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03310 }
03311
03320 inline GgMatrix ggLookat(
03321     const GgVector& e,                         // 視点の位置
03322     const GgVector& t,                         // 目標点の位置
03323     const GgVector& u                         // 上方向のベクトル
03324 )
03325 {
03326     GgMatrix m;
03327     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03328 }
03329
03341 inline GgMatrix ggOrthogonal(GLfloat left,
03342     GLfloat bottom, GLfloat top,
03343     GLfloat zNear, GLfloat zFar)
03344 {
03345     GgMatrix m;
03346     return m.loadOrthogonal(left, right, bottom, top, zNear, zFar);
03347 }
03348
03360 inline GgMatrix ggFrustum(
03361     GLfloat left, GLfloat right,
03362     GLfloat bottom, GLfloat top,
03363     GLfloat zNear, GLfloat zFar
03364 )
03365 {
03366     GgMatrix m;
03367     return m.loadFrustum(left, right, bottom, top, zNear, zFar);
03368 }
03369
03379 inline GgMatrix ggPerspective(
03380     GLfloat fovy, GLfloat aspect,
03381     GLfloat zNear, GLfloat zFar
03382 )
03383 {
03384     GgMatrix m;
03385     return m.loadPerspective(fovy, aspect, zNear, zFar);
03386 }
03387
03394 inline GgMatrix ggTranspose(const GgMatrix& m)
03395 {
03396     return m.transpose();
03397 }
03398
03405 inline GgMatrix ggInvert(const GgMatrix& m)
03406 {
```

```

03407     return m.invert();
03408 }
03409
03416 inline GgMatrix ggNormal(const GgMatrix& m)
03417 {
03418     return m.normal();
03419 }
03420
03424 class GgQuaternion : public GgVector
03425 {
03426     // GgQuaternion 型の四元数 p と四元数 q の積を四元数 r に求める
03427     void multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const;
03428
03429     // GgQuaternion 型の四元数 q が表す回転の変換行列を m に求める
03430     void toMatrix(GLfloat* m, const GLfloat* q) const;
03431
03432     // 回転の変換行列 m が表す四元数を q に求める
03433     void toQuaternion(GLfloat* q, const GLfloat* m) const;
03434
03435     // 球面線形補間 q と r を t で補間した四元数を p に求める
03436     void slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const;
03437
03438 public:
03439
03443     GgQuaternion()
03444     {
03445     }
03446
03455     constexpr GgQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w) :
03456         GgVector{ x, y, z, w }
03457     {
03458     }
03459
03465     constexpr GgQuaternion(GLfloat c) :
03466         GgQuaternion{ c, c, c, c }
03467     {
03468     }
03469
03475     GgQuaternion(const GLfloat* a) :
03476         GgQuaternion{ a[0], a[1], a[2], a[3] }
03477     {
03478     }
03479
03485     GgQuaternion(const GgVector& v) :
03486         GgQuaternion{ v[0], v[1], v[2], v[3] }
03487     {
03488     }
03489
03495     GLfloat norm() const
03496     {
03497         return ggLength4(*this);
03498     }
03499
03509     GgQuaternion& load(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03510     {
03511         data()[0] = x;
03512         data()[1] = y;
03513         data()[2] = z;
03514         data()[3] = w;
03515         return *this;
03516     }
03517
03524     GgQuaternion& load(const GLfloat* a)
03525     {
03526         return load(a[0], a[1], a[2], a[3]);
03527     }
03528
03535     GgQuaternion& load(const GgVector& v)
03536     {
03537         static_cast<GgVector>(*this) = v;
03538         return *this;
03539     }
03540
03547     GgQuaternion& load(const GgQuaternion& q)
03548     {
03549         *this = q;
03550         return *this;
03551     }
03552
03562     GgQuaternion& loadAdd(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03563     {
03564         data()[0] += x;
03565         data()[1] += y;
03566         data()[2] += z;
03567         data()[3] += w;
03568         return *this;
03569     }

```

```
03570
03571     GgQuaternion& loadAdd(const GLfloat* a)
03572     {
03573         return loadAdd(a[0], a[1], a[2], a[3]);
03574     }
03575
03576     GgQuaternion& loadAdd(const GgVector& v)
03577     {
03578         return loadAdd(v[0], v[1], v[2], v[3]);
03579     }
03580
03581     GgQuaternion& loadAdd(const GgQuaternion& q)
03582     {
03583         return loadAdd(q[0], q[1], q[2], q[3]);
03584     }
03585
03586     GgQuaternion& loadSubtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03587     {
03588         data()[0] -= x;
03589         data()[1] -= y;
03590         data()[2] -= z;
03591         data()[3] -= w;
03592         return *this;
03593     }
03594
03595     GgQuaternion& loadSubtract(const GLfloat* a)
03596     {
03597         return loadSubtract(a[0], a[1], a[2], a[3]);
03598     }
03599
03600     GgQuaternion& loadSubtract(const GgVector& v)
03601     {
03602         return loadSubtract(v[0], v[1], v[2], v[3]);
03603     }
03604
03605     GgQuaternion& loadSubtract(const GgQuaternion& q)
03606     {
03607         return loadSubtract(q[0], q[1], q[2], q[3]);
03608     }
03609
03610     GgQuaternion& loadMultiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03611     {
03612         const GLfloat a[] { x, y, z, w };
03613         return loadMultiply(a);
03614     }
03615
03616     GgQuaternion& loadMultiply(const GLfloat* a)
03617     {
03618         return load(multiply(a));
03619     }
03620
03621     GgQuaternion& loadMultiply(const GgVector& v)
03622     {
03623         return loadMultiply(v.data());
03624     }
03625
03626     GgQuaternion& loadMultiply(const GgQuaternion& q)
03627     {
03628         return loadMultiply(q.data());
03629     }
03630
03631     GgQuaternion& loadDivide(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03632     {
03633         const GLfloat a[] { x, y, z, w };
03634         return loadDivide(a);
03635     }
03636
03637     GgQuaternion& loadDivide(const GLfloat* a)
03638     {
03639         return load(divide(a));
03640     }
03641
03642     GgQuaternion& loadDivide(const GgVector& v)
03643     {
03644         return loadDivide(v.data());
03645     }
03646
03647     GgQuaternion& loadDivide(const GgQuaternion& q)
03648     {
03649         return loadDivide(q.data());
03650     }
03651
03652     GgQuaternion add(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03653     {
03654         GgQuaternion s{ *this };
03655         return s.loadAdd(x, y, z, w);
03656     }
```

```
03765
03772     GgQuaternion add(const GLfloat* a) const
03773     {
03774         return add(a[0], a[1], a[2], a[3]);
03775     }
03776
03783     GgQuaternion add(const GgVector& v) const
03784     {
03785         return add(v[0], v[1], v[2], v[3]);
03786     }
03787
03794     GgQuaternion add(const GgQuaternion& q) const
03795     {
03796         return add(q[0], q[1], q[2], q[3]);
03797     }
03798
03808     GgQuaternion subtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03809     {
03810         GgQuaternion s{ *this };
03811         return s.loadSubtract(x, y, z, w);
03812     }
03813
03820     GgQuaternion subtract(const GLfloat* a) const
03821     {
03822         return subtract(a[0], a[1], a[2], a[3]);
03823     }
03824
03831     GgQuaternion subtract(const GgVector& v) const
03832     {
03833         return subtract(v[0], v[1], v[2], v[3]);
03834     }
03835
03842     GgQuaternion subtract(const GgQuaternion& q) const
03843     {
03844         return subtract(q[0], q[1], q[2], q[3]);
03845     }
03846
03856     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03857     {
03858         const GLfloat a[] { x, y, z, w };
03859         return multiply(a);
03860     }
03861
03868     GgQuaternion multiply(const GLfloat* a) const
03869     {
03870         GgQuaternion s;
03871         multiply(s.data(), data(), a);
03872         return s;
03873     }
03874
03881     GgQuaternion multiply(const GgVector& v) const
03882     {
03883         return multiply(v.data());
03884     }
03885
03892     GgQuaternion multiply(const GgQuaternion& q) const
03893     {
03894         return multiply(q.data());
03895     }
03896
03906     GgQuaternion divide(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03907     {
03908         const GLfloat a[] { x, y, z, w };
03909         return divide(a);
03910     }
03911
03918     GgQuaternion divide(const GLfloat* a) const
03919     {
03920         GgQuaternion s, ia;
03921         ia.loadInvert(a);
03922         multiply(s.data(), data(), ia.data());
03923         return s;
03924     }
03925
03932     GgQuaternion divide(const GgVector& v) const
03933     {
03934         return divide(v.data());
03935     }
03936
03943     GgQuaternion divide(const GgQuaternion& q) const
03944     {
03945         return divide(q.data());
03946     }
03947
03948     // 演算子
03949     GgQuaternion& operator=(const GLfloat* a)
03950     {
```

```
03951     return load(a);
03952 }
03953 GgQuaternion& operator=(const GgVector& v)
03954 {
03955     return load(v);
03956 }
03957 GgQuaternion& operator+=(const GLfloat* a)
03958 {
03959     return loadAdd(a);
03960 }
03961 GgQuaternion& operator+=(const GgVector& v)
03962 {
03963     return loadAdd(v);
03964 }
03965 GgQuaternion& operator+=(const GgQuaternion& q)
03966 {
03967     return loadAdd(q);
03968 }
03969 GgQuaternion& operator-=(const GLfloat* a)
03970 {
03971     return loadSubtract(a);
03972 }
03973 GgQuaternion& operator-=(const GgVector& v)
03974 {
03975     return loadSubtract(v);
03976 }
03977 GgQuaternion& operator-=(const GgQuaternion& q)
03978 {
03979     return operator-=(q.data());
03980 }
03981 GgQuaternion& operator*=(const GLfloat* a)
03982 {
03983     return loadMultiply(a);
03984 }
03985 GgQuaternion& operator*=(const GgVector& v)
03986 {
03987     return loadMultiply(v);
03988 }
03989 GgQuaternion& operator*=(const GgQuaternion& q)
03990 {
03991     return operator*=(q.data());
03992 }
03993 GgQuaternion& operator/=(const GLfloat* a)
03994 {
03995     return loadDivide(a);
03996 }
03997 GgQuaternion& operator/=(const GgVector& v)
03998 {
03999     return loadDivide(v);
04000 }
04001 GgQuaternion& operator/=(const GgQuaternion& q)
04002 {
04003     return operator/=(q.data());
04004 }
04005 GgQuaternion operator+(const GLfloat* a) const
04006 {
04007     return add(a);
04008 }
04009 GgQuaternion operator+(const GgVector& v) const
04010 {
04011     return add(v);
04012 }
04013 GgQuaternion operator+(const GgQuaternion& q) const
04014 {
04015     return operator+(q.data());
04016 }
04017 GgQuaternion operator-(const GLfloat* a) const
04018 {
04019     return add(a);
04020 }
04021 GgQuaternion operator-(const GgVector& v) const
04022 {
04023     return add(v);
04024 }
04025 GgQuaternion operator-(const GgQuaternion& q) const
04026 {
04027     return operator-(q.data());
04028 }
04029 GgQuaternion operator*(const GLfloat* a) const
04030 {
04031     return multiply(a);
04032 }
04033 GgQuaternion operator*(const GgVector& v) const
04034 {
04035     return multiply(v);
04036 }
04037 GgQuaternion operator*(const GgQuaternion& q) const
```

```
04038 {
04039     return operator*(q.data());
04040 }
04041 GgQuaternion operator/(const GLfloat* a) const
04042 {
04043     return divide(a);
04044 }
04045 GgQuaternion operator/(const GgVector& v) const
04046 {
04047     return divide(v);
04048 }
04049 GgQuaternion operator/(const GgQuaternion& q) const
04050 {
04051     return operator/(q.data());
04052 }
04053
04054 GgQuaternion& loadMatrix(const GLfloat* a)
04055 {
04056     toQuaternion(data(), a);
04057     return *this;
04058 }
04059
04060 GgQuaternion& loadMatrix(const GgMatrix& m)
04061 {
04062     return loadMatrix(m.get());
04063 }
04064
04065 GgQuaternion& loadIdentity()
04066 {
04067     return load(0.0f, 0.0f, 0.0f, 1.0f);
04068 }
04069
04070 GgQuaternion& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
04071
04072 GgQuaternion& loadRotate(const GLfloat* v, GLfloat a)
04073 {
04074     return loadRotate(v[0], v[1], v[2], a);
04075 }
04076
04077 GgQuaternion& loadRotate(const GLfloat* v)
04078 {
04079     return loadRotate(v[0], v[1], v[2], v[3]);
04080 }
04081
04082 GgQuaternion& loadRotateX(GLfloat a);
04083
04084 GgQuaternion& loadRotateY(GLfloat a);
04085
04086 GgQuaternion& loadRotateZ(GLfloat a);
04087
04088 GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
04089 {
04090     GgQuaternion q;
04091     return multiply(q.loadRotate(x, y, z, a));
04092 }
04093
04094 GgQuaternion rotate(const GLfloat* v, GLfloat a) const
04095 {
04096     return rotate(v[0], v[1], v[2], a);
04097 }
04098
04099 GgQuaternion rotate(const GLfloat* v) const
04100 {
04101     return rotate(v[0], v[1], v[2], v[3]);
04102 }
04103
04104 GgQuaternion rotateX(GLfloat a) const
04105 {
04106     return rotate(1.0f, 0.0f, 0.0f, a);
04107 }
04108
04109 GgQuaternion rotateY(GLfloat a) const
04110 {
04111     return rotate(0.0f, 1.0f, 0.0f, a);
04112 }
04113
04114 GgQuaternion rotateZ(GLfloat a) const
04115 {
04116     return rotate(0.0f, 0.0f, 1.0f, a);
04117 }
04118
04119 GgQuaternion& loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll);
04120
04121 GgQuaternion& loadEuler(const GLfloat* e)
04122 {
04123     return loadEuler(e[0], e[1], e[2]);
04124 }
04125
```

```
04236
04237
04238
04239
04240
04241
04242
04243
04244
04245     GgQuaternion euler(GLfloat heading, GLfloat pitch, GLfloat roll) const
04246     {
04247         GgQuaternion r;
04248         return multiply(r.loadEuler(heading, pitch, roll));
04249     }
04250
04251
04252     GgQuaternion euler(const GLfloat* e) const
04253     {
04254         return euler(e[0], e[1], e[2]);
04255     }
04256
04257     GgQuaternion& loadSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04258     {
04259         slerp(data(), a, b, t);
04260         return *this;
04261     }
04262
04263     GgQuaternion& loadSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04264     {
04265         return loadSlerp(q.data(), r.data(), t);
04266     }
04267
04268     GgQuaternion& loadSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04269     {
04270         return loadSlerp(q.data(), a, t);
04271     }
04272
04273     GgQuaternion& loadSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04274     {
04275         return loadSlerp(a, q.data(), t);
04276     }
04277
04278     GgQuaternion& loadNormalize(const GLfloat* a);
04279
04280     GgQuaternion& loadNormalize(const GgQuaternion& q)
04281     {
04282         return loadNormalize(q.data());
04283     }
04284
04285     GgQuaternion& loadConjugate(const GLfloat* a);
04286
04287     GgQuaternion& loadConjugate(const GgQuaternion& q)
04288     {
04289         return loadConjugate(q.data());
04290     }
04291
04292     GgQuaternion& loadInvert(const GLfloat* a);
04293
04294     GgQuaternion& loadInvert(const GgQuaternion& q)
04295     {
04296         return loadInvert(q.data());
04297     }
04298
04299     GgQuaternion slerp(GLfloat* a, GLfloat t) const
04300     {
04301         GgQuaternion p;
04302         slerp(p.data(), data(), a, t);
04303         return p;
04304     }
04305
04306     GgQuaternion slerp(const GgQuaternion& q, GLfloat t) const
04307     {
04308         GgQuaternion p;
04309         slerp(p.data(), data(), q.data(), t);
04310         return p;
04311     }
04312
04313     GgQuaternion normalize() const
04314     {
04315         GgQuaternion q;
04316         q.loadNormalize(data());
04317         return q;
04318     }
04319
04320     GgQuaternion conjugate() const
04321     {
04322         GgQuaternion q;
04323         q.loadConjugate(data());
04324         return q;
04325     }
04326
04327     GgQuaternion invert() const
04328     {
04329         GgQuaternion q;
04330         q.loadInvert(data());
04331         return q;
04332     }
04333
```

```
04434     }
04435
04436     void get(GLfloat* a) const
04437     {
04438         a[0] = data()[0];
04439         a[1] = data()[1];
04440         a[2] = data()[2];
04441         a[3] = data()[3];
04442     }
04443
04444     void getMatrix(GLfloat* a) const
04445     {
04446         toMatrix(a, data());
04447     }
04448
04449     void getMatrix(GgMatrix& m) const
04450     {
04451         getMatrix(m.data());
04452     }
04453
04454     GgMatrix getMatrix() const
04455     {
04456         GgMatrix m;
04457         getMatrix(m);
04458         return m;
04459     }
04460
04461     void getConjugateMatrix(GLfloat* a) const
04462     {
04463         GgQuaternion c;
04464         c.loadConjugate(data());
04465         toMatrix(a, c.data());
04466     }
04467
04468     void getConjugateMatrix(GgMatrix& m) const
04469     {
04470         getConjugateMatrix(m.data());
04471     }
04472
04473     GgMatrix getConjugateMatrix() const
04474     {
04475         GgMatrix m;
04476         getConjugateMatrix(m);
04477         return m;
04478     }
04479
04480     inline GgQuaternion ggQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
04481     {
04482         GgQuaternion q;
04483         return q.load(x, y, z, w);
04484     }
04485
04486     inline GgQuaternion ggQuaternion(const GLfloat* a)
04487     {
04488         return ggQuaternion(a[0], a[1], a[2], a[3]);
04489     }
04490
04491     inline GgQuaternion ggIdentityQuaternion()
04492     {
04493         GgQuaternion q;
04494         return q.loadIdentity();
04495     }
04496
04497     inline GgQuaternion ggMatrixQuaternion(const GLfloat* a)
04498     {
04499         GgQuaternion q;
04500         return q.loadMatrix(a);
04501     }
04502
04503     inline GgMatrix ggMatrixQuaternion(const GgMatrix& m)
04504     {
04505         return ggMatrixQuaternion(m.get());
04506     }
04507
04508     inline GgMatrix ggQuaternionMatrix(const GgQuaternion& q)
04509     {
04510         GLfloat m[16];
04511         q.getMatrix(m);
04512         return GgMatrix{ m };
04513     }
04514
04515     inline GgMatrix ggQuaternionTransposeMatrix(const GgQuaternion& q)
04516     {
04517         GLfloat m[16];
04518         q.getMatrix(m);
04519         GgMatrix t;
```

```

04600     return t.loadTranspose(m);
04601 }
04602
04612 inline GgQuaternion ggRotateQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
04613 {
04614     GgQuaternion q;
04615     return q.loadRotate(x, y, z, a);
04616 }
04617
04625 inline GgQuaternion ggRotateQuaternion(const GLfloat* v, GLfloat a)
04626 {
04627     return ggRotateQuaternion(v[0], v[1], v[2], a);
04628 }
04629
04636 inline GgQuaternion ggRotateQuaternion(const GLfloat* v)
04637 {
04638     return ggRotateQuaternion(v[0], v[1], v[2], v[3]);
04639 }
04640
04649 inline GgQuaternion ggEulerQuaternion(GLfloat heading, GLfloat pitch, GLfloat roll)
04650 {
04651     GgQuaternion q;
04652     return q.loadEuler(heading, pitch, roll);
04653 }
04654
04661 inline GgQuaternion ggEulerQuaternion(const GLfloat* e)
04662 {
04663     return ggEulerQuaternion(e[0], e[1], e[2]);
04664 }
04665
04674 inline GgQuaternion ggSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04675 {
04676     GgQuaternion r;
04677     return r.loadSlerp(a, b, t);
04678 }
04679
04688 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04689 {
04690     return ggSlerp(q.data(), r.data(), t);
04691 }
04692
04701 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04702 {
04703     return ggSlerp(q.data(), a, t);
04704 }
04705
04714 inline GgQuaternion ggSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04715 {
04716     return ggSlerp(a, q.data(), t);
04717 }
04718
04725 inline GLfloat ggNorm(const GgQuaternion& q)
04726 {
04727     return q.norm();
04728 }
04729
04736 inline GgQuaternion ggNormalize(const GgQuaternion& q)
04737 {
04738     return q.normalize();
04739 }
04740
04747 inline GgQuaternion ggConjugate(const GgQuaternion& q)
04748 {
04749     return q.conjugate();
04750 }
04751
04758 inline GgQuaternion ggInvert(const GgQuaternion& q)
04759 {
04760     return q.invert();
04761 }
04762
04766 class GgTrackball : public GgQuaternion
04767 {
04768     bool drag;          // ドラッグ中か否か
04769     GLfloat start[2];  // ドラッグ開始位置
04770     GLfloat scale[2];  // マウスの絶対位置→ウィンドウ内での相対位置の換算係数
04771     GgQuaternion cq;   // 回転の初期値（四元数）
04772     GgMatrix rt;       // 回転の変換行列
04773
04774 public:
04775
04781     GgTrackball(const GgQuaternion& q = ggIdentityQuaternion())
04782     {
04783         reset(q);
04784     }
04785
04789     virtual ~GgTrackball()

```

```
04790 {
04791 }
04792
04793     GgTrackball& operator=(const GgQuaternion& q)
04794     {
04795         this->load(q);
04796         return *this;
04797     }
04798
04799     void region(GLfloat w, GLfloat h);
04800
04801     void region(int w, int h)
04802     {
04803         region(static_cast<GLfloat>(w), static_cast<GLfloat>(h));
04804     }
04805
04806     void begin(GLfloat x, GLfloat y);
04807
04808     void motion(GLfloat x, GLfloat y);
04809
04810     void rotate(const GgQuaternion& q);
04811
04812     void end(GLfloat x, GLfloat y);
04813
04814     void reset(const GgQuaternion& q = ggIdentityQuaternion());
04815
04816     const GLfloat* getStart() const
04817     {
04818         return start;
04819     }
04820
04821     const GLfloat& getStart(int direction) const
04822     {
04823         return start[direction];
04824     }
04825
04826     void getStart(GLfloat* position) const
04827     {
04828         position[0] = start[0];
04829         position[1] = start[1];
04830     }
04831
04832     const GLfloat* getScale() const
04833     {
04834         return scale;
04835     }
04836
04837     const GLfloat getScale(int direction) const
04838     {
04839         return scale[direction];
04840     }
04841
04842     void getScale(GLfloat* factor) const
04843     {
04844         factor[0] = scale[0];
04845         factor[1] = scale[1];
04846     }
04847
04848     const GgQuaternion& getQuaternion() const
04849     {
04850         return *this;
04851     }
04852
04853     const GgMatrix& getMatrix() const
04854     {
04855         return rt;
04856     }
04857
04858     const GLfloat* get() const
04859     {
04860         return rt.get();
04861     }
04862 };
04863
04864 extern bool ggSaveTga(
04865     const std::string& name,
04866     const void* buffer,
04867     unsigned int width,
04868     unsigned int height,
04869     unsigned int depth
04870 );
04871
04872 extern bool ggSaveColor(const std::string& name);
04873
04874 extern bool ggSaveDepth(const std::string& name);
04875
04876 extern bool ggReadImage(
```

```
05012     const std::string& name,
05013     std::vector<GLubyte>& image,
05014     GLsizei* pWidth,
05015     GLsizei* pHeight,
05016     GLenum* pFormat
05017 );
05018
05032 extern GLuint ggLoadTexture(
05033     const GLvoid* image,
05034     GLsizei width,
05035     GLsizei height,
05036     GLenum format = GL_RGB,
05037     GLenum type = GL_UNSIGNED_BYTE,
05038     GLenum internal = GL_RGB,
05039     GLenum wrap = GL_CLAMP_TO_EDGE,
05040     bool swizzle = true
05041 );
05042
05053 extern GLuint ggLoadImage(
05054     const std::string& name,
05055     GLsizei* pWidth = nullptr,
05056     GLsizei* pHeight = nullptr,
05057     GLenum internal = 0,
05058     GLenum wrap = GL_CLAMP_TO_EDGE
05059 );
05060
05072 extern void ggCreateNormalMap(
05073     const GLubyte* hmap,
05074     GLsizei width,
05075     GLsizei height,
05076     GLenum format,
05077     GLfloat nz,
05078     GLenum internal,
05079     std::vector<GgVector>& nmap
05080 );
05081
05092 extern GLuint ggLoadHeight(
05093     const std::string& name,
05094     GLfloat nz,
05095     GLsizei* pWidth = nullptr,
05096     GLsizei* pHeight = nullptr,
05097     GLenum internal = GL_RGBA
05098 );
05099
05113 extern GLuint ggCreateShader(
05114     const std::string& vsrc,
05115     const std::string& fsrc = "",
05116     const std::string& gsrc = "",
05117     GLint nvarying = 0,
05118     const char* const* varyings = nullptr,
05119     const std::string& vtext = "vertex shader",
05120     const std::string& ftext = "fragment shader",
05121     const std::string& gtext = "geometry shader");
05122
05133 extern GLuint ggLoadShader(
05134     const std::string& vert,
05135     const std::string& frag = "",
05136     const std::string& geom = "",
05137     GLint nvarying = 0,
05138     const char* const* varyings = nullptr
05139 );
05140
05149 inline GLuint ggLoadShader(
05150     const std::array<std::string, 3>& files,
05151     GLint nvarying = 0,
05152     const char* const* varyings = nullptr
05153 )
05154 {
05155     return ggLoadShader(files[0], files[1], files[2], nvarying, varyings);
05156 }
05157
05158 #if !defined(__APPLE__)
05166 extern GLuint ggCreateComputeShader(
05167     const std::string& csrc,
05168     const std::string& ctext = "compute shader"
05169 );
05170
05177 extern GLuint ggLoadComputeShader(const std::string& comp);
05178 #endif
05186 class GgTexture
05187 {
05188     // テクスチャ名
05189     GLuint texture;
05190
05191     // テクスチャの縦横の画素数
05192     GLsizei size[2];
```

```

05193
05194 public:
05195
05208     GgTexture(
05209         const GLvoid* image,
05210         GLsizei width,
05211         GLsizei height,
05212         GLenum format = GL_RGB,
05213         GLenum type = GL_UNSIGNED_BYTE,
05214         GLenum internal = GL_RGBA,
05215         GLenum wrap = GL_CLAMP_TO_EDGE,
05216         bool swizzle = true
05217     ) :
05218         texture{ ggLoadTexture(image, width, height, format, type, internal, wrap, swizzle) },
05219         size{ width, height }
05220     {
05221     }
05222
05226     virtual ~GgTexture()
05227     {
05228         glBindTexture(GL_TEXTURE_2D, 0);
05229         glDeleteTextures(1, &texture);
05230     }
05231
05235     GgTexture(const GgTexture& o) = delete;
05236
05240     GgTexture& operator=(const GgTexture& o) = delete;
05241
05245     void bind() const
05246     {
05247         glBindTexture(GL_TEXTURE_2D, texture);
05248     }
05249
05253     void unbind() const
05254     {
05255         glBindTexture(GL_TEXTURE_2D, 0);
05256     }
05257
05263     void swapRandB(bool swizzle) const;
05264
05270     const GLsizei& getWidth() const
05271     {
05272         return size[0];
05273     }
05274
05280     const GLsizei& getHeight() const
05281     {
05282         return size[1];
05283     }
05284
05290     void getSize(GLsizei* size) const
05291     {
05292         size[0] = getWidth();
05293         size[1] = getHeight();
05294     }
05295
05301     const GLsizei* getSize() const
05302     {
05303         return size;
05304     }
05305
05311     const GLuint& getTexture() const
05312     {
05313         return texture;
05314     }
05315 };
05316
05323 class GgColorTexture
05324 {
05325     // テクスチャ
05326     std::shared_ptr<GgTexture> texture;
05327
05328 public:
05329
05333     GgColorTexture()
05334     {
05335     }
05336
05349     GgColorTexture(
05350         const GLvoid* image,
05351         GLsizei width,
05352         GLsizei height,
05353         GLenum format = GL_RGB,
05354         GLenum type = GL_UNSIGNED_BYTE,
05355         GLenum internal = GL_RGB,
05356         GLenum wrap = GL_CLAMP_TO_EDGE,
05357         bool swizzle = true

```

```
05358      )
05359  {
05360    load(image, width, height, format, internal, wrap, swizzle);
05361  }
05362
05370  GgColorTexture(
05371    const std::string& name,
05372    GLenum internal = 0,
05373    GLenum wrap = GL_CLAMP_TO_EDGE
05374  )
05375  {
05376    load(name, internal, wrap);
05377  }
05378
05382  virtual ~GgColorTexture()
05383  {
05384  }
05385
05398  void load(
05399    const GLvoid* image,
05400    GLsizei width,
05401    GLsizei height,
05402    GLenum format = GL_RGB,
05403    GLenum type = GL_UNSIGNED_BYTE,
05404    GLenum internal = GL_RGB,
05405    GLenum wrap = GL_CLAMP_TO_EDGE,
05406    bool swizzle = true
05407  )
05408  {
05409    // テクスチャを作成する
05410    texture = std::make_shared<GgTexture>(image, width, height, format, type, internal, wrap,
05411    swizzle);
05412  }
05420
05421  void load(
05422    const std::string& name,
05423    GLenum internal = 0,
05424    GLenum wrap = GL_CLAMP_TO_EDGE
05425  );
05426
05433  class GgNormalTexture
05434  {
05435    // テクスチャ
05436    std::shared_ptr<GgTexture> texture;
05437
05438  public:
05439
05443  GgNormalTexture()
05444  {
05445  }
05446
05457  GgNormalTexture(
05458    const GLubyte* image,
05459    GLsizei width,
05460    GLsizei height,
05461    GLenum format = GL_RED,
05462    GLfloat nz = 1.0f,
05463    GLenum internal = GL_RGBA
05464  )
05465  {
05466    // 法線マップのテクスチャを作成する
05467    load(image, width, height, format, nz, internal);
05468  }
05469
05477  GgNormalTexture(
05478    const std::string& name,
05479    GLfloat nz = 1.0f,
05480    GLenum internal = GL_RGBA
05481  )
05482  {
05483    // 法線マップのテクスチャを作成する
05484    load(name, nz, internal);
05485  }
05486
05490  virtual ~GgNormalTexture()
05491  {
05492  }
05493
05504  void load(
05505    const GLubyte* hmap,
05506    GLsizei width,
05507    GLsizei height,
05508    GLenum format = GL_RED,
05509    GLfloat nz = 1.0f,
05510    GLenum internal = GL_RGBA
05511  )
```

```

05512 {
05513     // 法線マップ
05514     std::vector<GgVector> nmap;
05515
05516     // 法線マップを作成する
05517     ggCreateNormalMap(hmap, width, height, format, nz, internal, nmap);
05518
05519     // テクスチャを作成する
05520     texture = std::make_shared<GgTexture>(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal,
05521     GL_REPEAT);
05522 }
05523
05524     void load(
05525         const std::string& name,
05526         GLfloat nz = 1.0f,
05527         GLenum internal = GL_RGBA
05528     );
05529 };
05530
05531     template <typename T>
05532     class GgBuffer
05533     {
05534         // ターゲット
05535         const GLenum target;
05536
05537         // バッファオブジェクトのアライメントを考慮したデータの間隔
05538         const GLsizei stride;
05539
05540         // データの数
05541         const GLsizei count;
05542
05543         // バッファオブジェクト
05544         const GLuint buffer;
05545
05546     public:
05547
05548         GgBuffer<T>(
05549             GLenum target,
05550             const T* data,
05551             GLsizei stride,
05552             GLsizei count,
05553             GLenum usage
05554             ) :
05555             target{ target },
05556             stride{ stride },
05557             count{ count },
05558             buffer{ [] { GLuint buffer; glGenBuffers(1, &buffer); return buffer; } () }
05559     {
05560         // バッファオブジェクトのメモリを確保してデータを転送する
05561         glBindBuffer(target, buffer);
05562         glBufferData(target, getStride()* count, data, usage);
05563     }
05564
05565         virtual ~GgBuffer<T>()
05566     {
05567         // バッファオブジェクトを削除する
05568         glBindBuffer(target, 0);
05569         glDeleteBuffers(1, &buffer);
05570     }
05571
05572         GgBuffer<T>(& operator=(const GgBuffer<T>& o) = delete;
05573
05574         GgBuffer<T>& operator=(const GgBuffer<T>& o) = delete;
05575
05576         const GLuint& getTarget() const
05577     {
05578         return target;
05579     }
05580
05581         GLsizei getStride() const
05582     {
05583         return static_cast<GLsizeiptr>(stride);
05584     }
05585
05586         GLsizei getCount() const
05587     {
05588         return count;
05589     }
05590
05591         const GLuint& getBuffer() const
05592     {
05593         return buffer;
05594     }
05595
05596         void bind() const
05597     {
05598         glBindBuffer(target, buffer);
05599     }

```

