

ゲームグラフィックス特論

構築: Doxygen 1.12.0

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.	1
2 ggsample01	3
2.1 ゲーム グラフィックス特論A 第1回 宿題	3
2.2 宿題プログラムの作成に必要な環境	3
2.3 補足	3
2.4 宿題プログラム用補助プログラムについて	4
2.4.1 補助プログラムのドキュメント	4
2.4.2 補助プログラムの使い方	4
2.4.3 Oculus Rift を使う場合	5
2.4.4 Dear ImGui を使う場合	5
2.4.4.1 imconfig.h の変更点	6
3 名前空間索引	7
3.1 名前空間一覧	7
4 階層索引	9
4.1 クラス階層	9
5 クラス索引	11
5.1 クラス一覧	11
6 ファイル索引	13
6.1 ファイル一覧	13
7 名前空間詳解	15
7.1 gg 名前空間	15
7.1.1 詳解	18
7.1.2 列挙型詳解	18
7.1.2.1 BindingPoints	18
7.1.3 関数詳解	18
7.1.3.1 ggError()	18
7.1.3.2 ggFBOError()	19
7.1.3.3 ggArraysObj()	19
7.1.3.4 ggConjugate()	20
7.1.3.5 ggCreateComputeShader()	20
7.1.3.6 ggCreateNormalMap()	21
7.1.3.7 ggCreateShader()	22
7.1.3.8 ggCross() [1/2]	22
7.1.3.9 ggCross() [2/2]	23
7.1.3.10 ggDistance3() [1/2]	24
7.1.3.11 ggDistance3() [2/2]	24
7.1.3.12 ggDistance4() [1/2]	25
7.1.3.13 ggDistance4() [2/2]	26

7.1.3.14 ggDot3() [1/2]	27
7.1.3.15 ggDot3() [2/2]	27
7.1.3.16 ggDot4() [1/2]	28
7.1.3.17 ggDot4() [2/2]	29
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]	32
7.1.3.24 ggFrustum()	33
7.1.3.25 ggIdentity()	34
7.1.3.26 ggIdentityQuaternion()	35
7.1.3.27 ggInit()	35
7.1.3.28 ggInvert() [1/2]	35
7.1.3.29 ggInvert() [2/2]	36
7.1.3.30 ggLength3() [1/2]	36
7.1.3.31 ggLength3() [2/2]	37
7.1.3.32 ggLength4() [1/2]	38
7.1.3.33 ggLength4() [2/2]	38
7.1.3.34 ggLoadComputeShader()	39
7.1.3.35 ggLoadHeight()	40
7.1.3.36 ggLoadImage()	41
7.1.3.37 ggLoadShader() [1/2]	41
7.1.3.38 ggLoadShader() [2/2]	42
7.1.3.39 ggLoadSimpleObj() [1/2]	43
7.1.3.40 ggLoadSimpleObj() [2/2]	44
7.1.3.41 ggLoadTexture()	45
7.1.3.42 ggLookat() [1/3]	45
7.1.3.43 ggLookat() [2/3]	46
7.1.3.44 ggLookat() [3/3]	47
7.1.3.45 ggMatrixQuaternion() [1/2]	48
7.1.3.46 ggMatrixQuaternion() [2/2]	48
7.1.3.47 ggNorm()	49
7.1.3.48 ggNormal()	50
7.1.3.49 ggNormalize()	50
7.1.3.50 ggNormalize3() [1/4]	50
7.1.3.51 ggNormalize3() [2/4]	51
7.1.3.52 ggNormalize3() [3/4]	52
7.1.3.53 ggNormalize3() [4/4]	52
7.1.3.54 ggNormalize4() [1/4]	53
7.1.3.55 ggNormalize4() [2/4]	53

7.1.3.56 ggNormalize4() [3/4]	54
7.1.3.57 ggNormalize4() [4/4]	55
7.1.3.58 ggOrthogonal()	55
7.1.3.59 ggPerspective()	56
7.1.3.60 ggPointsCube()	57
7.1.3.61 ggPointsSphere()	57
7.1.3.62 ggQuaternionMatrix()	58
7.1.3.63 ggQuaternionTransposeMatrix()	58
7.1.3.64 ggReadImage()	59
7.1.3.65 ggRectangle()	60
7.1.3.66 ggRotate() [1/5]	61
7.1.3.67 ggRotate() [2/5]	61
7.1.3.68 ggRotate() [3/5]	62
7.1.3.69 ggRotate() [4/5]	62
7.1.3.70 ggRotate() [5/5]	63
7.1.3.71 ggRotateQuaternion() [1/3]	64
7.1.3.72 ggRotateQuaternion() [2/3]	64
7.1.3.73 ggRotateQuaternion() [3/3]	65
7.1.3.74 ggRotateX()	66
7.1.3.75 ggRotateY()	66
7.1.3.76 ggRotateZ()	67
7.1.3.77 ggSaveColor()	68
7.1.3.78 ggSaveDepth()	69
7.1.3.79 ggSaveTga()	70
7.1.3.80 ggScale() [1/3]	71
7.1.3.81 ggScale() [2/3]	71
7.1.3.82 ggScale() [3/3]	72
7.1.3.83 ggSlerp() [1/4]	72
7.1.3.84 ggSlerp() [2/4]	73
7.1.3.85 ggSlerp() [3/4]	73
7.1.3.86 ggSlerp() [4/4]	74
7.1.3.87 ggTranslate() [1/3]	75
7.1.3.88 ggTranslate() [2/3]	76
7.1.3.89 ggTranslate() [3/3]	76
7.1.3.90 ggTranspose()	77
7.1.3.91 operator*()	78
7.1.3.92 operator+() [1/2]	78
7.1.3.93 operator+() [2/2]	78
7.1.3.94 operator-() [1/2]	79
7.1.3.95 operator-() [2/2]	79
7.1.3.96 operator/()	79
7.1.4 変数詳解	80

7.1.4.1 ggBufferAlignment	80
8 クラス詳解	81
8.1 Config クラス	81
8.1.1 詳解	81
8.1.2 構築子と解体子	81
8.1.2.1 Config() [1/2]	81
8.1.2.2 Config() [2/2]	82
8.1.2.3 ~Config()	82
8.1.3 関数詳解	82
8.1.3.1 getHeight()	82
8.1.3.2 getWidth()	83
8.1.3.3 load()	83
8.1.3.4 save()	84
8.1.4 フレンドと関連関数の詳解	84
8.1.4.1 Menu	84
8.2 Draw クラス	85
8.2.1 詳解	85
8.2.2 構築子と解体子	85
8.2.2.1 Draw()	85
8.2.2.2 ~Draw()	85
8.2.3 関数詳解	86
8.2.3.1 draw()	86
8.3 GgApp クラス	86
8.3.1 詳解	87
8.3.2 構築子と解体子	87
8.3.2.1 GgApp() [1/3]	87
8.3.2.2 GgApp() [2/3]	87
8.3.2.3 GgApp() [3/3]	87
8.3.2.4 ~GgApp()	87
8.3.3 関数詳解	88
8.3.3.1 main()	88
8.3.3.2 operator=() [1/2]	88
8.3.3.3 operator=() [2/2]	88
8.4 gg::GgBuffer< T > クラステンプレート	89
8.4.1 詳解	89
8.4.2 構築子と解体子	89
8.4.2.1 GgBuffer() [1/3]	89
8.4.2.2 GgBuffer() [2/3]	90
8.4.2.3 GgBuffer() [3/3]	90
8.4.2.4 ~GgBuffer()	90
8.4.3 関数詳解	90

8.4.3.1 bind()	90
8.4.3.2 copy()	90
8.4.3.3 getBuffer()	91
8.4.3.4 getCount()	91
8.4.3.5 getStride()	91
8.4.3.6 getTarget()	92
8.4.3.7 map() [1/2]	92
8.4.3.8 map() [2/2]	92
8.4.3.9 operator=() [1/2]	93
8.4.3.10 operator=() [2/2]	93
8.4.3.11 read()	93
8.4.3.12 send()	93
8.4.3.13 unbind()	94
8.4.3.14 unmap()	94
8.5 gg::GgColorTexture クラス	94
8.5.1 詳解	94
8.5.2 構築子と解体子	95
8.5.2.1 GgColorTexture() [1/3]	95
8.5.2.2 GgColorTexture() [2/3]	95
8.5.2.3 GgColorTexture() [3/3]	96
8.5.2.4 ~GgColorTexture()	96
8.5.3 関数詳解	96
8.5.3.1 load() [1/2]	96
8.5.3.2 load() [2/2]	97
8.6 gg::GgElements クラス	98
8.6.1 詳解	100
8.6.2 構築子と解体子	100
8.6.2.1 GgElements() [1/2]	100
8.6.2.2 GgElements() [2/2]	100
8.6.2.3 ~GgElements()	100
8.6.3 関数詳解	101
8.6.3.1 draw()	101
8.6.3.2 getIndexBuffer()	101
8.6.3.3 getIndexCount()	101
8.6.3.4 load()	102
8.6.3.5 send()	102
8.7 gg::GgMatrix クラス	103
8.7.1 詳解	105
8.7.2 構築子と解体子	105
8.7.2.1 GgMatrix() [1/6]	105
8.7.2.2 GgMatrix() [2/6]	106
8.7.2.3 GgMatrix() [3/6]	106

8.7.2.4 GgMatrix() [4/6]	107
8.7.2.5 GgMatrix() [5/6]	107
8.7.2.6 GgMatrix() [6/6]	107
8.7.2.7 ~GgMatrix()	107
8.7.3 関数詳解	107
8.7.3.1 frustum()	107
8.7.3.2 get() [1/2]	108
8.7.3.3 get() [2/2]	109
8.7.3.4 invert()	110
8.7.3.5 loadFrustum()	110
8.7.3.6 loadIdentity()	111
8.7.3.7 loadInvert() [1/2]	111
8.7.3.8 loadInvert() [2/2]	112
8.7.3.9 loadLookat() [1/3]	113
8.7.3.10 loadLookat() [2/3]	113
8.7.3.11 loadLookat() [3/3]	114
8.7.3.12 loadNormal() [1/2]	115
8.7.3.13 loadNormal() [2/2]	116
8.7.3.14 loadOrthogonal()	116
8.7.3.15 loadPerspective()	117
8.7.3.16 loadRotate() [1/5]	118
8.7.3.17 loadRotate() [2/5]	118
8.7.3.18 loadRotate() [3/5]	119
8.7.3.19 loadRotate() [4/5]	119
8.7.3.20 loadRotate() [5/5]	120
8.7.3.21 loadRotateX()	121
8.7.3.22 loadRotateY()	122
8.7.3.23 loadRotateZ()	122
8.7.3.24 loadScale() [1/3]	123
8.7.3.25 loadScale() [2/3]	124
8.7.3.26 loadScale() [3/3]	124
8.7.3.27 loadTranslate() [1/3]	125
8.7.3.28 loadTranslate() [2/3]	126
8.7.3.29 loadTranslate() [3/3]	126
8.7.3.30 loadTranspose() [1/2]	127
8.7.3.31 loadTranspose() [2/2]	128
8.7.3.32 lookat() [1/3]	128
8.7.3.33 lookat() [2/3]	129
8.7.3.34 lookat() [3/3]	130
8.7.3.35 normal()	131
8.7.3.36 operator*() [1/3]	132
8.7.3.37 operator*() [2/3]	133

8.7.3.38 operator*() [3/3]	134
8.7.3.39 operator*=(()) [1/2]	135
8.7.3.40 operator*=(()) [2/2]	136
8.7.3.41 operator+() [1/2]	136
8.7.3.42 operator+() [2/2]	137
8.7.3.43 operator+=() [1/2]	137
8.7.3.44 operator+=() [2/2]	138
8.7.3.45 operator-() [1/2]	138
8.7.3.46 operator-() [2/2]	139
8.7.3.47 operator-=(()) [1/2]	140
8.7.3.48 operator-=(()) [2/2]	141
8.7.3.49 operator/() [1/2]	141
8.7.3.50 operator/() [2/2]	142
8.7.3.51 operator/=(()) [1/2]	143
8.7.3.52 operator/=(()) [2/2]	143
8.7.3.53 operator=() [1/3]	144
8.7.3.54 operator=() [2/3]	144
8.7.3.55 operator=() [3/3]	145
8.7.3.56 orthogonal()	145
8.7.3.57 perspective()	146
8.7.3.58 projection() [1/4]	147
8.7.3.59 projection() [2/4]	147
8.7.3.60 projection() [3/4]	147
8.7.3.61 projection() [4/4]	147
8.7.3.62 rotate() [1/5]	148
8.7.3.63 rotate() [2/5]	148
8.7.3.64 rotate() [3/5]	149
8.7.3.65 rotate() [4/5]	149
8.7.3.66 rotate() [5/5]	150
8.7.3.67 rotateX()	151
8.7.3.68 rotateY()	152
8.7.3.69 rotateZ()	153
8.7.3.70 scale() [1/3]	154
8.7.3.71 scale() [2/3]	155
8.7.3.72 scale() [3/3]	155
8.7.3.73 translate() [1/3]	156
8.7.3.74 translate() [2/3]	157
8.7.3.75 translate() [3/3]	157
8.7.3.76 transpose()	158
8.8 gg::GgNormalTexture クラス	159
8.8.1 詳解	160
8.8.2 構築子と解体子	160

8.8.2.1 GgNormalTexture() [1/3]	160
8.8.2.2 GgNormalTexture() [2/3]	160
8.8.2.3 GgNormalTexture() [3/3]	161
8.8.2.4 ~GgNormalTexture()	161
8.8.3 関数詳解	161
8.8.3.1 load() [1/2]	161
8.8.3.2 load() [2/2]	162
8.9 gg::GgPoints クラス	163
8.9.1 詳解	164
8.9.2 構築子と解体子	164
8.9.2.1 GgPoints() [1/2]	164
8.9.2.2 GgPoints() [2/2]	164
8.9.2.3 ~GgPoints()	165
8.9.3 関数詳解	165
8.9.3.1 draw()	165
8.9.3.2 getBuffer()	166
8.9.3.3 getCount()	166
8.9.3.4 load()	166
8.9.3.5 operator bool()	166
8.9.3.6 operator"!"()	167
8.9.3.7 send()	167
8.10 gg::GgPointShader クラス	167
8.10.1 詳解	168
8.10.2 構築子と解体子	168
8.10.2.1 GgPointShader() [1/3]	168
8.10.2.2 GgPointShader() [2/3]	168
8.10.2.3 GgPointShader() [3/3]	169
8.10.2.4 ~GgPointShader()	169
8.10.3 関数詳解	169
8.10.3.1 get()	169
8.10.3.2 load() [1/2]	170
8.10.3.3 load() [2/2]	171
8.10.3.4 loadMatrix() [1/2]	172
8.10.3.5 loadMatrix() [2/2]	172
8.10.3.6 loadModelviewMatrix() [1/2]	172
8.10.3.7 loadModelviewMatrix() [2/2]	173
8.10.3.8 loadProjectionMatrix() [1/2]	173
8.10.3.9 loadProjectionMatrix() [2/2]	174
8.10.3.10 unuse()	174
8.10.3.11 use() [1/5]	174
8.10.3.12 use() [2/5]	174
8.10.3.13 use() [3/5]	175

8.10.3.14 use() [4/5]	175
8.10.3.15 use() [5/5]	176
8.11 gg::GgQuaternion クラス	176
8.11.1 詳解	180
8.11.2 構築子と解体子	180
8.11.2.1 GgQuaternion() [1/7]	180
8.11.2.2 GgQuaternion() [2/7]	180
8.11.2.3 GgQuaternion() [3/7]	181
8.11.2.4 GgQuaternion() [4/7]	182
8.11.2.5 GgQuaternion() [5/7]	182
8.11.2.6 GgQuaternion() [6/7]	182
8.11.2.7 GgQuaternion() [7/7]	182
8.11.2.8 ~GgQuaternion()	183
8.11.3 関数詳解	183
8.11.3.1 add() [1/4]	183
8.11.3.2 add() [2/4]	183
8.11.3.3 add() [3/4]	184
8.11.3.4 add() [4/4]	184
8.11.3.5 conjugate()	185
8.11.3.6 divide() [1/4]	186
8.11.3.7 divide() [2/4]	187
8.11.3.8 divide() [3/4]	187
8.11.3.9 divide() [4/4]	188
8.11.3.10 euler() [1/2]	189
8.11.3.11 euler() [2/2]	189
8.11.3.12 get()	190
8.11.3.13 getConjugateMatrix() [1/3]	191
8.11.3.14 getConjugateMatrix() [2/3]	191
8.11.3.15 getConjugateMatrix() [3/3]	192
8.11.3.16 getMatrix() [1/3]	192
8.11.3.17 getMatrix() [2/3]	193
8.11.3.18 getMatrix() [3/3]	194
8.11.3.19 invert()	194
8.11.3.20 loadAdd() [1/4]	195
8.11.3.21 loadAdd() [2/4]	196
8.11.3.22 loadAdd() [3/4]	196
8.11.3.23 loadAdd() [4/4]	197
8.11.3.24 loadConjugate() [1/2]	197
8.11.3.25 loadConjugate() [2/2]	198
8.11.3.26 loadDivide() [1/4]	199
8.11.3.27 loadDivide() [2/4]	200
8.11.3.28 loadDivide() [3/4]	201

8.11.3.29 loadDivide() [4/4]	202
8.11.3.30 loadEuler() [1/2]	203
8.11.3.31 loadEuler() [2/2]	204
8.11.3.32 loadIdentity()	205
8.11.3.33 loadInvert() [1/2]	205
8.11.3.34 loadInvert() [2/2]	206
8.11.3.35 loadMatrix() [1/2]	207
8.11.3.36 loadMatrix() [2/2]	207
8.11.3.37 loadMultiply() [1/4]	208
8.11.3.38 loadMultiply() [2/4]	208
8.11.3.39 loadMultiply() [3/4]	209
8.11.3.40 loadMultiply() [4/4]	209
8.11.3.41 loadNormalize() [1/2]	210
8.11.3.42 loadNormalize() [2/2]	211
8.11.3.43 loadRotate() [1/3]	212
8.11.3.44 loadRotate() [2/3]	212
8.11.3.45 loadRotate() [3/3]	213
8.11.3.46 loadRotateX()	213
8.11.3.47 loadRotateY()	214
8.11.3.48 loadRotateZ()	214
8.11.3.49 loadSlerp() [1/4]	215
8.11.3.50 loadSlerp() [2/4]	216
8.11.3.51 loadSlerp() [3/4]	216
8.11.3.52 loadSlerp() [4/4]	217
8.11.3.53 loadSubtract() [1/4]	218
8.11.3.54 loadSubtract() [2/4]	218
8.11.3.55 loadSubtract() [3/4]	219
8.11.3.56 loadSubtract() [4/4]	219
8.11.3.57 multiply() [1/4]	220
8.11.3.58 multiply() [2/4]	220
8.11.3.59 multiply() [3/4]	221
8.11.3.60 multiply() [4/4]	221
8.11.3.61 norm()	222
8.11.3.62 normalize()	222
8.11.3.63 operator*() [1/3]	223
8.11.3.64 operator*() [2/3]	223
8.11.3.65 operator*() [3/3]	224
8.11.3.66 operator*=() [1/3]	224
8.11.3.67 operator*=() [2/3]	224
8.11.3.68 operator*=() [3/3]	225
8.11.3.69 operator+() [1/3]	225
8.11.3.70 operator+() [2/3]	226

8.11.3.71 operator+() [3/3]	226
8.11.3.72 operator+=() [1/3]	227
8.11.3.73 operator+=() [2/3]	227
8.11.3.74 operator+=() [3/3]	227
8.11.3.75 operator-() [1/3]	228
8.11.3.76 operator-() [2/3]	228
8.11.3.77 operator-() [3/3]	228
8.11.3.78 operator-=() [1/3]	229
8.11.3.79 operator-=() [2/3]	229
8.11.3.80 operator-=() [3/3]	229
8.11.3.81 operator/() [1/3]	230
8.11.3.82 operator/() [2/3]	230
8.11.3.83 operator/() [3/3]	231
8.11.3.84 operator/=() [1/3]	231
8.11.3.85 operator/=() [2/3]	232
8.11.3.86 operator/=() [3/3]	232
8.11.3.87 operator=() [1/4]	233
8.11.3.88 operator=() [2/4]	233
8.11.3.89 operator=() [3/4]	233
8.11.3.90 operator=() [4/4]	234
8.11.3.91 rotate() [1/3]	234
8.11.3.92 rotate() [2/3]	234
8.11.3.93 rotate() [3/3]	235
8.11.3.94 rotateX()	236
8.11.3.95 rotateY()	236
8.11.3.96 rotateZ()	237
8.11.3.97 slerp() [1/2]	237
8.11.3.98 slerp() [2/2]	238
8.11.3.99 subtract() [1/4]	238
8.11.3.100 subtract() [2/4]	239
8.11.3.101 subtract() [3/4]	239
8.11.3.102 subtract() [4/4]	240
8.12 gg::GgShader クラス	241
8.12.1 詳解	241
8.12.2 構築子と解体子	241
8.12.2.1 GgShader() [1/4]	241
8.12.2.2 GgShader() [2/4]	242
8.12.2.3 GgShader() [3/4]	242
8.12.2.4 GgShader() [4/4]	242
8.12.2.5 ~GgShader()	243
8.12.3 関数詳解	243
8.12.3.1 get()	243

8.12.3.2 operator=() [1/2]	243
8.12.3.3 operator=() [2/2]	243
8.12.3.4 unuse()	243
8.12.3.5 use()	244
8.13 gg::GgShape クラス	244
8.13.1 詳解	245
8.13.2 構築子と解体子	245
8.13.2.1 GgShape()	245
8.13.2.2 ~GgShape()	245
8.13.3 関数詳解	245
8.13.3.1 draw()	245
8.13.3.2 get()	246
8.13.3.3 getMode()	247
8.13.3.4 operator bool()	247
8.13.3.5 operator"!"()	247
8.13.3.6 setMode()	247
8.14 gg::GgSimpleObj クラス	248
8.14.1 詳解	248
8.14.2 構築子と解体子	248
8.14.2.1 GgSimpleObj()	248
8.14.2.2 ~GgSimpleObj()	249
8.14.3 関数詳解	249
8.14.3.1 draw()	249
8.14.3.2 get()	250
8.14.3.3 operator bool()	250
8.14.3.4 operator"!"()	250
8.15 gg::GgSimpleShader クラス	251
8.15.1 詳解	252
8.15.2 構築子と解体子	253
8.15.2.1 GgSimpleShader() [1/4]	253
8.15.2.2 GgSimpleShader() [2/4]	253
8.15.2.3 GgSimpleShader() [3/4]	253
8.15.2.4 GgSimpleShader() [4/4]	254
8.15.2.5 ~GgSimpleShader()	254
8.15.3 関数詳解	254
8.15.3.1 load() [1/2]	254
8.15.3.2 load() [2/2]	254
8.15.3.3 loadMatrix() [1/4]	255
8.15.3.4 loadMatrix() [2/4]	256
8.15.3.5 loadMatrix() [3/4]	256
8.15.3.6 loadMatrix() [4/4]	256
8.15.3.7 loadModelviewMatrix() [1/4]	257

8.15.3.8 loadModelviewMatrix() [2/4]	257
8.15.3.9 loadModelviewMatrix() [3/4]	258
8.15.3.10 loadModelviewMatrix() [4/4]	258
8.15.3.11 operator=()	259
8.15.3.12 use() [1/13]	259
8.15.3.13 use() [2/13]	259
8.15.3.14 use() [3/13]	260
8.15.3.15 use() [4/13]	260
8.15.3.16 use() [5/13]	261
8.15.3.17 use() [6/13]	261
8.15.3.18 use() [7/13]	262
8.15.3.19 use() [8/13]	262
8.15.3.20 use() [9/13]	263
8.15.3.21 use() [10/13]	263
8.15.3.22 use() [11/13]	264
8.15.3.23 use() [12/13]	264
8.15.3.24 use() [13/13]	264
8.16 gg::GgTexture クラス	265
8.16.1 詳解	266
8.16.2 構築子と解体子	266
8.16.2.1 GgTexture() [1/3]	266
8.16.2.2 GgTexture() [2/3]	266
8.16.2.3 GgTexture() [3/3]	267
8.16.2.4 ~GgTexture()	267
8.16.3 関数詳解	267
8.16.3.1 bind()	267
8.16.3.2 getHeight()	267
8.16.3.3 getSize() [1/2]	268
8.16.3.4 getSize() [2/2]	268
8.16.3.5 getTexture()	268
8.16.3.6 getWidth()	269
8.16.3.7 operator=() [1/2]	269
8.16.3.8 operator=() [2/2]	269
8.16.3.9 swapRandB()	269
8.16.3.10 unbind()	270
8.17 gg::GgTrackball クラス	270
8.17.1 詳解	274
8.17.2 構築子と解体子	274
8.17.2.1 GgTrackball()	274
8.17.2.2 ~GgTrackball()	275
8.17.3 関数詳解	275
8.17.3.1 begin()	275

8.17.3.2 end()	275
8.17.3.3 get()	276
8.17.3.4 getMatrix()	276
8.17.3.5 getQuaternion()	277
8.17.3.6 getScale() [1/3]	277
8.17.3.7 getScale() [2/3]	277
8.17.3.8 getScale() [3/3]	277
8.17.3.9 getStart() [1/3]	278
8.17.3.10 getStart() [2/3]	278
8.17.3.11 getStart() [3/3]	278
8.17.3.12 motion()	278
8.17.3.13 operator=()	279
8.17.3.14 region() [1/2]	279
8.17.3.15 region() [2/2]	280
8.17.3.16 reset()	281
8.17.3.17 rotate()	281
8.18 gg::GgTriangles クラス	282
8.18.1 詳解	283
8.18.2 構築子と解体子	283
8.18.2.1 GgTriangles() [1/2]	283
8.18.2.2 GgTriangles() [2/2]	284
8.18.2.3 ~GgTriangles()	284
8.18.3 関数詳解	284
8.18.3.1 draw()	284
8.18.3.2 getBuffer()	285
8.18.3.3 getCount()	285
8.18.3.4 load()	285
8.18.3.5 send()	286
8.19 gg::GgUniformBuffer< T > クラステンプレート	286
8.19.1 詳解	287
8.19.2 構築子と解体子	287
8.19.2.1 GgUniformBuffer() [1/3]	287
8.19.2.2 GgUniformBuffer() [2/3]	287
8.19.2.3 GgUniformBuffer() [3/3]	287
8.19.2.4 ~GgUniformBuffer()	288
8.19.3 関数詳解	288
8.19.3.1 bind()	288
8.19.3.2 copy()	288
8.19.3.3 fill()	289
8.19.3.4 getBuffer()	289
8.19.3.5 getCount()	289
8.19.3.6 getStride()	290

8.19.3.7 <code>getTarget()</code>	290
8.19.3.8 <code>load() [1/2]</code>	290
8.19.3.9 <code>load() [2/2]</code>	290
8.19.3.10 <code>map() [1/2]</code>	291
8.19.3.11 <code>map() [2/2]</code>	291
8.19.3.12 <code>read()</code>	291
8.19.3.13 <code>send()</code>	292
8.19.3.14 <code>unbind()</code>	292
8.19.3.15 <code>unmap()</code>	293
8.20 <code>gg::GgVector</code> クラス	293
8.20.1 詳解	294
8.20.2 構築子と解体子	294
8.20.2.1 <code>GgVector() [1/6]</code>	294
8.20.2.2 <code>GgVector() [2/6]</code>	295
8.20.2.3 <code>GgVector() [3/6]</code>	295
8.20.2.4 <code>GgVector() [4/6]</code>	296
8.20.2.5 <code>GgVector() [5/6]</code>	296
8.20.2.6 <code>GgVector() [6/6]</code>	296
8.20.2.7 <code>~GgVector()</code>	296
8.20.3 関数詳解	296
8.20.3.1 <code>distance3()</code>	296
8.20.3.2 <code>distance4()</code>	297
8.20.3.3 <code>dot3()</code>	297
8.20.3.4 <code>dot4()</code>	298
8.20.3.5 <code>length3()</code>	299
8.20.3.6 <code>length4()</code>	299
8.20.3.7 <code>normalize3()</code>	300
8.20.3.8 <code>normalize4()</code>	300
8.20.3.9 <code>operator*() [1/2]</code>	300
8.20.3.10 <code>operator*() [2/2]</code>	301
8.20.3.11 <code>operator*=(1/2)</code>	301
8.20.3.12 <code>operator*=(2/2)</code>	302
8.20.3.13 <code>operator+() [1/2]</code>	302
8.20.3.14 <code>operator+() [2/2]</code>	302
8.20.3.15 <code>operator+=() [1/2]</code>	303
8.20.3.16 <code>operator+=() [2/2]</code>	303
8.20.3.17 <code>operator-() [1/2]</code>	304
8.20.3.18 <code>operator-() [2/2]</code>	304
8.20.3.19 <code>operator-=(1/2)</code>	305
8.20.3.20 <code>operator-=(2/2)</code>	305
8.20.3.21 <code>operator/() [1/2]</code>	305
8.20.3.22 <code>operator/() [2/2]</code>	305

8.20.3.23 operator/() [1/2]	306
8.20.3.24 operator/() [2/2]	306
8.20.3.25 operator=() [1/2]	307
8.20.3.26 operator=() [2/2]	307
8.21 gg::GgVertex 構造体	307
8.21.1 詳解	308
8.21.2 構築子と解体子	308
8.21.2.1 GgVertex() [1/4]	308
8.21.2.2 GgVertex() [2/4]	308
8.21.2.3 GgVertex() [3/4]	308
8.21.2.4 GgVertex() [4/4]	309
8.21.3 メンバ詳解	309
8.21.3.1 normal	309
8.21.3.2 position	309
8.22 gg::GgVertexArray クラス	310
8.22.1 詳解	310
8.22.2 構築子と解体子	310
8.22.2.1 GgVertexArray() [1/3]	310
8.22.2.2 GgVertexArray() [2/3]	310
8.22.2.3 GgVertexArray() [3/3]	311
8.22.2.4 ~GgVertexArray()	311
8.22.3 関数詳解	311
8.22.3.1 bind()	311
8.22.3.2 get()	311
8.22.3.3 operator=() [1/2]	311
8.22.3.4 operator=() [2/2]	312
8.23 gg::GgSimpleShader::Light 構造体	312
8.23.1 詳解	313
8.23.2 メンバ詳解	313
8.23.2.1 ambient	313
8.23.2.2 diffuse	313
8.23.2.3 position	313
8.23.2.4 specular	313
8.24 gg::GgSimpleShader::LightBuffer クラス	314
8.24.1 詳解	315
8.24.2 構築子と解体子	315
8.24.2.1 LightBuffer() [1/2]	315
8.24.2.2 LightBuffer() [2/2]	316
8.24.2.3 ~LightBuffer()	316
8.24.3 関数詳解	316
8.24.3.1 load() [1/2]	316
8.24.3.2 load() [2/2]	317

8.24.3.3 loadAmbient() [1/3]	317
8.24.3.4 loadAmbient() [2/3]	317
8.24.3.5 loadAmbient() [3/3]	318
8.24.3.6 loadColor()	318
8.24.3.7 loadDiffuse() [1/3]	319
8.24.3.8 loadDiffuse() [2/3]	319
8.24.3.9 loadDiffuse() [3/3]	319
8.24.3.10 loadPosition() [1/4]	320
8.24.3.11 loadPosition() [2/4]	320
8.24.3.12 loadPosition() [3/4]	320
8.24.3.13 loadPosition() [4/4]	321
8.24.3.14 loadSpecular() [1/3]	321
8.24.3.15 loadSpecular() [2/3]	322
8.24.3.16 loadSpecular() [3/3]	322
8.24.3.17 select()	322
8.25 gg::GgSimpleShader::Material 構造体	323
8.25.1 詳解	324
8.25.2 メンバ詳解	324
8.25.2.1 ambient	324
8.25.2.2 diffuse	324
8.25.2.3 shininess	324
8.25.2.4 specular	325
8.26 gg::GgSimpleShader::MaterialBuffer クラス	325
8.26.1 詳解	326
8.26.2 構築子と解体子	327
8.26.2.1 MaterialBuffer() [1/2]	327
8.26.2.2 MaterialBuffer() [2/2]	327
8.26.2.3 ~MaterialBuffer()	327
8.26.3 関数詳解	327
8.26.3.1 load() [1/2]	327
8.26.3.2 load() [2/2]	328
8.26.3.3 loadAmbient() [1/3]	328
8.26.3.4 loadAmbient() [2/3]	328
8.26.3.5 loadAmbient() [3/3]	329
8.26.3.6 loadAmbientAndDiffuse() [1/3]	329
8.26.3.7 loadAmbientAndDiffuse() [2/3]	330
8.26.3.8 loadAmbientAndDiffuse() [3/3]	330
8.26.3.9 loadDiffuse() [1/3]	330
8.26.3.10 loadDiffuse() [2/3]	331
8.26.3.11 loadDiffuse() [3/3]	331
8.26.3.12 loadShininess() [1/2]	332
8.26.3.13 loadShininess() [2/2]	332

8.26.3.14 loadSpecular() [1/3]	332
8.26.3.15 loadSpecular() [2/3]	333
8.26.3.16 loadSpecular() [3/3]	333
8.26.3.17 select()	333
8.27 Menu クラス	334
8.27.1 詳解	334
8.27.2 構築子と解体子	334
8.27.2.1 Menu() [1/2]	334
8.27.2.2 Menu() [2/2]	335
8.27.2.3 ~Menu()	335
8.27.3 関数詳解	335
8.27.3.1 draw()	335
8.27.3.2 operator=()	335
8.27.4 フレンドと関連関数の詳解	335
8.27.4.1 Draw	335
8.28 GgApp::Window クラス	335
8.28.1 詳解	337
8.28.2 構築子と解体子	337
8.28.2.1 Window() [1/3]	337
8.28.2.2 Window() [2/3]	338
8.28.2.3 Window() [3/3]	338
8.28.2.4 ~Window()	338
8.28.3 関数詳解	338
8.28.3.1 get()	338
8.28.3.2 getAltArrowX()	339
8.28.3.3 getAltArrowY()	339
8.28.3.4 getAltArrow()	340
8.28.3.5 getArrow() [1/2]	340
8.28.3.6 getArrow() [2/2]	341
8.28.3.7 getArrowX()	342
8.28.3.8 getArrowY()	342
8.28.3.9 getAspect()	343
8.28.3.10 getControlArrow()	343
8.28.3.11 getControlArrowX()	344
8.28.3.12 getControlArrowY()	345
8.28.3.13 getFboHeight()	345
8.28.3.14 getFboSize() [1/2]	346
8.28.3.15 getFboSize() [2/2]	346
8.28.3.16 getFboWidth()	346
8.28.3.17 getHeight()	347
8.28.3.18 getKey()	347
8.28.3.19 getLastKey()	347

8.28.3.20 getMouse() [1/3]	348
8.28.3.21 getMouse() [2/3]	348
8.28.3.22 getMouse() [3/3]	348
8.28.3.23 getMouseX()	348
8.28.3.24 getMouseY()	349
8.28.3.25 getRotation()	349
8.28.3.26 getRotationMatrix()	349
8.28.3.27 getScrollMatrix()	349
8.28.3.28 getShiftArrow()	350
8.28.3.29 getShiftArrowX()	350
8.28.3.30 getShiftArrowY()	351
8.28.3.31 getSize() [1/2]	352
8.28.3.32 getSize() [2/2]	352
8.28.3.33 getTranslation()	352
8.28.3.34 getTranslationMatrix()	353
8.28.3.35 getUserPointer()	353
8.28.3.36 getWheel() [1/3]	353
8.28.3.37 getWheel() [2/3]	353
8.28.3.38 getWheel() [3/3]	354
8.28.3.39 getWheelX()	354
8.28.3.40 getWheelY()	354
8.28.3.41 getWidth()	355
8.28.3.42 operator bool()	355
8.28.3.43 operator=() [1/2]	355
8.28.3.44 operator=() [2/2]	355
8.28.3.45 reset()	356
8.28.3.46 resetRotation()	356
8.28.3.47 resetTranslation()	356
8.28.3.48 restoreViewport()	357
8.28.3.49 selectInterface()	357
8.28.3.50 setClose()	357
8.28.3.51 setKeyboardFunc()	357
8.28.3.52 setMouseFunc()	357
8.28.3.53 setResizeFunc()	358
8.28.3.54 setUserPointer()	358
8.28.3.55 setVelocity()	358
8.28.3.56 setWheelFunc()	358
8.28.3.57 shouldClose()	359
8.28.3.58 swapBuffers()	359
8.28.3.59 updateViewport()	359

9.1 Config.cpp ファイル	361
9.1.1 詳解	362
9.1.2 変数詳解	362
9.1.2.1 defaultLight	362
9.2 Config.hpp	362
9.3 Config.h ファイル	364
9.3.1 詳解	365
9.4 Config.h	366
9.5 Draw.cpp ファイル	367
9.5.1 詳解	367
9.6 Draw.hpp	368
9.7 Draw.h ファイル	369
9.7.1 詳解	370
9.8 Draw.h	370
9.9 gg.cpp ファイル	370
9.9.1 詳解	371
9.10 gg.hpp	371
9.11 gg.h ファイル	447
9.11.1 詳解	452
9.11.2 マクロ定義詳解	452
9.11.2.1 ggError	452
9.11.2.2 ggFBOError	452
9.11.3 型定義詳解	452
9.11.3.1 pathChar	452
9.11.3.2 pathString	453
9.11.4 関数詳解	453
9.11.4.1 TCHARToUtf8()	453
9.11.4.2 Utf8ToTCHAR()	453
9.12 gg.h	454
9.13 GgApp.cpp ファイル	508
9.13.1 詳解	508
9.14 GgApp.hpp	509
9.15 GgApp.h ファイル	522
9.15.1 詳解	523
9.15.2 マクロ定義詳解	523
9.15.2.1 GG_BUTTON_COUNT	523
9.15.2.2 GG_INTERFACE_COUNT	524
9.15.2.3 GG_USE_IMGUI	524
9.16 GgApp.h	524
9.17 ggsample01.cpp ファイル	532
9.17.1 マクロ定義詳解	532
9.17.1.1 CONFIG_FILE	532

9.17.1.2 PROJECT_NAME	532
9.18 ggsample01.cpp	533
9.19 main.cpp ファイル	533
9.19.1 詳解	534
9.19.2 マクロ定義詳解	534
9.19.2.1 HEADER_STR	534
9.19.3 関数詳解	535
9.19.3.1 main()	535
9.20 main.cpp	535
9.21 Menu.cpp ファイル	536
9.21.1 詳解	536
9.22 Menu.cpp	537
9.23 Menu.h ファイル	538
9.23.1 詳解	539
9.24 Menu.h	539
9.25 parseconfig.h ファイル	540
9.25.1 詳解	541
9.25.2 関数詳解	542
9.25.2.1 getString() [1/3]	542
9.25.2.2 getString() [2/3]	542
9.25.2.3 getString() [3/3]	543
9.25.2.4 getValue() [1/2]	543
9.25.2.5 getValue() [2/2]	544
9.25.2.6 getVector()	544
9.25.2.7 setString() [1/3]	545
9.25.2.8 setString() [2/3]	545
9.25.2.9 setString() [3/3]	546
9.25.2.10 setValue() [1/2]	546
9.25.2.11 setValue() [2/2]	547
9.25.2.12 setVector()	548
9.26 parseconfig.h	548
9.27 README.md ファイル	551
Index	553

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) 2011-2025 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

ggsample01

2.1 ゲームグラフィックス特論A 第1回 宿題

OpenGL の開発環境を整備してください。

- 宿題のひな形は [GitHub](#) にあります (宿題のひな形で使っている [補助プログラムの解説](#))。
- 詳しくは [講義のスライド](#) を参照してください。

2.2 宿題プログラムの作成に必要な環境

- Linux Mint 20.2 / Windows 10 (Visual Studio 2019) / macOS 12.1 (Xcode 13) 以降に対応しています。
- OpenGL 4.1 以降が実行できる環境 (対応した GPU を搭載したビデオカード や CPU) が必要です。
- macOS では M1 Mac (Apple Silicon) に対応した Universal Binary を作成するようにしています。
- Raspberry Pi 4B にも対応しました。make -f Makefile.rpi でビルドしてください。

2.3 補足

このプログラムを実行すると、次のような図形が表示されます。

- 今回はソースプログラムを修正していないので送る必要はありません。
- ひな形プログラムがコンパイル／実行できなかったら知らせてください。
- fork 推奨ですが解答をプレリクで受け取る気力はないと思います。

2.4 宿題プログラム用補助プログラムについて

ゲームグラフィックス特論 A/B で課す宿題プログラムでは、専用の補助プログラムを用意しています。これは以下の 3 つのファイルで構成されています。

- `gg.h / gg.cpp`
 - GLFW での利用を想定した OpenGL のローダとユーティリティ
- `Window.h`
 - ウィンドウやマウス関連のユーザインターフェースを管理する GLFW のラッパー

GLFW は OpenGL や、その後継の Vulkan を使用したアプリケーションを作成するための、非常にコンパクトなフレームワークです。本当はこれだけで簡単にアプリケーションが作れるのですが、授業内容とはあまり関係のない処理を分離するために、屋上屋ながら**この授業専用の**フレームワークを用意しました。なお、`gg.h / gg.cpp` には OpenGL の拡張機能を使用可能にする機能を含んでいますので、別に GLEW や glad、GL3Wなどを導入する必要はありません。

また、この `Window.h` には Dear ImGui をサポートする機能を含んでいます。このプログラム (`ggsample01`) には、その使い方のサンプルコードを示すために Dear ImGui のソースプログラムも含めています。日本語メニューの表示のために M+ FONTS の `Mplus1-Regular.ttf` もリポジトリに含めています。

このほか、この `Window.h` には Oculus Rift (DK1, DK2, CV1, S) をサポートする機能を組み込んでいます。これを使って、C++ だけで VR アプリケーション() が作れます。

2.4.1 補助プログラムのドキュメント

Doxxygen で生成したドキュメントの [HTML 版](#) を `html` フォルダに、[PDF 版](#) を `pdf` フォルダに置いています。

2.4.2 補助プログラムの使い方

補助プログラムを使用するには、最小限、GLFW が使える環境が必要です。ゲームグラフィックス特論 A/B の宿題のリポジトリには Windows 用および macOS 用にコンパイルしたライブラリファイル一式を含めていますので、これらについては宿題のために別に用意する必要はありません。Linux (および Raspberry Pi) では `libglfw3-dev` パッケージをインストールしておいてください (% sudo apt-get install libglfw3-dev)。`gg.h`, `gg.cpp`, `Window.h` だけを使うときは、それぞれの環境で GLFW をインストールしておいてください。この補助プログラムを使用した最小のプログラムは、多分こんな感じになります。このソースファイルと同じところに `gg.h`, `gg.cpp`, `Window.h` を置き、`gg.cpp` と一緒にコンパイルして、GLFW のライブラリファイルをリンクしてください。

```
#include "Window.h"

int main()
{
    Window::init();

    Window window;

    while (window)
    {
        // // ここで OpenGL による描画を行う
        //
    }
}
```

使用する OpenGL のバージョンは、`Window::init(major, minor)` の major と minor で指定できます。major を 0 にするか省略すると、OpenGL のバージョンの指定を行いません。その場合は、macOS 以外では恐らく OpenGL のハードウェアもしくはドライバで対応可能な最大のバージョンが使用されます。なお、3.2 以降を指定したときは macOS 対応の都合で forward compatible プロファイルと core プロファイルを有効にします。macOS の場合は `Window::init(3, 2)` もしくは `Window::init(4, 1)` を指定してください。

```
// for macOS
Window::init(4, 1);
```

2.4.3 Oculus Rift を使う場合

#include "Window.h" の前に #define USE_OCULUS_RIFT を置いてください。DK1 / DK2 用か CV1 / S 用かは、使用する LibOVR のバージョンが 1.0 以前か以降かで判断しています。ただし DK1 / DK2 用 (LibOVR 0.8) のサポートは、今後は継続しない可能性があります。

```
// ウィンドウ関連の処理
#define USE_OCULUS_RIFT
#include "Window.h"
```

あるいは、Window.h の中に #define USE_OCULUS_RIFT を置いてください。

```
// Oculus Rift を使うなら
#define USE_OCULUS_RIFT
```

実際の使い方は、「Oculus Rift に図形を表示するプログラムを C++ で作る」を参考にしてください。この記事では以前の補助プログラムを使って解説していますが、Window クラスの使い方は変わりません (以前の補助プログラムでは GgApplication クラス内に置いていました)。

2.4.4 Dear ImGui を使う場合

すべての #include "Window.h" の前に、#define USE_IMGUI を置いてください。

```
// ウィンドウ関連の処理
#define USE_IMGUI
#include "Window.h"
```

あるいは、Window.h の中に #define USE_IMGUI を置いてください。

```
// Dear ImGui を使うなら
#define USE_IMGUI
```

そして、OpenGL の描画ループの中で ImGui::NewFrame(); と ImGui::Render(); の間に Dear ImGui の API を置いてください。Dear ImGui のウィンドウの実際のレンダリング(ImGui_ImplOpenGL3_RenderDrawData(); の呼び出し)は window.swapbuffers() の内で行っているので、Dear ImGui の API と OpenGL の API は描画ループの中で混在していても構いません。

なお、Dear ImGui を有効にした場合は、Dear ImGui がマウスを使っているとき (io.WantCaptureMouse == true) に、Window クラスが保持しているマウスカーソルの位置を更新しないようにしています。また、Dear ImGui がキーボードを使っているとき (io.WantCaptureKeyboard == true) には、Window クラスはキーボードのイベントを処理しないようにしています。

```
// ウィンドウ関連の処理
```

```
#define USE_IMGUI
#include "Window.h"
```

```
int main()
{
    // ウィンドウ関連の初期設定
    Window::init(4, 1);

    // ウィンドウを作成する
    Window window("Window Title", 1280, 720);

    // ImGui の初期設定
    ImGui::StyleColorsDark();

    // 背景色を指定する
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

    // ウィンドウが開いている間繰り返す
    while (window)
    {
        // ImGui のフレームを準備する
        ImGui::NewFrame();

        // ImGui のフレームに一つ目の ImGui のウィンドウを描く
        ImGui::Begin("Control panel");
        ImGui::Text("Frame rate: %6.2f fps", ImGui::GetIO().Framerate);
        if (ImGui::Button("Quit")) window.setClose();
        ImGui::End();

        // ImGui のフレームに描画する
        ImGui::Render();

        // ウィンドウを消去する
        window.swapbuffers();
    }
}
```

```

glClear(GL_COLOR_BUFFER_BIT);

// ここで OpenGL による描画を行う
//

// カラーバッファを入れ替えてイベントを取り出す
window.swapBuffers();
}

}

```

このソースファイルと Dear ImGui に含まれる以下のファイル、および [gg.h](#), [gg.cpp](#), [Window.h](#) を同じところに置き、[gg.cpp](#) と以下のうちの *.cpp ファイルと一緒にコンパイルして、GLFW のライブラリファイルをリンクしてください。

```

imconfig.h
imgui.h
imgui_impl_glfw.h
imgui_impl_opengl3.h
imgui_impl_opengl3_loader.h
imgui_internal.h
imstb_rectpack.h
imstb_textedit.h
imstb_truetype.h

```

```

imgui.cpp
imgui_draw.cpp
imgui_impl_glfw.cpp
imgui_impl_opengl3.cpp
imgui_tables.cpp
imgui_widgets.cpp

```

2.4.4.1 imconfig.h の変更点

Dear ImGui はバージョン 1.86 から独自のローダを使用するようになったので、`IMGUI_IMPL_OPENGL_`
`_LOADER_CUSTOM` にこの授業オリジナルのローダ [gg.h](#)/[gg.cpp](#) を指定する必要はありません。ただし、Raspberry Pi では `imconfig.h` の**最後**で記号定数 `IMGUI_IMPL_OPENGL_ES3` を明示的に定義する必要があります。

```

// The Raspberry Pi needs to explicitly define the symbolic constant IMGUI_IMPL_OPENGL_ES3.
#if defined(__RASPBERRY_PI__)
#define IMGUI_IMPL_OPENGL_ES3
#endif

```

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](#) 15

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	103
gg::GgMatrix	293
gg::GgVector	176
gg::GgQuaternion	270
gg::GgTrackball	
Config	81
Draw	85
GgApp	86
gg::GgBuffer< T >	89
gg::GgColorTexture	94
gg::GgNormalTexture	159
gg::GgPointShader	167
gg::GgSimpleShader	251
gg::GgShader	241
gg::GgShape	244
gg::GgPoints	163
gg::GgTriangles	282
gg::GgElements	98
gg::GgSimpleObj	248
gg::GgTexture	265
gg::GgUniformBuffer< T >	286
gg::GgUniformBuffer< Light >	286
gg::GgSimpleShader::LightBuffer	314
gg::GgUniformBuffer< Material >	286
gg::GgSimpleShader::MaterialBuffer	325
gg::GgVertex	307
gg::GgVertexArray	310
gg::GgSimpleShader::Light	312
gg::GgSimpleShader::Material	323
Menu	334
GgApp::Window	335

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。

Config	81
Draw	85
GgApp	86
gg::GgBuffer< T >	89
gg::GgColorTexture	94
gg::GgElements	98
gg::GgMatrix	103
gg::GgNormalTexture	159
gg::GgPoints	163
gg::GgPointShader	167
gg::GgQuaternion	176
gg::GgShader	241
gg::GgShape	244
gg::GgSimpleObj	248
gg::GgSimpleShader	251
gg::GgTexture	265
gg::GgTrackball	270
gg::GgTriangles	282
gg::GgUniformBuffer< T >	286
gg::GgVector	293
gg::GgVertex	307
gg::GgVertexArray	310
gg::GgSimpleShader::Light	312
gg::GgSimpleShader::LightBuffer	314
gg::GgSimpleShader::Material	323
gg::GgSimpleShader::MaterialBuffer	325
Menu	334
GgApp::Window	335

Chapter 6

ファイル索引

6.1 ファイル一覧

ファイル一覧です。

Config.cpp	361
Config.h	364
Draw.cpp	367
Draw.h	369
gg.cpp	370
gg.h	447
GgApp.cpp	508
GgApp.h	522
ggsample01.cpp	532
main.cpp	533
Menu.cpp	536
Menu.h	538
parseconfig.h	540

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 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 >ext="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) [extern]
```

OpenGL のエラーをチェックする.

引数

name	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.
line	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 2609 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) [extern]
```

FBO のエラーをチェックする。

引数

<code>name</code>	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.
<code>line</code>	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 2653 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) [extern]
```

Wavefront OBJ ファイルを読み込む (Arrays 形式)

引数

<code>name</code>	ファイル名.
<code>normalize</code>	<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 5313 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

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

戻り値

四元数 *q* の共役四元数.

gg.h の 4715 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") [extern]
```

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 5086 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) [extern]
```

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

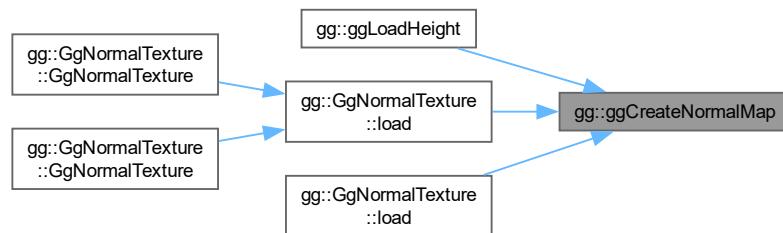
<i>hmap</i>	グレースケール画像のデータ.
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数.
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数.
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_RGBA).
<i>nz</i>	法線の z 成分の割合.
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット.
<i>nmap</i>	法線マップを格納する vector.

gg.cpp の 3899 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") [extern]
```

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

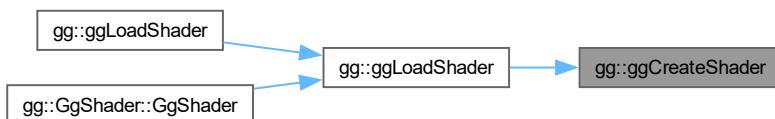
<i>vsrc</i>	バーテックスシェーダのソースプログラムの文字列.
<i>fsrc</i>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用).
<i>gsrc</i>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).
<i>vtext</i>	バーテックスシェーダのコンパイル時のメッセージに追加する文字列.
<i>ftext</i>	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.
<i>gtext</i>	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 4908 行目に定義があります。

被呼び出し関係図:



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

<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* の外積.

覚え書き

gg.h の 1990 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>	GgVector 型のベクトル.
<i>b</i>	GgVector 型のベクトル.

戻り値

a と *b* の距離.

gg.h の 2027 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

<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.

戻り値

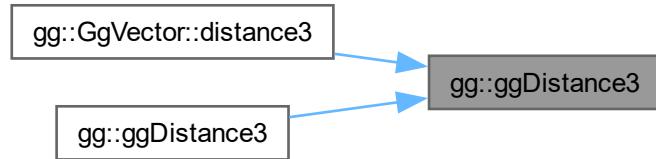
`a` と `b` の距離.

`gg.h` の 1466 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



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

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

戻り値

2つの `GgVector` a, b の距離.

`gg.h` の 2093 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

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



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

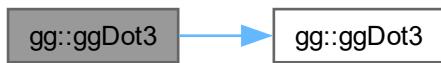
<i>a</i>	GgVector 型のベクトル.
<i>b</i>	GgVector 型のベクトル.

戻り値

a と *b* のそれぞれの 3 要素の内積.

gg.h の 2004 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

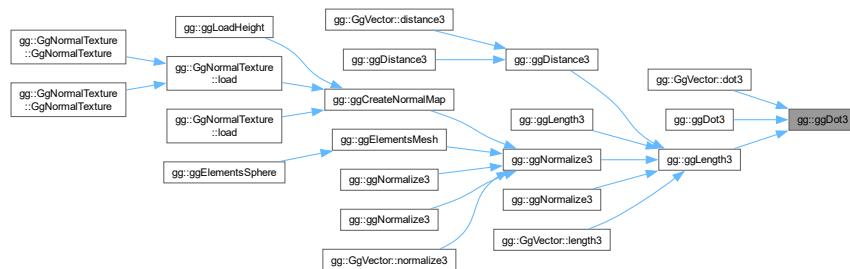
<i>a</i>	GLfloat 型の 3 要素の配列変数.
<i>b</i>	GLfloat 型の 3 要素の配列変数.

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。

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>	<code>GgVector</code> 型の変数.
<i>b</i>	<code>GgVector</code> 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 2070 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

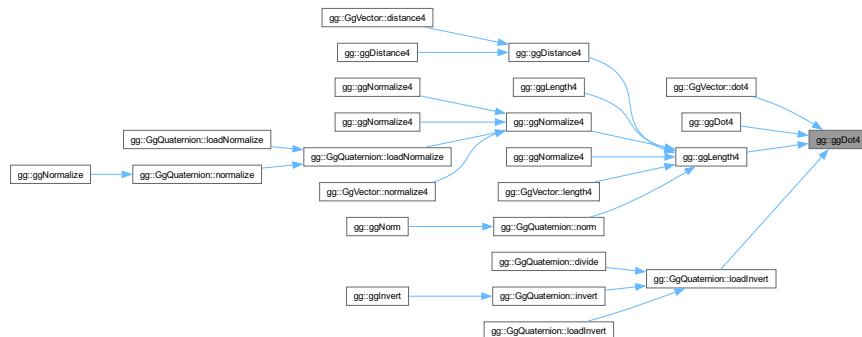
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	GLfloat 型の 4 要素の配列変数.