```

05652     }
05653
05654     void unbind() const
05655     {
05656         glBindBuffer(target, 0);
05657     }
05658
05659     void* map() const
05660     {
05661         glBindBuffer(target, buffer);
05662 #if defined(GL_GLES_PROTOTYPES)
05663         return glMapBufferRange(target, 0, getStride() * count, GL_MAP_WRITE_BIT);
05664 #else
05665         return glMapBuffer(target, GL_WRITE_ONLY);
05666 #endif
05667     }
05668
05669     void* map(GLint first, GLsizei count) const
05670     {
05671         // count が 0 なら全データをマップする
05672         if (count == 0) count = getCount();
05673         if (first + count > getCount()) count = getCount() - first;
05674
05675         glBindBuffer(target, buffer);
05676         return glMapBufferRange(target, getStride() * first, getStride() * count, GL_MAP_WRITE_BIT);
05677     }
05678
05679     void unmap() const
05680     {
05681         glUnmapBuffer(target);
05682     }
05683
05684     void send(const T* data, GLint first, GLsizei count) const
05685     {
05686         // count が 0 なら全データを転送する
05687         if (count == 0) count = getCount();
05688         if (first + count > getCount()) count = getCount() - first;
05689
05690         glBindBuffer(target, buffer);
05691         glBufferSubData(target, getStride() * first, getStride() * count, data);
05692     }
05693
05694     void read(T* data, GLint first, GLsizei count) const
05695     {
05696         // count が 0 なら全データを抽出する
05697         if (count == 0) count = getCount();
05698         if (first + count > getCount()) count = getCount() - first;
05699
05700         // データを既存のバッファオブジェクトに転送する
05701         glBindBuffer(target, buffer);
05702         glBufferSubData(target, getStride() * first, getStride() * count, data);
05703     }
05704
05705     void copy(GLuint src_buffer, GLint src_first = 0, GLint dst_first = 0, GLsizei count = 0) const
05706     {
05707         // count が 0 なら全データを複写する
05708         if (count == 0) count = getCount();
05709         if (src_first + count > getCount()) count = getCount() - src_first;
05710         if (dst_first + count > getCount()) count = getCount() - dst_first;
05711
05712         // データの間隔
05713         const GLsizei stride{ getStride() };
05714
05715         glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
05716         glBindBuffer(GL_COPY_WRITE_BUFFER, buffer);
05717         glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
05718             stride * src_first, stride * dst_first, stride * count);
05719         glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
05720         glBindBuffer(GL_COPY_READ_BUFFER, 0);
05721     };
05722
05723     template <typename T>
05724     class GgUniformBuffer
05725     {
05726         // ユニフォームバッファオブジェクト
05727         std::shared_ptr<GgBuffer<T>> uniform;
05728     };
05729
05730
05731
05732
05733
05734
05735
05736
05737
05738
05739
05740
05741
05742
05743
05744
05745
05746
05747
05748
05749
05750
05751
05752
05753
05754
05755
05756
05757
05758
05759
05760
05761
05762
05763
05764
05765
05766
05767
05768
05769
05770
05771
05772
05773
05774
05775
05776
05777
05778
05779
05780
05781
05782
05783
05784

```

```

05785 public:
05786     GgUniformBuffer<T>()
05791 {
05792 }
05793
05801     GgUniformBuffer<T>(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05802 {
05803     load(data, count, usage);
05804 }
05805
05813     GgUniformBuffer<T>(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05814 {
05815     load(data, count, usage);
05816 }
05817
05821     virtual ~GgUniformBuffer<T>()
05822 {
05823 }
05824
05830     const GLuint& getTarget() const
05831 {
05832     return uniform->getTarget();
05833 }
05834
05840     GLsizeiptr getStride() const
05841 {
05842     return uniform->getStride();
05843 }
05844
05850     const GLsizei& getCount() const
05851 {
05852     return uniform->getCount();
05853 }
05854
05860     const GLuint& getBuffer() const
05861 {
05862     return uniform->getBuffer();
05863 }
05864
05868     void bind() const
05869 {
05870     uniform->bind();
05871 }
05872
05876     void unbind() const
05877 {
05878     uniform->unbind();
05879 }
05880
05886     void* map() const
05887 {
05888     return uniform->map();
05889 }
05890
05898     void* map(GLint first, GLsizei count) const
05899 {
05900     return uniform->map(first, count);
05901 }
05902
05906     void unmap() const
05907 {
05908     uniform->unmap();
05909 }
05910
05918     void load(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05919 {
05920     // バッファオブジェクト上のデータの間隔
05921     const GLsizei stride{ ((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1 } *
05922         ggBufferAlignment };
05923
05924     // ユニフォームバッファオブジェクトを確保する
05925     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05926
05927     // 確保したユニフォームバッファオブジェクトにデータを転送する
05928     if (data) send(data, 0, sizeof(T), 0, count);
05929
05937     void load(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05938 {
05939     // バッファオブジェクト上のデータの間隔
05940     const GLsizei stride{ ((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1 } *
05941         ggBufferAlignment };
05942
05943     // ユニフォームバッファオブジェクトを確保する
05944     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05944

```

```
05945 // 確保したユニフォームバッファオブジェクトにデータを転送する
05946     fill(&data, 0, sizeof(T), 0, count);
05947 }
05948
05949 void send(
05950     const GLvoid* data,
05951     GLint offset = 0,
05952     GLsizei size = sizeof(T),
05953     GLint first = 0,
05954     GLsizei count = 0
05955 ) const
05956 {
05957     // count が 0 なら全データを転送する
05958     if (count == 0) count = getCount();
05959     if (first + count > getCount()) count = getCount() - first;
05960
05961     // 転送元のデータの先頭
05962     const char* source{ reinterpret_cast<const char*>(data) };
05963
05964     // ターゲット
05965     const GLuint target{ getTarget() };
05966
05967     // データの間隔
05968     const GLsizeiptr stride{ getStride() };
05969
05970     // first 番目のブロックから count 個の各ブロックの先頭から offset バイトの位置にデータを転送する
05971     bind();
05972     for (GLsizei i = 0; i < count; ++i)
05973     {
05974         glBufferSubData(target, stride * (first + i) + offset, size, source + size * i);
05975     }
05976
05977     void fill(
05978         const GLvoid* data,
05979         GLint offset = 0,
05980         GLsizei size = sizeof(T),
05981         GLint first = 0,
05982         GLsizei count = 0
05983     ) const
05984     {
05985         // count が 0 なら全データを転送する
05986         if (count == 0) count = getCount();
05987         if (first + count > getCount()) count = getCount() - first;
05988
05989         // ターゲット
05990         const GLuint target{ getTarget() };
05991
05992         // データの間隔
05993         const GLsizeiptr stride{ getStride() };
05994
05995         // first 番目のブロックから count 個の各ブロックの先頭から offset バイトの位置にデータを転送する
05996         bind();
05997         for (GLsizei i = 0; i < count; ++i)
05998         {
05999             glBufferSubData(target, stride * (first + i) + offset, size, data);
06000         }
06001
06002     }
06003
06004     void read(
06005         GLvoid* data,
06006         GLint offset = 0,
06007         GLsizei size = sizeof(T),
06008         GLint first = 0,
06009         GLsizei count = 0
06010     ) const
06011     {
06012         // count が 0 なら全データを転送する
06013         if (count == 0) count = getCount();
06014         if (first + count > getCount()) count = getCount() - first;
06015
06016         // 抽出先のデータの先頭
06017         char* const destination{ reinterpret_cast<char*>(data) };
06018
06019         // ターゲット
06020         const GLuint target{ getTarget() };
06021
06022         // データの間隔
06023         const GLsizeiptr stride{ getStride() };
06024
06025         // データをユニフォームバッファオブジェクトから抽出する
06026         bind();
06027 #if defined(GL_GLES_PROTOTYPES)
06028         const char* const source{ glMapBufferRange(target, stride * first, stride * count,
06029             GL_MAP_READ_BIT) };
06030         for (GLsizei i = 0; i < count; ++i)
06031         {
06032             glReadPixels(first + i * stride, offset, size, GL_RGBA, GL_UNSIGNED_BYTE, source + i * size);
06033         }
06034     }
06035 }
```

```

06058     const char* const begin{ source + stride * i + offset };
06059     const char* const end{ begin + size };
06060     std::copy(begin, end, destination + sizeof(T) * i);
06061 }
06062 glUnmapBuffer(target);
06063 #else
06064     for (GLsizei i = 0; i < count; ++i)
06065     {
06066         glGetBufferSubData(target, stride * (first + i) + offset, size, destination + sizeof(T) * i);
06067     }
06068 #endif
06069 }
06070
06071 void copy(
06072     GLuint src_buffer,
06073     GLint src_first = 0,
06074     GLint dst_first = 0,
06075     GLsizei count = 0
06076 ) const
06077 {
06078     // count が 0 なら全データを複写する
06079     if (count == 0) count = getCount();
06080     if (src.first + count > getCount()) count = getCount() - src.first;
06081     if (dst.first + count > getCount()) count = getCount() - dst.first;
06082
06083     // データの間隔
06084     const GLsizeiptr stride{ getStride() };
06085
06086     // ユニフォームバッファオブジェクトではブロックごとに転送する
06087     glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
06088     glBindBuffer(GL_COPY_WRITE_BUFFER, getBuffer());
06089     for (GLsizei i = 0; i < count; ++i)
06090     {
06091         glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
06092             stride * (src.first + i), stride * (dst.first + i), sizeof(T));
06093     }
06094     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
06095     glBindBuffer(GL_COPY_READ_BUFFER, 0);
06096 }
06097
06098 class GgVertexArray
06099 {
06100     // 頂点配列オブジェクト
06101     const GLuint vao;
06102
06103 public:
06104
06105     GgVertexArray(GLenum mode = 0) :
06106         vao{ [] { GLuint vao; glGenVertexArrays(1, &vao); return vao; } () }
06107     {
06108         glBindVertexArray(vao);
06109     }
06110     GgVertexArray(const GgVertexArray& o) = delete;
06111
06112     virtual ~GgVertexArray()
06113     {
06114         glBindVertexArray(0);
06115         glDeleteVertexArrays(1, &vao);
06116     }
06117
06118     GgVertexArray& operator=(const GgVertexArray& o) = delete;
06119
06120     const GLuint& get() const
06121     {
06122         return vao;
06123     }
06124
06125     void bind() const
06126     {
06127         glBindVertexArray(vao);
06128     }
06129
06130 class GgShape
06131 {
06132     // 頂点配列オブジェクト
06133     std::shared_ptr<GgVertexArray> object;
06134
06135     // 基本図形の種類
06136     GLenum mode;
06137
06138 public:
06139
06140     GgShape(GLenum mode = 0) :
06141         object{ std::make_shared<GgVertexArray>() },

```

```
06190     mode{ mode }
06191     {
06192     }
06193
06197     virtual ~GgShape()
06198     {
06199     }
06200
06206     virtual explicit operator bool() const noexcept
06207     {
06208         return object.get() != nullptr;
06209     }
06210
06216     virtual bool operator!() const noexcept
06217     {
06218         return !static_cast<bool>(*this);
06219     }
06220
06226     const GLuint& get() const
06227     {
06228         return object->get();
06229     }
06230
06236     void setMode(GLenum mode)
06237     {
06238         this->mode = mode;
06239     }
06240
06246     const GLenum& getMode() const
06247     {
06248         return this->mode;
06249     }
06250
06257     virtual void draw(GLint first = 0, GLsizei count = 0) const
06258     {
06259         object->bind();
06260     }
06261 };
06262
06266 class GgPoints
06267     : public GgShape
06268 {
06269     // 頂点バッファオブジェクト
06270     std::shared_ptr<GgBuffer<GgVector>> position;
06271
06272 public:
06273
06277     GgPoints(GLenum mode = GL_POINTS) :
06278         GgShape(mode)
06279     {
06280     }
06281
06290     GgPoints(
06291         const GgVector* pos,
06292         GLsizei countv,
06293         GLenum mode = GL_POINTS,
06294         GLenum usage = GL_STATIC_DRAW
06295     ) :
06296         GgPoints(mode)
06297     {
06298         load(pos, countv, usage);
06299     }
06300
06304     virtual ~GgPoints()
06305     {
06306     }
06307
06313     explicit operator bool() const noexcept
06314     {
06315         return position.get() != nullptr;
06316     }
06317
06323     bool operator!() const noexcept
06324     {
06325         return !static_cast<bool>(*this);
06326     }
06327
06333     const GLsizei& getCount() const
06334     {
06335         return position->getCount();
06336     }
06337
06343     const GLuint& getBuffer() const
06344     {
06345         return position->getBuffer();
06346     }
06347
```

```

06355     void send(const GgVector* pos, GLint first = 0, GLsizei count = 0) const
06356     {
06357         position->send(pos, first, count);
06358     }
06359
06360     void load(const GgVector* pos, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06361
06362     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06363 };
06364
06365 struct GgVertex
06366 {
06367     GgVector position;
06368     GgVector normal;
06369
06370     GgVertex()
06371     {
06372     }
06373
06374     GgVertex(const GgVector& pos, const GgVector& norm) :
06375         position(pos),
06376         normal(norm)
06377     {
06378     }
06379
06380     GgVertex(
06381         GLfloat px, GLfloat py, GLfloat pz,
06382         GLfloat nx, GLfloat ny, GLfloat nz
06383     ) :
06384         position{ px, py, pz, 1.0f },
06385         normal{ nx, ny, nz, 0.0f }
06386     {
06387     }
06388
06389     GgVertex(const GLfloat* pos, const GLfloat* norm) :
06390         GgVertex(pos[0], pos[1], pos[2], norm[0], norm[1], norm[2])
06391     {
06392     }
06393 };
06394
06395 class GgTriangles
06396     : public GgShape
06397 {
06398     // 頂点属性
06399     std::shared_ptr<GgBuffer<GgVertex>> vertex;
06400
06401     public:
06402
06403     GgTriangles(GLenum mode = GL_TRIANGLES) :
06404         GgShape(mode)
06405     {
06406     }
06407
06408     GgTriangles(
06409         const GgVertex* vert,
06410         GLsizei count,
06411         GLenum mode = GL_TRIANGLES,
06412         GLenum usage = GL_STATIC_DRAW
06413     ) :
06414         GgTriangles(mode)
06415     {
06416         load(vert, count, usage);
06417     }
06418
06419     virtual ~GgTriangles()
06420     {
06421     }
06422
06423     const GLsizei& getCount() const
06424     {
06425         return vertex->getCount();
06426     }
06427
06428     const GLuint& getBuffer() const
06429     {
06430         return vertex->getBuffer();
06431     }
06432
06433     void send(const GgVertex* vert, GLint first = 0, GLsizei count = 0) const
06434     {
06435         vertex->send(vert, first, count);
06436     }
06437
06438     void load(const GgVertex* vert, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06439
06440     virtual void draw(GLint first = 0, GLsizei count = 0) const;

```

```
06534     };
06535
06539     class GgElements
06540         : public GgTriangles
06541     {
06542         // インデックスを格納する頂点バッファオブジェクト
06543         std::shared_ptr<GgBuffer<GLuint>> index;
06544
06545     public:
06546
06552         GgElements(GLenum mode = GL_TRIANGLES) :
06553             GgTriangles(mode)
06554         {
06555     }
06556
06567         GgElements(
06568             const GgVertex* vert,
06569             GLsizei countv,
06570             const GLuint* face,
06571             GLsizei countf,
06572             GLenum mode = GL_TRIANGLES,
06573             GLenum usage = GL_STATIC_DRAW
06574         ) :
06575             GgElements(mode)
06576         {
06577             load(vert, countv, face, countf, usage);
06578         }
06579
06583         virtual ~GgElements()
06584     {
06585     }
06586
06591         const GLsizei& getIndexCount() const
06592     {
06593         return index->getCount();
06594     }
06595
06601         const GLuint& getIndexBuffer() const
06602     {
06603         return index->getBuffer();
06604     }
06605
06616         void send(
06617             const GgVertex* vert,
06618             GLuint firstv,
06619             GLsizei countv,
06620             const GLuint* face = nullptr,
06621             GLuint firstf = 0,
06622             GLsizei countf = 0
06623         ) const
06624     {
06625         GgTriangles::send(vert, firstv, countv);
06626         if (face != nullptr && countf > 0) index->send(face, firstf, countf);
06627     }
06628
06638         void load(
06639             const GgVertex* vert,
06640             GLsizei countv,
06641             const GLuint* face,
06642             GLsizei countf,
06643             GLenum usage = GL_STATIC_DRAW
06644         )
06645     {
06646         // 頂点バッファオブジェクトを作成する
06647         GgTriangles::load(vert, countv, usage);
06648
06649         // インデックスの頂点バッファオブジェクトを作成する
06650         index = std::make_shared<GgBuffer<GLuint>>(GL_ELEMENT_ARRAY_BUFFER, face,
06651             static_cast<GLsizei>(sizeof(GLuint)), countf, usage);
06652
06659         virtual void draw(GLint first = 0, GLsizei count = 0) const;
06660     };
06661
06672     extern std::shared_ptr<GgPoints> ggPointsCube(
06673         GLsizei countv,
06674         GLfloat length = 1.0f,
06675         GLfloat cx = 0.0f,
06676         GLfloat cy = 0.0f,
06677         GLfloat cz = 0.0f
06678     );
06679
06690     extern std::shared_ptr<GgPoints> ggPointsSphere(
06691         GLsizei countv,
06692         GLfloat radius = 0.5f,
06693         GLfloat cx = 0.0f,
06694         GLfloat cy = 0.0f,
```

```
06695     GLfloat cz = 0.0f
06696 );
06697
06705 extern std::shared_ptr<GgTriangles> ggRectangle(
06706     GLfloat width = 1.0f,
06707     GLfloat height = 1.0f
06708 );
06709
06718 extern std::shared_ptr<GgTriangles> ggEllipse(
06719     GLfloat width = 1.0f,
06720     GLfloat height = 1.0f,
06721     GLuint slices = 16
06722 );
06723
06735 extern std::shared_ptr<GgTriangles> ggArraysObj(
06736     const std::string& name,
06737     bool normalize = false
06738 );
06739
06751 extern std::shared_ptr<GgElements> ggElementsObj(
06752     const std::string& name,
06753     bool normalize = false
06754 );
06755
06768 extern std::shared_ptr<GgElements> ggElementsMesh(
06769     GLuint slices,
06770     GLuint stacks,
06771     const GLfloat(*pos)[3],
06772     const GLfloat(*norm)[3] = nullptr
06773 );
06774
06785 extern std::shared_ptr<GgElements> ggElementsSphere(
06786     GLfloat radius = 1.0f,
06787     int slices = 16,
06788     int stacks = 8
06789 );
06790
06797 class GgShader
06798 {
06799     // プログラム名
06800     const GLuint program;
06801
06802 public:
06803
06813     GgShader(
06814         const std::string& vert,
06815         const std::string& frag = "",
06816         const std::string& geom = "",
06817         int nvarying = 0,
06818         const char* const* varyings = nullptr
06819     ) :
06820         program(ggLoadShader(vert, frag, geom, nvarying, varyings))
06821     {
06822     }
06823
06831     GgShader(
06832         const std::array<std::string, 3>& files,
06833         int nvarying = 0,
06834         const char* const* varyings = nullptr
06835     ) :
06836         GgShader(files[0], files[1], files[2], nvarying, varyings)
06837     {
06838     }
06839
06843     GgShader(const GgShader& o) = delete;
06844
06848     virtual ~GgShader()
06849     {
06850         // 参照しているオブジェクトが一つだけならシェーダを削除する
06851         glUseProgram(0);
06852         glDeleteProgram(program);
06853     }
06854
06858     GgShader& operator=(const GgShader& o) = delete;
06859
06863     void use() const
06864     {
06865         glUseProgram(program);
06866     }
06867
06871     void unuse() const
06872     {
06873         glUseProgram(0);
06874     }
06875
06881     GLuint get() const
06882     {
```

```
06883     return program;
06884 }
06885 };
06886
06890 class GgPointShader
06891 {
06892     // シェーダー
06893     std::shared_ptr<GgShader> shader;
06894
06895     // 投影変換行列の uniform 変数の場所
06896     GLint mpLoc;
06897
06898     // モデルビュー変換行列の uniform 変数の場所
06899     GLint mvLoc;
06900
06901 public:
06902
06906     GgPointShader() :
06907         mpLoc{ -1 },
06908         mvLoc{ -1 }
06909     {
06910     }
06911
06921     GgPointShader(
06922         const std::string& vert,
06923         const std::string& frag = "",
06924         const std::string& geom = "",
06925         GLint nvarying = 0,
06926         const char* const* varyings = nullptr
06927     ) :
06928         GgPointShader()
06929     {
06930         load(vert, frag, geom, nvarying, varyings);
06931     }
06932
06940     GgPointShader(
06941         const std::array<std::string, 3>& files,
06942         int nvarying = 0,
06943         const char* const* varyings = nullptr
06944     ) :
06945         GgPointShader(files[0], files[1], files[2], nvarying, varyings)
06946     {
06947     }
06948
06952     virtual ~GgPointShader()
06953     {
06954     }
06955
06966     bool load(
06967         const std::string& vert,
06968         const std::string& frag = "",
06969         const std::string& geom = "",
06970         GLint nvarying = 0,
06971         const char* const* varyings = nullptr
06972     )
06973     {
06974         // シェーダを作成する
06975         shader = std::make_shared<GgShader>(vert, frag, geom, nvarying, varyings);
06976
06977         // プログラム名を取り出す
06978         const GLuint program(shader->get());
06979
06980         // プログラムオブジェクトが作成できていなければ戻る
06981         if (program == 0) return false;
06982
06983         // 変換行列の uniform 変数の場所
06984         mpLoc = glGetUniformLocation(program, "mp");
06985         mvLoc = glGetUniformLocation(program, "mv");
06986
06987         // プログラムオブジェクトの作成に成功した
06988         return true;
06989     }
06990
06999     bool load(
07000         const std::array<std::string, 3>& files,
07001         GLint nvarying = 0,
07002         const char* const* varyings = nullptr
07003     )
07004     {
07005         return load(files[0], files[1], files[2], nvarying, varyings);
07006     }
07007
07013     virtual void loadProjectionMatrix(const GLfloat* mp) const
07014     {
07015         glUniformMatrix4fv(mpLoc, 1, GL_FALSE, mp);
07016     }
07017
```

```

07023     virtual void loadProjectionMatrix(const GgMatrix& mp) const
07024     {
07025         loadProjectionMatrix(mp.get());
07026     }
07027
07033     virtual void loadModelviewMatrix(const GLfloat* mv) const
07034     {
07035         glUniformMatrix4fv(mvLoc, 1, GL_FALSE, mv);
07036     }
07037
07043     virtual void loadModelviewMatrix(const GgMatrix& mv) const
07044     {
07045         loadModelviewMatrix(mv.get());
07046     }
07047
07054     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07055     {
07056         loadProjectionMatrix(mp);
07057         loadModelviewMatrix(mv);
07058     }
07059
07066     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
07067     {
07068         loadMatrix(mp.get(), mv.get());
07069     }
07070
07074     virtual void use() const
07075     {
07076         shader->use();
07077     }
07078
07084     void use(const GLfloat* mp) const
07085     {
07086         use();
07087         loadProjectionMatrix(mp);
07088     }
07089
07095     void use(const GgMatrix& mp) const
07096     {
07097         use(mp.get());
07098     }
07099
07106     void use(const GLfloat* mp, const GLfloat* mv) const
07107     {
07108         use(mp);
07109         loadModelviewMatrix(mv);
07110     }
07111
07118     void use(const GgMatrix& mp, const GgMatrix& mv) const
07119     {
07120         use(mp.get(), mv.get());
07121     }
07122
07126     void unuse() const
07127     {
07128         shader->unuse();
07129     }
07130
07136     GLuint get() const
07137     {
07138         return shader->get();
07139     }
07140 };
07141
07145 class GgSimpleShader
07146     : public GgPointShader
07147 {
07148     // 材質データの uniform block のインデックス
07149     GLint materialIndex;
07150
07151     // 光源データの uniform block のインデックス
07152     GLint lightIndex;
07153
07154     // モデルビュー変換の法線変換行列の uniform 変数の場所
07155     GLint mnLoc;
07156
07157 public:
07158
07162     GgSimpleShader() :
07163         GgPointShader(),
07164         materialIndex{ -1 },
07165         lightIndex{ -1 },
07166         mnLoc{ -1 }
07167     {
07168     }
07169     GgSimpleShader(

```

```
07180     const std::string& vert,
07181     const std::string& frag = "",
07182     const std::string& geom = "",
07183     GLint nvarying = 0,
07184     const char* const* varyings = nullptr
07185 )
07186 {
07187     load(vert, frag, geom, nvarying, varyings);
07188 }
07189
07190 GgSimpleShader(
07191     const std::array<std::string, 3>& files,
07192     GLint nvarying = 0,
07193     const char* const* varyings = nullptr
07194 ) :
07195     GgSimpleShader(files[0], files[1], files[2], nvarying, varyings)
07196 {
07197 }
07198
07199 GgSimpleShader(const GgSimpleShader& o) :
07200     GgPointShader(o),
07201     materialIndex{ o.materialIndex },
07202     lightIndex{ o.lightIndex },
07203     mnLoc{ o.mnLoc }
07204 {
07205 }
07206
07207 virtual ~GgSimpleShader()
07208 {
07209 }
07210
07211 GgSimpleShader& operator=(const GgSimpleShader& o)
07212 {
07213     if (&o != this)
07214     {
07215         GgPointShader::operator=(o);
07216         materialIndex = o.materialIndex;
07217         lightIndex = o.lightIndex;
07218         mnLoc = o.mnLoc;
07219     }
07220
07221     return *this;
07222 }
07223
07224 bool load(
07225     const std::string& vert,
07226     const std::string& frag = "",
07227     const std::string& geom = "",
07228     GLint nvarying = 0,
07229     const char* const* varyings = nullptr
07230 );
07231
07232 bool load(
07233     const std::array<std::string, 3>& files,
07234     GLint nvarying = 0,
07235     const char* const* varyings = nullptr
07236 )
07237 {
07238     return load(files[0], files[1], files[2], nvarying, varyings);
07239 }
07240
07241 virtual void loadModelviewMatrix(const GLfloat* mv, const GLfloat* mn) const
07242 {
07243     GgPointShader::loadModelviewMatrix(mv);
07244     glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07245 }
07246
07247 virtual void loadModelviewMatrix(const GgMatrix& mv, const GgMatrix& mn) const
07248 {
07249     loadModelviewMatrix(mv.get(), mn.get());
07250 }
07251
07252 virtual void loadModelviewMatrix(const GLfloat* mv) const
07253 {
07254     loadModelviewMatrix(mv, GgMatrix(mv).normal().get());
07255 }
07256
07257 virtual void loadModelviewMatrix(const GgMatrix& mv) const
07258 {
07259     loadModelviewMatrix(mv.get());
07260 }
07261
07262 virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07263 {
07264     GgPointShader::loadMatrix(mp, mv);
07265     glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07266 }
```

```

07330
07338     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07339     {
07340         loadMatrix(mp.get(), mv.get(), mn.get());
07341     }
07342
07349     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07350     {
07351         loadMatrix(mp, mv, GgMatrix(mv).normal());
07352     }
07353
07360     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
07361     {
07362         loadMatrix(mp, mv, mv.normal());
07363     }
07364
07368     struct Light
07369     {
07370         GgVector ambient;
07371         GgVector diffuse;
07372         GgVector specular;
07373         GgVector position;
07374     };
07375
07379     class LightBuffer
07380         : public GgUniformBuffer<Light>
07381     {
07382     public:
07383
07391         LightBuffer(
07392             const Light* light = nullptr,
07393             GLsizei count = 1,
07394             GLenum usage = GL_STATIC_DRAW
07395         ) :
07396             GgUniformBuffer<Light>(light, count, usage)
07397         {}
07398
07407         LightBuffer(
07408             const Light& light,
07409             GLsizei count = 1,
07410             GLenum usage = GL_STATIC_DRAW
07411         ) :
07412             GgUniformBuffer<Light>(light, count, usage)
07413         {}
07414
07415
07419         ~LightBuffer()
07420     {
07421     }
07422
07433         void loadAmbient(
07434             GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07435             GLint first = 0, GLsizei count = 1
07436         ) const;
07437
07445         void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07446
07454         void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07455     {
07456         // first 番目のブロックから count 個の ambient 要素に値を設定する
07457         send(ambient, offsetof(Light, ambient), sizeof(Light::ambient), first, count);
07458     }
07459
07470         void loadDiffuse(
07471             GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07472             GLint first = 0, GLsizei count = 1
07473         ) const;
07474
07482         void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07483
07491         void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07492     {
07493         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07494         send(diffuse, offsetof(Light, diffuse), sizeof(Light::diffuse), first, count);
07495     }
07496
07507         void loadSpecular(
07508             GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07509             GLint first = 0, GLsizei count = 1
07510         ) const;
07511
07519         void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07520
07528         void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07529     {
07530         // first 番目のブロックから count 個の specular 要素に値を設定する

```

```
07531     send(specular, offsetof(Light, specular), sizeof(Light::specular), first, count);
07532 }
07533
07541 void loadColor(const Light& color, GLint first = 0, GLsizei count = 1) const;
07542
07553 void loadPosition(
07554     GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f,
07555     GLint first = 0, GLsizei count = 1
07556 ) const;
07557
07565 void loadPosition(const GgVector& position, GLint first = 0, GLsizei count = 1) const;
07566
07574 void loadPosition(const GLfloat* position, GLint first = 0, GLsizei count = 1) const
07575 {
07576     // first 番目のブロックから count 個の position 要素に値を設定する
07577     send(position, offsetof(Light, position), sizeof(Light::position), first, count);
07578 }
07579
07587 void loadPosition(const GgVector* position, GLint first = 0, GLsizei count = 1) const
07588 {
07589     loadPosition(position->data(), first, count);
07590 }
07591
07599 void load(const Light* light, GLint first = 0, GLsizei count = 1) const
07600 {
07601     send(light, 0, sizeof(Light), first, count);
07602 }
07603
07611 void load(const Light& light, GLint first = 0, GLsizei count = 1) const
07612 {
07613     load(&light, first, count);
07614 }
07615
07621 void select(GLint i = 0) const
07622 {
07623     // バッファオブジェクトの i 番目のブロックの位置
07624     const GLintptr offset(static_cast<GLintptr>(getStride()) * i);
07625     glBindBufferRange(getTarget(), LightBindingPoint, getBuffer(), offset, sizeof(Light));
07626 }
07627
07628
07632 struct Material
07633 {
07634     GgVector ambient;
07635     GgVector diffuse;
07636     GgVector specular;
07637     GLfloat shininess;
07638 };
07639
07643 class MaterialBuffer
07644     : public GgUniformBuffer<Material>
07645 {
07646     public:
07647
07655     MaterialBuffer(
07656         const Material* material = nullptr,
07657         GLsizei count = 1,
07658         GLenum usage = GL_STATIC_DRAW
07659     ) :
07660         GgUniformBuffer<Material>(material, count, usage)
07661     {
07662     }
07663
07671     MaterialBuffer(
07672         const Material& material,
07673         GLsizei count = 1,
07674         GLenum usage = GL_STATIC_DRAW
07675     ) :
07676         GgUniformBuffer<Material>(material, count, usage)
07677     {
07678     }
07679
07683     virtual ~MaterialBuffer()
07684     {
07685     }
07686
07697     void loadAmbient(
07698         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07699         GLint first = 0, GLsizei count = 1
07700     ) const;
07701
07709     void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07710
07718     void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07719 {
07720     // first 番目のブロックから count 個のブロックの ambient 要素に値を設定する
07721     send(ambient, offsetof(Material, ambient), sizeof(Material::ambient), first, count);
```

```

07722     }
07723
07734     void loadDiffuse(
07735         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07736         GLint first = 0, GLsizei count = 1
07737     ) const;
07738
07746     void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07747
07755     void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07756     {
07757         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07758         send(diffuse, offsetof(Material, diffuse), sizeof(Material::diffuse), first, count);
07759     }
07760
07771     void loadAmbientAndDiffuse(
07772         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07773         GLint first = 0, GLsizei count = 1
07774     ) const;
07775
07783     void loadAmbientAndDiffuse(const GgVector& color, GLint first = 0, GLsizei count = 1) const;
07784
07792     void loadAmbientAndDiffuse(const GLfloat* color, GLint first = 0, GLsizei count = 1) const;
07793
07804     void loadSpecular(
07805         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07806         GLint first = 0, GLsizei count = 1
07807     ) const;
07808
07816     void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07817
07825     void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07826     {
07827         // first 番目のブロックから count 個の specular 要素に値を設定する
07828         send(specular, offsetof(Material, specular), sizeof(Material::specular), first, count);
07829     }
07830
07838     void loadShininess(GLfloat shininess, GLint first = 0, GLsizei count = 1) const;
07839
07847     void loadShininess(const GLfloat* shininess, GLint first = 0, GLsizei count = 1) const;
07848
07856     void load(const Material* material, GLint first = 0, GLsizei count = 1) const
07857     {
07858         send(material, 0, sizeof(Material), first, count);
07859     }
07860
07868     void load(const Material& material, GLint first = 0, GLsizei count = 1) const
07869     {
07870         load(&material, first, count);
07871     }
07872
07878     void select(GLint i = 0) const
07879     {
07880         // パッファオブジェクトの i 番目のブロックの位置
07881         const GLintptr offset{ static_cast<GLintptr>(getStride()) * i };
07882         glBindBufferRange(getTarget(), MaterialBindingPoint, getBuffer(), offset, sizeof(Material));
07883     }
07884
07885
07889     void use() const
07890     {
07891         // プログラムオブジェクトは基底クラスで指定する
07892         GgPointShader::use();
07893     }
07894
07902     void use(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07903     {
07904         // プログラムオブジェクトを指定する
07905         use();
07906
07907         // 変換行列を設定する
07908         loadMatrix(mp, mv, mn);
07909     }
07910
07918     void use(const GgMatrixX& mp, const GgMatrixX& mv, const GgMatrixX& mn) const
07919     {
07920         use(mp.get(), mv.get(), mn.get());
07921     }
07922
07929     void use(const GLfloat* mp, const GLfloat* mv) const
07930     {
07931         use(mp, mv, GgMatrix(mv).normal().get());
07932     }
07933
07940     void use(const GgMatrixX& mp, const GgMatrixX& mv) const
07941     {
07942         use(mp, mv, mv.normal());
07943     }

```

```
07943     }
07944
07951     void use(const LightBuffer* light, GLint i = 0) const
07952     {
07953         // プログラムオブジェクトを指定する
07954         use();
07955
07956         // 光源を設定する
07957         light->select(i);
07958     }
07959
07960     void use(const LightBuffer& light, GLint i = 0) const
07961     {
07962         use(&light, i);
07963     }
07970
07980     void use(
07981         const GLfloat* mp,
07982         const GLfloat* mv,
07983         const GLfloat* mn,
07984         const LightBuffer* light,
07985         GLint i = 0
07986     ) const
07987     {
07988         // 光源を指定してプログラムオブジェクトを指定する
07989         use(light, i);
07990
07991         // 変換行列を設定する
07992         loadMatrix(mp, mv, mn);
07993     }
07994
08004     void use(
08005         const GgMatrix& mp,
08006         const GgMatrix& mv,
08007         const GgMatrix& mn,
08008         const LightBuffer& light,
08009         GLint i = 0
08010     ) const
08011     {
08012         use(mp.get(), mv.get(), mn.get(), &light, i);
08013     }
08014
08023     void use(
08024         const GLfloat* mp,
08025         const GLfloat* mv,
08026         const LightBuffer* light,
08027         GLint i = 0
08028     ) const
08029     {
08030         use(mp, mv, GgMatrix(mv).normal().get(), light, i);
08031     }
08032
08041     void use(
08042         const GgMatrix& mp,
08043         const GgMatrix& mv,
08044         const LightBuffer& light,
08045         GLint i = 0
08046     ) const
08047     {
08048         use(mp, mv, mv.normal(), light, i);
08049     }
08050
08058     void use(const GLfloat* mp, const LightBuffer* light, GLint i = 0) const
08059     {
08060         // 光源を指定してプログラムオブジェクトを指定する
08061         use(light, i);
08062
08063         // 投影変換行列を設定する
08064         loadProjectionMatrix(mp);
08065     }
08066
08074     void use(const GgMatrix& mp, const LightBuffer& light, GLint i = 0) const
08075     {
08076         // 光源を指定してプログラムオブジェクトを指定する
08077         use(mp.get(), &light, i);
08078     }
08079 };
08080
08091     extern bool ggLoadSimpleObj(
08092         const std::string& name,
08093         std::vector<std::array<GLuint, 3>>& group,
08094         std::vector<GgSimpleShader::Material>& material,
08095         std::vector<GgVertex>& vert,
08096         bool normalize = false
08097     );
08098
08099     extern bool ggLoadSimpleObj(
```

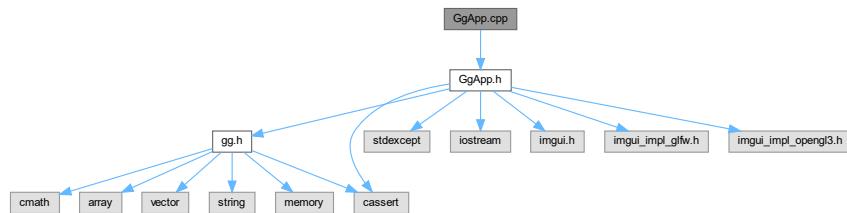
```

08111 const std::string& name,
08112 std::vector<std::array<GLuint, 3>>& group,
08113 std::vector<GgSimpleShader::Material>& material,
08114 std::vector<GgVertex>& vert,
08115 std::vector<GLuint>& face,
08116 bool normalize = false
08117 );
08118
08119 class GgSimpleObj
08120 {
08121 // 同じ材質を割り当てるポリゴングループごとの三角形数
08122 std::shared_ptr<std::vector<std::array<GLuint, 3>>> group;
08123
08124 // ポリゴングループごとの材質のユニフォームバッファ
08125 std::shared_ptr<GgSimpleShader::MaterialBuffer> material;
08126
08127 // この図形の形状データ
08128 std::shared_ptr<GgElements> data;
08129
08130 public:
08131
08132 GgSimpleObj(const std::string& name, bool normalize = false);
08133
08134 virtual ~GgSimpleObj()
08135 {
08136 }
08137
08138 explicit operator bool() const noexcept
08139 {
08140     return data.get() != nullptr;
08141 }
08142
08143 bool operator!() const noexcept
08144 {
08145     return !static_cast<bool>(*this);
08146 }
08147
08148 const GgTriangles* get() const
08149 {
08150     return data.get();
08151 }
08152
08153 virtual void draw(GLint first = 0, GLsizei count = 0) const;
08154 };
08155
08156 };
08157
08158
08159
08160
08161
08162
08163
08164
08165
08166
08167
08168
08169
08170
08171
08172
08173
08174
08175
08176
08177
08178
08179
08180
08181
08182
08183
08184
08185
08186
08187

```

## 9.37 GgApp.cpp ファイル

#include "GgApp.h"  
GgApp.cpp の依存先関係図:



### 9.37.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラスの実装

著者

Kohe Tokoi

日付

November 22, 2022

GgApp.cpp に定義があります。

## 9.38 GgApp.cpp

### [詳解]

```
00001
00008 #include "GgApp.h"
00009
00010 //
00011 // GLFW のエラー表示
00012 //
00013 static void glfwErrorCallback(int error, const char* description)
00014 {
00015 #if defined(__aarch64__)
00016     if (error == 65544) return;
00017 #endif
00018     throw std::runtime_error(description);
00019 }
00020
00021 //
00022 // GgApp クラスのコンストラクタ
00023 //
00024 GgApp::GgApp(int major, int minor)
00025 {
00026     // GLFW のエラー処理関数を登録する
00027     glfwSetErrorCallback(glfwErrorCallback);
00028
00029     // GLFW を初期化する
00030     if (glfwInit() == GLFW_FALSE) throw std::runtime_error("Can't initialize GLFW");
00031
00032     // OpenGL の major 番号が指定されていれば
00033     if (major > 0)
00034     {
00035         // OpenGL のバージョンを指定する
00036         glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, major);
00037         glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, minor);
00038
00039 #if defined(GL_GLES_PROTOTYPES)
00040     // OpenGL ES 3 のコンテキストを指定する
00041     glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
00042     glfwWindowHint(GLFW_CONTEXT_CREATION_API, GLFW_EGL_CONTEXT_API);
00043 #else
00044     // OpenGL Version 3.2 以降なら
00045     if (major * 10 + minor >= 32)
00046     {
00047         // Core Profile を選択する (macOS の都合)
00048         glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00049         glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00050     }
00051 #endif
00052 }
00053
00054 #if defined(GG_USE_OOCULUS_RIFT)
00055     // Oculus Rift では SRGB でレンダリングする
00056     glfwWindowHint(GLFW_SRGB_CAPABLE, GL_TRUE);
00057 #endif
00058
00059 #if defined(IMGUI_VERSION)
00060     // ImGui のバージョンをチェックする
00061     IMGUI_CHECKVERSION();
00062
00063     // ImGui のコンテキストを作成する
00064     ImGui::CreateContext();
00065 #endif
00066 }
00067
00068 //
```

```

00069 // デストラクタ
00070 //
00071 GgApp::~GgApp()
00072 {
00073 #if defined(IMGUI_VERSION)
00074 // Shutdown Platform/Renderer bindings
00075 ImGui_ImplOpenGL3_Shutdown();
00076 ImGui_ImplGlfw_Shutdown();
00077 ImGui::DestroyContext();
00078 #endif
00079
00080 // プログラム終了時に GLFW を終了する
00081 glfwTerminate();
00082 }
00083
00084 //
00085 // マウスや矢印キーによる平行移動量を初期化する
00086 //
00087 void GgApp::Window::HumanInterface::resetTranslation()
00088 {
00089 // 平行移動量を初期化する
00090 for (auto& t : translation)
00091 {
00092 std::fill(t.begin(), t.end(), GgVector{ 0.0f, 0.0f, 0.0f, 1.0f });
00093 }
00094
00095 // 矢印キーの設定値を初期化する
00096 std::fill(arrow.begin(), arrow.end(), std::array<int, 2>{ 0, 0 });
00097
00098 // マウスホイールの回転量を初期化する
00099 std::fill(wheel.begin(), wheel.end(), 0.0f);
00100 }
00101
00102 //
00103 // 平行移動量と回転量を更新する (X, Y のみ, Z は wheel() で計算する)
00104 //
00105 void GgApp::Window::HumanInterface::calcTranslation(int button, const std::array<GLfloat, 3>&
velocity)
00106 {
00107 // マウスの相対変位
00108 assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00109 const auto dx{ (mouse[0] - rotation[button].getStart(0)) * rotation[button].getScale(0) };
00110 const auto dy{ (rotation[button].getStart(1) - mouse[1]) * rotation[button].getScale(1) };
00111
00112 // 平行移動量
00113 auto& t{ translation[button] };
00114
00115 // 平行移動量の更新
00116 t[1][0] = dx * velocity[0] + t[0][0];
00117 t[1][1] = dy * velocity[1] + t[0][1];
00118
00119 // 回転量の更新
00120 rotation[button].motion(mouse[0], mouse[1]);
00121 }
00122
00123 //
00124 // ウィンドウのサイズ変更時の処理
00125 //
00126 void GgApp::Window::resize(GLFWwindow* window, int width, int height)
00127 {
00128 // このインスタンスの this ポインタを得る
00129 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00130
00131 if (instance)
00132 {
00133 // ウィンドウのサイズを保存する
00134 instance->size[0] = width;
00135 instance->size[1] = height;
00136
00137 // トラックボール処理の範囲を設定する
00138 for (auto& current_if : instance->interfaceData)
00139 {
00140 for (auto& t : current_if.rotation)
00141 {
00142 t.region(width, height);
00143 }
00144 }
00145
00146 // ビューポートを更新する
00147 instance->updateViewport();
00148
00149 // ユーザー定義のコールバック関数の呼び出し
00150 if (instance->resizeFunc) (*instance->resizeFunc)(instance, width, height);
00151 }
00152 }
00153
00154 //

```

```
00155 // キーボードをタイプした時の処理
00156 //
00157 void GgApp::Window::keyboard(GLFWwindow* window, int key, int scancode, int action, int mods)
00158 {
00159 #if defined(IMGUI_VERSION)
00160 // ImGui がキーボードを使うときはキーボードの処理を行わない
00161 if (ImGui::GetIO().WantCaptureKeyboard) return;
00162 #endif
00163
00164 // このインスタンスの this ポインタを得る
00165 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00166
00167 if (instance && action)
00168 {
00169 // ユーザー定義のコールバック関数の呼び出し
00170 if (instance->keyboardFunc) (*instance->keyboardFunc)(instance, key, scancode, action, mods);
00171
00172 // 対象のユーザインターフェース
00173 auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00174
00175 switch (key)
00176 {
00177 case GLFW_KEY_HOME:
00178
00179 // トラックボールを初期化する
00180 instance->resetRotation();
00181 [[fallthrough]];
00182
00183 case GLFW_KEY_END:
00184
00185 // 平行移動量を初期化する
00186 instance->resetTranslation();
00187 break;
00188
00189 case GLFW_KEY_UP:
00190
00191 if (mods & GLFW_MOD_SHIFT)
00192     current_if.arrow[1][1]++;
00193 else if (mods & GLFW_MOD_CONTROL)
00194     current_if.arrow[2][1]++;
00195 else if (mods & GLFW_MOD_ALT)
00196     current_if.arrow[3][1]++;
00197 else
00198     current_if.arrow[0][1]++;
00199 break;
00200
00201 case GLFW_KEY_DOWN:
00202
00203 if (mods & GLFW_MOD_SHIFT)
00204     current_if.arrow[1][1]--;
00205 else if (mods & GLFW_MOD_CONTROL)
00206     current_if.arrow[2][1]--;
00207 else if (mods & GLFW_MOD_ALT)
00208     current_if.arrow[3][1]--;
00209 else
00210     current_if.arrow[0][1]--;
00211 break;
00212
00213 case GLFW_KEY_RIGHT:
00214
00215 if (mods & GLFW_MOD_SHIFT)
00216     current_if.arrow[1][0]++;
00217 else if (mods & GLFW_MOD_CONTROL)
00218     current_if.arrow[2][0]++;
00219 else if (mods & GLFW_MOD_ALT)
00220     current_if.arrow[3][0]++;
00221 else
00222     current_if.arrow[0][0]++;
00223 break;
00224
00225 case GLFW_KEY_LEFT:
00226
00227 if (mods & GLFW_MOD_SHIFT)
00228     current_if.arrow[1][0]--;
00229 else if (mods & GLFW_MOD_CONTROL)
00230     current_if.arrow[2][0]--;
00231 else if (mods & GLFW_MOD_ALT)
00232     current_if.arrow[3][0]--;
00233 else
00234     current_if.arrow[0][0]--;
00235 break;
00236
00237 default:
00238     break;
00239 }
00240
00241 current_if.lastKey = key;
```

```

00242     }
00243 }
00244
00245 // 
00246 // マウスボタンを操作したときの処理
00247 //
00248 void GgApp::Window::mouse(GLFWwindow* window, int button, int action, int mods)
00249 {
00250 #if defined(IMGUI_VERSION)
00251 // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00252 if (ImGui::GetIO().WantCaptureMouse) return;
00253 #endif
00254
00255 // このインスタンスの this ポインタを得る
00256 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00257
00258 // マウスボタンの状態を記録する
00259 assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00260 instance->status[button] = action != GLFW_RELEASE;
00261
00262 if (instance)
00263 {
00264 // ユーザー定義のコールバック関数の呼び出し
00265 if (instance->mouseFunc) (*instance->mouseFunc)(instance, button, action, mods);
00266
00267 // 対象のユーザインターフェース
00268 auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00269
00270 // マウスの現在位置を得る
00271 const auto x{ current_if.mouse[0] };
00272 const auto y{ current_if.mouse[1] };
00273
00274 if (x < 0 || x >= instance->size[0] || y < 0 || y >= instance->size[1]) return;
00275
00276 if (action)
00277 {
00278 // ドラッグ開始
00279 current_if.rotation[button].begin(x, y);
00280 }
00281 else
00282 {
00283 // ドラッグ終了
00284 current_if.translation[button][0] = current_if.translation[button][1];
00285 current_if.rotation[button].end(x, y);
00286 }
00287 }
00288 }
00289
00290 // 
00291 // マウスホイールを操作した時の処理
00292 //
00293 void GgApp::Window::wheel(GLFWwindow* window, double x, double y)
00294 {
00295 #if defined(IMGUI_VERSION)
00296 // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00297 if (ImGui::GetIO().WantCaptureMouse) return;
00298 #endif
00299
00300 // このインスタンスの this ポインタを得る
00301 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00302
00303 if (instance)
00304 {
00305 // ユーザー定義のコールバック関数の呼び出し
00306 if (instance->wheelFunc) (*instance->wheelFunc)(instance, x, y);
00307
00308 // 対象のユーザインターフェース
00309 auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00310
00311 // マウスホイールの回転量の保存
00312 current_if.wheel[0] += static_cast<GLfloat>(x);
00313 current_if.wheel[1] += static_cast<GLfloat>(y);
00314
00315 // マウスによる平行移動量の z 値の更新
00316 const auto z{ current_if.wheel[1] * instance->velocity[2] };
00317 for (auto& t : current_if.translation) t[1][2] = z;
00318 }
00319 }
00320
00321 // 
00322 // Window クラスのコンストラクタ
00323 //
00324 GgApp::Window::Window(const std::string& title, int width, int height, int fullscreen, GLFWwindow*
share) :
00325 window{ nullptr },
00326 size{ width, height },
00327 fboSize{ width, height },

```

```
00328 #if defined(IMGUI_VERSION)
00329     menubarHeight{ 0 },
00330 #endif
00331     aspect{ 1.0f },
00332     velocity{ 1.0f, 1.0f, 0.1f },
00333     status{ false },
00334     interfaceNo{ 0 },
00335     userPointer{ nullptr },
00336     resizeFunc{ nullptr },
00337     keyboardFunc{ nullptr },
00338     mouseFunc{ nullptr },
00339     wheelFunc{ nullptr }
00340 {
00341     // ディスプレイの情報
00342     GLFWmonitor* monitor{ nullptr };
00343
00344     // フルスクリーン表示
00345     if (fullscreen > 0)
00346     {
00347         // 接続されているモニタの数を数える
00348         int mcount;
00349         auto** const monitors{ glfwGetMonitors(&mcount) };
00350
00351         // セカンダリモニタがあればそれを使う
00352         if (fullscreen > mcount) fullscreen = mcount;
00353         monitor = monitors[fullscreen - 1];
00354
00355         // モニタのモードを調べる
00356         const auto* mode{ glfwGetVideoMode(monitor) };
00357
00358         // ウィンドウのサイズをディスプレイのサイズにする
00359         width = mode->width;
00360         height = mode->height;
00361     }
00362
00363     // GLFW のウィンドウを作成する
00364     window = glfwCreateWindow(width, height, title.c_str(), monitor, share);
00365
00366     // ウィンドウが作成できなければエラー
00367     if (!window) throw std::runtime_error("Unable to open the GLFW window.");
00368
00369     // 現在のウィンドウを処理対象にする
00370     glfwMakeContextCurrent(window);
00371
00372     // ゲームグラフィックス特論の都合による初期化を行う
00373     ggInit();
00374
00375     // このインスタンスの this ポインタを記録しておく
00376     glfwSetWindowUserPointer(window, this);
00377
00378     // キーボードを操作した時の処理を登録する
00379     glfwSetKeyCallback(window, keyboard);
00380
00381     // マウスボタンを操作したときの処理を登録する
00382     glfwSetMouseButtonCallback(window, mouse);
00383
00384     // マウスホイール操作時に呼び出す処理を登録する
00385     glfwSetScrollCallback(window, wheel);
00386
00387     // ウィンドウのサイズ変更時に呼び出す処理を登録する
00388     glfwSetFramebufferSizeCallback(window, resize);
00389
00390     // 垂直同期タイミングに合わせる
00391     glfwSwapInterval(1);
00392
00393     // 実際のフレームバッファのサイズを取得する
00394     glfwGetFramebufferSize(window, &width, &height);
00395
00396     // ビューポートと投影変換行列を初期化する
00397     resize(window, width, height);
00398
00399 #if defined(IMGUI_VERSION)
00400     // 最初のウィンドウを開いたとき
00401     static bool firstTime{ true };
00402     if (firstTime)
00403     {
00404         // Setup Platform/Renderer bindings
00405         ImGui_ImplGlfw_InitForOpenGL(window, true);
00406         ImGui_ImplOpenGL3_Init(nullptr);
00407
00408         // 実行済みであることを記録する
00409         firstTime = false;
00410     }
00411 #endif
00412 }
00413
00414 //
```

```

00415 // イベントを取得してループを継続すべきかどうか調べる
00416 //
00417 GgApp::Window::operator bool()
00418 {
00419     // イベントを取り出す
00420     glfwPollEvents();
00421
00422     // ウィンドウを閉じるべきなら false を返す
00423     if (shouldClose()) return false;
00424
00425     // 対象のユーザインターフェース
00426     auto& current_if{ interfaceData[interfaceNo] };
00427
00428 #if defined(IMGUI_VERSION)
00429     // ImGui の新規フレームを作成する
00430     ImGui_ImplOpenGL3_NewFrame();
00431     ImGui_ImplGlfw_NewFrame();
00432
00433     // ImGui の状態を取り出す
00434     const ImGuiIO& io{ ImGui::GetIO() };
00435
00436     // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00437     if (io.WantCaptureMouse) return true;
00438
00439     // マウスの位置を更新する
00440     current_if.mouse = std::array<GLfloat, 2>{ io.MousePos.x, io.MousePos.y };
00441 #else
00442     // マウスの現在位置を調べる
00443     double x, y;
00444     glfwGetCursorPos(window, &x, &y);
00445
00446     // マウスの位置を更新する
00447     current_if.mouse = std::array<GLfloat, 2>{ static_cast<GLfloat>(x), static_cast<GLfloat>(y) };
00448 #endif
00449
00450     // マウスドラッグ
00451     for (int button = GLFW_MOUSE_BUTTON_1; button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT; ++button)
00452     {
00453         // マウスボタンを押していたら
00454         if (status[button])
00455         {
00456             // 現在位置と平行移動量を更新する
00457             current_if.calcTranslation(button, velocity);
00458         }
00459     }
00460
00461     return true;
00462 }
00463
00464 //
00465 // カラーバッファを入れ替える
00466 //
00467 void GgApp::Window::swapBuffers() const
00468 {
00469 #if defined(IMGUI_VERSION)
00470     // ImGui の描画データがあればフレームをレンダリングする
00471     ImDrawData* data{ ImGui::GetDrawData() };
00472     if (data) ImGui_ImplOpenGL3_RenderDrawData(data);
00473 #endif
00474
00475     // エラーチェック
00476     ggError();
00477
00478     // カラーバッファを入れ替える
00479     glfwSwapBuffers(window);
00480 }
00481
00482 //
00483 // ビューポートのサイズを更新する
00484 //
00485 void GgApp::Window::updateViewport()
00486 {
00487     // フレームバッファの大きさを求める
00488     glfwGetFramebufferSize(window, &fboSize[0], &fboSize[1]);
00489
00490 #if defined(IMGUI_VERSION)
00491     // フレームバッファの高さからメニューバーの高さを減じる
00492     fboSize[1] -= menubarHeight;
00493 #endif
00494
00495     // ウィンドウの縦横比を保存する
00496     aspect = static_cast<GLfloat>(fboSize[0]) / static_cast<GLfloat>(fboSize[1]);
00497
00498     // ビューポートを設定する
00499     restoreViewport();
00500 }
00501

```

```
00502 #if defined(GG_USE_OCUlus_RIFT)
00503 # if OVR_PRODUCT_VERSION > 0
00504 //
00505 // グラフィックスカードのデフォルトの LUID を得る
00506 //
00507 ovrGraphicsLuid GgApp::Oculus::GetDefaultAdapterLuid()
00508 {
00509     ovrGraphicsLuid luid = ovrGraphicsLuid();
00510
00511 # if defined(_MSC_VER)
00512     IDXGIFactory* factory{ nullptr };
00513
00514     if (SUCCEEDED(CreateDXGIFactory(IID_PPV_ARGS(&factory))))
00515     {
00516         IDXGIAdapter* adapter{ nullptr };
00517
00518         if (SUCCEEDED(factory->EnumAdapters(0, &adapter)))
00519         {
00520             DXGI_ADAPTER_DESC desc;
00521
00522             adapter->GetDesc(&desc);
00523             memcpy(&luid, &desc.AdapterLuid, sizeof luid);
00524             adapter->Release();
00525         }
00526
00527         factory->Release();
00528     }
00529 # endif
00530
00531     return luid;
00532 }
00533
00534 //
00535 // グラフィックスカードの LUID の比較
00536 //
00537 int GgApp::Oculus::Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs)
00538 {
00539     return memcmp(&lhs, &rhs, sizeof(ovrGraphicsLuid));
00540 }
00541 # endif
00542
00543 //
00544 // コンストラクタ
00545 //
00546 GgApp::Oculus::Oculus() :
00547     session{ nullptr },
00548     oculusFbo{ 0 },
00549     screen{ -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, },
00550     mirrorFbo{ 0 },
00551     window{ nullptr },
00552 # if OVR_PRODUCT_VERSION > 0
00553     frameIndex{ 0 },
00554     oculusDepth{ 0 },
00555     mirrorWidth{ 1280 },
00556     mirrorHeight{ 640 },
00557 # endif
00558     mirrorTexture{ nullptr }
00559 {
00560 }
00561
00562 //
00563 // Oculus Rift のセッションを作成する
00564 //
00565 GgApp::Oculus& GgApp::Oculus::initialize(const Window& window)
00566 {
00567     // Oculus Rift のコンテキスト
00568     static Oculus oculus;
00569
00570     // 既に Oculus Rift のセッションが作成されていたら参照を返す
00571     if (oculus.session) return oculus;
00572
00573     // 最初に呼び出したときだけ実行する
00574     static bool firstTime{ true };
00575     if (firstTime)
00576     {
00577         // Oculus Rift (LibOVR) を初期化する
00578         ovrInitParams initParams{ ovrInit_RequestVersion, OVR_MINOR_VERSION, NULL, 0, 0 };
00579         if (OVR_FAILURE(ovr_Initialize(&initParams)))
00580             throw std::runtime_error("Can't initialize LibOVR");
00581
00582         // アプリケーションの終了時に LibOVR を終了する
00583         atexit(ovr_Shutdown);
00584
00585         // 実行済みであることを記録する
00586         firstTime = false;
00587     }
00588 }
```

```

00589 // Oculus Rift のセッションを作成する
00590 ovrGraphicsLuid luid;
00591 if (OVR_FAILURE(ovr.Create(&oculus.session, &luid)))
00592     throw std::runtime_error("Can't create Oculus Rift session");
00593
00594 # if OVR_PRODUCT_VERSION > 0
00595 // デフォルトのグラフィックスアダプタが使われているか確かめる
00596 if (Compare(luid, GetDefaultAdapterLuid()))
00597     throw std::runtime_error("Graphics adapter is not default");
00598 # endif
00599
00600 // session が無効ならエラー
00601 if (!oculus.session) std::runtime_error("Unable to use the Oculus Rift.");
00602
00603 // ミラー表示を行うウィンドウを設定する
00604 oculus.window = &window;
00605
00606 // Oculus Rift の情報を取り出す
00607 oculus.hmdDesc = ovr_GetHmdDesc(oculus.session);
00608
00609 # if defined(_DEBUG)
00610 // Oculus Rift の情報を表示する
00611 std::cerr
00612     << "\nProduct name: " << oculus.hmdDesc.ProductName
00613     << "\nResolution: " << oculus.hmdDesc.Resolution.w << " x " << oculus.hmdDesc.Resolution.h
00614     << "\nDefault Fov: (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].LeftTan
00615     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].DownTan
00616     << ")" - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].RightTan
00617     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].UpTan
00618     << ") \n        (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].LeftTan
00619     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].DownTan
00620     << ")" - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].RightTan
00621     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].UpTan
00622     << ") \nMaximum Fov: (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].LeftTan
00623     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].DownTan
00624     << ")" - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].RightTan
00625     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].UpTan
00626     << ") \n        (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].LeftTan
00627     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].DownTan
00628     << ")" - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].RightTan
00629     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].UpTan
00630     << ") \n" << std::endl;
00631 # endif
00632
00633 // Oculus Rift に転送する描画データを作成する
00634 # if OVR_PRODUCT_VERSION > 0
00635 oculus.layerData.Header.Type = ovrLayerType_EyeFov;
00636 # else
00637 oculus.layerData.Header.Type = ovrLayerType_EyeFovDepth;
00638 # endif
00639
00640 // OpenGL なので左下が原点
00641 oculus.layerData.Header.Flags = ovrLayerFlag_TextureOriginAtBottomLeft;
00642
00643 // Oculus Rift のレンダリングに使う FBO を作成する
00644 glGenFramebuffers(ovrEye_Count, oculus.oculusFbo);
00645
00646 // 全ての目について
00647 for (int eye = 0; eye < ovrEye_Count; ++eye)
00648 {
00649     // Oculus Rift の視野を取得する
00650     const auto& fov{ oculus.hmdDesc.DefaultEyeFov[ovrEyeType(eye)] };
00651
00652     // Oculus Rift 用の FBO のサイズを求める
00653     const auto textureSize{ ovr_GetFovTextureSize(oculus.session, ovrEyeType(eye), fov, 1.0f) };
00654
00655     // Oculus Rift のスクリーンのサイズを保存する
00656     oculus.screen[eye][0] = -fov.LeftTan;
00657     oculus.screen[eye][1] = fov.RightTan;
00658     oculus.screen[eye][2] = -fov.DownTan;
00659     oculus.screen[eye][3] = fov.UpTan;
00660
00661 # if OVR_PRODUCT_VERSION > 0
00662
00663     // 描画データに視野を設定する
00664     oculus.layerData.Fov[eye] = fov;
00665
00666     // 描画データにビューポートを設定する
00667     oculus.layerData.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00668     oculus.layerData.Viewport[eye].Size = textureSize;
00669
00670     // Oculus Rift 用の FBO のカラー・バッファとして使うテクスチャセットの特性
00671     const ovrTextureSwapChainDesc colorDesc
00672     {
00673         ovrTexture_2D,                                // Type
00674         OVR_FORMAT_R8G8B8A8_UNORM_SRGB,               // Format
00675         1,                                         // ArraySize

```

```

00676     textureSize.w,                      // Width
00677     textureSize.h,                      // Height
00678     1,                                // MipLevels
00679     1,                                // SampleCount
00680     ovrFalse,                          // StaticImage
00681     0, 0
00682 };
00683
00684 // Oculus Rift 用の FBO のレンダーターゲットとして使うテクスチャチェインを作成する
00685 oculus.layerData.ColorTexture[eye] = nullptr;
00686 if (OVR_SUCCESS(ovr_CreateTextureSwapChainGL(oculus.session, &colorDesc,
00687 &oculus.layerData.ColorTexture[eye])))
00688 {
00689     // 作成したテクスチャチェインの長さを取得する
00690     int length(0);
00691     if (OVR_SUCCESS(ovr_GetTextureSwapChainLength(oculus.session, oculus.layerData.ColorTexture[eye],
00692 &length)))
00693     {
00694         // テクスチャチェインの個々の要素について
00695         for (int i = 0; i < length; ++i)
00696         {
00697             // テクスチャのパラメータを設定する
00698             GLuint texId{ 0 };
00699             ovr_GetTextureSwapChainBufferGL(oculus.session, oculus.layerData.ColorTexture[eye], i,
00700 &texId);
00701             glBindTexture(GL_TEXTURE_2D, texId);
00702             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00703             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00704             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00705             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00706         }
00707
00708         // Oculus Rift 用の FBO のデプスバッファとして使うテクスチャを作成する
00709         glGenTextures(1, oculus.oculusDepth + eye);
00710         glBindTexture(GL_TEXTURE_2D, oculus.oculusDepth[eye]);
00711         glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h, 0,
00712 GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
00713         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00714         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00715         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00716         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00717
00718 # else
00719     // 描画データに視野を設定する
00720     oculus.layerData.EyeFov.Fov[eye] = fov;
00721
00722     // 描画データにビューポートを設定する
00723     oculus.layerData.EyeFov.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00724     oculus.layerData.EyeFov.Viewport[eye].Size = textureSize;
00725
00726     // Oculus Rift 用の FBO のカラーバッファとして使うテクスチャセットを作成する
00727     ovrSwapTextureSet* colorTexture{ nullptr };
00728     ovr_CreateSwapTextureSetGL(oculus.session, GL_SRGB8_ALPHA8, textureSize.w, textureSize.h,
00729 &colorTexture);
00730     oculus.layerData.EyeFov.ColorTexture[eye] = colorTexture;
00731
00732     // Oculus Rift 用の FBO のデプスバッファとして使うテクスチャセットを作成する
00733     ovrSwapTextureSet* depthTexture{ nullptr };
00734     ovr_CreateSwapTextureSetGL(oculus.session, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h,
00735 &depthTexture);
00736     oculus.layerData.EyeFovDepth.DepthTexture[eye] = depthTexture;
00737
00738 # endif
00739 }
00740
00741 # if OVR_PRODUCT_VERSION > 0
00742
00743     // 姿勢のトラッキングにおける床の高さを 0 に設定する
00744     ovr_SetTrackingOriginType(oculus.session, ovrTrackingOrigin_FloorLevel);
00745
00746     // ミラー表示用の FBO を作成する
00747     const GLsizei* size{ oculus.window->getSize() };
00748     const ovrMirrorTextureDesc mirrorDesc
00749     {
00750         OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00751         oculus.mirrorWidth = size[0],   // Width
00752         oculus.mirrorHeight = size[1],  // Height
00753         0                           // Flags
00754     };
00755
00756     // ミラー表示用の FBO のカラーバッファとして使うテクスチャを作成する

```