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)  [extern]
```

メッシュ形状を作成する (Elements 形式).

引数

<i>slices</i>	メッシュの横方向の分割数.
<i>stacks</i>	メッシュの縦方向の分割数.
<i>pos</i>	メッシュの頂点の位置.
<i>norm</i>	メッシュの頂点の法線, nullptr 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 5347 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) [extern]
```

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

引数

<code>name</code>	ファイル名.
<code>normalize</code>	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 5329 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) [extern]
```

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 5442 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) [extern]
```

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 5288 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

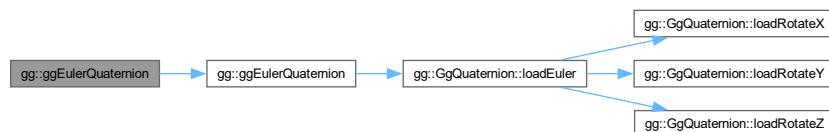
<code>e</code>	オイラー角を表す <code>GLfloat</code> 型の 3 要素の配列変数 (<code>heading, pitch, roll</code>).
----------------	---

戻り値

回転を表す四元数。

`gg.h` の 4629 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

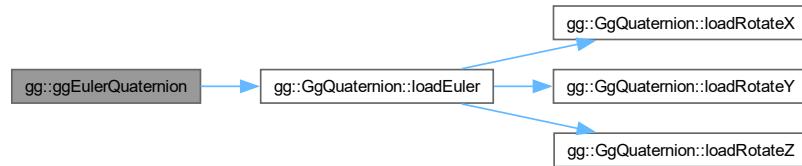
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転を表す四元数.

gg.h の 4617 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 3382 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 3107 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 4515 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 () [extern]
```

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 1354 行目に定義があります。

被呼び出し関係図:



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 3427 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

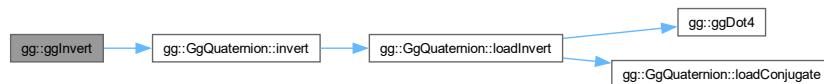
<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

四元数 `q` の逆元。

`gg.h` の 4726 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

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

戻り値

a の 3 要素の長さ.

gg.h の 2015 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

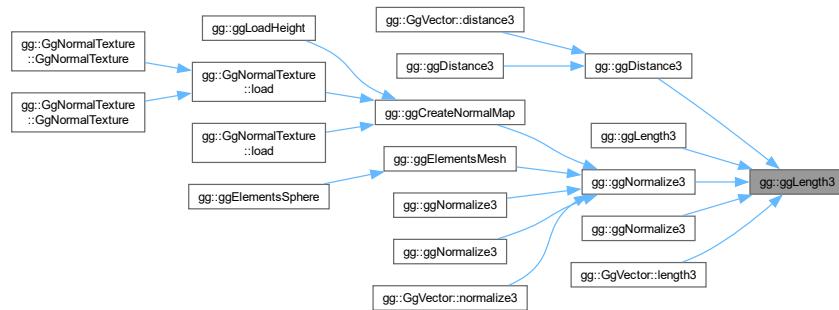
a の 3 要素の長さ.

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	GgVector 型の変数.
---	----------------

戻り値

a の 4 要素の長さ.

gg.h の 2081 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

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

戻り値

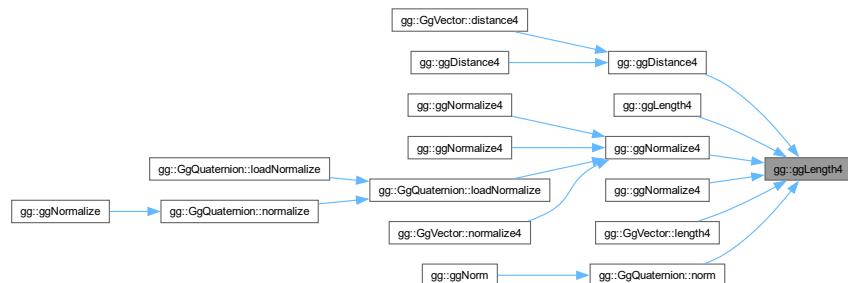
a の 4 要素の長さ.

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) [extern]
```

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

comp	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 5127 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) [extern]
```

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

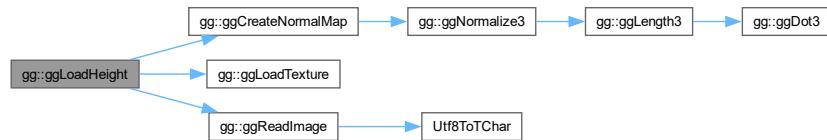
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

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 3984 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) [extern]
```

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

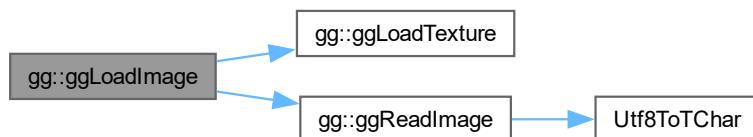
<i>name</i>	読み込むファイル名。
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる。
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

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 3850 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 <i>varying</i> 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト (<i>nullptr</i> 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 5115 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) [extern]
```

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 <i>varying</i> 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト (<i>nullptr</i> なら不使用).

戻り値

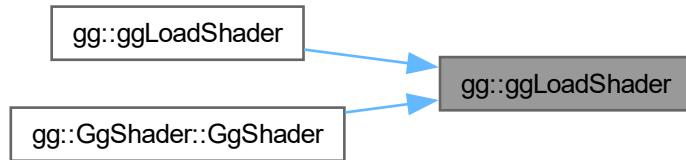
プログラムオブジェクトのプログラム名(作成できなければ 0).

gg.cpp の 5053 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) [extern]
  
```

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

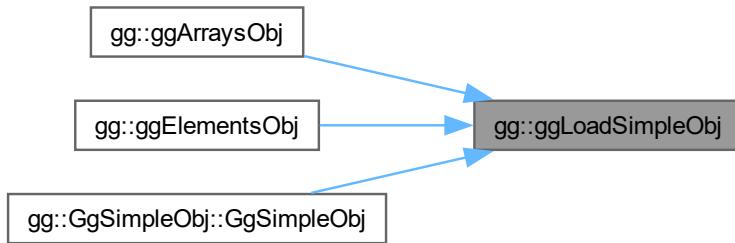
<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの GgSimpleShader::Material 型の材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>normalize</i>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

[gg.cpp](#) の 4660 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) [extern]
```

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 4749 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) [extern]
```

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

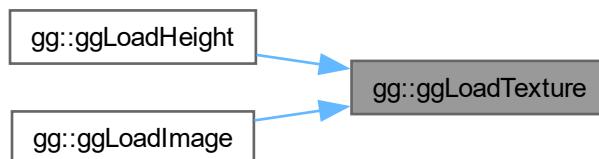
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャの作成のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <i>image</i> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <i>image</i> の縦の画素数.
<i>format</i>	<i>image</i> のフォーマット.
<i>type</i>	<i>image</i> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは <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 3802 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 GgVector 型の変数.
<i>t</i>	目標点の位置を格納した GgVector 型の変数.
<i>u</i>	上方向のベクトルを格納した GgVector 型の変数.

戻り値

求めたビュー変換行列.

[gg.h](#) の 3342 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 GLfloat 型の 3 要素の配列変数.
<i>t</i>	目標点の位置を格納した GLfloat 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 GLfloat 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 3324 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 3306 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:



7.1.3.45 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

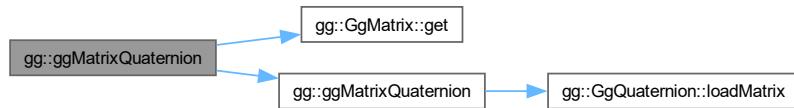
m	GgMatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4539 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

a	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 4527 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>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

四元数 `q` のノルム。

`gg.h` の 4693 行目に定義があります。

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

<i>m</i>	元の変換行列.
----------	---------

戻り値

m の法線変換行列.

gg.h の 3438 行目に定義があります。

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

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

戻り値

四元数 *q* を正規化した四元数.

gg.h の 4704 行目に定義があります。

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

a	GgVector 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 2041 行目に定義があります。

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

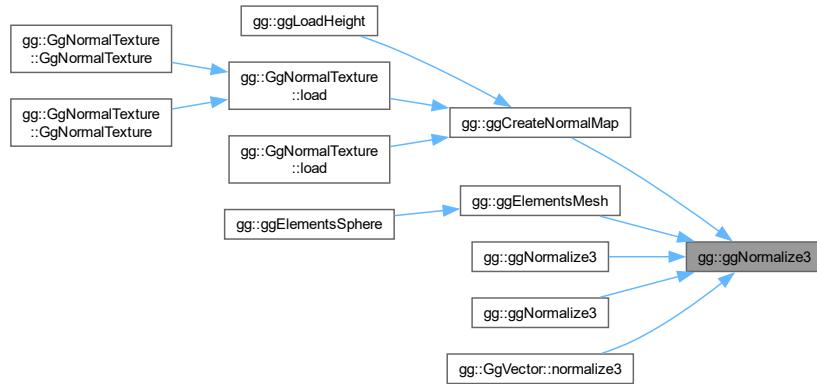
a	GLfloat 型の 3 要素の配列変数。
b	a の 3 要素を正規化した結果を格納する GLfloat 型の 3 要素の配列変数。

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

<code>a</code>	<code>GgVector</code> 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 2057 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 2104 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

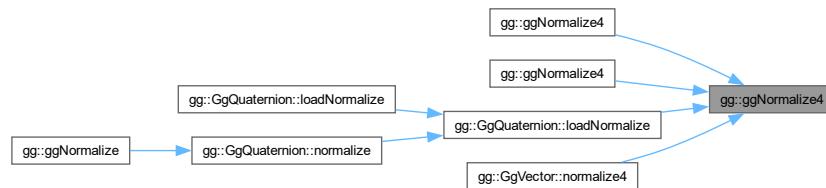
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	<i>a</i> を正規化した結果を格納する GLfloat 型の 4 要素の配列変数.

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

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

gg.h の 2116 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 3363 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 3401 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:



7.1.3.60 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) [extern]
```

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 5214 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) [extern]
```

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 5240 行目に定義があります。

7.1.3.62 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

<i>q</i>	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 4550 行目に定義があります。

呼び出し関係図:



7.1.3.63 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

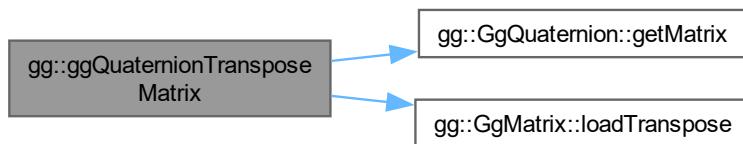
<i>q</i>	元の四元数.
----------	--------

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 4563 行目に定義があります。

呼び出し関係図:



7.1.3.64 ggReadImage()

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

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

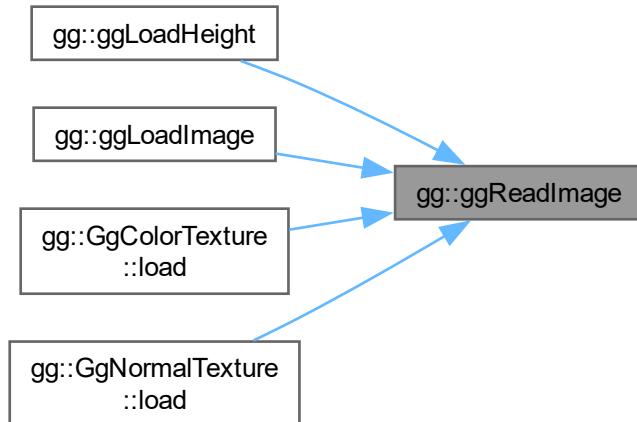
読み込みに成功すれば `true`, 失敗すれば `false`.

`gg.cpp` の 3681 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:



7.1.3.65 `ggRectangle()`

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

矩形状に 2 枚の三角形を生成する.

引数

<code>width</code>	矩形の横幅.
<code>height</code>	矩形の高さ.

戻り値

`GgTriangles` 型のポインタ.

`gg.cpp` の 5267 行目に定義があります。

7.1.3.66 `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

<code>r</code>	回転軸のベクトルと回転角を表す <code>GgVector</code> 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 3286 行目に定義があります。

呼び出し関係図:



7.1.3.67 `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

<code>r</code>	回転軸のベクトルを表す <code>GgVector</code> 型の変数.
<code>a</code>	回転角.

戻り値

`r` を軸に `a` だけ回転する変換行列.

`gg.h` の 3262 行目に定義があります。

呼び出し関係図:



7.1.3.68 `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

<code>r</code>	回転軸のベクトルと回転角を表す <code>GLfloat</code> 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 3274 行目に定義があります。

呼び出し関係図:



7.1.3.69 `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 3249 行目に定義があります。

呼び出し関係図:



7.1.3.70 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 <i>x</i> 成分.
<i>y</i>	回転軸の <i>y</i> 成分.
<i>z</i>	回転軸の <i>z</i> 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 3236 行目に定義があります。

呼び出し関係図:



7.1.3.71 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 4604 行目に定義があります。

呼び出し関係図:



7.1.3.72 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 4593 行目に定義があります。

呼び出し関係図:



7.1.3.73 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

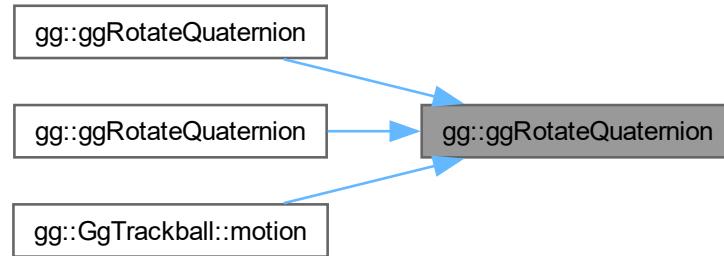
回転を表す四元数.

gg.h の 4580 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.74 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 3197 行目に定義があります。

呼び出し関係図:



7.1.3.75 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 3209 行目に定義があります。

呼び出し関係図:



7.1.3.76 ggRotateZ()

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

z 軸中心の回転の変換行列を返す.

引数

a	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 3221 行目に定義があります。

呼び出し関係図:



7.1.3.77 **ggSaveColor()**

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

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 3625 行目](#)に定義がります。

呼び出し関係図:



7.1.3.78 ggSaveDepth()

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

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 3651 行目](#)に定義がります。

呼び出し関係図:



7.1.3.79 ggSaveTga()

```
bool gg::ggSaveTga (
    const std::string & name,
    const void * buffer,
    unsigned int width,
    unsigned int height,
    unsigned int depth) [extern]
```

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

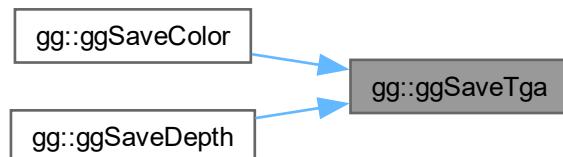
保存に成功すれば true, 失敗すれば false.

gg.cpp の 3535 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:



7.1.3.80 ggScale() [1/3]

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

拡大縮小の変換行列を返す.

引数

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

戻り値

拡大縮小の変換行列.

gg.h の 3185 行目に定義があります。

呼び出し関係図:



7.1.3.81 ggScale() [2/3]

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

拡大縮小の変換行列を返す.

引数

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

戻り値

拡大縮小の変換行列.

gg.h の 3173 行目に定義があります。

呼び出し関係図:



7.1.3.82 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 3161 行目に定義があります。

呼び出し関係図:



7.1.3.83 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>	<code>GgQuaternion</code> 型の四元数。
<i>r</i>	<code>GgQuaternion</code> 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 4656 行目に定義があります。

呼び出し関係図:



7.1.3.84 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

q	GgQuaternion 型の四元数.
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
t	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 4669 行目に定義があります。

呼び出し関係図:



7.1.3.85 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 <code>GLfloat</code> 型の 4 要素の配列変数.
<i>q</i>	<code>GgQuaternion</code> 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

a, q を *t* で内分した四元数.

`gg.h` の 4682 行目に定義があります。

呼び出し関係図:



7.1.3.86 `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

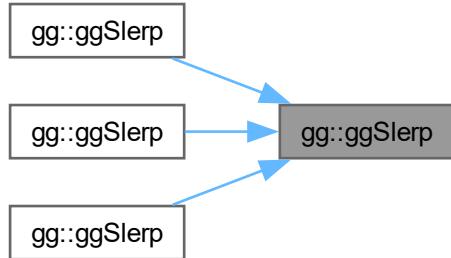
a, b を *t* で内分した四元数.

`gg.h` の 4642 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:



7.1.3.87 ggTranslate() [1/3]

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

平行移動の変換行列を返す.

引数

t	移動量の GgVector 型の変数.
---	---------------------

戻り値

平行移動の変換行列.

gg.h の 3146 行目に定義があります。

呼び出し関係図:



7.1.3.88 ggTranslate() [2/3]

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

平行移動の変換行列を返す。

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

平行移動の変換行列。

[gg.h](#) の 3134 行目に定義があります。

呼び出し関係図:



7.1.3.89 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 3122 行目に定義があります。

呼び出し関係図:



7.1.3.90 ggTranspose()

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

転置行列を返す.

引数

<i>m</i>	元の変換行列.
----------	---------

戻り値

m の転置行列.

gg.h の 3416 行目に定義があります。

呼び出し関係図:



7.1.3.91 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 1963 行目に定義があります。

7.1.3.92 operator+() [1/2]

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

単項の + 演算子なので何もしない.

引数

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

戻り値

`v` の値.

`gg.h` の 1916 行目に定義があります。

7.1.3.93 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

<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` の各要素を足した和.

`gg.h` の 1928 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]
```

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 1939 行目に定義があります。

7.1.3.95 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

a	GLfloat 型の値.
b	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 1951 行目に定義があります。

7.1.3.96 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

a	GLfloat 型の値.
b	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 1975 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 Config クラス

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

公開メンバ関数

- `Config ()`
- `Config (const std::string &filename)`
- `virtual ~Config ()`
- `auto getWidth () const`
- `auto getHeight () const`
- `bool load (const std::string &filename)`
- `bool save (const std::string &filename) const`

フレンド

- class `Menu`

8.1.1 詳解

構成データ

`Config.h` の 21 行目に定義があります。

8.1.2 構築子と解体子

8.1.2.1 Config() [1/2]

```
Config::Config ()
```

コンストラクタ

`Config.cpp` の 25 行目に定義があります。

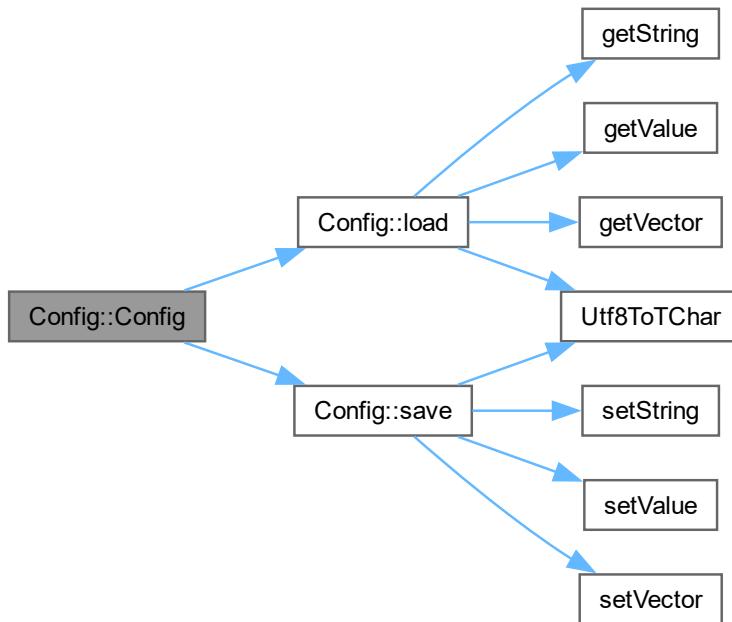
8.1.2.2 Config() [2/2]

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

ファイルから構成データを読み込むコンストラクタ

[Config.cpp](#) の 42 行目に定義があります。

呼び出し関係図:



8.1.2.3 ~Config()

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

デストラクタ

[Config.cpp](#) の 52 行目に定義があります。

8.1.3 関数詳解

8.1.3.1 getHeight()

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

ウィンドウの横幅を得る

[Config.h](#) の 72 行目に定義があります。

8.1.3.2 getWidth()

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

ウィンドウの横幅を得る

[Config.h](#) の 64 行目に定義があります。

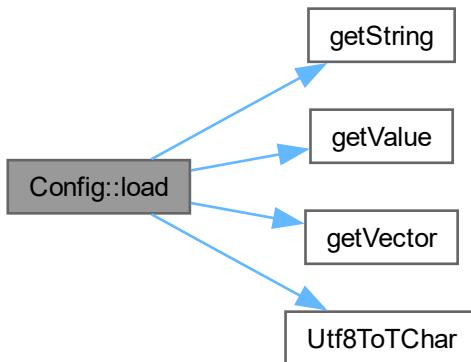
8.1.3.3 load()

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

設定ファイルを読み込む

[Config.cpp](#) の 59 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:



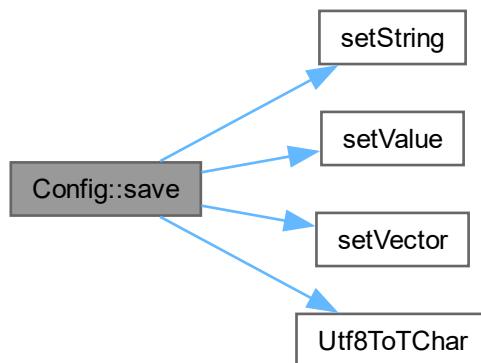
8.1.3.4 save()

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

設定ファイルを書き出す

[Config.cpp](#) の 109 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:



8.1.4 フレンドと関連関数の詳解

8.1.4.1 Menu

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

[Config.h](#) の 24 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](#)

8.2 Draw クラス

```
#include <Draw.h>
```

公開メンバ関数

- `Draw (const Menu &menu)`
- `virtual ~Draw ()`
- `void draw (const GgMatrix &mp, const GgMatrix &mv) const`

8.2.1 詳解

図形の描画クラス

`Draw.h` の 17 行目に定義があります。

8.2.2 構築子と解体子

8.2.2.1 Draw()

```
Draw::Draw (const Menu & menu)
```

コンストラクタ

`Draw.cpp` の 13 行目に定義があります。

8.2.2.2 ~Draw()

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

デストラクタ

`Draw.cpp` の 29 行目に定義があります。

8.2.3 関数詳解

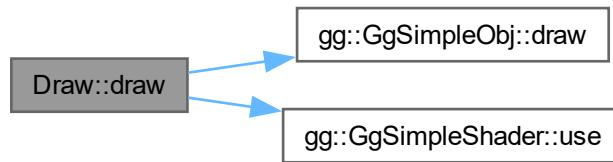
8.2.3.1 draw()

```
void Draw::draw (
    const GgMatrix & mp,
    const GgMatrix & mv) const
```

描画する

[Draw.cpp](#) の 36 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:

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

8.3 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](#)
- [GgApp \(GgApp &&w\)=default](#)
- virtual [~GgApp \(\)](#)
- [GgApp & operator= \(const GgApp &w\)=delete](#)
- [GgApp & operator= \(GgApp &&w\)=default](#)
- int [main \(int argc, const char *const *argv\)](#)

8.3.1 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 88 行目に定義がります。

8.3.2 構築子と解体子

8.3.2.1 GgApp() [1/3]

```
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 48 行目に定義がります。

8.3.2.2 GgApp() [2/3]

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

コピーコンストラクタは使用しない

8.3.2.3 GgApp() [3/3]

```
GgApp::GgApp (
    GgApp && w) [default]
```

ムーブコンストラクタはデフォルトのものを使用する

8.3.2.4 ~GgApp()

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

デストラクタ。

GgApp.cpp の 95 行目に定義がります。

8.3.3 関数詳解

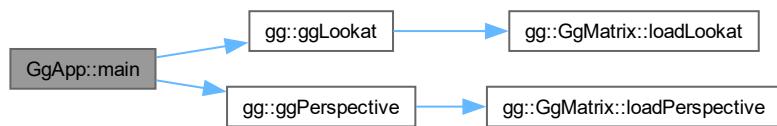
8.3.3.1 main()

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

アプリケーション本体。

[ggsample01.cpp](#) の 28 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:



8.3.3.2 operator=() [1/2]

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

代入演算子は使用しない

8.3.3.3 operator=() [2/2]

```
GgApp & GgApp::operator= (
    GgApp && w) [default]
```

ムーブ代入演算子はデフォルトのものを使用する

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](#)
- [ggsample01.cpp](#)

8.4 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)`
- `GgBuffer (const GgBuffer< T > &buffer)=delete`
- `GgBuffer (GgBuffer< T > &&buffer)=default`
- `virtual ~GgBuffer ()`
- `GgBuffer< T > & operator= (const GgBuffer< T > &buffer)=delete`
- `GgBuffer< T > & operator= (GgBuffer< T > &&buffer)=default`
- `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`

8.4.1 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 5520 行目に定義があります。

8.4.2 構築子と解体子

8.4.2.1 GgBuffer() [1/3]

```
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 (nullptr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 5532 行目に定義があります。

8.4.2.2 GgBuffer() [2/3]

```
template<typename T >
gg::GgBuffer< T >::GgBuffer (
    const GgBuffer< T > & buffer) [delete]
コピーコンストラクタは使用しない.
```

8.4.2.3 GgBuffer() [3/3]

```
template<typename T >
gg::GgBuffer< T >::GgBuffer (
    GgBuffer< T > && buffer) [default]
ムーブコンストラクタはデフォルトのものを使用する.
```

8.4.2.4 ~GgBuffer()

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

gg.h の 5532 行目に定義があります。

8.4.3 関数詳解

8.4.3.1 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 5635 行目に定義があります。
```

8.4.3.2 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 5740 行目に定義がります。

8.4.3.3 **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 5627 行目に定義がります。

8.4.3.4 **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 5617 行目に定義がります。

8.4.3.5 **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 5607 行目に定義がります。

8.4.3.6 `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 5597 行目に定義があります。

8.4.3.7 `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 5653 行目に定義があります。

8.4.3.8 `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 5670 行目に定義があります。

8.4.3.9 operator=() [1/2]

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

代入演算子は使用しない。

8.4.3.10 operator=() [2/2]

```
template<typename T >
GgBuffer< T > & gg::GgBuffer< T >::operator= (
    GgBuffer< T > && buffer) [default]
```

ムーブ代入演算子はデフォルトのものを使用する。

8.4.3.11 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 5713 行目に定義があります。

8.4.3.12 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 5695 行目に定義があります。

8.4.3.13 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 5643 行目に定義があります。

8.4.3.14 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 5683 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](#)

8.5 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 \(const std::string &name, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)

8.5.1 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 5299 行目に定義があります。

8.5.2 構築子と解体子

8.5.2.1 GgColorTexture() [1/3]

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

コンストラクタ.

[gg.h](#) の 5309 行目に定義があります。

8.5.2.2 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]
```

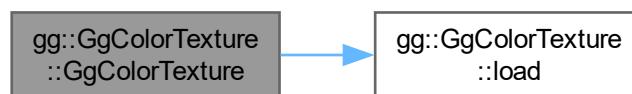
メモリ上のデータからカラーのテクスチャを作成するコンストラクタ.

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 5325 行目に定義があります。

呼び出し関係図:



8.5.2.3 GgColorTexture() [3/3]

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

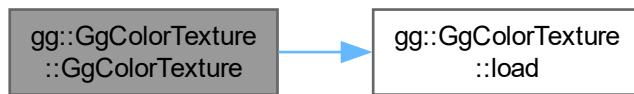
TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成するコンストラクタ.

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 5346 行目に定義があります。

呼び出し関係図:



8.5.2.4 ~GgColorTexture()

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

デストラクタ.

gg.h の 5358 行目に定義があります。

8.5.3 関数詳解

8.5.3.1 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]
```

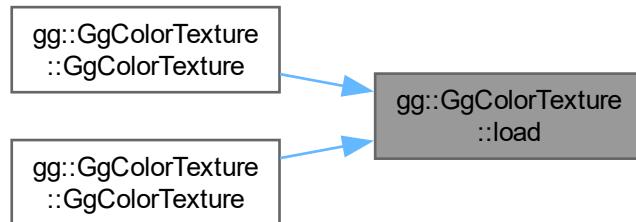
メモリ上のデータを読み込んでテクスチャを作成する.

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 5374 行目に定義があります。

被呼び出し関係図:



8.5.3.2 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>	<code>glTexImage2D()</code> 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 (<code>GL_CLAMP_TO_EDGE</code> , <code>GL_CLAMP_TO_BORDER</code> , <code>GL_REPEAT</code> , <code>GL_MIRRORED_REPEAT</code>).

gg.cpp の 4046 行目に定義がります。

呼び出し関係図:



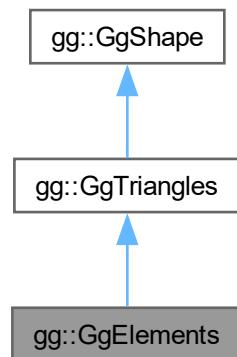
このクラス詳解は次のファイルから抽出されました:

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

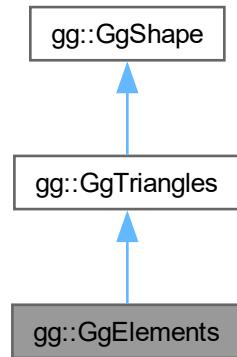
8.6 gg::GgElements クラス

```
#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 count, 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`

基底 クラス [gg::GgTriangles](#) に属する継承公開 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)`

基底 クラス [gg::GgShape](#) に属する継承公開 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`

8.6.1 詳解

三角形で表した形状データ (Elements 形式).

[gg.h](#) の 6535 行目に定義があります。

8.6.2 構築子と解体子

8.6.2.1 GgElements() [1/2]

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

コンストラクタ.

引数

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

[gg.h](#) の 6548 行目に定義があります。

8.6.2.2 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

<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>countv</i>	頂点数.
<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 6563 行目に定義があります。

8.6.2.3 ~GgElements()

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

デストラクタ.

[gg.h](#) の 6579 行目に定義があります。

8.6.3 関数詳解

8.6.3.1 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 5201 行目に定義があります。

呼び出し関係図:



8.6.3.2 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 6597 行目に定義があります。

8.6.3.3 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 6587 行目に定義があります。

8.6.3.4 `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

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

`gg.h` の 6634 行目に定義があります。

8.6.3.5 `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

<code>vert</code>	頂点属性が格納されている領域の先頭のポインタ.
<code>firstv</code>	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.
<code>countv</code>	頂点のデータの数 (頂点数).
<code>face</code>	三角形の頂点インデックスデータ.
<code>firstf</code>	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.
<code>countf</code>	三角形の頂点数.

`gg.h` の 6612 行目に定義があります。

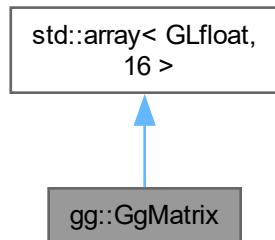
このクラス詳解は次のファイルから抽出されました:

- `gg.h`
- `gg.cpp`

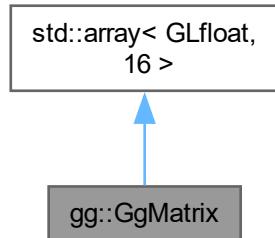
8.7 gg::GgMatrix クラス

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

gg::GgMatrix の継承関係図



gg::GgMatrix 連携図



公開 メンバ関数

- `GgMatrix ()=default`
- `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 (const GLfloat *a)`
- `constexpr GgMatrix (GLfloat c)`
- `GgMatrix (const GgMatrix &m)=default`
- `GgMatrix (GgMatrix &&m)=default`
- `~GgMatrix ()=default`
- `GgMatrix & operator= (const GgMatrix &m)=default`
- `GgMatrix & operator= (GgMatrix &&m)=default`
- `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`

8.7.1 詳解

変換行列.

`gg.h` の 2124 行目に定義があります。

8.7.2 構築子と解体子

8.7.2.1 GgMatrix() [1/6]

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

コンストラクタ. 被呼び出し関係図:



8.7.2.2 GgMatrix() [2/6]

```
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 2159 行目に定義があります。

8.7.2.3 GgMatrix() [3/6]

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

コンストラクタ。

引数

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

gg.h の 2174 行目に定義があります。

8.7.2.4 GgMatrix() [4/6]

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

コンストラクタ.

引数

c	GLfloat 型の値.
---	--------------

gg.h の 2184 行目に定義があります。

8.7.2.5 GgMatrix() [5/6]

```
gg::GgMatrix::GgMatrix (
    const GgMatrix & m) [default]
```

コピー構造.

8.7.2.6 GgMatrix() [6/6]

```
gg::GgMatrix::GgMatrix (
    GgMatrix && m) [default]
```

ムーブ構造.

8.7.2.7 ~GgMatrix()

```
gg::GgMatrix::~GgMatrix () [default]
```

デストラクタ.

8.7.3 関数詳解

8.7.3.1 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

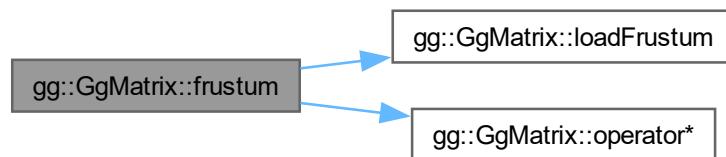
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2963 行目に定義があります。

呼び出し関係図:



8.7.3.2 `get()` [1/2]

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

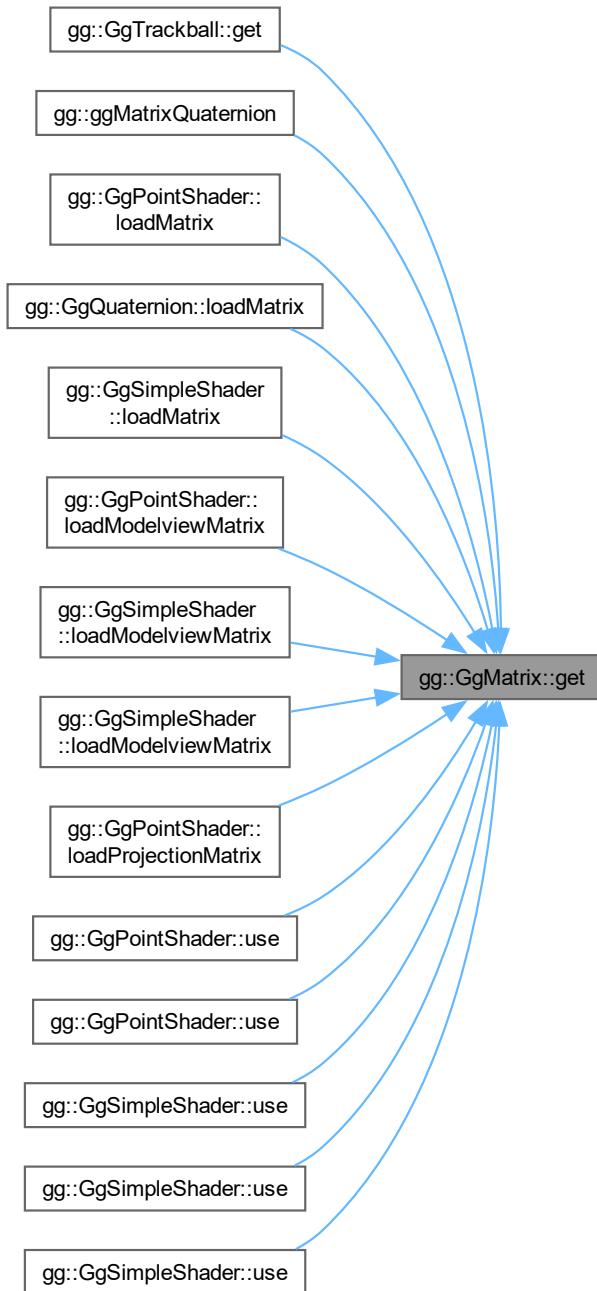
変換行列を取り出す.

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 3086 行目に定義があります。

被呼び出し関係図:



8.7.3.3 `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 3096 行目に定義があります。

8.7.3.4 invert()

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

逆行列を返す.

戻り値

逆行列.

gg.h の 3007 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:



8.7.3.5 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

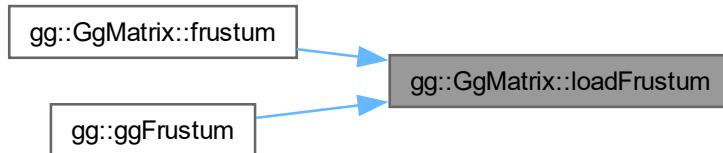
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 3123 行目に定義があります。

被呼び出し関係図:



8.7.3.6 loadIdentity()

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

単位行列を格納する.

gg.cpp の 2734 行目に定義があります。

被呼び出し関係図:



8.7.3.7 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 2686 行目に定義があります。

呼び出し関係図:



8.7.3.8 `loadInvert()` [2/2]

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

逆行列を格納する.

引数

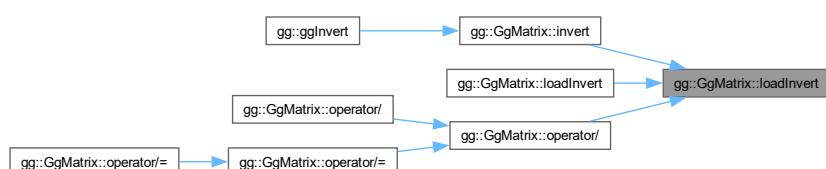
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列.
----------------	-------------------------------------

戻り値

設定した `a` の逆行列.

`gg.cpp` の 2909 行目に定義があります。

被呼び出し関係図:



8.7.3.9 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 2606 行目に定義があります。

呼び出し関係図:



8.7.3.10 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

<i>e</i>	視点の位置の配列変数.
<i>t</i>	目標点の位置の配列変数.
<i>u</i>	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 2593 行目に定義があります。

呼び出し関係図:



8.7.3.11 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

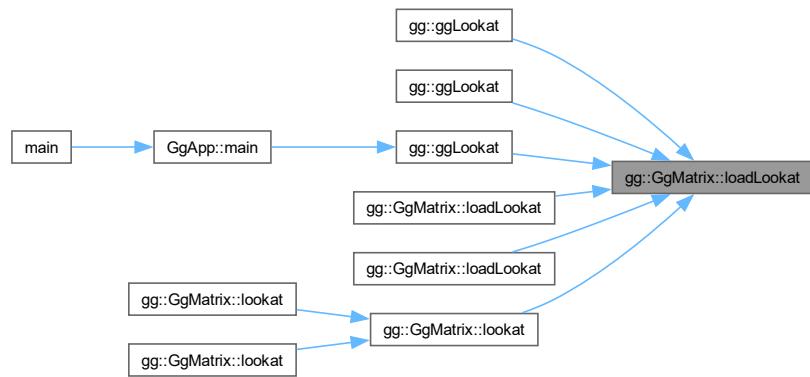
<i>ex</i>	視点の位置の x 座標値.
<i>ey</i>	視点の位置の y 座標値.
<i>ez</i>	視点の位置の z 座標値.
<i>tx</i>	目標点の位置の x 座標値.
<i>ty</i>	目標点の位置の y 座標値.
<i>tz</i>	目標点の位置の z 座標値.
<i>ux</i>	上方向のベクトルの x 成分.
<i>uy</i>	上方向のベクトルの y 成分.
<i>uz</i>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列.

gg.cpp の 3023 行目に定義があります。

被呼び出し関係図:



8.7.3.12 loadNormal() [1/2]

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

法線変換行列を格納する.

引数

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

戻り値

設定した *m* の法線変換行列.

gg.h の 2705 行目に定義があります。

呼び出し関係図:



8.7.3.13 loadNormal() [2/2]

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

法線変換行列を格納する.

引数

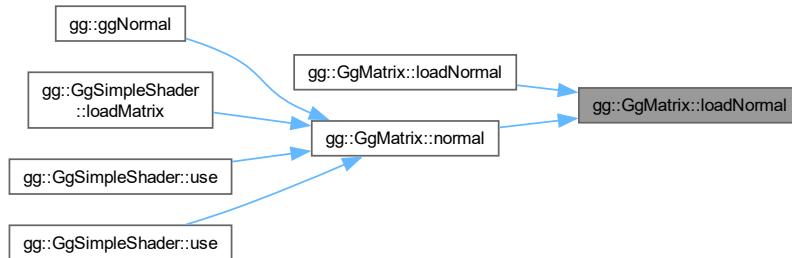
<i>a</i>	GLfloat 型の 16 要素の変換行列.
----------	------------------------

戻り値

設定した m の法線変換行列.

gg.cpp の 2992 行目に定義があります。

被呼び出し関係図:



8.7.3.14 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

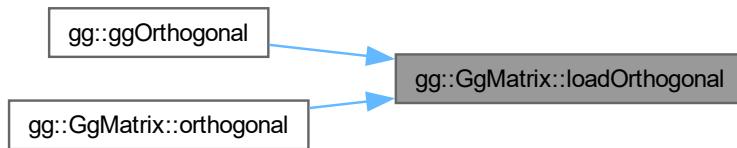
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した直交投影変換行列.

gg.cpp の 3082 行目に定義があります。

被呼び出し関係図:



8.7.3.15 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

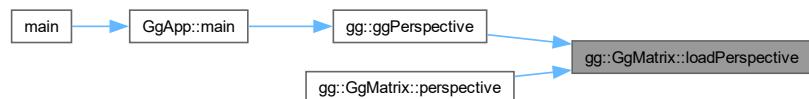
<i>fovy</i>	y 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 3164 行目に定義があります。

被呼び出し関係図:



8.7.3.16 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 <code>GgVector</code> 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 2562 行目に定義があります。

呼び出し関係図:



8.7.3.17 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 <code>GgVector</code> 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 2540 行目に定義があります。

呼び出し関係図:



8.7.3.18 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 2551 行目に定義があります。

呼び出し関係図:



8.7.3.19 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 2528 行目に定義があります。

呼び出し関係図:



8.7.3.20 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

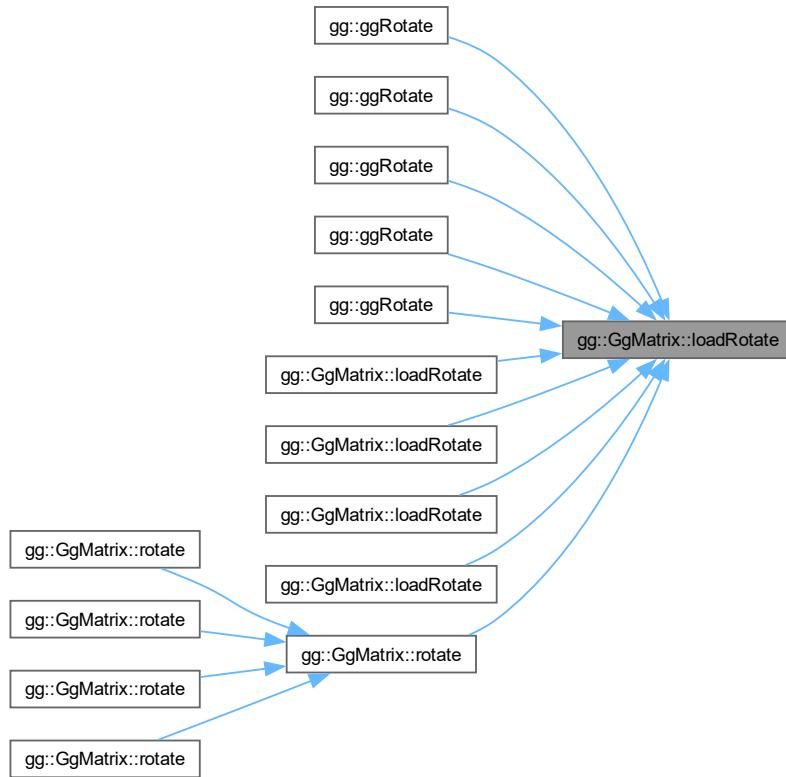
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

設定した変換行列.

[gg.cpp](#) の 2839 行目に定義があります。

被呼び出し関係図:



8.7.3.21 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

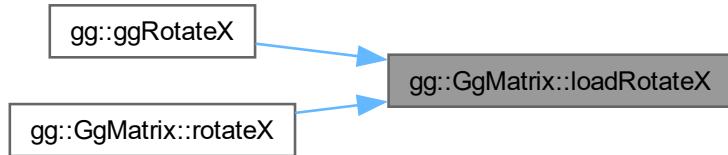
a	回転角.
---	------

戻り値

設定した変換行列。

`gg.cpp` の 2782 行目に定義があります。

被呼び出し関係図:



8.7.3.22 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

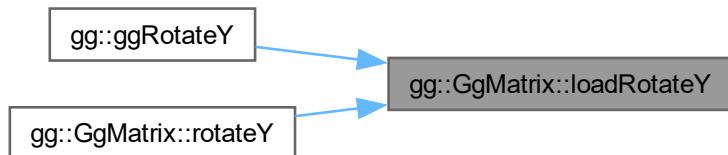
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2801 行目に定義があります。

被呼び出し関係図:



8.7.3.23 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

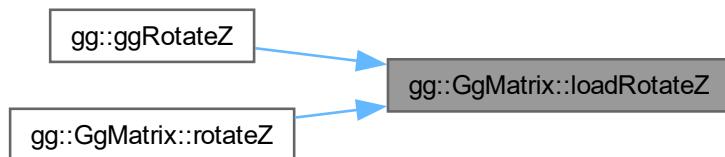
a	回転角.
---	------

戻り値

設定した変換行列.

gg.cpp の 2820 行目に定義があります。

呼び出し関係図:



8.7.3.24 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 2481 行目に定義があります。

呼び出し関係図:



8.7.3.25 `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

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

戻り値

設定した変換行列。

`gg.h` の 2470 行目に定義があります。

呼び出し関係図:



8.7.3.26 `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

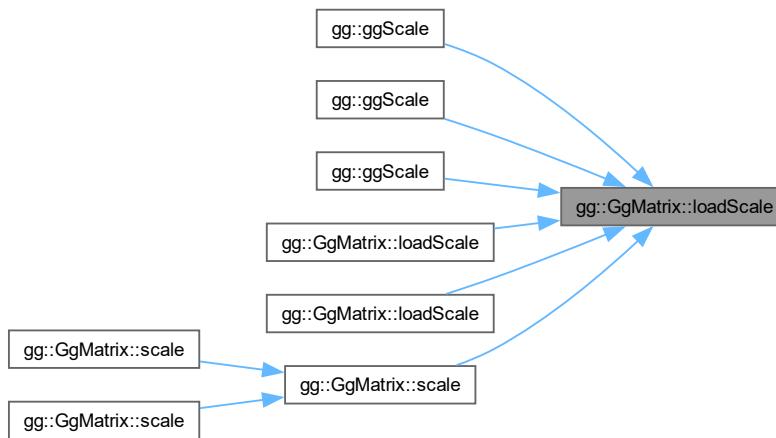
<code>x</code>	x 方向の拡大率。
<code>y</code>	y 方向の拡大率。
<code>z</code>	z 方向の拡大率。
<code>w</code>	w 拡大率のスケールファクタ (= 1.0f)。

戻り値

設定した変換行列.

`gg.cpp` の 2766 行目に定義があります。

被呼び出し関係図:



8.7.3.27 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 <code>GgVector</code> 型の変数.
----------------	----------------------------------

戻り値

設定した変換行列.

`gg.h` の 2448 行目に定義があります。

呼び出し関係図:



8.7.3.28 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

<i>t</i>	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 2437 行目に定義があります。

呼び出し関係図:



8.7.3.29 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

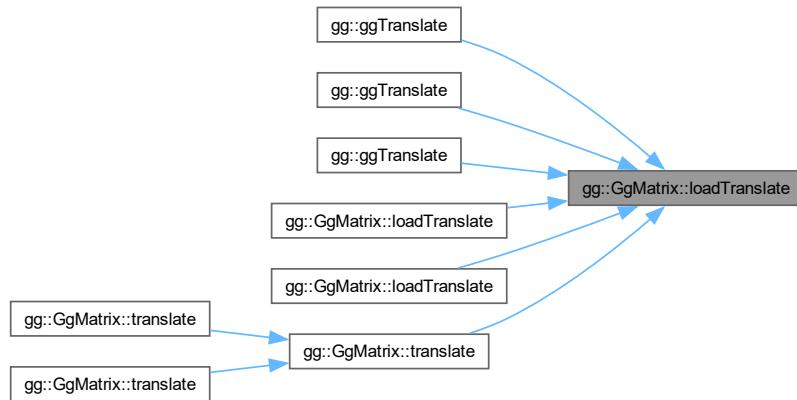
<i>x</i>	x 方向の移動量.
<i>y</i>	y 方向の移動量.
<i>z</i>	z 方向の移動量.
<i>w</i>	w 移動量のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2750 行目に定義があります。

呼び出し関係図:



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

戻り値

設定した `m` の転置行列.

gg.h の 2667 行目に定義があります。

呼び出し関係図:



8.7.3.31 `loadTranspose()` [2/2]

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

転置行列を格納する。

引数

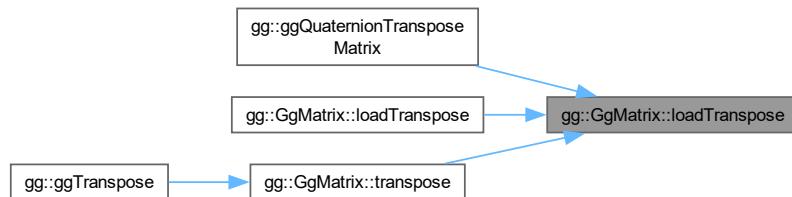
<i>a</i>	GLfloat 型の 16 要素の変換行列。
----------	------------------------

戻り値

設定した *a* の転置行列。

`gg.cpp` の 2881 行目に定義があります。

被呼び出し関係図:



8.7.3.32 `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

<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

ビュー変換行列を乗じた変換行列.

gg.h の 2926 行目に定義があります。

呼び出し関係図:



8.7.3.33 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 GLfloat 型の 3 要素の配列変数.
<i>t</i>	目標点の位置を格納した GLfloat 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 GLfloat 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 2913 行目に定義があります。

呼び出し関係図:



8.7.3.34 `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

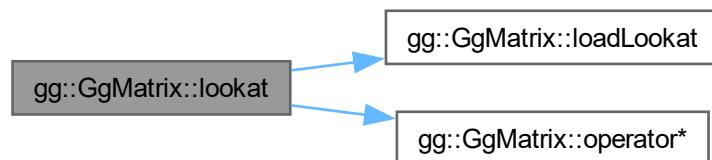
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

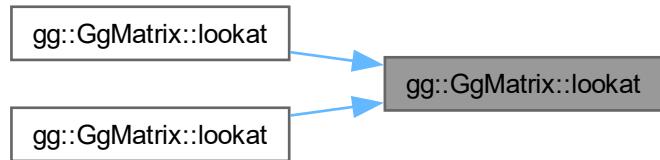
ビュー変換行列を乗じた変換行列.

`gg.h` の 2895 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:



8.7.3.35 normal()

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

法線変換行列を返す。

戻り値

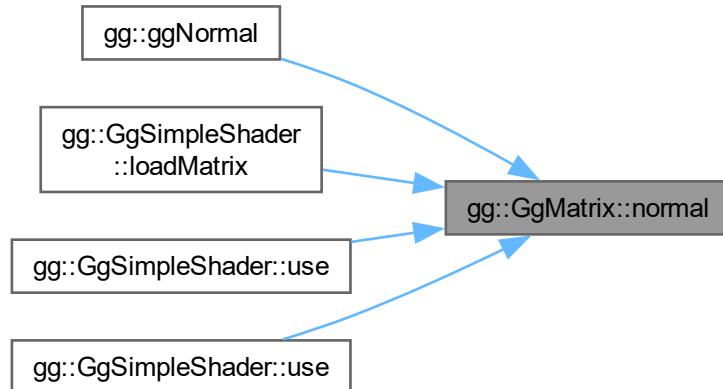
法線変換行列。

gg.h の 3018 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:



8.7.3.36 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 2339 行目に定義があります。

呼び出し関係図:



8.7.3.37 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

v	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 3074 行目に定義があります。

8.7.3.38 `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

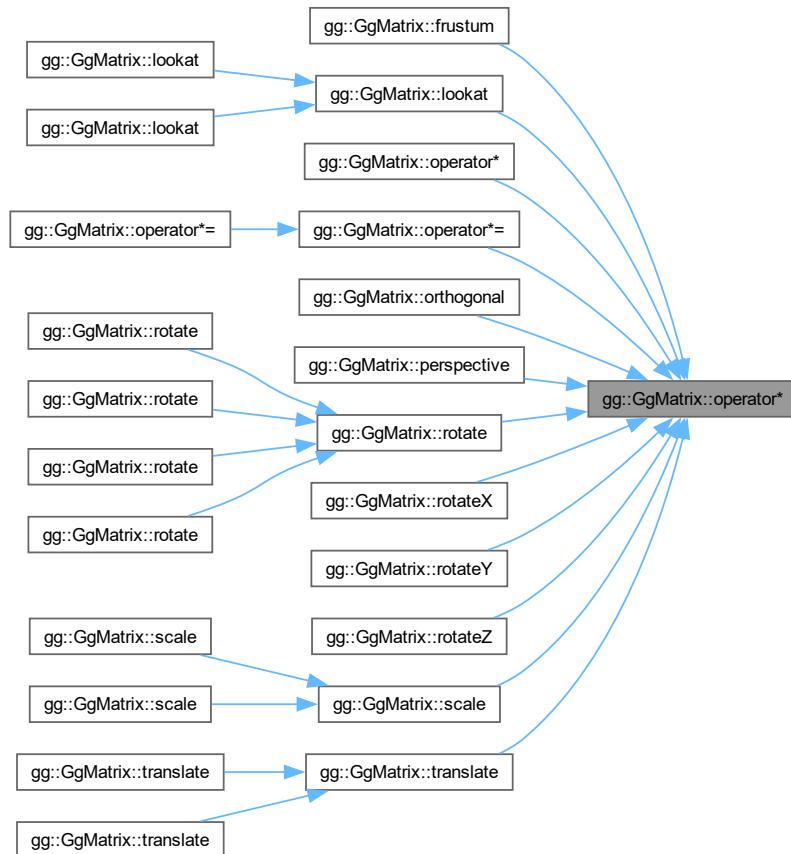
a	変換行列を格納した <code>GLfloat</code> 型の 16 要素の配列変数.
---	---

戻り値

変換行列に a を乗じた `GgMatrix` 型の変換行列.

`gg.h` の 2326 行目に定義があります。

被呼び出し関係図:



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

戻り値

`m` を掛けた変換行列の参照。

`gg.h` の 2362 行目に定義があります。

呼び出し関係図:



8.7.3.40 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

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 を掛けた変換行列の参照。

gg.h の 2350 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:



8.7.3.41 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

<code>m</code>	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 2245 行目に定義があります。

呼び出し関係図:



8.7.3.42 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

<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

変換行列に `a` を加えた GgMatrix 型の変換行列.

gg.h の 2232 行目に定義があります。

被呼び出し関係図:



8.7.3.43 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

<code>m</code>	<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 2268 行目に定義があります。

呼び出し関係図:



8.7.3.44 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

<code>a</code>	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

`a` を加えた変換行列の参照.

`gg.h` の 2256 行目に定義があります。

被呼び出し関係図:



8.7.3.45 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

<code>m</code>	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 2292 行目に定義があります。

呼び出し関係図:



8.7.3.46 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

<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

変換行列から `a` を引いた GgMatrix 型の変換行列.

gg.h の 2279 行目に定義があります。

被呼び出し関係図:



8.7.3.47 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 2315 行目に定義があります。

呼び出し関係図:



8.7.3.48 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 2303 行目に定義があります。

被呼び出し関係図:



8.7.3.49 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 2387 行目に定義があります。

呼び出し関係図:



8.7.3.50 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

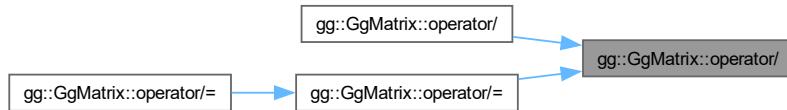
変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2373 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:



8.7.3.51 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

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

戻り値

`m` で割った変換行列の参照。

`gg.h` の 2410 行目に定義があります。

呼び出し関係図:



8.7.3.52 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

<code>a</code>	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

`a` で割った変換行列の参照.

`gg.h` の 2398 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:



8.7.3.53 operator=() [1/3]

```
GgMatrix & gg::GgMatrix::operator= (
    const GgMatrix & m) [default]
```

代入演算子.

8.7.3.54 operator=() [2/3]

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

配列変数の値を格納する.

引数

<code>a</code>	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 2220 行目に定義があります。

呼び出し関係図:



8.7.3.55 operator=() [3/3]

```
GgMatrix & gg::GgMatrix::operator= (
    GgMatrix && m) [default]
```

ムーブ代入演算子.

8.7.3.56 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

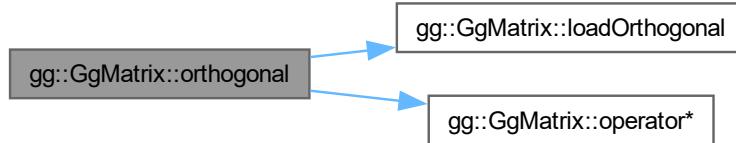
<code>left</code>	ウィンドウの左端の位置.
<code>right</code>	ウィンドウの右端の位置.
<code>bottom</code>	ウィンドウの下端の位置.
<code>top</code>	ウィンドウの上端の位置.
<code>zNear</code>	視点から前方面までの位置.
<code>zFar</code>	視点から後方面までの位置.

戻り値

直交投影変換行列を乗じた変換行列.

`gg.h` の 2942 行目に定義があります。

呼び出し関係図:



8.7.3.57 `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

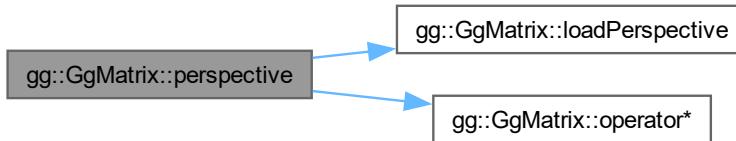
<i>fovy</i>	y 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

`gg.h` の 2982 行目に定義があります。

呼び出し関係図:



8.7.3.58 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 3063 行目に定義があります。

8.7.3.59 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 3052 行目に定義があります。

8.7.3.60 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 3041 行目に定義があります。

8.7.3.61 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 <code>GLfloat</code> 型の 4 要素の配列変数.
<i>v</i>	元のベクトルの <code>GLfloat</code> 型の 4 要素の配列変数.

`gg.h` の 3030 行目に定義がります。

8.7.3.62 `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 <code>GgVector</code> 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 2876 行目に定義がります。

呼び出し関係図:



8.7.3.63 `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

<i>r</i>	回転軸の方向ベクトルを格納した <code>GgVector</code> 型の変数.
<i>a</i>	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 2854 行目に定義があります。

呼び出し関係図:



8.7.3.64 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

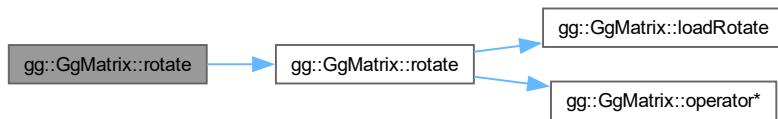
r	回転軸の方向ベクトルと回転角を格納した GLfloat 型の 4 要素の配列変数 (x, y, z, a).
-----	--

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 2865 行目に定義があります。

呼び出し関係図:



8.7.3.65 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

<i>r</i>	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 (<i>x</i> , <i>y</i> , <i>z</i>).
<i>a</i>	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 2842 行目に定義があります。

呼び出し関係図:

8.7.3.66 `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

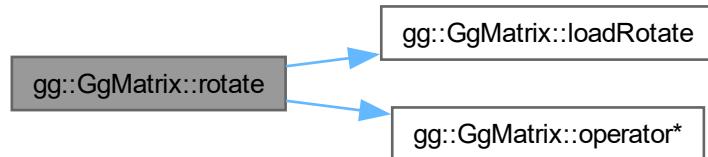
<i>x</i>	回転軸の <i>x</i> 成分.
<i>y</i>	回転軸の <i>y</i> 成分.
<i>z</i>	回転軸の <i>z</i> 成分.
<i>a</i>	回転角.

戻り値

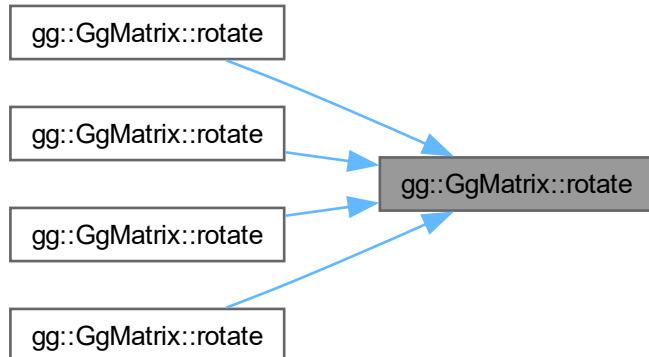
(x, y, z) を軸にさらに a 回転した変換行列.

gg.h の 2829 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:



8.7.3.67 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

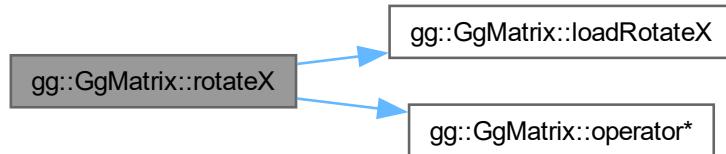
a	回転角.
---	------

戻り値

x 軸中心にさらに a 回転した変換行列。

gg.h の 2790 行目に定義があります。

呼び出し関係図:



8.7.3.68 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

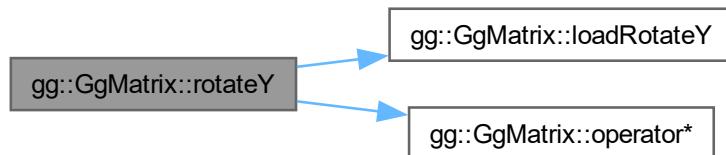
a	回転角。
-----	------

戻り値

y 軸中心にさらに a 回転した変換行列。

gg.h の 2802 行目に定義があります。

呼び出し関係図:



8.7.3.69 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

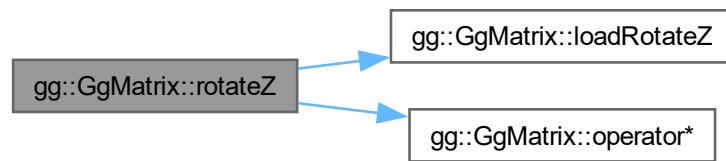
a	回転角.
---	------

戻り値

z 軸中心にさらに a 回転した変換行列.

gg.h の 2814 行目に定義があります。

呼び出し関係図:



8.7.3.70 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

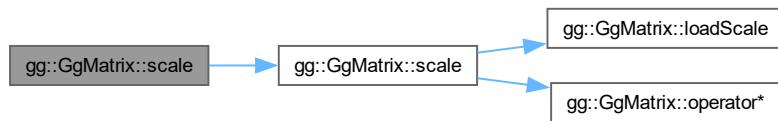
s	拡大率の GgVector 型の変数.
---	---------------------

戻り値

拡大縮小した結果の変換行列.

gg.h の 2779 行目に定義があります。

呼び出し関係図:



8.7.3.71 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

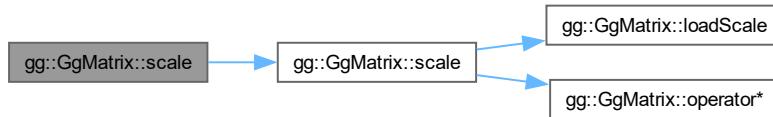
s	拡大率の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

拡大縮小した結果の変換行列.

gg.h の 2768 行目に定義があります。

呼び出し関係図:



8.7.3.72 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

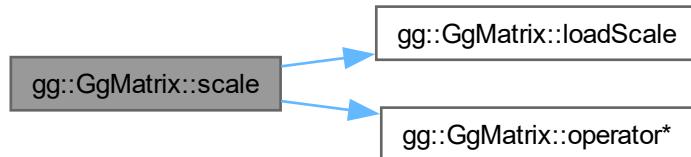
x	x 方向の拡大率.
y	y 方向の拡大率.
z	z 方向の拡大率.
w	w 移動量のスケールファクタ (= 1.0f).

戻り値

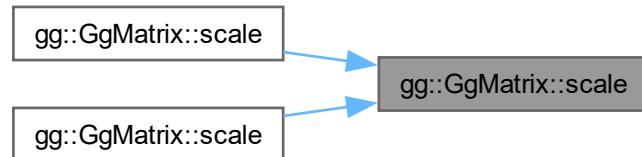
拡大縮小した結果の変換行列.

`gg.h` の 2756 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:



8.7.3.73 `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

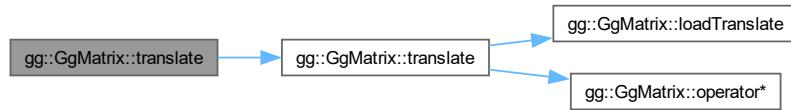
<code>t</code>	移動量の <code>GgVector</code> 型の変数.
----------------	----------------------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2742 行目に定義があります。

呼び出し関係図:



8.7.3.74 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

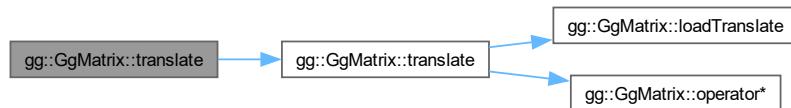
<i>t</i>	移動量の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2731 行目に定義があります。

呼び出し関係図:



8.7.3.75 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

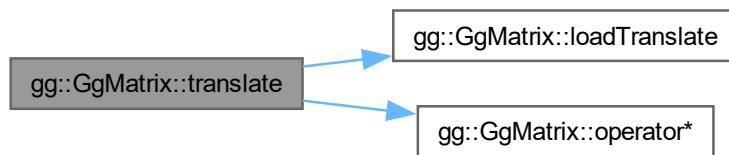
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

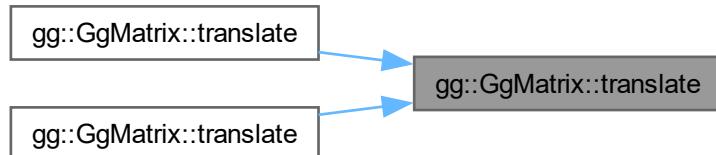
平行移動した結果の変換行列.

`gg.h` の 2719 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:



8.7.3.76 transpose()

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

転置行列を返す.

戻り値

転置行列。

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:



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](#)

8.8 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 \(const std::string &name, GLfloat nz=1.0f, GLenum internal=GL_RGBA\)](#)

8.8.1 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 5409 行目に定義があります。

8.8.2 構築子と解体子

8.8.2.1 GgNormalTexture() [1/3]

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

コンストラクタ.

[gg.h](#) の 5419 行目に定義があります。

8.8.2.2 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 5433 行目に定義があります。

呼び出し関係図:



8.8.2.3 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 5453 行目に定義があります。

呼び出し関係図:



8.8.2.4 ~GgNormalTexture()

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

デストラクタ.

gg.h の 5466 行目に定義があります。

8.8.3 関数詳解

8.8.3.1 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

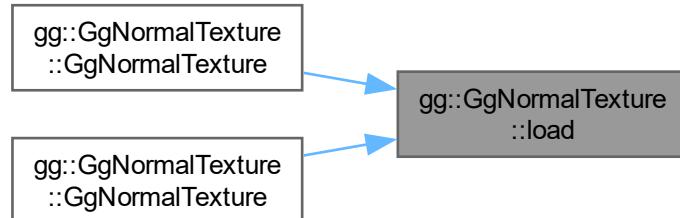
<i>hmap</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット (<code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code> , <code>GL_RGBA</code>).
<i>nz</i>	法線マップの <i>z</i> 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

[gg.h](#) の 5480 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:



8.8.3.2 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.

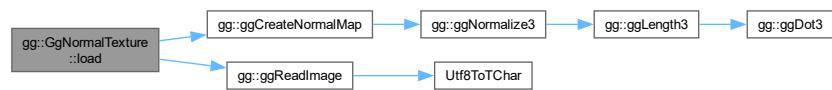
引数

<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 <i>z</i> 成分の値.

<i>internal</i>	テクスチャの内部フォーマット。
-----------------	-----------------

gg.cpp の 4082 行目に定義があります。

呼び出し関係図:



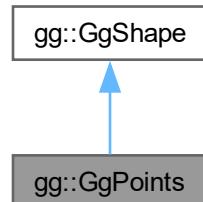
このクラス詳解は次のファイルから抽出されました:

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

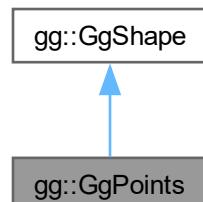
8.9 gg::GgPoints クラス

#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 countv, 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`

基底クラス `gg::GgShape` に属する継承公開メンバ関数

- `GgShape (GLenum mode=0)`
- `virtual ~GgShape ()`
- `const GLuint & get () const`
- `void setMode (GLenum mode)`
- `const GLenum & getMode () const`

8.9.1 詳解

点。

`gg.h` の 6262 行目に定義がります。

8.9.2 構築子と解体子

8.9.2.1 `GgPoints()` [1/2]

```
gg::GgPoints::GgPoints (
    GLenum mode = GL_POINTS) [inline]
```

コンストラクタ。

`gg.h` の 6273 行目に定義がります。

8.9.2.2 `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 6286 行目に定義があります。

8.9.2.3 ~GgPoints()

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

デストラクタ.

gg.h の 6300 行目に定義があります。

8.9.3 関数詳解

8.9.3.1 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 5158 行目に定義があります。

呼び出し関係図:



8.9.3.2 `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 6339 行目に定義があります。

8.9.3.3 `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 6329 行目に定義があります。

8.9.3.4 `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

<code>pos</code>	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.
<code>count</code>	頂点のデータの数 (頂点数).
<code>usage</code>	バッファオブジェクトの使い方.

`gg.cpp` の 5145 行目に定義があります。

8.9.3.5 `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 6309 行目に定義があります。

8.9.3.6 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 6319 行目に定義があります。

8.9.3.7 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

<code>pos</code>	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.
<code>first</code>	転送先のバッファオブジェクトの先頭の要素番号.
<code>count</code>	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 6351 行目に定義があります。

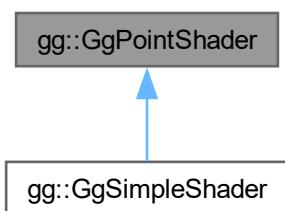
このクラス詳解は次のファイルから抽出されました:

- `gg.h`
- `gg.cpp`

8.10 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`

8.10.1 詳解

点のシェーダ.

`gg.h` の 6896 行目に定義があります。

8.10.2 構築子と解体子

8.10.2.1 `GgPointShader()` [1/3]

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

コンストラクタ.

`gg.h` の 6912 行目に定義があります。

8.10.2.2 `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>	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 6927 行目に定義があります。

8.10.2.3 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 6946 行目に定義があります。

8.10.2.4 ~GgPointShader()

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

デストラクタ.

gg.h の 6958 行目に定義があります。

8.10.3 関数詳解

8.10.3.1 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 7142 行目に定義があります。

8.10.3.2 `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 7005 行目に定義があります。

8.10.3.3 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

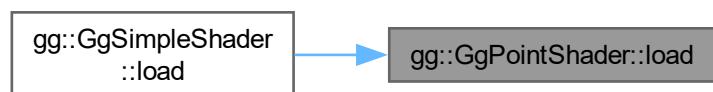
<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

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 6972 行目に定義があります。

被呼び出し関係図:



8.10.3.4 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 7072 行目に定義があります。

呼び出し関係図:



8.10.3.5 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 7060 行目に定義があります。

8.10.3.6 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。

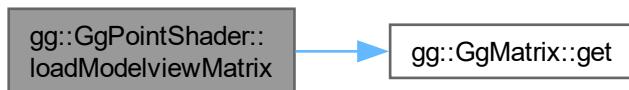
引数

<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 7049 行目に定義があります。

呼び出し関係図:



8.10.3.7 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 7039 行目に定義があります。

8.10.3.8 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 7029 行目に定義がります。

呼び出し関係図:



8.10.3.9 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 7019 行目に定義がります。

8.10.3.10 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 7132 行目に定義がります。

8.10.3.11 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 7080 行目に定義がります。

8.10.3.12 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>	GgMatrix 型の投影変換行列.
-----------	--------------------

gg.h の 7101 行目に定義があります。

呼び出し関係図:



8.10.3.13 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>	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.h の 7124 行目に定義があります。

呼び出し関係図:



8.10.3.14 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 7090 行目に定義があります。

8.10.3.15 `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 7112 行目に定義があります。

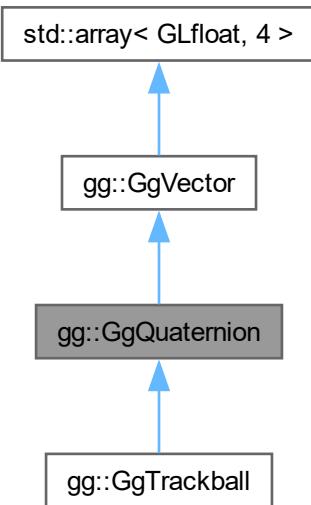
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

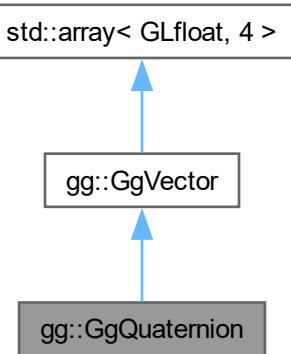
8.11 gg::GgQuaternion クラス

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

gg::GgQuaternion の継承関係図



gg::GgQuaternion 連携図



公開 メンバ関数

- `GgQuaternion ()=default`
- `constexpr GgQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `constexpr GgQuaternion (GLfloat c)`
- `GgQuaternion (const GLfloat *a)`
- `GgQuaternion (const GgVector &v)`
- `GgQuaternion (const GgQuaternion &q)=default`
- `GgQuaternion (GgQuaternion &&q)=default`
- `~GgQuaternion ()=default`
- `GgQuaternion & operator= (const GgQuaternion &q)=default`
- `GgQuaternion & operator= (GgQuaternion &&q)=default`
- `GLfloat norm () const`
- `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`

基底クラス `gg::GgVector` に属する継承公開メンバ関数

- `GgVector ()=default`
- `constexpr GgVector (GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3)`
- `constexpr GgVector (const GLfloat *a)`
- `constexpr GgVector (GLfloat c)`
- `GgVector (const GgVector &v)=default`
- `GgVector (GgVector &&v)=default`
- `~GgVector ()=default`
- `GgVector & operator= (const GgVector &v)=default`
- `GgVector & operator= (GgVector &&v)=default`
- `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/ (GLfloat c) const`
- `GgVector & operator/= (GgVector &v)`
- `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

8.11.1 詳解

四元数.

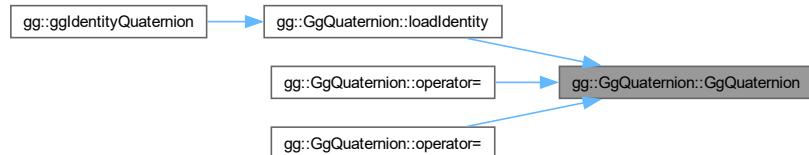
`gg.h` の 3446 行目に定義があります。

8.11.2 構築子と解体子

8.11.2.1 `GgQuaternion()` [1/7]

`gg::GgQuaternion::GgQuaternion ()` [default]

コンストラクタ. 被呼び出し関係図:



8.11.2.2 `GgQuaternion()` [2/7]

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

コンストラクタ.

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.

`gg.h` の 3475 行目に定義があります。

8.11.2.3 GgQuaternion() [3/7]

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

コンストラクタ.

引数

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

gg.h の 3485 行目に定義があります。

8.11.2.4 GgQuaternion() [4/7]

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

コンストラクタ.

引数

<code>a</code>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------------	--------------------------------

gg.h の 3495 行目に定義があります。

8.11.2.5 GgQuaternion() [5/7]

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

コンストラクタ.

引数

<code>v</code>	四元数を格納した GgVector 型の変数.
----------------	-------------------------

gg.h の 3505 行目に定義があります。

8.11.2.6 GgQuaternion() [6/7]

```
gg::GgQuaternion::GgQuaternion (
    const GgQuaternion & q)  [default]
```

コピーコンストラクタ.

8.11.2.7 GgQuaternion() [7/7]

```
gg::GgQuaternion::GgQuaternion (
    GgQuaternion && q)  [default]
```

ムーブコンストラクタ.

8.11.2.8 ~GgQuaternion()

```
gg::GgQuaternion::~GgQuaternion () [default]
```

デストラクタ.

8.11.3 関数詳解

8.11.3.1 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

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

戻り値

q を加えた四元数.

gg.h の 3788 行目に定義があります。

呼び出し関係図:



8.11.3.2 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

<i>v</i>	四元数を格納した GgVector 型の変数.
----------	-------------------------

戻り値

\mathbf{v} を加えた四元数.

`gg.h` の 3777 行目に定義があります。

呼び出し関係図:



8.11.3.3 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

\mathbf{a} を加えた四元数.

`gg.h` の 3766 行目に定義があります。

呼び出し関係図:



8.11.3.4 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 <i>x</i> 要素.
<i>y</i>	加える四元数の <i>y</i> 要素.
<i>z</i>	加える四元数の <i>z</i> 要素.
<i>w</i>	加える四元数の <i>w</i> 要素.

戻り値

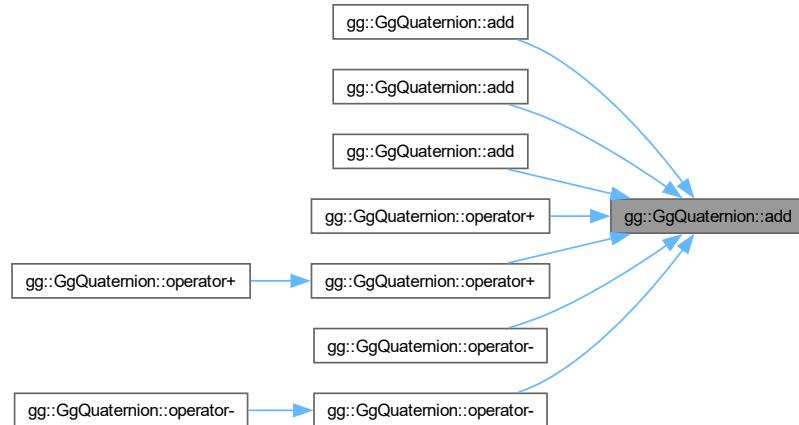
(*x*, *y*, *z*, *w*) を加えた四元数.

gg.h の 3754 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:



8.11.3.5 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 4411 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:



8.11.3.6 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

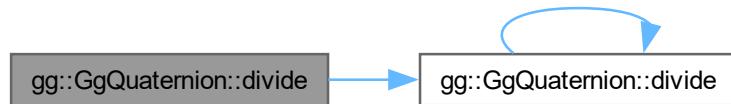
<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

`q` で割った四元数.

`gg.h` の 3937 行目に定義があります。

呼び出し関係図:



8.11.3.7 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

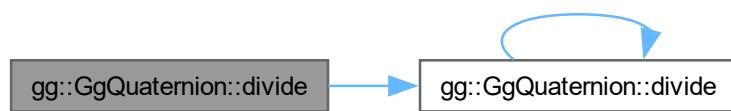
v	四元数を格納した GgVector 型の変数.
---	-------------------------

戻り値

v で割った四元数.

gg.h の 3926 行目に定義があります。

呼び出し関係図:



8.11.3.8 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

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 3912 行目に定義があります。

呼び出し関係図:



8.11.3.9 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

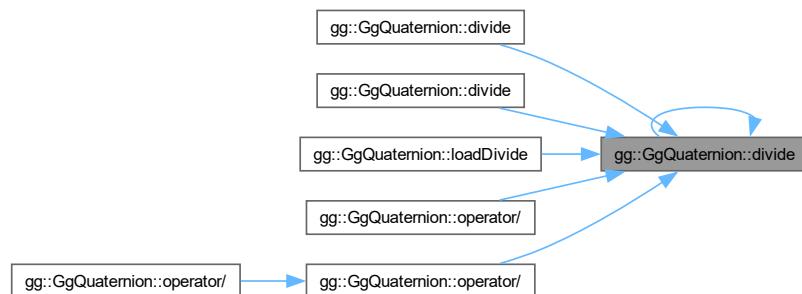
(*x*, *y*, *z*, *w*) を割った四元数.

gg.h の 3900 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:



8.11.3.10 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

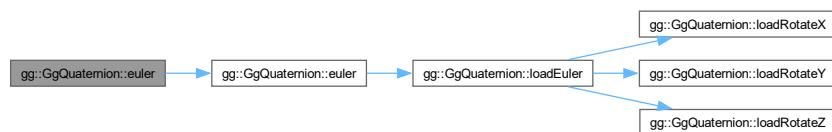
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転した四元数.

gg.h の 4251 行目に定義があります。

呼び出し関係図:



8.11.3.11 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 4239 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:



8.11.3.12 `get()`

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

四元数を取り出す.

引数

<code>a</code>	四元数を格納する <code>GLfloat</code> 型の 4 要素の配列変数.
----------------	---

gg.h の 4435 行目に定義があります。

8.11.3.13 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 4502 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:



8.11.3.14 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

<i>m</i>	回転の変換行列を格納する GgMatrix 型の変数。
----------	-----------------------------

[gg.h](#) の 4492 行目に定義があります。

呼び出し関係図:



8.11.3.15 getConjugateMatrix() [3/3]

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

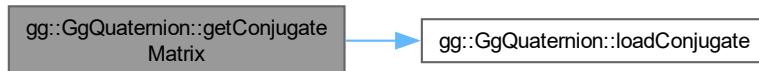
四元数の共役が表す回転の変換行列を a に求める。

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 4480 行目に定義があります。

呼び出し関係図:



8.11.3.16 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

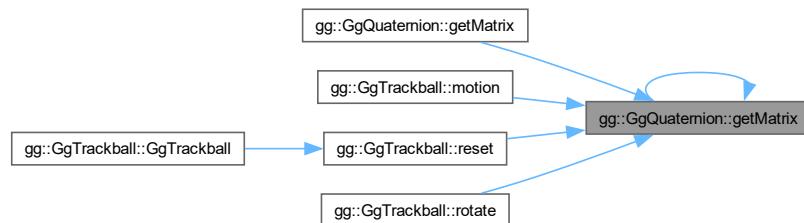
回転の変換を表す [GgMatrix](#) 型の変換行列.

[gg.h](#) の 4468 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:



8.11.3.17 `getMatrix()` [2/3]

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

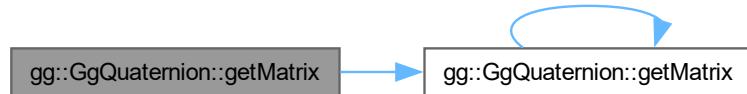
四元数が表す回転の変換行列を `m` に求める.

引数

<code>m</code>	回転の変換行列を格納する GgMatrix 型の変数.
----------------	---

[gg.h](#) の 4458 行目に定義があります。

呼び出し関係図:



8.11.3.18 getMatrix() [3/3]

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

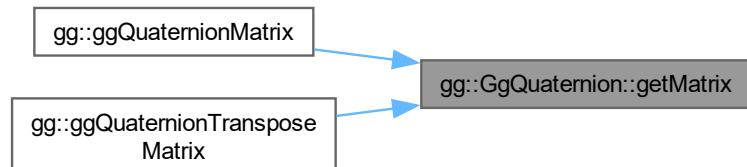
四元数が表す回転の変換行列を *a* に求める。

引数

<i>a</i>	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 4448 行目に定義があります。

被呼び出し関係図:



8.11.3.19 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 4423 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:



8.11.3.20 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

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

戻り値

q を加えた四元数.

gg.h の 3592 行目に定義があります。

呼び出し関係図:



8.11.3.21 `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 <code>GgVector</code> 型の変数。
----------------	--------------------------------------

戻り値

`v` を加えた四元数。

`gg.h` の 3581 行目に定義があります。

呼び出し関係図:



8.11.3.22 `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

<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` を加えた四元数。

`gg.h` の 3570 行目に定義があります。

呼び出し関係図:



8.11.3.23 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

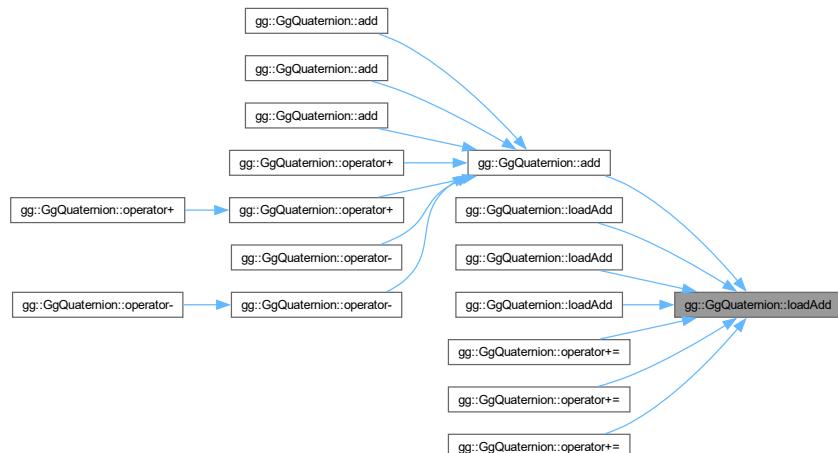
<i>x</i>	加える四元数の <i>x</i> 要素.
<i>y</i>	加える四元数の <i>y</i> 要素.
<i>z</i>	加える四元数の <i>z</i> 要素.
<i>w</i>	加える四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を加えた四元数.

gg.h の 3554 行目に定義があります。

被呼び出し関係図:



8.11.3.24 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>	GgQuaternion 型の四元数.
----------------	---------------------

戻り値

共役四元数.

gg.h の 4342 行目に定義があります。

呼び出し関係図:



8.11.3.25 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

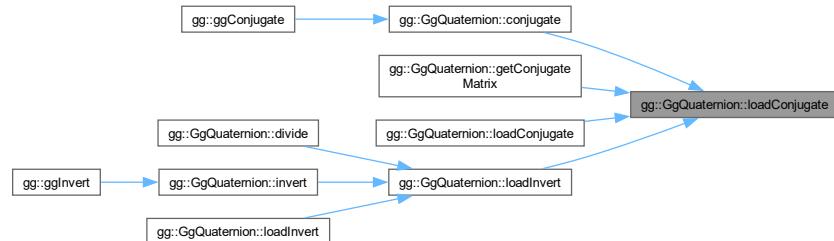
<code>a</code>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------------	--------------------------------

戻り値

共役四元数.

gg.cpp の 3378 行目に定義があります。

被呼び出し関係図:



8.11.3.26 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

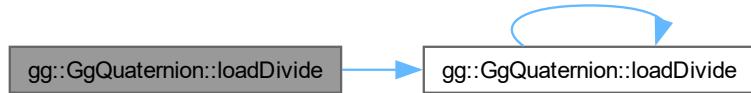
<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

`q` で割った四元数.

`gg.h` の 3740 行目に定義があります。

呼び出し関係図:



8.11.3.27 `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

<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 3729 行目に定義があります。

呼び出し関係図:



8.11.3.28 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 3718 行目に定義があります。

呼び出し関係図:



8.11.3.29 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

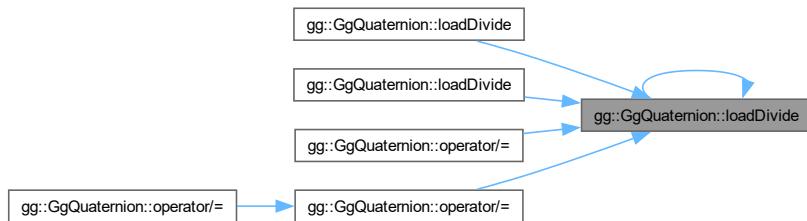
(x, y, z, w) を割った四元数.

gg.h の 3706 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:



8.11.3.30 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

e	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 4226 行目に定義があります。

呼び出し関係図:



8.11.3.31 `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

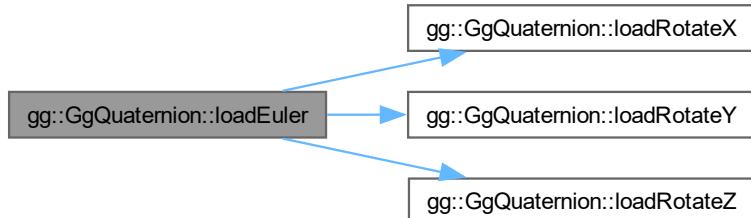
<i>heading</i>	y 軸中心の回転角。
<i>pitch</i>	x 軸中心の回転角。
<i>roll</i>	z 軸中心の回転角。

戻り値

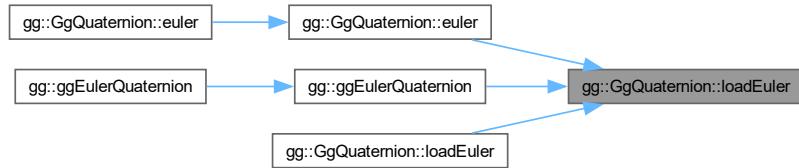
格納した回転を表す四元数。

`gg.cpp` の 3350 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:



8.11.3.32 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 4076 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:



8.11.3.33 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

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

戻り値

四元数の逆元.

`gg.h` の 4361 行目に定義があります。

呼び出し関係図:



8.11.3.34 `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

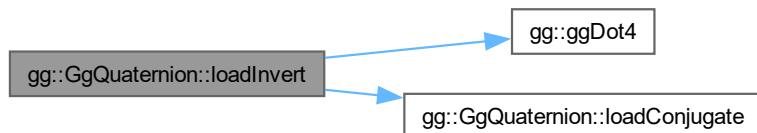
<code>a</code>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
----------------	---

戻り値

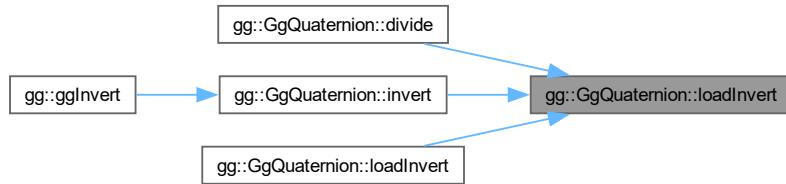
四元数の逆元.

`gg.cpp` の 3392 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:



8.11.3.35 `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

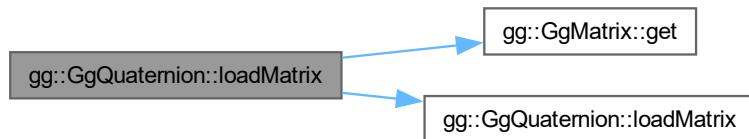
<code>m</code>	<code>Ggmatrix</code> 型の変換行列。
----------------	-------------------------------

戻り値

`m` による回転の変換に相当する四元数。

`gg.h` の 4066 行目に定義があります。

呼び出し関係図:



8.11.3.36 `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

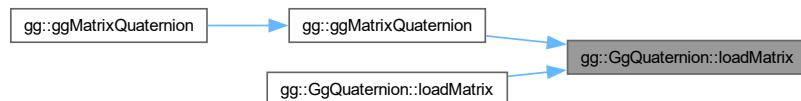
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 4054 行目に定義があります。

呼び出し関係図:



8.11.3.37 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 3692 行目に定義があります。

呼び出し関係図:



8.11.3.38 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

<code>v</code>	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 3681 行目に定義があります。

呼び出し関係図:



8.11.3.39 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

<code>a</code>	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 3670 行目に定義があります。

8.11.3.40 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

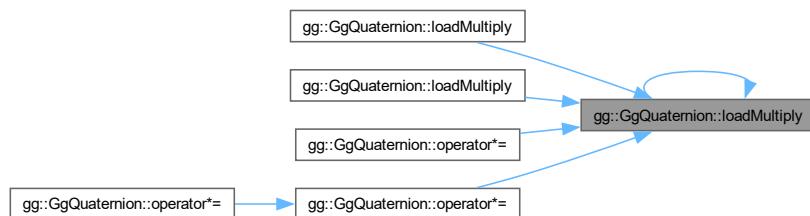
(x, y, z, w) を掛けた四元数.

gg.h の 3658 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:



8.11.3.41 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

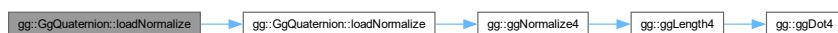
<code>q</code>	GgQuaternion 型の四元数.
----------------	---------------------

戻り値

正規化された四元数.

gg.h の 4323 行目に定義があります。

呼び出し関係図:



8.11.3.42 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

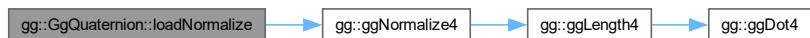
<code>a</code>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------------	--------------------------------

戻り値

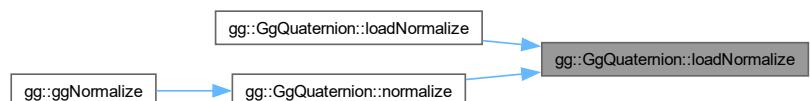
正規化された四元数.

gg.cpp の 3366 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:



8.11.3.43 `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 4110 行目に定義があります。

呼び出し関係図:



8.11.3.44 `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

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

戻り値

格納された回転を表す四元数。

`gg.h` の 4099 行目に定義があります。

呼び出し関係図:



8.11.3.45 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

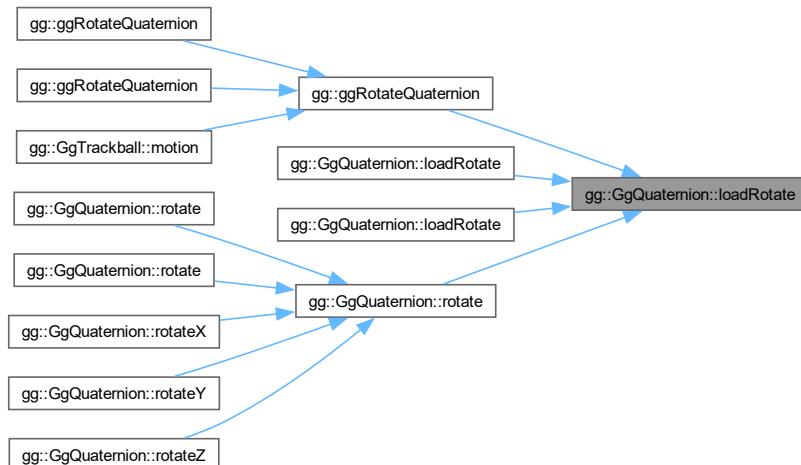
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

格納された回転を表す四元数。

gg.cpp の 3292 行目に定義があります。

被呼び出し関係図:



8.11.3.46 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

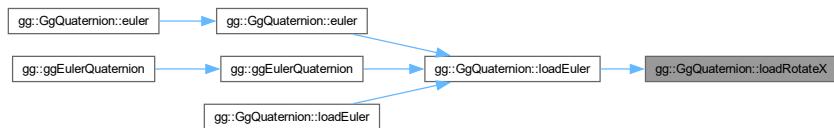
<i>a</i>	回転角.
----------	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3314 行目に定義があります。

被呼び出し関係図:



8.11.3.47 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

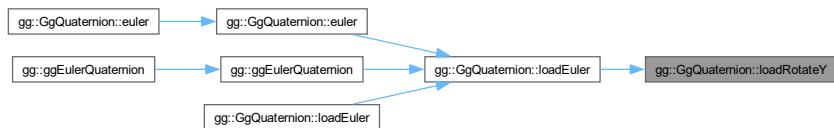
<i>a</i>	回転角.
----------	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3326 行目に定義があります。

被呼び出し関係図:



8.11.3.48 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

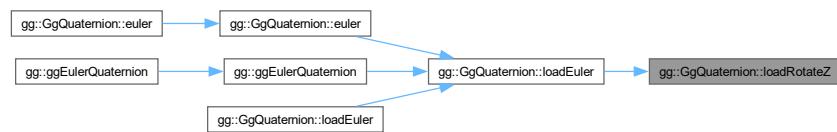
a	回転角.
---	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3338 行目に定義があります。

呼び出し関係図:



8.11.3.49 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 4278 行目に定義があります。

呼び出し関係図:



8.11.3.50 `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>	<code>GgQuaternion</code> 型の四元数。
<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

格納した *q*, *a* を *t* で内分した四元数。

`gg.h` の 4291 行目に定義があります。

呼び出し関係図:



8.11.3.51 `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 <code>GLfloat</code> 型の 4 要素の配列変数。
<i>q</i>	<code>GgQuaternion</code> 型の四元数。
<i>t</i>	補間パラメータ。

戻り値

格納した a, q を t で内分した四元数.

gg.h の 4304 行目に定義があります。

呼び出し関係図:



8.11.3.52 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

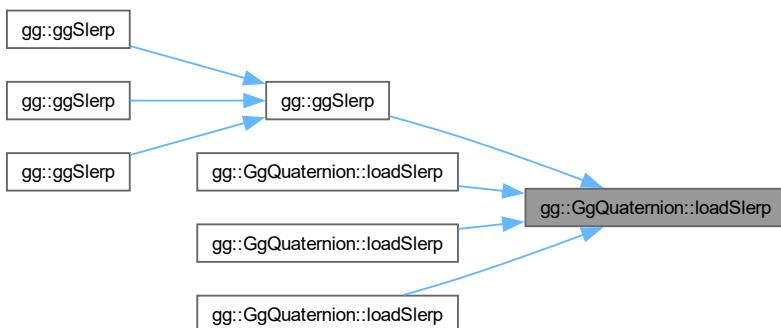
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した a, b を t で内分した四元数.

gg.h の 4264 行目に定義があります。

被呼び出し関係図:



8.11.3.53 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

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

戻り値

q を引いた四元数。

gg.h の 3644 行目に定義があります。

呼び出し関係図:



8.11.3.54 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

<i>v</i>	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 3633 行目に定義があります。

呼び出し関係図:



8.11.3.55 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

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 3622 行目に定義があります。

呼び出し関係図:



8.11.3.56 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

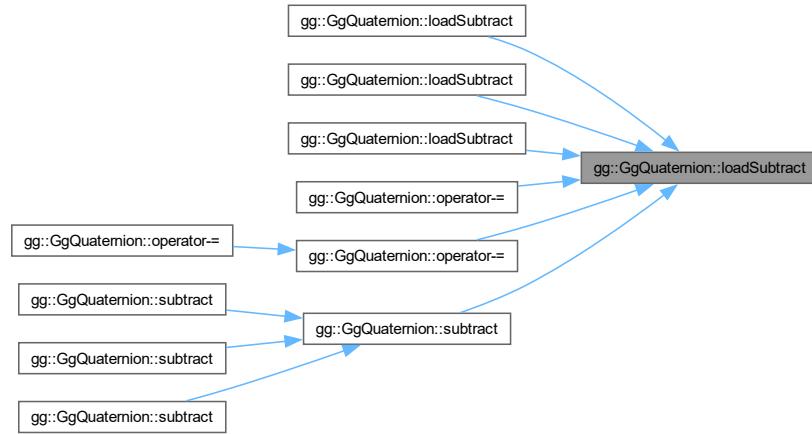
x	引く四元数の x 要素.
y	引く四元数の y 要素.
z	引く四元数の z 要素.
w	引く四元数の w 要素.

戻り値

(x, y, z, w) を引いた四元数.

gg.h の 3606 行目に定義があります。

被呼び出し関係図:



8.11.3.57 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 3886 行目に定義があります。

8.11.3.58 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

v	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 3875 行目に定義がります。

8.11.3.59 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a を掛けた四元数.

gg.h の 3862 行目に定義がります。

8.11.3.60 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 3850 行目に定義がります。

8.11.3.61 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 3540 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:



8.11.3.62 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 4399 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:



8.11.3.63 operator*() [1/3]

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

gg.h の 4031 行目に定義があります。

呼び出し関係図:



8.11.3.64 operator*() [2/3]

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

gg.h の 4027 行目に定義があります。

8.11.3.65 operator*() [3/3]

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

gg.h の 4023 行目に定義があります。

被呼び出し関係図:



8.11.3.66 operator*=(()) [1/3]

```
GgQuaternion & gg::GgQuaternion::operator*=
    (const GgQuaternion & q) [inline]
```

gg.h の 3983 行目に定義があります。

呼び出し関係図:

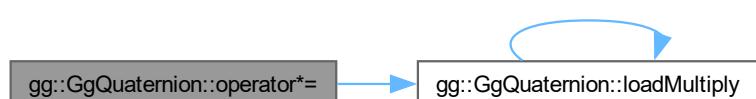


8.11.3.67 operator*=(()) [2/3]

```
GgQuaternion & gg::GgQuaternion::operator*=
    (const GgVector & v) [inline]
```

gg.h の 3979 行目に定義があります。

呼び出し関係図:

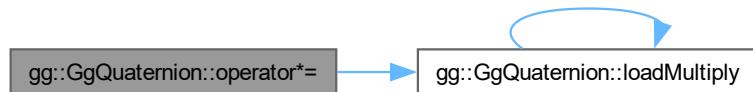


8.11.3.68 operator*=() [3/3]

```
GgQuaternion & gg::GgQuaternion::operator*=(  
    const GLfloat * a) [inline]
```

gg.h の 3975 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:

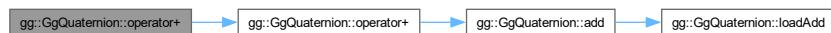


8.11.3.69 operator+() [1/3]

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

gg.h の 4007 行目に定義があります。

呼び出し関係図:



8.11.3.70 operator+() [2/3]

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

gg.h の 4003 行目に定義があります。

呼び出し関係図:



8.11.3.71 operator+() [3/3]

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

gg.h の 3999 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:



8.11.3.72 operator+=() [1/3]

```
GgQuaternion & gg::GgQuaternion::operator+= (const GgQuaternion & q) [inline]
```

gg.h の 3959 行目に定義がります。

呼び出し関係図:

**8.11.3.73 operator+=() [2/3]**

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

gg.h の 3955 行目に定義がります。

呼び出し関係図:

**8.11.3.74 operator+=() [3/3]**

```
GgQuaternion & gg::GgQuaternion::operator+= (const GLfloat * a) [inline]
```

gg.h の 3951 行目に定義がります。

呼び出し関係図:



8.11.3.75 operator-() [1/3]

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

gg.h の 4019 行目に定義があります。

呼び出し関係図:



8.11.3.76 operator-() [2/3]

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

gg.h の 4015 行目に定義があります。

呼び出し関係図:



8.11.3.77 operator-() [3/3]

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

gg.h の 4011 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:

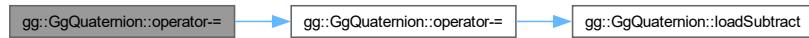


8.11.3.78 operator=() [1/3]

```
GgQuaternion & gg::GgQuaternion::operator-= (const GgQuaternion & q) [inline]
```

gg.h の 3971 行目に定義がります。

呼び出し関係図:

**8.11.3.79 operator=() [2/3]**

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

gg.h の 3967 行目に定義がります。

呼び出し関係図:

**8.11.3.80 operator=() [3/3]**

```
GgQuaternion & gg::GgQuaternion::operator-= (const GLfloat * a) [inline]
```

gg.h の 3963 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:

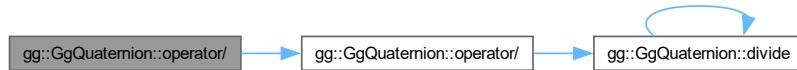


8.11.3.81 operator/() [1/3]

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

gg.h の 4043 行目に定義があります。

呼び出し関係図:



8.11.3.82 operator/() [2/3]

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

gg.h の 4039 行目に定義があります。

呼び出し関係図:



8.11.3.83 operator/() [3/3]

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

gg.h の 4035 行目に定義があります。

呼び出し関係図:



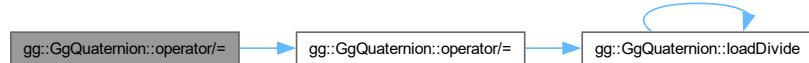
被呼び出し関係図:

**8.11.3.84 operator/() [1/3]**

```
GgQuaternion & gg::GgQuaternion::operator/= (
    const GgQuaternion & q) [inline]
```

gg.h の 3995 行目に定義があります。

呼び出し関係図:



8.11.3.85 operator/() [2/3]

```
GgQuaternion & gg::GgQuaternion::operator/= (
    const GgVector & v) [inline]
```

gg.h の 3991 行目に定義がります。

呼び出し関係図:



8.11.3.86 operator/() [3/3]

```
GgQuaternion & gg::GgQuaternion::operator/= (
    const GLfloat * a) [inline]
```

gg.h の 3987 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.87 operator=() [1/4]

```
GgQuaternion & gg::GgQuaternion::operator= (
    const GgQuaternion & q)  [default]
```

代入演算子. 被呼び出し関係図:

**8.11.3.88 operator=() [2/4]**

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

gg.h の 3947 行目に定義がります。

呼び出し関係図:

**8.11.3.89 operator=() [3/4]**

```
GgQuaternion & gg::GgQuaternion::operator= (
    const GLfloat * a)  [inline]
```

gg.h の 3943 行目に定義がります。

呼び出し関係図:



8.11.3.90 operator=() [4/4]

```
GgQuaternion & gg::GgQuaternion::operator= (
    GgQuaternion && q) [default]
```

ムーブ代入演算子。

8.11.3.91 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

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

戻り値

回転した四元数。

gg.h の 4172 行目に定義があります。

呼び出し関係図:



8.11.3.92 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 GLfloat 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 4161 行目に定義があります。

呼び出し関係図:



8.11.3.93 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

<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

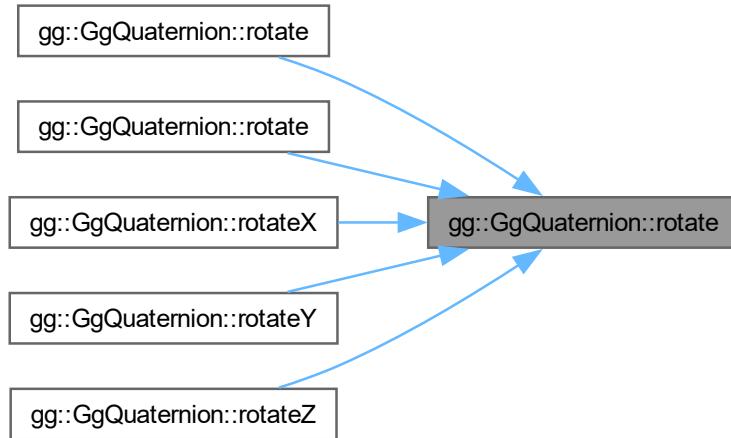
回転した四元数.

gg.h の 4148 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:



8.11.3.94 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 4183 行目に定義があります。

呼び出し関係図:



8.11.3.95 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 4194 行目に定義があります。

呼び出し関係図:



8.11.3.96 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 4205 行目に定義があります。

呼び出し関係図:



8.11.3.97 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>	GgQuaternion 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 4387 行目に定義があります。

8.11.3.98 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 GLfloat 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 4373 行目に定義があります。

8.11.3.99 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>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

q を引いた四元数.

gg.h の 3836 行目に定義があります。

呼び出し関係図:



8.11.3.100 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

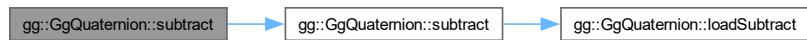
v	四元数を格納した GgVector 型の変数.
---	-------------------------

戻り値

v を引いた四元数.

gg.h の 3825 行目に定義があります。

呼び出し関係図:



8.11.3.101 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

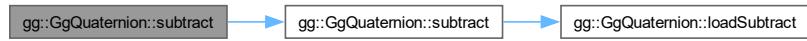
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a を引いた四元数.

gg.h の 3814 行目に定義があります。

呼び出し関係図:



8.11.3.102 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

<i>x</i>	引く四元数の <i>x</i> 要素.
<i>y</i>	引く四元数の <i>y</i> 要素.
<i>z</i>	引く四元数の <i>z</i> 要素.
<i>w</i>	引く四元数の <i>w</i> 要素.

戻り値

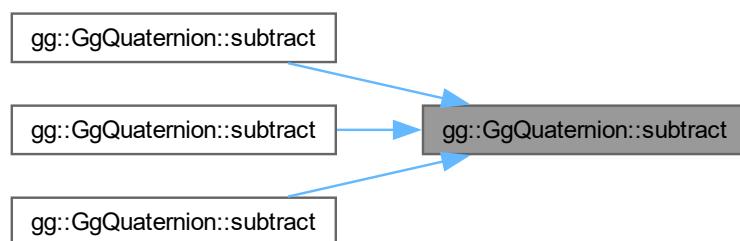
(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 3802 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](#)

8.12 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 &shader)=delete`
- `GgShader (GgShader &&shader)=default`
- `virtual ~GgShader ()`
- `GgShader & operator= (const GgShader &shader)=delete`
- `GgShader & operator= (GgShader &&shader)=default`
- `void use () const`
- `void unuse () const`
- `GLuint get () const`

8.12.1 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 6793 行目に定義があります。

8.12.2 構築子と解体子

8.12.2.1 GgShader() [1/4]

```
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

<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)。
<code>geom</code>	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)。
<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。

gg.h の 6809 行目に定義があります。

呼び出し関係図:



8.12.2.2 GgShader() [2/4]

```
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 6827 行目に定義があります。

8.12.2.3 GgShader() [3/4]

```
gg::GgShader::GgShader (
    const GgShader & shader) [delete]
```

コピー構造体は使用しない.

8.12.2.4 GgShader() [4/4]

```
gg::GgShader::GgShader (
    GgShader && shader) [default]
```

ムーブ構造体はデフォルトのものを使用する.

8.12.2.5 ~GgShader()

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

デストラクタ。

gg.h の 6849 行目に定義がります。

8.12.3 関数詳解

8.12.3.1 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 6887 行目に定義がります。

8.12.3.2 operator=() [1/2]

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

代入演算子は使用しない。

8.12.3.3 operator=() [2/2]

```
GgShader & gg::GgShader::operator= (
    GgShader && shader) [default]
```

ムーブ代入演算子はデフォルトのものを使用する。

8.12.3.4 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 6877 行目に定義がります。

8.12.3.5 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 6869 行目に定義があります。

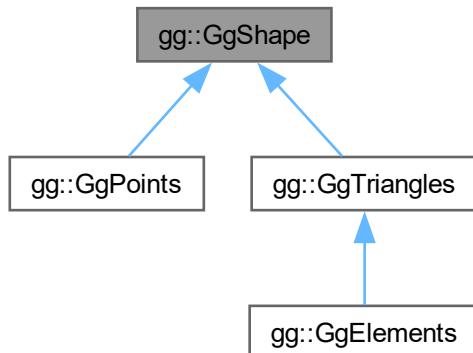
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.13 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

8.13.1 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 6169 行目に定義があります。

8.13.2 構築子と解体子

8.13.2.1 GgShape()

```
gg::GgShape::GgShape (
    GLenum mode = 0) [inline]
```

コンストラクタ。

引数

<i>mode</i>	基本図形の種類。
-------------	----------

gg.h の 6184 行目に定義があります。

8.13.2.2 ~GgShape()

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

デストラクタ。

gg.h の 6193 行目に定義があります。

8.13.3 関数詳解

8.13.3.1 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。

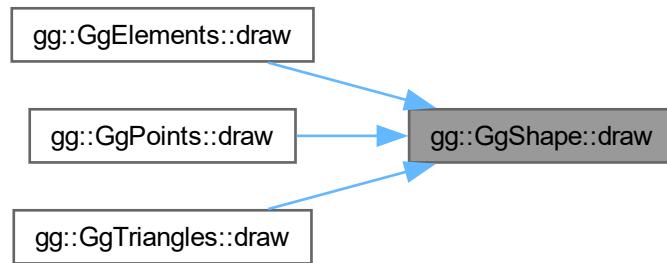
引数

<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::GgElements`, `gg::GgPoints`, `gg::GgTriangles`で再実装されています。

`gg.h` の 6253 行目に定義があります。

被呼び出し関係図:



8.13.3.2 `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 6222 行目に定義があります。

被呼び出し関係図:



8.13.3.3 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 6242 行目に定義があります。

8.13.3.4 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 6202 行目に定義があります。

8.13.3.5 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 6212 行目に定義があります。

8.13.3.6 setMode()

```
void gg::GgShape::setMode (
    GLenum mode) [inline]
```

基本図形の設定.

引数

<i>mode</i>	基本図形の種類.
-------------	----------

gg.h の 6232 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

8.14 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`

8.14.1 詳解

Wavefront OBJ 形式のファイル (Arrays 形式).

gg.h の 8128 行目に定義があります。

8.14.2 構築子と解体子

8.14.2.1 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 6022 行目に定義がります。

呼び出し関係図:



8.14.2.2 ~GgSimpleObj()

virtual gg::GgSimpleObj::~GgSimpleObj () [inline], [virtual]

デストラクタ.

gg.h の 8151 行目に定義がります。

8.14.3 関数詳解

8.14.3.1 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>	描画する最初のパート番号.
<i>count</i>	描画するパートの数, 0 なら全部のパートを描く.

gg.cpp の 6050 行目に定義がります。

被呼び出し関係図:



8.14.3.2 `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 8180 行目に定義がります。

呼び出し関係図:



8.14.3.3 `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 8160 行目に定義がります。

8.14.3.4 `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 8170 行目に定義がります。

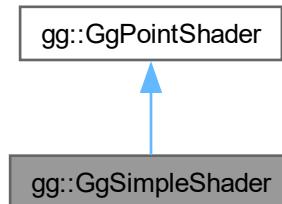
このクラス詳解は次のファイルから抽出されました:

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

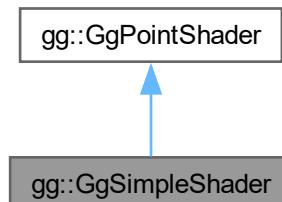
8.15 gg::GgSimpleShader クラス

```
#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

基底クラス **gg::GgPointShader** に属する継承公開メンバ関数

- **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
- 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

8.15.1 詳解

三角形に単純な陰影付けを行うシェーダ。

gg.h の 7151 行目に定義があります。

8.15.2 構築子と解体子

8.15.2.1 GgSimpleShader() [1/4]

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

コンストラクタ.

[gg.h](#) の 7168 行目に定義がります。

8.15.2.2 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 7185 行目に定義がります。

8.15.2.3 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 7203 行目に定義がります。

8.15.2.4 GgSimpleShader() [4/4]

```
gg::GgSimpleShader::GgSimpleShader (
    const GgSimpleShader & o) [inline]
```

コピー コンストラクタ.

[gg.h](#) の 7215 行目に定義があります。

8.15.2.5 ~GgSimpleShader()

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

デストラクタ.

[gg.h](#) の 7226 行目に定義があります。

8.15.3 関数詳解

8.15.3.1 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>	パーティ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 <i>varying</i> 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト (<i>nullptr</i> 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 7272 行目に定義があります。

8.15.3.2 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

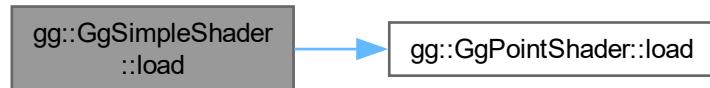
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列.
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>nvarying</i>	フィードバックする <i>varying</i> 変数の数 (0なら不使用).
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト (<i>nullptr</i> なら不使用).

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 6005 行目に定義があります。

呼び出し関係図:

8.15.3.3 `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>	<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.

`gg::GgPointShader`を再実装しています。

`gg.h` の 7366 行目に定義があります。

呼び出し関係図:



8.15.3.4 loadMatrix() [2/4]

```
virtual void gg::GgSimpleShader::loadMatrix (
    const GgMatrix & mp,
    const GgMatrix & mv,
    const GgMatrix & mn) const [inline], [virtual]
```

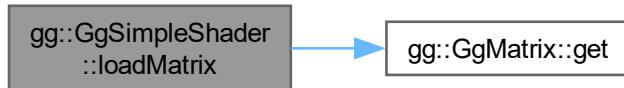
投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

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 7344 行目に定義があります。

呼び出し関係図:



8.15.3.5 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 7355 行目に定義があります。

8.15.3.6 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 7331 行目に定義があります。

8.15.3.7 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>	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 7319 行目に定義があります。

呼び出し関係図:



8.15.3.8 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>	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 7299 行目に定義があります。

呼び出し関係図:



8.15.3.9 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 7309 行目に定義があります。

8.15.3.10 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 7287 行目に定義があります。

8.15.3.11 operator=()

```
GgSimpleShader & gg::GgSimpleShader::operator= (
    const GgSimpleShader & o) [inline]
```

代入演算子.

gg.h の 7233 行目に定義があります。

8.15.3.12 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 7895 行目に定義があります。

呼び出し関係図:



8.15.3.13 use() [2/13]

```
void gg::GgSimpleShader::use (
    const GgMatrix & mp,
    const GgMatrix & mv) const [inline]
```

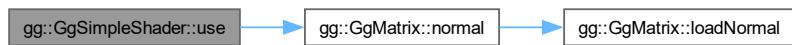
投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する.

引数

<i>mp</i>	<i>GgMatrix</i> 型の投影変換行列.
<i>mv</i>	<i>GgMatrix</i> 型のモデルビュー変換行列.

gg.h の 7946 行目に定義があります。

呼び出し関係図:



8.15.3.14 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 7924 行目に定義があります。

呼び出し関係図:



8.15.3.15 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 8010 行目に定義があります。

呼び出し関係図:



8.15.3.16 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>	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>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 8047 行目に定義があります。

呼び出し関係図:



8.15.3.17 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 8080 行目に定義があります。

呼び出し関係図:



8.15.3.18 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 7935 行目に定義があります。

8.15.3.19 use() [8/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv,
    const GLfloat * 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 7908 行目に定義があります。

8.15.3.20 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 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 7986 行目に定義があります。

8.15.3.21 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 8029 行目に定義があります。

8.15.3.22 `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 <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 8064 行目に定義があります。

8.15.3.23 `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 <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 7972 行目に定義があります。

8.15.3.24 `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 <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 7957 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](#)

8.16 gg::GgTexture クラス

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

公開 メンバ 関数

- [GgTexture \(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE, bool swizzle=true\)](#)
- [GgTexture \(const GgTexture &texture\)=delete](#)
- [GgTexture \(GgTexture &&texture\)=default](#)
- [virtual ~GgTexture \(\)](#)
- [GgTexture & operator= \(const GgTexture &texture\)=delete](#)
- [GgTexture & operator= \(GgTexture &&texture\)=default](#)
- [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 GLuint * getTexture \(\) const](#)
- [const GLuint & getTexture \(\) const](#)

8.16.1 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 5152 行目に定義があります。

8.16.2 構築子と解体子

8.16.2.1 GgTexture() [1/3]

```
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, <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>	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 <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> .