```

00757 if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, &mirrorDesc, &oculus.mirrorTexture)))
00758 {
00759     // 作成したテクスチャのテクスチャ名を得る
00760     GLuint texId{ 0 };
00761     if (OVR_SUCCESS(ovr_GetMirrorTextureBufferGL(oculus.session, oculus.mirrorTexture, &texId)))
00762     {
00763         // ミラー表示用の FBO を作成してテクスチャをカラーバッファとして組み込む
00764         glGenFramebuffers(1, &oculus.mirrorFbo);
00765         glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00766         glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texId, 0);
00767         glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
00768         glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00769     }
00770 }
00771
00772 # else
00773
00774 // 作成したテクスチャのテクスチャ名を得る
00775 if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, GL_SRGB8_ALPHA8, width, height,
00776 reinterpret_cast<ovrTexture*>(&mirrorTexture)))
00777 {
00778     // ミラー表示用の FBO を作成してテクスチャをカラーバッファとして組み込む
00779     oculus.mirrorFbo = 0;
00780     glGenFramebuffers(1, &oculus.mirrorFbo);
00781     glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00782     glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
00783     mirrorTexture->OGL.TexId, 0);
00784     glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
00785     glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00786 }
00787
00788 // Oculus Rift にレンダリングするときは sRGB カラースペースを使う
00789 glEnable(GL_FRAMEBUFFER_SRGB);
00790
00791 // フロントバッファに描く
00792 glDrawBuffer(GL_FRONT);
00793
00794 // Oculus Rift への表示では垂直同期タイミングに合わせない
00795 glfwSwapInterval(0);
00796
00797 return oculus;
00798 }
00799
00800 //
00801 // Oculus Rift のセッションを破棄する
00802 //
00803 void GgApp::Oculus::terminate()
00804 {
00805     // session が無効なら何もしない
00806     if (!session) return;
00807
00808     // ミラー表示用の FBO を作っていたら削除する
00809     if (mirrorFbo)
00810     {
00811         glDeleteFramebuffers(1, &mirrorFbo);
00812         mirrorFbo = 0;
00813     }
00814
00815     // ミラー表示用の FBO のカラーバッファ用のテクスチャを作っていたら削除する
00816     if (mirrorTexture)
00817     {
00818 # if OVR_PRODUCT_VERSION > 0
00819         ovr_DestroyMirrorTexture(session, mirrorTexture);
00820 # else
00821         glDeleteTextures(1, &mirrorTexture->OGL.TexId);
00822         ovr_DestroyMirrorTexture(session, reinterpret_cast<ovrTexture*>(mirrorTexture));
00823 # endif
00824         mirrorTexture = nullptr;
00825     }
00826
00827     // 全ての目について
00828     for (int eye = 0; eye < ovrEye_Count; ++eye)
00829     {
00830         // Oculus Rift へのレンダリング用の FBO を削除する
00831         glDeleteFramebuffers(1, oculusFbo + eye);
00832         oculusFbo[eye] = 0;
00833
00834 # if OVR_PRODUCT_VERSION > 0
00835
00836         // レンダリングターゲットを使ったテクスチャを削除する
00837         if (layerData.ColorTexture[eye])
00838         {
00839             ovr_DestroyTextureSwapChain(session, layerData.ColorTexture[eye]);
00840             layerData.ColorTexture[eye] = nullptr;
00841         }

```

```

00842     // デブスバッファとして使ったテクスチャを削除する
00843     glDeleteTextures(1, oculusDepth + eye);
00844     oculusDepth[eye] = 0;
00845
00846 # else
00847
00848     // レンダリングターゲットに使ったテクスチャを削除する
00849     auto* const colorTexture(layerData.EyeFov.ColorTexture[eye]);
00850     for (int i = 0; i < colorTexture->TextureCount; ++i)
00851     {
00852         const auto* const ctex(reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[i]));
00853         glDeleteTextures(1, &ctex->OGL.TexId);
00854     }
00855     ovr_DestroySwapTextureSet(session, colorTexture);
00856
00857     // デブスバッファとして使ったテクスチャを削除する
00858     auto* const depthTexture(layerData.EyeFovDepth.DepthTexture[eye]);
00859     for (int i = 0; i < depthTexture->TextureCount; ++i)
00860     {
00861         const auto* const dtex(reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[i]));
00862         glDeleteTextures(1, &dtex->OGL.TexId);
00863     }
00864     ovr_DestroySwapTextureSet(session, depthTexture);
00865
00866 # endif
00867 }
00868
00869 // Oculus Rift のセッションを破棄する
00870 ovr_Destroy(session);
00871 session = nullptr;
00872
00873 // カラースペースを元に戻す
00874 glDisable(GL_FRAMEBUFFER_SRGB);
00875
00876 // バックバッファに描く
00877 glDrawBuffer(GL_BACK);
00878
00879 // 垂直同期タイミングに合わせる
00880 glfwSwapInterval(1);
00881
00882 }
00883
00884 //
00885 // Oculus Rift による描画開始
00886 //
00887 bool GgApp::Oculus::begin()
00888 {
00889 # if OVR_PRODUCT_VERSION > 0
00890
00891     // セッションの状態を取得する
00892     ovrSessionStatus sessionStatus;
00893     ovr_GetSessionStatus(session, &sessionStatus);
00894
00895     // アプリケーションが終了を要求しているときはウィンドウのクローズフラグを立てる
00896     if (sessionStatus.ShouldQuit) window->setClose(GLFW_TRUE);
00897
00898     // Oculus Rift に表示されていないときは戻る
00899     if (!sessionStatus.IsVisible) return false;
00900
00901     // 現在の状態をトラッキングの原点にする
00902     if (sessionStatus.ShouldRecenter) ovr_RecenterTrackingOrigin(session);
00903
00904     // HmdToEyeOffset などは実行時に変化するので毎フレーム ovr_GetRenderDesc() で ovrEyeRenderDesc を取得する
00905     const ovrEyeRenderDesc eyeRenderDesc[]
00906     {
00907         ovr_GetRenderDesc(session, ovrEyeType(0), hmdDesc.DefaultEyeFov[0]),
00908         ovr_GetRenderDesc(session, ovrEyeType(1), hmdDesc.DefaultEyeFov[1])
00909     };
00910
00911     // Oculus Rift のスクリーンのヘッドトラッキング位置からの変位を取得する
00912     const ovrPose hmdToEyePose[]
00913     {
00914         eyeRenderDesc[0].HmdToEyePose,
00915         eyeRenderDesc[1].HmdToEyePose
00916     };
00917
00918     // 視点の姿勢情報を取得する
00919     ovr.GetEyePoses(session, frameIndex, ovrTrue, hmdToEyePose, layerData.RenderPose,
00920     &layerData.SensorSampleTime);
00921 # else
00922
00923     // フレームのタイミング計測開始
00924     const auto ftiming(ovr_GetPredictedDisplayTime(session, 0));
00925
00926     // sensorSampleTime の取得は可能な限り ovr_GetTrackingState() の近くで行う
00927     layerData.EyeFov.SensorSampleTime = ovr_GetTimeInSeconds();

```

```

00928
00929 // ヘッドトラッキングの状態を取得する
00930 const auto hmdState.ovr_GetTrackingState(session, ftiming, ovrTrue));
00931
00932 // Oculus Rift のスクリーンのヘッドトラッキング位置からの変位を取得する
00933 const ovrVector3f hmdToEyeViewOffset[]
00934 {
00935     eyeRenderDesc[0].HmdToEyeViewOffset,
00936     eyeRenderDesc[1].HmdToEyeViewOffset
00937 };
00938
00939 // 視点の姿勢情報を求める
00940 ovr_CalcEyePoses(hmdState.HeadPose.ThePose, hmdToEyeViewOffset, eyePose);
00941
00942 # endif
00943
00944 return true;
00945 }
00946
00947 //
00948 // Oculus Rift の描画する目の指定
00949 //
00950 void GgApp::Oculus::select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation)
00951 {
00952 # if OVR_PRODUCT_VERSION > 0
00953
00954 // Oculus Rift にレンダリングする FBO に切り替える
00955 if (layerData.ColorTexture[eye])
00956 {
00957     // FBO のカラーバッファに使う現在のテクスチャのインデックスを取得する
00958     int curIndex;
00959     ovr_GetTextureSwapChainCurrentIndex(session, layerData.ColorTexture[eye], &curIndex);
00960
00961     // FBO のカラーバッファに使うテクスチャを取得する
00962     GLuint curTexId;
00963     ovr_GetTextureSwapChainBufferGL(session, layerData.ColorTexture[eye], curIndex, &curTexId);
00964
00965     // FBO を設定する
00966     glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
00967     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, curTexId, 0);
00968     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, oculusDepth[eye], 0);
00969
00970     // ビューポートを設定する
00971     const auto& vp{ layerData.Viewport[eye] };
00972     glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
00973 }
00974
00975 // Oculus Rift の片目の位置と回転を取得する
00976 const auto& p{ layerData.RenderPose[eye].Position };
00977 const auto& o{ layerData.RenderPose[eye].Orientation };
00978
00979 # else
00980
00981 // レンダーターゲットに描画する前にレンダーターゲットのインデックスをインクリメントする
00982 auto* const colorTexture{ layerData.EyeFov.ColorTexture[eye] };
00983 colorTexture->CurrentIndex = (colorTexture->CurrentIndex + 1) % colorTexture->TextureCount;
00984 auto* const depthTexture{ layerData.EyeFovDepth.DepthTexture[eye] };
00985 depthTexture->CurrentIndex = (depthTexture->CurrentIndex + 1) % depthTexture->TextureCount;
00986
00987 // レンダーターゲットを切り替える
00988 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
00989 const auto& ctex{
00990     reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[colorTexture->CurrentIndex]) };
00991     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ctex->OGL.TexId, 0);
00992 const auto& dtex{
00993     reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[depthTexture->CurrentIndex]) };
00994     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, dtex->OGL.TexId, 0);
00995
00996 // ビューポートを設定する
00997 const auto& vp{ layerData.EyeFov.Viewport[eye] };
00998     glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
00999
01000 // Oculus Rift の片目の位置と回転を取得する
01001 const auto& p{ eyePose[eye].Position };
01002 const auto& o{ eyePose[eye].Orientation };
01003
01004 // Oculus Rift のスクリーンの大きさを返す
01005 screen[0] = this->screen[eye][0];
01006 screen[1] = this->screen[eye][1];
01007 screen[2] = this->screen[eye][2];
01008 screen[3] = this->screen[eye][3];
01009
01010 // Oculus Rift の位置を返す
01011 position[0] = p.x;
01012 position[1] = p.y;

```

```
01013     position[2] = p.z;
01014
01015     // Oculus Rift の方向を返す
01016     orientation[0] = o.x;
01017     orientation[1] = o.y;
01018     orientation[2] = o.z;
01019     orientation[3] = o.w;
01020 }
01021
01022 //
01023 // Time Warp 处理に使う投影変換行列の成分の設定 (DK1, DK2)
01024 //
01025 void GgApp::Oculus::timewarp(const GgMatrix& projection)
01026 {
01027 # if OVR_PRODUCT_VERSION < 1
01028     // TimeWarp に使う変換行列の成分を設定する
01029     auto& posTimewarpProjectionDesc{ layerData.EyeFovDepth.ProjectionDesc };
01030     posTimewarpProjectionDesc.Projection22 = (projection.get()[4 * 2 + 2] + projection.get()[4 * 3 + 2]) * 0.5f;
01031     posTimewarpProjectionDesc.Projection23 = projection.get()[4 * 2 + 3] * 0.5f;
01032     posTimewarpProjectionDesc.Projection32 = projection.get()[4 * 3 + 2];
01033 # endif
01034 }
01035
01036 //
01037 // 図形の描画を完了する (CV1 以降)
01038 //
01039 void GgApp::Oculus::commit(int eye)
01040 {
01041 # if OVR_PRODUCT_VERSION > 0
01042     // GL_COLOR_ATTACHMENT0 に割り当てられたテクスチャが wglDXUnlockObjectsNV() によって
01043     // アンロックされるために次のフレームの処理において無効な GL_COLOR_ATTACHMENT0 が
01044     // FBO に結合されるのを避ける
01045     glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
01046     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
01047     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, 0, 0);
01048
01049     // 保留中の変更を layerData.ColorTexture[eye] に反映しインデックスを更新する
01050     ovrCommitTextureSwapChain(session, layerData.ColorTexture[eye]);
01051 # endif
01052 }
01053
01054 //
01055 // フレームを転送する
01056 //
01057 bool GgApp::Oculus::submit(bool mirror)
01058 {
01059     // エラーチェック
01060     ggError();
01061
01062 # if OVR_PRODUCT_VERSION > 0
01063     // 描画データを Oculus Rift に転送する
01064     const auto* const layers{ &layerData.Header };
01065     if (OVR_FAILURE(ovrSubmitFrame(session, frameIndex++, nullptr, &layers, 1))) return false;
01066 # else
01067     // Oculus Rift 上の描画位置と拡大率を求める
01068     ovrViewScaleDesc viewScaleDesc;
01069     viewScaleDesc.HmdSpaceToWorldScaleInMeters = 1.0f;
01070     viewScaleDesc.HmdToEyeViewOffset[0] = eyeRenderDesc[0].HmdToEyeViewOffset;
01071     viewScaleDesc.HmdToEyeViewOffset[1] = eyeRenderDesc[1].HmdToEyeViewOffset;
01072
01073     // 描画データを更新する
01074     layerData.EyeFov.RenderPose[0] = eyePose[0];
01075     layerData.EyeFov.RenderPose[1] = eyePose[1];
01076
01077     // 描画データを Oculus Rift に転送する
01078     const auto* const layers{ &layerData.Header };
01079     if (OVR_FAILURE(ovrSubmitFrame(session, 0, &viewScaleDesc, &layers, 1))) return false;
01080 # endif
01081
01082     // ミラー表示
01083     if (mirror)
01084     {
01085 # if OVR_PRODUCT_VERSION > 0
01086         const auto& sx1{ mirrorWidth };
01087         const auto& sy1{ mirrorHeight };
01088 # else
01089         const auto& sx1{ mirrorTexture->OGL.Header.TextureSize.w };
01090         const auto& sy1{ mirrorTexture->OGL.Header.TextureSize.h };
01091 # endif
01092
01093     // ミラー表示のウインドウのサイズ
01094     GLsizei size[2];
01095     window->getSize(size);
01096
01097     // ミラー表示の表示領域
01098     GLint dx0{ 0 }, dx1{ size[0] }, dy0{ 0 }, dy1{ size[1] };
```

```

01099
01100 // ミラー表示がウインドウからはみ出ないようにする
01101 if ((size[0] *= syl) < (size[1] *= sx1))
01102 {
01103     const GLint tyl{ size[0] / sx1 };
01104     dy0 = (dy1 - tyl) / 2;
01105     dy1 = dy0 + tyl;
01106 }
01107 else
01108 {
01109     const GLint tx1{ size[1] / syl };
01110     dx0 = (dx1 - tx1) / 2;
01111     dx1 = dx0 + tx1;
01112 }
01113
01114 // レンダリング結果をミラー表示用のフレームバッファにも転送する
01115 glBindFramebuffer(GL_READ_FRAMEBUFFER, mirrorFbo);
01116 glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
01117 glBlitFramebuffer(0, syl, sx1, 0, dx0, dy0, dx1, dy1, GL_COLOR_BUFFER_BIT, GL_NEAREST);
01118 glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
01119
01120 // 残っている OpenGL コマンドを実行する
01121 glFlush();
01122 }
01123
01124 return true;
01125 }
01126 #endif

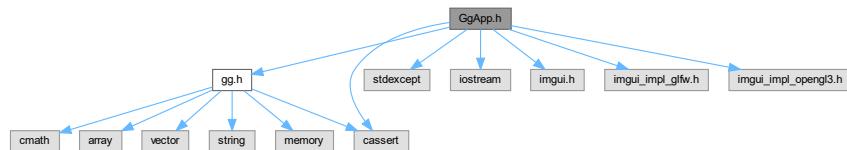
```

## 9.39 GgApp.h ファイル

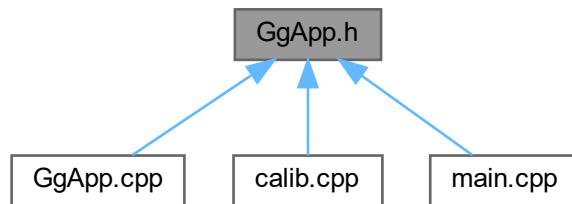
```

#include "gg.h"
#include <cassert>
#include <stdexcept>
#include <iostream>
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"
GgApp.h の依存先関係図:

```



被依存関係図:



## クラス

- class [GgApp](#)
- class [GgApp::Window](#)

## マクロ定義

- `#define GG_USE_IMGUI`
- `#define GG_BUTTON_COUNT 3`
- `#define GG_INTERFACE_COUNT 5`

### 9.39.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラスの定義

著者

Kohe Tokoi

日付

November 22, 2022

[GgApp.h](#) に定義がります。

### 9.39.2 マクロ定義詳解

#### 9.39.2.1 GG\_BUTTON\_COUNT

```
#define GG_BUTTON_COUNT 3
```

[GgApp.h](#) の 19 行目に定義がります。

#### 9.39.2.2 GG\_INTERFACE\_COUNT

```
#define GG_INTERFACE_COUNT 5
```

[GgApp.h](#) の 24 行目に定義がります。

### 9.39.2.3 GG\_USE\_IMGUI

```
#define GG_USE_IMGUI
```

GgApp.h の 12 行目に定義があります。

## 9.40 GgApp.h

### [詳解]

```
00001 #pragma once
00002
00010
00011 // Dear ImGui を使うなら
00012 #define GG_USE_IMGUI
00013
00014 // Oculus Rift を使うなら
00015 // #define GG_USE_OOCULUS_RIFT
00016
00017 // 使用するマウスのボタン数
00018 #if !defined(GG_BUTTON_COUNT)
00019 # define GG_BUTTON_COUNT 3
00020 #endif
00021
00022 // 使用するユーザインターフェースの数
00023 #if !defined(GG_INTERFACE_COUNT)
00024 # define GG_INTERFACE_COUNT 5
00025 #endif
00026
00027 // 補助プログラム
00028 #include "gg.h"
00029 using namespace gg;
00030
00031 // 標準ライブラリ
00032 #include <cassert>
00033 #include <stdexcept>
00034 #include <iostream>
00035
00036 // ImGui の組み込み
00037 #if defined(GG_USE_IMGUI)
00038 # include "imgui.h"
00039 # include "imgui_impl_glfw.h"
00040 # include "imgui_impl_opengl3.h"
00041 #endif
00042
00043 // Oculus Rift SDK ライブライ (LibOVR) の組み込み
00044 #if defined(GG_USE_OOCULUS_RIFT)
00045 # if defined(_MSC_VER)
00046 # define GLFW_EXPOSE_NATIVE_WIN32
00047 # define GLFW_EXPOSE_NATIVE_WGL
00048 # include <GLFW/glfw3native.h>
00049 # define OVR_OS_WIN32
00050 # undef APIENTRY
00051 # pragma comment(lib, "LibOVR.lib")
00052 # endif
00053 # include <OVR_CAPI_GL.h>
00054 # include <Extras/OVR_Math.h>
00055 # if OVR_PRODUCT_VERSION > 0
00056 # include <dxgi.h> // GetDefaultAdapterLuid のため
00057 # pragma comment(lib, "dxgi.lib")
00058 # endif
00059 #endif
00060
00061
00062 class GgApp
00063 {
00064 public:
00065
00066     GgApp(int major = 0, int minor = 1);
00067
00068     GgApp(const GgApp& w) = delete;
00069
00070     virtual ~GgApp();
00071
00072     GgApp& operator=(const GgApp& w) = delete;
00073
00074     int main(int argc, const char* const* argv);
00075
00076     class Window
00077     {
```

```
00104 // ウィンドウの識別子
00105 GLFWwindow* window;
00106
00107 // ビューポートの横幅と高さ
00108 std::array<GLsizei, 2> size;
00109
00110 // フレームバッファの横幅と高さ
00111 std::array<GLsizei, 2> fboSize;
00112
00113 #if defined(IMGUI_VERSION)
00114 // メニューバーの高さ
00115 GLsizei menuBarHeight;
00116 #endif
00117
00118 // ビューポートの縦横比
00119 GLfloat aspect;
00120
00121 // マウスの移動速度[X/Y/Z]
00122 std::array<GLfloat, 3> velocity;
00123
00124 // マウスボタンの状態
00125 std::array<bool, GG_BUTTON_COUNT> status;
00126
00127 // ユーザインターフェースのデータ構造
00128 struct HumanInterface
00129 {
00130     // 最後にタイプしたキー
00131     int lastKey;
00132
00133     // 矢印キー
00134     std::array<std::array<int, 2>, 4> arrow;
00135
00136     // マウスの現在位置
00137     std::array<GLfloat, 2> mouse;
00138
00139     // マウスホイールの回転量
00140     std::array<GLfloat, 2> wheel;
00141
00142     // 平行移動量[ボタン][直前/更新][X/Y/Z]
00143     std::array<std::array<GGVector, 2>, GG_BUTTON_COUNT> translation;
00144
00145     // トランクボール
00146     std::array<GGTrackball, GG_BUTTON_COUNT> rotation;
00147
00148     // コンストラクタ
00149     HumanInterface() :
00150         lastKey{ 0 },
00151         arrow{},
00152         mouse{},
00153         wheel{},
00154         translation{}
00155     {
00156         resetTranslation();
00157     }
00158
00159     //
00160     // マウスや矢印キーによる平行移動量を初期化する
00161     //
00162     void resetTranslation();
00163
00164     //
00165     // 平行移動量と回転量を更新する (X, Y のみ, Z は wheel() で計算する)
00166     //
00167     void calcTranslation(int button, const std::array<GLfloat, 3>& velocity);
00168 };
00169
00170 // ヒューマンインターフェースデバイスのデータ
00171 std::array<HumanInterface, GG_INTERFACE_COUNT> interfaceData;
00172
00173 // ヒューマンインターフェースデバイスの番号
00174 int interfaceNo;
00175
00176 //
00177 // ユーザー定義のコールバック関数へのポインタ
00178 //
00179 void* userPointer;
00180 void (*resizeFunc)(const Window* window, int width, int height);
00181 void (*keyboardFunc)(const Window* window, int key, int scanCode, int action, int mods);
00182 void (*mouseFunc)(const Window* window, int button, int action, int mods);
00183 void (*wheelFunc)(const Window* window, double x, double y);
00184
00185 //
00186 // ウィンドウのサイズ変更時の処理
00187 //
00188 static void resize(GLFWwindow* window, int width, int height);
00189
00190 //
```

```
00191 // キーボードをタイプした時の処理
00192 //
00193 static void keyboard(GLFWwindow* window, int key, int scancode, int action, int mods);
00194 //
00195 // マウスボタンを操作したときの処理
00196 //
00197 static void mouse(GLFWwindow* window, int button, int action, int mods);
00198 //
00199 // マウスホイールを操作した時の処理
00200 //
00201 static void wheel(GLFWwindow* window, double x, double y);
00202 //
00203 public:
00204
00205     Window(const std::string& title = "GLFW Window", int width = 640, int height = 480,
00206            int fullscreen = 0, GLFWwindow* share = nullptr);
00207
00208     Window(const Window& w) = delete;
00209
00210     virtual ~Window()
00211     {
00212         // ウィンドウが作成されていなければ戻る
00213         if (!window) return;
00214
00215         // ウィンドウを破棄する
00216         glfwDestroyWindow(window);
00217     }
00218
00219     Window& operator=(const Window& w) = delete;
00220
00221     auto* get() const
00222     {
00223         return window;
00224     }
00225
00226     void setClose(int flag = GLFW_TRUE) const
00227     {
00228         glfwSetWindowShouldClose(window, flag);
00229     }
00230
00231     bool shouldClose() const
00232     {
00233         // ウィンドウを閉じるべきなら true を返す
00234         return glfwWindowShouldClose(window) != GLFW_FALSE;
00235     }
00236
00237     explicit operator bool();
00238
00239     void swapBuffers() const;
00240
00241     void restoreViewport() const
00242     {
00243         if (!glfwGetWindowAttrib(window, GLFW_ICONIFIED)) glViewport(0, 0, fboSize[0], fboSize[1]);
00244     }
00245
00246     void updateViewport();
00247
00248 #if defined(IMGUI_VERSION)
00249     void setMenubarHeight(GLsizei height)
00250     {
00251         // メニューバーの高さを保存する
00252         menubarHeight = height;
00253
00254         // ビューポートを復帰する
00255         updateViewport();
00256     }
00257 #endif
00258
00259     auto getWidth() const
00260     {
00261         return size[0];
00262     }
00263
00264     auto getHeight() const
00265     {
00266         return size[1];
00267     }
00268
00269     auto getFboWidth() const
00270     {
00271         return fboSize[0];
00272     }
00273
00274     auto getFboHeight() const
00275     {
```

```
00350     return fboSize[1];
00351 }
00352
00353     const auto& getSize() const
00354 {
00355     return size;
00356 }
00357
00358     const auto& getFboSize() const
00359 {
00360     return fboSize;
00361 }
00362
00363     auto getAspect() const
00364 {
00365     return aspect;
00366 }
00367
00368     bool getKey(int key) const
00369 {
00370 #if defined(IMGUI_VERSION)
00371     // ImGui がキーボードを使うときはキーボードの処理を行わない
00372     if (ImGui::GetIO().WantCaptureKeyboard) return false;
00373 #endif
00374
00375     return glfwGetKey(window, key) != GLFW_RELEASE;
00376 }
00377
00378     void selectInterface(int no)
00379 {
00380     assert(static_cast<size_t>(no) < interfaceData.size());
00381     interfaceNo = no;
00382 }
00383
00384     void setVelocity(GLfloat vx, GLfloat vy, GLfloat vz = 0.1f)
00385 {
00386     velocity = std::array<GLfloat, 3>{ vx, vy, vz };
00387 }
00388
00389     int getLastKey()
00390 {
00391     auto& current_if{ interfaceData[interfaceNo] };
00392     const int key{ current_if.lastKey };
00393     current_if.lastKey = 0;
00394     return key;
00395 }
00396
00397     auto getArrow(int direction = 0, int mods = 0) const
00398 {
00399     const auto& current_if{ interfaceData[interfaceNo] };
00400     return static_cast<GLfloat>(current_if.arrow[mods & 3][direction & 1]);
00401 }
00402
00403     auto getArrowX(int mods = 0) const
00404 {
00405     return getArrow(0, mods);
00406 }
00407
00408     auto getArrowY(int mods = 0) const
00409 {
00410     return getArrow(1, mods);
00411 }
00412
00413     void getArrow(GLfloat* arrow, int mods = 0) const
00414 {
00415     arrow[0] = getArrowX(mods);
00416     arrow[1] = getArrowY(mods);
00417 }
00418
00419     auto getShiftArrowX() const
00420 {
00421     return getArrow(0, 1);
00422 }
00423
00424     auto getShiftArrowY() const
00425 {
00426     return getArrow(1, 1);
00427 }
00428
00429     void getShiftArrow(GLfloat* shift_arrow) const
00430 {
00431     shift_arrow[0] = getShiftArrowX();
00432     shift_arrow[1] = getShiftArrowY();
00433 }
00434
00435     auto getControlArrowX() const
00436 {
```

```
00519     return getArrow(0, 2);
00520 }
00521
00527 auto getControlArrowY() const
00528 {
00529     return getArrow(1, 2);
00530 }
00531
00537 void getControlArrow(GLfloat* control_arrow) const
00538 {
00539     control_arrow[0] = getControlArrowX();
00540     control_arrow[1] = getControlArrowY();
00541 }
00542
00548 auto getAltArrowX() const
00549 {
00550     return getArrow(0, 3);
00551 }
00552
00558 auto getAltArrowY() const
00559 {
00560     return getArrow(1, 3);
00561 }
00562
00568 void getAltArrow(GLfloat* alt_arrow) const
00569 {
00570     alt_arrow[0] = getAltArrowX();
00571     alt_arrow[1] = getAltArrowY();
00572 }
00573
00579 const auto* getMouse() const
00580 {
00581     const auto& current_if{ interfaceData[interfaceNo] };
00582     return current_if.mouse.data();
00583 }
00584
00590 void getMouse(GLfloat* position) const
00591 {
00592     const auto& current_if{ interfaceData[interfaceNo] };
00593     position[0] = current_if.mouse[0];
00594     position[1] = current_if.mouse[1];
00595 }
00596
00603 auto getMouse(int direction) const
00604 {
00605     const auto& current_if{ interfaceData[interfaceNo] };
00606     return current_if.mouse[direction & 1];
00607 }
00608
00614 auto getMouseX() const
00615 {
00616     const auto& current_if{ interfaceData[interfaceNo] };
00617     return current_if.mouse[0];
00618 }
00619
00625 auto getMouseY() const
00626 {
00627     const auto& current_if{ interfaceData[interfaceNo] };
00628     return current_if.mouse[1];
00629 }
00630
00636 const auto* getWheel() const
00637 {
00638     const auto& current_if{ interfaceData[interfaceNo] };
00639     return current_if.wheel.data();
00640 }
00641
00647 void getWheel(GLfloat* rotation) const
00648 {
00649     const auto& current_if{ interfaceData[interfaceNo] };
00650     rotation[0] = current_if.wheel[0];
00651     rotation[1] = current_if.wheel[1];
00652 }
00653
00660 auto getWheel(int direction) const
00661 {
00662     const auto& current_if{ interfaceData[interfaceNo] };
00663     return current_if.wheel[direction & 1];
00664 }
00665
00671 auto getWheelX() const
00672 {
00673     const auto& current_if{ interfaceData[interfaceNo] };
00674     return current_if.wheel[0];
00675 }
00676
00682 auto getWheely() const
```

```
00683 {
00684     const auto& current_if{ interfaceData[interfaceNo] };
00685     return current_if.wheel[1];
00686 }
00687
00694 const auto& getTranslation(int button = GLFW_MOUSE_BUTTON_1) const
00695 {
00696     const auto& current_if{ interfaceData[interfaceNo] };
00697     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG.BUTTON_COUNT);
00698     return current_if.translation[button][1];
00699 }
00700
00707 auto getTranslationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00708 {
00709     const auto& current_if{ interfaceData[interfaceNo] };
00710     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG.BUTTON_COUNT);
00711     const auto& t{ current_if.translation[button][1] };
00712     GgMatrix m;
00713     m[ 1] = m[ 2] = m[ 3] = m[ 4] = m[ 6] = m[ 7] = m[ 8] = m[ 9] = m[11] = 0.0f;
00714     m[ 0] = m[ 5] = m[10] = m[15] = 1.0f;
00715     m[12] = t[0];
00716     m[13] = t[1];
00717     m[14] = t[2];
00718     return m;
00719 }
00720
00727 auto getScrollMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00728 {
00729     const auto& current_if{ interfaceData[interfaceNo] };
00730     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG.BUTTON_COUNT);
00731     const auto& t{ current_if.translation[button][1] };
00732     GgMatrix m;
00733     m[ 0] = m[ 5] = t[2] + 1.0f;
00734     m[ 1] = m[ 2] = m[ 3] = m[ 4] = m[ 6] = m[ 7] = m[ 8] = m[ 9] = m[11] = m[14] = 0.0f;
00735     m[10] = m[15] = 1.0f;
00736     m[12] = t[0];
00737     m[13] = t[1];
00738     return m;
00739 }
00740
00747 auto getRotation(int button = GLFW_MOUSE_BUTTON_1) const
00748 {
00749     const auto& current_if{ interfaceData[interfaceNo] };
00750     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG.BUTTON_COUNT);
00751     return current_if.rotation[button].getQuaternion();
00752 }
00753
00760 auto getRotationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00761 {
00762     const auto& current_if{ interfaceData[interfaceNo] };
00763     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG.BUTTON_COUNT);
00764     return current_if.rotation[button].getMatrix();
00765 }
00766
00770 void resetRotation()
00771 {
00772     // トラックボールを初期化する
00773     for (auto& tb : interfaceData[interfaceNo].rotation) tb.reset();
00774 }
00775
00779 void resetTranslation()
00780 {
00781     // 現在のインターフェースの平行移動量を初期化する
00782     interfaceData[interfaceNo].resetTranslation();
00783 }
00784
00788 void reset()
00789 {
00790     // トラックボール処理を初期化する
00791     resetRotation();
00792
00793     // 平行移動量を初期化する
00794     resetTranslation();
00795 }
00796
00802 void* getUserPointer() const
00803 {
00804     return userPointer;
00805 }
00806
00812 void setUserPointer(void* pointer)
00813 {
00814     userPointer = pointer;
00815 }
00816
00822 void setResizeFunc(void (*func)(const Window* window, int width, int height))
00823 {
```