[gg.h](#) の 5174 行目に定義があります。

8.16.2.2 GgTexture() [2/3]

```
gg::GgTexture::GgTexture (
    const GgTexture & texture) [delete]
```

コピー構築子は使用しない.

8.16.2.3 GgTexture() [3/3]

```
gg::GgTexture::GgTexture ( GgTexture && texture) [default]
```

ムーブコンストラクタはデフォルトのものを使用する。

8.16.2.4 ~GgTexture()

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

デストラクタ。

[gg.h](#) の 5202 行目に定義があります。

8.16.3 関数詳解

8.16.3.1 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 5221 行目に定義があります。

8.16.3.2 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 5256 行目に定義があります。

被呼び出し関係図:



8.16.3.3 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 5277 行目に定義があります。

8.16.3.4 getSize() [2/2]

```
void gg::GgTexture::getSize (
    GLsizei * size) const [inline]
```

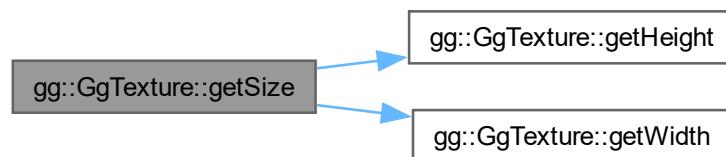
使用しているテクスチャのサイズを取り出す。

引数

size	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 5266 行目に定義があります。

呼び出し関係図:



8.16.3.5 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 5287 行目に定義があります。

8.16.3.6 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 5246 行目に定義があります。

被呼び出し関係図:



8.16.3.7 operator=() [1/2]

```
GgTexture & gg::GgTexture::operator= (
    const GgTexture & texture) [delete]
```

代入演算子は使用しない。

8.16.3.8 operator=() [2/2]

```
GgTexture & gg::GgTexture::operator= (
    GgTexture && texture) [default]
```

ムーブ代入演算子はデフォルトのものを使用する。

8.16.3.9 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 4024 行目に定義があります。

8.16.3.10 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 5229 行目に定義があります。

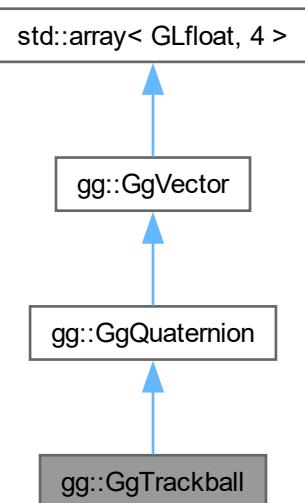
このクラス詳解は次のファイルから抽出されました:

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

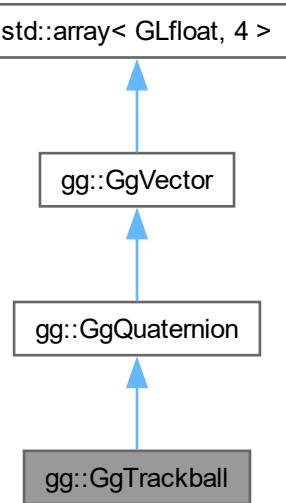
8.17 gg::GgTrackball クラス

```
#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())`
- `~GgTrackball ()=default`
- `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`

基底クラス [gg::GgQuaternion](#) に属する継承公開メンバ関数

- `GgQuaternion ()=default`
- `constexpr GgQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `constexpr GgQuaternion (GLfloat c)`
- `GgQuaternion (const GLfloat *a)`

- `GgQuaternion (const GgVector &v)`
- `GgQuaternion (const GgQuaternion &q)=default`
- `GgQuaternion (GgQuaternion &&q)=default`
- `~GgQuaternion ()=default`
- `GgQuaternion & operator= (const GgQuaternion &q)=default`
- `GgQuaternion & operator= (GgQuaternion &&q)=default`
- `GLfloat norm () const`
- `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`

基底クラス [gg::GgVector](#) に属する継承公開メンバ関数

- [GgVector \(\)=default](#)
- [constexpr GgVector \(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3\)](#)
- [constexpr GgVector \(const GLfloat *a\)](#)
- [constexpr GgVector \(GLfloat c\)](#)
- [GgVector \(const GgVector &v\)=default](#)
- [GgVector \(GgVector &&v\)=default](#)
- [~GgVector \(\)=default](#)
- [GgVector & operator= \(const GgVector &v\)=default](#)
- [GgVector & operator= \(GgVector &&v\)=default](#)
- [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/ \(GLfloat c\) const](#)
- [GgVector & operator/= \(GgVector &v\)](#)
- [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](#)

8.17.1 詳解

簡易トラックボール処理。

[gg.h](#) の 4734 行目に定義があります。

8.17.2 構築子と解体子

8.17.2.1 GgTrackball()

```
gg::GgTrackball::GgTrackball (
    const GgQuaternion & q = ggIdentityQuaternion() ) [inline]
```

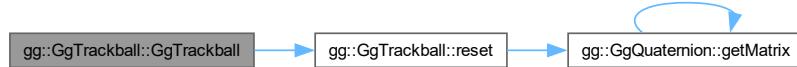
コンストラクタ。

引数

q	トラックボールの回転の初期値の四元数.
-----	---------------------

gg.h の 4749 行目に定義があります。

呼び出し関係図:



8.17.2.2 ~GgTrackball()

gg::GgTrackball::~GgTrackball () [default]

デストラクタ.

8.17.3 関数詳解

8.17.3.1 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

x	現在のマウスの x 座標.
y	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 3447 行目に定義があります。

8.17.3.2 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 3513 行目に定義があります。

8.17.3.3 `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 4927 行目に定義があります。

呼び出し関係図:



8.17.3.4 `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 4917 行目に定義があります。

8.17.3.5 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 4907 行目に定義があります。

8.17.3.6 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 4876 行目に定義があります。

8.17.3.7 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 4896 行目に定義があります。

8.17.3.8 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 4886 行目に定義があります。

8.17.3.9 `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 4847 行目に定義があります。

8.17.3.10 `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 4865 行目に定義があります。

8.17.3.11 `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 4855 行目に定義があります。

8.17.3.12 `motion()`

```
void gg::GgTrackball::motion (
    GLfloat x,
    GLfloat y)
```

回転の変換行列を計算する.

引数

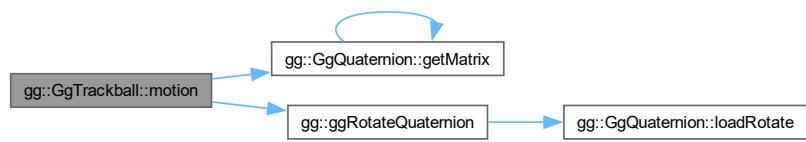
<i>x</i>	現在のマウスの x 座標.
<i>y</i>	現在のマウスの y 座標.

覚え書き

マウスのドラッグ中に呼び出す.

gg.cpp の 3463 行目に定義があります。

呼び出し関係図:



8.17.3.13 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 4764 行目に定義があります。

呼び出し関係図:



8.17.3.14 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

<i>w</i>	領域の横幅.
<i>h</i>	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 3434 行目に定義があります。

呼び出し関係図:



8.17.3.15 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

<i>w</i>	領域の横幅.
<i>h</i>	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 4790 行目に定義があります。

呼び出し関係図:



8.17.3.16 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 3416 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:



8.17.3.17 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

<code>q</code>	修正分の回転角の四元数。
----------------	--------------

gg.cpp の 3492 行目に定義がります。

呼び出し関係図:



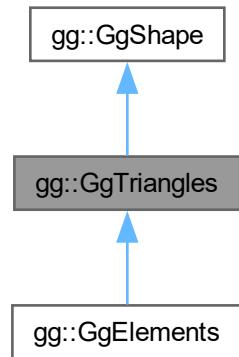
このクラス詳解は次のファイルから抽出されました:

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

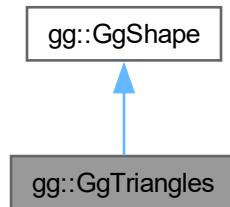
8.18 gg::GgTriangles クラス

```
#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`

基底 クラス [gg::GgShape](#) に属する継承公開 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`

8.18.1 詳解

三角形で表した形状データ (Arrays 形式).

`gg.h` の 6438 行目に定義があります。

8.18.2 構築子と解体子

8.18.2.1 GgTriangles() [1/2]

```

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

コンストラクタ.

引数

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

gg.h の 6451 行目に定義があります。

8.18.2.2 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 6464 行目に定義があります。

8.18.2.3 ~GgTriangles()

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

デストラクタ.

gg.h の 6478 行目に定義があります。

8.18.3 関数詳解

8.18.3.1 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 5189 行目に定義があります。

呼び出し関係図:



8.18.3.2 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 6497 行目に定義があります。

8.18.3.3 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 6487 行目に定義があります。

8.18.3.4 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 5170 行目に定義があります。

8.18.3.5 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 6509 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](#)

8.19 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](#)

8.19.1 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 5766 行目に定義があります。

8.19.2 構築子と解体子

8.19.2.1 GgUniformBuffer() [1/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer () [inline]
```

コンストラクタ.

[gg.h](#) の 5769 行目に定義があります。

8.19.2.2 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プ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 5769 行目に定義があります。

8.19.2.3 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プ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 5769 行目に定義があります。

8.19.2.4 ~GgUniformBuffer()

```
template<typename T >
virtual gg::GgUniformBuffer< T >::~GgUniformBuffer () [inline], [virtual]
```

デストラクタ.

gg.h の 5769 行目に定義があります。

8.19.3 関数詳解

8.19.3.1 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 5854 行目に定義があります。

8.19.3.2 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 (buffer) の先頭のデータの位置.
<i>dst_first</i>	複写先 (getBuffer()) 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 6065 行目に定義があります。

8.19.3.3 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 5982 行目に定義があります。

8.19.3.4 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 5846 行目に定義があります。

8.19.3.5 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 5836 行目に定義があります。

8.19.3.6 `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 5826 行目に定義があります。

8.19.3.7 `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 5816 行目に定義があります。

8.19.3.8 `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

<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 5923 行目に定義があります。

8.19.3.9 `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

<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 5904 行目](#)に定義があります。

8.19.3.10 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 5872 行目](#)に定義があります。

8.19.3.11 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 5884 行目](#)に定義があります。

8.19.3.12 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 6017 行目に定義があります。

8.19.3.13 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 5944 行目に定義があります。

8.19.3.14 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 5862 行目に定義があります。

8.19.3.15 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 5892 行目に定義があります。

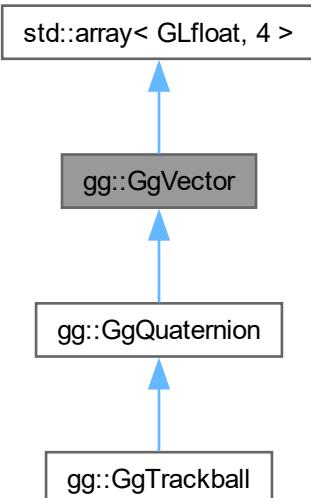
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

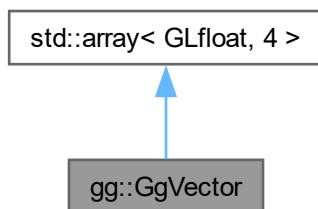
8.20 gg::GgVector クラス

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

gg::GgVector の継承関係図



gg::GgVector 連携図



公開メンバ関数

- `GgVector ()=default`
- `constexpr GgVector (GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3)`
- `constexpr GgVector (const GLfloat *a)`
- `constexpr GgVector (GLfloat c)`
- `GgVector (const GgVector &v)=default`
- `GgVector (GgVector &&v)=default`
- `~GgVector ()=default`
- `GgVector & operator= (const GgVector &v)=default`
- `GgVector & operator= (GgVector &&v)=default`
- `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/ (GLfloat c) const`
- `GgVector & operator/= (GgVector &v)`
- `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`

8.20.1 詳解

4要素の単精度実数の配列。

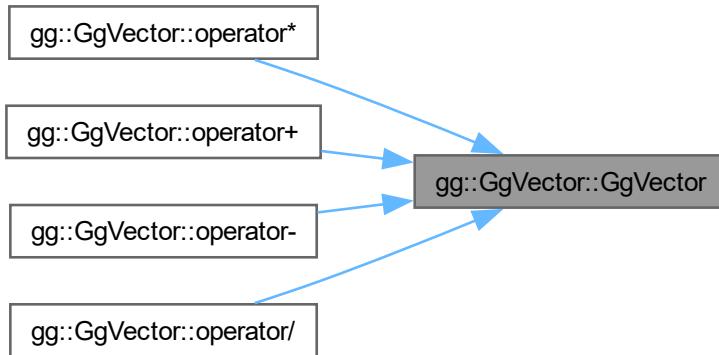
`gg.h` の 1571 行目に定義があります。

8.20.2 構築子と解体子

8.20.2.1 `GgVector()` [1/6]

```
gg::GgVector::GgVector () [default]
```

コンストラクタ、被呼び出し関係図:



8.20.2.2 GgVector() [2/6]

```
gg::GgVector::GgVector (
    GLfloat v0,
    GLfloat v1,
    GLfloat v2,
    GLfloat v3) [inline], [constexpr]
```

コンストラクタ.

引数

v0	GLfloat 型の値.
v1	GLfloat 型の値.
v2	GLfloat 型の値.
v3	GLfloat 型の値.

gg.h の 1588 行目に定義があります。

8.20.2.3 GgVector() [3/6]

```
gg::GgVector::GgVector (
    const GLfloat * a) [inline], [constexpr]
```

コンストラクタ.

引数

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

gg.h の 1598 行目に定義があります。

8.20.2.4 GgVector() [4/6]

```
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 1608 行目に定義があります。

8.20.2.5 GgVector() [5/6]

```
gg::GgVector::GgVector (
    const GgVector & v) [default]
```

コピーコンストラクタ.

8.20.2.6 GgVector() [6/6]

```
gg::GgVector::GgVector (
    GgVector && v) [default]
```

ムーブコンストラクタ.

8.20.2.7 ~GgVector()

```
gg::GgVector::~GgVector () [default]
```

デストラクタ.

8.20.3 関数詳解

8.20.3.1 distance3()

```
GLfloat gg::GgVector::distance3 (
    const GgVector & v) const [inline]
```

GgVector 型の 3 要素の距離.

引数

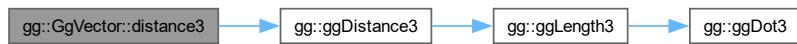
<i>v</i>	GgVector 型のベクトル.
----------	------------------

戻り値

オブジェクトと v の 3 要素の距離.

gg.h の 1848 行目に定義があります。

呼び出し関係図:



8.20.3.2 distance4()

```
GLfloat gg::GgVector::distance4 (
    const GgVector & v) const [inline]
```

GgVector 型の 4 要素の距離.

引数

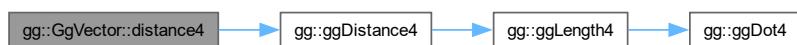
v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v の 4 要素の距離.

gg.h の 1892 行目に定義があります。

呼び出し関係図:



8.20.3.3 dot3()

```
GLfloat gg::GgVector::dot3 (
    const GgVector & v) const [inline]
```

GgVector 型の 3 要素の内積.

引数

<code>v</code>	<code>GgVector</code> 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 1827 行目に定義があります。

呼び出し関係図:



8.20.3.4 `dot4()`

```
GLfloat gg::GgVector::dot4 (
    const GgVector & v) const [inline]
```

`GgVector` 型の 4 要素の内積.

引数

<code>v</code>	<code>GgVector</code> 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 1871 行目に定義があります。

呼び出し関係図:



8.20.3.5 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 1837 行目に定義があります。

呼び出し関係図:



8.20.3.6 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 1881 行目に定義があります。

呼び出し関係図:



8.20.3.7 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 1858 行目に定義があります。

呼び出し関係図:



8.20.3.8 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 1902 行目に定義があります。

呼び出し関係図:



8.20.3.9 operator*() [1/2]

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

GgVector 型の積を返す.

引数

v	GgVector 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 1736 行目に定義があります。

8.20.3.10 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

c	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 1747 行目に定義があります。

呼び出し関係図:



8.20.3.11 operator*=(()) [1/2]

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

GgVector 型を乗算する.

引数

<code>v</code>	<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乗算したオブジェクトの参照.

`gg.h` の 1757 行目に定義があります。

8.20.3.12 `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>	<code>GgVector</code> 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 1769 行目に定義があります。

8.20.3.13 `operator+()` [1/2]

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

`GgVector` 型の和を返す.

引数

<code>v</code>	<code>GgVector</code> 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 1644 行目に定義があります。

8.20.3.14 `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 1655 行目に定義があります。

呼び出し関係図:



8.20.3.15 operator+=() [1/2]

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

GgVector 型を加算する.

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加算したオブジェクトの参照.

gg.h の 1666 行目に定義があります。

8.20.3.16 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>	GLfloat 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 1678 行目に定義があります。

8.20.3.17 operator-() [1/2]

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

`GgVector` 型の差を返す.

引数

<code>v</code>	GgVector 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 1690 行目に定義があります。

8.20.3.18 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

<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.

`gg.h` の 1701 行目に定義があります。

呼び出し関係図:



8.20.3.19 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 1712 行目に定義があります。

8.20.3.20 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

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の参照.

[gg.h](#) の 1724 行目に定義があります。

8.20.3.21 operator/() [1/2]

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

GgVector 型の商を返す.

引数

v	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 1781 行目に定義があります。

8.20.3.22 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.

`gg.h` の 1792 行目に定義があります。

呼び出し関係図:



8.20.3.23 operator/() [1/2]

```
GgVector & gg::GgVector::operator/= (
    GgVector & v) [inline]
```

`GgVector` 型を除算する.

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乗算したオブジェクトの参照.

`gg.h` の 1803 行目に定義があります。

8.20.3.24 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

<code>c</code>	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 1815 行目に定義があります。

8.20.3.25 operator=() [1/2]

```
GgVector & gg::GgVector::operator= (
    const GgVector & v) [default]
```

代入演算子.

8.20.3.26 operator=() [2/2]

```
GgVector & gg::GgVector::operator= (
    GgVector && v) [default]
```

ムーブ代入演算子.

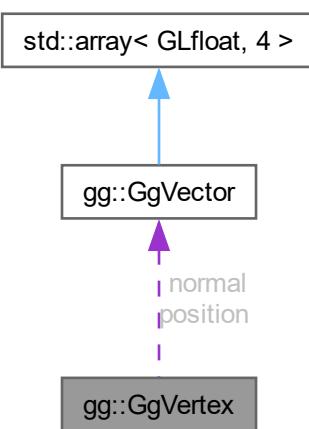
このクラス詳解は次のファイルから抽出されました:

- `gg.h`

8.21 gg::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.

8.21.1 詳解

三角形の頂点データ.

`gg.h` の 6377 行目に定義があります。

8.21.2 構築子と解体子

8.21.2.1 `GgVertex()` [1/4]

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

コンストラクタ.

`gg.h` の 6388 行目に定義があります。

8.21.2.2 `GgVertex()` [2/4]

```
gg::GgVertex::GgVertex (
    const GgVector & pos,
    const GgVector & norm) [inline]
```

コンストラクタ.

引数

<code>pos</code>	<code>GgVector</code> 型の位置データ.
<code>norm</code>	<code>GgVector</code> 型の法線データ.

`gg.h` の 6398 行目に定義があります。

8.21.2.3 `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>	<code>GgVector</code> 型の位置データの x 成分.
<i>py</i>	<code>GgVector</code> 型の位置データの y 成分.
<i>pz</i>	<code>GgVector</code> 型の位置データの z 成分.
<i>nx</i>	<code>GgVector</code> 型の法線データの x 成分.
<i>ny</i>	<code>GgVector</code> 型の法線データの y 成分.
<i>nz</i>	<code>GgVector</code> 型の法線データの z 成分.

`gg.h` の 6414 行目に定義がります。

8.21.2.4 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 6429 行目に定義がります。

8.21.3 メンバ詳解

8.21.3.1 normal

`GgVector` `gg::GgVertex::normal`

法線.

`gg.h` の 6383 行目に定義がります。

8.21.3.2 position

`GgVector` `gg::GgVertex::position`

位置.

`gg.h` の 6380 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`

8.22 gg::GgVertexArray クラス

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

公開 メンバ 関数

- `GgVertexArray (GLenum mode=0)`
- `GgVertexArray (const GgVertexArray &array)=delete`
- `GgVertexArray (GgVertexArray &&array)=default`
- `virtual ~GgVertexArray ()`
- `GgVertexArray & operator= (const GgVertexArray &array)=delete`
- `GgVertexArray & operator= (GgVertexArray &&array)=default`
- `const GLuint & get () const`
- `void bind () const`

8.22.1 詳解

頂点配列 クラス.

`gg.h` の 6096 行目に定義があります。

8.22.2 構築子と解体子

8.22.2.1 GgVertexArray() [1/3]

```
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 6108 行目に定義があります。

8.22.2.2 GgVertexArray() [2/3]

```
gg::GgVertexArray::GgVertexArray (
    const GgVertexArray & array) [delete]
```

コピー コンストラクタは使用しない.

8.22.2.3 GgVertexArray() [3/3]

```
gg::GgVertexArray::GgVertexArray (
    GgVertexArray && array) [default]
```

ムーブコンストラクタはデフォルトのものを使用する。

8.22.2.4 ~GgVertexArray()

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

デストラクタ。

gg.h の 6127 行目に定義があります。

8.22.3 関数詳解

8.22.3.1 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 6156 行目に定義があります。

8.22.3.2 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 6148 行目に定義があります。

8.22.3.3 operator=() [1/2]

```
GgVertexArray & gg::GgVertexArray::operator= (
    const GgVertexArray & array) [delete]
```

代入演算子は使用しない。

8.22.3.4 operator=() [2/2]

```
GgVertexArray & gg::GgVertexArray::operator= (
    GgVertexArray && array) [default]
```

ムーブ代入演算子はデフォルトのものを使用する。

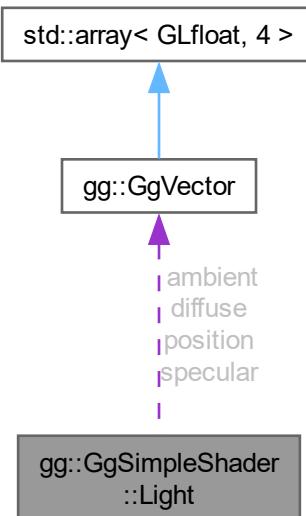
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.23 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.

8.23.1 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 7374 行目に定義があります。

8.23.2 メンバ詳解

8.23.2.1 ambient

`GgVector gg::GgSimpleShader::Light::ambient`

光源強度の環境光成分。

gg.h の 7376 行目に定義があります。

8.23.2.2 diffuse

`GgVector gg::GgSimpleShader::Light::diffuse`

光源強度の拡散反射光成分。

gg.h の 7377 行目に定義があります。

8.23.2.3 position

`GgVector gg::GgSimpleShader::Light::position`

光源の位置。

gg.h の 7379 行目に定義があります。

8.23.2.4 specular

`GgVector gg::GgSimpleShader::Light::specular`

光源強度の鏡面反射光成分。

gg.h の 7378 行目に定義があります。

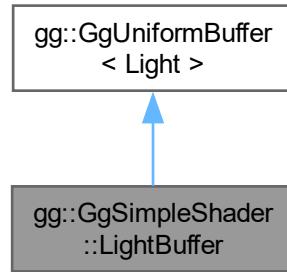
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

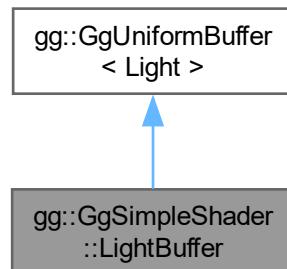
8.24 gg::GgSimpleShader::LightBuffer クラス

```
#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

基底クラス `gg::GgUniformBuffer< Light >` に属する継承公開メンバ関数

- `GgUniformBuffer` ()
- `GgUniformBuffer` (const `Light` *data, GLsizei count, GLenum usage=GL_STATIC_DRAW)
- `GgUniformBuffer` (const `Light` &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 `Light` *data, GLsizei count, GLenum usage=GL_STATIC_DRAW)
- void `load` (const `Light` &data, GLsizei count, GLenum usage=GL_STATIC_DRAW)
- void `send` (const `GLvoid` *data, GLint offset=0, GLsizei size=`sizeof(Light)`, GLint first=0, GLsizei count=0) const
- void `fill` (const `GLvoid` *data, GLint offset=0, GLsizei size=`sizeof(Light)`, GLint first=0, GLsizei count=0) const
- void `read` (`GLvoid` *data, GLint offset=0, GLsizei size=`sizeof(Light)`, GLint first=0, GLsizei count=0) const
- void `copy` (`GLuint` src_buffer, GLint src_first=0, GLint dst_first=0, GLsizei count=0) const

8.24.1 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 7385 行目に定義があります。

8.24.2 構築子と解体子

8.24.2.1 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>	<code>GgSimpleShader::Light</code> 型の光源データのポインタ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Light</code> 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 7397 行目に定義があります。

8.24.2.2 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>	<code>GgSimpleShader::Light</code> 型の光源データ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Light</code> 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 7413 行目に定義があります。

8.24.2.3 ~LightBuffer()

```
virtual gg::GgSimpleShader::LightBuffer::~LightBuffer () [inline], [virtual]
```

デストラクタ.

[gg.h](#) の 7425 行目に定義があります。

8.24.3 関数詳解

8.24.3.1 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 7617 行目に定義があります。

8.24.3.2 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 GgSimpleShader::Light 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 7605 行目に定義があります。

8.24.3.3 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 GgVector 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 5515 行目に定義があります。

8.24.3.4 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 <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 7460 行目に定義があります。

8.24.3.5 `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 5489 行目に定義があります。

8.24.3.6 `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 <code>GgSimpleShader::Light</code> 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 5642 行目に定義があります。

8.24.3.7 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 GgVector 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 5567 行目に定義があります。

8.24.3.8 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 GLfloat 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 7497 行目に定義があります。

8.24.3.9 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 5541 行目に定義がります。

8.24.3.10 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 GgVector 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 5696 行目に定義がります。

8.24.3.11 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 GgVector 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 7593 行目に定義がります。

8.24.3.12 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 GLfloat 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 7580 行目に定義があります。

8.24.3.13 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 5670 行目に定義があります。

8.24.3.14 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 GgVector 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 5619 行目に定義があります。

8.24.3.15 `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 <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 7534 行目に定義があります。

8.24.3.16 `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 5593 行目に定義があります。

8.24.3.17 `select()`

```
void gg::GgSimpleShader::LightBuffer::select (
    GLint i = 0) const [inline]
```

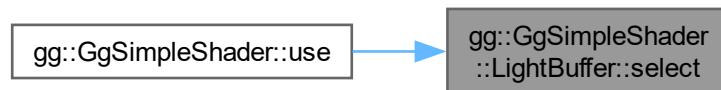
光源を選択する。

引数

<i>i</i>	光源データの uniform block のインデックス.
----------	-------------------------------

gg.h の 7627 行目に定義があります。

被呼び出し関係図:



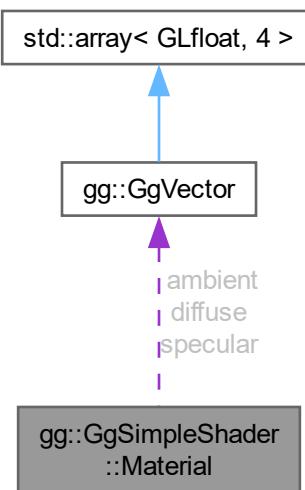
このクラス詳解は次のファイルから抽出されました:

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

8.25 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.

8.25.1 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 7638 行目に定義があります。

8.25.2 メンバ詳解

8.25.2.1 ambient

`GgVector gg::GgSimpleShader::Material::ambient`

環境光に対する反射係数.

[gg.h](#) の 7640 行目に定義があります。

8.25.2.2 diffuse

`GgVector gg::GgSimpleShader::Material::diffuse`

拡散反射係数.

[gg.h](#) の 7641 行目に定義があります。

8.25.2.3 shininess

`GLfloat gg::GgSimpleShader::Material::shininess`

輝き係数.

[gg.h](#) の 7643 行目に定義があります。

8.25.2.4 specular

`GgVector gg::GgSimpleShader::Material::specular`

鏡面反射係数。

`gg.h` の 7642 行目に定義があります。

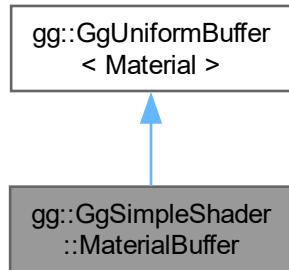
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

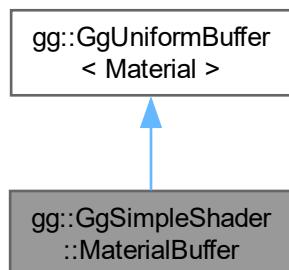
8.26 gg::GgSimpleShader::MaterialBuffer クラス

#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`

基底クラス `gg::GgUniformBuffer< Material >` に属する継承公開メンバ関数

- `GgUniformBuffer ()`
- `GgUniformBuffer (const Material *data, GLsizei count, GLenum usage=GL_STATIC_DRAW)`
- `GgUniformBuffer (const Material &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 Material *data, GLsizei count, GLenum usage=GL_STATIC_DRAW)`
- `void load (const Material &data, GLsizei count, GLenum usage=GL_STATIC_DRAW)`
- `void send (const GLvoid *data, GLint offset=0, GLsizei size=sizeof(Material), GLint first=0, GLsizei count=0) const`
- `void fill (const GLvoid *data, GLint offset=0, GLsizei size=sizeof(Material), GLint first=0, GLsizei count=0) const`
- `void read (GLvoid *data, GLint offset=0, GLsizei size=sizeof(Material), 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

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 7649 行目に定義があります。

8.26.2 構築子と解体子

8.26.2.1 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>	GgSimpleShader::Material 型の材質データのポインタ.
<i>count</i>	バッファ中の GgSimpleShader::Material 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, glBufferData() の第 4 引数の usage に渡される.

gg.h の 7661 行目に定義があります。

8.26.2.2 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>	GgSimpleShader::Material 型の材質データ.
<i>count</i>	バッファ中の GgSimpleShader::Material 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, glBufferData() の第 4 引数の usage に渡される.

gg.h の 7677 行目に定義があります。

8.26.2.3 ~MaterialBuffer()

```
virtual gg::GgSimpleShader::MaterialBuffer::~MaterialBuffer () [inline], [virtual]
デストラクタ.
```

gg.h の 7689 行目に定義があります。

8.26.3 関数詳解

8.26.3.1 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 7874 行目に定義があります。

8.26.3.2 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 7862 行目に定義があります。

8.26.3.3 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 GgVector 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 5748 行目に定義があります。

8.26.3.4 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 <code>GLfloat</code> 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 7724 行目に定義がります。

8.26.3.5 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 5722 行目に定義がります。

8.26.3.6 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 <code>GgVector</code> 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 5852 行目に定義がります。

8.26.3.7 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 <code>GLfloat</code> 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 5875 行目に定義があります。

8.26.3.8 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 5826 行目に定義があります。

8.26.3.9 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 GgVector 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 5800 行目に定義があります。

8.26.3.10 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 7761 行目に定義があります。

8.26.3.11 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 5774 行目に定義があります。

8.26.3.12 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 5988 行目に定義があります。

8.26.3.13 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 5967 行目に定義があります。

8.26.3.14 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 GgVector 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 5944 行目に定義があります。

8.26.3.15 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 <code>GLfloat</code> 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 7831 行目に定義があります。

8.26.3.16 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 5918 行目に定義があります。

8.26.3.17 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 7884 行目に定義がります。

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

8.27 Menu クラス

```
#include <Menu.h>
```

公開メンバ関数

- Menu (const Config &config)
- Menu (const Menu &menu)=delete
- virtual ~Menu ()
- Menu & operator= (const Menu &menu)=delete
- void draw ()

フレンド

- class Draw

8.27.1 詳解

メニューの描画

Menu.h の 20 行目に定義がります。

8.27.2 構築子と解体子

8.27.2.1 Menu() [1/2]

```
Menu::Menu (
    const Config & config)
```

コンストラクタ

Menu.cpp の 13 行目に定義がります。

8.27.2.2 Menu() [2/2]

```
Menu::Menu (const Menu & menu) [delete]
```

8.27.2.3 ~Menu()

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

デストラクタ

Menu.cpp の 49 行目に定義があります。

8.27.3 関数詳解

8.27.3.1 draw()

```
void Menu::draw ()
```

描画する

Menu.cpp の 74 行目に定義があります。

8.27.3.2 operator=()

```
Menu & Menu::operator= (const Menu & menu) [delete]
```

8.27.4 フレンドと関連関数の詳解

8.27.4.1 Draw

```
friend class Draw [friend]
```

Menu.h の 23 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](#)

8.28 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`
- `Window (Window &&w)=default`
- `virtual ~Window ()`
- `Window & operator= (const Window &w)=delete`
- `Window & operator= (Window &&w)=default`
- `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`
- `void getSize (GLsizei *size) const`
- `const auto & getFboSize () const`
- `void getFboSize (GLsizei *fboSize) 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 getMouseX () const`
- `auto getMouseY () 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))

8.28.1 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 136 行目に定義があります。

8.28.2 構築子と解体子

8.28.2.1 Window() [1/3]

```
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 348 行目に定義があります。

呼び出し関係図:



8.28.2.2 Window() [2/3]

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

コピー構造関数は使用しない

8.28.2.3 Window() [3/3]

```
GgApp::Window::Window (
    Window && w) [default]
```

ムーブ構造関数はデフォルトのものを使用する

8.28.2.4 ~Window()

```
virtual GgApp::Window::~Window () [inline], [virtual]
```

デストラクタ.

[GgApp.h](#) の 266 行目に定義があります。

8.28.3 関数詳解

8.28.3.1 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 290 行目に定義があります。

8.28.3.2 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 614 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:



8.28.3.3 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 624 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:



8.28.3.4 getAltArrow()

```
void GgApp::Window::getAltArrow (
    GLfloat * alt_arrow) const [inline]
```

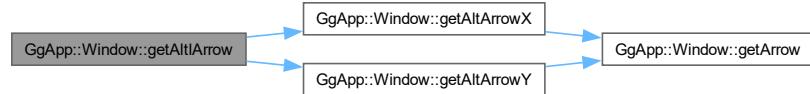
ALT キーを押しながら矢印キーを押したときの現在の値を得る。

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 634 行目に定義があります。

呼び出し関係図:



8.28.3.5 getArrow() [1/2]

```
void GgApp::Window::getArrow (
    GLfloat * arrow,
    int mods = 0) const [inline]
```

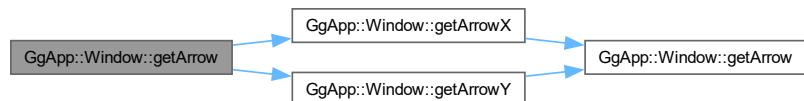
矢印キーの現在の値を得る。

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 541 行目に定義がります。

呼び出し関係図:



8.28.3.6 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

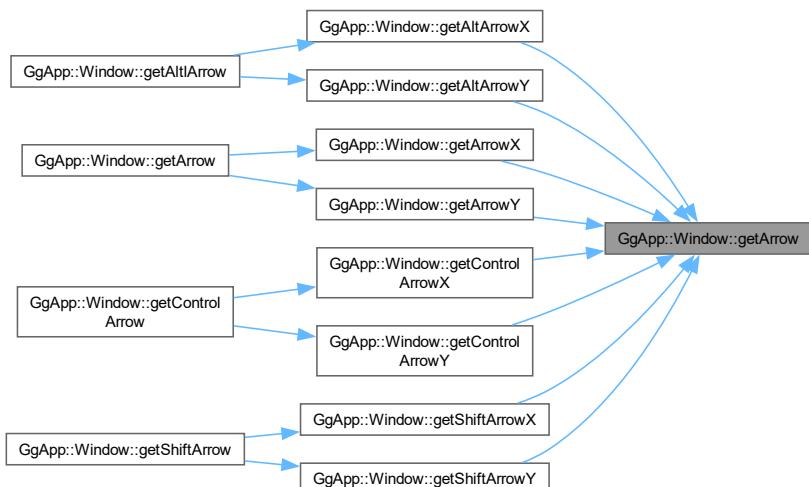
<i>direction</i>	方向 (0: X, 1:Y).
<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).

戻り値

矢印キーの値.

GgApp.h の 507 行目に定義がります。

被呼び出し関係図:



8.28.3.7 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 519 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:



8.28.3.8 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 530 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:



8.28.3.9 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 444 行目に定義があります。

8.28.3.10 getControlArrow()

```
void GgApp::Window::getControlArrow (
    GLfloat * control_arrow) const [inline]
```

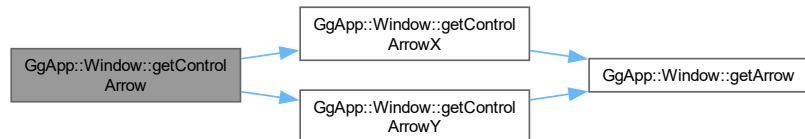
CTRL キーを押しながら矢印キーを押したときの現在の値を得る.

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 603 行目に定義があります。

呼び出し関係図:



8.28.3.11 `getControlArrowX()`

```
auto GgApp::Window::getControlArrowX () const [inline]
```

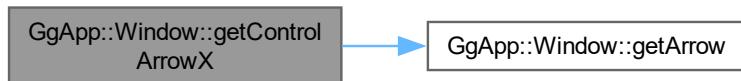
CTRL キーを押しながら矢印キーを押したときの現在の X 値を得る.

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 583 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:



8.28.3.12 getControlArrowY()

```
auto GgApp::Window::getControlArrowY () const [inline]
```

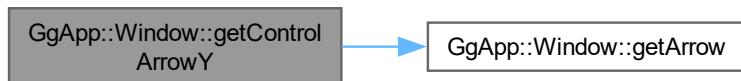
CTRL キーを押しながら矢印キーを押したときの現在の Y 値を得る.

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 593 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:



8.28.3.13 getFboHeight()

```
auto GgApp::Window::getFboHeight () const [inline]
```

FBO の高さを得る.

戻り値

FBO の高さ.

[GgApp.h](#) の 392 行目に定義があります。

被呼び出し関係図:



8.28.3.14 `getFboSize()` [1/2]

```
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 423 行目に定義があります。

8.28.3.15 `getFboSize()` [2/2]

```
void GgApp::Window::getFboSize (
    GLsizei * fboSize) const [inline]
```

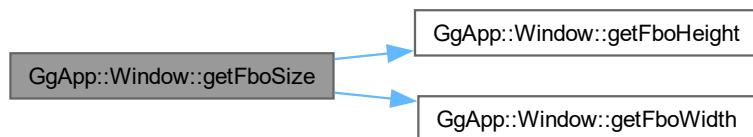
FBO のサイズを得る.

引数

<code>size</code>	FBO の幅と高さを格納した <code>GLsizei</code> 型の 2 要素の配列.
-------------------	---

[GgApp.h](#) の 433 行目に定義があります。

呼び出し関係図:



8.28.3.16 `getFboWidth()`

```
auto GgApp::Window::getFboWidth () const [inline]
FBO の横幅を得る.
```

戻り値

FBO の横幅.

[GgApp.h](#) の 382 行目に定義があります。

被呼び出し関係図:



8.28.3.17 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 372 行目に定義があります。

被呼び出し関係図:



8.28.3.18 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 454 行目に定義があります。

8.28.3.19 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 492 行目に定義があります。

8.28.3.20 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 645 行目に定義があります。

8.28.3.21 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 656 行目に定義があります。

8.28.3.22 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 669 行目に定義があります。

8.28.3.23 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 680 行目に定義があります。

8.28.3.24 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 691 行目に定義があります。

8.28.3.25 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 819 行目に定義があります。

8.28.3.26 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 832 行目に定義があります。

8.28.3.27 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

<code>button</code>	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 796 行目に定義があります。

8.28.3.28 getShiftArrow()

```
void GgApp::Window::getShiftArrow (
    GLfloat * shift_arrow) const [inline]
```

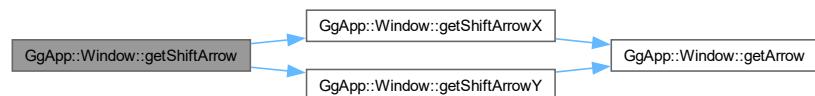
SHIFT キーを押しながら矢印キーを押したときの現在の値を得る.

引数

<code>shift_arrow</code>	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 572 行目に定義があります。

呼び出し関係図:



8.28.3.29 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

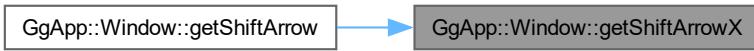
SHIFT キーを押しながら矢印キーを押したときの現在の X 値.

GgApp.h の 552 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:



8.28.3.30 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 562 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:



8.28.3.31 getSize() [1/2]

`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 402 行目に定義があります。

8.28.3.32 getSize() [2/2]

`void GgApp::Window::getSize (`
 `GLsizei * size) const [inline]`

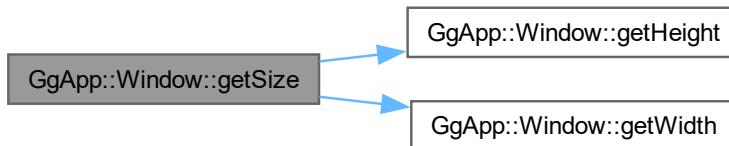
ウィンドウのサイズを得る。

引数

<code>size</code>	ウィンドウの幅と高さを格納した <code>GLsizei</code> 型の 2 要素の配列。
-------------------	--

[GgApp.h](#) の 412 行目に定義があります。

呼び出し関係図:



8.28.3.33 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

<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

平行移動量を格納した GLfloat[3] の配列のポインタ.

GgApp.h の 760 行目に定義があります。

8.28.3.34 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

<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

視点座標系で平行移動を行う GgMatrix 型の変換行列.

GgApp.h の 773 行目に定義があります。

8.28.3.35 getUserPointer()

```
void * GgApp::Window::getUserPointer () const [inline]
```

ユーザー pointer を取り出す.

戻り値

保存されているユーザ pointer.

GgApp.h の 874 行目に定義があります。

8.28.3.36 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 702 行目に定義があります。

8.28.3.37 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 713 行目に定義がります。

8.28.3.38 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 726 行目に定義がります。

8.28.3.39 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 737 行目に定義がります。

8.28.3.40 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 748 行目に定義がります。

8.28.3.41 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 362 行目に定義があります。

被呼び出し関係図:



8.28.3.42 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 441 行目に定義があります。

8.28.3.43 operator=() [1/2]

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

代入演算子は使用しない

8.28.3.44 operator=() [2/2]

```
Window & GgApp::Window::operator= (
    Window && w) [default]
```

ムーブ代入演算子はデフォルトのものを使用する

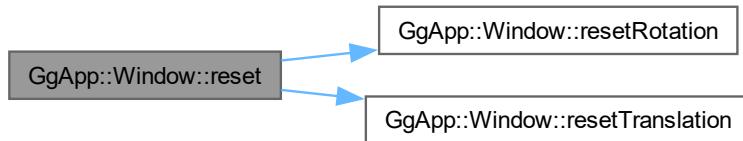
8.28.3.45 reset()

```
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 860 行目に定義があります。

呼び出し関係図:



8.28.3.46 resetRotation()

```
void GgApp::Window::resetRotation () [inline]
```

トラックボール処理を初期化する。

[GgApp.h](#) の 842 行目に定義があります。

被呼び出し関係図:



8.28.3.47 resetTranslation()

```
void GgApp::Window::resetTranslation () [inline]
```

平行移動量を初期化する。

[GgApp.h](#) の 851 行目に定義があります。

被呼び出し関係図:



8.28.3.48 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 331 行目に定義があります。

8.28.3.49 selectInterface()

```
void GgApp::Window::selectInterface (
    int no) [inline]
```

インターフェースを選択する。

引数

<i>no</i>	インターフェース番号。
-----------	-------------

[GgApp.h](#) の 469 行目に定義があります。

8.28.3.50 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

<i>flag</i>	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 300 行目に定義があります。

8.28.3.51 setKeyboardFunc()

```
void GgApp::Window::setKeyboardFunc (
    void(* func )(const Window *window, int key, int scancode, int action, int mods)) [inline]
```

ユーザ定義の keyboard 関数を設定する。

引数

<i>func</i>	ユーザ定義の keyboard 関数, キーボードの操作時に呼び出される。
-------------	---------------------------------------

[GgApp.h](#) の 904 行目に定義があります。

8.28.3.52 setMouseFunc()

```
void GgApp::Window::setMouseFunc (
    void(* func )(const Window *window, int button, int action, int mods)) [inline]
```

ユーザ定義の mouse 関数を設定する。

引数

<i>func</i>	ユーザ定義の mouse 関数、マウスボタンの操作時に呼び出される。
-------------	------------------------------------

GgApp.h の 914 行目に定義があります。

8.28.3.53 setResizeFunc()

```
void GgApp::Window::setResizeFunc (
    void(* func )(const Window *window, int width, int height)) [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 resize 関数、ウィンドウのサイズ変更時に呼び出される。
-------------	---------------------------------------

GgApp.h の 894 行目に定義があります。

8.28.3.54 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 884 行目に定義があります。

8.28.3.55 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 482 行目に定義があります。

8.28.3.56 setWheelFunc()

```
void GgApp::Window::setWheelFunc (
    void(* func )(const Window *window, double x, double y)) [inline]
```

ユーザ定義の wheel 関数を設定する。

引数

<i>func</i>	ユーザ定義の wheel 関数、マウスホイールの操作時に呼び出される。
-------------	-------------------------------------

GgApp.h の 924 行目に定義があります。

8.28.3.57 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 310 行目に定義があります。

8.28.3.58 swapBuffers()

```
void GgApp::Window::swapBuffers () const
```

カラーバッファを入れ替える。

GgApp.cpp の 491 行目に定義があります。

8.28.3.59 updateViewport()

```
void GgApp::Window::updateViewport ()
```

ビューポートのサイズを更新する。

GgApp.cpp の 509 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:

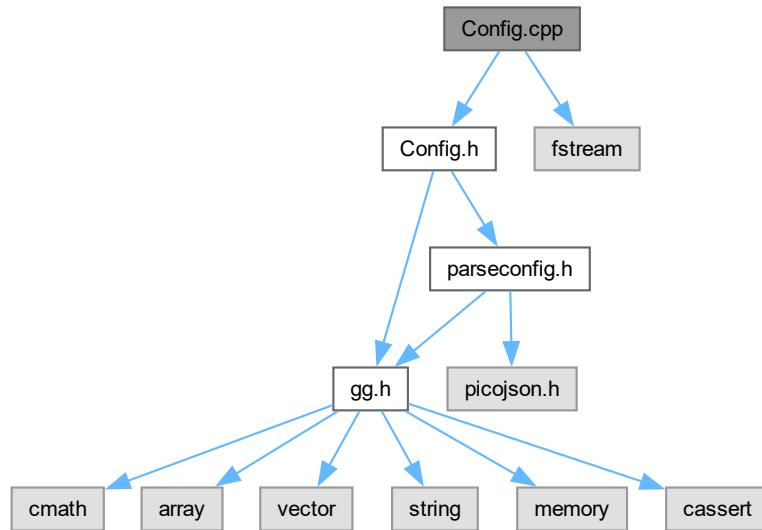
- [GgApp.h](#)
- [GgApp.cpp](#)

Chapter 9

ファイル詳解

9.1 Config.cpp ファイル

```
#include "Config.h"
#include <fstream>
Config.cpp の依存先関係図:
```



変数

- `constexpr GgSimpleShader::Light defaultLight`

9.1.1 詳解

設定構造体の実装

著者

Kohe Tokoi

日付

November 15, 2022

[Config.cpp](#) に定義があります。

9.1.2 変数詳解

9.1.2.1 defaultLight

`GgSimpleShader::Light defaultLight [constexpr]`

初期値:

```
{
    { 0.2f, 0.2f, 0.2f, 1.0f },
    { 1.0f, 1.0f, 1.0f, 0.0f },
    { 1.0f, 1.0f, 1.0f, 0.0f },
    { 0.5f, 0.5f, 1.0f, 1.0f }
}
```

[Config.cpp](#) の 14 行目に定義があります。

9.2 Config.cpp

[詳解]

```
00001
00008 #include "Config.h"
00009
00010 // 標準ライブラリ
00011 #include <fstream>
00012
00013 // デフォルトの光源データ
00014 constexpr GgSimpleShader::Light defaultLight
00015 {
00016     { 0.2f, 0.2f, 0.2f, 1.0f }, // 環境光成分
00017     { 1.0f, 1.0f, 1.0f, 0.0f }, // 拡散反射光成分
00018     { 1.0f, 1.0f, 1.0f, 0.0f }, // 鏡面反射光成分
00019     { 0.5f, 0.5f, 1.0f, 1.0f } // 光源位置
00020 };
00021
00022 //
00023 // コンストラクタ
00024 //
00025 Config::Config() :
00026     winSize{ 960, 540 },
00027     menuFont{ "Mplus1-Regular.ttf" },
00028     menuFontSize{ 20.0f },
00029     light{ defaultLight },
00030     model{ "logo.obj" },
00031 #if defined(GL_GLES_PROTOTYPES)
00032     shader{ "simple_es3.vert", "simple_es3.frag" }
00033 #else
00034     shader{ "simple.vert", "simple.frag" }
00035 #endif
00036 {
```

```
00037 }
00038
00039 // ファイルから構成データを読み込むコンストラクタ
00040 // Config::Config(const std::string& filename) :
00041 // Config{}
00042 Config::Config(const std::string& filename) :
00043     Config{}
00044 {
00045     // 構成ファイルが読み込めなかったらデフォルト値の構成ファイルを作る
00046     if (!load(filename)) save(filename);
00047 }
00048
00049 //
00050 // デストラクタ
00051 //
00052 Config::~Config()
00053 {
00054 }
00055
00056 //
00057 // 設定ファイルを読み込む
00058 //
00059 bool Config::load(const std::string& filename)
00060 {
00061     // 構成ファイルを開く
00062     std::ifstream file{ Utf8ToTChar(filename) };
00063
00064     // 開けなかったらエラー
00065     if (!file) return false;
00066
00067     // JSON の読み込み
00068     picojson::value value;
00069     file >> value;
00070     file.close();
00071
00072     // 構成データの取り出し
00073     const auto& object{ value.get<picojson::object>() };
00074
00075     //
00076     // 構成データの読み込み
00077     //
00078
00079     // ウィンドウサイズ
00080     getValue(object, "window_size", winSize);
00081
00082     // メニューフォント
00083     getString(object, "menu_font", menuFont);
00084
00085     // メニューフォントサイズ
00086     getValue(object, "menu_font_size", menuFontSize);
00087
00088     // 光源
00089     getVector(object, "ambient", light.ambient);
00090     getVector(object, "diffuse", light.diffuse);
00091     getVector(object, "specular", light.specular);
00092     getVector(object, "position", light.position);
00093
00094     // モデル
00095     getString(object, "model", model);
00096
00097     // シェーダ
00098     getString(object, "shader", shader);
00099
00100    // オブジェクトが空だったらエラー
00101    if (object.empty()) return false;
00102
00103    return true;
00104 }
00105
00106 //
00107 // 設定ファイルを書き出す
00108 //
00109 bool Config::save(const std::string& filename) const
00110 {
00111     // 構成ファイルを開く
00112     std::ofstream file{ Utf8ToTChar(filename) };
00113
00114     // 開けなかったらエラー
00115     if (!file) return false;
00116
00117     // 構成データの書き出しに使うオブジェクト
00118     picojson::object object;
00119
00120     //
00121     // 構成データの書き出し
00122     //
00123 }
```

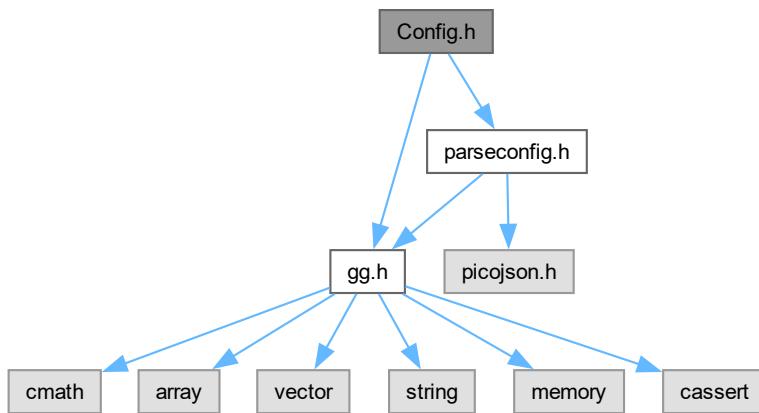
```

00124 // ウィンドウサイズ
00125 setValue(object, "window_size", winSize);
00126
00127 // メニューフォント
00128 setString(object, "menu_font", menuFont);
00129
00130 // メニューフォントサイズ
00131 setValue(object, "menu_font_size", menuFontSize);
00132
00133 // 光源
00134 setVector(object, "ambient", light.ambient);
00135 setVector(object, "diffuse", light.diffuse);
00136 setVector(object, "specular", light.specular);
00137 setVector(object, "position", light.position);
00138
00139 // モデル
00140 setString(object, "model", model);
00141
00142 // シエーダ
00143 setString(object, "shader", shader);
00144
00145 // 構成出データをシリализして JSON で保存
00146 picojson::value v{ object };
00147 file << v.serialize(true);
00148 file.close();
00149
00150 return true;
00151 }

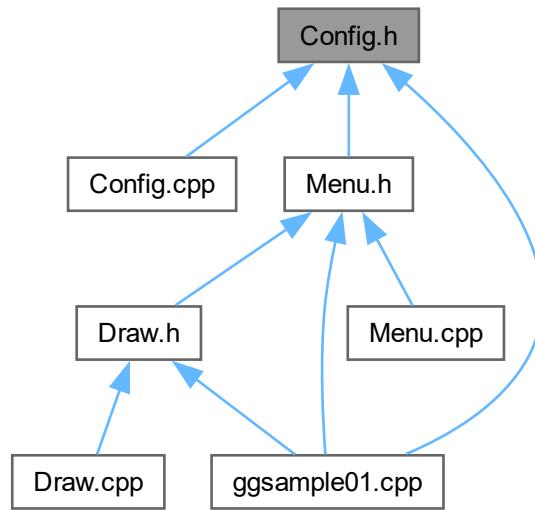
```

9.3 Config.h ファイル

```
#include "gg.h"
#include "parseconfig.h"
Config.h の依存先関係図:
```



被依存関係図:



クラス

- class [Config](#)

9.3.1 詳解

構成データクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Config.h](#) に定義があります。

9.4 Config.h

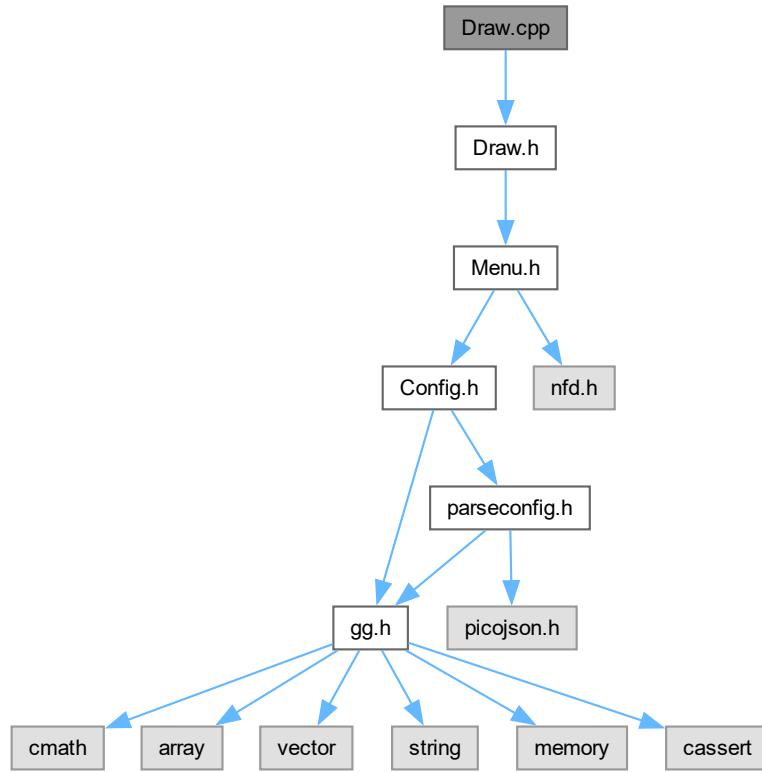
[詳解]

```
00001 #pragma once
00002
00010
00011 // 補助プログラム
00012 #include "gg.h"
00013 using namespace gg;
00014
00015 // 構成ファイルの読み取り補助
00016 #include "parseconfig.h"
00017
00021 class Config
00022 {
00023     // メニュークラスから参照する
00024     friend class Menu;
00025
00026     // ウィンドウサイズ
00027     std::array<GLsizei, 2> winSize;
00028
00029     // メニューフォント名
00030     std::string menuFont;
00031
00032     // メニューフォントサイズ
00033     float menuFontSize;
00034
00035     // 光源
00036     GgSimpleShader::Light light;
00037
00038     // 形状ファイル名
00039     std::string model;
00040
00041     // シェーダのソースファイル名
00042     std::array<std::string, 3> shader;
00043
00044 public:
00045
00049     Config();
00050
00054     Config(const std::string& filename);
00055
00059     virtual ~Config();
00060
00064     auto getWidth() const
00065     {
00066         return winSize[0];
00067     }
00068
00072     auto getHeight() const
00073     {
00074         return winSize[1];
00075     }
00076
00080     bool load(const std::string& filename);
00081
00085     bool save(const std::string& filename) const;
00086 };
```

9.5 Draw.cpp ファイル

```
#include "Draw.h"
```

Draw.cpp の依存先関係図:



9.5.1 詳解

図形の描画クラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

[Draw.cpp](#) に定義があります。

9.6 Draw.cpp

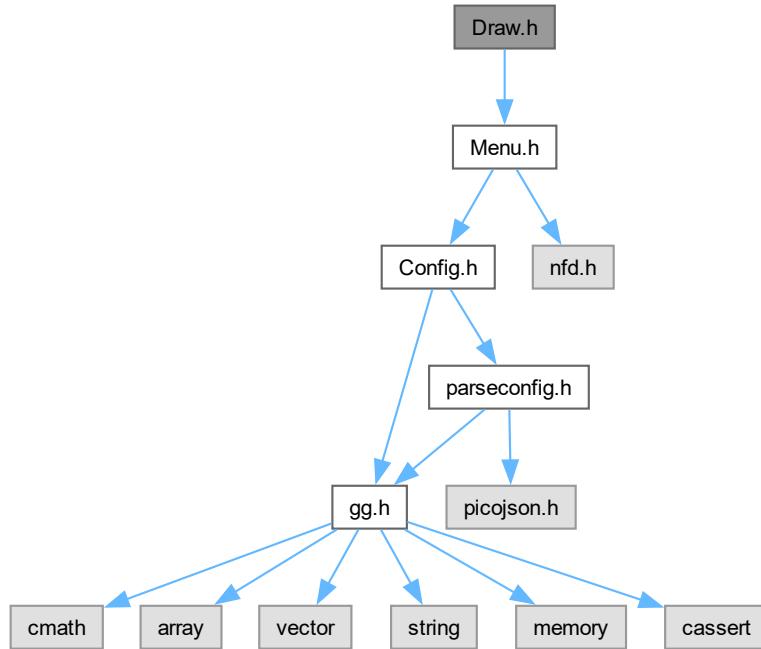
[詳解]

```
00001
00008 #include "Draw.h"
00009
00010 //
00011 // コンストラクタ
00012 //
00013 Draw::Draw(const Menu& menu) :
00014     light{ *menu.light.get() },
00015     model{ *menu.model.get() },
00016     shader{ *menu.shader.get() }
00017 {
00018     // 背景色を設定する
00019     glClearColor(0.1f, 0.2f, 0.3f, 0.0f);
00020
00021     // 隠面消去処理を設定する
00022     glEnable(GL_DEPTH_TEST);
00023     glEnable(GL_CULL_FACE);
00024 }
00025
00026 //
00027 // デストラクタ
00028 //
00029 Draw::~Draw()
00030 {
00031 }
00032
00033 //
00034 // 描画する
00035 //
00036 void Draw::draw(const GgMatrix& mp, const GgMatrix& mv) const
00037 {
00038     // シェーダプログラムを指定する
00039     shader.use(mp, mv, light);
00040
00041     // 図形を描画する
00042     model.draw();
00043 }
```

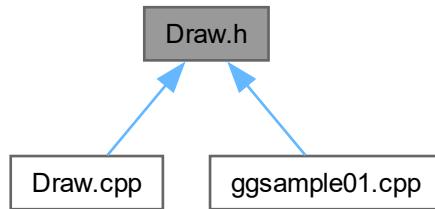
9.7 Draw.h ファイル

```
#include "Menu.h"
```

Draw.h の依存先関係図:



被依存関係図:



クラス

- class **Draw**

9.7.1 詳解

図形の描画クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Draw.h](#) に定義があります。

9.8 Draw.h

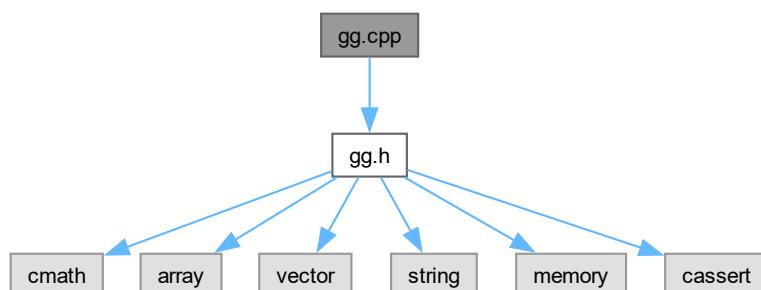
[詳解]

```
00001 #pragma once
00002
00010
00011 // メニュー
00012 #include "Menu.h"
00013
00017 class Draw
00018 {
00019     // 光源
00020     const GgSimpleShader::LightBuffer& light;
00021
00022     // モデル
00023     const GgSimpleObj& model;
00024
00025     // シェーダ
00026     const GgSimpleShader& shader;
00027
00028 public:
00029
00033     Draw(const Menu& menu);
00034
00038     virtual ~Draw();
00039
00043     void draw(const GgMatrix& mp, const GgMatrix& mv) const;
00044 };
```

9.9 gg.cpp ファイル

#include "gg.h"

gg.cpp の依存先関係図:



9.9.1 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

July 17, 2025

gg.cpp に定義があります。

9.10 gg.cpp

[詳解]

```
00001 /*
00002
00003 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) 2011-2025 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
00010 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00011 copies or substantial portions of the Software.
00012
00013 The above copyright notice and this permission notice shall be included in
00014 all copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
00019 KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
00020 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
00021 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00022
00023 */
00024
00032 #include "gg.h"
00033
00035
00036 // 標準ライブラリ
00037 #include <cfloat>
00038 #include <cstdlib>
00039 #include <iostream>
00040 #include <fstream>
00041 #include <sstream>
00042 #include <limits>
00043 #include <map>
00044
00046 #define READ_TEXTURE_COORDINATE_FROM_OBJ 0
00047
00048 // Windows (Visual Studio) のとき
00049 #if defined(_MSC_VER)
00050 // リリースビルドではコンソールを出さない
00051 # if !defined(_DEBUG)
00052 # pragma comment(linker, "/subsystem:\\"windows\\" /entry:\\"mainCRTStartup\\")
00053 # endif
00054 // リンクするライブラリ
00055 # pragma comment(lib, "lib\\\" GLFW3_PLATFORM \"\\\" GLFW3_CONFIGURATION \"\\\" glfw3.lib")
00056
00057 //
00058 // 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
00059 //
00060 // Original author: mt-u
00061 // Copyright (c) 2013 mt-u
00062 // https://gist.github.com/mt-u/6878251
00063 //
```

```

00064 // Modified by: Kohe Tokoi
00065 //
00066
00067 //
00068 // UTF-8 文字列を CString に変換する
00069 //
00070 pathString Utf8ToTChar(const std::string& string)
00071 {
00072     // 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
00073     const INT length{ MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, NULL, 0) };
00074
00075     // 変換結果の格納に必要な長さのメモリを確保する
00076     std::vector<WCHAR> utf16(length);
00077
00078     // UTF-8 文字列を UTF-16 に変換する
00079     MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, utf16.data(), length);
00080
00081     // 変換した文字列を CString にして返す
00082     return CString{ utf16.data(), static_cast<int>(wcslen(utf16.data())) };
00083 }
00084
00085 //
00086 // CString を UTF-8 文字列に変換する
00087 //
00088 std::string TCharToUtf8(const pathString& cstring)
00089 {
00090     // 与えられた文字列を一旦 UTF-16 に変換する
00091     CStringW cstringw{ cstring };
00092
00093     // UTF-16 を UTF-8 に変換した後の文字列の長さを求める
00094     const INT length{ WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, NULL, 0, NULL, NULL) };
00095
00096     // 変換結果の格納に必要な長さのメモリを確保する
00097     std::vector<CHAR> utf8(length);
00098
00099     // UTF-16 を UTF-8 に変換する
00100     WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, utf8.data(), length, NULL, NULL);
00101
00102     // 変換した文字列を std::string にして返す
00103     return std::string{ utf8.data(), strlen(utf8.data()) };
00104 }
00105 #endif
00106
00107 // OpenGL 3.2 の API のエントリポイント
00108 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
00109 PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
00110 PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
00111 PFNGLACTIVETEXTUREPROCP glActiveTexture;
00112 PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
00113 PFNGLATTACHSHADERPROC glAttachShader;
00114 PFNGLBEGINCONDITIONALRENDERNVPROC glBeginConditionalRenderNV;
00115 PFNGLBEGINCONDITIONALRENDERPROC glBeginConditionalRender;
00116 PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00117 PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00118 PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00119 PFNGLBEGINQUERYPROC glBeginQuery;
00120 PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
00121 PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
00122 PFNGLBINDBUFFERBASEPROC glBindBufferBase;
00123 PFNGLBINDBUFFERPROC glBindBuffer;
00124 PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
00125 PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
00126 PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
00127 PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
00128 PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
00129 PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
00130 PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
00131 PFNGLBINDIMAGETEXTURERESPROC glBindImageTextures;
00132 PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00133 PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00134 PFNGLBINDERRENDERBUFFERPROC glBindRenderbuffer;
00135 PFNGLBINDSAMPLERPROC glBindSampler;
00136 PFNGLBINDSAMPLERSPROC glBindSamplers;
00137 PFNGLBINDTEXTUREPROC glBindTexture;
00138 PFNGLBINDTEXTURESPROC glBindTextures;
00139 PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
00140 PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
00141 PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
00142 PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
00143 PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
00144 PFNGLBLENDBARRIEKHRPROC glBindBarrierKHR;
00145 PFNGLBLENDBARRIERNVPROC glBindBarrierNV;
00146 PFNGLBLENDCOLORPROC glBindColor;
00147 PFNGLBLENDEQUATIONIARBPROC glBindEquationiARB;
00148 PFNGLBLENDEQUATIONIPROC glBindEquationi;
00149 PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00150 PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;

```

```
00151 PFNGLBLENEQUATIONSEPARATEIPROC glBlendEquationSeparatei;
00152 PFNGLBLENEQUATIONSEPARATEPROC glBlendEquationSeparate;
00153 PFNGLBLENDFUNCARBPROC glBlendFunciARB;
00154 PFNGLBLENDFUNCIPROC glBlendFunci;
00155 PFNGLBLENDFUNCPROC glBlendFunc;
00156 PFNGLBLENDFUNCSEPARATEIARBPROC glBlendFuncSeparateiARB;
00157 PFNGLBLENDFUNCSEPARATEIPROC glBlendFuncSeparatei;
00158 PFNGLBLENDFUNCSEPARATEPROC glBlendFuncSeparate;
00159 PFNGLBLENDPARAMETERINVPROC glBlendParameteriNV;
00160 PFNGLBLITFRAMEBUFFERPROC glBlitFramebuffer;
00161 PFNGLBLITNAMEDFRAMEBUFFERPROC glBlitNamedFramebuffer;
00162 PFNGLBUFFERADDRESSRANGEVPROC glBufferAddressRangeNV;
00163 PFNGLBUFFERDATAPROC glBufferData;
00164 PFNGLBUFFERPAGECOMMITMENTARBPROC glBufferPageCommitmentARB;
00165 PFNGLBUFFERSTORAGEPROC glBufferStorage;
00166 PFNGLBUFFERSUBDATAPROC glBufferSubData;
00167 PFNGLCALLCOMMANDLISTNVPROC glCallCommandListNV;
00168 PFNGLCHECKFRAMEBUFFERSTATUSPROC glCheckFramebufferStatus;
00169 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glCheckNamedFramebufferStatusEXT;
00170 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glCheckNamedFramebufferStatus;
00171 PFNGLCLAMPCOLORPROC glClampColor;
00172 PFNGLCLEARBUFFERDATAPROC glClearBufferData;
00173 PFNGLCLEARBUFFERFIPROC glClearBufferfi;
00174 PFNGLCLEARBUFFERFVPROC glClearBufferfv;
00175 PFNGLCLEARBUFFERIVPROC glClearBufferiv;
00176 PFNGLCLEARBUFFERSUBDATAPROC glClearBufferSubData;
00177 PFNGLCLEARBUFFERUIVPROC glClearBufferui;
00178 PFNGLCLEARCOLORPROC glClearColor;
00179 PFNGLCLEARDEPTHFPROC glClearDepthf;
00180 PFNGLCLEARDEPTHPROC glClearDepth;
00181 PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
00182 PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
00183 PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
00184 PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
00185 PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
00186 PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
00187 PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
00188 PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferui;
00189 PFNGLCLEARPROC glClear;
00190 PFNGLCLEARSTENCILPROC glClearStencil;
00191 PFNGLCLEARTEXIMAGEPROC glClearTexImage;
00192 PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
00193 PFNGLCLIENTATTRIBDEFAULTEXTPROC glClientAttribDefaultEXT;
00194 PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
00195 PFNGLCLIPCONTROLPROC glClipControl;
00196 PFNGLCOLORFORMATNVPROC glColorFormatNV;
00197 PFNGLCOLORMASKIPROC glColorMaski;
00198 PFNGLCOLORMASKPROC glColorMask;
00199 PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
00200 PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
00201 PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
00202 PFNGLCOMPILESHADERPROC glCompileShader;
00203 PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
00204 PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
00205 PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
00206 PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
00207 PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
00208 PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
00209 PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
00210 PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
00211 PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
00212 PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
00213 PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
00214 PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
00215 PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
00216 PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
00217 PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
00218 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1D;
00219 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
00220 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00221 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00222 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00223 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00224 PFNGLCONSERVATIVEASTERPARAMETERFNVPROC glConservativeRasterParameterfNV;
00225 PFNGLCONSERVATIVEASTERPARAMETERINVPROC glConservativeRasterParameteriNV;
00226 PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
00227 PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
00228 PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
00229 PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
00230 PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
00231 PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
00232 PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
00233 PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
00234 PFNGLCOPYPATHNVPROC glCopyPathNV;
00235 PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
00236 PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
00237 PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
```

```
00238 PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
00239 PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
00240 PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
00241 PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
00242 PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
00243 PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
00244 PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
00245 PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
00246 PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
00247 PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
00248 PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationNV;
00249 PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTableNV;
00250 PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancedNV;
00251 PFNGLCOVERFILLPATHNVPROC glCoverFillPathNV;
00252 PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancedNV;
00253 PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathNV;
00254 PFNGLCREATEBUFFERSPROC glCreateBuffers;
00255 PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
00256 PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
00257 PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
00258 PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
00259 PFNGLCREATEPROGRAMPROC glCreateProgram;
00260 PFNGLCREATEQUERIESPROC glCreateQueries;
00261 PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
00262 PFNGLCREATESAMPLERSPROC glCreateSamplers;
00263 PFNGLCREATESHADERPROC glCreateShader;
00264 PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
00265 PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
00266 PFNGLCREATESTATESNVPROC glCreateStatesNV;
00267 PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCleventARB;
00268 PFNGLCREATETEXTURESPROC glCreateTextures;
00269 PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
00270 PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
00271 PFNGLCULLFACEPROC glCullFace;
00272 PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
00273 PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
00274 PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
00275 PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
00276 PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
00277 PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
00278 PFNGLDELETEBUFFERSPROC glDeleteBuffers;
00279 PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
00280 PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
00281 PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
00282 PFNGLDELETEPATHSNVPROC glDeletePathsNV;
00283 PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
00284 PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
00285 PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
00286 PFNGLDELETEPROGRAMPROC glDeleteProgram;
00287 PFNGLDELETEQUERIESPROC glDeleteQueries;
00288 PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
00289 PFNGLDELETESAMPLERSPROC glDeleteSamplers;
00290 PFNGLDELETESHADERPROC glDeleteShader;
00291 PFNGLDELETESTATESNVPROC glDeleteStatesNV;
00292 PFNGLDELETESYNCPROC glDeleteSync;
00293 PFNGLDELETETEXTURESPROC glDeleteTextures;
00294 PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
00295 PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
00296 PFNGLDEPTHFUNCPROC glDepthFunc;
00297 PFNGLDEPTHMASKPROC glDepthMask;
00298 PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
00299 PFNGLDEPTHRANGEFPROC glDepthRangef;
00300 PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
00301 PFNGLDEPTHRANGEPROC glDepthRange;
00302 PFNGLDETACHSHADERPROC glDetachShader;
00303 PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
00304 PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
00305 PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
00306 PFNGLDISABLEIPROC glDisablei;
00307 PFNGLDISABLEPROC glDisable;
00308 PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexAttribArrayAttribEXT;
00309 PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexAttribArrayAttrib;
00310 PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexAttribArrayEXT;
00311 PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArrayAttrib;
00312 PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
00313 PFNGLDISPATCHCOMPUTEINDIRECTPROC glDispatchComputeIndirect;
00314 PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
00315 PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
00316 PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
00317 PFNGLDRAWARRAYSINSTANCEBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
00318 PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
00319 PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
00320 PFNGLDRAWARRAYSPROC glDrawArrays;
00321 PFNGLDRAWBUFFERPROC glDrawBuffer;
00322 PFNGLDRAWBUFFERSPROC glDrawBuffers;
00323 PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
00324 PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
```

```
00325 PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
00326 PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
00327 PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
00328 PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
00329 PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
00330 PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
00331 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC glDrawElementsInstancedBaseVertexBaseInstance;
00332 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
00333 PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
00334 PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
00335 PFNGLDRAWELEMENTSPROC glDrawElements;
00336 PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00337 PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00338 PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
00339 PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
00340 PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
00341 PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00342 PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00343 PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00344 PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00345 PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00346 PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00347 PFNGLENABLEIPROC glEnablei;
00348 PFNGLENABLEPROC glEnable;
00349 PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexAttribArrayAttribEXT;
00350 PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00351 PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00352 PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArrayArray;
00353 PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
00354 PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
00355 PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
00356 PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
00357 PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
00358 PFNGLENDQUERYPROC glEndQuery;
00359 PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
00360 PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
00361 PFNGLFENCESYNCNPROC glFenceSync;
00362 PFNGLFINISHPROC glFinish;
00363 PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00364 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00365 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
00366 PFNGLFLUSHPROC glFlush;
00367 PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00368 PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00369 PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00370 PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00371 PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00372 PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00373 PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
00374 PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC glFramebufferSampleLocationsfvARB;
00375 PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
00376 PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00377 PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00378 PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00379 PFNGLFRAMEBUFFERTEXTUREARPROC glFramebufferTextureARB;
00380 PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00381 PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
00382 PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00383 PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00384 PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
00385 PFNGLFRONTFACEPROC glFrontFace;
00386 PFNGLGENBUFFERSPROC glGenBuffers;
00387 PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
00388 PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
00389 PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
00390 PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
00391 PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
00392 PFNGLGENPATHSNVPROC glGenPathsNV;
00393 PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
00394 PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
00395 PFNGLGENQUERIESPROC glGenQueries;
00396 PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
00397 PFNGLGENSAMPLERSPROC glGenSamplers;
00398 PFNGLGENTEXTURESPROC glGenTextures;
00399 PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
00400 PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
00401 PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
00402 PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
00403 PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
00404 PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
00405 PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
00406 PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
00407 PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
00408 PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
00409 PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
00410 PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
00411 PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
```

```

00412 PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
00413 PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
00414 PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
00415 PFNGLGETBOOLEANVPROC glGetBooleanv;
00416 PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00417 PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00418 PFNGLGETBUFFERPARAMETERUI64VNVPROC glGetBufferParameterui64vNV;
00419 PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00420 PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00421 PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00422 PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
00423 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00424 PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
00425 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00426 PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00427 PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00428 PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00429 PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00430 PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00431 PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
00432 PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00433 PFNGLGETDOUBLEVPROC glGetDoublev;
00434 PFNGLGETERRORPROC glGetError;
00435 PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
00436 PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
00437 PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
00438 PFNGLGETFLOATI_VPROC glGetFloati_v;
00439 PFNGLGETFLOATVPROC glGetFloatv;
00440 PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
00441 PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
00442 PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
00443 PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
00444 PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
00445 PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
00446 PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
00447 PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
00448 PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
00449 PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
00450 PFNGLGETINTEGER64VPROC glGetInteger64v;
00451 PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00452 PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00453 PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00454 PFNGLGETINTEGERUI64VNVPROC glGetIntegerui64vNV;
00455 PFNGLGETINTEGERVPROC glGetIntegerv;
00456 PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
00457 PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
00458 PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
00459 PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
00460 PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
00461 PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
00462 PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGenfvEXT;
00463 PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
00464 PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
00465 PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
00466 PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
00467 PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
00468 PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
00469 PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIiivEXT;
00470 PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
00471 PFNGLGETMULTITEXPARAMETERIVEXTPROC glGetMultiTexParameterivEXT;
00472 PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
00473 PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
00474 PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
00475 PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC glGetNamedBufferParameterui64vNV;
00476 PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
00477 PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
00478 PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
00479 PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
00480 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC glGetNamedFramebufferAttachmentParameterivEXT;
00481 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
00482 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
00483 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
00484 PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
00485 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVECTPROC glGetNamedProgramLocalParameterdveEXT;
00486 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00487 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIiivEXT;
00488 PFNGLGETNAMEDPROGRAMLOCALPARAMETERUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00489 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00490 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00491 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00492 PFNGLGETNAMEDSTRINGARBPROC glGetStringARB;
00493 PFNGLGETNAMEDSTRINGIVARBPROC glGetStringivARB;
00494 PFNGLGETNCOMPRESSEDTEXTIMAGEARBPROC glGetnCompressedTexImageARB;
00495 PFNGLGETNCOMPRESSEDTEXTIMAGEPROC glGetnCompressedTexImage;
00496 PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00497 PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00498 PFNGLGETNTEXIMAGEPROC glGetnTexImage;

```

```
00499 PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
00500 PFNGLGETNUNIFORMDVPROC glGetnUniformdv;
00501 PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
00502 PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00503 PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00504 PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
00505 PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
00506 PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00507 PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
00508 PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
00509 PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
00510 PFNGLGETOBJECTLABELPROC glGetObjectLabel;
00511 PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
00512 PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
00513 PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
00514 PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
00515 PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
00516 PFNGLGETPATHMETRICRANGEPROC glGetPathMetricRangeNV;
00517 PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
00518 PFNGLGETPATHPARAMETERFVNPROC glGetPathParameterfvNV;
00519 PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
00520 PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
00521 PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
00522 PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
00523 PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
00524 PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
00525 PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
00526 PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
00527 PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
00528 PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
00529 PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
00530 PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
00531 PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
00532 PFNGLGETPOINTERI_VEXTPROC glGetPointeri_vEXT;
00533 PFNGLGETPOINTERVPROC glGetPointerv;
00534 PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
00535 PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
00536 PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
00537 PFNGLGETPROGRAMIVPROC glGetProgramiv;
00538 PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
00539 PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
00540 PFNGLGETPROGRAMRESOURCEFVNPROC glGetProgramResourcefvNV;
00541 PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00542 PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00543 PFNGLGETPROGRAMRESOURCELLOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00544 PFNGLGETPROGRAMRESOURCELLOCATIONPROC glGetProgramResourceLocation;
00545 PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00546 PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00547 PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
00548 PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00549 PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00550 PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00551 PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00552 PFNGLGETQUERYIVPROC glGetQueryiv;
00553 PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
00554 PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
00555 PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00556 PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00557 PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00558 PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00559 PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00560 PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
00561 PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
00562 PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
00563 PFNGLGETSHADERIVPROC glGetShaderiv;
00564 PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
00565 PFNGLGETSHADERSOURCEPROC glGetShaderSource;
00566 PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
00567 PFNGLGETSTRINGIPROC glGetStringi;
00568 PFNGLGETSTRINGPROC glGetString;
00569 PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
00570 PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
00571 PFNGLGETSYNCIVPROC glGetSynciv;
00572 PFNGLGETTEXIMAGEPROC glGetTexImage;
00573 PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00574 PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00575 PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00576 PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00577 PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00578 PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
00579 PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00580 PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
00581 PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00582 PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00583 PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00584 PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
00585 PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
```

```

00586 PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
00587 PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
00588 PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
00589 PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIiivEXT;
00590 PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiiv;
00591 PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
00592 PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
00593 PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIiivEXT;
00594 PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00595 PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00596 PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
00597 PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00598 PFNGLGETTRANSFORMFEEDBACKI64_VPROC glGetTransformFeedbacki64_v;
00599 PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00600 PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
00601 PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00602 PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00603 PFNGLGETUNIFORMDVPROC glGetUniformdv;
00604 PFNGLGETUNIFORMFVPROC glGetUniformfv;
00605 PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00606 PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
00607 PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
00608 PFNGLGETUNIFORMIVPROC glGetUniformiv;
00609 PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
00610 PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineui;
00611 PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
00612 PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
00613 PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
00614 PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00615 PFNGLGETVERTEXARRAYINDEXEDDIVPROC glGetVertexArrayIndexeddiv;
00616 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00617 PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00618 PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00619 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00620 PFNGLGETVERTEXARRAYPOINTERVECTPROC glGetVertexArrayPointervEXT;
00621 PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribArraybdv;
00622 PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribArraybfv;
00623 PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribArraybiv;
00624 PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribArrayui;
00625 PFNGLGETVERTEXATTRIBIVPROC glGetVertexAttribArraybiv;
00626 PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribArrayldv;
00627 PFNGLGETVERTEXATTRIBL64VNVPROC glGetVertexAttribArraybli64vNV;
00628 PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribArraybli64vARB;
00629 PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribArraybli64vNV;
00630 PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribArrayPointerv;
00631 PFNGLGETVKPROCAADDRNVPROC glGetVkProcAddrNV;
00632 PFNGLHINTPROC glHint;
00633 PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00634 PFNGLINSEVENTMARKEREXTPROC glInsertEventMarkerEXT;
00635 PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00636 PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
00637 PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
00638 PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
00639 PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
00640 PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
00641 PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
00642 PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00643 PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00644 PFNGLISBUFFERPROC glIsBuffer;
00645 PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
00646 PFNGLCOMMANDLISTNVPROC glIsCommandListNV;
00647 PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00648 PFNGLISENABLEDIPROC glIsEnabledi;
00649 PFNGLISENABLEDPROC glIsEnabled;
00650 PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00651 PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
00652 PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
00653 PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00654 PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00655 PFNGLISPATHNVPROC glIsPathNV;
00656 PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00657 PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00658 PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00659 PFNGLISPROGRAMPROC glIsProgram;
00660 PFNGLISQUERYPROC glIsQuery;
00661 PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00662 PFNGLISSAMPLERPROC glIsSampler;
00663 PFNGLISSHADERPROC glIsShader;
00664 PFNGLISSTATENVPROC glIsStateNV;
00665 PFNGLISSYNCPROC glIsSync;
00666 PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00667 PFNGLISTTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00668 PFNGLISTTEXTUREPROC glIsTexture;
00669 PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00670 PFNGLISVERTEXARRAYPROC glIsVertexArray;
00671 PFNGLLABELOBJECTTEXTPROC glLabelObjectEXT;
00672 PFNGLLINEWIDTHPROC glLineWidth;

```

```
00673 PFNGLLINKPROGRAMPROC glLinkProgram;
00674 PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
00675 PFNGLLOGICOPPROC glLogicOp;
00676 PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
00677 PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
00678 PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
00679 PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
00680 PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
00681 PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
00682 PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
00683 PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
00684 PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
00685 PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
00686 PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
00687 PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
00688 PFNGLMAPBUFFERPROC glMapBuffer;
00689 PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
00690 PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
00691 PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
00692 PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
00693 PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
00694 PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00695 PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
00696 PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
00697 PFNGLMATRIXLOADDEXTPROC glMatrixLoaddEXT;
00698 PFNGLMATRIXLOADFEXTPROC glMatrixLoadfEXT;
00699 PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00700 PFNGLMATRIXLOADTRANPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
00701 PFNGLMATRIXLOADTRANPOSEDEXTPROC glMatrixLoadTransposedEXT;
00702 PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
00703 PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
00704 PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
00705 PFNGLMATRIXMULTDEXTPROC glMatrixMultdEXT;
00706 PFNGLMATRIXMULTFEXTPROC glMatrixMultfEXT;
00707 PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
00708 PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
00709 PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefEXT;
00710 PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
00711 PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
00712 PFNGLMATRIXXPUSHEXTPROC glMatrixPushEXT;
00713 PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedEXT;
00714 PFNGLMATRIXROTATEFEXTPROC glMatrixRotatefEXT;
00715 PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
00716 PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
00717 PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
00718 PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
00719 PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
00720 PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
00721 PFNGLMEMORYBARRIERPROC glMemoryBarrier;
00722 PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
00723 PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
00724 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
00725 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
00726 PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC glMultiDrawArraysIndirectCountARB;
00727 PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00728 PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
00729 PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00730 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00731 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00732 PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC glMultiDrawElementsIndirectCountARB;
00733 PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00734 PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00735 PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00736 PFNGLMULTITEXCOORDINTEREXTPROC glMultiTexCoordPointerEXT;
00737 PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00738 PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00739 PFNGLMULTITEXENVVEXTPROC glMultiTexEnvivEXT;
00740 PFNGLMULTITEXENVVEXTPROC glMultiTexEnvivEXT;
00741 PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00742 PFNGLMULTITEXGENDEVENTPROC glMultiTexGendvEXT;
00743 PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
00744 PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
00745 PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
00746 PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
00747 PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
00748 PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
00749 PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
00750 PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
00751 PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
00752 PFNGLMULTITEXPARAMETERIEEXTPROC glMultiTexParameteriEXT;
00753 PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
00754 PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
00755 PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
00756 PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
00757 PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
00758 PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
00759 PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
```

```

00760 PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
00761 PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
00762 PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
00763 PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
00764 PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
00765 PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
00766 PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubDataEXT;
00767 PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
00768 PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
00769 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
00770 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
00771 PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
00772 PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC glNamedFramebufferParameteri;
00773 PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
00774 PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00775 PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00776 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glNamedFramebufferSampleLocationsfvARB;
00777 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNPROC glNamedFramebufferSampleLocationsfvNV;
00778 PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00779 PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00780 PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00781 PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00782 PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00783 PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00784 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00785 PFNGLNAMEDFRAMEBUFFERTEXTUREREPROC glNamedFramebufferTexture;
00786 PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00787 PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00788 PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fvEXT;
00789 PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00790 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IBEXTPROC glNamedProgramLocalParameterI4iEXT;
00791 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00792 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00793 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uivEXT;
00794 PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00795 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
00796 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
00797 PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00798 PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00799 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERGEEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00800 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00801 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00802 PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00803 PFNGLNAMESTRINGARBPROC glNamedStringARB;
00804 PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00805 PFNGLOBJECTLABELPROC glObjectLabel;
00806 PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
00807 PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
00808 PFNGLPATCHPARAMETERIPROC glPatchParameteri;
00809 PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00810 PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00811 PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00812 PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00813 PFNGLPATHGLYPHINDEXEXARRAYNVPROC glPathGlyphIndexArrayNV;
00814 PFNGLPATHGLYPHINDEXDEXRANGENVPROC glPathGlyphIndexRangeNV;
00815 PFNGLPATHGLYPHRANGEENVPROC glPathGlyphRangeNV;
00816 PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
00817 PFNGLPATHMEMORYGLYPHINDEXEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
00818 PFNGLPATHPARAMETERFNVPROC glPathParameterfvNV;
00819 PFNGLPATHPARAMETERFVNPROC glPathParameterfvNV;
00820 PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
00821 PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
00822 PFNGLPATHSTENCILDEPTHHOFFSETNVPROC glPathStencilDepthOffsetNV;
00823 PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00824 PFNGLPATHSTRINGNVPROC glPathStringNV;
00825 PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00826 PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00827 PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00828 PFNGLPIXELSTOREFPROC glPixelStoref;
00829 PFNGLPIXELSTOREIPROC glPixelStorei;
00830 PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
00831 PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
00832 PFNGLPOINTPARAMETERIPROC glPointParameteri;
00833 PFNGLPOINTPARAMETERIPROC glPointParameteri;
00834 PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00835 PFNGLPOINTSIZEPROC glPointSize;
00836 PFNGLPOLYGONMODEPROC glPolygonMode;
00837 PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00838 PFNGLPOLYGONOFFSETPROC glPolygonOffset;
00839 PFNGLPOPDEBUGGROUPPROC glPopDebugGroup;
00840 PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00841 PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
00842 PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
00843 PFNGLPROGRAMBINARYPROC glProgramBinary;
00844 PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
00845 PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;

```

```
00846 PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
00847 PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
00848 PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
00849 PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
00850 PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dv;
00851 PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
00852 PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
00853 PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
00854 PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
00855 PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
00856 PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;
00857 PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
00858 PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
00859 PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniform1iEXT;
00860 PFNGLPROGRAMUNIFORM1IPROC glProgramUniform1i;
00861 PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
00862 PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
00863 PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
00864 PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniform1ui64NV;
00865 PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
00866 PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
00867 PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
00868 PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
00869 PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
00870 PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
00871 PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
00872 PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
00873 PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
00874 PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
00875 PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
00876 PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
00877 PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
00878 PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
00879 PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
00880 PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
00881 PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
00882 PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
00883 PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
00884 PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
00885 PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
00886 PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
00887 PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
00888 PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
00889 PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
00890 PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
00891 PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
00892 PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
00893 PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
00894 PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
00895 PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
00896 PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
00897 PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
00898 PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
00899 PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
00900 PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
00901 PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
00902 PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
00903 PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
00904 PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
00905 PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
00906 PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
00907 PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
00908 PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
00909 PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
00910 PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
00911 PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
00912 PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
00913 PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
00914 PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
00915 PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
00916 PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
00917 PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
00918 PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
00919 PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
00920 PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
00921 PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
00922 PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
00923 PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
00924 PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
00925 PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
00926 PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
00927 PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
00928 PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
00929 PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
00930 PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
00931 PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
00932 PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
```

```

00933 PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
00934 PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
00935 PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
00936 PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
00937 PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
00938 PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
00939 PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
00940 PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
00941 PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
00942 PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
00943 PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
00944 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
00945 PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
00946 PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
00947 PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dvEXT;
00948 PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
00949 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
00950 PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
00951 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
00952 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
00953 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00954 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
00955 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
00956 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
00957 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
00958 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
00959 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
00960 PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dv;
00961 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
00962 PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
00963 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
00964 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dv;
00965 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
00966 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
00967 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
00968 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
00969 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
00970 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
00971 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dvEXT;
00972 PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dv;
00973 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
00974 PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
00975 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
00976 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dv;
00977 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
00978 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
00979 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
00980 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dv;
00981 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
00982 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
00983 PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
00984 PFNGLPROGRAMUNIFORMUI64VNVPROC glProgramUniformui64vNV;
00985 PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
00986 PFNGLPUSHCLIENTATTRIBDEFAULTTEXTPROC glPushClientAttribDefaultEXT;
00987 PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
00988 PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
00989 PFNGLQUERYCOUNTERPROC glQueryCounter;
00990 PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
00991 PFNGLREADBUFFERPROC glReadBuffer;
00992 PFNGLREADNPPIXELSARBPROC glReadnPixelsARB;
00993 PFNGLREADNPPIXELSPROC glReadnPixels;
00994 PFNGLREADPIXELSPROC glReadPixels;
00995 PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
00996 PFNGLRENDERBUFFERSTORAGERAMGENVPROC glRenderbufferStorageMultisampleCoverageNV;
00997 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
00998 PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
00999 PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
01000 PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
01001 PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
01002 PFNGLSAMPLEMASKIPROC glSampleMaski;
01003 PFNGLSAMPLEPARAMETERFPROC glSamplerParameterf;
01004 PFNGLSAMPLEPARAMETERFVPROC glSamplerParameterfv;
01005 PFNGLSAMPLEPARAMETERIIVPROC glSamplerParameterIiiv;
01006 PFNGLSAMPLEPARAMETERIPROC glSamplerParameterIi;
01007 PFNGLSAMPLEPARAMETERIUIVPROC glSamplerParameterIuiv;
01008 PFNGLSAMPLEPARAMETERIVPROC glSamplerParameteriv;
01009 PFNGLSCISSORARRAYVPROC glScissorArrayv;
01010 PFNGLSCISSORINDEXEDPROC glScissorIndexed;
01011 PFNGLSCISSORINDEXEDVPROC glScissorIndexeddv;
01012 PFNGLSCISSORPROC glScissor;
01013 PFNGLSECONDARYCOLORFORMATINVPROC glSecondaryColorFormatNV;
01014 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
01015 PFNGLSHADERBINARYPROC glShaderBinary;
01016 PFNGLSHADERSOURCEPROC glShaderSource;
01017 PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
01018 PFNGLSIGNALLVKFENCENVPROC glSignalVkFenceNV;
01019 PFNGLSIGNALLVKSEMAPHORENVPROC glSignalVkSemaphoreNV;

```

```
01020 PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
01021 PFNGLSTATECAPTURENVPROC glStateCaptureNV;
01022 PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
01023 PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
01024 PFNGLSTENCILFUNCPROC glStencilFunc;
01025 PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
01026 PFNGLSTENCILMASKPROC glStencilMask;
01027 PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
01028 PFNGLSTENCILOPPROC glStencilOp;
01029 PFNGLSTENCILOPSEPARATEPROC glStencilOpSeparate;
01030 PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
01031 PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
01032 PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01033 PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01034 PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01035 PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
01036 PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
01037 PFNGLTEXBUFFERARBPROC glTexBufferARB;
01038 PFNGLTEXBUFFERPROC glTexBuffer;
01039 PFNGLTEXBUFFERRANGEPROC glTexBufferSize;
01040 PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
01041 PFNGLTEXIMAGE1DPROC glTexImage1D;
01042 PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
01043 PFNGLTEXIMAGE2DPROC glTexImage2D;
01044 PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
01045 PFNGLTEXIMAGE3DPROC glTexImage3D;
01046 PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
01047 PFNGLTEXPARAMETERFPROC glTexParameterf;
01048 PFNGLTEXPARAMETERFVPROC glTexParameterfv;
01049 PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
01050 PFNGLTEXPARAMETERIIPROC glTexParameterIi;
01051 PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01052 PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01053 PFNGLTEXTSTORAGE1DPROC glTexStorage1D;
01054 PFNGLTEXTSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
01055 PFNGLTEXTSTORAGE2DPROC glTexStorage2D;
01056 PFNGLTEXTSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
01057 PFNGLTEXTSTORAGE3DPROC glTexStorage3D;
01058 PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01059 PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01060 PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01061 PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
01062 PFNGLTEXTUREBARRIERPROC glTextureBarrier;
01063 PFNGLTEXTUREBUFFERREXTPROC glTextureBufferEXT;
01064 PFNGLTEXTUREBUFFERPROC glTextureBuffer;
01065 PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01066 PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01067 PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
01068 PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
01069 PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
01070 PFNGLTEXTUREPAGECOMMITMENTTEXTPROC glTexturePageCommitmentEXT;
01071 PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
01072 PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
01073 PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
01074 PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
01075 PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
01076 PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIiivEXT;
01077 PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiiv;
01078 PFNGLTEXTUREPARAMETERIPROC glTextureParameteri;
01079 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01080 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01081 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01082 PFNGLTEXTUREPARAMETERIVPROC glTextureParameteriv;
01083 PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
01084 PFNGLTEXTURERESTORAGE1DEXTPROC glTextureStorage1DEXT;
01085 PFNGLTEXTURERESTORAGE1DPROC glTextureStorage1D;
01086 PFNGLTEXTURERESTORAGE2DEXTPROC glTextureStorage2DEXT;
01087 PFNGLTEXTURERESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
01088 PFNGLTEXTURERESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
01089 PFNGLTEXTURERESTORAGE2DPROC glTextureStorage2D;
01090 PFNGLTEXTURERESTORAGE3DEXTPROC glTextureStorage3DEXT;
01091 PFNGLTEXTURERESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
01092 PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
01093 PFNGLTEXTURERESTORAGE3DPROC glTextureStorage3D;
01094 PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
01095 PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
01096 PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
01097 PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
01098 PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
01099 PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
01100 PFNGLTEXTUREVIEWPROC glTextureView;
01101 PFNGLTRANSFORMFEEDBACKBASEPROC glTransformFeedbackBufferBase;
01102 PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
01103 PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
01104 PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
01105 PFNGLUNIFORM1DPROC glUniform1d;
01106 PFNGLUNIFORM1DVPROC glUniform1dv;
```

```
01107 PFNGLUNIFORM1FPROC glUniform1f;
01108 PFNGLUNIFORM1FVPROC glUniform1fv;
01109 PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
01110 PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
01111 PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
01112 PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
01113 PFNGLUNIFORM1IPROC glUniform1i;
01114 PFNGLUNIFORM1IVPROC glUniform1iv;
01115 PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01116 PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01117 PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
01118 PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
01119 PFNGLUNIFORM1UIPROC glUniform1ui;
01120 PFNGLUNIFORM1UIVPROC glUniform1uiv;
01121 PFNGLUNIFORM2DPROC glUniform2d;
01122 PFNGLUNIFORM2DVPROC glUniform2dv;
01123 PFNGLUNIFORM2FPROC glUniform2f;
01124 PFNGLUNIFORM2FVPROC glUniform2fv;
01125 PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
01126 PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
01127 PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
01128 PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
01129 PFNGLUNIFORM2IPROC glUniform2i;
01130 PFNGLUNIFORM2IVPROC glUniform2iv;
01131 PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01132 PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01133 PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
01134 PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
01135 PFNGLUNIFORM2UIVPROC glUniform2uiv;
01136 PFNGLUNIFORM2UIVPROC glUniform2uiv;
01137 PFNGLUNIFORM3DPROC glUniform3d;
01138 PFNGLUNIFORM3DVPROC glUniform3dv;
01139 PFNGLUNIFORM3FFPROC glUniform3f;
01140 PFNGLUNIFORM3FVPROC glUniform3fv;
01141 PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
01142 PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
01143 PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
01144 PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
01145 PFNGLUNIFORM3IPROC glUniform3i;
01146 PFNGLUNIFORM3IVPROC glUniform3iv;
01147 PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01148 PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01149 PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
01150 PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
01151 PFNGLUNIFORM3UIPROC glUniform3ui;
01152 PFNGLUNIFORM3UIVPROC glUniform3uiv;
01153 PFNGLUNIFORM4DPROC glUniform4d;
01154 PFNGLUNIFORM4DVPROC glUniform4dv;
01155 PFNGLUNIFORM4FPROC glUniform4f;
01156 PFNGLUNIFORM4FVPROC glUniform4fv;
01157 PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
01158 PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
01159 PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
01160 PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
01161 PFNGLUNIFORM4IPROC glUniform4i;
01162 PFNGLUNIFORM4IVPROC glUniform4iv;
01163 PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01164 PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01165 PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
01166 PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
01167 PFNGLUNIFORM4UIPROC glUniform4ui;
01168 PFNGLUNIFORM4UIVPROC glUniform4uiv;
01169 PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01170 PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01171 PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01172 PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
01173 PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
01174 PFNGLUNIFORMMATRIX2DPROC glUniformMatrix2dv;
01175 PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
01176 PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
01177 PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
01178 PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
01179 PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
01180 PFNGLUNIFORMMATRIX3DVPROC glUniformMatrix3dv;
01181 PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
01182 PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
01183 PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
01184 PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
01185 PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
01186 PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
01187 PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
01188 PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
01189 PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
01190 PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
01191 PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
01192 PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
01193 PFNGLUNIFORMUI64NVPROC glUniformui64NV;
```

```
01194 PFNGLUNIFORMUI64VNVPROC glUniformui64vNV;
01195 PFNGLUNMAPBUFFERPROC glUnmapBuffer;
01196 PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
01197 PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
01198 PFNGLUSEPROGRAMPROC glUseProgram;
01199 PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
01200 PFNGLUSESHEADERPROGRAMEXTPROC glUseShaderProgramEXT;
01201 PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
01202 PFNGLVALIDATEPROGRAMPROC glValidateProgram;
01203 PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
01204 PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
01205 PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribIFormat;
01206 PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
01207 PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01208 PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01209 PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
01210 PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01211 PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01212 PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01213 PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
01214 PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01215 PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
01216 PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01217 PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
01218 PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
01219 PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01220 PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01221 PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01222 PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01223 PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
01224 PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
01225 PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01226 PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01227 PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
01228 PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01229 PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
01230 PFNGLVERTEXATTRIB1DPROC glVertexAttribAttrib1d;
01231 PFNGLVERTEXATTRIB1DVPROC glVertexAttribAttrib1dv;
01232 PFNGLVERTEXATTRIB1FPROC glVertexAttribAttrib1f;
01233 PFNGLVERTEXATTRIB1FVPROC glVertexAttribAttrib1fv;
01234 PFNGLVERTEXATTRIB1SPROC glVertexAttribAttrib1s;
01235 PFNGLVERTEXATTRIB1SVPROC glVertexAttribAttrib1sv;
01236 PFNGLVERTEXATTRIB2DPROC glVertexAttribAttrib2d;
01237 PFNGLVERTEXATTRIB2DVPROC glVertexAttribAttrib2dv;
01238 PFNGLVERTEXATTRIB2FPROC glVertexAttribAttrib2f;
01239 PFNGLVERTEXATTRIB2FVPROC glVertexAttribAttrib2fv;
01240 PFNGLVERTEXATTRIB2SPROC glVertexAttribAttrib2s;
01241 PFNGLVERTEXATTRIB2SVPROC glVertexAttribAttrib2sv;
01242 PFNGLVERTEXATTRIB3DPROC glVertexAttribAttrib3d;
01243 PFNGLVERTEXATTRIB3DVPROC glVertexAttribAttrib3dv;
01244 PFNGLVERTEXATTRIB3FPROC glVertexAttribAttrib3f;
01245 PFNGLVERTEXATTRIB3FVPROC glVertexAttribAttrib3fv;
01246 PFNGLVERTEXATTRIB3SPROC glVertexAttribAttrib3s;
01247 PFNGLVERTEXATTRIB3SVPROC glVertexAttribAttrib3sv;
01248 PFNGLVERTEXATTRIB4BVPROC glVertexAttribAttrib4bv;
01249 PFNGLVERTEXATTRIB4DPROC glVertexAttribAttrib4d;
01250 PFNGLVERTEXATTRIB4DVPROC glVertexAttribAttrib4dv;
01251 PFNGLVERTEXATTRIB4FPROC glVertexAttribAttrib4f;
01252 PFNGLVERTEXATTRIB4FVPROC glVertexAttribAttrib4fv;
01253 PFNGLVERTEXATTRIB4IVPROC glVertexAttribAttrib4iv;
01254 PFNGLVERTEXATTRIB4NBVPROC glVertexAttribAttrib4Nbv;
01255 PFNGLVERTEXATTRIB4NIVPROC glVertexAttribAttrib4Niv;
01256 PFNGLVERTEXATTRIB4NSVPROC glVertexAttribAttrib4Nsv;
01257 PFNGLVERTEXATTRIB4NUBPROC glVertexAttribAttrib4Nub;
01258 PFNGLVERTEXATTRIB4NUBVPROC glVertexAttribAttrib4Nubv;
01259 PFNGLVERTEXATTRIB4NUIVPROC glVertexAttribAttrib4Nuiv;
01260 PFNGLVERTEXATTRIB4NUSVPROC glVertexAttribAttrib4Nusv;
01261 PFNGLVERTEXATTRIB4SVPROC glVertexAttribAttrib4s;
01262 PFNGLVERTEXATTRIB4SVPROC glVertexAttribAttrib4sv;
01263 PFNGLVERTEXATTRIB4UBVPROC glVertexAttribAttrib4ubv;
01264 PFNGLVERTEXATTRIB4UIVPROC glVertexAttribAttrib4uiv;
01265 PFNGLVERTEXATTRIB4USVPROC glVertexAttribAttrib4usv;
01266 PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01267 PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01268 PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01269 PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01270 PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01271 PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01272 PFNGLVERTEXATTRIBI1IIVPROC glVertexAttribI1iiv;
01273 PFNGLVERTEXATTRIBI1IUIPROC glVertexAttribI1ui;
01274 PFNGLVERTEXATTRIBI1IUIVPROC glVertexAttribI1uiv;
01275 PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01276 PFNGLVERTEXATTRIBI2IIVPROC glVertexAttribI2iv;
01277 PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
01278 PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
01279 PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
01280 PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
```

```

01281 PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01282 PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
01283 PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01284 PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
01285 PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
01286 PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
01287 PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
01288 PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
01289 PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
01290 PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
01291 PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
01292 PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
01293 PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
01294 PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
01295 PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
01296 PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
01297 PFNGLVERTEXATTRIBL1I64VNVPROC glVertexAttribL1i64vNV;
01298 PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
01299 PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
01300 PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
01301 PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribL1ui64vNV;
01302 PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
01303 PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
01304 PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
01305 PFNGLVERTEXATTRIBL2I64VNVPROC glVertexAttribL2i64vNV;
01306 PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
01307 PFNGLVERTEXATTRIBL2UI64VNVPROC glVertexAttribL2ui64vNV;
01308 PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01309 PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
01310 PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
01311 PFNGLVERTEXATTRIBL3I64VNVPROC glVertexAttribL3i64vNV;
01312 PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
01313 PFNGLVERTEXATTRIBL3UI64VNVPROC glVertexAttribL3ui64vNV;
01314 PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01315 PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
01316 PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
01317 PFNGLVERTEXATTRIBL4I64VNVPROC glVertexAttribL4i64vNV;
01318 PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
01319 PFNGLVERTEXATTRIBL4UI64VNVPROC glVertexAttribL4ui64vNV;
01320 PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
01321 PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
01322 PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
01323 PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
01324 PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
01325 PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
01326 PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
01327 PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
01328 PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
01329 PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
01330 PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
01331 PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
01332 PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
01333 PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
01334 PFNGLVIEWPORTARRAYVPROC glVertexArrayv;
01335 PFNGLVIEWPORTINDEXEDFPROC glVertexIndexeddf;
01336 PFNGLVIEWPORTINDEXEDFVPROC glVertexIndexedfv;
01337 PFNGLVIEWPORTPOSITIONWSCALENVPROC glVertexPositionWScaleNV;
01338 PFNGLVIEWPORTPROC glVertexport;
01339 PFNGLVIEWPORTSWIZZLENVPROC glVertexportSwizzleNV;
01340 PFNGLWAITSYNCPROC glWaitSync;
01341 PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01342 PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01343 PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01344 #endif
01345
01346
01347
01348 GLint gg::ggBufferAlignment(0);
01349
01350
01351 //
01352 // ゲームグラフィックス特論の都合にもとづく初期化
01353 //
01354 void gg::ggInit()
01355 {
01356     // 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
01357     if (ggBufferAlignment) return;
01358
01359     // macOS 以外で OpenGL 3.2 以降の API を取得する
01360 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
01361     glActiveProgramEXT = PFNGLACTIVEPROGRAMEXTPROC(glfwGetProcAddress("glActiveProgramEXT"));
01362     glActiveShaderProgram = PFNGLACTIVESHADERPROGRAMPROC(glfwGetProcAddress("glActiveShaderProgram"));
01363     glActiveTexture = PFNGLACTIVETEXTUREPROC(glfwGetProcAddress("glActiveTexture"));
01364     glApplyFramebufferAttachmentCMAAINTEL =
01365         PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC(glfwGetProcAddress("glApplyFramebufferAttachmentCMAAINTEL"));
01366     glAttachShader = PFNGLATTACHSHADERPROC(glfwGetProcAddress("glAttachShader"));
01367     glBeginConditionalRender =
01368         PFNGLBEGINCONDITIONALRENDERPROC(glfwGetProcAddress("glBeginConditionalRender"));
01369     glBeginConditionalRenderNV =

```