```
00824     resizeFunc = func;
00825 }
00826
00832     void setKeyboardFunc(void (*func)(const Window* window, int key, int scancode, int action, int
00833     mods))
00834     {
00835         keyboardFunc = func;
00836     }
00837
00842     void setMouseFunc(void (*func)(const Window* window, int button, int action, int mods))
00843     {
00844         mouseFunc = func;
00845     }
00846
00852     void setWheelFunc(void (*func)(const Window* window, double x, double y))
00853     {
00854         wheelFunc = func;
00855     }
00856 };
00857
00858 #if defined(GG_USE_OOCULUS_RIFT)
00859 class Oculus
00860 {
00861     // Oculus Rift のセッション
00862     ovrSession session;
00863
00864     // Oculus Rift の状態
00865     ovrHmdDesc hmdDesc;
00866
00867     // Oculus Rift へのレンダリングに使う FBO
00868     GLuint oculusFbo[ovrEye_Count];
00869
00870     // Oculus Rift のスクリーンのサイズ
00871     GLfloat screen[ovrEye_Count][4];
00872
00873     // ミラー表示用の FBO
00874     GLuint mirrorFbo;
00875
00876     // Oculus Rift のミラー表示を行うウィンドウ
00877     const Window* window;
00878
00879     // if OVR_PRODUCT_VERSION > 0
00880
00881     // Oculus Rift に送る描画データ
00882     ovrLayerEyeFov layerData;
00883
00884     // Oculus Rift にレンダリングするフレームの番号
00885     long long frameIndex;
00886
00887     // Oculus Rift へのレンダリングに使う FBO のデブステクスチャ
00888     GLuint oculusDepth[ovrEye_Count];
00889
00890     // ミラー表示用の FBO のサイズ
00891     int mirrorWidth, mirrorHeight;
00892
00893     // グラフィックスカードのデフォルトの LUID を得る
00894     static ovrGraphicsLuid GetDefaultAdapterLuid();
00895
00896     // グラフィックスカードの LUID の比較
00897     static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);
00898
00899 # else
00900
00901     // Oculus Rift に送る描画データ
00902     ovrLayerUnion layerData;
00903
00904     // Oculus Rift のレンダリング情報
00905     ovrEyeRenderDesc eyeRenderDesc[ovrEye_Count];
00906
00907     // Oculus Rift の視点情報
00908     ovrPosef eyePose[ovrEye_Count];
00909
00910     // ミラー表示用の FBO のカラー テクスチャ
00911     ovrGLTexture* mirrorTexture;
00912
00913 # endif
00914
00915     // コンストラクタ
00916     //
```

```

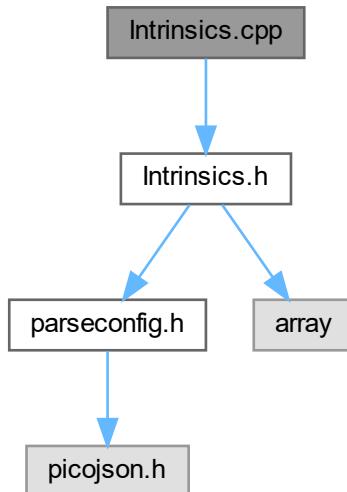
00931     Oculus();
00932
00933     //
00934     // デストラクタ
00935     //
00936     virtual ~Oculus() = default;
00937
00938 public:
00939
00940     // シングルトンなのでコピー・ムーブ禁止
00941     Oculus(const Oculus&) = delete;
00942     Oculus& operator=(const Oculus&) = delete;
00943     Oculus(Oculus&&) = delete;
00944     Oculus& operator=(Oculus&&) = delete;
00945
00951     static Oculus& initialize(const Window& window);
00952
00956     void terminate();
00957
00963     bool begin();
00964
00973     void select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation);
00974
00980     void timewarp(const GgMatrix& projection);
00981
00987     void commit(int eye);
00988
00995     bool submit(bool mirror = true);
00996 }
00997 #endif
00998 };

```

## 9.41 Intrinsicss.cpp ファイル

#include "Intrinsicss.h"

Intrinsicss.cpp の依存先関係図:



### 9.41.1 詳解

キャプチャデバイス固有のパラメータクラスの定義

著者

Kohe Tokoi

日付

March 6, 2024

[Intrinsics.cpp](#) に定義があります。

## 9.42 Intrinsics.cpp

[詳解]

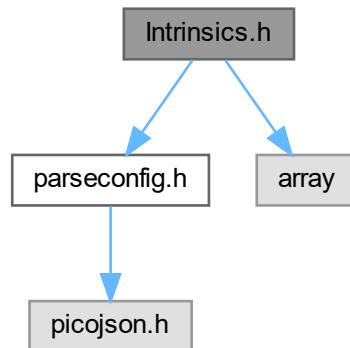
```

00001
00008 #include "Intrinsics.h"
00009 //
00010 //
00011 // 構成ファイルを読み込むときに使う
00012 // キャプチャデバイス固有のパラメータの構造体のコンストラクタ
00013 //
00014 Intrinsics::Intrinsics(const picojson::object& object)
00015 {
00016     // キャプチャデバイスの画角
00017     getValue(object, "fov", fov);
00018
00019     // キャプチャデバイスの中心位置
00020     getValue(object, "center", center);
00021
00022     // キャプチャデバイスの解像度
00023     getValue(object, "size", size);
00024
00025     // キャプチャデバイスの周波数
00026     getValue(object, "fps", fps);
00027 }
00028
00029 //
00030 // スクリーンの対角線長をもとにしてキャプチャデバイスのレンズの縦横の画角を変更する
00031 //
00032 void Intrinsics::setFov(float focal)
00033 {
00034     // 撮像面の画素数の対角線長×2
00035     const auto d{ hypotf(static_cast<float>(size[0]), static_cast<float>(size[1])) * 2.0f };
00036
00037     // 撮像面の画素数の対角線長に対するスクリーンの幅と高さの比の 1/2 (d は対角線長の 2 倍だから)
00038     const auto w{ size[0] / d };
00039     const auto h{ size[1] / d };
00040
00041     // 焦点距離が 1 のときの撮像面の対角線長
00042     const auto t{ sensorSize / focal };
00043
00044     // 焦点距離が 1 のときの撮像面の対角線長をこの比で横と縦に振り分ける
00045     const auto u{ t * w };
00046     const auto v{ t * h };
00047
00048     // 画角を求め単位を度に換算して設定する
00049     constexpr auto s{ 360.0f / 3.14159265359f };
00050     setFov(atan(u) * s, atan(v) * s);
00051 }
```

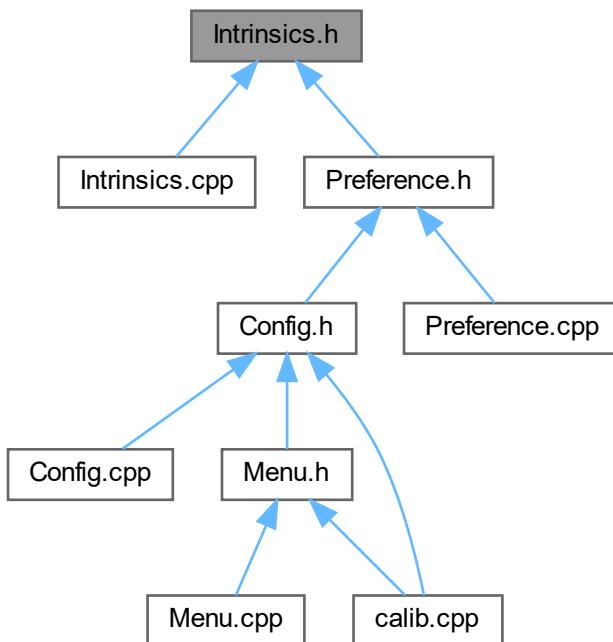
## 9.43 Intrinsics.h ファイル

```
#include "parseconfig.h"
#include <array>
```

Intrinsics.h の依存先関係図:



被依存関係図:



クラス

- struct [Intrinsics](#)

### 9.43.1 詳解

キャプチャデバイス固有のパラメータクラスの定義

著者

Kohe Tokoi

日付

March 6, 2024

[Intrinsics.h](#) に定義があります。

## 9.44 Intrinsics.h

[詳解]

```

00001 #pragma once
00002
00010
00011 // 構成ファイルの読み取り補助
00012 #include "parseconfig.h"
00013
00014 // 標準ライブラリ
00015 #include <array>
00016
00020 struct Intrinsics
00021 {
00023     std::array<float, 2> fov;
00024
00026     std::array<float, 2> center;
00027
00029     std::array<int, 2> size;
00030
00032     double fps;
00033
00039     Intrinsics()
00040     : Intrinsics{ { 50.03f, 38.58f }, { 0.0f, 0.0f }, { 640, 480 }, 30.0 }
00041     {
00042     }
00043
00053     Intrinsics(const std::array<float, 2>& fov, const std::array<float, 2>& center,
00054         const std::array<int, 2> size, double fps)
00055     : fov{ fov }
00056     , center{ center }
00057     , size{ size }
00058     , fps{ fps }
00059     {
00060     }
00061
00066     Intrinsics(const picojson::object& object);
00067
00069     static constexpr auto sensorSize{ 35.0f };
00070
00076     void setFov(float focal);
00077
00084     void setFov(float fovx, float fovy)
00085     {
00086         // キャプチャデバイスのレンズの縦横の画角を設定する
00087         fov[0] = fovx;
00088         fov[1] = fovy;
00089     }
00090
00097     void setCenter(float x, float y)
00098     {
00099         // キャプチャデバイスのレンズの中心の位置を設定する
00100         center[0] = x;
00101         center[1] = y;
00102     }
00103
00104

```

```

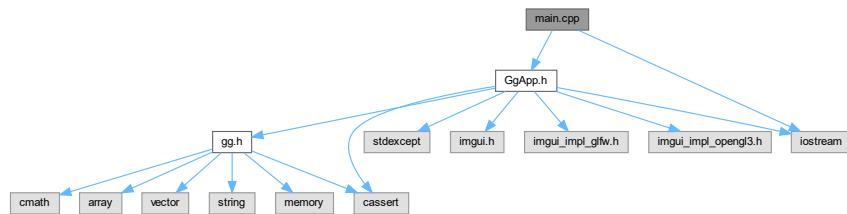
00111 void setSize(int width, int height)
00112 {
00113     // 装置の解像度を設定する
00114     size[0] = width;
00115     size[1] = height;
00116 }
00117
00118
00124 void setFps(double frequency)
00125 {
00126     // 装置のフレームレートを設定する
00127     fps = frequency;
00128 }
00129 };

```

## 9.45 main.cpp ファイル

```
#include "GgApp.h"
#include <iostream>
```

main.cpp の依存先関係図:



## マクロ定義

- `#define HEADER_STR "ゲーム グラフィックス特論"`

## 関数

- `int main (int argc, const char *const *argv)`

### 9.45.1 詳解

#### ゲーム グラフィックス特論宿題 アプリケーション

#### 著者

Kohe Tokoi

#### 日付

February 20, 2024

main.cpp に定義があります。

## 9.45.2 マクロ定義詳解

### 9.45.2.1 HEADER\_STR

```
#define HEADER_STR "ゲームグラフィックス特論"
```

main.cpp の 18 行目に定義があります。

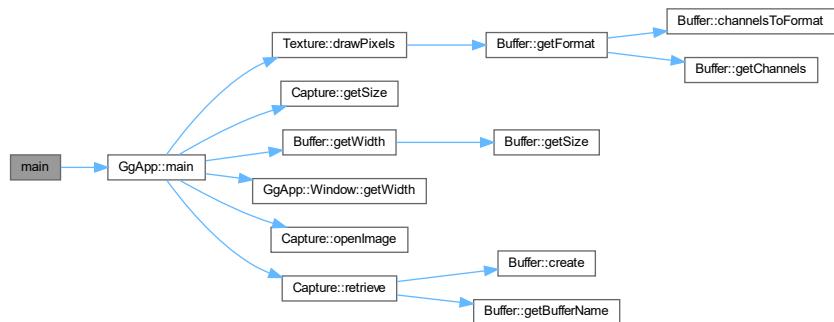
## 9.45.3 関数詳解

### 9.45.3.1 main()

```
int main (
    int argc,
    const char *const * argv )
```

main.cpp の 23 行目に定義があります。

呼び出し関係図:



## 9.46 main.cpp

### [詳解]

```
00001
00008 #include "GgApp.h"
00009
00010 // MessageBox の準備
00011 #if defined(_MSC_VER)
00012 # include <atlstr.h>
00013 #elif defined(_APPLE_)
00014 # include <CoreFoundation/CoreFoundation.h>
00015 #else
00016 # include <iostream>
00017 #endif
```

```

00018 #define HEADER_STR "ゲームグラフィックス特論"
00019 //
00020 // メインプログラム
00021 //
00022 //
00023 int main(int argc, const char* const* argv) try
00024 {
00025     // アプリケーションのオブジェクトを生成する
00026 #if defined(GL_GLES_PROTOTYPES)
00027     GgApp app(3, 1);
00028 #else
00029     GgApp app(4, 1);
00030 #endif
00031
00032     // アプリケーションを実行する
00033     return app.main(argc, argv);
00034 }
00035 catch (const std::runtime_error &e)
00036 {
00037     // エラーメッセージを表示する
00038 #if defined(_MSC_VER)
00039     MessageBox(NULL, CString(e.what()), TEXT(HEADER_STR), MB_ICONERROR);
00040 #elif defined(__APPLE__)
00041     // the following code is copied from
00042     // http://blog.jorgearimany.com/2010/05/messagebox-from-windows-to-mac.html
00043     CFStringRef msg_ref = CFStringCreateWithCString(NULL, e.what(), kCFStringEncodingUTF8);
00044
00045     // result code from the message box
00046     CFOptionFlags result;
00047
00048     // launch the message box
00049     CFUserNotificationDisplayAlert(
00050         0,                                // no timeout
00051         KCFUserNotificationNoteAlertLevel, // change it depending message_type flags ( MB_ICONASTERISK.... etc.)
00052         NULL,                            // icon url, use default, you can change it depending
00053         NULL,                            // not used
00054         NULL,                            // localization of strings
00055         CFSTR(HEADER_STR),             // header text
00056         msg_ref,                         // message text
00057         NULL,                            // default "ok" text in button
00058         NULL,                            // alternate button title
00059         NULL,                            // other button title, null--> no other button
00060         &result                           // response flags
00061     );
00062
00063     // Clean up the strings
00064     CFRelease(msg_ref);
00065 #else
00066     std::cerr << HEADER_STR << ":" << e.what() << '\n';
00067 #endif
00068
00069     // プログラムを終了する
00070     return EXIT_FAILURE;
00071 }

```

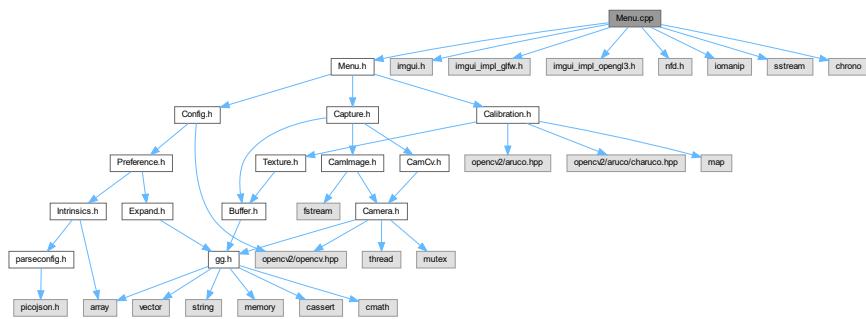
## 9.47 Menu.cpp ファイル

```

#include "Menu.h"
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"
#include "nfd.h"
#include <iomanip>
#include <sstream>
#include <chrono>

```

Menu.cpp の依存先関係図:



## 変数

- constexpr nfdfilteritem\_t jsonFilter [] { { "JSON", "json" } }
- constexpr nfdfilteritem\_t imageFilter [] { "Images", "png,jpg,jpeg,jfif,bmp,dib" }
- constexpr nfdfilteritem\_t movieFilter [] { "Movies", "mp4,m4v,mpg,mov,avi,ogg,mkv" }

### 9.47.1 詳解

メニューの描画クラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

Menu.cpp に定義があります。

### 9.47.2 変数詳解

#### 9.47.2.1 imageFilter

```
constexpr nfdfilteritem_t imageFilter[] { "Images", "png,jpg,jpeg,jfif,bmp,dib" } [constexpr]
```

Menu.cpp の 22 行目に定義があります。

### 9.47.2.2 jsonFilter

```
constexpr nfdfilteritem_t jsonFilter[] { { "JSON", "json" } } [constexpr]
```

Menu.cpp の 19 行目に定義があります。

### 9.47.2.3 movieFilter

```
constexpr nfdfilteritem_t movieFilter[] { "Movies", "mp4,m4v,mpg,mov,avi,ogg,mkv" } [constexpr]
```

Menu.cpp の 25 行目に定義があります。

## 9.48 Menu.cpp

### [詳解]

```
00001
00008 #include "Menu.h"
00009
00010 // ImGui
00011 #include "imgui.h"
00012 #include "imgui_impl_glfw.h"
00013 #include "imgui_impl_opengl3.h"
00014
00015 // ファイルダイアログ
00016 #include "nfd.h"
00017
00018 // JSON ファイル名のフィルタ
00019 constexpr nfdfilteritem_t jsonFilter[] { { "JSON", "json" } };
00020
00021 // 画像ファイル名のフィルタ
00022 constexpr nfdfilteritem_t imageFilter[] { "Images", "png,jpg,jpeg,jfif,bmp,dib" };
00023
00024 // 動画ファイル名のフィルタ
00025 constexpr nfdfilteritem_t movieFilter[] { "Movies", "mp4,m4v,mpg,mov,avi,ogg,mkv" };
00026
00027 // 標準ライブライアリ
00028 #include <iomanip>
00029 #include <sstream>
00030 #include <chrono>
00031
00032 //
00033 // 画像ファイルを開く
00034 //
00035 void Menu::openImage()
00036 {
00037     // ファイルダイアログから得るパス
00038     nfdchart* filepath;
00039
00040     // ファイルダイアログを開く
00041     if (NFD_OpenDialog(&filepath, imageFilter, 1, NULL) == NFD_OKAY)
00042     {
00043         // ダイアログで指定した画像ファイルが開けたら
00044         if (capture.openImage(filepath))
00045         {
00046             // 構成データの解像度と画角を開いた画像に合わせる
00047             setSize(capture.getSize());
00048         }
00049     else
00050     {
00051         // 開けなかった
00052         errorMessage = u8"画像ファイルが開けません";
00053     }
00054
00055     // ファイルバスの取り出しに使ったメモリを開放する
00056     NFD_FreePath(filepath);
00057 }
00058 }
00059
00060 //
00061 // 動画ファイルを開く
```

```

00062 //
00063 void Menu::openMovie()
00064 {
00065     // ファイルダイアログから得るパス
00066     nfdchar_t* filepath;
00067
00068     // ファイルダイアログを開く
00069     if (NFD_OpenDialog(&filepath, movieFilter, 1, NULL) == NFD_OKAY)
00070     {
00071         // 入力特性をファイルに切り替えて
00072         backend = cv::CAP_FFMPEG;
00073
00074         // ファイルのリストを取り出し
00075         auto& fileList{ config.deviceList.at(backend) };
00076         const auto fileListLength{ static_cast<int>(fileList.size()) };
00077
00078         // ファイルのリストの各ファイルについて
00079         for (deviceNumber = 0; deviceNumber < fileListLength; ++deviceNumber)
00080         {
00081             // 選択したファイルと同じものがあればそれを選択する
00082             if (fileList[deviceNumber] == filepath) break;
00083         }
00084
00085         // 選択したファイルがファイルのリストの中になければ
00086         if (deviceNumber == fileListLength)
00087         {
00088             // その先頭にファイルパスを挿入して
00089             fileList.insert(fileList.begin(), filepath);
00090
00091             // そのエントリを選択する
00092             deviceNumber = 0;
00093         }
00094
00095         // ダイアログで指定した動画ファイルが開けたら
00096         if (capture.openMovie(filepath, backend))
00097         {
00098             // 構成データの解像度と画角を開いた画像に合わせる
00099             setSize(capture.getSize());
00100         }
00101     else
00102     {
00103         // 開けなかった
00104         errorMessage = u8"動画ファイルが開けません";
00105     }
00106
00107     // ファイルパスの取り出しに使ったメモリを開放する
00108     NFD_FreePath(filepath);
00109 }
00110 }
00111
00112 void Menu::openDevice()
00113 {
00114     // バックエンドが GStreamer なら
00115     if (backend == cv::CAP_GSTREAMER)
00116     {
00117         // ファイルのリストを取り出し
00118         const auto& pipeline{ config.deviceList.at(backend)[deviceNumber] };
00119
00120         // ダイアログで指定したバイオラインが開けなかったら
00121         if (!capture.openMovie(pipeline, backend))
00122         {
00123             // 開けなかった
00124             errorMessage = u8"バイオラインが開けません";
00125         }
00126     }
00127     else if (backend != cv::CAP_FFMPEG)
00128     {
00129         // ダイアログで指定したキャプチャデバイスが開けなかったら
00130         if (!capture.openDevice(deviceNumber, intrinsics.size, intrinsics.fps, backend,
00131             codecNumber > 0 ? config.codecList[codecNumber] : ""))
00132         {
00133             // 開けなかった
00134             errorMessage = u8"キャプチャデバイスが開けません";
00135         }
00136     }
00137 }
00138
00139 }
00140 }
00141
00142 /**
00143 // 構成ファイルを読み込む
00144 */
00145 void Menu::loadConfig()
00146 {
00147     // ファイルダイアログから得るパス
00148     nfdchar_t* filepath;
00149
00150     // ファイルダイアログを開く
00151     if (NFD_OpenDialog(&filepath, jsonFilter, 1, NULL) == NFD_OKAY)

```

```
00152 {
00153     // 現在の構成を構成ファイルの内容にする
00154     if (const_cast<Config&>(config).load(filepath))
00155     {
00156         // 読み込んだ構成の数が現在の選択よりも少ないとときは最初の項目の構成にする
00157         if (preferenceNumber > static_cast<int>(config.preferenceList.size()))
00158             preferenceNumber = 0;
00159
00160         // 現在の設定に反映する
00161         settings = config.settings;
00162     }
00163     else
00164     {
00165         // 読み込めなかった
00166         errorMessage = u8"構成ファイルが読み込めません";
00167     }
00168
00169     // ファイルパスの取り出しに使ったメモリを開放する
00170     NFD_FreePath(filepath);
00171 }
00172 }
00173
00174 //
00175 // 構成ファイルを保存する
00176 //
00177 void Menu::saveConfig()
00178 {
00179     // ファイルダイアログから得るパス
00180     nfdchar_t* filepath;
00181
00182     // ファイルダイアログを開く
00183     if (NFD_SaveDialog(&filepath, jsonFilter, 1, NULL, "*.json") == NFD_OKAY)
00184     {
00185         // 現在の設定で構成を更新する
00186         const_cast<Config&>(config).settings = settings;
00187
00188         // 現在の構成を保存する
00189         if (!config.save(filepath))
00190         {
00191             // 保存できなかった
00192             errorMessage = u8"構成ファイルが保存できません";
00193         }
00194
00195         // ファイルパスの取り出しに使ったメモリを開放する
00196         NFD_FreePath(filepath);
00197     }
00198 }
00199
00200 //
00201 // 較正ファイルを読み込む
00202 //
00203 void Menu::loadParameters()
00204 {
00205     // ファイルダイアログから得るパス
00206     nfdchar_t* filepath;
00207
00208     // ファイルダイアログを開く
00209     if (NFD_OpenDialog(&filepath, jsonFilter, 1, NULL) == NFD_OKAY)
00210     {
00211         // 現在のキャリブレーションパラメータを較正ファイルの内容にする
00212         if (!calibration.loadParameters(filepath))
00213         {
00214             // 読み込めなかった
00215             errorMessage = u8"較正ファイルが読み込めません";
00216         }
00217         else
00218         {
00219             // 較正ファイルを読み込んだ時はボードのコーナー表示を消す
00220             detectBoard = false;
00221         }
00222
00223         // ファイルパスの取り出しに使ったメモリを開放する
00224         NFD_FreePath(filepath);
00225     }
00226 }
00227
00228 //
00229 // 較正ファイルを保存する
00230 //
00231 void Menu::saveParameters()
00232 {
00233     // キャリブレーションが完了していなければ戻る
00234     if (!calibration.finished()) return;
00235
00236     // 現在時刻の取得
00237     const auto now{ std::chrono::system_clock::to_time_t(std::chrono::system_clock::now()) };
00238     const std::tm* const localTime{ std::localtime(&now) };
```

```

00239 // 時刻を文字列に変換
00240 const auto timeString{ std::put_time(localTime, "cal-%Y%m%d%H%M%S.json") };
00241 const auto pathString{ static_cast<std::ostringstream&>(std::ostringstream() << timeString).str() };
00242 };
00243 // ファイルダイアログから得るパス
00244 nfdchar_t* filepath;
00245
00246 // ファイルダイアログを開く
00247 if (NFD_SaveDialog(&filepath, jsonFilter, 1, NULL, pathString.c_str()) == NFD_OKAY)
00248 {
00249     // 現在のキャリブレーションパラメータを構成ファイルに保存する
00250     if (!calibration.saveParameters(filepath))
00251     {
00252         // 保存できなかった
00253         errorMessage = u8"較正ファイルが保存できません";
00254     }
00255
00256     // ファイルバスの取り出しに使ったメモリを開放する
00257     NFD_FreePath(filepath);
00258 }
00259 }
00260 }
00261
00262 //
00263 // フォルダ内のファイルを使って較正する
00264 //
00265 void Menu::calibrateByFiles()
00266 {
00267     // ファイルダイアログから得るパス
00268     const nfdpathset_t* outPaths;
00269
00270     // ファイルダイアログを開く
00271     if (NFD_OpenDialogMultiple(outPaths, imageFilter, 1, NULL) == NFD_OKAY)
00272     {
00273         // ファイルバスの一覧を得る
00274         nfdpathsetenum_t enumerator;
00275         NFD_PathSet_GetEnum(outPaths, &enumerator);
00276
00277         // ファイルバスの一覧からファイルバスを一つずつ取り出して
00278         for (nfdchar_t* path = NULL; NFD_PathSet_EnumNext(&enumerator, &path) && path;)
00279         {
00280             // 画像ファイルを読み出す
00281             auto image{ CamImage::load(path) };
00282
00283             // 読み出したファイルに中身が入っていたら
00284             if (!image.empty())
00285             {
00286                 // 解析用のバッファを作って
00287                 Buffer buffer{ image.cols, image.rows, image.channels() };
00288
00289                 // バッファにデータを書き込んで
00290                 buffer.setdata(image.cols * image.rows * image.channels(), image.data);
00291
00292                 // マーカとボードを検出して
00293                 calibration.detect(buffer, true);
00294
00295                 // コーナーを記録する
00296                 calibration.recordCorners();
00297             }
00298
00299             // ファイルバスの取り出しに使ったメモリを開放する
00300             NFD_PathSet_FreePath(path);
00301         }
00302
00303         // ファイルバスの一覧に使ったメモリを開放する
00304         NFD_PathSet_FreeEnum(&enumerator);
00305
00306         // フォルダのバスに使ったメモリを開放する
00307         NFD_PathSet_Free(outPaths);
00308     }
00309 }
00310
00311 //
00312 // コンストラクタ
00313 //
00314 Menu::Menu(const Config& config, Capture& capture, Calibration& calibration)
00315 : config{ config }
00316 , settings{ config.settings }
00317 , capture{ capture }
00318 , calibration{ calibration }
00319 , deviceNumber{ 0 }
00320 , codecNumber{ 0 }
00321 , preferenceNumber{ 0 }
00322 , backend{ cv::CAP_ANY }
00323 , pose{ ggIdentity() }
00324 , menubarHeight{ 0 }

```

```
00325 , showControlPanel{ true }
00326 , showInformationPanel{ false }
00327 , quit{ false }
00328 , errorMessage{ nullptr }
00329 , detectMarker{ false }
00330 , detectBoard{ false }
00331 {
00332 // ファイルダイアログ (Native File Dialog Extended) を初期化する
00333 NFD_Init();
00334
00335 // Dear ImGui の入力デバイス
00336 //io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;           // キーボードコントロールを使う
00337 //io.ConfigFlags |= ImGuiConfigFlags_NavEnableGamepad;           // ゲームパッドを使う
00338
00339 // Dear ImGui のスタイル
00340 //ImGui::StyleColorsDark();                                         // 暗めのスタイル
00341 //ImGui::StyleColorsClassic();                                     // 以前のスタイル
00342
00343 // 日本語を表示できるメニューフォントを読み込む
00344 if (!ImGui::GetIO().Fonts->AddFontFromTTF(config.menuFont.c_str(), config.menuFontSize,
00345     nullptr, ImGui::GetIO().Fonts->GetGlyphRangesJapanese()))
00346 {
00347     // メニューフォントが読み込めなかったらエラーにする
00348     throw std::runtime_error("Cannot find any menu fonts.");
00349 }
00350 }
00351
00352 //
00353 // デストラクタ
00354 //
00355 Menu::~Menu()
00356 {
00357     // ファイルダイアログ (Native File Dialog Extended) を終了する
00358     NFD_Quit();
00359 }
00360
00361 //
00362 // 解像度の初期値を設定する
00363 //
00364 void Menu::setSize(const std::array<int, 2>& size)
00365 {
00366     // 解像度の調整値を設定する
00367     intrinsics.size = size;
00368
00369     // 投影像の画角と中心位置を設定する
00370     intrinsics.setFov(settings.focal);
00371     intrinsics.setCenter(0.0f, 0.0f);
00372 }
00373
00374 //
00375 // シェーダを設定する
00376 //
00377 std::array<GLsizei, 2> Menu::setup(GLfloat aspect) const
00378 {
00379     // シェーダを設定する
00380     return config.preferenceList[preferenceNumber].getShader().setup(settings.samples, aspect,
00381         pose, intrinsics.fov, intrinsics.center, settings.getFocal(), config.background);
00382 }
00383
00384 //
00385 // メニューの描画
00386 //
00387 void Menu::draw()
00388 {
00389     // ImGui のフレームを準備する
00390     ImGui::NewFrame();
00391
00392     // メインメニューバー
00393     if (ImGui::BeginMainMenuBar())
00394     {
00395         // ファイルメニュー
00396         if (ImGui::BeginMenu(u8"ファイル"))
00397         {
00398             // 画像ファイルを開く
00399             if (ImGui::MenuItem(u8"画像ファイルを開く")) openImage();
00400
00401             // 動画ファイルを開く
00402             if (ImGui::MenuItem(u8"動画ファイルを開く")) openMovie();
00403
00404             // ChArUco Board の作成
00405             if (ImGui::MenuItem(u8"ボード画像を保存"))
00406             {
00407                 // ChArUco Board の画像を作成する
00408                 cv::Mat boardImage;
00409                 calibration.drawBoard(boardImage, 980, 692);
00410
00411                 // ファイルに保存する
00412             }
00413         }
00414     }
00415 }
```