```

01368 PFNGLBEGINCONDITIONALRENDERNVPROC(glfwGetProcAddress("glBeginConditionalRenderNV"));
01369 glBeginPerfMonitorAMD = PFNGLBEGINPERFMONITORAMDPROC(glfwGetProcAddress("glBeginPerfMonitorAMD"));
01370 glBeginPerfQueryINTEL = PFNGLBEGINPERFQUERYINTELPROC(glfwGetProcAddress("glBeginPerfQueryINTEL"));
01371 glBeginQuery = PFNGLBEGINQUERYPROC(glfwGetProcAddress("glBeginQuery"));
01372 glBeginQueryIndexed = PFNGLBEGINQUERYINDEXEDPROC(glfwGetProcAddress("glBeginQueryIndexed"));
01373 glBeginTransformFeedback =
01374 PFNGLBEGINTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBeginTransformFeedback"));
01375 glBindAttribLocation = PFNGLBINDATTRIBLOCATIONPROC(glfwGetProcAddress("glBindAttribLocation"));
01376 glBindBuffer = PFNGLBINDBUFFERPROC(glfwGetProcAddress("glBindBuffer"));
01377 glBindBufferBase = PFNGLBINDBUFFERBASEPROC(glfwGetProcAddress("glBindBufferBase"));
01378 glBindBufferRange = PFNGLBINDBUFFERRANGEPROC(glfwGetProcAddress("glBindBufferRange"));
01379 glBindBuffersBase = PFNGLBINDBUFFERSBASEPROC(glfwGetProcAddress("glBindBuffersBase"));
01380 glBindBuffersRange = PFNGLBINDBUFFERSRANGEPROC(glfwGetProcAddress("glBindBuffersRange"));
01381 glBindFragDataLocation =
01382 PFNGLBINDFRAGDATALOCATIONPROC(glfwGetProcAddress("glBindFragDataLocation"));
01383 glBindFragDataLocationIndexed =
01384 PFNGLBINDFRAGDATALOCATIONINDEXEDPROC(glfwGetProcAddress("glBindFragDataLocationIndexed"));
01385 glBindFramebuffer = PFNGLBINDFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
01386 glBindImageTexture = PFNGLBINDIMAGETEXTUREPROC(glfwGetProcAddress("glBindImageTexture"));
01387 glBindImageTextures = PFNGLBINDIMAGETEXTURESPROC(glfwGetProcAddress("glBindImageTextures"));
01388 glBindMultiTextureEXT = PFNGLBINDMULTITEXTUREEXTPROC(glfwGetProcAddress("glBindMultiTextureEXT"));
01389 glBindProgramPipeline = PFNGLBINDPROGRAMPIPELINEPROC(glfwGetProcAddress("glBindProgramPipeline"));
01390 glBindRenderbuffer = PFNGLBINDRENDERBUFFERPROC(glfwGetProcAddress("glBindRenderbuffer"));
01391 glBindSampler = PFNGLBINDSAMPLERPROC(glfwGetProcAddress("glBindSampler"));
01392 glBindSamplers = PFNGLBINDSAMPLERSPROC(glfwGetProcAddress("glBindSamplers"));
01393 glBindTexture = PFNGLBINDTEXTUREPROC(glfwGetProcAddress("glBindTexture"));
01394 glBindTextureUnit = PFNGLBINDTEXTUREUNITPROC(glfwGetProcAddress("glBindTextureUnit"));
01395 glBindTextures = PFNGLBINDTEXTURESPROC(glfwGetProcAddress("glBindTextures"));
01396 glBindTransformFeedback =
01397 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 = PFNGLBLENEQUATIONPROC(glfwGetProcAddress("glBlendEquation"));
01405 glBindEquationSeparate =
01406 PFNGLBLENEQUATIONSEPARATEPROC(glfwGetProcAddress("glBlendEquationSeparate"));
01407 glBindEquationSeparatei =
01408 PFNGLBLENEQUATIONSEPARATEIPROC(glfwGetProcAddress("glBlendEquationSeparatei"));
01409 glBindEquationSeparateiARB =
01410 PFNGLBLENEQUATIONSEPARATEIARBPROC(glfwGetProcAddress("glBlendEquationSeparateiARB"));
01411 glBindEquationi = PFNGLBLENEQUATIONIPROC(glfwGetProcAddress("glBlendEquationi"));
01412 glBindEquationiARB = PFNGLBLENEQUATIONIARBPROC(glfwGetProcAddress("glBlendEquationiARB"));
01413 glBindFunc = PFNGLBLENDFUNCPROC(glfwGetProcAddress("glBlendFunc"));
01414 glBindFuncSeparate = PFNGLBLENDFUNCSEPARATEPROC(glfwGetProcAddress("glBlendFuncSeparate"));
01415 glBindFuncSeparatei = PFNGLBLENDFUNCSEPARATEIPROC(glfwGetProcAddress("glBlendFuncSeparatei"));
01416 glBindFuncSeparateiARB =
01417 PFNGLBLENDFUNCSEPARATEIARBPROC(glfwGetProcAddress("glBlendFuncSeparateiARB"));
01418 glBindFunci = PFNGLBLENDFUNCIPROC(glfwGetProcAddress("glBlendFunci"));
01419 glBindFunciARB = PFNGLBLENDFUNCIARBPROC(glfwGetProcAddress("glBlendFunciARB"));
01420 glBindParameteriNV = PFNGLBLENDPARAMETERINVPROC(glfwGetProcAddress("glBlendParameteriNV"));
01421 glBindFramebuffer = PFNGLBLITFRAMEBUFFERPROC(glfwGetProcAddress("glBlitFramebuffer"));
01422 glBindNamedFramebuffer =
01423 PFNGLBLITNAMEDFRAMEBUFFERPROC(glfwGetProcAddress("glBlitNamedFramebuffer"));
01424 glBindBufferAddressRangeNV =
01425 PFNGLBUFFERADDRESSRANGENVPROC(glfwGetProcAddress("glBufferAddressRangeNV"));
01426 glBindBufferData = PFNGLBUFFERDATAPROC(glfwGetProcAddress("glBufferData"));
01427 glBindBufferPageCommitmentARB =
01428 PFNGLBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glBufferPageCommitmentARB"));
01429 glBindBufferStorage =
01430 PFNGLBUFFERSTORAGEPROC(glfwGetProcAddress("glBufferStorage"));
01431 glBindBufferSubData = PFNGLBUFFERSUBDATAPROC(glfwGetProcAddress("glBufferSubData"));
01432 glBindCallCommandListNV = PFNGLCALLCOMMANDLISTNVPROC(glfwGetProcAddress("glCallCommandListNV"));
01433 glBindCheckFramebufferStatus =
01434 PFNGLCHECKFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckFramebufferStatus"));
01435 glBindCheckNamedFramebufferStatus =
01436 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckNamedFramebufferStatus"));
01437 glBindCheckNamedFramebufferStatusEXT =
01438 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC(glfwGetProcAddress("glCheckNamedFramebufferStatusEXT"));
01439 glBindClampColor = PFNGLCLAMPCOLORPROC(glfwGetProcAddress("glClampColor"));
01440 glBindClear = PFNGLCLEARPROC(glfwGetProcAddress("glClear"));
01441 glBindClearBufferData = PFNGLCLEARBUFFERDATAPROC(glfwGetProcAddress("glClearBufferData"));
01442 glBindClearBufferSubData = PFNGLCLEARBUFFERSUBDATAPROC(glfwGetProcAddress("glClearBufferSubData"));
01443 glBindClearBufferfi = PFNGLCLEARBUFFERFIPROC(glfwGetProcAddress("glClearBufferfi"));
01444 glBindClearBufferfv = PFNGLCLEARBUFFERFVPROC(glfwGetProcAddress("glClearBufferfv"));
01445 glBindClearBufferiv = PFNGLCLEARBUFFERIVPROC(glfwGetProcAddress("glClearBufferiv"));
01446 glBindClearBufferuiv = PFNGLCLEARBUFFERUIVPROC(glfwGetProcAddress("glClearBufferuiv"));
01447 glBindClearColor = PFNGLCLEARCOLORPROC(glfwGetProcAddress("glClearColor"));
01448 glBindClearDepth = PFNGLCLEARDEPTHPROC(glfwGetProcAddress("glClearDepth"));
01449 glBindClearDepthf = PFNGLCLEARDEPTHFPROC(glfwGetProcAddress("glClearDepthf"));
01450 glBindClearNamedBufferData =
01451 PFNGLCLEARNAMEDBUFFERDATAPROC(glfwGetProcAddress("glClearNamedBufferData"));
01452 glBindClearNamedBufferDataEXT =
01453 PFNGLCLEARNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataEXT"));
01454 glBindClearNamedBufferSubData =
01455 PFNGLCLEARNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glClearNamedBufferSubData"));

```

```

01437     glClearNamedBufferSubDataEXT =
01438     PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferSubDataEXT"));
01439     glClearNamedFramebufferfi =
01440     PFNGLCLEARNAMEDFRAMEBUFFERFIPROC(glfwGetProcAddress("glClearNamedFramebufferfi"));
01441     glClearNamedFramebufferfv =
01442     PFNGLCLEARNAMEDFRAMEBUFFERFVPROC(glfwGetProcAddress("glClearNamedFramebufferfv"));
01443     glClearNamedFramebufferiv =
01444     PFNGLCLEARNAMEDFRAMEBUFFERIVPROC(glfwGetProcAddress("glClearNamedFramebufferiv"));
01445     glClearNamedFramebufferui =
01446     PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC(glfwGetProcAddress("glClearNamedFramebufferui"));
01447     glClearStencil = 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     glCompressedTextureSubImage1DEXT =
01497     PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage1DEXT"));
01498     glCompressedTextureSubImage2D =
01499     PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTextureSubImage2D"));
01500     glCompressedTextureSubImage2DEXT =
01501     PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage2DEXT"));
01502     glCompressedTextureSubImage3D =
01503     PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTextureSubImage3D"));
01504     glCompressedTextureSubImage3DEXT =
01505     PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage3DEXT"));
01506     glConservativeRasterParameterfNV =
01507     PFNGLCONSERVATIVERASTERPARAMETERFNVPROC(glfwGetProcAddress("glConservativeRasterParameterfNV"));
01508     glConservativeRasterParameteriNV =
01509     PFNGLCONSERVATIVERASTERPARAMETERINVPROC(glfwGetProcAddress("glConservativeRasterParameteriNV"));
01510     glCopyBufferSubData = PFNGLCOPYBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyBufferSubData"));
01511     glCopyImageSubData = PFNGLCOPYIMAGESUBDATAPROC(glfwGetProcAddress("glCopyImageSubData"));
01512     glCopyMultiTexImage1DEXT =
01513     PFNGLCOPYMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage1DEXT"));
01514     glCopyMultiTexImage2DEXT =
01515     PFNGLCOPYMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage2DEXT"));
01516     glCopyMultiTexSubImage1DEXT =
01517     PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage1DEXT"));
01518     glCopyMultiTexSubImage2DEXT =
01519     PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage2DEXT"));
01520     glCopyMultiTexSubImage3DEXT =
01521     PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage3DEXT"));
01522     glCopyNamedBufferSubData =
01523     PFNGLCOPYNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyNamedBufferSubData"));

```

```

01486 glCopyPathNV = PFNGLCOPYPATHNVPROC(glfwGetProcAddress("glCopyPathNV"));
01487 glCopyTexImage1D = PFNGLCOPYTEXIMAGE1DPROC(glfwGetProcAddress("glCopyTexImage1D"));
01488 glCopyTexImage2D = PFNGLCOPYTEXIMAGE2DPROC(glfwGetProcAddress("glCopyTexImage2D"));
01489 glCopyTexSubImage1D = PFNGLCOPYTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCopyTexSubImage1D"));
01490 glCopyTexSubImage2D = PFNGLCOPYTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCopyTexSubImage2D"));
01491 glCopyTexSubImage3D = PFNGLCOPYTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCopyTexSubImage3D"));
01492 glCopyTextureImage1DEXT =
    PFNGLCOPYTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureImage1DEXT"));
01493 glCopyTextureImage2DEXT =
    PFNGLCOPYTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureImage2DEXT"));
01494 glCopyTextureSubImage1D =
    PFNGLCOPYTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCopyTextureSubImage1D"));
01495 glCopyTextureSubImage1DEXT =
    PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage1DEXT"));
01496 glCopyTextureSubImage2D =
    PFNGLCOPYTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCopyTextureSubImage2D"));
01497 glCopyTextureSubImage2DEXT =
    PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage2DEXT"));
01498 glCopyTextureSubImage3D =
    PFNGLCOPYTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCopyTextureSubImage3D"));
01499 glCopyTextureSubImage3DEXT =
    PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage3DEXT"));
01500 glCoverFillPathInstancedNV =
    PFNGLCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverFillPathInstancedNV"));
01501 glCoverFillPathNV = PFNGLCOVERFILLPATHNVPROC(glfwGetProcAddress("glCoverFillPathNV"));
01502 glCoverStrokePathInstancedNV =
    PFNGLCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverStrokePathInstancedNV"));
01503 glCoverStrokePathNV = PFNGLCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glCoverStrokePathNV"));
01504 glCoverageModulationNV =
    PFNGLCOVERAGEMODULATIONNVPROC(glfwGetProcAddress("glCoverageModulationNV"));
01505 glCoverageModulationTableNV =
    PFNGLCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glCoverageModulationTableNV"));
01506 glCreateBuffers = PFNGLCREATEBUFFERSPROC(glfwGetProcAddress("glCreateBuffers"));
01507 glCreateCommandListsNV =
    PFNGLCREATECOMMANDLISTSNVPROC(glfwGetProcAddress("glCreateCommandListsNV"));
01508 glCreateFramebuffers = PFNGLCREATEFRAMEBUFFERSPROC(glfwGetProcAddress("glCreateFramebuffers"));
01509 glCreatePerfQueryINTEL =
    PFNGLCREATEPERFQUERYINTELPROC(glfwGetProcAddress("glCreatePerfQueryINTEL"));
01510 glCreateProgram = PFNGLCREATEPROGRAMPROC(glfwGetProcAddress("glCreateProgram"));
01511 glCreateProgramPipelines =
    PFNGLCREATEPROGRAMPIPELINESPROC(glfwGetProcAddress("glCreateProgramPipelines"));
01512 glCreateQueries = PFNGLCREATEQUERIESPROC(glfwGetProcAddress("glCreateQueries"));
01513 glCreateRenderbuffers = PFNGLCREATERENDERBUFFERSPROC(glfwGetProcAddress("glCreateRenderbuffers"));
01514 glCreateSamplers = PFNGLCREATESAMPLERSPROC(glfwGetProcAddress("glCreateSamplers"));
01515 glCreateShader = PFNGLCREATESHADERPROC(glfwGetProcAddress("glCreateShader"));
01516 glCreateShaderProgramEXT =
    PFNGLCREATESHADERPROGRAMEXTPROC(glfwGetProcAddress("glCreateShaderProgramEXT"));
01517 glCreateShaderProgramv =
    PFNGLCREATESHADERPROGRAMVPROC(glfwGetProcAddress("glCreateShaderProgramv"));
01518 glCreateStatesNV = PFNGLCREATESTATESNVPROC(glfwGetProcAddress("glCreateStatesNV"));
01519 glCreateSyncFromCleventARB =
    PFNGLCREATESYNCFROMCLEVENTARBPROC(glfwGetProcAddress("glCreateSyncFromCleventARB"));
01520 glCreateTextures = PFNGLCREATETEXTURESPROC(glfwGetProcAddress("glCreateTextures"));
01521 glCreateTransformFeedbacks =
    PFNGLCREATETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glCreateTransformFeedbacks"));
01522 glCreateVertexArrays = PFNGLCREATEVERTEXARRAYSPROC(glfwGetProcAddress("glCreateVertexArrays"));
01523 glCullFace = PFNGLCULLFACEPROC(glfwGetProcAddress("glCullFace"));
01524 glDebugMessageCallback =
    PFNGLDEBUGMESSAGECALLBACKPROC(glfwGetProcAddress("glDebugMessageCallback"));
01525 glDebugMessageCallbackARB =
    PFNGLDEBUGMESSAGECALLBACKARBPROC(glfwGetProcAddress("glDebugMessageCallbackARB"));
01526 glDebugMessageControl = PFNGLDEBUGMESSAGECONTROLPROC(glfwGetProcAddress("glDebugMessageControl"));
01527 glDebugMessageControlARB =
    PFNGLDEBUGMESSAGECONTROLARBPROC(glfwGetProcAddress("glDebugMessageControlARB"));
01528 glDebugMessageInsert = PFNGLDEBUGMESSAGEINSERTPROC(glfwGetProcAddress("glDebugMessageInsert"));
01529 glDebugMessageInsertARB =
    PFNGLDEBUGMESSAGEINSERTARBPROC(glfwGetProcAddress("glDebugMessageInsertARB"));
01530 glDeleteBuffers = PFNGLDELETEBUFFERSPROC(glfwGetProcAddress("glDeleteBuffers"));
01531 glDeleteCommandListsNV =
    PFNGLDELETECOMMANDLISTSNVPROC(glfwGetProcAddress("glDeleteCommandListsNV"));
01532 glDeleteFramebuffers = PFNGLDELETEFRAMEBUFFERSPROC(glfwGetProcAddress("glDeleteFramebuffers"));
01533 glDeleteNamedStringARB =
    PFNGLDELETENAMEDSTRINGARBPROC(glfwGetProcAddress("glDeleteNamedStringARB"));
01534 glDeletePathsNV = PFNGLDELETEPATHSNVPROC(glfwGetProcAddress("glDeletePathsNV"));
01535 glDeletePerfMonitorsAMD =
    PFNGLDELETEPERFMONITORSAMDPROC(glfwGetProcAddress("glDeletePerfMonitorsAMD"));
01536 glDeletePerfQueryINTEL =
    PFNGLDELETEPERFQUERYINTELPROC(glfwGetProcAddress("glDeletePerfQueryINTEL"));
01537 glDeleteProgram = PFNGLDELETEPROGRAMPROC(glfwGetProcAddress("glDeleteProgram"));
01538 glDeleteProgramPipelines =
    PFNGLDELETEPROGRAMPIPELINESPROC(glfwGetProcAddress("glDeleteProgramPipelines"));
01539 glDeleteQueries = PFNGLDELETEQUERIESPROC(glfwGetProcAddress("glDeleteQueries"));
01540 glDeleteRenderbuffers = PFNGLDELETERENDERBUFFERSPROC(glfwGetProcAddress("glDeleteRenderbuffers"));
01541 glDeleteSamplers = PFNGLDELETESAMPLERSPROC(glfwGetProcAddress("glDeleteSamplers"));
01542 glDeleteShader = PFNGLDELETESHADERPROC(glfwGetProcAddress("glDeleteShader"));
01543 glDeleteStatesNV = PFNGLDELETESTATESNVPROC(glfwGetProcAddress("glDeleteStatesNV"));
01544 glDeleteSync = PFNGLDELETESYNCPROC(glfwGetProcAddress("glDeleteSync"));

```

```

01545     glDeleteTextures = PFNGLDELETETEXTURESPROC(glfwGetProcAddress("glDeleteTextures"));
01546     glDeleteTransformFeedbacks =
01547     PFNGLDELETETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glDeleteTransformFeedbacks"));
01548     glDeleteVertexArrays = PFNGLDELETEVERTEXARRAYSPROC(glfwGetProcAddress("glDeleteVertexArrays"));
01549     glDepthFunc = PFNGLDEPTHFUNCPROC(glfwGetProcAddress("glDepthFunc"));
01549     glDepthMask = PFNGLDEPTHMASKPROC(glfwGetProcAddress("glDepthMask"));
01550     glDepthRange = PFNGLDEPTHRANGEPROC(glfwGetProcAddress("glDepthRange"));
01551     glDepthRangeArrayv = PFNGLDEPTHRANGEARRAYVPROC(glfwGetProcAddress("glDepthRangeArrayv"));
01552     glDepthRangeIndexed = PFNGLDEPTHRANGEINDEXEDPROC(glfwGetProcAddress("glDepthRangeIndexed"));
01553     glDetachShader = PFNGLDETACHSHADERPROC(glfwGetProcAddress("glDetachShader"));
01554     glDisable = PFNGLDISABLEPROC(glfwGetProcAddress("glDisable"));
01555     glDisableClientStateIndexedEXT;
01556     glDisableClientStateIndexedEXT =
01557     PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glDisableClientStateIndexedEXT"));
01557     glDisableClientStateiEXT =
01558     PFNGLDISABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glDisableClientStateiEXT"));
01558     glDisableIndexedEXT = PFNGLDISABLEINDEXEDEXTPROC(glfwGetProcAddress("glDisableIndexedEXT"));
01559     glDisableVertexAttribArrayAttrib =
01560     PFNGLDISABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttrib"));
01560     glDisableVertexAttribArrayAttribEXT =
01561     PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttribEXT"));
01561     glDisableVertexAttribArrayEXT =
01562     PFNGLDISABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayEXT"));
01562     glDisableVertexAttribArrayAttribArray =
01563     PFNGLDISABLEVERTEXATTRIBARRAYPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttribArray"));
01563     glDisablei = PFNGLDISABLEIPROC(glfwGetProcAddress("glDisablei"));
01564     glDispatchCompute = PFNGLDISPATCHCOMPUTEPROC(glfwGetProcAddress("glDispatchCompute"));
01565     glDispatchComputeGroupSizeARB =
01566     PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC(glfwGetProcAddress("glDispatchComputeGroupSizeARB"));
01566     glDispatchComputeIndirect =
01567     PFNGLDISPATCHCOMPUTEINDIRECTPROC(glfwGetProcAddress("glDispatchComputeIndirect"));
01567     glDrawArrays = PFNGLDRAWARRAYSPROC(glfwGetProcAddress("glDrawArrays"));
01568     glDrawArraysIndirect = PFNGLDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glDrawArraysIndirect"));
01569     glDrawArraysInstanced = PFNGLDRAWARRAYSINSTANCEDPROC(glfwGetProcAddress("glDrawArraysInstanced"));
01570     glDrawArraysInstancedARB =
01571     PFNGLDRAWARRAYSINSTANCEDARBPROC(glfwGetProcAddress("glDrawArraysInstancedARB"));
01571     glDrawArraysInstancedBaseInstance =
01572     PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawArraysInstancedBaseInstance"));
01572     glDrawArraysInstancedEXT =
01573     PFNGLDRAWARRAYSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawArraysInstancedEXT"));
01574     glDrawBuffer = PFNGLDRAWBUFFERPROC(glfwGetProcAddress("glDrawBuffer"));
01574     glDrawBuffers = PFNGLDRAWBUFFERSPROC(glfwGetProcAddress("glDrawBuffers"));
01575     glDrawCommandsAddressNV =
01576     PFNGLDRAWCOMMANDSADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsAddressNV"));
01576     glDrawCommandsNV = PFNGLDRAWCOMMANDSNVPROC(glfwGetProcAddress("glDrawCommandsNV"));
01577     glDrawCommandsStatesAddressNV =
01578     PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsStatesAddressNV"));
01578     glDrawCommandsStatesNV =
01579     PFNGLDRAWCOMMANDSSTATESNVPROC(glfwGetProcAddress("glDrawCommandsStatesNV"));
01579     glDrawElements = PFNGLDRAWELEMENTSPROC(glfwGetProcAddress("glDrawElements"));
01580     glDrawElementsBaseVertex =
01581     PFNGLDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsBaseVertex"));
01581     glDrawElementsIndirect =
01582     PFNGLDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glDrawElementsIndirect"));
01582     glDrawElementsInstanced =
01583     PFNGLDRAWELEMENTSINSTANCEDPROC(glfwGetProcAddress("glDrawElementsInstanced"));
01583     glDrawElementsInstancedARB =
01584     PFNGLDRAWELEMENTSINSTANCEDARBPROC(glfwGetProcAddress("glDrawElementsInstancedARB"));
01584     glDrawElementsInstancedBaseInstance =
01585     PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseInstance"));
01585     glDrawElementsInstancedBaseVertex =
01586     PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertex"));
01586     glDrawElementsInstancedBaseVertexBaseInstance =
01587     PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertexBaseInstance"));
01587     glDrawElementsInstancedEXT =
01588     PFNGLDRAWELEMENTSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawElementsInstancedEXT"));
01588     glDrawRangeElements = PFNGLDRAWRANGEELEMENTSPROC(glfwGetProcAddress("glDrawRangeElements"));
01589     glDrawRangeElementsBaseVertex =
01590     PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawRangeElementsBaseVertex"));
01590     glDrawTransformFeedback =
01591     PFNGLDRAWTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glDrawTransformFeedback"));
01591     glDrawTransformFeedbackInstanced =
01592     PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackInstanced"));
01592     glDrawTransformFeedbackStream =
01593     PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC(glfwGetProcAddress("glDrawTransformFeedbackStream"));
01593     glDrawTransformFeedbackStreamInstanced =
01594     PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackStreamInstanced"));
01594     glDrawVkImageNV = PFNGLDRAWVKIMAGENVPROC(glfwGetProcAddress("glDrawVkImageNV"));
01595     glEdgeFlagFormatNV = PFNGLEDGEFLAGFORMATNVPROC(glfwGetProcAddress("glEdgeFlagFormatNV"));
01596     glEnable = PFNGLENABLEPROC(glfwGetProcAddress("glEnable"));
01597     glEnableClientStateIndexedEXT =
01598     PFNGLENABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glEnableClientStateIndexedEXT"));
01598     glEnableClientStateiEXT =
01599     PFNGLENABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glEnableClientStateiEXT"));
01599     glEnableIndexedEXT = PFNGLENABLEINDEXEDEXTPROC(glfwGetProcAddress("glEnableIndexedEXT"));
01600     glEnableVertexAttribArrayAttrib =
01600     PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttrib"));

```

```

01601     glEnableVertexAttribArrayEXT =
01602     PFNGLENABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttribEXT"));
01603     glEnableVertexAttribArray =
01604     PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArray"));
01605     glEnablei = PFNGLENABLEIPROC(glfwGetProcAddress("glEnablei"));
01606     glEndConditionalRender =
01607     PFNGLENDCONDITIONALRENDERPROC(glfwGetProcAddress("glEndConditionalRender"));
01608     glEndConditionalRenderNV =
01609     PFNGLENDCONDITIONALRENDERNVPROC(glfwGetProcAddress("glEndConditionalRenderNV"));
01610     glEndQuery = PFNGLENDQUERYPROC(glfwGetProcAddress("glEndQuery"));
01611     glEndQueryIndexed = PFNGLENDQUERYINDEXEDPROC(glfwGetProcAddress("glEndQueryIndexed"));
01612     glEndTransformFeedback =
01613     PFNGLENDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glEndTransformFeedback"));
01614     glEvaluateDepthValuesARB =
01615     PFNGLEVALUATEDEPTHVALUESARBPROC(glfwGetProcAddress("glEvaluateDepthValuesARB"));
01616     glFenceSync = PFNGLFENCESYNCPROC(glfwGetProcAddress("glFenceSync"));
01617     glFinish = PFNGLFINISHPROC(glfwGetProcAddress("glFinish"));
01618     glFlush = PFNGLFLUSHPROC(glfwGetProcAddress("glFlush"));
01619     glFlushMappedBufferRange =
01620     PFNGLFLUSHMAPPEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedBufferRange"));
01621     glFlushMappedNamedBufferRange =
01622     PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedNamedBufferRange"));
01623     glFlushMappedNamedBufferRangeEXT =
01624     PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glFlushMappedNamedBufferRangeEXT"));
01625     glFogCoordFormatNV = PFNGLFOGCOORDFORMATNVPROC(glfwGetProcAddress("glFogCoordFormatNV"));
01626     glFragmentCoverageColorNV =
01627     PFNGLFRAGMENTCOVERAGECOLORNVPROC(glfwGetProcAddress("glFragmentCoverageColorNV"));
01628     glFramebufferDrawBufferEXT =
01629     PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC(glfwGetProcAddress("glFramebufferDrawBufferEXT"));
01630     glFramebufferDrawBuffersEXT =
01631     PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC(glfwGetProcAddress("glFramebufferDrawBuffersEXT"));
01632     glFramebufferParameteri =
01633     PFNGLFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glFramebufferParameteri"));
01634     glFramebufferReadBufferEXT =
01635     PFNGLFRAMEBUFFERREADBUFFEREXTPROC(glfwGetProcAddress("glFramebufferReadBufferEXT"));
01636     glFramebufferRenderbuffer =
01637     PFNGLFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glFramebufferRenderbuffer"));
01638     glFramebufferSampleLocationsfvARB =
01639     PFNGLFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvARB"));
01640     glFramebufferSampleLocationsfvNV =
01641     PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvNV"));
01642     glFramebufferTexture = PFNGLFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glFramebufferTexture"));
01643     glFramebufferTexture1D =
01644     PFNGLFRAMEBUFFERTEXTURE1DPROC(glfwGetProcAddress("glFramebufferTexture1D"));
01645     glFramebufferTexture2D =
01646     PFNGLFRAMEBUFFERTEXTURE2DPROC(glfwGetProcAddress("glFramebufferTexture2D"));
01647     glFramebufferTexture3D =
01648     PFNGLFRAMEBUFFERTEXTURE3DPROC(glfwGetProcAddress("glFramebufferTexture3D"));
01649     glFramebufferTextureARB =
01650     PFNGLFRAMEBUFFERTEXTUREARBPROC(glfwGetProcAddress("glFramebufferTextureARB"));
01651     glFramebufferTextureFaceARB =
01652     PFNGLFRAMEBUFFERTEXTUREFACEARBPROC(glfwGetProcAddress("glFramebufferTextureFaceARB"));
01653     glFramebufferTextureLayer =
01654     PFNGLFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glFramebufferTextureLayer"));
01655     glFramebufferTextureLayerARB =
01656     PFNGLFRAMEBUFFERTEXTURELAYERARBPROC(glfwGetProcAddress("glFramebufferTextureLayerARB"));
01657     glFramebufferTextureMultiviewOVR =
01658     PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC(glfwGetProcAddress("glFramebufferTextureMultiviewOVR"));
01659     glFrontFace = PFNGLFRONTFACEPROC(glfwGetProcAddress("glFrontFace"));
01660     glGenBuffers = PFNGLGENBUFFERSPROC(glfwGetProcAddress("glGenBuffers"));
01661     glGenFramebuffers = PFNGLGENFRAMEBUFFERSPROC(glfwGetProcAddress("glGenFramebuffers"));
01662     glGenPathsNV = PFNGLGENPATHSNVPROC(glfwGetProcAddress("glGenPathsNV"));
01663     glGenPerfMonitorsAMD = PFNGLGENPERFMONITORSAMDPROC(glfwGetProcAddress("glGenPerfMonitorsAMD"));
01664     glGenProgramPipelines = PFNGLGENPROGRAMPIPELINESPROC(glfwGetProcAddress("glGenProgramPipelines"));
01665     glGenQueries = PFNGLGENQUERIESPROC(glfwGetProcAddress("glGenQueries"));
01666     glGenRenderbuffers = PFNGLGENRENDERBUFFERSPROC(glfwGetProcAddress("glGenRenderbuffers"));
01667     glGenSamplers = PFNGLGENSAMPLERSPROC(glfwGetProcAddress("glGenSamplers"));
01668     glGenTextures = PFNGLGENTEXTURESPROC(glfwGetProcAddress("glGenTextures"));
01669     glGenTransformFeedbacks =
01670     PFNGLGENTRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glGenTransformFeedbacks"));
01671     glGenVertexArrays = PFNGLGENVERTEXARRAYSPROC(glfwGetProcAddress("glGenVertexArrays"));
01672     glGenerateMipmap = PFNGLGENERATEMIPMAPPROC(glfwGetProcAddress("glGenerateMipmap"));
01673     glGenerateMultiTexMipmapEXT =
01674     PFNGLGENERATEMULTITEMPMAPEXTPROC(glfwGetProcAddress("glGenerateMultiTexMipmapEXT"));
01675     glGenerateTextureMipmap =
01676     PFNGLGENERATETEXTUREMIPMAPPROC(glfwGetProcAddress("glGenerateTextureMipmap"));
01677     glGenerateTextureMipmapEXT =
01678     PFNGLGENERATETEXTUREMIPMAPEXTPROC(glfwGetProcAddress("glGenerateTextureMipmapEXT"));
01679     glGetActiveAtomicCounterBufferiv =
01680     PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC(glfwGetProcAddress("glGetActiveAtomicCounterBufferiv"));
01681     glGetActiveAttrib = PFNGLGETACTIVEATTRIBPROC(glfwGetProcAddress("glGetActiveAttrib"));
01682     glGetActiveSubroutineName =
01683     PFNGLGETACTIVESUBROUTINENAMEPROC(glfwGetProcAddress("glGetActiveSubroutineName"));

```

```

01656     glGetActiveSubroutineUniformName =
01657     PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveSubroutineUniformName"));
01658     glGetActiveSubroutineUniformiv =
01659     PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC(glfwGetProcAddress("glGetActiveSubroutineUniformiv"));
01660     glGetActiveUniform = PFNGLGETACTIVEUNIFORMPROC(glfwGetProcAddress("glGetActiveUniform"));
01661     glGetActiveUniformBlockName =
01662     PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC(glfwGetProcAddress("glGetActiveUniformBlockName"));
01663     glGetActiveUniformBlockiv =
01664     PFNGLGETACTIVEUNIFORMBLOCKIVPROC(glfwGetProcAddress("glGetActiveUniformBlockiv"));
01665     glGetActiveUniformName =
01666     PFNGLGETACTIVEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveUniformName"));
01667     glGetActiveUniformsiv = PFNGLGETACTIVEUNIFORMSIVPROC(glfwGetProcAddress("glGetActiveUniformsiv"));
01668     glGetAttachedShaders = PFNGLGETATTACHEDSHADERSPROC(glfwGetProcAddress("glGetAttachedShaders"));
01669     glGetAttribLocation = PFNGLGETATTRIBLOCATIONPROC(glfwGetProcAddress("glGetAttribLocation"));
01670     glGetBooleanIndexedvEXT =
01671     PFNGLGETBOOLEANINDEXEDVEXTPROC(glfwGetProcAddress("glGetBooleanIndexedvEXT"));
01672     glGetBooleani_v = PFNGLGETBOOLEANI_VPROC(glfwGetProcAddress("glGetBooleani_v"));
01673     glGetBooleanv = PFNGLGETBOOLEANVPROC(glfwGetProcAddress("glGetBooleanv"));
01674     glGetBufferParameteri64v =
01675     PFNGLGETBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetBufferParameteri64v"));
01676     glGetBufferParameteriv =
01677     PFNGLGETBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetBufferParameteriv"));
01678     glGetBufferParameterui64NV =
01679     PFNGLGETBUFFERPARAMETERUI64NVPROC(glfwGetProcAddress("glGetBufferParameterui64NV"));
01680     glGetBufferPointerv = PFNGLGETBUFFERPOINTERVERPROC(glfwGetProcAddress("glGetBufferPointerv"));
01681     glGetBufferData = PFNGLGETBUFFERSUBDATAPROC(glfwGetProcAddress("glGetBufferData"));
01682     glGetCommandHeaderNV = PFNGLGETCOMMANDHEADERNVPROC(glfwGetProcAddress("glGetCommandHeaderNV"));
01683     glGetCompressedMultiTexImageEXT =
01684     PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedMultiTexImageEXT"));
01685     glGetCompressedTexImage =
01686     PFNGLGETCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetCompressedTexImage"));
01687     glGetCompressedTextureImage =
01688     PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureImage"));
01689     glGetCompressedTextureImageEXT =
01690     PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedTextureImageEXT"));
01691     glGetCompressedTextureSubImage =
01692     PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureSubImage"));
01693     glGetCoverageModulationTableNV =
01694     PFNGLGETCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glGetCoverageModulationTableNV"));
01695     glGetDebugMessageLog = PFNGLGETDEBUGMESSAGELOGPROC(glfwGetProcAddress("glGetDebugMessageLog"));
01696     glGetDebugMessageLogARB =
01697     PFNGLGETDEBUGMESSAGELOGARBPROC(glfwGetProcAddress("glGetDebugMessageLogARB"));
01698     glGetDoubleIndexedvEXT =
01699     PFNGLGETDOUBLEINDEXEDVEXTPROC(glfwGetProcAddress("glGetDoubleIndexedvEXT"));
01700     glGetDoublei_v = PFNGLGETDOUBLEI_VPROC(glfwGetProcAddress("glGetDoublei_v"));
01701     glGetDoublei_vEXT = PFNGLGETDOUBLEI_VEXTPROC(glfwGetProcAddress("glGetDoublei_vEXT"));
01702     glGetDoublev = PFNGLGETDOUBLEVPROC(glfwGetProcAddress("glGetDoublev"));
01703     glGetError = PFNGLGETERRORPROC(glfwGetProcAddress("glGetError"));
01704     glGetFirstPerfQueryIdINTEL =
01705     PFNGLGETFIRSTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetFirstPerfQueryIdINTEL"));
01706     glGetFloatIndexedvEXT = PFNGLGETFLOATINDEXEDVEXTPROC(glfwGetProcAddress("glGetFloatIndexedvEXT"));
01707     glGetFloati_v = PFNGLGETFLOATI_VPROC(glfwGetProcAddress("glGetFloati_v"));
01708     glGetFloati_vEXT = PFNGLGETFLOATI_VEXTPROC(glfwGetProcAddress("glGetFloati_vEXT"));
01709     glGetFloatv = PFNGLGETFLOATVPROC(glfwGetProcAddress("glGetFloatv"));
01710     glGetFragDataIndex = PFNGLGETFRAGDATAINDEXPROC(glfwGetProcAddress("glGetFragDataIndex"));
01711     glGetFragDataLocation = PFNGLGETFRAGDATALOCATIONPROC(glfwGetProcAddress("glGetFragDataLocation"));
01712     glGetFramebufferAttachmentParameteriv =
01713     PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferAttachmentParameteriv"));
01714     glGetFramebufferParameteriv =
01715     PFNGLGETFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferParameteriv"));
01716     glGetFramebufferParameterivEXT =
01717     PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetFramebufferParameterivEXT"));
01718     glGetGraphicsResetStatus =
01719     PFNGLGETGRAPHICSRESETSTATUSPROC(glfwGetProcAddress("glGetGraphicsResetStatus"));
01720     glGetGraphicsResetStatusARB =
01721     PFNGLGETGRAPHICSRESETSTATUSARBPROC(glfwGetProcAddress("glGetGraphicsResetStatusARB"));
01722     glGetImageHandleARB = PFNGLGETIMAGEHANDLEARBPROC(glfwGetProcAddress("glGetImageHandleARB"));
01723     glGetImageHandleNV = PFNGLGETIMAGEHANDLENVPROC(glfwGetProcAddress("glGetImageHandleNV"));
01724     glGetInteger64i_v = PFNGLGETINTEGER64I_VPROC(glfwGetProcAddress("glGetInteger64i_v"));
01725     glGetInteger64v = PFNGLGETINTEGER64VPROC(glfwGetProcAddress("glGetInteger64v"));
01726     glGetIntegerIndexedvEXT =
01727     PFNGLGETINTEGERINDEXEDVEXTPROC(glfwGetProcAddress("glGetIntegerIndexedvEXT"));
01728     glGetIntegeri_v = PFNGLGETINTEGERI_VPROC(glfwGetProcAddress("glGetIntegeri_v"));
01729     glGetIntegerui64i_vNV = PFNGLGETINTEGERUI64I_VNVPROC(glfwGetProcAddress("glGetIntegerui64i_vNV"));
01730     glGetIntegerui64vNV = PFNGLGETINTEGERUI64NVPROC(glfwGetProcAddress("glGetIntegerui64vNV"));
01731     glGetIntegerv = PFNGLGETINTEGERGENFVEXTPROC(glfwGetProcAddress("glGetIntegerv"));
01732     glGetInternalformatSampleivNV =
01733     PFNGLGETINTERNALFORMATSAMPLEIVNVPROC(glfwGetProcAddress("glGetInternalformatSampleivNV"));
01734     glGetInternalformati64v =
01735     PFNGLGETINTERNALFORMATI64VPROC(glfwGetProcAddress("glGetInternalformati64v"));
01736     glGetInternalformativ = PFNGLGETINTERNALFORMATIVPROC(glfwGetProcAddress("glGetInternalformativ"));
01737     glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01738     glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01739     glGetMultiTexGendvEXT = PFNGLGETMULTITEXGENFVEXTPROC(glfwGetProcAddress("glGetMultiTexGendvEXT"));
01740     glGetMultiTexGenfvEXT = PFNGLGETMULTITEXGENFVEXTPROC(glfwGetProcAddress("glGetMultiTexGenfvEXT"));
01741     glGetMultiTexGenivEXT = PFNGLGETMULTITEXGENIVEXTPROC(glfwGetProcAddress("glGetMultiTexGenivEXT"));
01742     glGetMultiTexImageEXT = PFNGLGETMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetMultiTexImageEXT"));

```

```

01717     glGetMultiTexLevelParameterfvEXT =
01718     PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterfvEXT"));
01719     glGetMultiTexLevelParameterivEXT =
01720     PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterivEXT"));
01721     glGetMultiTexParameterIivEXT =
01722     PFNGLGETMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIivEXT"));
01723     glGetMultiTexParameterIuivEXT =
01724     PFNGLGETMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIuivEXT"));
01725     glGetMultiTexParameterfvEXT =
01726     PFNGLGETMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterfvEXT"));
01727     glGetMultiTexParameterivEXT =
01728     PFNGLGETMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterivEXT"));
01729     glGetMultisamplefv = PFNGLGETMULTISAMPLEFVPROC(glfwGetProcAddress("glGetMultisamplefv"));
01730     glGetNamedBufferParameteri64v =
01731     PFNGLGETNAMEDBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetNamedBufferParameteri64v"));
01732     glGetNamedBufferParameteriv =
01733     PFNGLGETNAMEDBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedBufferParameteriv"));
01734     glGetNamedBufferParameterivEXT =
01735     PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedBufferParameterivEXT"));
01736     glGetNamedBufferParameterui64vNV =
01737     PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC(glfwGetProcAddress("glGetNamedBufferParameterui64vNV"));
01738     glGetNamedBufferPointerv =
01739     PFNGLGETNAMEDBUFFERPOINTERVPROC(glfwGetProcAddress("glGetNamedBufferPointerv"));
01740     glGetNamedBufferPointervEXT =
01741     PFNGLGETNAMEDBUFFERPOINTERVEXTPROC(glfwGetProcAddress("glGetNamedBufferPointervEXT"));
01742     glGetNamedBufferData =
01743     PFNGLGETNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glGetNamedBufferData"));
01744     glGetNamedBufferDataEXT =
01745     PFNGLGETNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glGetNamedBufferDataEXT"));
01746     glGetNamedFramebufferAttachmentParameteriv =
01747     PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameteriv"));
01748     glGetNamedFramebufferAttachmentParameterivEXT =
01749     PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameterivEXT"));
01750     glGetNamedFramebufferParameteri =
01751     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIIVPROC(glfwGetProcAddress("glGetNamedFramebufferParameteri"));
01752     glGetNamedFramebufferParameteriEXT =
01753     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriEXT"));
01754     glGetNamedProgramLocalParameterIivEXT =
01755     PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIivEXT"));
01756     glGetNamedProgramLocalParameterIuivEXT =
01757     PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIuivEXT"));
01758     glGetNamedProgramLocalParameterIuivEXT =
01759     PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIuivEXT"));
01760     glGetNamedProgramLocalParameterdvEXT =
01761     PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterdvEXT"));
01762     glGetNamedProgramLocalParameterdvEXT =
01763     PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterdvEXT"));
01764     glGetNamedProgramStringEXT =
01765     PFNGLGETNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glGetNamedProgramStringEXT"));
01766     glGetNamedProgramivEXT =
01767     PFNGLGETNAMEDPROGRAMIVEXTPROC(glfwGetProcAddress("glGetNamedProgramivEXT"));
01768     glGetNamedRenderbufferParameteriv =
01769     PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedRenderbufferParameteriv"));
01770     glGetNamedRenderbufferParameterivEXT =
01771     PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedRenderbufferParameterivEXT"));
01772     glGetNamedStringARB = PFNGLGETNAMEDSTRINGARBPROC(glfwGetProcAddress("glGetNamedStringARB"));
01773     glGetNamedStringivARB = PFNGLGETNAMEDSTRINGIVARBPROC(glfwGetProcAddress("glGetNamedStringivARB"));
01774     glGetNextPerfQueryIdINTEL =
01775     PFNGLGETNEXTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetNextPerfQueryIdINTEL"));
01776     glGetObjectLabel = PFNGLGETOBJECTLABELPROC(glfwGetProcAddress("glGetObjectLabel"));
01777     glGetObjectLabelEXT = PFNGLGETOBJECTLABELEXTPROC(glfwGetProcAddress("glGetObjectLabelEXT"));
01778     glGetObjectPtrLabel = PFNGLGETOBJECTPTRLABELPROC(glfwGetProcAddress("glGetObjectPtrLabel"));
01779     glGetPathCommandsNV = PFNGLGETPATHCOMMANDSNVPROC(glfwGetProcAddress("glGetPathCommandsNV"));
01780     glGetPathCoordsNV = PFNGLGETPATHCOORDSNVPROC(glfwGetProcAddress("glGetPathCoordsNV"));
01781     glGetPathDashArrayNV = PFNGLGETPATHDASHARRAYNVPROC(glfwGetProcAddress("glGetPathDashArrayNV"));
01782     glGetPathLengthNV = PFNGLGETPATHLENGTHNVPROC(glfwGetProcAddress("glGetPathLengthNV"));
01783     glGetPathMetricRangeNV =
01784     PFNGLGETPATHMETRICRANGEPROC(glfwGetProcAddress("glGetPathMetricRangeNV"));
01785     glGetPathMetricsNV = PFNGLGETPATHMETRICSNVPROC(glfwGetProcAddress("glGetPathMetricsNV"));
01786     glGetPathParameterfvNV =
01787     PFNGLGETPATHPARAMETERFVNVPROC(glfwGetProcAddress("glGetPathParameterfvNV"));
01788     glGetPathParameterivNV =
01789     PFNGLGETPATHPARAMETERIVNVPROC(glfwGetProcAddress("glGetPathParameterivNV"));
01790     glGetPathSpacingNV = PFNGLGETPATHSPACINGNVPROC(glfwGetProcAddress("glGetPathSpacingNV"));
01791     glGetPerfCounterInfoINTEL =
01792     PFNGLGETPERFCOUNTERINFOINTELPROC(glfwGetProcAddress("glGetPerfCounterInfoINTEL"));
01793     glGetPerfMonitorCounterDataAMD =
01794     PFNGLGETPERFMONITORCOUNTERDATAAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterDataAMD"));
01795     glGetPerfMonitorCounterInfoAMD =
01796     PFNGLGETPERFMONITORCOUNTERINFOAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterInfoAMD"));
01797     glGetPerfMonitorCounterStringAMD =
01798     PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterStringAMD"));
01799     glGetPerfMonitorCountersAMD =
01800     PFNGLGETPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glGetPerfMonitorCountersAMD"));
01801     glGetPerfMonitorGroupStringAMD =
01802     PFNGLGETPERFMONITORGROUPSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupStringAMD"));
01803     glGetPerfMonitorGroupsAMD =
01804     PFNGLGETPERFMONITORGROUPSAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupsAMD"));
01805     glGetPerfQueryDataINTEL =

```

```

01765 PFNGLGETPERFQUERYDATAINTELPROC(glfwGetProcAddress("glGetPerfQueryDataINTEL"));
01766     glGetPerfQueryIdByNameINTEL =
01767 PFNGLGETPERFQUERYIDBYNAMEINTELPROC(glfwGetProcAddress("glGetPerfQueryIdByNameINTEL"));
01768     glGetPerfQueryInfoINTEL =
01769 PFNGLGETPERFQUERYINFOINTELPROC(glfwGetProcAddress("glGetPerfQueryInfoINTEL"));
01770     glGetPointerIndexedvEXT =
01771 PFNGLGETPOINTERINDEXEDVEXTPROC(glfwGetProcAddress("glGetPointerIndexedvEXT"));
01772     glGetPointeri_vEXT = PFNGLGETPOINTERI_VEXTPROC(glfwGetProcAddress("glGetPointeri_vEXT"));
01773     glGetProgramBinary = PFNGLGETPROGRAMBINARYPROC(glfwGetProcAddress("glGetProgramBinary"));
01774     glGetProgramInfoLog = PFNGLGETPROGRAMINFOLOGPROC(glfwGetProcAddress("glGetProgramInfoLog"));
01775     glGetProgramInterfaceiv =
01776 PFNGLGETPROGRAMINTERFACEIVPROC(glfwGetProcAddress("glGetProgramInterfaceiv"));
01777     glGetProgramPipelineInfoLog =
01778 PFNGLGETPROGRAMPIPELINEINFOLOGPROC(glfwGetProcAddress("glGetProgramPipelineInfoLog"));
01779     glGetProgramPipelineiv =
01780 PFNGLGETPROGRAMPIPELINEIVPROC(glfwGetProcAddress("glGetProgramPipelineiv"));
01781     glGetProgramResourceIndex =
01782 PFNGLGETPROGRAMRESOURCEINDEXPROC(glfwGetProcAddress("glGetProgramResourceIndex"));
01783     glGetProgramResourceLocation =
01784 PFNGLGETPROGRAMRESOURCELLOCATIONPROC(glfwGetProcAddress("glGetProgramResourceLocation"));
01785     glGetProgramResourceLocationIndex =
01786 PFNGLGETPROGRAMRESOURCELLOCATIONINDEXPROC(glfwGetProcAddress("glGetProgramResourceLocationIndex"));
01787     glGetProgram resourceName =
01788 PFNGLGETPROGRAMRESOURCENAMEPROC(glfwGetProcAddress("glGetProgram resourceName"));
01789     glGetProgramResourcefvNV =
01790 PFNGLGETPROGRAMRESOURCEFVNVPROC(glfwGetProcAddress("glGetProgramResourcefvNV"));
01791     glGetProgramResourceiv =
01792 PFNGLGETPROGRAMRESOURCEIVPROC(glfwGetProcAddress("glGetProgramResourceiv"));
01793     glGetProgramStageiv = PFNGLGETPROGRAMSTAGEIVPROC(glfwGetProcAddress("glGetProgramStageiv"));
01794     glGetProgramiv = PFNGLGETPROGRAMIVPROC(glfwGetProcAddress("glGetProgramiv"));
01795     glGetQueryBufferObjecti64v =
01796 PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
01797     glGetQueryBufferObjecti64v =
01798 PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
01799     glGetQueryBufferObjectui64v =
01800 PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
01801     glGetQueryBufferObjectui64v =
01802 PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
01803     glGetQueryObjecti64v =
01804 PFNGLGETQUERYOBJECTI64VPROC(glfwGetProcAddress("glGetQueryObjecti64v"));
01805     glGetQueryObjectui64v =
01806 PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
01807     glGetQueryObjectuiv =
01808 PFNGLGETQUERYOBJECTUIVPROC(glfwGetProcAddress("glGetQueryObjectuiv"));
01809     glGetQueryIndexediv =
01810 PFNGLGETQUERYINDEXEDIVPROC(glfwGetProcAddress("glGetQueryIndexediv"));
01811     glGetQueryObjectui64v =
01812 PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
01813     glGetQueryObjectiv =
01814 PFNGLGETQUERYOBJECTIVPROC(glfwGetProcAddress("glGetQueryObjectiv"));
01815     glGetString =
01816 PFNGLGETSTRINGPROC(glfwGetProcAddress("glGetString"));
01817     glGetStringi =
01818 PFNGLGETSTRINGIPROC(glfwGetProcAddress("glGetStringi"));
01819     glGetSubroutineIndex =
01820 PFNGLGETSUBROUTINEINDEXPROC(glfwGetProcAddress("glGetSubroutineIndex"));
01821     glGetSubroutineUniformLocation =
01822 PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetSubroutineUniformLocation"));
01823     glGetSynciv =
01824 PFNGLGETSYNCIVPROC(glfwGetProcAddress("glGetSynciv"));
01825     glGetTexImage =
01826 PFNGLGETTEXIMAGEPROC(glfwGetProcAddress("glGetTexImage"));
01827     glGetTexLevelParameterfv =
01828 PFNGLGETTEXLEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTexLevelParameterfv"));
01829     glGetTexLevelParameteriv =
01830 PFNGLGETTEXLEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTexLevelParameteriv"));
01831     glGetTexParameterIiiv =
01832 PFNGLGETTEXPARAMETERIIIVPROC(glfwGetProcAddress("glGetTexParameterIiiv"));
01833     glGetTexParameterIiui =
01834 PFNGLGETTEXPARAMETERIIUIVPROC(glfwGetProcAddress("glGetTexParameterIiui"));
01835     glGetTexParameterfv =
01836 PFNGLGETTEXPARAMETERFVPROC(glfwGetProcAddress("glGetTexParameterfv"));
01837     glGetTexParameteriv =
01838 PFNGLGETTEXPARAMETERIVPROC(glfwGetProcAddress("glGetTexParameteriv"));
01839     glGetTextureHandleARB =
01840 PFNGLGETTEXTUREHANDLEARBPROC(glfwGetProcAddress("glGetTextureHandleARB"));
01841     glGetTextureHandleNV =
01842 PFNGLGETTEXTUREHANDLENVPROC(glfwGetProcAddress("glGetTextureHandleNV"));
01843     glGetTextureImage =
01844 PFNGLGETTEXTUREIMAGEPROC(glfwGetProcAddress("glGetTextureImage"));
01845     glGetTextureImageEXT =
01846 PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTextureImageEXT"));
01847     glGetTextureLevelParameterfv =
01848 PFNGLGETTEXTURELEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTextureLevelParameterfv"));
01849     glGetTextureLevelParameterfvEXT =
01850 PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterfvEXT"));
01851     glGetTextureLevelParameteriv =
01852 PFNGLGETTEXTURELEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTextureLevelParameteriv"));
01853     glGetTextureLevelParameterivEXT =

```

```

01825 PFNGLGETTEXTURELEVELPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameteriivEXT"));
01825     glGetTextureParameterIiv =
01826 PFNGLGETTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glGetTextureParameterIiv"));
01826     glGetTextureParameterIivEXT =
01827 PFNGLGETTEXTUREPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIivEXT"));
01827     glGetTextureParameterIiuv =
01828 PFNGLGETTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glGetTextureParameterIiuv"));
01828     glGetTextureParameterIiuvEXT =
01829 PFNGLGETTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIiuvEXT"));
01829     glGetTextureParameterIfv =
01830 PFNGLGETTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glGetTextureParameterfv"));
01830     glGetTextureParameterfvEXT =
01831 PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
01831     glGetTextureParameteriv =
01832 PFNGLGETTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glGetTextureParameteriv"));
01832     glGetTextureParameterivEXT =
01833 PFNGLGETTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureParameterivEXT"));
01833     glGetTextureSamplerHandleARB =
01834 PFNGLGETTEXTURESAMPLERHANDLEARBPROC(glfwGetProcAddress("glGetTextureSamplerHandleARB"));
01834     glGetTextureSamplerHandleEnv =
01835 PFNGLGETTEXTURESAMPLERHANDLENVPROC(glfwGetProcAddress("glGetTextureSamplerHandleNV"));
01835     glGetTextureSubImage = PFNGLGETTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetTextureSubImage"));
01836     glGetTransformFeedbackVarying =
01837 PFNGLGETTRANSFORMFEEDBACKVARYINGPROC(glfwGetProcAddress("glGetTransformFeedbackVarying"));
01837     glGetTransformFeedbacki64_v =
01838 PFNGLGETTRANSFORMFEEDBACKI64_VPROC(glfwGetProcAddress("glGetTransformFeedbacki64_v"));
01838     glGetTransformFeedbacki_v =
01839 PFNGLGETTRANSFORMFEEDBACKI_VPROC(glfwGetProcAddress("glGetTransformFeedbacki_v"));
01839     glGetTransformFeedbackiv =
01840 PFNGLGETTRANSFORMFEEDBACKIVPROC(glfwGetProcAddress("glGetTransformFeedbackiv"));
01840     glGetUniformBlockIndex =
01841 PFNGLGETUNIFORMBLOCKINDEXPROC(glfwGetProcAddress("glGetUniformBlockIndex"));
01841     glGetUniformIndices = PFNGLGETUNIFORMINDICESPROC(glfwGetProcAddress("glGetUniformIndices"));
01842     glGetUniformLocation = PFNGLGETUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetUniformLocation"));
01843     glGetUniformSubroutineuiv =
01844 PFNGLGETUNIFORMSUBROUTINEUIVPROC(glfwGetProcAddress("glGetUniformSubroutineuiv"));
01844     glGetUniformdv = PFNGLGETUNIFORMDVPROC(glfwGetProcAddress("glGetUniformdv"));
01845     glGetUniformfv = PFNGLGETUNIFORMFVPROC(glfwGetProcAddress("glGetUniformfv"));
01846     glGetUniformi64vARB = PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64vARB"));
01847     glGetUniformi64vNV = PFNGLGETUNIFORMI64VNPROC(glfwGetProcAddress("glGetUniformi64vNV"));
01848     glGetUniformiv = PFNGLGETUNIFORMIVPROC(glfwGetProcAddress("glGetUniformiv"));
01849     glGetUniformui64vARB = PFNGLGETUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetUniformui64vARB"));
01850     glGetUniformui64vNV = PFNGLGETUNIFORMUI64VNPROC(glfwGetProcAddress("glGetUniformui64vNV"));
01851     glGetUniformuiv = PFNGLGETUNIFORMUIVPROC(glfwGetProcAddress("glGetUniformuiv"));
01852     glGetVertexArrayIndexed64iv =
01853 PFNGLGETVERTEXARRAYINDEXED64IVPROC(glfwGetProcAddress("glGetVertexArrayIndexed64iv"));
01853     glGetVertexArrayIndexeddiv =
01854 PFNGLGETVERTEXARRAYINDEXEDIVPROC(glfwGetProcAddress("glGetVertexArrayIndexeddiv"));
01854     glGetVertexArrayIntegeri_vEXT =
01855 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegeri_vEXT"));
01855     glGetVertexArrayIntegervEXT =
01856 PFNGLGETVERTEXARRAYINTEGERVEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegervEXT"));
01856     glGetVertexArrayPointeri_vEXT =
01857 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayPointeri_vEXT"));
01857     glGetVertexArrayPointervEXT =
01858 PFNGLGETVERTEXARRAYPOINTERVEXTPROC(glfwGetProcAddress("glGetVertexArrayPointervEXT"));
01858     glGetVertexArrayiv = PFNGLGETVERTEXARRAYIVPROC(glfwGetProcAddress("glGetVertexArrayiv"));
01859     glGetVertexAttribArrayIiiv = PFNGLGETVERTEXATTRIBIIVPROC(glfwGetProcAddress("glGetVertexAttribArrayIiiv"));
01860     glGetVertexAttribArrayIiuv = PFNGLGETVERTEXATTRIBIUIVPROC(glfwGetProcAddress("glGetVertexAttribArrayIiuv"));
01861     glGetVertexAttribArrayIiudv = PFNGLGETVERTEXATTRIBLUDVPROC(glfwGetProcAddress("glGetVertexAttribArrayIiudv"));
01862     glGetVertexAttribArrayIi64vNV =
01863 PFNGLGETVERTEXATTRIBL64VNPROC(glfwGetProcAddress("glGetVertexAttribArrayIi64vNV"));
01863     glGetVertexAttribArrayIi64vARB =
01864 PFNGLGETVERTEXATTRIBLUI64VARBPROC(glfwGetProcAddress("glGetVertexAttribArrayIi64vARB"));
01864     glGetVertexAttribArrayIi64vNV =
01865 PFNGLGETVERTEXATTRIBLUI64VNPROC(glfwGetProcAddress("glGetVertexAttribArrayIi64vNV"));
01865     glGetVertexAttribArrayPointerv =
01866 PFNGLGETVERTEXATTRIBPOINTERVPROC(glfwGetProcAddress("glGetVertexAttribArrayPointerv"));
01866     glGetVertexAttribArrayAttribbdv = PFNGLGETVERTEXATTRIBBDVPROC(glfwGetProcAddress("glGetVertexAttribArrayAttribbdv"));
01867     glGetVertexAttribArrayAttribfv = PFNGLGETVERTEXATTRIBFVPROC(glfwGetProcAddress("glGetVertexAttribArrayAttribfv"));
01868     glGetVertexAttribArrayAttribiv = PFNGLGETVERTEXATTRIBIVPROC(glfwGetProcAddress("glGetVertexAttribArrayAttribiv"));
01869     glGetVKProcAddrNV = PFNGLGETVKPROCAADDRNVPROC(glfwGetProcAddress("glGetVKProcAddrNV"));
01870     glGetCompressedTexImage =
01871 PFNGLGETNCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetnCompressedTexImage"));
01871     glGetnCompressedTexImageARB =
01872 PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC(glfwGetProcAddress("glGetnCompressedTexImageARB"));
01872     glGetnTexImage = PFNGLGETNTEXIMAGEPROC(glfwGetProcAddress("glGetnTexImage"));
01873     glGetnTexImageARB = PFNGLGETNTEXIMAGEARBPROC(glfwGetProcAddress("glGetnTexImageARB"));
01874     glGetnUniformdv = PFNGLGETNUNIFORMDVPROC(glfwGetProcAddress("glGetnUniformdv"));
01875     glGetnUniformdvarb = PFNGLGETNUNIFORMDVARBPROC(glfwGetProcAddress("glGetnUniformdvarb"));
01876     glGetnUniformfv = PFNGLGETNUNIFORMFVPROC(glfwGetProcAddress("glGetnUniformfv"));
01877     glGetnUniformfvarb = PFNGLGETNUNIFORMFVARBPROC(glfwGetProcAddress("glGetnUniformfvarb"));
01878     glGetnUniformi64vARB = PFNGLGETNUNIFORMI64VARBPROC(glfwGetProcAddress("glGetnUniformi64vARB"));
01879     glGetnUniformiv = PFNGLGETNUNIFORMIVPROC(glfwGetProcAddress("glGetnUniformiv"));
01880     glGetnUniformivARB = PFNGLGETNUNIFORMIVARBPROC(glfwGetProcAddress("glGetnUniformivARB"));
01881     glGetnUniformui64vARB = PFNGLGETNUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetnUniformui64vARB"));
01882     glGetnUniformuiv = PFNGLGETNUNIFORMUIVPROC(glfwGetProcAddress("glGetnUniformuiv"));

```