```

00412     saveImage(boardImage, "ChArUcoBoard.png");
00413 }
00414
00415 // 構成ファイルを開く
00416 if (ImGui::MenuItem(u8"構成ファイルを開く")) loadConfig();
00417
00418 // 構成ファイルを保存する
00419 if (ImGui::MenuItem(u8"構成ファイルを保存")) saveConfig();
00420
00421 // キャリプレーションパラメータファイルを開く
00422 if (ImGui::MenuItem(u8"較正ファイルを開く")) loadParameters();
00423
00424 // キャリプレーションパラメータファイルを保存する
00425 if (ImGui::MenuItem(u8"較正ファイルを保存")) saveParameters();
00426
00427 // フォルダ内の画像ファイルを使って較正する
00428 if (ImGui::MenuItem(u8"較正用画像を開く")) calibrateByFiles();
00429
00430 // 終了
00431 quit = ImGui::MenuItem(u8"終了");
00432
00433 // File メニュー修了
00434 ImGui::EndMenu();
00435 }
00436
00437 // ウィンドウメニュー
00438 if (ImGui::BeginMenu(u8"ウィンドウ"))
00439 {
00440     // コントロールパネルの表示
00441     ImGui::MenuItem(u8"コントロールパネル", NULL, &showControlPanel);
00442
00443     // 情報パネルの表示
00444     ImGui::MenuItem(u8"情報", NULL, &showInformationPanel);
00445
00446     // File メニュー修了
00447     ImGui::EndMenu();
00448 }
00449
00450 // メニューバーの高さを保存しておく
00451 menubarHeight = static_cast<GLsizei>(ImGui::GetWindowHeight());
00452
00453 // メインメニュー終了
00454 ImGui::EndMainMenuBar();
00455
00456
00457 // コントロールパネル
00458 if (showControlPanel)
00459 {
00460     // ウィンドウの位置とサイズ
00461     ImGui::SetNextWindowPos(ImVec2(2.0f, 2.0f + menubarHeight), ImGuiCond.Once);
00462     ImGui::SetNextWindowSize(ImVec2(231, 640), ImGuiCond.Once);
00463     ImGui::Begin(u8"コントロールパネル", &showControlPanel);
00464
00465     // 投影方式の選択
00466     if (ImGui::BeginCombo(u8"投影方式", getPreference().getDescription().c_str()))
00467     {
00468         // すべての投影方式について
00469         for (int i = 0; i < static_cast<int>(config.preferenceList.size()); ++i)
00470         {
00471             // その投影方式が選択されていれば真
00472             const bool selected{ i == preferenceNumber };
00473
00474             // 投影方式を（それが現在の投影方式ならハイライトして）コンボボックスに表示する
00475             if (ImGui::Selectable(getPreference(i).getDescription().c_str(), selected))
00476             {
00477                 // 表示した投影方式が選択されたいたらそれを現在の選択とする
00478                 preferenceNumber = i;
00479
00480                 // 選択した投影方式のキャプチャデバイス固有のパラメータをコピーする
00481                 intrinsics = getPreference().getIntrinsics();
00482             }
00483
00484             // この選択を次にコンボボックスを開いたときのデフォルトにしておく
00485             if (selected) ImGui::SetItemDefaultFocus();
00486         }
00487         ImGui::EndCombo();
00488     }
00489
00490 // レンズの画角を設定する
00491 ImGui::DragFloat2(u8"画角", intrinsics.fov.data(), 0.1f, -360.0f, 360.0f, "%.2f");
00492
00493 // レンズの中心位置
00494 ImGui::DragFloat2(u8"中心", intrinsics.center.data(), 0.001f, -1.0f, 1.0f, "%.4f");
00495
00496 // キャプチャデバイス固有のパラメータを元に戻す
00497 if (ImGui::Button(u8"戻復"))
00498 {

```

```

00499     // 選択した投影方式のキャプチャデバイス固有のパラメータを回復する
00500     intrinsics = getPreference().getIntrinsics();
00501 }
00502
00503     // 姿勢
00504     ImGui::SliderAngle(u8"方位", &settings.euler[1], -180.0f, 180.0f, "%.2f");
00505     ImGui::SliderAngle(u8"仰角", &settings.euler[0], -180.0f, 180.0f, "%.2f");
00506     ImGui::SliderAngle(u8"傾斜", &settings.euler[2], -180.0f, 180.0f, "%.2f");
00507     pose = ggRotateY(settings.euler[1]).rotateX(settings.euler[0]).rotateZ(settings.euler[2]);
00508
00509     // 焦点距離
00510     ImGui::SliderFloat(u8"焦点距", &settings.focal, settings.focalRange[0], settings.focalRange[1], "%.1f");
00511
00512     // 姿勢を元に戻す
00513     if (ImGui::Button(u8"復帰"))
00514     {
00515         settings.euler = config.settings.euler;
00516         settings.focal = config.settings.focal;
00517         settings.focalRange = config.settings.focalRange;
00518     }
00519
00520     // 較正関連項目
00521     ImGui::Separator();
00522
00523     // 辞書の選択
00524     if (ImGui::BeginCombo(u8"辞書", settings.dictionaryName.c_str()))
00525     {
00526         // すべての辞書について
00527         for (auto d = calibration.dictionaryList.begin(); d != calibration.dictionaryList.end(); ++d)
00528         {
00529             // その設定が現在選択されている設定なら真
00530             const bool selected(d->first == settings.dictionaryName);
00531
00532             // 設定を(それが現在の設定ならハイライトして)コンボボックスに表示する
00533             if (ImGui::Selectable(d->first.c_str(), d->first == settings.dictionaryName))
00534             {
00535                 // 表示した設定が選択されていたらそれを現在の選択とする
00536                 settings.dictionaryName = d->first;
00537
00538                 // 選択した ArUco Marker の辞書を設定する
00539                 calibration.setDictionary(settings.dictionaryName, settings.checkerLength);
00540             }
00541
00542             // この選択を次にコンボボックスを開いたときのデフォルトにしておく
00543             if (selected) ImGui::SetItemDefaultFocus();
00544         }
00545     }
00546     ImGui::EndCombo();
00547
00548     // ChArUco Board の大きさ
00549     if (ImGui::InputFloat2(u8"升目長", settings.checkerLength.data(), "%2f cm"))
00550     {
00551         // ChArUco Board を作り直す
00552         calibration.createBoard(settings.checkerLength);
00553     }
00554
00555     // ArUco Marker の検出
00556     ImGui::Checkbox(u8"マーカ検出", &detectMarker);
00557
00558     // ArUco Marker を検出するなら
00559     if (detectMarker)
00560     {
00561         // ChArUco Board の検出
00562         ImGui::SameLine();
00563         ImGui::Checkbox(u8"ボード検出", &detectBoard);
00564     }
00565     else
00566     {
00567         // ArUco Marker を検出していなければ ChArUco Board は検出しない
00568         detectBoard = false;
00569     }
00570
00571     // 「標本」ボタンをクリックしたとき ChArUco Board の検出中なら
00572     if (ImGui::Button(u8"標本") && detectBoard) calibration.recordCorners();
00573
00574     // 1つでも標本を取得していれば
00575     if (calibration.getCornerCount() > 0)
00576     {
00577         // 標本の「消去」ボタンを表示する
00578         ImGui::SameLine();
00579         if (ImGui::Button(u8"消去")) calibration.discardCorners();
00580
00581         // 標本を6つ以上取得していれば
00582         if (calibration.getCornerCount() >= 6)
00583         {
00584             // 「較正」ボタンを表示する

```

```

00585     ImGui::SameLine();
00586     if (ImGui::Button(u8"較正")) calibration.calibrate();
00587
00588     // 調整が完了していれば
00589     if (calibration.finished())
00590     {
00591         // 「完了」を表示する
00592         ImGui::SameLine();
00593         ImGui::TextColored(ImVec4(0.2f, 1.0f, 0.0f, 1.0f), "%s", u8"完了");
00594     }
00595 }
00596
00597 // 装置関連項目
00598 ImGui::Separator();
00599 ImGui::Text("%s", u8"以下の変更は [開始] で反映します");
00600
00601 // デバイスプリファレンスを選択する
00602 if (ImGui::BeginCombo(u8"装置特性", config.backendList.at(backend)))
00603 {
00604     // すべての表示方式について
00605     for (auto& [apiId, apiName] : config.backendList)
00606     {
00607         // その表示方式が選択されていれば真
00608         const bool selected{ apiId == backend };
00609
00610         // 装置特性を（それが現在の装置特性ならハイライトして）コンボボックスに表示する
00611         if (ImGui::Selectable(apiName, selected))
00612         {
00613             // 表示した装置特性が選択されていたらそれを現在の選択とする
00614             backend = apiId;
00615
00616             // 切り替え前の装置特性のデバイスが存在しなければ最初のデバイスの番号を選択する
00617             if (deviceNumber < 0) deviceNumber = 0;
00618
00619             // 選択されているデバイスの番号が接続されたキャプチャデバイスの数を超えないようにする
00620             const int count{ config.getDeviceCount(backend) };
00621             if (deviceNumber >= count) deviceNumber = count - 1;
00622         }
00623
00624         // この選択を次にコンボボックスを開いたときのデフォルトにしておく
00625         if (selected) ImGui::SetItemDefaultFocus();
00626     }
00627     ImGui::EndCombo();
00628 }
00629
00630 // キャプチャデバイスが存在すれば
00631 if (deviceNumber >= 0)
00632 {
00633     // キャプチャデバイスの選択コンボボックス
00634     if (ImGui::BeginCombo(u8"入力源", config.getDeviceName(backend, deviceNumber).c_str()))
00635     {
00636         // すべてのキャプチャデバイスについて
00637         for (int i = 0; i < static_cast<int>(config.getDeviceList(backend).size()); ++i)
00638         {
00639             // キャプチャデバイス名を（それを選択していればハイライトして）コンボボックスに表示する
00640             if (ImGui::Selectable(config.getDeviceName(backend, i).c_str(), i == deviceNumber))
00641             {
00642                 // 表示したキャプチャデバイスが選択されていたらそのキャプチャデバイスを選択する
00643                 deviceNumber = i;
00644
00645                 // この選択を次にコンボボックスを開いたときのデフォルトにしておく
00646                 ImGui::SetItemDefaultFocus();
00647             }
00648         }
00649     }
00650     ImGui::EndCombo();
00651 }
00652 }
00653 else
00654 {
00655     // 使えるキャプチャデバイスがない
00656     ImGui::TextColored(ImVec4(1.0f, 0.2f, 0.0f, 1.0f), "%s", u8"キャプチャデバイスが見つかりません");
00657 }
00658
00659 // キャプチャするサイズとフレームレート
00660 ImGui::InputInt2(u8"解像度", intrinsics.size.data());
00661 ImGui::InputDouble(u8"周波数", &intrinsics.fps, 1.0f, 1.0f, "%1f");
00662
00663 // コーデックを選択する
00664 if (ImGui::BeginCombo(u8"符号化", config.codecList[codecNumber]))
00665 {
00666     // すべてのコーデックについて
00667     for (int i = 0; i < static_cast<int>(config.codecList.size()); ++i)
00668     {
00669         // コーデックを（それを選択していればハイライトして）コンボボックスに表示する
00670         if (ImGui::Selectable(config.codecList[i], i == codecNumber))
00671     }

```

```
00672      // 表示したキャプチャデバイスが選択されていたらそのキャプチャデバイスを選択する
00673      codecNumber = i;
00674
00675      // この選択を次にコンボボックスを開いたときのデフォルトにしておく
00676      ImGui::SetItemDefaultFocus();
00677  }
00678  }
00679  ImGui::EndCombo();
00680 }
00681
00682 // キャプチャの開始と停止
00683 if (capture)
00684 {
00685     // キャプチャスレッドが重いしているので止める
00686     if (ImGui::Button(u8"停止")) capture.stop();
00687     ImGui::SameLine();
00688     ImGui::TextColored(ImVec4(0.2f, 1.0f, 0.0f, 1.0f), "%s", u8"取得中");
00689 }
00690 else
00691 {
00692     // キャプチャスレッドが止まっているので
00693     if (ImGui::Button(u8"開始") && deviceNumber >= 0)
00694     {
00695         // キャプチャデバイスを開いてから
00696         openDevice();
00697
00698         // キャプチャスレッドを動かす
00699         capture.start();
00700     }
00701     ImGui::SameLine();
00702     ImGui::TextColored(ImVec4(1.0f, 0.2f, 0.0f, 1.0f), "%s", u8"停止中");
00703 }
00704 ImGui::End();
00705 }
00706
00707 // 情報パネル
00708 if (showInformationPanel)
00709 {
00710     // ウィンドウの位置とサイズ
00711     ImGui::SetNextWindowPos(ImVec2(235.0f, 2.0f + menubarHeight), ImGuiCond_Once);
00712     ImGui::SetNextWindowSize(ImVec2(214, 134), ImGuiCond_Once);
00713     ImGui::Begin(u8"情報", &showInformationPanel);
00714
00715     // フレームレートの表示
00716     ImGui::Text("Frame rate: %.2f fps", ImGui::GetIO().Framerate);
00717
00718     // 検出数の表示
00719     ImGui::Text("Detected corners: %d", calibration.getCornerCount());
00720
00721     // 標本数の表示
00722     ImGui::Text("Sampled corners: %d", calibration.getCornerCount());
00723
00724     // 再投影誤差の表示
00725     ImGui::Text("Reprojection error: %.6f", calibration.getReprojectionError());
00726
00727     ImGui::End();
00728 }
00729
00730 // エラーメッセージが設定されていたら
00731 if (errorMessage)
00732 {
00733     // ウィンドウの位置・サイズとタイトル
00734     ImGui::SetNextWindowPos(ImVec2(60, 60), ImGuiCond_Once);
00735     ImGui::SetNextWindowSize(ImVec2(240, 92), ImGuiCond_Always);
00736
00737     // ウィンドウを表示するとき true
00738     bool status{ true };
00739
00740     // エラーメッセージウィンドウを表示する
00741     ImGui::Begin(u8"エラー", &status);
00742
00743     // エラーメッセージの表示
00744     ImGui::TextColored(ImVec4(1.0f, 0.2f, 0.0f, 1.0f), "%s", errorMessage);
00745
00746     // クローズボックスか「閉じる」ボタンをクリックしたら
00747     if (!status || ImGui::Button(u8"閉じる"))
00748     {
00749         // エラーメッセージを消去する
00750         errorMessage = nullptr;
00751     }
00752     ImGui::End();
00753 }
00754
00755 // ImGui のフレームに描画する
00756 ImGui::Render();
00757 }
00758 }
```

```

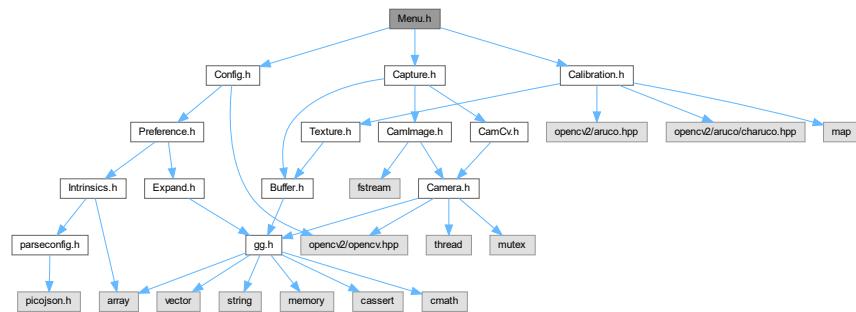
00759 // 画像の保存
00760 // 画像の保存
00761 //
00762 void Menu::saveImage(const cv::Mat& image, const std::string& filename)
00763 {
00764     // 画像ファイル名のフィルタ
00765     constexpr nfdfilteritem_t imageFilter[] { "Images", "png,jpg,jpeg,jfif,bmp,dib" };
00766
00767     // ファイルダイアログから得るパス
00768     nfdchart* filepath;
00769
00770     // ファイルダイアログを開く
00771     if (NFD_SaveDialog(&filepath, imageFilter, 1, NULL, filename.c_str()) == NFD_OKAY)
00772     {
00773         // ファイルに保存する
00774         if (!CamImage::save(filepath, image)) errorMessage = u8"ファイルが保存できませんでした";
00775
00776         // ファイルバスの取り出しに使ったメモリを開放する
00777         NFD_FreePath(filepath);
00778     }
00779 }

```

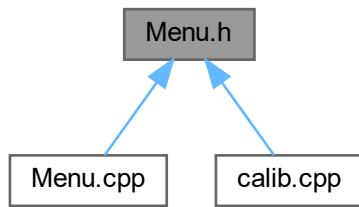
## 9.49 Menu.h ファイル

```
#include "Config.h"
#include "Capture.h"
#include "Calibration.h"
```

Menu.h の依存先関係図:



被依存関係図:



## クラス

- class Menu

### 9.49.1 詳解

メニューの描画クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

Menu.h に定義があります。

## 9.50 Menu.h

### [詳解]

```
00001 #pragma once
00002
00010
00011 // 構成データ
00012 #include "Config.h"
00013
00014 // キャプチャデバイス
00015 #include "Capture.h"
00016
00017 // 軸正オブジェクト
00018 #include "Calibration.h"
00019
00023 class Menu
00024 {
00026     const Config& config;
00027
00029     Settings settings;
00030
00032     Intrinsics intrinsics;
00033
00035     Capture& capture;
00036
00038     Calibration& calibration;
00039
00041     int deviceNumber;
00042
00044     int codecNumber;
00045
00047     int preferenceNumber;
00048
00049     cv::VideoCaptureAPIs backend;
00051
00053     GgMatrix pose;
00054
00056     GLsizei menubarHeight;
00057
00059     bool showControlPanel;
00060
00062     bool showInformationPanel;
00063
00065     bool quit;
00066
00068     const char* errorMessage;
00069
00073     void openImage();
00074
00078     void openMovie();
00079
00083     void openDevice();
00084
00088     void loadConfig();
00089
00093     void saveConfig();
00094
```

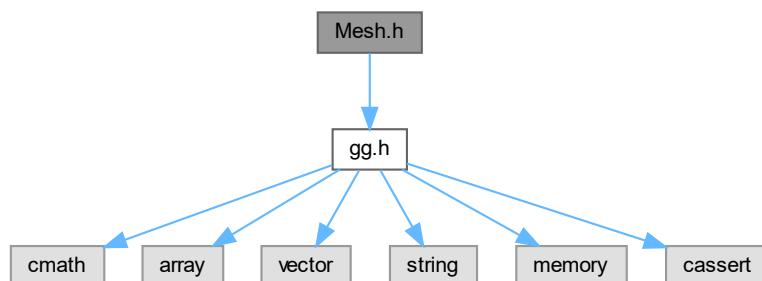
```

00098 void loadParameters();
00099
00103 void saveParameters();
00104
00108 void calibrateByFiles();
00109
00115 const auto& getPreference(int i) const
00116 {
00117     return config.preferenceList[i];
00118 }
00119
00123 const auto& getPreference() const
00124 {
00125     return getPreference(preferenceNumber);
00126 }
00127
00128 public:
00129     bool detectMarker;
00132
00134     bool detectBoard;
00135
00143     Menu(const Config& config, Capture& capture, Calibration& calibration);
00144
00150     Menu(const Menu& menu) = delete;
00151
00155     virtual ~Menu();
00156
00162     Menu& operator=(const Menu& menu) = delete;
00163
00169     explicit operator bool() const
00170 {
00171     return !quit;
00172 }
00173
00179     const auto& getPose() const
00180 {
00181     return pose;
00182 }
00183
00189     auto getMenubarHeight() const
00190 {
00191     return menubarHeight;
00192 }
00193
00204     void setSize(const std::array<int, 2>& size);
00205
00215     std::array<GLsizei, 2> setup(GLfloat aspect) const;
00216
00220     void draw();
00221
00228     void saveImage(const cv::Mat& image, const std::string& filename = "*.jpg");
00229 };

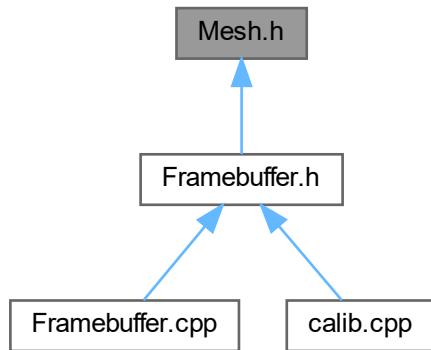
```

## 9.51 Mesh.h ファイル

#include "gg.h"  
Mesh.h の依存先関係図:



被依存関係図:



クラス

- class [Mesh](#)

### 9.51.1 詳解

メッシュの描画クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Mesh.h](#) に定義があります。

## 9.52 Mesh.h

### [詳解]

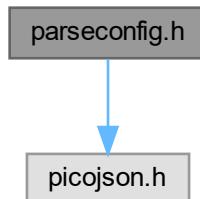
```
00001 #pragma once
00002
00010
00011 // 補助プログラム
00012 #include "gg.h"
00013
00017 class Mesh
00018 {
00020     const GLuint vao;
00021
00022 public:
00023
```

```
00027     Mesh()
00028     : vao{ [] { GLuint vao; glGenVertexArrays(1, &vao); return vao; }() }
00029     {
00030         // 頂点配列オブジェクトが作れなかったら落とす
00031         assert(vao);
00032     }
00033
00037     Mesh(const Mesh& mesh) = delete;
00038
00042     virtual ~Mesh()
00043     {
00044         glDeleteVertexArrays(1, &vao);
00045     }
00046
00050     Mesh& operator=(const Mesh& mesh) = delete;
00051
00057     void draw(const std::array<GLsizei, 2>& size) const
00058     {
00059         // 描画
00060         glBindVertexArray(vao);
00061         glDrawArraysInstanced(GL_TRIANGLE_STRIP, 0, size[0] * 2, size[1] - 1);
00062         glBindVertexArray(0);
00063     }
00064 };
```

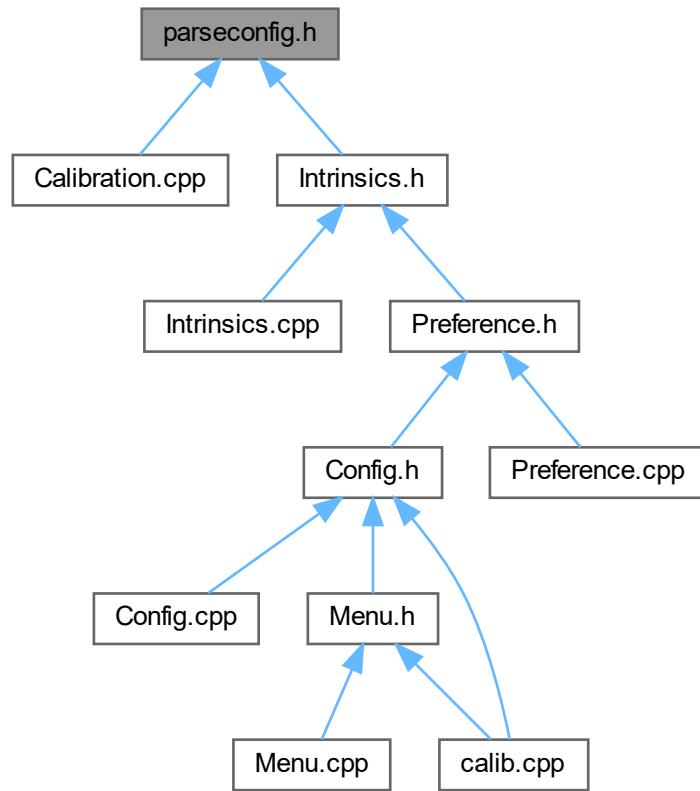
## 9.53 parseconfig.h ファイル

```
#include "picojson.h"
```

parseconfig.h の依存先関係図:



被依存関係図:



## 関数

- template<typename T >  
bool **getValue** (const picojson::object &object, const std::string &key, T &scalar)
- template<typename T , size\_t U>  
bool **getValue** (const picojson::object &object, const std::string &key, std::array< T, U > &vector)
- bool **getString** (const picojson::object &object, const std::string &key, std::string &string)
- template<size\_t U>  
bool **getString** (const picojson::object &object, const std::string &key, std::array< std::string, U > &strings)
- bool **getString** (const picojson::object &object, const std::string &key, std::vector< std::string > &strings)
- template<typename T >  
void **setValue** (picojson::object &object, const std::string &key, const T &scalar)
- template<typename T , size\_t U>  
void **setValue** (picojson::object &object, const std::string &key, const std::array< T, U > &vector)
- void **setString** (picojson::object &object, const std::string &key, const std::string &string)
- template<size\_t U>  
void **setString** (picojson::object &object, const std::string &key, const std::array< std::string, U > &strings)
- void **setString** (picojson::object &object, const std::string &key, const std::vector< std::string > &strings)

### 9.53.1 詳解

構成ファイルの読み取り補助

著者

Kohe Tokoi

日付

November 15, 2022

[parseconfig.h](#) に定義があります。

### 9.53.2 関数詳解

#### 9.53.2.1 getString() [1/3]

```
template<size_t U>
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトから文字列の配列を取得する

テンプレート引数

<i>U</i>	構成ファイルから取得する配列の要素の文字列の数
----------	-------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

[parseconfig.h](#) の 105 行目に定義があります。

#### 9.53.2.2 getString() [2/3]

```
bool getString (
    const picojson::object & object,
```

```
const std::string & key,
std::string & string ) [inline]
```

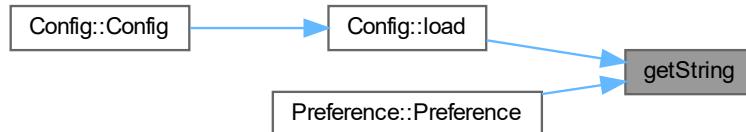
構成ファイルの JSON オブジェクトから文字列を取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>string</i>	取得した文字列を格納する変数

parseconfig.h の 81 行目に定義があります。

被呼び出し関係図:



### 9.53.2.3 getString() [3/3]

```
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトから文字列のベクトルを取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

parseconfig.h の 138 行目に定義があります。

### 9.53.2.4 getValue() [1/2]

```
template<typename T , size_t U>
```

```
bool getValue (
    const picojson::object & object,
    const std::string & key,
    std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトから数値の配列を取得する

テンプレート引数

<i>T</i>	構成ファイルから取得する配列の要素の数値のデータ型
<i>U</i>	構成ファイルから取得する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>vector</i>	取得した数値の配列を格納する変数

parseconfig.h の 48 行目に定義があります。

### 9.53.2.5 `getValue()` [2/2]

```
template<typename T >
bool getValue (
    const picojson::object & object,
    const std::string & key,
    T & scalar )
```

構成ファイルの JSON オブジェクトから数値を取得する

テンプレート引数

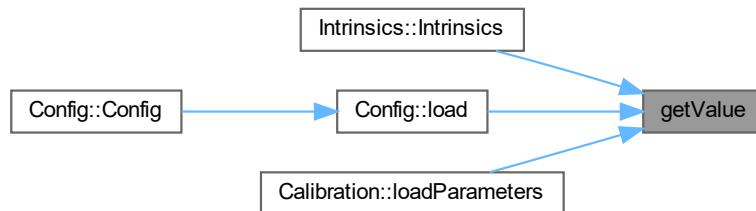
<i>T</i>	構成ファイルから取得する数値のデータ型
----------	---------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>scalar</i>	取得した JSON オブジェクトの数値を格納する変数

parseconfig.h の 23 行目に定義があります。

被呼び出し関係図:



### 9.53.2.6 setString() [1/3]

```
template<size_t U>
void setString (
    picojson::object & object,
    const std::string & key,
    const std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトに文字列の配列を設定する

テンプレート引数

$U$	構成ファイルに設定する配列の要素の文字列の数
-----	------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

`parseconfig.h` の 225 行目に定義があります。

### 9.53.2.7 setString() [2/3]

```
void setString (
    picojson::object & object,
    const std::string & key,
    const std::string & string ) [inline]
```

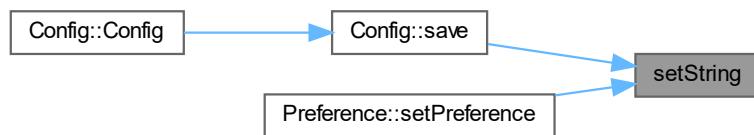
構成ファイルの JSON オブジェクトに文字列を設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>string</i>	設定する文字列

parseconfig.h の 210 行目に定義があります。

被呼び出し関係図:



### 9.53.2.8 setString() [3/3]

```
void setString (
    picojson::object & object,
    const std::string & key,
    const std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトに文字列のベクトルを設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

parseconfig.h の 250 行目に定義があります。

### 9.53.2.9 setValue() [1/2]

```
template<typename T , size_t U>
void setValue (
    picojson::object & object,
```

```
const std::string & key,  
const std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトに数値の配列を設定する

テンプレート引数

<i>T</i>	構成ファイルに設定する配列の要素の数値のデータ型
<i>U</i>	構成ファイルに設定する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>vector</i>	設定する数値の配列

[parseconfig.h](#) の 185 行目に定義があります。

#### 9.53.2.10 setValue() [2/2]

```
template<typename T >
void setValue (
    picojson::object & object,
    const std::string & key,
    const T & scalar )
```

構成ファイルの JSON オブジェクトに数値を設定する

テンプレート引数

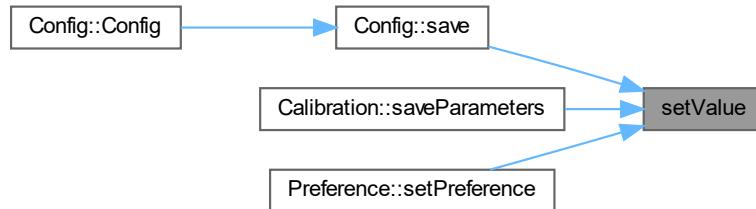
<i>T</i>	構成ファイルに設定する数値のデータ型
----------	--------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>scalar</i>	設定する数値

[parseconfig.h](#) の 169 行目に定義があります。

被呼び出し関係図:



## 9.54 parseconfig.h

### [詳解]

```

00001 #pragma once
00002
00010
00011 // JSON
00012 #include "picojson.h"
00013
00022 template <typename T>
00023 bool getValue(const picojson::object& object,
00024     const std::string& key, T& scalar)
00025 {
00026     // key に一致するオブジェクトを探す
00027     const auto&& value{ object.find(key) };
00028
00029     // オブジェクトが無いか数値でなかったら戻る
00030     if (value == object.end() || !value->second.is<double>()) return false;
00031
00032     // 数値として格納する
00033     scalar = static_cast<T>(value->second.get<double>());
00034
00035     return true;
00036 }
00037
00047 template <typename T, size_t U>
00048 bool getValue(const picojson::object& object,
00049     const std::string& key, std::array<T, U>& vector)
00050 {
00051     // key に一致するオブジェクトを探す
00052     const auto&& value{ object.find(key) };
00053
00054     // オブジェクトが無いか配列でなかったら戻る
00055     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00056
00057     // 配列を取り出す
00058     const auto& array{ value->second.get<picojson::array>() };
00059
00060     // 配列の要素数とデータの格納先の要素数の少ない方の数
00061     const auto n{ std::min(U, array.size()) };
00062
00063     // 配列の要素について
00064     for (size_t i = 0; i < n; ++i)
00065     {
00066         // 要素が数値なら格納する
00067         if (array[i].is<double>()) vector[i] = static_cast<T>(array[i].get<double>());
00068     }
00069
00070     return true;
00071 }
00072
00080 inline
00081 bool getString(const picojson::object& object,
00082     const std::string& key, std::string& string)
00083 {
00084     // key に一致するオブジェクトを探す
00085     const auto&& value{ object.find(key) };
00086
  
```

```

00087 // オブジェクトが無いか文字列でなかったら戻る
00088 if (value == object.end() || !value->second.is<std::string>()) return false;
00089
00090 // 文字列として格納する
00091 string = value->second.get<std::string>();
00092
00093 return true;
00094 }
00095
00104 template <size_t U>
00105 bool getString(const picojson::object& object,
00106 const std::string& key, std::array<std::string, U>& strings)
00107 {
00108 // key に一致するオブジェクトを探す
00109 const auto&& value{ object.find(key) };
00110
00111 // オブジェクトが無いか配列でなかったら戻る
00112 if (value == object.end() || !value->second.is<picojson::array>()) return false;
00113
00114 // 配列を取り出す
00115 const auto& array{ value->second.get<picojson::array>() };
00116
00117 // 配列の要素数とデータの格納先の要素数の少ない方の数
00118 const auto n{ std::min(U, array.size()) };
00119
00120 // 配列の要素について
00121 for (size_t i = 0; i < n; ++i)
00122 {
00123 // 要素が文字列なら文字列として格納する
00124 strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
00125 }
00126
00127 return true;
00128 }
00129
00137 inline
00138 bool getString(const picojson::object& object,
00139 const std::string& key, std::vector<std::string>& strings)
00140 {
00141 // key に一致するオブジェクトを探す
00142 const auto&& value{ object.find(key) };
00143
00144 // オブジェクトが無いか配列でなかったら戻る
00145 if (value == object.end() || !value->second.is<picojson::array>()) return false;
00146
00147 // 配列を取り出す
00148 const auto& array{ value->second.get<picojson::array>() };
00149
00150 // 配列のすべての要素について
00151 for (auto& element : array)
00152 {
00153 // 要素が文字列なら文字列として格納する
00154 strings.emplace_back(element.is<std::string>() ? element.get<std::string>() : "");
00155 }
00156
00157 return true;
00158 }
00159
00168 template <typename T>
00169 void setValue(picojson::object& object,
00170 const std::string& key, const T& scalar)
00171 {
00172 object.emplace(key, picojson::value(static_cast<double>(scalar)));
00173 }
00174
00184 template <typename T, size_t U>
00185 void setValue(picojson::object& object,
00186 const std::string& key, const std::array<T, U>& vector)
00187 {
00188 // picojson の配列
00189 picojson::array array;
00190
00191 // 配列のすべての要素について
00192 for (const auto& element : vector)
00193 {
00194 // 要素を picojson::array に追加する
00195 array.emplace_back(picojson::value(static_cast<double>(element)));
00196 }
00197
00198 // オブジェクトに追加する
00199 object.emplace(key, array);
00200 }
00201
00209 inline
00210 void setString(picojson::object& object,
00211 const std::string& key, const std::string& string)
00212 {

```

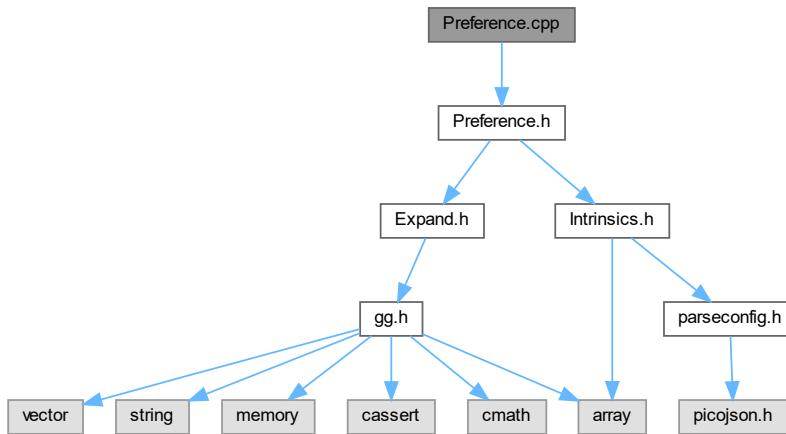
```

00213     object.emplace(key, picojson::value(string));
00214 }
00215
00224 template <size_t U>
00225 void setString(picojson::object& object,
00226   const std::string& key, const std::array<std::string, U>& strings)
00227 {
00228     // picojson の配列
00229     picojson::array array;
00230
00231     // 配列のすべての要素について
00232     for (const auto& string : strings)
00233     {
00234         // 要素を picojson::array に追加する
00235         array.emplace_back(picojson::value(string));
00236     }
00237
00238     // オブジェクトに追加する
00239     object.emplace(key, array);
00240 }
00241
00249 inline
00250 void setString(picojson::object& object,
00251   const std::string& key, const std::vector<std::string>& strings)
00252 {
00253     // picojson の配列
00254     picojson::array array;
00255
00256     // 配列のすべての要素について
00257     for (auto& string : strings)
00258     {
00259         // 要素を picojson::array に追加する
00260         array.emplace_back(picojson::value(string));
00261     }
00262
00263     // オブジェクトに追加する
00264     object.emplace(key, array);
00265 }

```

## 9.55 Preference.cpp ファイル

#include "Preference.h"  
 Preference.cpp の依存先関係図:



### 9.55.1 詳解

キャプチャデバイスの構成データクラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

[Preference.cpp](#) に定義があります。

## 9.56 Preference.cpp

[詳解]

```

00001
00008 #include "Preference.h"
00009
00010 //
00011 // キャプチャデバイスの構成データのデフォルトコンストラクタ
00012 //
00013 Preference::Preference()
00014 : Preference{ "Default", "orthographic.vert", "normal.frag" }
00015 {
00016 }
00017
00018 //
00019 // シェーダのソースファイルを指定するときに使う
00020 // キャプチャデバイスの構成データのコンストラクタ
00021 //
00022 Preference::Preference(const std::string& description,
00023 const std::string& vert, const std::string& frag,
00024 const Intrinsics& intrinsics)
00025 : description{ description }
00026 , source{ vert, frag }
00027 , intrinsics{ intrinsics }
00028 , shader{ nullptr }
00029 {
00030 }
00031
00032 //
00033 // 構成ファイルを読み込むときに使う
00034 // キャプチャデバイスの構成データのコンストラクタ
00035 //
00036 Preference::Preference(const picojson::object& object)
00037 : intrinsics{ object }
00038 , shader{ nullptr }
00039 {
00040 // 説明の文字列
00041 getString(object, "description", description);
00042
00043 // 展開用シェーダのファイル名
00044 getString(object, "shader", source);
00045 }
00046
00047 //
00048 // 構成データのデストラクタ
00049 //
00050 Preference::~Preference()
00051 {
00052 // static メンバにしている std::map の中身を先に消去しておく
00053 shaderList.clear();
00054 }
00055
00056 //
00057 // シェーダをビルドする
00058 //
00059 void Preference::buildShader()
00060 {
00061 // ソースファイル名をつないでシェーダを検索するキーを作る
00062 const auto key{ source[0] + "\t" + source[1] };
00063
00064 // キーがシェーダリストに無ければシェーダを構築して追加する
00065 shader = &shaderList.try_emplace(key, source[0], source[1]).first->second;
00066 }
00067
00068 //

```

```

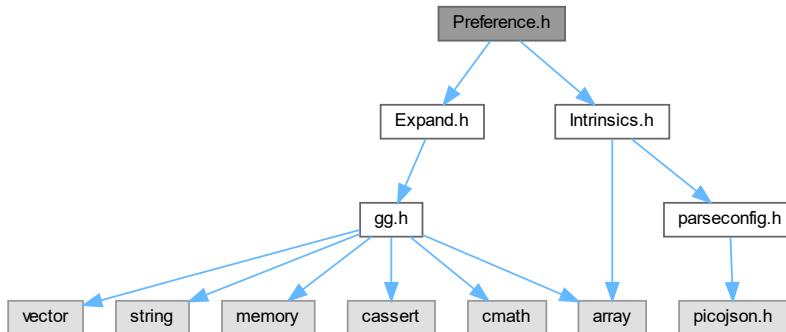
00069 // 構成を JSON オブジェクトに格納する
00070 //
00071 void Preference::setPreference(picojson::object& object) const
00072 {
00073     // 説明の文字列
00074     setString(object, "description", description);
00075
00076     // 展開用シェーダのファイル名
00077     setString(object, "shader", source);
00078
00079     // キャプチャデバイスのレンズの縦横の画角
00080     setValue(object, "fov", intrinsics.fov);
00081
00082     // キャプチャデバイスのレンズの中心（主点）位置
00083     setValue(object, "center", intrinsics.center);
00084
00085     // キャプチャデバイスの解像度
00086     setValue(object, "resolution", intrinsics.size);
00087
00088     // キャプチャデバイスのフレームレート
00089     setValue(object, "fps", intrinsics.fps);
00090 }
00091
00092 // すべての構成のシェーダーのリスト
00093 std::map<std::string, Expand> Preference::shaderList;

```

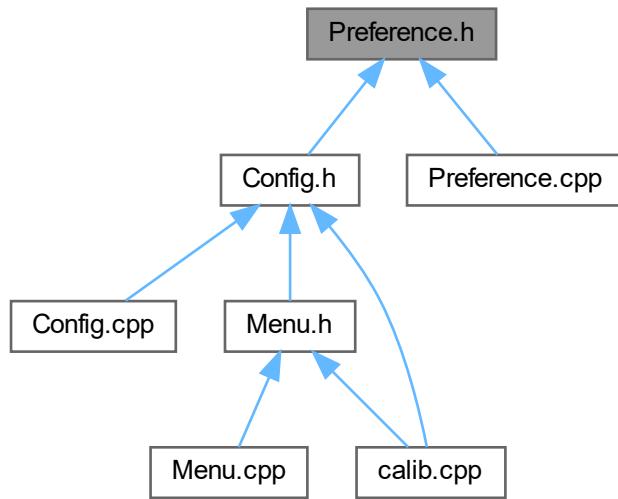
## 9.57 Preference.h ファイル

```
#include "Expand.h"
#include "Intrinsics.h"
```

Preference.h の依存先関係図:



被依存関係図:



クラス

- class [Preference](#)

### 9.57.1 詳解

キャプチャデバイスの構成データクラスの定義

著者

Kohe Tokoi

日付

March 6, 2024

[Preference.h](#) に定義があります。

## 9.58 Preference.h

### [詳解]

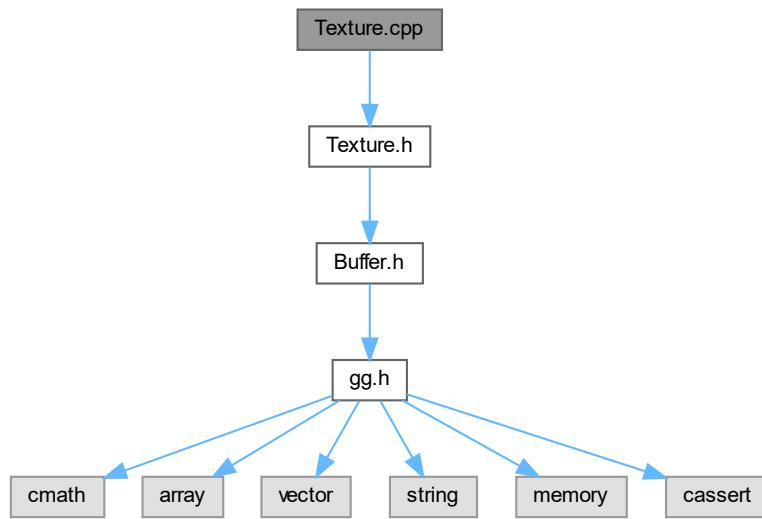
```
00001 #pragma once
00002
00010
00011 // 展開用シェーダ
00012 #include "Expand.h"
00013
00014 // キャプチャデバイス固有のパラメータ
00015 #include "Intrinsics.h"
00016
00020 class Preference
00021 {
00023     std::string description;
00024
00026     std::array<std::string, 2> source;
00027
00029     const Intrinsics intrinsics;
00030
00032     const Expand* shader;
00033
00035     static std::map<std::string, Expand> shaderList;
00036
00037 public:
00038
00044     Preference();
00045
00055     Preference(const std::string& description,
00056         const std::string& vert, const std::string& frag,
00057         const Intrinsics& intrinsics = Intrinsics{});
00058
00063     Preference(const picojson::object& object);
00064
00068     virtual ~Preference();
00069
00073     void buildShader();
00074
00080     const auto& getDescription() const
00081     {
00082         return description;
00083     }
00084
00090     const auto& getIntrinsics() const
00091     {
00092         return intrinsics;
00093     }
00094
00100     const auto& getShader() const
00101     {
00102         return *shader;
00103     }
00104
00110     void setPreference(picojson::object& object) const;
00111 };
```

## 9.59 README.md ファイル

## 9.60 Texture.cpp ファイル

```
#include "Texture.h"
```

Texture.cpp の依存先関係図:



### 9.60.1 詳解

テクスチャクラスの実装

著者

Kohe Tokoi

日付

February 20, 2024

[Texture.cpp](#) に定義があります。

## 9.61 Texture.cpp

### [詳解]

```

00001
00008 #include "Texture.h"
00009
00010 //
00011 // テクスチャを作成する
00012 //
00013 void Texture::create(GLsizei width, GLsizei height, int channels)
00014 {
00015     // 既存のバッファを破棄して新しいバッファを作成する
00016     Buffer::create(width, height, channels);
00017
00018     // 指定したサイズが保持しているテクスチャのサイズと同じなら何もしない
00019     if (width == textureSize[0] && height == textureSize[1]
00020         && channels == textureChannels) return;
00021
00022     // テクスチャのサイズとチャンネル数を記録する
00023     textureSize = std::array<int, 2>{ width, height };
00024     textureChannels = channels;
00025
00026     // 以前のテクスチャを削除する
00027     glDeleteTextures(1, &textureName);
00028
00029     // テクスチャを作成する
00030     glGenTextures(1, &textureName);
00031
00032     // テクスチャのメモリを確保してデータを転送する
00033     glBindTexture(GL_TEXTURE_2D, textureName);
00034     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0,
00035                 channelsToFormat(channels), GL_UNSIGNED_BYTE, nullptr);
00036     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00037     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00038     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00039     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00040     glBindTexture(GL_TEXTURE_2D, 0);
00041 }
00042
00043 //
00044 // テクスチャをコピーする
00045 //
00046 void Texture::copy(const Buffer& texture) noexcept
00047 {
00048     // コピー元と同じサイズの空のテクスチャを作る
00049     Texture::create(texture.getWidth(), texture.getHeight(),
00050                     texture.getChannels());
00051
00052     // 作成したテクスチャのバッファにコピーする
00053     copyBuffer(texture);
00054 }
00055
00056 //
00057 // 代入演算子
00058 //
00059 Texture& Texture::operator=(const Texture& texture)
00060 {
00061     // 代入元と代入先が同じでなければ
00062     if (&texture != this)
00063     {
00064         // 引数のテクスチャをこのテクスチャにコピーする
00065         Texture::copy(texture);
00066     }
00067
00068     // このテクスチャを返す
00069     return *this;
00070 }
00071
00072 //
00073 // ムーブ代入演算子
00074 //
00075 Texture& Texture::operator=(Texture&& texture) noexcept
00076 {
00077     // 代入元と代入先が同じでなければ
00078     if (&texture != this)
00079     {
00080         // 引数のテクスチャをこのテクスチャにコピーする
00081         Texture::copy(texture);
00082
00083         // ムーブ元のバッファを破棄する
00084         texture.Buffer::discard();
00085
00086         // ムーブ元のテクスチャを破棄する
00087         texture.Texture::discard();
00088     }

```

```

00089 // このテクスチャを返す
00090     return *this;
00091 }
00092 }
00093 //
00094 //
00095 // テクスチャを破棄する
00096 //
00097 void Texture::discard()
00098 {
00099     // デフォルトのテクスチャに戻す
00100     glBindTexture(GL_TEXTURE_2D, 0);
00101
00102     // テクスチャを削除する
00103     glDeleteTextures(1, &textureName);
00104     textureName = 0;
00105
00106     // テクスチャのサイズを 0 にする
00107     textureSize = std::array<int, 2>{ 0, 0 };
00108     textureChannels = 0;
00109 }
00110
00111 //
00112 // テクスチャからピクセルバッファオブジェクトにデータをコピーする
00113 //
00114 void Texture::readPixels(
00115 #if defined(USE_PIXEL_BUFFER_OBJECT)
00116     GLuint buffer) const
00117 #else
00118     std::vector<GLubyte>& buffer)
00119 #endif
00120 {
00121     // 読み出し元のテクスチャを結合する
00122     glBindTexture(GL_TEXTURE_2D, textureName);
00123
00124 #if defined(USE_PIXEL_BUFFER_OBJECT)
00125     // 書き込み先のピクセルバッファオブジェクトを指定する
00126     glBindBuffer(GL_PIXEL_PACK_BUFFER, buffer);
00127
00128     // テクスチャの内容をピクセルバッファオブジェクトに書き込む
00129     glGetTexImage(GL_TEXTURE_2D, 0, getFormat(), GL_UNSIGNED_BYTE, 0);
00130
00131     // 書き込み先のピクセルバッファオブジェクトの結合を解除する
00132     glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
00133
00134 #else
00135
00136     // テクスチャの内容をピクセルバッファオブジェクトに書き込む
00137     glGetTexImage(GL_TEXTURE_2D, 0, getFormat(), GL_UNSIGNED_BYTE, buffer.data());
00138
00139 #endif
00140
00141     // 読み出し元のテクスチャの結合を解除する
00142     glBindTexture(GL_TEXTURE_2D, 0);
00143 }
00144
00145 //
00146 // ピクセルバッファオブジェクトからテクスチャにデータをコピーする
00147 //
00148 void Texture::drawPixels(
00149 #if defined(USE_PIXEL_BUFFER_OBJECT)
00150     GLuint
00151 #else
00152     const std::vector<GLubyte>&
00153 #endif
00154     buffer) const
00155 {
00156     // 書き込み先のテクスチャを結合する
00157     glBindTexture(GL_TEXTURE_2D, textureName);
00158
00159 #if defined(USE_PIXEL_BUFFER_OBJECT)
00160
00161     // 読み出し元のピクセルバッファオブジェクトを指定する
00162     glBindBuffer(GL_PIXEL_UNPACK_BUFFER, buffer);
00163
00164     // ピクセルバッファオブジェクトの内容をテクスチャに書き込む
00165     glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, textureSize[0], textureSize[1],
00166         getFormat(), GL_UNSIGNED_BYTE, 0);
00167
00168     // 読み出し元のピクセルバッファオブジェクトの結合を解除する
00169     glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
00170
00171 #else
00172
00173     // ピクセルバッファオブジェクトの内容をテクスチャに書き込む
00174     glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, textureSize[0], textureSize[1],
00175         getFormat(), GL_UNSIGNED_BYTE, buffer.data());

```

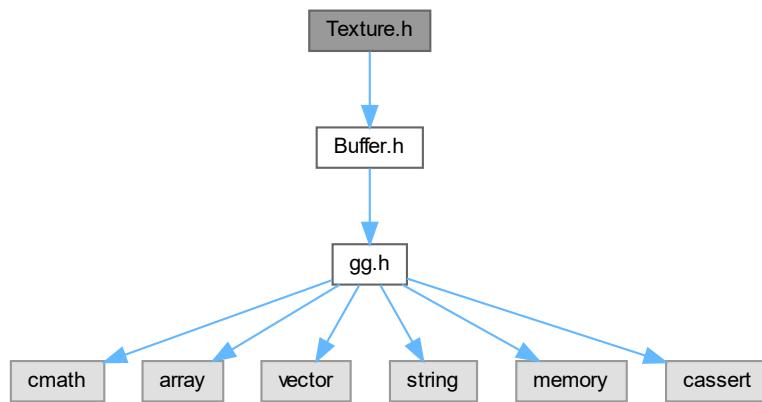
```

00176
00177 #endif
00178 // 書き込み先のテクスチャの結合を解除する
00180 glBindTexture(GL_TEXTURE_2D, 0);
00181 }

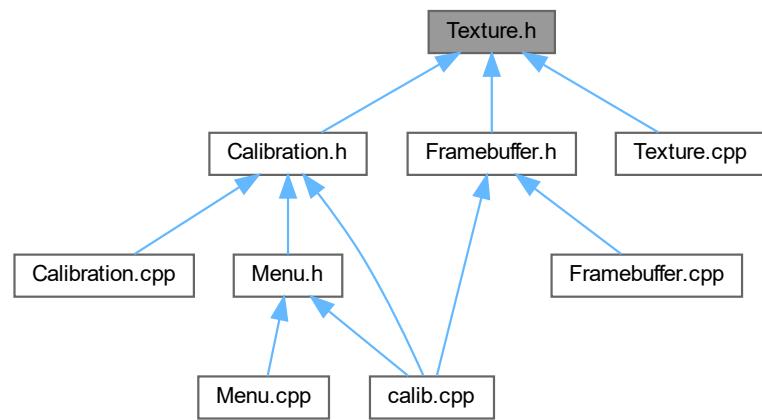
```

## 9.62 Texture.h ファイル

#include "Buffer.h"  
 Texture.h の依存先関係図:



被依存関係図:



クラス

- class Texture

### 9.62.1 詳解

テクスチャクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Texture.h](#) に定義があります。

## 9.63 Texture.h

### [詳解]

```

00001 #pragma once
00002
00010
00011 // パッファクラス
00012 #include "Buffer.h"
00013
00017 class Texture : public Buffer
00018 {
00020     std::array<int, 2> textureSize;
00021
00023     int textureChannels;
00024
00026     GLuint textureName;
00027
00028 public:
00029
00033     Texture()
00034         : Buffer{}
00035         , textureSize{ 0, 0 }
00036         , textureChannels{ 0 }
00037         , textureName{ 0 }
00038     {
00039     }
00040
00048     Texture(GLsizei width, GLsizei height, int channels)
00049     {
00050         Texture::create(width, height, channels);
00051     }
00052
00058     Texture(const Texture& texture)
00059     {
00060         Texture::copy(texture);
00061     }
00062
00068     Texture(Texture&& texture) noexcept
00069     {
00070         *this = std::move(texture);
00071     }
00072
00076     virtual ~Texture()
00077     {
00078         Texture::discard();
00079     }
00080
00087     Texture& operator=(const Texture& texture);
00088
00095     Texture& operator=(Texture&& texture) noexcept;
00096
00108     virtual void create(GLsizei width, GLsizei height, int channels);
00109
00122     virtual void copy(const Buffer& texture) noexcept;
00123
00127     virtual void discard();
00128

```

```
00134     auto getTextureName() const
00135     {
00136         return textureName;
00137     }
00138
00142     virtual const std::array<GLsizei, 2>& getSize() const
00143     {
00144         return textureSize;
00145     }
00146
00150     virtual int getChannels() const
00151     {
00152         return textureChannels;
00153     }
00154
00164     void bindTexture(int unit = 0) const
00165     {
00166         // テクスチャユニットを指定する
00167         glActiveTexture(GL_TEXTURE0 + unit);
00168
00169         // テクスチャを指定する
00170         glBindTexture(GL_TEXTURE_2D, textureName);
00171     }
00172
00176     void unbindTexture() const
00177     {
00178         // デフォルトのテクスチャユニットに戻す
00179         glActiveTexture(GL_TEXTURE0);
00180
00181         // デフォルトのテクスチャに戻す
00182         glBindTexture(GL_TEXTURE_2D, 0);
00183     }
00184
00193     void readPixels(
00194 #if defined(USE_PIXEL_BUFFER_OBJECT)
00195         GLuint buffer) const
00196 #else
00197         std::vector<GLubyte>& buffer)
00198 #endif
00199     ;
00200
00209     void readPixels(Buffer& buffer)
00210 #if defined(USE_PIXEL_BUFFER_OBJECT)
00211     const
00212 #endif
00213     {
00214         // このテクスチャのバッファから引数のオブジェクトのバッファにコピーする
00215         readPixels();
00216
00217         // 引数のオブジェクトのサイズをこのテクスチャに合わせてコピーする
00218         buffer.Buffer::copy(*this);
00219     }
00220
00224     void readPixels()
00225 #if defined(USE_PIXEL_BUFFER_OBJECT)
00226     const
00227 #endif
00228     {
00229         // このテクスチャからこのバッファにコピーする
00230         readPixels(getBufferName());
00231     }
00232
00241     void drawPixels(
00242 #if defined(USE_PIXEL_BUFFER_OBJECT)
00243         GLuint
00244 #else
00245         const std::vector<GLubyte>&
00246 #endif
00247         buffer) const;
00248
00257     void drawPixels(const Texture& texture)
00258     {
00259         // このテクスチャのサイズを引数のオブジェクトのサイズに合わせてコピーする
00260         copy(texture);
00261
00262         // バッファからこのテクスチャにコピーする
00263         drawPixels();
00264     }
00265
00274     void drawPixels(const Buffer& buffer)
00275     {
00276         // このテクスチャのサイズを引数のオブジェクトのサイズに合わせてコピーする
00277         copy(buffer);
00278
00279         // バッファからこのテクスチャにコピーする
00280         drawPixels();
00281     }
```