```

01883     glGetnUniformuivARB = PFNGLGETNUNIFORMUIVARBPROC(glfwGetProcAddress("glGetnUniformuivARB"));
01884     glHint = PFNGLHINTPROC(glfwGetProcAddress("glHint"));
01885     glIndexFormatNV = PFNGLINDEXFORMATNVPROC(glfwGetProcAddress("glIndexFormatNV"));
01886     glInsertEventMarkerEXT =
01887         PFNGLINSETEVENTMARKEREXTPROC(glfwGetProcAddress("glInsertEventMarkerEXT"));
01888     glInterpolatePathsNV = PFNGLINTERPOLATEPATHSNVPROC(glfwGetProcAddress("glInterpolatePathsNV"));
01889     glInvalidateBufferData =
01890         PFNGLINVALIDATEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
01891     glInvalidateBufferData =
01892         PFNGLINVALIDATEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
01893     glInvalidateFramebuffer =
01894         PFNGLINVALIDATEFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateFramebuffer"));
01895     glInvalidateNamedFramebufferData =
01896         PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferData"));
01897     glInvalidateNamedFramebufferSubData =
01898         PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferSubData"));
01899     glInvalidateSubFramebuffer =
01900         PFNGLINVALIDATESUBFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateSubFramebuffer"));
01901     glInvalidateTexImage = PFNGLINVALIDATETEXIMAGEPROC(glfwGetProcAddress("glInvalidateTexImage"));
01902     glInvalidateTexSubImage =
01903         PFNGLINVALIDATETEXSUBIMAGEPROC(glfwGetProcAddress("glInvalidateTexSubImage"));
01904     glIsBuffer = PFNGLISBUFFERPROC(glfwGetProcAddress("glIsBuffer"));
01905     glIsBufferResidentNV = PFNGLISBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsBufferResidentNV"));
01906     glIsCommandListNV = PFNGLISCOMMANDLISTNVPROC(glfwGetProcAddress("glIsCommandListNV"));
01907     glIsEnabled = PFNGLISENABLEDPROC(glfwGetProcAddress("glIsEnabled"));
01908     glIsEnabledIndexedEXT = PFNGLISENABLEDEXTEDPROC(glfwGetProcAddress("glIsEnabledIndexedEXT"));
01909     glIsEnablededi = PFNGLISENABLEDEDIPROC(glfwGetProcAddress("glIsEnablededi"));
01910     glIsFramebuffer = PFNGLISFRAMEBUFFERPROC(glfwGetProcAddress("glIsFramebuffer"));
01911     glIsImageHandleResidentARB =
01912         PFNGLISIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsImageHandleResidentARB"));
01913     glIsImageHandleResidentNV =
01914         PFNGLISIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsImageHandleResidentNV"));
01915     glIsNamedBufferResidentNV =
01916         PFNGLISNAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsNamedBufferResidentNV"));
01917     glIsNamedStringARB = PFNGLISNAMEDSTRINGARBPROC(glfwGetProcAddress("glIsNamedStringARB"));
01918     glIsPathNV = PFNGLISPATHNVPROC(glfwGetProcAddress("glIsPathNV"));
01919     glIsPointInFillPathNV = PFNGLISPOINTINFILLPATHNVPROC(glfwGetProcAddress("glIsPointInFillPathNV"));
01920     glIsPointInStrokePathNV =
01921         PFNGLISPOINTINSTROKEPATHNVPROC(glfwGetProcAddress("glIsPointInStrokePathNV"));
01922     glIsProgram = PFNGLISPROGRAMPROC(glfwGetProcAddress("glIsProgram"));
01923     glIsProgramPipeline = PFNGLISPROGRAMPIPELINEPROC(glfwGetProcAddress("glIsProgramPipeline"));
01924     glIsQuery = PFNGLISQUERYPROC(glfwGetProcAddress("glIsQuery"));
01925     glIsRenderbuffer = PFNGLISRENDERBUFFERPROC(glfwGetProcAddress("glIsRenderbuffer"));
01926     glIsSampler = PFNGLISSAMPLERPROC(glfwGetProcAddress("glIsSampler"));
01927     glIsShader = PFNGLISSHADERPROC(glfwGetProcAddress("glIsShader"));
01928     glIsStateNV = PFNGLISSTATEENVPROC(glfwGetProcAddress("glIsStateNV"));
01929     glIsSync = PFNGLISSYNCPROC(glfwGetProcAddress("glIsSync"));
01930     glIsTexture = PFNGLISTEXTUREPROC(glfwGetProcAddress("glIsTexture"));
01931     glIsTextureHandleResidentARB =
01932         PFNGLISTTEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsTextureHandleResidentARB"));
01933     glIsTextureHandleResidentNV =
01934         PFNGLISTTEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsTextureHandleResidentNV"));
01935     glIsTransformFeedback = PFNGLISTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glIsTransformFeedback"));
01936     glIsVertexArray = PFNGLISVERTEXARRAYPROC(glfwGetProcAddress("glIsVertexArray"));
01937     glLabelObjectEXT = PFNGLLABELOBJECTEXTPROC(glfwGetProcAddress("glLabelObjectEXT"));
01938     glLineWidth = PFNGLLINEWIDTHPROC(glfwGetProcAddress("glLineWidth"));
01939     glLinkProgram = PFNGLLINKPROGRAMPROC(glfwGetProcAddress("glLinkProgram"));
01940     glListDrawCommandsStatesClientNV =
01941         PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC(glfwGetProcAddress("glListDrawCommandsStatesClientNV"));
01942     glLogicOp = PFNGLLOGICOPPROC(glfwGetProcAddress("glLogicOp"));
01943     glMakeBufferNonResidentNV =
01944         PFNGLMAKEBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferNonResidentNV"));
01945     glMakeBufferResidentNV =
01946         PFNGLMAKEBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferResidentNV"));
01947     glMakeImageHandleNonResidentARB =
01948         PFNGLMAKEIMAGENONRESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleNonResidentARB"));
01949     glMakeImageHandleNonResidentNV =
01950         PFNGLMAKEIMAGENONRESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleNonResidentNV"));
01951     glMakeImageHandleResidentARB =
01952         PFNGLMAKEIMAGERESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleResidentARB"));
01953     glMakeImageHandleResidentNV =
01954         PFNGLMAKEIMAGERESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleResidentNV"));
01955     glMakeNamedBufferNonResidentNV =
01956         PFNGLMAKEBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferNonResidentNV"));
01957     glMakeNamedBufferResidentNV =
01958         PFNGLMAKEBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferResidentNV"));
01959     glMakeTextureHandleNonResidentARB =
01960         PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentARB"));
01961     glMakeTextureHandleNonResidentNV =
01962         PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentNV"));
01963     glMakeTextureHandleResidentARB =
01964         PFNGLMAKETEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleResidentARB"));
01965     glMakeTextureHandleResidentNV =
01966         PFNGLMAKETEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleResidentNV"));
01967     glMapBuffer = PFNGLMAPBUFFERPROC(glfwGetProcAddress("glMapBuffer"));
01968     glMapBufferRange = PFNGLMAPBUFFERRANGEPROC(glfwGetProcAddress("glMapBufferRange"));
01969     glMapNamedBuffer = PFNGLMAPNAMEDBUFFERPROC(glfwGetProcAddress("glMapNamedBuffer"));

```

```

01943     glMapNamedBufferEXT = PFNGLMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glMapNamedBufferEXT"));
01944     glMapNamedBufferRange = PFNGLMAPNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glMapNamedBufferRange"));
01945     glMapNamedBufferRangeEXT =
01946         PFNGLMAPNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glMapNamedBufferRangeEXT"));
01947     glMatrixFrustumEXT = PFNGLMATRIXFRUSTUMEXTPROC(glfwGetProcAddress("glMatrixFrustumEXT"));
01948     glMatrixLoad3x2fNV = PFNGLMATRIXLOAD3X2FNVPROC(glfwGetProcAddress("glMatrixLoad3x2fNV"));
01949     glMatrixLoad3x3fNV = PFNGLMATRIXLOAD3X3FNVPROC(glfwGetProcAddress("glMatrixLoad3x3fNV"));
01949     glMatrixLoadIdentityEXT =
01950         PFNGLMATRIXLOADIDENTITYEXTPROC(glfwGetProcAddress("glMatrixLoadIdentityEXT"));
01950     glMatrixLoadTranspose3x3fNV =
01951         PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixLoadTranspose3x3fNV"));
01951     glMatrixLoadTransposedEXT =
01952         PFNGLMATRIXLOADTRANSPOSEDEXTPROC(glfwGetProcAddress("glMatrixLoadTransposedEXT"));
01952     glMatrixLoadTransposefEXT =
01953         PFNGLMATRIXLOADTRANSPOSEFEXTPROC(glfwGetProcAddress("glMatrixLoadTransposefEXT"));
01953     glMatrixLoaddEXT = PFNGLMATRIXLOADDEXTPROC(glfwGetProcAddress("glMatrixLoaddEXT"));
01954     glMatrixLoadfEXT = PFNGLMATRIXLOADFEXTPROC(glfwGetProcAddress("glMatrixLoadfEXT"));
01955     glMatrixMult3x2fNV = PFNGLMATRIXMULT3X2FNVPROC(glfwGetProcAddress("glMatrixMult3x2fNV"));
01956     glMatrixMult3x3fNV = PFNGLMATRIXMULT3X3FNVPROC(glfwGetProcAddress("glMatrixMult3x3fNV"));
01957     glMatrixMultTranspose3x3fNV =
01958         PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixMultTranspose3x3fNV"));
01958     glMatrixMultTransposedEXT =
01959         PFNGLMATRIXMULTTRANSPOSEDEXTPROC(glfwGetProcAddress("glMatrixMultTransposedEXT"));
01959     glMatrixMultTransposefEXT =
01960         PFNGLMATRIXMULTTRANSPOSEFEXTPROC(glfwGetProcAddress("glMatrixMultTransposefEXT"));
01961     glMatrixMultdEXT = PFNGLMATRIXMULTDEXTPROC(glfwGetProcAddress("glMatrixMultdEXT"));
01962     glMatrixMultfEXT = PFNGLMATRIXMULTFEXTPROC(glfwGetProcAddress("glMatrixMultfEXT"));
01962     glMatrixOrthoEXT = PFNGLMATRIXORTHOEXTPROC(glfwGetProcAddress("glMatrixOrthoEXT"));
01963     glMatrixPopEXT = PFNGLMATRIXPOPEXTPROC(glfwGetProcAddress("glMatrixPopEXT"));
01964     glMatrixPushEXT = PFNGLMATRIXPUSHEXTPROC(glfwGetProcAddress("glMatrixPushEXT"));
01965     glMatrixRotatedEXT = PFNGLMATRIXROTATEDEXTPROC(glfwGetProcAddress("glMatrixRotatedEXT"));
01966     glMatrixRotatefEXT = PFNGLMATRIXROTATEFEXTPROC(glfwGetProcAddress("glMatrixRotatefEXT"));
01967     glMatrixScaledEXT = PFNGLMATRIXSCALEDEXTPROC(glfwGetProcAddress("glMatrixScaledEXT"));
01968     glMatrixScalefEXT = PFNGLMATRIXSCALEFEXTPROC(glfwGetProcAddress("glMatrixScalefEXT"));
01969     glMatrixTranslatedEXT = PFNGLMATRIXTRANSLATEDEXTPROC(glfwGetProcAddress("glMatrixTranslatedEXT"));
01970     glMatrixTranslatefEXT = PFNGLMATRIXTRANSLATEFEXTPROC(glfwGetProcAddress("glMatrixTranslatefEXT"));
01971     glMaxShaderCompilerThreadsARB =
01972         PFNGLMAXSHADERCOMPILERTHREADSARBPROC(glfwGetProcAddress("glMaxShaderCompilerThreadsARB"));
01972     glMemoryBarrier = PFNGLMEMORYBARRIERPROC(glfwGetProcAddress("glMemoryBarrier"));
01973     glMemoryBarrierByRegion =
01974         PFNGLMEMORYBARRIERBYREGIONPROC(glfwGetProcAddress("glMemoryBarrierByRegion"));
01974     glMinSampleShading = PFNGLMINSAMPLESHADINGPROC(glfwGetProcAddress("glMinSampleShading"));
01975     glMinSampleShadingARB = PFNGLMINSAMPLESHADINGARBPROC(glfwGetProcAddress("glMinSampleShadingARB"));
01976     glMultiDrawArrays = PFNGLMULTIDRAWARRAYSPROC(glfwGetProcAddress("glMultiDrawArrays"));
01977     glMultiDrawArraysIndirect =
01978         PFNGLMULTIDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glMultiDrawArraysIndirect"));
01978     glMultiDrawArraysIndirectBindlessCountNV =
01979         PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessCountNV"));
01979     glMultiDrawArraysIndirectBindlessNV =
01980         PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessNV"));
01980     glMultiDrawArraysIndirectCountARB =
01981         PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawArraysIndirectCountARB"));
01982     glMultiDrawElements = PFNGLMULTIDRAWELEMENTSPROC(glfwGetProcAddress("glMultiDrawElements"));
01982     glMultiDrawElementsBaseVertex =
01983         PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glMultiDrawElementsBaseVertex"));
01983     glMultiDrawElementsIndirect =
01984         PFNGLMULTIDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glMultiDrawElementsIndirect"));
01984     glMultiDrawElementsIndirectBindlessCountNV =
01985         PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessCountNV"));
01985     glMultiDrawElementsIndirectBindlessNV =
01986         PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessNV"));
01986     glMultiDrawElementsIndirectCountARB =
01987         PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawElementsIndirectCountARB"));
01987     glMultiTexBufferEXT = PFNGLMULTITEXBUFFEREXTPROC(glfwGetProcAddress("glMultiTexBufferEXT"));
01988     glMultiTexCoordPointerEXT =
01989         PFNGLMULTITEXCOORDPOINTEREXTPROC(glfwGetProcAddress("glMultiTexCoordPointerEXT"));
01989     glMultiTexEnvfEXT = PFNGLMULTITEXENVFEXTPROC(glfwGetProcAddress("glMultiTexEnvfEXT"));
01990     glMultiTexEnvfvEXT = PFNGLMULTITEXENVFVEXTPROC(glfwGetProcAddress("glMultiTexEnvfvEXT"));
01991     glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
01992     glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
01993     glMultiTexGendEXT = PFNGLMULTITEXGENDEXTPROC(glfwGetProcAddress("glMultiTexGendEXT"));
01994     glMultiTexGendvEXT = PFNGLMULTITEXGENDVEXTPROC(glfwGetProcAddress("glMultiTexGendvEXT"));
01995     glMultiTexGenfEXT = PFNGLMULTITEXGENFEXTPROC(glfwGetProcAddress("glMultiTexGenfEXT"));
01996     glMultiTexGenfvEXT = PFNGLMULTITEXGENFVEXTPROC(glfwGetProcAddress("glMultiTexGenfvEXT"));
01997     glMultiTexGeniEXT = PFNGLMULTITEXGENIEXTPROC(glfwGetProcAddress("glMultiTexGeniEXT"));
01998     glMultiTexGenivEXT = PFNGLMULTITEXGENIVEXTPROC(glfwGetProcAddress("glMultiTexGenivEXT"));
01999     glMultiTexImage1DEXT = PFNGLMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexImage1DEXT"));
02000     glMultiTexImage2DEXT = PFNGLMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexImage2DEXT"));
02001     glMultiTexImage3DEXT = PFNGLMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexImage3DEXT"));
02002     glMultiTexParameterIiEXT =
02003         PFNGLMULTITEXPARAMETERIIEXTPROC(glfwGetProcAddress("glMultiTexParameterIiEXT"));
02003     glMultiTexParameterIiivEXT =
02004         PFNGLMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIuiivEXT"));
02004     glMultiTexParameterfEXT =
02005         PFNGLMULTITEXPARAMETERFEXTPROC(glfwGetProcAddress("glMultiTexParameterfEXT"));
02005     glMultiTexParameterfvEXT =
02005         PFNGLMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glMultiTexParameterfvEXT"));

```

```

02006     glMultiTexParameteriEXT =
02007     PFNGLMULTITEXPARAMETERIEXTPROC(glfwGetProcAddress("glMultiTexParameteriEXT"));
02008     glMultiTexParameterivEXT =
02009     PFNGLMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glMultiTexParameterivEXT"));
02010     glMultiTexRenderbufferEXT =
02011     PFNGLMULTITEXRENDERBUFFEREXTPROC(glfwGetProcAddress("glMultiTexRenderbufferEXT"));
02012     glMultiTexSubImage1DEXT =
02013     PFNGLMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexSubImage1DEXT"));
02014     glMultiTexSubImage2DEXT =
02015     PFNGLMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexSubImage2DEXT"));
02016     glMultiTexSubImage3DEXT =
02017     PFNGLMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexSubImage3DEXT"));
02018     glNamedBufferData = PFNGLNAMEDBUFFERDATAPROC(glfwGetProcAddress("glNamedBufferData"));
02019     glNamedBufferDataEXT = PFNGLNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glNamedBufferDataEXT"));
02020     glNamedBufferPageCommitmentARB =
02021     PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glNamedBufferPageCommitmentARB"));
02022     glNamedBufferPageCommitmentEXT =
02023     PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glNamedBufferPageCommitmentEXT"));
02024     glNamedBufferStorage = PFNGLNAMEDBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedBufferStorage"));
02025     glNamedBufferStorageEXT =
02026     PFNGLNAMEDBUFFERSTORAGEXTPROC(glfwGetProcAddress("glNamedBufferStorageEXT"));
02027     glNamedBufferSubData = PFNGLNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glNamedBufferSubData"));
02028     glNamedBufferSubDataEXT =
02029     PFNGLNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedBufferSubDataEXT"));
02030     glNamedCopyBufferSubDataEXT =
02031     PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedCopyBufferSubDataEXT"));
02032     glNamedFramebufferDrawBuffer =
02033     PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffer"));
02034     glNamedFramebufferDrawBuffers =
02035     PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffers"));
02036     glNamedFramebufferParameteri =
02037     PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glNamedFramebufferParameteri"));
02038     glNamedFramebufferParameteriEXT =
02039     PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC(glfwGetProcAddress("glNamedFramebufferParameteriEXT"));
02040     glNamedFramebufferReadBuffer =
02041     PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC(glfwGetProcAddress("glNamedFramebufferReadBuffer"));
02042     glNamedFramebufferRenderbuffer =
02043     PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glNamedFramebufferRenderbuffer"));
02044     glNamedFramebufferRenderbufferEXT =
02045     PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC(glfwGetProcAddress("glNamedFramebufferRenderbufferEXT"));
02046     glNamedFramebufferSampleLocationsfvARB =
02047     PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvARB"));
02048     glNamedFramebufferSampleLocationsfvNV =
02049     PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvNV"));
02050     glNamedFramebufferTexture =
02051     PFNGLNAMEDFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glNamedFramebufferTexture"));

```

```

PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glfwGetProcAddress("glNamedRenderbufferStorageEXT"));
02052 glnamedRenderbufferStorageMultisample =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glfwGetProcAddress("glNamedRenderbufferStorageMultisample");
02053 glnamedRenderbufferStorageMultisampleCoverageEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC glfwGetProcAddress("glNamedRenderbufferStorageMultisampleCoverageEXT");
02054 glnamedRenderbufferStorageMultisampleEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glfwGetProcAddress("glNamedRenderbufferStorageMultisampleEXT");
02055 glnamedStringARB = PFNGLNAMEDSTRINGARBPROC glfwGetProcAddress("glNamedStringARB");
02056 glnormalFormatNV = PFNGLNORMALFORMATNVPROC glfwGetProcAddress("glNormalFormatNV");
02057 glabelObjectLabel = PFNGLOBJECTLABELPROC glfwGetProcAddress("glabelObjectLabel");
02058 glabelObjectPtrLabel = PFNGLOBJECTPTRLABELPROC glfwGetProcAddress("glabelObjectPtrLabel");
02059 glabelPatchParameterfv = PFNGLPATCHPARAMETERFVPROC glfwGetProcAddress("glabelPatchParameterfv");
02060 glabelPatchParameteri = PFNGLPATCHPARAMETERIPROC glfwGetProcAddress("glabelPatchParameteri");
02061 glabelPathCommandsNV = PFNGLPATHCOMMANDSNVPROC glfwGetProcAddress("glabelPathCommandsNV");
02062 glabelPathCoordsNV = PFNGLPATHCOORDSNVPROC glfwGetProcAddress("glabelPathCoordsNV");
02063 glabelPathCoverDepthFuncNV =
PFNGLPATHCOVERDEPTHFUNCNVPROC glfwGetProcAddress("glabelPathCoverDepthFuncNV");
02064 glabelPathDashArrayNV = PFNGLPATHDASHARRAYNVPROC glfwGetProcAddress("glabelPathDashArrayNV");
02065 glabelPathGlyphIndexArrayNV =
PFNGLPATHGLYPHINDEXARRAYNVPROC glfwGetProcAddress("glabelPathGlyphIndexArrayNV");
02066 glabelPathGlyphIndexRangeNV =
PFNGLPATHGLYPHINDEXRANGENVPROC glfwGetProcAddress("glabelPathGlyphIndexRangeNV");
02067 glabelPathGlyphRangeNV = PFNGLPATHGLYPHRANGENVPROC glfwGetProcAddress("glabelPathGlyphRangeNV");
02068 glabelPathGlyphsNV = PFNGLPATHGLYPHSNVPROC glfwGetProcAddress("glabelPathGlyphsNV");
02069 glabelPathMemoryGlyphIndexArrayNV =
PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC glfwGetProcAddress("glabelPathMemoryGlyphIndexArrayNV");
02070 glabelPathParameterfNV = PFNGLPATHPARAMETERFNVPROC glfwGetProcAddress("glabelPathParameterfNV");
02071 glabelPathParameterfvNV = PFNGLPATHPARAMETERFVNVPROC glfwGetProcAddress("glabelPathParameterfvNV");
02072 glabelPathParameteriNV = PFNGLPATHPARAMETERINVNPROC glfwGetProcAddress("glabelPathParameteriNV");
02073 glabelPathParameterivNV = PFNGLPATHPARAMETERIVNVPROC glfwGetProcAddress("glabelPathParameterivNV");
02074 glabelPathStencilDepthOffsetNV =
PFNGLPATHSTENCILDEPTHOFFSETNVPROC glfwGetProcAddress("glabelPathStencilDepthOffsetNV");
02075 glabelPathStencilFuncNV = PFNGLPATHSTENCILFUNCNVPROC glfwGetProcAddress("glabelPathStencilFuncNV");
02076 glabelPathStringNV = PFNGLPATHSTRINGNVPROC glfwGetProcAddress("glabelPathStringNV");
02077 glabelPathSubCommandsNV = PFNGLPATHSUBCOMMANDSNVPROC glfwGetProcAddress("glabelPathSubCommandsNV");
02078 glabelPathSubCoordsNV = PFNGLPATHSUBCOORDSNVPROC glfwGetProcAddress("glabelPathSubCoordsNV");
02079 glabelPauseTransformFeedback =
PFNGLPAUSETRANSFORMFEEDBACKPROC glfwGetProcAddress("glabelPauseTransformFeedback");
02080 glabelPixelStoref = PFNGLPIXELSTOREFFPROC glfwGetProcAddress("glabelPixelStoref");
02081 glabelPixelStorei = PFNGLPIXELSTOREIPROC glfwGetProcAddress("glabelPixelStorei");
02082 glabelPointAlongPathNV = PFNGLPOINTALONGPATHNVPROC glfwGetProcAddress("glabelPointAlongPathNV");
02083 glabelPointParameterf = PFNGLPOINTPARAMETERFPROC glfwGetProcAddress("glabelPointParameterf");
02084 glabelPointParameterfv = PFNGLPOINTPARAMETERFVPROC glfwGetProcAddress("glabelPointParameterfv");
02085 glabelPointParameteri = PFNGLPOINTPARAMETERIPROC glfwGetProcAddress("glabelPointParameteri");
02086 glabelPointParameteriv = PFNGLPOINTPARAMETERIVPROC glfwGetProcAddress("glabelPointParameteriv");
02087 glabelPointSize = PFNGLPOINTSIZEPROC glfwGetProcAddress("glabelPointSize");
02088 glabelPolygonMode = PFNGLPOLYGMODEPROC glfwGetProcAddress("glabelPolygonMode");
02089 glabelPolygonOffset = PFNGLPOLYGONOFFSETPROC glfwGetProcAddress("glabelPolygonOffset");
02090 glabelPolygonOffsetClampEXT =
PFNGLPOLYGONOFFSETCLAMPEXTPROC glfwGetProcAddress("glabelPolygonOffsetClampEXT");
02091 glabelPopDebugGroup = PFNGLPOPDEBUGGROUPUPPROC glfwGetProcAddress("glabelPopDebugGroup");
02092 glabelPopGroupMarkerEXT = PFNGLPOPGROUPMARKEREXTPROC glfwGetProcAddress("glabelPopGroupMarkerEXT");
02093 glabelPrimitiveBoundingBoxARB =
PFNGLPRIMITIVEBOUNDINGBOXARBPROC glfwGetProcAddress("glabelPrimitiveBoundingBoxARB");
02094 glabelPrimitiveRestartIndex =
PFNGLPRIMITIVERESTARTINDEXPROC glfwGetProcAddress("glabelPrimitiveRestartIndex");
02095 glabelProgramBinary = PFNGLPROGRAMBINARYPROC glfwGetProcAddress("glabelProgramBinary");
02096 glabelProgramParameteri = PFNGLPROGRAMPARAMETERIPROC glfwGetProcAddress("glabelProgramParameteri");
02097 glabelProgramParameteriARB =
PFNGLPROGRAMPARAMETERIARBPROC glfwGetProcAddress("glabelProgramParameteriARB");
02098 glabelProgramPathFragmentInputGenNV =
PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glfwGetProcAddress("glabelProgramPathFragmentInputGenNV");
02099 glabelProgramUniformId = PFNGLPROGRAMUNIFORM1DPROC glfwGetProcAddress("glabelProgramUniformId");
02100 glabelProgramUniformIdEXT = PFNGLPROGRAMUNIFORM1DEXTPROC glfwGetProcAddress("glabelProgramUniformIdEXT");
02101 glabelProgramUniformIdfv = PFNGLPROGRAMUNIFORM1DVPROC glfwGetProcAddress("glabelProgramUniformIdfv");
02102 glabelProgramUniformIdfvEXT =
PFNGLPROGRAMUNIFORM1DVEXTPROC glfwGetProcAddress("glabelProgramUniformIdfvEXT");
02103 glabelProgramUniformlf = PFNGLPROGRAMUNIFORM1FFPROC glfwGetProcAddress("glabelProgramUniformlf");
02104 glabelProgramUniformlfEXT = PFNGLPROGRAMUNIFORM1FEXTPROC glfwGetProcAddress("glabelProgramUniformlfEXT");
02105 glabelProgramUniformlfv = PFNGLPROGRAMUNIFORM1FVPROC glfwGetProcAddress("glabelProgramUniformlfv");
02106 glabelProgramUniformlfvEXT =
PFNGLPROGRAMUNIFORM1FVEXTPROC glfwGetProcAddress("glabelProgramUniformlfvEXT");
02107 glabelProgramUniformli = PFNGLPROGRAMUNIFORM1IPROC glfwGetProcAddress("glabelProgramUniformli");
02108 glabelProgramUniformli64ARB =
PFNGLPROGRAMUNIFORM1I64ARBPROC glfwGetProcAddress("glabelProgramUniformli64ARB");
02109 glabelProgramUniformli64NV =
PFNGLPROGRAMUNIFORM1I64NVPROC glfwGetProcAddress("glabelProgramUniformli64NV");
02110 glabelProgramUniformli64vARB =
PFNGLPROGRAMUNIFORM1I64VARBPROC glfwGetProcAddress("glabelProgramUniformli64vARB");
02111 glabelProgramUniformli64vNV =
PFNGLPROGRAMUNIFORM1I64VNVPROC glfwGetProcAddress("glabelProgramUniformli64vNV");
02112 glabelProgramUniformliEXT = PFNGLPROGRAMUNIFORM1IEXTPROC glfwGetProcAddress("glabelProgramUniformliEXT");
02113 glabelProgramUniformliv = PFNGLPROGRAMUNIFORM1IVPROC glfwGetProcAddress("glabelProgramUniformliv");
02114 glabelProgramUniformlivEXT =
PFNGLPROGRAMUNIFORM1IVEXTPROC glfwGetProcAddress("glabelProgramUniformlivEXT");
02115 glabelProgramUniformlui = PFNGLPROGRAMUNIFORM1UIPROC glfwGetProcAddress("glabelProgramUniformlui");
02116 glabelProgramUniformlui64ARB =

```

```

02117 PFNGLPROGRAMUNIFORM1UI64ARBPROC(glfwGetProcAddress("glProgramUniformui64ARB"));
02118     glProgramUniformui64NV =
02119 PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
02120     glProgramUniformui64vARB =
02121 PFNGLPROGRAMUNIFORM1UI64VARBPROC(glfwGetProcAddress("glProgramUniformui64vARB"));
02122     glProgramUniformui64vNV =
02123 PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniformui64vNV"));
02124     glProgramUniformuiEXT =
02125 PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniformuiEXT"));
02126     glProgramUniformluiv = PFNGLPROGRAMUNIFORM1UIVPROC(glfwGetProcAddress("glProgramUniformluiv"));
02127     glProgramUniformluivEXT =
02128 PFNGLPROGRAMUNIFORM1UIVEXTPROC(glfwGetProcAddress("glProgramUniformluivEXT"));
02129     glProgramUniform2d = PFNGLPROGRAMUNIFORM2DPROC(glfwGetProcAddress("glProgramUniform2d"));
02130     glProgramUniform2dEXT = PFNGLPROGRAMUNIFORM2DEXTPROC(glfwGetProcAddress("glProgramUniform2dEXT"));
02131     glProgramUniform2dV = PFNGLPROGRAMUNIFORM2DVPROC(glfwGetProcAddress("glProgramUniform2dV"));
02132     glProgramUniform2dVEXT =
02133 PFNGLPROGRAMUNIFORM2DVEXTPROC(glfwGetProcAddress("glProgramUniform2dVEXT"));
02134     glProgramUniform2f = PFNGLPROGRAMUNIFORM2FFPROC(glfwGetProcAddress("glProgramUniform2f"));
02135     glProgramUniform2fEXT = PFNGLPROGRAMUNIFORM2FEXTPROC(glfwGetProcAddress("glProgramUniform2fEXT"));
02136     glProgramUniform2fv = PFNGLPROGRAMUNIFORM2FVPROC(glfwGetProcAddress("glProgramUniform2fv"));
02137     glProgramUniform2fvEXT =
02138 PFNGLPROGRAMUNIFORM2FVEXTPROC(glfwGetProcAddress("glProgramUniform2fvEXT"));
02139     glProgramUniform2i = PFNGLPROGRAMUNIFORM2IPROC(glfwGetProcAddress("glProgramUniform2i"));
02140     glProgramUniform2i64ARB =
02141 PFNGLPROGRAMUNIFORM2I64ARBPROC(glfwGetProcAddress("glProgramUniform2i64ARB"));
02142     glProgramUniform2i64NV =
02143 PFNGLPROGRAMUNIFORM2I64NVPROC(glfwGetProcAddress("glProgramUniform2i64NV"));
02144     glProgramUniform2i64VARB =
02145 PFNGLPROGRAMUNIFORM2I64VARBPROC(glfwGetProcAddress("glProgramUniform2i64VARB"));
02146     glProgramUniform2i64vNV =
02147 PFNGLPROGRAMUNIFORM2I64VNVPROC(glfwGetProcAddress("glProgramUniform2i64vNV"));
02148     glProgramUniform2iEXT =
02149 PFNGLPROGRAMUNIFORM2IEXTPROC(glfwGetProcAddress("glProgramUniform2iEXT"));
02150     glProgramUniform2iv = PFNGLPROGRAMUNIFORM2IVPROC(glfwGetProcAddress("glProgramUniform2iv"));
02151     glProgramUniform2ui =
02152 PFNGLPROGRAMUNIFORM2UI64ARBPROC(glfwGetProcAddress("glProgramUniform2ui64ARB"));
02153     glProgramUniform2ui64NV =
02154 PFNGLPROGRAMUNIFORM2UI64NVPROC(glfwGetProcAddress("glProgramUniform2ui64NV"));
02155     glProgramUniform2uiEXT =
02156 PFNGLPROGRAMUNIFORM2UIEXTPROC(glfwGetProcAddress("glProgramUniform2uiEXT"));
02157     glProgramUniform2uiv = PFNGLPROGRAMUNIFORM2UIVPROC(glfwGetProcAddress("glProgramUniform2uiv"));
02158     glProgramUniform2uivEXT =
02159 PFNGLPROGRAMUNIFORM2UIVEXTPROC(glfwGetProcAddress("glProgramUniform2uivEXT"));
02160     glProgramUniform3d = PFNGLPROGRAMUNIFORM3DPROC(glfwGetProcAddress("glProgramUniform3d"));
02161     glProgramUniform3dEXT = PFNGLPROGRAMUNIFORM3DEXTPROC(glfwGetProcAddress("glProgramUniform3dEXT"));
02162     glProgramUniform3dv = PFNGLPROGRAMUNIFORM3DVPROC(glfwGetProcAddress("glProgramUniform3dv"));
02163     glProgramUniform3dVEXT =
02164 PFNGLPROGRAMUNIFORM3DVEXTPROC(glfwGetProcAddress("glProgramUniform3dVEXT"));
02165     glProgramUniform3f = PFNGLPROGRAMUNIFORM3FFPROC(glfwGetProcAddress("glProgramUniform3f"));
02166     glProgramUniform3fEXT = PFNGLPROGRAMUNIFORM3FEXTPROC(glfwGetProcAddress("glProgramUniform3fEXT"));
02167     glProgramUniform3fv = PFNGLPROGRAMUNIFORM3FVPROC(glfwGetProcAddress("glProgramUniform3fv"));
02168     glProgramUniform3fvEXT =
02169 PFNGLPROGRAMUNIFORM3FVEXTPROC(glfwGetProcAddress("glProgramUniform3fvEXT"));
02170     glProgramUniform3i =
02171 PFNGLPROGRAMUNIFORM3IPROC(glfwGetProcAddress("glProgramUniform3i"));
02172     glProgramUniform3i64ARB =
02173 PFNGLPROGRAMUNIFORM3I64ARBPROC(glfwGetProcAddress("glProgramUniform3i64ARB"));
02174     glProgramUniform3i64NV =
02175 PFNGLPROGRAMUNIFORM3I64NVPROC(glfwGetProcAddress("glProgramUniform3i64NV"));
02176     glProgramUniform3i64VARB =
02177 PFNGLPROGRAMUNIFORM3I64VARBPROC(glfwGetProcAddress("glProgramUniform3i64VARB"));
02178     glProgramUniform3i64vNV =
02179 PFNGLPROGRAMUNIFORM3I64VNVPROC(glfwGetProcAddress("glProgramUniform3i64vNV"));
02180     glProgramUniform3iEXT =
02181 PFNGLPROGRAMUNIFORM3IEXTPROC(glfwGetProcAddress("glProgramUniform3iEXT"));
02182     glProgramUniform3iv = PFNGLPROGRAMUNIFORM3IVPROC(glfwGetProcAddress("glProgramUniform3iv"));
02183     glProgramUniform3ivEXT =
02184 PFNGLPROGRAMUNIFORM3IVEXTPROC(glfwGetProcAddress("glProgramUniform3ivEXT"));
02185     glProgramUniform3ui =
02186 PFNGLPROGRAMUNIFORM3UIPROC(glfwGetProcAddress("glProgramUniform3ui"));
02187     glProgramUniform3ui64ARB =
02188 PFNGLPROGRAMUNIFORM3UI64ARBPROC(glfwGetProcAddress("glProgramUniform3ui64ARB"));
02189     glProgramUniform3ui64NV =
02190 PFNGLPROGRAMUNIFORM3UI64NVPROC(glfwGetProcAddress("glProgramUniform3ui64NV"));
02191     glProgramUniform3ui64VARB =
02192 PFNGLPROGRAMUNIFORM3UI64VARBPROC(glfwGetProcAddress("glProgramUniform3ui64VARB"));
02193     glProgramUniform3ui64vNV =
02194 PFNGLPROGRAMUNIFORM3UI64VNVPROC(glfwGetProcAddress("glProgramUniform3ui64vNV"));
02195     glProgramUniform3uiEXT =
02196 PFNGLPROGRAMUNIFORM3UIEXTPROC(glfwGetProcAddress("glProgramUniform3uiEXT"));
02197     glProgramUniform3uiv = PFNGLPROGRAMUNIFORM3UIVPROC(glfwGetProcAddress("glProgramUniform3uiv"));
02198     glProgramUniform3uivEXT =
02199 PFNGLPROGRAMUNIFORM3UIVEXTPROC(glfwGetProcAddress("glProgramUniform3uivEXT"));
02200     glProgramUniform4d = PFNGLPROGRAMUNIFORM4DPROC(glfwGetProcAddress("glProgramUniform4d"));

```

```
02172     glProgramUniform4dEXT = PFNGLPROGRAMUNIFORM4DEXTPROC(glfwGetProcAddress("glProgramUniform4dEXT"));
02173     glProgramUniform4dv = PFNGLPROGRAMUNIFORM4DVPROC(glfwGetProcAddress("glProgramUniform4dv"));
02174     glProgramUniform4dvEXT =
02175         PFNGLPROGRAMUNIFORM4DVEXTPROC(glfwGetProcAddress("glProgramUniform4dvEXT"));
02176     glProgramUniform4f = PFNGLPROGRAMUNIFORM4FPROC(glfwGetProcAddress("glProgramUniform4f"));
02177     glProgramUniform4fEXT = PFNGLPROGRAMUNIFORM4FEXTPROC(glfwGetProcAddress("glProgramUniform4fEXT"));
02178     glProgramUniform4fv = PFNGLPROGRAMUNIFORM4FVPROC(glfwGetProcAddress("glProgramUniform4fv"));
02179     glProgramUniform4fvEXT =
02180         PFNGLPROGRAMUNIFORM4FVEXTPROC(glfwGetProcAddress("glProgramUniform4fvEXT"));
02181     glProgramUniform4i = PFNGLPROGRAMUNIFORM4IPROC(glfwGetProcAddress("glProgramUniform4i"));
02182     glProgramUniform4i64ARB =
02183         PFNGLPROGRAMUNIFORM4I64ARBPROC(glfwGetProcAddress("glProgramUniform4i64ARB"));
02184     glProgramUniform4i64NV =
02185         PFNGLPROGRAMUNIFORM4I64NVPROC(glfwGetProcAddress("glProgramUniform4i64NV"));
02186     glProgramUniform4i64vARB =
02187         PFNGLPROGRAMUNIFORM4I64VARBPROC(glfwGetProcAddress("glProgramUniform4i64vARB"));
02188     glProgramUniform4i64vNV =
02189         PFNGLPROGRAMUNIFORM4I64VNVPROC(glfwGetProcAddress("glProgramUniform4i64vNV"));
02190     glProgramUniform4iEXT = PFNGLPROGRAMUNIFORM4IEXTPROC(glfwGetProcAddress("glProgramUniform4iEXT"));
02191     glProgramUniform4iv = PFNGLPROGRAMUNIFORM4IVPROC(glfwGetProcAddress("glProgramUniform4iv"));
02192     glProgramUniform4ivEXT =
02193         PFNGLPROGRAMUNIFORM4IVEXTPROC(glfwGetProcAddress("glProgramUniform4ivEXT"));
02194     glProgramUniform4ui =
02195         PFNGLPROGRAMUNIFORM4UIPROC(glfwGetProcAddress("glProgramUniform4ui"));
02196     glProgramUniform4ui64ARB =
02197         PFNGLPROGRAMUNIFORM4UI64ARBPROC(glfwGetProcAddress("glProgramUniform4ui64ARB"));
02198     glProgramUniform4ui64NV =
02199         PFNGLPROGRAMUNIFORM4UI64NVPROC(glfwGetProcAddress("glProgramUniform4ui64NV"));
02200     glProgramUniform4ui64vARB =
02201         PFNGLPROGRAMUNIFORM4UI64VARBPROC(glfwGetProcAddress("glProgramUniform4ui64vARB"));
02202     glProgramUniform4ui64vNV =
02203         PFNGLPROGRAMUNIFORM4UI64VNVPROC(glfwGetProcAddress("glProgramUniform4ui64vNV"));
02204     glProgramUniform4uiEXT =
02205         PFNGLPROGRAMUNIFORM4UIEXTPROC(glfwGetProcAddress("glProgramUniform4uiEXT"));
02206     glProgramUniform4uiv =
02207         PFNGLPROGRAMUNIFORM4UIVPROC(glfwGetProcAddress("glProgramUniform4uiv"));
02208     glProgramUniform4uivEXT =
02209         PFNGLPROGRAMUNIFORM4UIVEXTPROC(glfwGetProcAddress("glProgramUniform4uivEXT"));
02210     glProgramUniformHandleui64ARB =
02211         PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glProgramUniformHandleui64ARB"));
02212     glProgramUniformHandleui64NV =
02213         PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glProgramUniformHandleui64NV"));
02214     glProgramUniformHandleui64vARB =
02215         PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glProgramUniformHandleui64vARB"));
02216     glProgramUniformHandleui64vNV =
02217         PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glProgramUniformHandleui64vNV"));
02218     glProgramUniformMatrix2dv =
02219         PFNGLPROGRAMUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glProgramUniformMatrix2dv"));
02220     glProgramUniformMatrix2dEXT =
02221         PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2dEXT"));
02222     glProgramUniformMatrix2fv =
02223         PFNGLPROGRAMUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glProgramUniformMatrix2fv"));
02224     glProgramUniformMatrix2fvEXT =
02225         PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2fvEXT"));
02226     glProgramUniformMatrix2x3dv =
02227         PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dv"));
02228     glProgramUniformMatrix2x3dvEXT =
02229         PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dvEXT"));
02230     glProgramUniformMatrix2x3fv =
02231         PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fv"));
02232     glProgramUniformMatrix2x3fvEXT =
02233         PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fvEXT"));
02234     glProgramUniformMatrix2x4dv =
02235         PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dv"));
02236     glProgramUniformMatrix2x4dEXT =
02237         PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dEXT"));
02238     glProgramUniformMatrix2x4fv =
02239         PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fv"));
02240     glProgramUniformMatrix2x4fvEXT =
02241         PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fvEXT"));
02242     glProgramUniformMatrix3dv =
02243         PFNGLPROGRAMUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glProgramUniformMatrix3dv"));
02244     glProgramUniformMatrix3dEXT =
02245         PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3dEXT"));
02246     glProgramUniformMatrix3fv =
02247         PFNGLPROGRAMUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glProgramUniformMatrix3fv"));
02248     glProgramUniformMatrix3fvEXT =
02249         PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3fvEXT"));
02250     glProgramUniformMatrix3x2dv =
02251         PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dv"));
02252     glProgramUniformMatrix3x2dEXT =
02253         PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dEXT"));
02254     glProgramUniformMatrix3x2fv =
02255         PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fv"));
02256     glProgramUniformMatrix3x2fvEXT =
02257         PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fvEXT"));
02258     glProgramUniformMatrix3x4dv =
02259         PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dv"));
02260     glProgramUniformMatrix3x4dEXT =
```

```

0221 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dvEXT"));
0222     glProgramUniformMatrix3x4fv =
0223 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fv"));
0224     glProgramUniformMatrix3x4fvEXT =
0225 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fvEXT"));
0226     glProgramUniformMatrix4d =
0227 PFNGLPROGRAMUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glProgramUniformMatrix4dv"));
0228     glProgramUniformMatrix4dEXT =
0229 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4dvEXT"));
0230     glProgramUniformMatrix4fv =
0231 PFNGLPROGRAMUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));
0232     glProgramUniformMatrix4fvEXT =
0233 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fvEXT"));
0234     glProgramUniformMatrix4x2d =
0235 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dv"));
0236     glProgramUniformMatrix4x2dEXT =
0237 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dvEXT"));
0238     glProgramUniformMatrix4x2fv =
0239 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fv"));
0240     glProgramUniformMatrix4x2fvEXT =
0241 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fvEXT"));
0242     glProgramUniformMatrix4x3d =
0243 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dv"));
0244     glProgramUniformMatrix4x3dEXT =
0245 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dvEXT"));
0246     glProgramUniformMatrix4x3fv =
0247 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fv"));
0248     glProgramUniformMatrix4x3fvEXT =
0249 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fvEXT"));
0250     glProgramUniformui64NV =
0251 PFNGLPROGRAMUNIFORMUI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
0252     glProgramUniformui64vNV =
0253 PFNGLPROGRAMUNIFORMUI64VNVPROC(glfwGetProcAddress("glProgramUniformui64vNV"));
0254     glProvokingVertex = PFNGLPROVOKINGVERTEXPROC(glfwGetProcAddress("glProvokingVertex"));
0255     glPushClientAttribDefaultEXT =
0256 PFNGLPUSHCLIENTATTRIBDEFAULTTEXTPROC(glfwGetProcAddress("glPushClientAttribDefaultEXT"));
0257     glPushDebugGroup = PFNGLPUSHDEBUGGROUPPROC(glfwGetProcAddress("glPushDebugGroup"));
0258     glPushGroupMarkerEXT = PFNGLPUSHGROUPMARKEREXTPROC(glfwGetProcAddress("glPushGroupMarkerEXT"));
0259     glQueryCounter = PFNGLQUERYCOUNTERPROC(glfwGetProcAddress("glQueryCounter"));
0260     glRasterSamplesEXT = PFNGLRASTERSAMPLESEXTPROC(glfwGetProcAddress("glRasterSamplesEXT"));
0261     glReadBuffer = PFNGLREADBUFFERPROC(glfwGetProcAddress("glReadBuffer"));
0262     glReadPixels = PFNGLREADPIXELSPROC(glfwGetProcAddress("glReadPixels"));
0263     glReadnPixels = PFNGLREADNPPIXELSPROC(glfwGetProcAddress("glReadnPixels"));
0264     glReadnPixelsARB = PFNGLREADNPPIXELSARBPROC(glfwGetProcAddress("glReadnPixelsARB"));
0265     glReleaseShaderCompiler =
0266 PFNGLRELEASESHADERCOMPILERPROC(glfwGetProcAddress("glReleaseShaderCompiler"));
0267     glRenderbufferStorage = PFNGLRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glRenderbufferStorage"));
0268     glRenderbufferStorageMultisample =
0269 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glRenderbufferStorageMultisample"));
0270     glRenderbufferStorageMultisampleCoverageNV =
0271 PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC(glfwGetProcAddress("glRenderbufferStorageMultisampleCoverageNV"));
0272     glResolveDepthValuesNV =
0273 PFNGLRESOLVEDEPTHVALUESNVPROC(glfwGetProcAddress("glResolveDepthValuesNV"));
0274     glResumeTransformFeedback =
0275 PFNGLRESUMETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glResumeTransformFeedback"));
0276     glSampleCoverage = PFNGLSAMPLECOVERAGEPROC(glfwGetProcAddress("glSampleCoverage"));
0277     glSampleMaski = PFNGLSAMPLEMASKIPROC(glfwGetProcAddress("glSampleMaski"));
0278     glSamplerParameterIiv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameterIiv"));
0279     glSamplerParameterIuiv =
0280 PFNGLSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glSamplerParameterIuiv"));
0281     glSamplerParameterf = PFNGLSAMPLERPARAMETERFPROC(glfwGetProcAddress("glSamplerParameterf"));
0282     glSamplerParameterfv = PFNGLSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glSamplerParameterfv"));
0283     glSamplerParameteri = PFNGLSAMPLERPARAMETERIIPROC(glfwGetProcAddress("glSamplerParameteri"));
0284     glSamplerParameteriv = PFNGLSAMPLERPARAMETERIRVPROC(glfwGetProcAddress("glSamplerParameteriv"));
0285     glScissor = PFNGLSCISSORPROC(glfwGetProcAddress("glScissor"));
0286     glScissorArrayv = PFNGLSCISSORARRAYVPROC(glfwGetProcAddress("glScissorArrayv"));
0287     glScissorIndexed = PFNGLSCISSORINDEXEDPROC(glfwGetProcAddress("glScissorIndexed"));
0288     glScissorIndexedv = PFNGLSCISSORINDEXEDVPROC(glfwGetProcAddress("glScissorIndexedv"));
0289     glSecondaryColorFormatNV =
0290 PFNGLSECONDARYCOLORFORMATNVPROC(glfwGetProcAddress("glSecondaryColorFormatNV"));
0291     glSelectPerfMonitorCountersAMD =
0292 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glSelectPerfMonitorCountersAMD"));
0293     glShaderBinary = PFNGLSHADERBINARYPROC(glfwGetProcAddress("glShaderBinary"));
0294     glShaderSource = PFNGLSHADERSOURCEPROC(glfwGetProcAddress("glShaderSource"));
0295     glShaderStorageBlockBinding =
0296 PFNGLSHADERSTORAGEBLOCKBINDINGPROC(glfwGetProcAddress("glShaderStorageBlockBinding"));
0297     glSignalVkFenceNV = PFNGLSIGNALKVFENCENVPROC(glfwGetProcAddress("glSignalVkFenceNV"));
0298     glSignalVkSemaphoreNV = PFNGLSIGNALKSEMAPHORENVPROC(glfwGetProcAddress("glSignalVkSemaphoreNV"));
0299     glSpecializeShaderARB = PFNGLSPECIALIZE_SHADERARBPROC(glfwGetProcAddress("glSpecializeShaderARB"));
0300     glStateCaptureNV = PFNGLSTATECAPTURENVPROC(glfwGetProcAddress("glStateCaptureNV"));
0301     glStencilFillPathInstancedNV =
0302 PFNGLSTENCILFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilFillPathInstancedNV"));
0303     glStencilFillPathNV = PFNGLSTENCILFILLPATHNVPROC(glfwGetProcAddress("glStencilFillPathNV"));
0304     glStencilFunc = PFNGLSTENCILFUNCPROC(glfwGetProcAddress("glStencilFunc"));
0305     glStencilFuncSeparate = PFNGLSTENCILFUNCSEPARATEPROC(glfwGetProcAddress("glStencilFuncSeparate"));
0306     glStencilMask = PFNGLSTENCILMASKPROC(glfwGetProcAddress("glStencilMask"));
0307     glStencilMaskSeparate = PFNGLSTENCILMASKSEPARATEPROC(glfwGetProcAddress("glStencilMaskSeparate"));

```

```
02280     glStencilOp = PFNGLSTENCILOPPROC(glfwGetProcAddress("glStencilOp"));
02281     glStencilOpSeparate = PFNGLSTENCILOPSEPARATEPROC(glfwGetProcAddress("glStencilOpSeparate"));
02282     glStencilStrokePathInstancedNV =
02283         PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilStrokePathInstancedNV"));
02284     glStencilStrokePathNV = PFNGLSTENCILSTROKEPATHNVPROC(glfwGetProcAddress("glStencilStrokePathNV"));
02285     glStencilThenCoverFillPathInstancedNV =
02286         PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathInstancedNV"));
02287     glStencilThenCoverFillPathNV =
02288         PFNGLSTENCILTHENCOVERFILLPATHNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathNV"));
02289     glStencilThenCoverStrokePathInstancedNV =
02290         PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathInstancedNV"));
02291     glStencilThenCoverStrokePathNV =
02292         PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathNV"));
02293     glSubpixelPrecisionBiasNV =
02294         PFNGLSUBPIXELPRECISIONBIASNVPROC(glfwGetProcAddress("glSubpixelPrecisionBiasNV"));
02295     glTexBuffer = PFNGLTEXBUFFERPROC(glfwGetProcAddress("glTexBuffer"));
02296     glTexBufferARB = PFNGLTEXBUFFERARBPROC(glfwGetProcAddress("glTexBufferARB"));
02297     glTexBufferSize = PFNGLTEXBUFFERRANGEPROC(glfwGetProcAddress("glTexBufferSize"));
02298     glTexCoordFormatNV = PFNGLTEXCOORDFORMATNVPROC(glfwGetProcAddress("glTexCoordFormatNV"));
02299     glTexImage1D = PFNGLTEXIMAGE1DPROC(glfwGetProcAddress("glTexImage1D"));
02300     glTexImage2D = PFNGLTEXIMAGE2DPROC(glfwGetProcAddress("glTexImage2D"));
02301     glTexImage2DMultisample =
02302         PFNGLTEXIMAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage2DMultisample"));
02303     glTexImage3D = PFNGLTEXIMAGE3DPROC(glfwGetProcAddress("glTexImage3D"));
02304     glTexImage3DMultisample =
02305         PFNGLTEXIMAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage3DMultisample"));
02306     glTexParameterCommitmentARB =
02307         PFNGLTEXPAGECOMMITMENTARBPROC(glfwGetProcAddress("glTexParameterCommitmentARB"));
02308     glTexParameterIiv = PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameterIiv"));
02309     glTexParameterIuiv = PFNGLTEXPARAMETERUIVPROC(glfwGetProcAddress("glTexParameterIuiv"));
02310     glTexParameterIfv = PFNGLTEXPARAMETERFPROC(glfwGetProcAddress("glTexParameterf"));
02311     glTexParameterfv = PFNGLTEXPARAMETERFVPROC(glfwGetProcAddress("glTexParameterfv"));
02312     glTexParameteriv = PFNGLTEXPARAMETERIPROC(glfwGetProcAddress("glTexParameteriv"));
02313     glTexParameterStorage1D = PFNGLTEXSTORAGE1DPROC(glfwGetProcAddress("glTexParameterStorage1D"));
02314     glTexParameterStorage2D = PFNGLTEXSTORAGE2DPROC(glfwGetProcAddress("glTexParameterStorage2D"));
02315     glTexParameterStorage3D = PFNGLTEXSTORAGE3DPROC(glfwGetProcAddress("glTexParameterStorage3D"));
02316     glTexParameterStorage2DMultisample =
02317         PFNGLTEXSTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexParameterStorage2DMultisample"));
02318     glTexParameterStorage3DMultisample =
02319         PFNGLTEXSTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexParameterStorage3DMultisample"));
02320     glTexParameterSubImage1D = PFNGLTEXSUBIMAGE1DPROC(glfwGetProcAddress("glTexParameterSubImage1D"));
02321     glTexParameterSubImage2D = PFNGLTEXSUBIMAGE2DPROC(glfwGetProcAddress("glTexParameterSubImage2D"));
02322     glTexParameterSubImage3D = PFNGLTEXSUBIMAGE3DPROC(glfwGetProcAddress("glTexParameterSubImage3D"));
02323     glTextureBarrier = PFNGLTEXTUREBARRIERPROC(glfwGetProcAddress("glTextureBarrier"));
02324     glTextureBarrierNV = PFNGLTEXTUREBARRIERNVPROC(glfwGetProcAddress("glTextureBarrierNV"));
02325     glTextureBuffer = PFNGLTEXTUREBUFFERPROC(glfwGetProcAddress("glTextureBuffer"));
02326     glTextureBufferEXT = PFNGLTEXTUREBUFFEREXTPROC(glfwGetProcAddress("glTextureBufferEXT"));
02327     glTextureBufferRange = PFNGLTEXTUREBUFFERRANGEPROC(glfwGetProcAddress("glTextureBufferRange"));
02328     glTextureBufferRangeEXT =
02329         PFNGLTEXTUREBUFFERRANGEEXTPROC(glfwGetProcAddress("glTextureBufferRangeEXT"));
02330     glTextureImage1DEXT = PFNGLTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glTextureImage1DEXT"));
02331     glTextureImage2DEXT = PFNGLTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glTextureImage2DEXT"));
02332     glTextureImage3DEXT = PFNGLTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glTextureImage3DEXT"));
02333     glTexturePageCommitmentEXT =
02334         PFNGLTEXTUREPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glTexturePageCommitmentEXT"));
02335     glTextureParameterIiv = PFNGLTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glTextureParameterIiv"));
02336     glTextureParameterIivEXT =
02337         PFNGLTEXTUREPARAMETERUIVPROC(glfwGetProcAddress("glTextureParameterIuiv"));
02338     glTextureParameterIfv = PFNGLTEXTUREPARAMETERFPROC(glfwGetProcAddress("glTextureParameterf"));
02339     glTextureParameterfvEXT =
02340         PFNGLTEXTUREPARAMETERFEXTPROC(glfwGetProcAddress("glTextureParameterfEXT"));
02341     glTextureParameterfv = PFNGLTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glTextureParameterfv"));
02342     glTextureParameterfvEXT =
02343         PFNGLTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glTextureParameterfvEXT"));
02344     glTextureParameteri = PFNGLTEXTUREPARAMETERIPROC(glfwGetProcAddress("glTextureParameteri"));
02345     glTextureParameteriEXT =
02346         PFNGLTEXTUREPARAMETERIEXTPROC(glfwGetProcAddress("glTextureParameteriEXT"));
02347     glTextureParameteriv = PFNGLTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glTextureParameteriv"));
02348     glTextureParameterivEXT =
02349         PFNGLTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glTextureParameterivEXT"));
02350     glTextureRenderbufferEXT =
02351         PFNGLTEXTURERENDERBUFFEREXTPROC(glfwGetProcAddress("glTextureRenderbufferEXT"));
02352     glTextureStorage1D = PFNGLTEXTURESTORAGE1DPROC(glfwGetProcAddress("glTextureStorage1D"));
02353     glTextureStorage1DEXT = PFNGLTEXTURESTORAGE1DEXTPROC(glfwGetProcAddress("glTextureStorage1DEXT"));
02354     glTextureStorage2D = PFNGLTEXTURESTORAGE2DPROC(glfwGetProcAddress("glTextureStorage2D"));
02355     glTextureStorage2DEXT = PFNGLTEXTURESTORAGE2DEXTPROC(glfwGetProcAddress("glTextureStorage2DEXT"));
02356     glTextureStorage2DMultisample =
02357         PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage2DMultisample"));
02358     glTextureStorage2DMultisampleEXT =
02359         PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage2DMultisampleEXT"));
02360     glTextureStorage3D = PFNGLTEXTURESTORAGE3DPROC(glfwGetProcAddress("glTextureStorage3D"));
02361     glTextureStorage3DEXT = PFNGLTEXTURESTORAGE3DEXTPROC(glfwGetProcAddress("glTextureStorage3DEXT"));
```