```
00282
00286     void drawPixels()
00287 #if defined(USE_PIXEL_BUFFER_OBJECT)
00288     const
00289 #endif
00290 {
00291     // このバッファからこのテクスチャにコピーする
00292     drawPixels(getBufferName());
00293 }
00294 };
```

# Index

- ggError
  - gg, 16
- ggFBOError
  - gg, 17
- ~Buffer
  - Buffer, 92
- ~Calibration
  - Calibration, 108
- ~CamCv
  - CamCv, 120
- ~CamImage
  - CamImage, 140
- ~Camera
  - Camera, 129
- ~Capture
  - Capture, 144
- ~Config
  - Config, 151
- ~Expand
  - Expand, 160
- ~Framebuffer
  - Framebuffer, 166
- ~GgApp
  - GgApp, 175
- ~GgBuffer
  - gg::GgBuffer< T >, 178
- ~GgColorTexture
  - gg::GgColorTexture, 184
- ~GgElements
  - gg::GgElements, 189
- ~GgNormalTexture
  - gg::GgNormalTexture, 249
- ~GgPointShader
  - gg::GgPointShader, 258
- ~GgPoints
  - gg::GgPoints, 253
- ~GgShader
  - gg::GgShader, 333
- ~GgShape
  - gg::GgShape, 335
- ~GgSimpleObj
  - gg::GgSimpleObj, 339
- ~GgSimpleShader
  - gg::GgSimpleShader, 344
- ~GgTexture
  - gg::GgTexture, 357
- ~GgTrackball
  - gg::GgTrackball, 363
- ~GgTriangles
  - gg::GgTriangles, 373
- ~GgUniformBuffer
  - gg::GgUniformBuffer< T >, 377
- ~GgVertexArray
  - gg::GgVertexArray, 401
- ~LightBuffer
  - gg::GgSimpleShader::LightBuffer, 413
- ~MaterialBuffer
  - gg::GgSimpleShader::MaterialBuffer, 424
- ~Menu
  - Menu, 433
- ~Mesh
  - Mesh, 439
- ~Preference
  - Preference, 442
- ~Texture
  - Texture, 451
- ~Window
  - GgApp::Window, 464

add

- gg::GgQuaternion, 269–271

ambient

- gg::GgSimpleShader::Light, 409
- gg::GgSimpleShader::Material, 421

begin

- gg::GgTrackball, 363

bind

- gg::GgBuffer< T >, 178
- gg::GgTexture, 358
- gg::GgUniformBuffer< T >, 377
- gg::GgVertexArray, 402

bindBuffer

- Buffer, 92

bindFramebuffer

- Framebuffer, 166

BindingPoints

- gg, 16

bindTexture

- Texture, 451

Buffer, 89

- ~Buffer, 92
- bindBuffer, 92
- Buffer, 90–92
- channelsToFormat, 92
- copy, 94
- copyBuffer, 95
- create, 95
- discard, 96

formatToChannels, 96  
 getAspect, 97  
 getBufferName, 97  
 getChannels, 98  
 getData, 98  
 getFormat, 99  
 getHeight, 99  
 getSize, 100  
 getWidth, 101  
 map, 102  
 operator=, 102, 103  
 resize, 104  
 setData, 105  
 unbindBuffer, 105  
 unmap, 106  
**Buffer.cpp**, 489  
**Buffer.h**, 493  
 USE\_PIXEL\_BUFFER\_OBJECT, 494  
**buildShader**  
 Preference, 442  
  
**calib.cpp**, 496  
 CONFIG\_FILE, 497  
**calibrate**  
 Calibration, 108  
**Calibration**, 106  
 ~Calibration, 108  
 calibrate, 108  
 Calibration, 107  
 createBoard, 108  
 detect, 109  
 dictionaryList, 118  
 discardCorners, 110  
 drawBoard, 110  
 finished, 111  
 getAllMarkerPoses, 111  
 getCameraMatrix, 112  
 getCornerCount, 112  
 get Corners Count, 113  
 getDistortionCoefficients, 113  
 getReprojectionError, 113  
 loadParameters, 114  
 operator=, 115  
 recordCorners, 115  
 saveParameters, 115  
 setDictionary, 117  
**Calibration.cpp**, 498  
**Calibration.h**, 504  
**CamCv**, 119  
 ~CamCv, 120  
 CamCv, 120  
 close, 120  
 decreaseExposure, 120  
 decreaseGain, 121  
 getCodec, 121, 122  
 getFps, 122  
 getPosition, 123  
 increaseExposure, 123  
 increaseGain, 123  
 open, 124  
 setExposure, 125  
 setGain, 126  
 setPosition, 126  
**CamCv.h**, 507  
**Camera**, 127  
 ~Camera, 129  
 Camera, 128, 129  
 capture, 129  
 captured, 136  
 channels, 136  
 close, 130  
 copyFrame, 130  
 decreaseExposure, 131  
 decreaseGain, 131  
 frame, 136  
 getChannels, 131  
 getFps, 132  
 getFrames, 132  
 getHeight, 132  
 getSize, 133  
 getWidth, 133  
 in, 136  
 increaseExposure, 133  
 increaseGain, 133  
 interval, 137  
 isRunning, 134  
 mtx, 137  
 operator=, 134  
 out, 137  
 pixels, 137  
 running, 137  
 start, 134  
 stop, 135  
 thr, 138  
 total, 138  
 transmit, 135  
**Camera.h**, 511  
**CamImage**, 138  
 ~CamImage, 140  
 CamImage, 139  
 isOpened, 140  
 load, 140  
 open, 141  
 save, 142  
**CamImage.h**, 515  
**Capture**, 143  
 ~Capture, 144  
 Capture, 144  
 close, 145  
 getFps, 145  
 getSize, 146  
 isOpended, 146  
 openDevice, 146  
 openImage, 147  
 openMovie, 147  
 operator bool, 148  
 retrieve, 148

start, 149  
stop, 149  
capture  
    Camera, 129  
Capture.cpp, 519  
Capture.h, 521  
captured  
    Camera, 136  
center  
    Intrinsics, 407  
channels  
    Camera, 136  
channelsToFormat  
    Buffer, 92  
checkerLength  
    Settings, 446  
close  
    CamCv, 120  
    Camera, 130  
    Capture, 145  
Config, 150  
    ~Config, 151  
    Config, 151  
    getCheckerLength, 152  
    getDeviceCount, 152  
    getDeviceList, 152  
    getDeviceName, 153  
    getDictionaryName, 154  
    getHeight, 154  
    getInitialImage, 154  
    getTitle, 154  
    getWidth, 155  
    initialize, 155  
    load, 155  
    Menu, 157  
    save, 156  
Config.cpp, 523  
    getAnyList, 524  
Config.h, 530  
CONFIG\_FILE  
    calib.cpp, 497  
conjugate  
    gg::GgQuaternion, 272  
copy  
    Buffer, 94  
    Framebuffer, 167  
    gg::GgBuffer< T >, 178  
    gg::GgUniformBuffer< T >, 378  
    Texture, 452  
copyBuffer  
    Buffer, 95  
copyFrame  
    Camera, 130  
create  
    Buffer, 95  
    Framebuffer, 168  
    Texture, 453  
createBoard  
Calibration, 108  
decreaseExposure  
    CamCv, 120  
    Camera, 131  
decreaseGain  
    CamCv, 121  
    Camera, 131  
defaultEuler  
    Settings, 446  
defaultFocal  
    Settings, 446  
defaultFocalRange  
    Settings, 446  
detect  
    Calibration, 109  
detectBoard  
    Menu, 438  
detectMarker  
    Menu, 438  
dictionaryList  
    Calibration, 118  
dictionaryName  
    Settings, 446  
diffuse  
    gg::GgSimpleShader::Light, 410  
    gg::GgSimpleShader::Material, 421  
discard  
    Buffer, 96  
    Framebuffer, 169  
    Texture, 454  
discardCorners  
    Calibration, 110  
distance3  
    gg::GgVector, 386  
distance4  
    gg::GgVector, 387  
divide  
    gg::GgQuaternion, 273–275  
dot3  
    gg::GgVector, 388  
dot4  
    gg::GgVector, 388  
draw  
    Framebuffer, 169  
    gg::GgElements, 189  
    gg::GgPoints, 253  
    gg::GgShape, 336  
    gg::GgSimpleObj, 340  
    gg::GgTriangles, 373  
    Menu, 433  
    Mesh, 439  
drawBoard  
    Calibration, 110  
drawPixels  
    Texture, 454–456  
end  
    gg::GgTrackball, 364

euler  
 gg::GgQuaternion, 275, 276  
 Settings, 446

Expand, 158  
 ~Expand, 160  
 Expand, 158  
 getProgram, 160  
 operator=, 160  
 setup, 160

Expand.cpp, 532

Expand.h, 534

fill  
 gg::GgUniformBuffer< T >, 378

finished  
 Calibration, 111

focal  
 Settings, 447

focalRange  
 Settings, 447

formatToChannels  
 Buffer, 96

fov  
 Intrinsics, 407

fps  
 Intrinsics, 408

frame  
 Camera, 136

Framebuffer, 162  
 ~Framebuffer, 166  
 bindFramebuffer, 166  
 copy, 167  
 create, 168  
 discard, 169  
 draw, 169  
 Framebuffer, 163, 164  
 getChannels, 170  
 getFramebufferName, 170  
 getSize, 170  
 operator=, 171  
 unbindFramebuffer, 172  
 update, 172, 173

Framebuffer.cpp, 537

Framebuffer.h, 541

frustum  
 gg::GgMatrix, 196

get  
 gg::GgMatrix, 197, 198  
 gg::GgPointShader, 258  
 gg::GgQuaternion, 277  
 gg::GgShader, 333  
 gg::GgShape, 336  
 gg::GgSimpleObj, 340  
 gg::GgTrackball, 364  
 gg::GgVertexArray, 402  
 GgApp::Window, 464

getAllMarkerPoses  
 Calibration, 111

getAltArrowX  
 GgApp::Window, 464

getAltArrowY  
 GgApp::Window, 465

getAltArrow  
 GgApp::Window, 466

getAnyList  
 Config.cpp, 524

getArrow  
 GgApp::Window, 467

getArrowX  
 GgApp::Window, 468

getArrowY  
 GgApp::Window, 469

getAspect  
 Buffer, 97  
 GgApp::Window, 469

getBuffer  
 gg::GgBuffer< T >, 179  
 gg::GgPoints, 254  
 gg::GgTriangles, 374  
 gg::GgUniformBuffer< T >, 379

getBufferName  
 Buffer, 97

getCameraMatrix  
 Calibration, 112

getChannels  
 Buffer, 98  
 Camera, 131  
 Framebuffer, 170  
 Texture, 457

getCheckerLength  
 Config, 152

getCodec  
 CamCv, 121, 122

getConjugateMatrix  
 gg::GgQuaternion, 277–279

getControlArrow  
 GgApp::Window, 470

getControlArrowX  
 GgApp::Window, 470

getControlArrowY  
 GgApp::Window, 471

getCornerCount  
 Calibration, 112

getCornersCount  
 Calibration, 113

getCount  
 gg::GgBuffer< T >, 179  
 gg::GgPoints, 254  
 gg::GgTriangles, 374  
 gg::GgUniformBuffer< T >, 379

getData  
 Buffer, 98

getDescription  
 Preference, 442

getDeviceCount  
 Config, 152

getDeviceList  
    Config, 152  
getDeviceName  
    Config, 153  
getDictionaryName  
    Config, 154  
getDistortionCoefficients  
    Calibration, 113  
getFboHeight  
    GgApp::Window, 472  
getFboSize  
    GgApp::Window, 472  
getFboWidth  
    GgApp::Window, 472  
getFocal  
    Settings, 445  
getFormat  
    Buffer, 99  
getFps  
    CamCv, 122  
    Camera, 132  
    Capture, 145  
getFramebufferName  
    Framebuffer, 170  
getFrames  
    Camera, 132  
getHeight  
    Buffer, 99  
    Camera, 132  
    Config, 154  
    gg::GgTexture, 358  
    GgApp::Window, 472  
getIndexBuffer  
    gg::GgElements, 190  
getIndexCount  
    gg::GgElements, 190  
getInitialImage  
    Config, 154  
getIntrinsics  
    Preference, 442  
getKey  
    GgApp::Window, 473  
getLastKey  
    GgApp::Window, 473  
getMatrix  
    gg::GgQuaternion, 279–281  
    gg::GgTrackball, 365  
getMenubarHeight  
    Menu, 434  
getMode  
    gg::GgShape, 337  
getMouse  
    GgApp::Window, 473, 474  
getMouseX  
    GgApp::Window, 474  
getMouseY  
    GgApp::Window, 474  
getPose  
    Menu, 434  
    CamCv, 123  
getProgram  
    Expand, 160  
getQuaternion  
    gg::GgTrackball, 365  
getReprojectionError  
    Calibration, 113  
getRotation  
    GgApp::Window, 474  
getRotationMatrix  
    GgApp::Window, 475  
getScale  
    gg::GgTrackball, 365, 366  
getScrollMatrix  
    GgApp::Window, 475  
getShader  
    Preference, 443  
getShiftArrow  
    GgApp::Window, 476  
getShiftArrowX  
    GgApp::Window, 476  
getShiftArrowY  
    GgApp::Window, 477  
getSize  
    Buffer, 100  
    Camera, 133  
    Capture, 146  
    Framebuffer, 170  
    gg::GgTexture, 358, 359  
    GgApp::Window, 477  
    Texture, 457  
getStart  
    gg::GgTrackball, 366, 367  
getStride  
    gg::GgBuffer< T >, 179  
    gg::GgUniformBuffer< T >, 379  
getString  
    parseconfig.h, 726, 727  
getTarget  
    gg::GgBuffer< T >, 179  
    gg::GgUniformBuffer< T >, 379  
getTexture  
    gg::GgTexture, 359  
getTextureName  
    Texture, 458  
getTitle  
    Config, 154  
getTranslation  
    GgApp::Window, 478  
getTranslationMatrix  
    GgApp::Window, 478  
getUserPointer  
    GgApp::Window, 479  
getValue  
    parseconfig.h, 727, 728  
getWheel

GgApp::Window, 479  
 getWheelX  
     GgApp::Window, 480  
 getWheelY  
     GgApp::Window, 480  
 getWidth  
     Buffer, 101  
     Camera, 133  
     Config, 155  
     gg::GgTexture, 360  
     GgApp::Window, 480  
 gg, 13  
     \_ggError, 16  
     \_ggFBOError, 17  
     BindingPoints, 16  
     ggArraysObj, 17  
     ggBufferAlignment, 87  
     ggConjugate, 18  
     ggCreateComputeShader, 18  
     ggCreateNormalMap, 19  
     ggCreateShader, 20  
     ggCross, 21, 22  
     ggDistance3, 22, 23  
     ggDistance4, 24  
     ggDot3, 26, 27  
     ggDot4, 28  
     ggElementsMesh, 29  
     ggElementsObj, 30  
     ggElementsSphere, 31  
     ggEllipse, 31  
     ggEulerQuaternion, 32  
     ggFrustum, 33  
     ggIdentity, 34  
     ggIdentityQuaternion, 35  
     ggInit, 35  
     ggInvert, 36, 37  
     ggLength3, 37, 38  
     ggLength4, 39  
     ggLoadComputeShader, 40  
     ggLoadHeight, 41  
     ggLoadImage, 42  
     ggLoadShader, 42, 43  
     ggLoadSimpleObj, 44, 45  
     ggLoadTexture, 45  
     ggLookat, 46–48  
     ggMatrixQuaternion, 49  
     ggNorm, 50  
     ggNormal, 51  
     ggNormalize, 51  
     ggNormalize3, 52–54  
     ggNormalize4, 54–56  
     ggOrthogonal, 57  
     ggPerspective, 58  
     ggPointsCube, 59  
     ggPointsSphere, 59  
     ggQuaternion, 60  
     ggQuaternionMatrix, 61  
     ggQuaternionTransposeMatrix, 62  
     ggReadImage, 62  
     ggRectangle, 64  
     ggRotate, 64–67  
     ggRotateQuaternion, 68, 69  
     ggRotateX, 70  
     ggRotateY, 71  
     ggRotateZ, 72  
     ggSaveColor, 72  
     ggSaveDepth, 73  
     ggSaveTga, 73  
     ggScale, 74–76  
     ggSlerp, 76–78  
     ggTranslate, 80, 81  
     ggTranspose, 82  
     LightBindingPoint, 16  
     MaterialBindingPoint, 16  
     operator\*, 84  
     operator+, 84, 85  
     operator-, 85, 86  
     operator/, 86  
     gg.cpp, 543  
     gg.h, 620  
         ggError, 624  
         ggFBOError, 624  
         pathChar, 625  
         pathString, 625  
         TCharToUtf8, 625  
         Utf8ToTChar, 625  
     gg::GgBuffer< T >, 177  
         ~GgBuffer, 178  
         bind, 178  
         copy, 178  
         getBuffer, 179  
         getCount, 179  
         getStride, 179  
         getTarget, 179  
         GgBuffer, 177, 178  
         map, 180  
         operator=, 181  
         read, 181  
         send, 181  
         unbind, 182  
         unmap, 182  
     gg::GgColorTexture, 182  
         ~GgColorTexture, 184  
         GgColorTexture, 183, 184  
         load, 185, 186  
     gg::GgElements, 187  
         ~GgElements, 189  
         draw, 189  
         getIndexBuffer, 190  
         getIndexCount, 190  
         GgElements, 188  
         load, 190  
         send, 191  
     gg::GgMatrix, 191  
         frustum, 196  
         get, 197, 198

GgMatrix, 194, 195  
invert, 199  
loadFrustum, 199  
loadIdentity, 200  
loadInvert, 200, 201  
loadLookat, 202, 203  
loadNormal, 204, 205  
loadOrthogonal, 205  
loadPerspective, 206  
loadRotate, 207–210  
loadRotateX, 210  
loadRotateY, 211  
loadRotateZ, 212  
loadScale, 212–214  
loadTranslate, 214–216  
loadTranspose, 216, 217  
lookat, 218, 219  
normal, 220  
operator $*$ , 221, 222  
operator $*=$ , 223, 224  
operator $+$ , 225  
operator $+=$ , 226, 227  
operator $-$ , 227, 228  
operator $-=$ , 228, 229  
operator $/$ , 230  
operator $/=$ , 231  
operator $=$ , 232  
orthogonal, 233  
perspective, 234  
projection, 234–236  
rotate, 236–238  
rotateX, 239  
rotateY, 240  
rotateZ, 241  
scale, 242, 243  
translate, 244–246  
transpose, 246  
gg::GgNormalTexture, 247  
~GgNormalTexture, 249  
GgNormalTexture, 248, 249  
load, 249, 250  
gg::GgPoints, 251  
~GgPoints, 253  
draw, 253  
getBuffer, 254  
getCount, 254  
GgPoints, 252  
load, 254  
operator bool, 255  
operator!, 255  
send, 255  
gg::GgPointShader, 256  
~GgPointShader, 258  
get, 258  
GgPointShader, 257  
load, 258, 259  
loadMatrix, 260  
loadModelviewMatrix, 261  
loadProjectionMatrix, 261, 262  
unuse, 262  
use, 262–264  
gg::GgQuaternion, 265  
add, 269–271  
conjugate, 272  
divide, 273–275  
euler, 275, 276  
get, 277  
getConjugateMatrix, 277–279  
getMatrix, 279–281  
GgQuaternion, 268, 269  
invert, 281  
load, 282, 283  
loadAdd, 284–286  
loadConjugate, 287  
loadDivide, 288–290  
loadEuler, 291, 292  
loadIdentity, 292  
loadInvert, 293, 294  
loadMatrix, 295  
loadMultiply, 296–298  
loadNormalize, 299  
loadRotate, 300, 301  
loadRotateX, 302  
loadRotateY, 303  
loadRotateZ, 303  
loadSlerp, 304–306  
loadSubtract, 307–309  
multiply, 309–311  
norm, 311  
normalize, 312  
operator $*$ , 312, 313  
operator $*=$ , 313, 314  
operator $+$ , 315  
operator $+=$ , 316, 317  
operator $-$ , 317, 318  
operator $-=$ , 318, 319  
operator $/$ , 320  
operator $/=$ , 321, 322  
operator $=$ , 322, 323  
rotate, 323, 324  
rotateX, 325  
rotateY, 326  
rotateZ, 327  
slerp, 327, 328  
subtract, 328–330  
gg::GgShader, 331  
~GgShader, 333  
get, 333  
GgShader, 332, 333  
operator $=$ , 333  
unuse, 333  
use, 334  
gg::GgShape, 334  
~GgShape, 335  
draw, 336  
get, 336

getMode, 337  
 GgShape, 335  
 operator bool, 337  
 operator!, 337  
 setMode, 338  
 gg::GgSimpleObj, 338  
 ~GgSimpleObj, 339  
 draw, 340  
 get, 340  
 GgSimpleObj, 339  
 operator bool, 340  
 operator!, 341  
 gg::GgSimpleShader, 341  
 ~GgSimpleShader, 344  
 GgSimpleShader, 343, 344  
 load, 344, 345  
 loadMatrix, 346, 347  
 loadModelviewMatrix, 348, 349  
 operator=, 349  
 use, 350–355  
 gg::GgSimpleShader::Light, 409  
 ambient, 409  
 diffuse, 410  
 position, 410  
 specular, 410  
 gg::GgSimpleShader::LightBuffer, 411  
 ~LightBuffer, 413  
 LightBuffer, 412  
 load, 413  
 loadAmbient, 414, 415  
 loadColor, 415  
 loadDiffuse, 415, 416  
 loadPosition, 417, 418  
 loadSpecular, 418, 419  
 select, 420  
 gg::GgSimpleShader::Material, 420  
 ambient, 421  
 diffuse, 421  
 shininess, 422  
 specular, 422  
 gg::GgSimpleShader::MaterialBuffer, 422  
 ~MaterialBuffer, 424  
 load, 424, 425  
 loadAmbient, 425, 426  
 loadAmbientAndDiffuse, 427  
 loadDiffuse, 428, 429  
 loadShininess, 429, 430  
 loadSpecular, 430, 431  
 MaterialBuffer, 423, 424  
 select, 431  
 gg::GgTexture, 356  
 ~GgTexture, 357  
 bind, 358  
 getHeight, 358  
 getSize, 358, 359  
 getTexture, 359  
 getWidth, 360  
 GgTexture, 357, 358  
 operator=, 360  
 swapRandB, 360  
 unbind, 361  
 gg::GgTrackball, 361  
 ~GgTrackball, 363  
 begin, 363  
 end, 364  
 get, 364  
 getMatrix, 365  
 getQuaternion, 365  
 getScale, 365, 366  
 getStart, 366, 367  
 GgTrackball, 363  
 motion, 367  
 operator=, 368  
 region, 368, 369  
 reset, 370  
 rotate, 370  
 gg::GgTriangles, 371  
 ~GgTriangles, 373  
 draw, 373  
 getBuffer, 374  
 getCount, 374  
 GgTriangles, 372  
 load, 374  
 send, 375  
 gg::GgUniformBuffer< T >, 375  
 ~GgUniformBuffer, 377  
 bind, 377  
 copy, 378  
 fill, 378  
 getBuffer, 379  
 getCount, 379  
 getStride, 379  
 getTarget, 379  
 GgUniformBuffer, 376, 377  
 load, 380  
 map, 381  
 read, 381  
 send, 382  
 unbind, 382  
 unmap, 382  
 gg::GgVector, 383  
 distance3, 386  
 distance4, 387  
 dot3, 388  
 dot4, 388  
 GgVector, 384, 386  
 length3, 389  
 length4, 389  
 normalize3, 390  
 normalize4, 390  
 operator\*, 391  
 operator\*=, 392  
 operator+, 393  
 operator+=, 393, 394  
 operator-, 394, 395  
 operator-=, 395

operator/, 396  
operator/=, 397  
gg::GgVertex, 397  
  GgVertex, 398–400  
  normal, 400  
  position, 400  
gg::GgVertexArray, 401  
  ~GgVertexArray, 401  
  bind, 402  
  get, 402  
  GgVertexArray, 401  
  operator=, 402  
GG\_BUTTON\_COUNT  
  GgApp.h, 695  
GG\_INTERFACE\_COUNT  
  GgApp.h, 695  
GG\_USE\_IMGUI  
  GgApp.h, 695  
GgApp, 174  
  ~GgApp, 175  
  GgApp, 175  
  main, 175  
  operator=, 176  
GgApp.cpp, 680  
GgApp.h, 694  
  GG\_BUTTON\_COUNT, 695  
  GG\_INTERFACE\_COUNT, 695  
  GG\_USE\_IMGUI, 695  
GgApp::Window, 462  
  ~Window, 464  
  get, 464  
  getAltArrowX, 464  
  getAltArrowY, 465  
  getAltArrow, 466  
  getArrow, 467  
  getArrowX, 468  
  getArrowY, 469  
  getAspect, 469  
  getControlArrow, 470  
  getControlArrowX, 470  
  getControlArrowY, 471  
  getFboHeight, 472  
  getFboSize, 472  
  getFboWidth, 472  
  getHeight, 472  
  getKey, 473  
  getLastKey, 473  
  getMouse, 473, 474  
  getMouseX, 474  
  getMouseY, 474  
  getRotation, 474  
  getRotationMatrix, 475  
  getScrollMatrix, 475  
  getShiftArrow, 476  
  getShiftArrowX, 476  
  getShiftArrowY, 477  
  getSize, 477  
  getTranslation, 478  
getTranslationMatrix, 478  
getUserPointer, 479  
getWheel, 479  
getWheelX, 480  
getWheelY, 480  
getWidth, 480  
operator bool, 481  
operator=, 481  
reset, 481  
resetRotation, 482  
resetTranslation, 482  
restoreViewport, 483  
selectInterface, 483  
setClose, 483  
setKeyboardFunc, 485  
setMouseFunc, 485  
setResizeFunc, 485  
setUserPointer, 486  
setVelocity, 486  
setWheelFunc, 486  
shouldClose, 487  
swapBuffers, 487  
updateViewport, 487  
Window, 463, 464  
ggArraysObj  
  gg, 17  
GgBuffer  
  gg::GgBuffer< T >, 177, 178  
ggBufferAlignment  
  gg, 87  
GgColorTexture  
  gg::GgColorTexture, 183, 184  
ggConjugate  
  gg, 18  
ggCreateComputeShader  
  gg, 18  
ggCreateNormalMap  
  gg, 19  
ggCreateShader  
  gg, 20  
ggCross  
  gg, 21, 22  
ggDistance3  
  gg, 22, 23  
ggDistance4  
  gg, 24  
ggDot3  
  gg, 26, 27  
ggDot4  
  gg, 28  
GgElements  
  gg::GgElements, 188  
ggElementsMesh  
  gg, 29  
ggElementsObj  
  gg, 30  
ggElementsSphere  
  gg, 31

ggEllipse  
     gg, 31  
 ggError  
     gg.h, 624  
 ggEulerQuaternion  
     gg, 32  
 ggFBOError  
     gg.h, 624  
 ggFrustum  
     gg, 33  
 ggIdentity  
     gg, 34  
 ggIdentityQuaternion  
     gg, 35  
 ggInit  
     gg, 35  
 ggInvert  
     gg, 36, 37  
 ggLength3  
     gg, 37, 38  
 ggLength4  
     gg, 39  
 ggLoadComputeShader  
     gg, 40  
 ggLoadHeight  
     gg, 41  
 ggLoadImage  
     gg, 42  
 ggLoadShader  
     gg, 42, 43  
 ggLoadSimpleObj  
     gg, 44, 45  
 ggLoadTexture  
     gg, 45  
 ggLookat  
     gg, 46–48  
 GgMatrix  
     gg::GgMatrix, 194, 195  
 ggMatrixQuaternion  
     gg, 49  
 ggNorm  
     gg, 50  
 ggNormal  
     gg, 51  
 ggNormalize  
     gg, 51  
 ggNormalize3  
     gg, 52–54  
 ggNormalize4  
     gg, 54–56  
 GgNormalTexture  
     gg::GgNormalTexture, 248, 249  
 ggOrthogonal  
     gg, 57  
 ggPerspective  
     gg, 58  
 GgPoints  
     gg::GgPoints, 252  
 ggPointsCube  
     gg, 59  
 GgPointShader  
     gg::GgPointShader, 257  
 ggPointsSphere  
     gg, 59  
 GgQuaternion  
     gg::GgQuaternion, 268, 269  
 ggQuaternion  
     gg, 60  
 ggQuaternionMatrix  
     gg, 61  
 ggQuaternionTransposeMatrix  
     gg, 62  
 ggReadImage  
     gg, 62  
 ggRectangle  
     gg, 64  
 ggRotate  
     gg, 64–67  
 ggRotateQuaternion  
     gg, 68, 69  
 ggRotateX  
     gg, 70  
 ggRotateY  
     gg, 71  
 ggRotateZ  
     gg, 72  
 ggSaveColor  
     gg, 72  
 ggSaveDepth  
     gg, 73  
 ggSaveTga  
     gg, 73  
 ggScale  
     gg, 74–76  
 GgShader  
     gg::GgShader, 332, 333  
 GgShape  
     gg::GgShape, 335  
 GgSimpleObj  
     gg::GgSimpleObj, 339  
 GgSimpleShader  
     gg::GgSimpleShader, 343, 344  
 ggSlerp  
     gg, 76–78  
 GgTexture  
     gg::GgTexture, 357, 358  
 GgTrackball  
     gg::GgTrackball, 363  
 ggTranslate  
     gg, 80, 81  
 ggTranspose  
     gg, 82  
 GgTriangles  
     gg::GgTriangles, 372  
 GgUniformBuffer  
     gg::GgUniformBuffer< T >, 376, 377

GgVector  
    gg::GgVector, 384, 386

GgVertex  
    gg::GgVertex, 398–400

GgVertexArray  
    gg::GgVertexArray, 401

HEADER\_STR  
    main.cpp, 708

imageFilter  
    Menu.cpp, 710

in  
    Camera, 136

increaseExposure  
    CamCv, 123  
    Camera, 133

increaseGain  
    CamCv, 123  
    Camera, 133

initialize  
    Config, 155

interval  
    Camera, 137

Intrinsics, 403  
    center, 407  
    fov, 407  
    fps, 408  
    Intrinsics, 403, 404  
    sensorSize, 408  
    setCenter, 405  
    setFov, 405, 406  
    setFps, 406  
    setSize, 407  
    size, 408

Intrinsics.cpp, 703

Intrinsics.h, 704

invert  
    gg::GgMatrix, 199  
    gg::GgQuaternion, 281

isOpened  
    Capture, 146

isOpen  
    CamImage, 140

isRunning  
    Camera, 134

jsonFilter  
    Menu.cpp, 710

length3  
    gg::GgVector, 389

length4  
    gg::GgVector, 389

LightBindingPoint  
    gg, 16

LightBuffer  
    gg::GgSimpleShader::LightBuffer, 412

load  
    CamImage, 140  
    Config, 155  
    gg::GgColorTexture, 185, 186  
    gg::GgElements, 190  
    gg::GgNormalTexture, 249, 250  
    gg::GgPoints, 254  
    gg::GgPointShader, 258, 259  
    gg::GgQuaternion, 282, 283  
    gg::GgSimpleShader, 344, 345  
    gg::GgSimpleShader::LightBuffer, 413  
    gg::GgSimpleShader::MaterialBuffer, 424, 425  
    gg::GgTriangles, 374  
    gg::GgUniformBuffer< T >, 380  
    loadAdd  
        gg::GgQuaternion, 284–286  
    loadAmbient  
        gg::GgSimpleShader::LightBuffer, 414, 415  
        gg::GgSimpleShader::MaterialBuffer, 425, 426  
    loadAmbientAndDiffuse  
        gg::GgSimpleShader::MaterialBuffer, 427  
    loadColor  
        gg::GgSimpleShader::LightBuffer, 415  
    loadConjugate  
        gg::GgQuaternion, 287  
    loadDiffuse  
        gg::GgSimpleShader::LightBuffer, 415, 416  
        gg::GgSimpleShader::MaterialBuffer, 428, 429  
    loadDivide  
        gg::GgQuaternion, 288–290  
    loadEuler  
        gg::GgQuaternion, 291, 292  
    loadFrustum  
        gg::GgMatrix, 199  
    loadIdentity  
        gg::GgMatrix, 200  
        gg::GgQuaternion, 292  
    loadInvert  
        gg::GgMatrix, 200, 201  
        gg::GgQuaternion, 293, 294  
    loadLookat  
        gg::GgMatrix, 202, 203  
    loadMatrix  
        gg::GgPointShader, 260  
        gg::GgQuaternion, 295  
        gg::GgSimpleShader, 346, 347  
    loadModelviewMatrix  
        gg::GgPointShader, 261  
        gg::GgSimpleShader, 348, 349  
    loadMultiply  
        gg::GgQuaternion, 296–298  
    loadNormal  
        gg::GgMatrix, 204, 205  
    loadNormalize  
        gg::GgQuaternion, 299  
    loadOrthogonal  
        gg::GgMatrix, 205  
    loadParameters  
        Calibration, 114

loadPerspective  
gg::GgMatrix, 206

loadPosition  
gg::GgSimpleShader::LightBuffer, 417, 418

loadProjectionMatrix  
gg::GgPointShader, 261, 262

loadRotate  
gg::GgMatrix, 207–210  
gg::GgQuaternion, 300, 301

loadRotateX  
gg::GgMatrix, 210  
gg::GgQuaternion, 302

loadRotateY  
gg::GgMatrix, 211  
gg::GgQuaternion, 303

loadRotateZ  
gg::GgMatrix, 212  
gg::GgQuaternion, 303

loadScale  
gg::GgMatrix, 212–214

loadShininess  
gg::GgSimpleShader::MaterialBuffer, 429, 430

loadSlerp  
gg::GgQuaternion, 304–306

loadSpecular  
gg::GgSimpleShader::LightBuffer, 418, 419  
gg::GgSimpleShader::MaterialBuffer, 430, 431

loadSubtract  
gg::GgQuaternion, 307–309

loadTranslate  
gg::GgMatrix, 214–216

loadTranspose  
gg::GgMatrix, 216, 217

lookat  
gg::GgMatrix, 218, 219

main  
GgApp, 175  
main.cpp, 708

main.cpp, 707  
HEADER\_STR, 708  
main, 708

map  
Buffer, 102  
gg::GgBuffer< T >, 180  
gg::GgUniformBuffer< T >, 381

MaterialBindingPoint  
gg, 16

MaterialBuffer  
gg::GgSimpleShader::MaterialBuffer, 423, 424

Menu, 432  
~Menu, 433  
Config, 157  
detectBoard, 438  
detectMarker, 438  
draw, 433  
getMenubarHeight, 434  
getPose, 434  
Menu, 432, 433

operator bool, 435

operator=, 435

saveImage, 435

setSize, 436

setup, 437

Menu.cpp, 709  
imageFilter, 710  
jsonFilter, 710  
movieFilter, 711

Menu.h, 720

Mesh, 438  
~Mesh, 439  
draw, 439  
Mesh, 439  
operator=, 440

Mesh.h, 722

motion  
gg::GgTrackball, 367

movieFilter  
Menu.cpp, 711

mtx  
Camera, 137

multiply  
gg::GgQuaternion, 309–311

norm  
gg::GgQuaternion, 311

normal  
gg::GgMatrix, 220  
gg::GgVertex, 400

normalize  
gg::GgQuaternion, 312

normalize3  
gg::GgVector, 390

normalize4  
gg::GgVector, 390

open  
CamCv, 124  
CamImage, 141

openDevice  
Capture, 146

openImage  
Capture, 147

openMovie  
Capture, 147

operator bool  
Capture, 148  
gg::GgPoints, 255  
gg::GgShape, 337  
gg::GgSimpleObj, 340  
GgApp::Window, 481  
Menu, 435

operator!  
gg::GgPoints, 255  
gg::GgShape, 337  
gg::GgSimpleObj, 341

operator\*  
gg, 84

gg::GgMatrix, 221, 222  
gg::GgQuaternion, 312, 313  
gg::GgVector, 391  
operator\*=  
    gg::GgMatrix, 223, 224  
    gg::GgQuaternion, 313, 314  
    gg::GgVector, 392  
operator+  
    gg, 84, 85  
    gg::GgMatrix, 225  
    gg::GgQuaternion, 315  
    gg::GgVector, 393  
operator+=  
    gg::GgMatrix, 226, 227  
    gg::GgQuaternion, 316, 317  
    gg::GgVector, 393, 394  
operator-  
    gg, 85, 86  
    gg::GgMatrix, 227, 228  
    gg::GgQuaternion, 317, 318  
    gg::GgVector, 394, 395  
operator-=  
    gg::GgMatrix, 228, 229  
    gg::GgQuaternion, 318, 319  
    gg::GgVector, 395  
operator/  
    gg, 86  
    gg::GgMatrix, 230  
    gg::GgQuaternion, 320  
    gg::GgVector, 396  
operator/=  
    gg::GgMatrix, 231  
    gg::GgQuaternion, 321, 322  
    gg::GgVector, 397  
operator=  
    Buffer, 102, 103  
    Calibration, 115  
    Camera, 134  
    Expand, 160  
    Framebuffer, 171  
    gg::GgBuffer< T >, 181  
    gg::GgMatrix, 232  
    gg::GgQuaternion, 322, 323  
    gg::GgShader, 333  
    gg::GgSimpleShader, 349  
    gg::GgTexture, 360  
    gg::GgTrackball, 368  
    gg::GgVertexArray, 402  
    GgApp, 176  
    GgApp::Window, 481  
    Menu, 435  
    Mesh, 440  
    Texture, 458, 459  
orthogonal  
    gg::GgMatrix, 233  
out  
    Camera, 137  
parseconfig.h, 724  
getString, 726, 727  
getValue, 727, 728  
setString, 729, 730  
setValue, 730, 732  
pathChar  
    gg.h, 625  
pathString  
    gg.h, 625  
perspective  
    gg::GgMatrix, 234  
pixels  
    Camera, 137  
position  
    gg::GgSimpleShader::Light, 410  
    gg::GgVertex, 400  
Preference, 440  
    ~Preference, 442  
    buildShader, 442  
    getDescription, 442  
    getIntrinsics, 442  
    getShader, 443  
    Preference, 441  
    setPreference, 443  
Preference.cpp, 735  
Preference.h, 737  
projection  
    gg::GgMatrix, 234–236  
read  
    gg::GgBuffer< T >, 181  
    gg::GgUniformBuffer< T >, 381  
README.md, 740  
readPixels  
    Texture, 459–461  
recordCorners  
    Calibration, 115  
region  
    gg::GgTrackball, 368, 369  
reset  
    gg::GgTrackball, 370  
    GgApp::Window, 481  
resetRotation  
    GgApp::Window, 482  
resetTranslation  
    GgApp::Window, 482  
resize  
    Buffer, 104  
restoreViewport  
    GgApp::Window, 483  
retrieve  
    Capture, 148  
rotate  
    gg::GgMatrix, 236–238  
    gg::GgQuaternion, 323, 324  
    gg::GgTrackball, 370  
rotateX  
    gg::GgMatrix, 239  
    gg::GgQuaternion, 325  
rotateY

gg::GgMatrix, 240  
 gg::GgQuaternion, 326  
 rotateZ  
   gg::GgMatrix, 241  
   gg::GgQuaternion, 327  
 running  
   Camera, 137  
  
 samples  
   Settings, 447  
 save  
   CamImage, 142  
   Config, 156  
 saveImage  
   Menu, 435  
 saveParameters  
   Calibration, 115  
 scale  
   gg::GgMatrix, 242, 243  
 select  
   gg::GgSimpleShader::LightBuffer, 420  
   gg::GgSimpleShader::MaterialBuffer, 431  
 selectInterface  
   GgApp::Window, 483  
 send  
   gg::GgBuffer< T >, 181  
   gg::GgElements, 191  
   gg::GgPoints, 255  
   gg::GgTriangles, 375  
   gg::GgUniformBuffer< T >, 382  
 sensorSize  
   Intrinsics, 408  
 setCenter  
   Intrinsics, 405  
 setClose  
   GgApp::Window, 483  
 setData  
   Buffer, 105  
 setDictionary  
   Calibration, 117  
 setExposure  
   CamCv, 125  
 setFov  
   Intrinsics, 405, 406  
 setFps  
   Intrinsics, 406  
 setGain  
   CamCv, 126  
 setKeyboardFunc  
   GgApp::Window, 485  
 setMode  
   gg::GgShape, 338  
 setMouseFunc  
   GgApp::Window, 485  
 setPosition  
   CamCv, 126  
 setPreference  
   Preference, 443  
 setResizeFunc  
  
   GgApp::Window, 485  
 setSize  
   Intrinsics, 407  
   Menu, 436  
 setString  
   parseconfig.h, 729, 730  
 Settings, 444  
   checkerLength, 446  
   defaultEuler, 446  
   defaultFocal, 446  
   defaultFocalRange, 446  
   dictionaryName, 446  
   euler, 446  
   focal, 447  
   focalRange, 447  
   getFocal, 445  
   samples, 447  
   Settings, 445  
 setup  
   Expand, 160  
   Menu, 437  
 setUserPointer  
   GgApp::Window, 486  
 setValue  
   parseconfig.h, 730, 732  
 setVelocity  
   GgApp::Window, 486  
 setWheelFunc  
   GgApp::Window, 486  
 shininess  
   gg::GgSimpleShader::Material, 422  
 shouldClose  
   GgApp::Window, 487  
 size  
   Intrinsics, 408  
 slerp  
   gg::GgQuaternion, 327, 328  
 specular  
   gg::GgSimpleShader::Light, 410  
   gg::GgSimpleShader::Material, 422  
 start  
   Camera, 134  
   Capture, 149  
 stop  
   Camera, 135  
   Capture, 149  
 subtract  
   gg::GgQuaternion, 328–330  
 swapBuffers  
   GgApp::Window, 487  
 swapRandB  
   gg::GgTexture, 360  
  
 TCharToUtf8  
   gg.h, 625  
 Texture, 448  
   ~Texture, 451  
   bindTexture, 451  
   copy, 452

create, 453  
discard, 454  
drawPixels, 454–456  
getChannels, 457  
getSize, 457  
getTextureName, 458  
operator=, 458, 459  
readPixels, 459–461  
Texture, 449, 450  
unbindTexture, 461  
Texture.cpp, 740  
Texture.h, 743  
thr  
    Camera, 138  
total  
    Camera, 138  
translate  
    gg::GgMatrix, 244–246  
transmit  
    Camera, 135  
transpose  
    gg::GgMatrix, 246  
  
unbind  
    gg::GgBuffer< T >, 182  
    gg::GgTexture, 361  
    gg::GgUniformBuffer< T >, 382  
unbindBuffer  
    Buffer, 105  
unbindFramebuffer  
    Framebuffer, 172  
unbindTexture  
    Texture, 461  
unmap  
    Buffer, 106  
    gg::GgBuffer< T >, 182  
    gg::GgUniformBuffer< T >, 382  
unuse  
    gg::GgPointShader, 262  
    gg::GgShader, 333  
update  
    Framebuffer, 172, 173  
updateViewport  
    GgApp::Window, 487  
use  
    gg::GgPointShader, 262–264  
    gg::GgShader, 334  
    gg::GgSimpleShader, 350–355  
USE\_PIXEL\_BUFFER\_OBJECT  
    Buffer.h, 494  
Utf8ToTChar  
    gg.h, 625  
  
Window  
    GgApp::Window, 463, 464