```

02344     glTextureStorage3DMultisample =
02345     PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage3DMultisample"));
02346     glTextureStorage3DMultisampleEXT =
02347     PFNGLTEXTURERESTORAGE3DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage3DMultisampleEXT"));
02348     glTextureSubImage1D = PFNGLTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glTextureSubImage1D"));
02349     glTextureSubImage1DEXT =
02350     PFNGLTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glTextureSubImage1DEXT"));
02351     glTextureSubImage2D = PFNGLTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glTextureSubImage2D"));
02352     glTextureSubImage2DEXT =
02353     PFNGLTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glTextureSubImage2DEXT"));
02354     glTextureSubImage3D = PFNGLTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glTextureSubImage3D"));
02355     glTextureSubImage3DEXT =
02356     PFNGLTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glTextureSubImage3DEXT"));
02357     glTextureView = PFNGLTEXTUREVIEWPROC(glfwGetProcAddress("glTextureView"));
02358     glTransformFeedbackBufferBase =
02359     PFNGLTRANSFORMFEEDBACKBASEPROC(glfwGetProcAddress("glTransformFeedbackBufferBase"));
02360     glTransformFeedbackBufferRange =
02361     PFNGLTRANSFORMFEEDBACKRANGEPROC(glfwGetProcAddress("glTransformFeedbackBufferRange"));
02362     glTransformFeedbackVaryings =
02363     PFNGLTRANSFORMFEEDBACKVARYINGSPROC(glfwGetProcAddress("glTransformFeedbackVaryings"));
02364     glTransformPathNV = PFNGLTRANSFORMPATHNVPROC(glfwGetProcAddress("glTransformPathNV"));
02365     glUniform1d = PFNGLUNIFORM1DPROC(glfwGetProcAddress("glUniform1d"));
02366     glUniform1dv = PFNGLUNIFORM1DVPROC(glfwGetProcAddress("glUniform1dv"));
02367     glUniform1f = PFNGLUNIFORM1FPROC(glfwGetProcAddress("glUniform1f"));
02368     glUniform1fv = PFNGLUNIFORM1FVPROC(glfwGetProcAddress("glUniform1fv"));
02369     glUniform1i = PFNGLUNIFORM1IPROC(glfwGetProcAddress("glUniform1i"));
02370     glUniform1i64ARB = PFNGLUNIFORM1I64ARBPROC(glfwGetProcAddress("glUniform1i64ARB"));
02371     glUniform1i64NV = PFNGLUNIFORM1I64NVPROC(glfwGetProcAddress("glUniform1i64NV"));
02372     glUniform1i64vARB = PFNGLUNIFORM1I64VARBPROC(glfwGetProcAddress("glUniform1i64vARB"));
02373     glUniform1i64vNV = PFNGLUNIFORM1I64VNVPROC(glfwGetProcAddress("glUniform1i64vNV"));
02374     glUniform1liv = PFNGLUNIFORM1IVPROC(glfwGetProcAddress("glUniform1liv"));
02375     glUniform1lui = PFNGLUNIFORM1UIPROC(glfwGetProcAddress("glUniform1lui"));
02376     glUniform1lui64ARB = PFNGLUNIFORM1UI64ARBPROC(glfwGetProcAddress("glUniform1lui64ARB"));
02377     glUniform1lui64NV = PFNGLUNIFORM1UI64NVPROC(glfwGetProcAddress("glUniform1lui64NV"));
02378     glUniform1lui64vARB = PFNGLUNIFORM1UI64VARBPROC(glfwGetProcAddress("glUniform1lui64vARB"));
02379     glUniform1lui64vNV = PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniform1lui64vNV"));
02380     glUniform2d = PFNGLUNIFORM2DPROC(glfwGetProcAddress("glUniform2d"));
02381     glUniform2dv = PFNGLUNIFORM2DVPROC(glfwGetProcAddress("glUniform2dv"));
02382     glUniform2f = PFNGLUNIFORM2FPROC(glfwGetProcAddress("glUniform2f"));
02383     glUniform2fv = PFNGLUNIFORM2FVPROC(glfwGetProcAddress("glUniform2fv"));
02384     glUniform2i = PFNGLUNIFORM2IPROC(glfwGetProcAddress("glUniform2i"));
02385     glUniform2i64ARB = PFNGLUNIFORM2I64ARBPROC(glfwGetProcAddress("glUniform2i64ARB"));
02386     glUniform2i64NV = PFNGLUNIFORM2I64NVPROC(glfwGetProcAddress("glUniform2i64NV"));
02387     glUniform2i64vARB = PFNGLUNIFORM2I64VARBPROC(glfwGetProcAddress("glUniform2i64vARB"));
02388     glUniform2i64vNV = PFNGLUNIFORM2I64VNVPROC(glfwGetProcAddress("glUniform2i64vNV"));
02389     glUniform2iv = PFNGLUNIFORM2IVPROC(glfwGetProcAddress("glUniform2iv"));
02390     glUniform2ui = PFNGLUNIFORM2UIPROC(glfwGetProcAddress("glUniform2ui"));
02391     glUniform2ui64ARB = PFNGLUNIFORM2UI64ARBPROC(glfwGetProcAddress("glUniform2ui64ARB"));
02392     glUniform2ui64NV = PFNGLUNIFORM2UI64NVPROC(glfwGetProcAddress("glUniform2ui64NV"));
02393     glUniform2ui64vARB = PFNGLUNIFORM2UI64VARBPROC(glfwGetProcAddress("glUniform2ui64vARB"));
02394     glUniform2ui64vNV = PFNGLUNIFORM2UI64VNVPROC(glfwGetProcAddress("glUniform2ui64vNV"));
02395     glUniform3d = PFNGLUNIFORM3DPROC(glfwGetProcAddress("glUniform3d"));
02396     glUniform3dv = PFNGLUNIFORM3DVPROC(glfwGetProcAddress("glUniform3dv"));
02397     glUniform3f = PFNGLUNIFORM3FPROC(glfwGetProcAddress("glUniform3f"));
02398     glUniform3fv = PFNGLUNIFORM3FVPROC(glfwGetProcAddress("glUniform3fv"));
02399     glUniform3i = PFNGLUNIFORM3IPROC(glfwGetProcAddress("glUniform3i"));
02400     glUniform3i64ARB = PFNGLUNIFORM3I64ARBPROC(glfwGetProcAddress("glUniform3i64ARB"));
02401     glUniform3i64NV = PFNGLUNIFORM3I64NVPROC(glfwGetProcAddress("glUniform3i64NV"));
02402     glUniform3i64vARB = PFNGLUNIFORM3I64VARBPROC(glfwGetProcAddress("glUniform3i64vARB"));
02403     glUniform3i64vNV = PFNGLUNIFORM3I64VNVPROC(glfwGetProcAddress("glUniform3i64vNV"));
02404     glUniform3uiv = PFNGLUNIFORM3UIVPROC(glfwGetProcAddress("glUniform3uiv"));
02405     glUniform4d = PFNGLUNIFORM4DPROC(glfwGetProcAddress("glUniform4d"));
02406     glUniform4dv = PFNGLUNIFORM4DVPROC(glfwGetProcAddress("glUniform4dv"));
02407     glUniform4f = PFNGLUNIFORM4FPROC(glfwGetProcAddress("glUniform4f"));
02408     glUniform4fv = PFNGLUNIFORM4FVPROC(glfwGetProcAddress("glUniform4fv"));
02409     glUniform4i = PFNGLUNIFORM4IPROC(glfwGetProcAddress("glUniform4i"));
02410     glUniform4i64ARB = PFNGLUNIFORM4I64ARBPROC(glfwGetProcAddress("glUniform4i64ARB"));
02411     glUniform4i64NV = PFNGLUNIFORM4I64NVPROC(glfwGetProcAddress("glUniform4i64NV"));
02412     glUniform4i64vARB = PFNGLUNIFORM4I64VARBPROC(glfwGetProcAddress("glUniform4i64vARB"));
02413     glUniform4i64vNV = PFNGLUNIFORM4I64VNVPROC(glfwGetProcAddress("glUniform4i64vNV"));
02414     glUniform4iv = PFNGLUNIFORM4IVPROC(glfwGetProcAddress("glUniform4iv"));
02415     glUniform4ui = PFNGLUNIFORM4UIPROC(glfwGetProcAddress("glUniform4ui"));
02416     glUniform4ui64ARB = PFNGLUNIFORM4UI64ARBPROC(glfwGetProcAddress("glUniform4ui64ARB"));
02417     glUniform4ui64NV = PFNGLUNIFORM4UI64NVPROC(glfwGetProcAddress("glUniform4ui64NV"));
02418     glUniform4ui64vARB = PFNGLUNIFORM4UI64VARBPROC(glfwGetProcAddress("glUniform4ui64vARB"));
02419     glUniform4ui64vNV = PFNGLUNIFORM4UI64VNVPROC(glfwGetProcAddress("glUniform4ui64vNV"));
02420     glUniform4uiv = PFNGLUNIFORM4UIVPROC(glfwGetProcAddress("glUniform4uiv"));
02421     glUniformBlockBinding = PFNGLUNIFORMBLOCKBINDINGPROC(glfwGetProcAddress("glUniformBlockBinding"));
02422     glUniformHandleui64ARB =

```

```

0243     PFNGLUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glUniformHandleui64ARB"));
0244     glUniformHandleui64NV = PFNGLUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glUniformHandleui64NV"));
0245     glUniformHandleui64vARB =
0246     PFNGLUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glUniformHandleui64vNV"));
0247     glUniformMatrix2dv = PFNGLUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glUniformMatrix2dv"));
0248     glUniformMatrix2fv = PFNGLUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glUniformMatrix2fv"));
0249     glUniformMatrix2x3dv = PFNGLUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glUniformMatrix2x3dv"));
0250     glUniformMatrix2x3fv = PFNGLUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glUniformMatrix2x3fv"));
0251     glUniformMatrix2x4dv = PFNGLUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glUniformMatrix2x4dv"));
0252     glUniformMatrix2x4fv = PFNGLUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glUniformMatrix2x4fv"));
0253     glUniformMatrix3dv = PFNGLUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glUniformMatrix3dv"));
0254     glUniformMatrix3fv = PFNGLUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glUniformMatrix3fv"));
0255     glUniformMatrix3x2dv = PFNGLUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glUniformMatrix3x2dv"));
0256     glUniformMatrix3x2fv = PFNGLUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glUniformMatrix3x2fv"));
0257     glUniformMatrix3x4dv = PFNGLUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glUniformMatrix3x4dv"));
0258     glUniformMatrix3x4fv = PFNGLUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glUniformMatrix3x4fv"));
0259     glUniformMatrix4dv = PFNGLUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glUniformMatrix4dv"));
0260     glUniformMatrix4fv = PFNGLUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glUniformMatrix4fv"));
0261     glUniformMatrix4x2dv = PFNGLUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glUniformMatrix4x2dv"));
0262     glUniformMatrix4x2fv = PFNGLUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glUniformMatrix4x2fv"));
0263     glUniformMatrix4x3dv = PFNGLUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glUniformMatrix4x3dv"));
0264     glUniformMatrix4x3fv = PFNGLUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glUniformMatrix4x3fv"));
0265     glUniformSubroutinesuiv =
0266     PFNGLUNIFORMSUBROUTINESUIVPROC(glfwGetProcAddress("glUniformSubroutinesuiv"));
0267     glUniformui64NV = PFNGLUNIFORMUI64NVPROC(glfwGetProcAddress("glUniformui64NV"));
0268     glUnformui64vNV = PFNGLUNIFORMUI64VNVPROC(glfwGetProcAddress("glUniformui64vNV"));
0269     glUnmapBuffer = PFNGLUNMAPBUFFERPROC(glfwGetProcAddress("glUnmapBuffer"));
0270     glUnmapNamedBuffer = PFNGLUNMAPNAMEDBUFFERPROC(glfwGetProcAddress("glUnmapNamedBuffer"));
0271     glUnmapNamedBufferEXT = PFNGLUNMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glUnmapNamedBufferEXT"));
0272     glUseProgram = PFNGLUSEPROGRAMPROC(glfwGetProcAddress("glUseProgram"));
0273     glUseProgramStages = PFNGLUSEPROGRAMSTAGESPROC(glfwGetProcAddress("glUseProgramStages"));
0274     glUseShaderProgramEXT = PFNGLUSESADERPROGRAMEXTPROC(glfwGetProcAddress("glUseShaderProgramEXT"));
0275     glValidateProgram = PFNGLVALIDATEPROGRAMPROC(glfwGetProcAddress("glValidateProgram"));
0276     glValidateProgramPipeline =
0277     PFNGLVALIDATEPROGRAMPIPELINEPROC(glfwGetProcAddress("glValidateProgramPipeline"));
0278     glVertexArrayAttribBinding =
0279     PFNGLVERTEXARRAYATTRIBBINDINGPROC(glfwGetProcAddress("glVertexArrayAttribBinding"));
0280     glVertexArrayAttribFormat =
0281     PFNGLVERTEXARRAYATTRIBFORMATPROC(glfwGetProcAddress("glVertexArrayAttribFormat"));
0282     glVertexArrayAttribIFormat =
0283     PFNGLVERTEXARRAYATTRIBIFORMATPROC(glfwGetProcAddress("glVertexArrayAttribIFormat"));
0284     glVertexArrayAttribLFormat =
0285     PFNGLVERTEXARRAYATTRIBLFORMATPROC(glfwGetProcAddress("glVertexArrayAttribLFormat"));
0286     glVertexArrayBindVertexBufferEXT =
0287     PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC(glfwGetProcAddress("glVertexArrayBindVertexBufferEXT"));
0288     glVertexArrayBindingDivisor =
0289     PFNGLVERTEXARRAYBINDINGDIVISORPROC(glfwGetProcAddress("glVertexArrayBindingDivisor"));
0290     glVertexArrayColorOffsetEXT =
0291     PFNGLVERTEXARRAYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArrayColorOffsetEXT"));
0292     glVertexArrayEdgeFlagOffsetEXT =
0293     PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayEdgeFlagOffsetEXT"));
0294     glVertexArrayElementBuffer =
0295     PFNGLVERTEXARRAYELEMENTBUFFERPROC(glfwGetProcAddress("glVertexArrayElementBuffer"));
0296     glVertexArrayFogCoordOffsetEXT =
0297     PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayFogCoordOffsetEXT"));
0298     glVertexArrayIndexOffsetEXT =
0299     PFNGLVERTEXARRAYINDEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayIndexOffsetEXT"));
0300     glVertexArrayMultiTexCoordOffsetEXT =
0301     PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayMultiTexCoordOffsetEXT"));
0302     glVertexArrayNormalOffsetEXT =
0303     PFNGLVERTEXARRAYNORMALOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayNormalOffsetEXT"));
0304     glVertexArraySecondaryColorOffsetEXT =
0305     PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArraySecondaryColorOffsetEXT"));
0306     glVertexArrayTexCoordOffsetEXT =
0307     PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayTexCoordOffsetEXT"));
0308     glVertexArrayVertexAttribBindingEXT =
0309     PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribBindingEXT"));
0310     glVertexArrayVertexAttribDivisorEXT =
0311     PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribDivisorEXT"));
0312     glVertexArrayVertexAttribFormatEXT =
0313     PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribFormatEXT"));
0314     glVertexArrayVertexAttribIFormatEXT =
0315     PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIFormatEXT"));
0316     glVertexArrayVertexAttribIOffsetEXT =
0317     PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIOffsetEXT"));
0318     glVertexArrayVertexAttribLFormatEXT =
0319     PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLFormatEXT"));
0320     glVertexArrayVertexAttribLOffsetEXT =
0321     PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLOffsetEXT"));
0322     glVertexArrayVertexAttribOffsetEXT =
0323     PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribOffsetEXT"));
0324     glVertexArrayVertexBindingDivisorEXT =
0325     PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexBindingDivisorEXT"));
0326     glVertexArrayVertexBuffer =
0327     PFNGLVERTEXARRAYVERTEXBUFFERPROC(glfwGetProcAddress("glVertexArrayVertexBuffer"));

```

```

02480     glVertexArrayVertexBuffers =
02481         PFNGLVERTEXARRAYVERTEXBUFFERSPROC(glfwGetProcAddress("glVertexArrayVertexBuffers"));
02482     glVertexArrayVertexOffsetEXT =
02483         PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexOffsetEXT"));
02484     glVertexAttrib1d = PFNGLVERTEXATTRIB1DPROC(glfwGetProcAddress("glVertexAttrib1d"));
02485     glVertexAttrib1dv = PFNGLVERTEXATTRIB1DVPROC(glfwGetProcAddress("glVertexAttrib1dv"));
02486     glVertexAttrib1fv = PFNGLVERTEXATTRIB1FVPROC(glfwGetProcAddress("glVertexAttrib1fv"));
02487     glVertexAttrib1ls = PFNGLVERTEXATTRIB1SPROC(glfwGetProcAddress("glVertexAttrib1ls"));
02488     glVertexAttrib1sv = PFNGLVERTEXATTRIB1SVPROC(glfwGetProcAddress("glVertexAttrib1sv"));
02489     glVertexAttrib2d = PFNGLVERTEXATTRIB2DPROC(glfwGetProcAddress("glVertexAttrib2d"));
02490     glVertexAttrib2dv = PFNGLVERTEXATTRIB2DVPROC(glfwGetProcAddress("glVertexAttrib2dv"));
02491     glVertexAttrib2f = PFNGLVERTEXATTRIB2FPROC(glfwGetProcAddress("glVertexAttrib2f"));
02492     glVertexAttrib2fv = PFNGLVERTEXATTRIB2FVPROC(glfwGetProcAddress("glVertexAttrib2fv"));
02493     glVertexAttrib2s = PFNGLVERTEXATTRIB2SPROC(glfwGetProcAddress("glVertexAttrib2s"));
02494     glVertexAttrib2sv = PFNGLVERTEXATTRIB2SVPROC(glfwGetProcAddress("glVertexAttrib2sv"));
02495     glVertexAttrib3d = PFNGLVERTEXATTRIB3DPROC(glfwGetProcAddress("glVertexAttrib3d"));
02496     glVertexAttrib3dv = PFNGLVERTEXATTRIB3DVPROC(glfwGetProcAddress("glVertexAttrib3dv"));
02497     glVertexAttrib3f = PFNGLVERTEXATTRIB3FPROC(glfwGetProcAddress("glVertexAttrib3f"));
02498     glVertexAttrib3fv = PFNGLVERTEXATTRIB3FVPROC(glfwGetProcAddress("glVertexAttrib3fv"));
02499     glVertexAttrib3s = PFNGLVERTEXATTRIB3SPROC(glfwGetProcAddress("glVertexAttrib3s"));
02500     glVertexAttrib3sv = PFNGLVERTEXATTRIB3SVPROC(glfwGetProcAddress("glVertexAttrib3sv"));
02501     glVertexAttrib4Nbv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4Nbv"));
02502     glVertexAttrib4Niv = PFNGLVERTEXATTRIB4NIVPROC(glfwGetProcAddress("glVertexAttrib4Niv"));
02503     glVertexAttrib4Nsv = PFNGLVERTEXATTRIB4NSVPROC(glfwGetProcAddress("glVertexAttrib4Nsv"));
02504     glVertexAttrib4Nub = PFNGLVERTEXATTRIB4NUBPROC(glfwGetProcAddress("glVertexAttrib4Nub"));
02505     glVertexAttrib4Nuv = PFNGLVERTEXATTRIB4NUVPROC(glfwGetProcAddress("glVertexAttrib4Nuv"));
02506     glVertexAttrib4Nusv = PFNGLVERTEXATTRIB4NUSVPROC(glfwGetProcAddress("glVertexAttrib4Nusv"));
02507     glVertexAttrib4bv = PFNGLVERTEXATTRIB4BVPROC(glfwGetProcAddress("glVertexAttrib4bv"));
02508     glVertexAttrib4d = PFNGLVERTEXATTRIB4DPROC(glfwGetProcAddress("glVertexAttrib4d"));
02509     glVertexAttrib4dv = PFNGLVERTEXATTRIB4DVPROC(glfwGetProcAddress("glVertexAttrib4dv"));
02510     glVertexAttrib4f = PFNGLVERTEXATTRIB4FPROC(glfwGetProcAddress("glVertexAttrib4f"));
02511     glVertexAttrib4fv = PFNGLVERTEXATTRIB4FVPROC(glfwGetProcAddress("glVertexAttrib4fv"));
02512     glVertexAttrib4iv = PFNGLVERTEXATTRIB4IVPROC(glfwGetProcAddress("glVertexAttrib4iv"));
02513     glVertexAttrib4s = PFNGLVERTEXATTRIB4SPROC(glfwGetProcAddress("glVertexAttrib4s"));
02514     glVertexAttrib4sv = PFNGLVERTEXATTRIB4SVPROC(glfwGetProcAddress("glVertexAttrib4sv"));
02515     glVertexAttrib4ubv = PFNGLVERTEXATTRIB4UBVPROC(glfwGetProcAddress("glVertexAttrib4ubv"));
02516     glVertexAttrib4uiv = PFNGLVERTEXATTRIB4UIVPROC(glfwGetProcAddress("glVertexAttrib4uiv"));
02517     glVertexAttrib4usv = PFNGLVERTEXATTRIB4USVPROC(glfwGetProcAddress("glVertexAttrib4usv"));
02518     glVertexAttribBinding = PFNGLVERTEXATTRIBBINDINGPROC(glfwGetProcAddress("glVertexAttribBinding"));
02519     glVertexAttribDivisor = PFNGLVERTEXATTRIBDIVISORPROC(glfwGetProcAddress("glVertexAttribDivisor"));
02520     glVertexAttribDivisorARB =
02521         PFNGLVERTEXATTRIBDIVISORARBPROC(glfwGetProcAddress("glVertexAttribDivisorARB"));
02522     glVertexAttribFormat = PFNGLVERTEXATTRIBFORMATPROC(glfwGetProcAddress("glVertexAttribFormat"));
02523     glVertexAttribFormatNV =
02524         PFNGLVERTEXATTRIBFORMATNVPROC(glfwGetProcAddress("glVertexAttribFormatNV"));
02525     glVertexAttribIi = PFNGLVERTEXATTRIBI1IPROC(glfwGetProcAddress("glVertexAttribIi"));
02526     glVertexAttribIiiv = PFNGLVERTEXATTRIBI1IVPROC(glfwGetProcAddress("glVertexAttribIiiv"));
02527     glVertexAttribIiui = PFNGLVERTEXATTRIBI1UIPROC(glfwGetProcAddress("glVertexAttribIiui"));
02528     glVertexAttribIiuiiv = PFNGLVERTEXATTRIBI1UIVPROC(glfwGetProcAddress("glVertexAttribIiuiiv"));
02529     glVertexAttribI2i = PFNGLVERTEXATTRIBI2IPROC(glfwGetProcAddress("glVertexAttribI2i"));
02530     glVertexAttribI2iiv = PFNGLVERTEXATTRIBI2IVPROC(glfwGetProcAddress("glVertexAttribI2iiv"));
02531     glVertexAttribI2iui = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2iui"));
02532     glVertexAttribI2iuiiv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2iuiiv"));
02533     glVertexAttribI3i = PFNGLVERTEXATTRIBI3IPROC(glfwGetProcAddress("glVertexAttribI3i"));
02534     glVertexAttribI3iiv = PFNGLVERTEXATTRIBI3IVPROC(glfwGetProcAddress("glVertexAttribI3iiv"));
02535     glVertexAttribI3ui = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3ui"));
02536     glVertexAttribI3uiv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3uiv"));
02537     glVertexAttribI4i = PFNGLVERTEXATTRIBI4IPROC(glfwGetProcAddress("glVertexAttribI4i"));
02538     glVertexAttribI4iv = PFNGLVERTEXATTRIBI4IVPROC(glfwGetProcAddress("glVertexAttribI4iv"));
02539     glVertexAttribI4sv = PFNGLVERTEXATTRIBI4SVPROC(glfwGetProcAddress("glVertexAttribI4sv"));
02540     glVertexAttribI4ubv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4ubv"));
02541     glVertexAttribI4uiv = PFNGLVERTEXATTRIBI4UIVPROC(glfwGetProcAddress("glVertexAttribI4uiv"));
02542     glVertexAttribI4usv = PFNGLVERTEXATTRIBI4USVPROC(glfwGetProcAddress("glVertexAttribI4usv"));
02543     glVertexAttribIFormat = PFNGLVERTEXATTRIBIFORMATPROC(glfwGetProcAddress("glVertexAttribIFormat"));
02544     glVertexAttribIFormatNV =
02545         PFNGLVERTEXATTRIBIFORMATNVPROC(glfwGetProcAddress("glVertexAttribIFormatNV"));
02546     glVertexAttribIPointer =
02547         PFNGLVERTEXATTRIBIPOINTERC(glfwGetProcAddress("glVertexAttribIPointer"));
02548     glVertexAttribL1d = PFNGLVERTEXATTRIBL1DPROC(glfwGetProcAddress("glVertexAttribL1d"));
02549     glVertexAttribL1dv = PFNGLVERTEXATTRIBL1DVPROC(glfwGetProcAddress("glVertexAttribL1dv"));
02550     glVertexAttribL1i64NV = PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
02551     glVertexAttribL1i64NV =
02552         PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
02553     glVertexAttribL1i64ARB =
02554         PFNGLVERTEXATTRIBL1I64VARBPROC(glfwGetProcAddress("glVertexAttribL1i64vARB"));
02555     glVertexAttribL1i64NV =
02556         PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64vNV"));
02557     glVertexAttribL2d = PFNGLVERTEXATTRIBL2DPROC(glfwGetProcAddress("glVertexAttribL2d"));
02558     glVertexAttribL2dv = PFNGLVERTEXATTRIBL2DVPROC(glfwGetProcAddress("glVertexAttribL2dv"));

```

```

02556     glVertexAttribL2i64NV = PFNGLVERTEXATTRIBL2I64NVPROC(glfwGetProcAddress("glVertexAttribL2i64NV"));
02557     glVertexAttribL2i64vNV =
02558     PFNGLVERTEXATTRIBL2I64VNVPROC(glfwGetProcAddress("glVertexAttribL2i64vNV"));
02559     glVertexAttribL2ui64NV =
02560     PFNGLVERTEXATTRIBL2UI64VNVPROC(glfwGetProcAddress("glVertexAttribL2ui64NV"));
02561     glVertexAttribL2ui64vNV =
02562     PFNGLVERTEXATTRIBL2UI64VNVPROC(glfwGetProcAddress("glVertexAttribL2ui64vNV"));
02563     glVertexAttribL3d =
02564     PFNGLVERTEXATTRIBL3DPROC(glfwGetProcAddress("glVertexAttribL3d"));
02565     glVertexAttribL3dv =
02566     PFNGLVERTEXATTRIBL3DVPROC(glfwGetProcAddress("glVertexAttribL3dv"));
02567     glVertexAttribL3i64NV =
02568     PFNGLVERTEXATTRIBL3I64VNVPROC(glfwGetProcAddress("glVertexAttribL3i64NV"));
02569     glVertexAttribL3i64vNV =
02570     PFNGLVERTEXATTRIBL3I64VNVPROC(glfwGetProcAddress("glVertexAttribL3i64vNV"));
02571     glVertexAttribL3ui64NV =
02572     PFNGLVERTEXATTRIBL3UI64VNVPROC(glfwGetProcAddress("glVertexAttribL3ui64NV"));
02573     glVertexAttribL3ui64vNV =
02574     PFNGLVERTEXATTRIBL3UI64VNVPROC(glfwGetProcAddress("glVertexAttribL3ui64vNV"));
02575     glVertexAttribL4d =
02576     PFNGLVERTEXATTRIBL4DPROC(glfwGetProcAddress("glVertexAttribL4d"));
02577     glVertexAttribL4dv =
02578     PFNGLVERTEXATTRIBL4DVPROC(glfwGetProcAddress("glVertexAttribL4dv"));
02579     glVertexAttribL4i64NV =
02580     PFNGLVERTEXATTRIBL4I64VNVPROC(glfwGetProcAddress("glVertexAttribL4i64NV"));
02581     glVertexAttribL4i64vNV =
02582     PFNGLVERTEXATTRIBL4I64VNVPROC(glfwGetProcAddress("glVertexAttribL4i64vNV"));
02583     glVertexAttribL4ui64NV =
02584     PFNGLVERTEXATTRIBL4UI64VNVPROC(glfwGetProcAddress("glVertexAttribL4ui64NV"));
02585     glVertexAttribL4ui64vNV =
02586     PFNGLVERTEXATTRIBL4UI64VNVPROC(glfwGetProcAddress("glVertexAttribL4ui64vNV"));
02587     glVertexAttribLFormat =
02588     PFNGLVERTEXATTRIBLFORMATPROC(glfwGetProcAddress("glVertexAttribLFormat"));
02589     glVertexAttribLFormatNV =
02590     PFNGLVERTEXATTRIBLFORMATNVPROC(glfwGetProcAddress("glVertexAttribLFormatNV"));
02591     glVertexAttribLPointer =
02592     PFNGLVERTEXATTRIBLPOINTERPROC(glfwGetProcAddress("glVertexAttribLPointer"));
02593     glVertexAttribP1ui =
02594     PFNGLVERTEXATTRIBP1UIPROC(glfwGetProcAddress("glVertexAttribP1ui"));
02595     glVertexAttribP1uiv =
02596     PFNGLVERTEXATTRIBP1UIVPROC(glfwGetProcAddress("glVertexAttribP1uiv"));
02597     glVertexAttribP2ui =
02598     PFNGLVERTEXATTRIBP2UIPROC(glfwGetProcAddress("glVertexAttribP2ui"));
02599     glVertexAttribP3ui =
02600     PFNGLVERTEXATTRIBP3UIPROC(glfwGetProcAddress("glVertexAttribP3ui"));
02601     glVertexAttribP3uiv =
02602     PFNGLVERTEXATTRIBP3UIVPROC(glfwGetProcAddress("glVertexAttribP3uiv"));
02603     glVertexAttribP4ui =
02604     PFNGLVERTEXATTRIBP4UIPROC(glfwGetProcAddress("glVertexAttribP4ui"));
02605     glVertexAttribP4uiv =
02606     PFNGLVERTEXATTRIBP4UIVPROC(glfwGetProcAddress("glVertexAttribP4uiv"));
02607     glVertexAttribPointer =
02608     PFNGLVERTEXATTRIBPOINTERPROC(glfwGetProcAddress("glVertexAttribPointer"));
02609     glVertexBindingDivisor =
02610     PFNGLVERTEXBINDINGDIVISORPROC(glfwGetProcAddress("glVertexBindingDivisor"));
02611     glVertexFormatNV =
02612     PFNGLVERTEXFORMATNVPROC(glfwGetProcAddress("glVertexFormatNV"));
02613     glViewport =
02614     PFNGLVIEWPORTPROC(glfwGetProcAddress("glViewport"));
02615     glViewportArray =
02616     PFNGLVIEWPORTARRAYPROC(glfwGetProcAddress("glViewportArray"));
02617     glViewportIndexeddf =
02618     PFNGLVIEWPORTINDEXEDFPROC(glfwGetProcAddress("glViewportIndexeddf"));
02619     glViewportIndexeddfv =
02620     PFNGLVIEWPORTINDEXEDFVPROC(glfwGetProcAddress("glViewportIndexeddfv"));
02621     glViewportPositionWScaleNV =
02622     PFNGLVIEWPORTPOSITIONWSCALENVPROC(glfwGetProcAddress("glViewportPositionWScaleNV"));
02623     glViewportSwizzleNV =
02624     PFNGLVIEWPORTSWIZZLENVPROC(glfwGetProcAddress("glViewportSwizzleNV"));
02625     glWaitSync =
02626     PFNGLWAITSYNCPROC(glfwGetProcAddress("glWaitSync"));
02627     glWaitVkSemaphoreNV =
02628     PFNGLWAITVKSEMAPHORENVPROC(glfwGetProcAddress("glWaitVkSemaphoreNV"));
02629     glWeightPathsNV =
02630     PFNGLWEIGHTPATHSNVPROC(glfwGetProcAddress("glWeightPathsNV"));
02631     glWindowRectanglesEXT =
02632     PFNGLWINDOWRECTANGLESEXTPROC(glfwGetProcAddress("glWindowRectanglesEXT"));
02633     #endif
02634
02635 // 使用している GPU のバッファアライメントを調べる
02636     glGetIntegerv(GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT, &ggBufferAlignment);
02637 }
02638
02639 // OpenGL のエラーをチェックする
02640 //
02641 // 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
02642 //
02643 // 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
02644 //
02645 void gg::ggError(const std::string& name, unsigned int line)
02646 {
02647     const GLenum error{ glfwGetError() };
02648
02649     if (error != GL_NO_ERROR)
02650     {
02651         if (!name.empty())
02652         {
02653             std::cerr << name;
02654             if (line > 0) std::cerr << " (" << line << ")";
02655             std::cerr << ":" << " ";
02656         }
02657
02658         switch (error)
02659         {
02660             case GL_INVALID_ENUM:
02661                 std::cerr << "An unacceptable value is specified for an enumerated argument" << std::endl;
02662                 break;
02663             case GL_INVALID_VALUE:
02664                 std::cerr << "A numeric argument is out of range" << std::endl;
02665                 break;
02666         }
02667     }
02668 }

```

```

02630     case GL_INVALID_OPERATION:
02631         std::cerr << "The specified operation is not allowed in the current state" << std::endl;
02632         break;
02633     case GL_OUT_OF_MEMORY:
02634         std::cerr << "There is not enough memory left to execute the command" << std::endl;
02635         break;
02636     case GL_INVALID_FRAMEBUFFER_OPERATION:
02637         std::cerr << "The specified operation is not allowed current frame buffer" << std::endl;
02638         break;
02639     default:
02640         std::cerr << "An OpenGL error has occurred: " << std::hex << std::showbase << error << std::endl;
02641         break;
02642     }
02643 }
02645
02646 // FBO のエラーをチェックする
02647 // 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
02648 //
02649 // 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
02650 //
02651 // 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
02652 //
02653 void gg::ggFBOError(const std::string& name, unsigned int line)
02654 {
02655     const auto status{ glCheckFramebufferStatus(GL_FRAMEBUFFER) };
02656
02657     if (status != GL_FRAMEBUFFER_COMPLETE)
02658     {
02659         if (!name.empty())
02660         {
02661             std::cerr << name;
02662             if (line > 0) std::cerr << " (" << line << ")";
02663             std::cerr << ":" ;
02664         }
02665
02666         switch (status)
02667         {
02668             case GL_FRAMEBUFFER_UNDEFINED:
02669                 std::cerr << "the default framebuffer does not exist" << std::endl;
02670                 break;
02671             case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
02672                 std::cerr << "Framebuffer incomplete, duplicate attachment" << std::endl;
02673                 break;
02674             case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
02675                 std::cerr << "Framebuffer incomplete, missing attachment" << std::endl;
02676                 break;
02677             case GL_FRAMEBUFFER_UNSUPPORTED:
02678                 std::cerr << "Unsupported framebuffer internal" << std::endl;
02679                 break;
02680             case GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE:
02681                 std::cerr << "The value of GL_RENDERBUFFER_SAMPLES is not"
02682                 " the same for all attached renderbuffers or,"
02683                 " if the attached images are a mix of renderbuffers and textures,"
02684                 " the value of GL_RENDERBUFFER_SAMPLES is not zero" << std::endl;
02685                 break;
02686 #if !defined(GL_GLES_PROTOTYPES)
02687             case GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS:
02688                 std::cerr << "Any framebuffer attachment is layered,"
02689                 " and any populated attachment is not layered,"
02690                 " or if all populated color attachments are not from textures"
02691                 " of the same target" << std::endl;
02692                 break;
02693             case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER:
02694                 std::cerr << "Framebuffer incomplete, missing draw buffer" << std::endl;
02695                 break;
02696             case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
02697                 std::cerr << "Framebuffer incomplete, missing read buffer" << std::endl;
02698                 break;
02699 #endif
02700         default:
02701             std::cerr << "Programming error; will fail on all hardware: " << std::hex << std::showbase <<
02702             status << std::endl;
02703             break;
02704     }
02705 }
02706
02707 //
02708 // 変換行列：行列とベクトルの積 c ← a × b
02709 //
02710 void gg::GgMatrix::projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02711 {
02712     for (int i = 0; i < 4; ++i)
02713     {
02714         c[i] = a[0 + i] * b[0] + a[4 + i] * b[1] + a[8 + i] * b[2] + a[12 + i] * b[3];
02715     }
}

```

```
02716 }
02717 //
02718 // 変換行列：行列と行列の積  $c \leftarrow a \times b$ 
02719 //
02720 //
02721 void gg::GgMatrix::multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02722 {
02723     for (int i = 0; i < 16; ++i)
02724     {
02725         int j = i & 3, k = i & ~3;
02726
02727         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];
02728     }
02729 }
02730
02731 //
02732 // 変換行列：単位行列を設定する
02733 //
02734 gg::GgMatrix& gg::GgMatrix::loadIdentity()
02735 {
02736     *this =
02737     {
02738         1.0f, 0.0f, 0.0f, 0.0f,
02739         0.0f, 1.0f, 0.0f, 0.0f,
02740         0.0f, 0.0f, 1.0f, 0.0f,
02741         0.0f, 0.0f, 0.0f, 1.0f
02742     };
02743
02744     return *this;
02745 }
02746
02747 //
02748 // 変換行列：平行移動変換行列を設定する
02749 //
02750 gg::GgMatrix& gg::GgMatrix::loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02751 {
02752     *this =
02753     {
02754         w, 0.0f, 0.0f, 0.0f,
02755         0.0f, w, 0.0f, 0.0f,
02756         0.0f, 0.0f, w, 0.0f,
02757         x, y, z, w
02758     };
02759
02760     return *this;
02761 }
02762
02763 //
02764 // 変換行列：拡大縮小変換行列を設定する
02765 //
02766 gg::GgMatrix& gg::GgMatrix::loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02767 {
02768     *this =
02769     {
02770         x, 0.0f, 0.0f, 0.0f,
02771         0.0f, y, 0.0f, 0.0f,
02772         0.0f, 0.0f, z, 0.0f,
02773         0.0f, 0.0f, 0.0f, w
02774     };
02775
02776     return *this;
02777 }
02778
02779 //
02780 // 変換行列：x 軸中心の回転変換行列を設定する
02781 //
02782 gg::GgMatrix& gg::GgMatrix::loadRotateX(GLfloat a)
02783 {
02784     const auto c{ cosf(a) };
02785     const auto s{ sinf(a) };
02786
02787     *this =
02788     {
02789         1.0f, 0.0f, 0.0f, 0.0f,
02790         0.0f, c, s, 0.0f,
02791         0.0f, -s, c, 0.0f,
02792         0.0f, 0.0f, 0.0f, 1.0f
02793     };
02794
02795     return *this;
02796 }
02797
02798 //
02799 // 変換行列：y 軸中心の回転変換行列を設定する
02800 //
02801 gg::GgMatrix& gg::GgMatrix::loadRotateY(GLfloat a)
02802 {
```

```

02803     const auto c{ cosf(a) };
02804     const auto s{ sinf(a) };
02805
02806     *this =
02807     {
02808         c,      0.0f, -s,      0.0f,
02809         0.0f,  1.0f,  0.0f,  0.0f,
02810         s,      0.0f,  c,      0.0f,
02811         0.0f,  0.0f,  0.0f,  1.0f
02812     };
02813
02814     return *this;
02815 }
02816
02817 ///////////////////////////////////////////////////////////////////
02818 // 変換行列：z 軸中心の回転変換行列を設定する
02819 //
02820 gg::GgMatrix& gg::GgMatrix::loadRotateZ(GLfloat a)
02821 {
02822     const auto c{ cosf(a) };
02823     const auto s{ sinf(a) };
02824
02825     *this =
02826     {
02827         c,      s,      0.0f,  0.0f,
02828         -s,     c,      0.0f,  0.0f,
02829         0.0f,  0.0f,  1.0f,  0.0f,
02830         0.0f,  0.0f,  0.0f,  1.0f
02831     };
02832
02833     return *this;
02834 }
02835
02836 ///////////////////////////////////////////////////////////////////
02837 // 変換行列：任意軸中心の回転変換行列を設定する
02838 //
02839 gg::GgMatrix& gg::GgMatrix::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
02840 {
02841     const auto d{ sqrtf(x * x + y * y + z * z) };
02842
02843     if (d > 0.0f)
02844     {
02845         const auto l{ x / d }, m{ y / d }, n{ z / d };
02846         const auto l2{ l * l }, m2{ m * m }, n2{ n * n };
02847         const auto lm{ l * m }, mn{ m * n }, nl{ n * l };
02848         const auto c{ cosf(a) }, cl{ 1.0f - c };
02849         const auto s{ sinf(a) };
02850
02851         *this =
02852         {
02853             (1.0f - l2) * c + l2,
02854             lm * cl + n * s,
02855             nl * cl - m * s,
02856             0.0f,
02857
02858             lm * cl - n * s,
02859             (1.0f - m2) * c + m2,
02860             mn * cl + l * s,
02861             0.0f,
02862
02863             nl * cl + m * s,
02864             mn * cl - l * s,
02865             (1.0f - n2) * c + n2,
02866             0.0f,
02867
02868             0.0f,
02869             0.0f,
02870             0.0f,
02871             1.0f,
02872         };
02873     }
02874
02875     return *this;
02876 }
02877
02878 ///////////////////////////////////////////////////////////////////
02879 // 変換行列：転置行列を設定する
02880 //
02881 gg::GgMatrix& gg::GgMatrix::loadTranspose(const GLfloat* marray)
02882 {
02883     *this =
02884     {
02885         marray[ 0], marray[ 4],
02886         marray[ 4], marray[ 8],
02887         marray[ 8], marray[12],
02888         marray[12], marray[ 1],
02889     };

```

```

02890     marray[ 5],
02891     marray[ 9],
02892     marray[13],
02893     marray[ 2],
02894     marray[ 6],
02895     marray[10],
02896     marray[14],
02897     marray[ 3],
02898     marray[ 7],
02899     marray[11],
02900     marray[15]
02901 };
02902
02903     return *this;
02904 }
02905
02906 //
02907 // 変換行列：逆行列を設定する
02908 //
02909 gg::GgMatrix& gg::GgMatrix::loadInvert(const GLfloat* marray)
02910 {
02911     GLfloat lu[20];
02912     GLfloat* plu[4];
02913
02914     // j 行の要素の値の絶対値の最大値を plu[j][4] に求める
02915     for (int j = 0; j < 4; ++j)
02916     {
02917         auto max{ fabs(*plu[j] = lu + 5 * j) = *(marray++) };
02918
02919         for (int i = 0; ++i < 4; )
02920         {
02921             auto a{ fabs(plu[j][i] = *(marray++) ) };
02922             if (a > max) max = a;
02923         }
02924         if (max == 0.0f) return *this;
02925         plu[j][4] = 1.0f / max;
02926     }
02927
02928     // ピボットを考慮した LU 分解
02929     for (int j = 0; j < 4; ++j)
02930     {
02931         auto max{ fabs(plu[j][j] * plu[j][4]) };
02932         int i = j;
02933
02934         for (int k = j; ++k < 4; )
02935         {
02936             auto a{ fabs(plu[k][j] * plu[k][4]) };
02937             if (a > max)
02938             {
02939                 max = a;
02940                 i = k;
02941             }
02942         }
02943         if (i > j)
02944         {
02945             auto* t{ plu[j] };
02946             plu[j] = plu[i];
02947             plu[i] = t;
02948         }
02949         if (plu[j][j] == 0.0f) return *this;
02950         for (int k = j; ++k < 4; )
02951         {
02952             plu[k][j] /= plu[j][j];
02953             for (int i = j; ++i < 4; )
02954             {
02955                 plu[k][i] -= plu[j][i] * plu[k][j];
02956             }
02957         }
02958     }
02959
02960     // LU 分解から逆行列を求める
02961     for (int k = 0; k < 4; ++k)
02962     {
02963         // array に単位行列を設定する
02964         for (int i = 0; i < 4; ++i)
02965         {
02966             (*this)[i * 4 + k] = (plu[i] == lu + k * 5) ? 1.0f : 0.0f;
02967         }
02968         // lu から逆行列を求める
02969         for (int i = 0; i < 4; ++i)
02970         {
02971             for (int j = i; ++j < 4; )
02972             {
02973                 (*this)[j * 4 + k] -= (*this)[i * 4 + k] * plu[j][i];
02974             }
02975         }
02976         for (int i = 4; --i >= 0; )

```

```

02977    {
02978        for (int j = i; ++j < 4;)
02979        {
02980            (*this)[i * 4 + k] -= plu[i][j] * (*this)[j * 4 + k];
02981        }
02982        (*this)[i * 4 + k] /= plu[i][i];
02983    }
02984 }
02985
02986 return *this;
02987 }
02988
02989 // 
02990 // 変換行列：法線変換行列を設定する
02991 //
02992 gg::GgMatrix& gg::GgMatrix::loadNormal(const GLfloat* marray)
02993 {
02994     *this =
02995     {
02996         marray[ 5] * marray[10] - marray[ 6] * marray[ 9],
02997         marray[ 6] * marray[ 8] - marray[ 4] * marray[10],
02998         marray[ 4] * marray[ 9] - marray[ 5] * marray[ 8],
02999         0.0f,
03000
03001         marray[ 9] * marray[ 2] - marray[10] * marray[ 1],
03002         marray[10] * marray[ 0] - marray[ 8] * marray[ 2],
03003         marray[ 8] * marray[ 1] - marray[ 9] * marray[ 0],
03004         0.0f,
03005
03006         marray[ 1] * marray[ 6] - marray[ 2] * marray[ 5],
03007         marray[ 2] * marray[ 4] - marray[ 0] * marray[ 6],
03008         marray[ 0] * marray[ 5] - marray[ 1] * marray[ 4],
03009         0.0f,
03010
03011         0.0f,
03012         0.0f,
03013         0.0f,
03014         1.0f
03015     };
03016
03017     return *this;
03018 }
03019
03020 // 
03021 // 変換行列：ビュー変換行列を設定する
03022 //
03023 gg::GgMatrix& gg::GgMatrix::loadLookat(
03024     GLfloat ex, GLfloat ey, GLfloat ez,
03025     GLfloat tx, GLfloat ty, GLfloat tz,
03026     GLfloat ux, GLfloat uy, GLfloat uz
03027 )
03028 {
03029     // z 軸 = e - t
03030     const auto zx{ ex - tx };
03031     const auto zy{ ey - ty };
03032     const auto zz{ ez - tz };
03033
03034     // z 軸の長さ
03035     const auto z{ sqrt(zx * zx + zy * zy + zz * zz) };
03036
03037     // x 軸 = u × z 軸
03038     const auto xx{ uy * zz - uz * zy };
03039     const auto xy{ uz * zx - ux * zz };
03040     const auto xz{ ux * zy - uy * zx };
03041
03042     // x 軸の長さ
03043     const auto x{ sqrt(xx * xx + xy * xy + xz * xz) };
03044
03045     // y 軸 = z 軸 × x 軸
03046     const auto yx{ zy * xx - zz * xy };
03047     const auto yy{ zz * xx - zx * xz };
03048     const auto yz{ zx * xy - zy * xx };
03049
03050     // y 軸の長さ
03051     const auto y{ sqrt(yx * yx + yy * yy + yz * yz) };
03052
03053     // y 軸の長さをチェック
03054     if (fabs(y) < std::numeric_limits<float>::epsilon()) return *this;
03055
03056     (*this)[ 0] = xx / x;
03057     (*this)[ 1] = yx / y;
03058     (*this)[ 2] = zx / z;
03059     (*this)[ 3] = 0.0f,
03060
03061     (*this)[ 4] = xy / x;
03062     (*this)[ 5] = yy / y;
03063     (*this)[ 6] = zy / z;

```

```
03064     (*this)[ 7] = 0.0f,
03065
03066     (*this)[ 8] = xz / x;
03067     (*this)[ 9] = yz / y;
03068     (*this)[10] = zz / z;
03069     (*this)[11] = 0.0f,
03070
03071     (*this)[12] = -(ex * (*this)[ 0] + ey * (*this)[ 4] + ez * (*this)[ 8]);
03072     (*this)[13] = -(ex * (*this)[ 1] + ey * (*this)[ 5] + ez * (*this)[ 9]);
03073     (*this)[14] = -(ex * (*this)[ 2] + ey * (*this)[ 6] + ez * (*this)[10]);
03074     (*this)[15] = 1.0f;
03075
03076     return *this;
03077 }
03078
03079 /**
03080 // 変換行列：平行投影変換行列を設定する
03081 /**
03082 gg::GgMatrix& gg::GgMatrix::loadOrthogonal(
03083     GLfloat left, GLfloat right,
03084     GLfloat bottom, GLfloat top,
03085     GLfloat zNear, GLfloat zFar
03086 )
03087 {
03088     const auto dx{ right - left };
03089     const auto dy{ top - bottom };
03090     const auto dz{ zFar - zNear };
03091
03092     if (dx == 0.0f || dy == 0.0f || dz == 0.0f) return *this;
03093
03094     *this =
03095     {
03096         2.0f / dx,
03097         0.0f,
03098         0.0f,
03099         0.0f,
03100
03101         0.0f,
03102         2.0f / dy,
03103         0.0f,
03104         0.0f,
03105
03106         0.0f,
03107         0.0f,
03108         -2.0f / dz,
03109         0.0f,
03110
03111         -(right + left) / dx,
03112         -(top + bottom) / dy,
03113         -(zFar + zNear) / dz,
03114         1.0f
03115     };
03116
03117     return *this;
03118 }
03119
03120 /**
03121 // 変換行列：透視投影変換行列を設定する
03122 /**
03123 gg::GgMatrix& gg::GgMatrix::loadFrustum(
03124     GLfloat left, GLfloat right,
03125     GLfloat bottom, GLfloat top,
03126     GLfloat zNear, GLfloat zFar
03127 )
03128 {
03129     const auto dx{ right - left };
03130     const auto dy{ top - bottom };
03131     const auto dz{ zFar - zNear };
03132
03133     if (dx == 0.0f || dy == 0.0f || dz == 0.0f) return *this;
03134
03135     *this =
03136     {
03137         2.0f * zNear / dx,
03138         0.0f,
03139         0.0f,
03140         0.0f,
03141
03142         0.0f,
03143         2.0f * zNear / dy,
03144         0.0f,
03145         0.0f,
03146
03147         (right + left) / dx,
03148         (top + bottom) / dy,
03149         -(zFar + zNear) / dz,
03150         -1.0f,
```

```

03151     0.0f,
03152     0.0f,
03153     0.0f,
03154     -2.0f * zFar * zNear / dz,
03155     0.0f
03156   };
03157
03158   return *this;
03159 }
03160
03161 // 
03162 // 変換行列：画角から透視投影変換行列を設定する
03163 //
03164 gg::GgMatrix& gg::GgMatrix::loadPerspective(
03165   GLfloat fovy, GLfloat aspect,
03166   GLfloat zNear, GLfloat zFar
03167 )
03168 {
03169   const auto dz{ zFar - zNear };
03170
03171   if (dz == 0.0f) return *this;
03172
03173   const auto f{ 1.0f / tanf(fovy * 0.5f) };
03174
03175   *this =
03176   {
03177     f / aspect,
03178     0.0f,
03179     0.0f,
03180     0.0f,
03181
03182     0.0f,
03183     f,
03184     0.0f,
03185     0.0f,
03186
03187     0.0f,
03188     0.0f,
03189     -(zFar + zNear) / dz,
03190     -1.0f,
03191
03192     0.0f,
03193     0.0f,
03194     -2.0f * zFar * zNear / dz,
03195     0.0f
03196   };
03197
03198   return *this;
03199 }
03200
03201 // 
03202 // 四元数：GgQuaternion 型の四元数 p, q の積を r に求める
03203 //
03204 void gg::GgQuaternion::multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const
03205 {
03206   r[0] = p[1] * q[2] - p[2] * q[1] + p[0] * q[3] + p[3] * q[0];
03207   r[1] = p[2] * q[0] - p[0] * q[2] + p[1] * q[3] + p[3] * q[1];
03208   r[2] = p[0] * q[1] - p[1] * q[0] + p[2] * q[3] + p[3] * q[2];
03209   r[3] = p[3] * q[3] - p[0] * q[0] - p[1] * q[1] - p[2] * q[2];
03210 }
03211
03212 // 
03213 // 四元数：GgQuaternion 型の四元数 q が表す変換行列を m に求める
03214 //
03215 void gg::GgQuaternion::toMatrix(GLfloat* m, const GLfloat* q) const
03216 {
03217   const auto xx{ q[0] * q[0] * 2.0f };
03218   const auto yy{ q[1] * q[1] * 2.0f };
03219   const auto zz{ q[2] * q[2] * 2.0f };
03220   const auto xy{ q[0] * q[1] * 2.0f };
03221   const auto yz{ q[1] * q[2] * 2.0f };
03222   const auto zx{ q[2] * q[0] * 2.0f };
03223   const auto xw{ q[0] * q[3] * 2.0f };
03224   const auto yw{ q[1] * q[3] * 2.0f };
03225   const auto zw{ q[2] * q[3] * 2.0f };
03226
03227   m[ 0] = 1.0f - yy - zz;
03228   m[ 1] = xy + zw;
03229   m[ 2] = zx - yw;
03230   m[ 4] = xy - zw;
03231   m[ 5] = 1.0f - zz - xx;
03232   m[ 6] = yz + xw;
03233   m[ 8] = zx + yw;
03234   m[ 9] = yz - xw;
03235   m[10] = 1.0f - xx - yy;
03236   m[ 3] = m[ 7] = m[11] = m[12] = m[13] = m[14] = 0.0f;
03237   m[15] = 1.0f;

```

```

03238 }
03239
03240 // 四元数：回転変換行列 a が表す四元数を q に求める
03241 // 03242 //
03243 void gg::GgQuaternion::toQuaternion(GLfloat* q, const GLfloat* a) const
03244 {
03245     const auto tr{ a[0] + a[5] + a[10] + a[15] };
03246
03247     if (tr > 0.0f)
03248     {
03249         q[3] = sqrtf(tr) * 0.5f;
03250         q[0] = (a[6] - a[9]) * 0.25f / q[3];
03251         q[1] = (a[8] - a[2]) * 0.25f / q[3];
03252         q[2] = (a[1] - a[4]) * 0.25f / q[3];
03253     }
03254 }
03255
03256 //
03257 // 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
03258 //
03259 void gg::GgQuaternion::slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const
03260 {
03261     const auto qr{ ggDot3(q, r) };
03262     const auto ss{ 1.0f - qr * qr };
03263
03264     if (ss == 0.0f)
03265     {
03266         if (p != q)
03267         {
03268             p[0] = q[0];
03269             p[1] = q[1];
03270             p[2] = q[2];
03271             p[3] = q[3];
03272         }
03273     }
03274     else
03275     {
03276         const auto sp{ sqrt(ss) };
03277         const auto ph{ acos(qr) };
03278         const auto pt{ ph * t };
03279         const auto t1{ sin(pt) / sp };
03280         const auto t0{ sin(ph - pt) / sp };
03281
03282         p[0] = q[0] * t0 + r[0] * t1;
03283         p[1] = q[1] * t0 + r[1] * t1;
03284         p[2] = q[2] * t0 + r[2] * t1;
03285         p[3] = q[3] * t0 + r[3] * t1;
03286     }
03287 }
03288
03289 //
03290 // 四元数：(x, y, z) を軸とし角度 a 回転する四元数を求める
03291 //
03292 gg::GgQuaternion& gg::GgQuaternion::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03293 {
03294     const auto l{ x * x + y * y + z * z };
03295     const auto w{ cosf(a *= 0.5f) };
03296
03297     if (fabs(l) > std::numeric_limits<float>::epsilon())
03298     {
03299         const auto s{ sinf(a) / sqrtf(l) };
03300
03301         *this = GgQuaternion{ x * s, y * s, z * s, w };
03302     }
03303     else
03304     {
03305         *this = GgQuaternion{ 0.0f, 0.0f, 0.0f, w };
03306     }
03307
03308     return *this;
03309 }
03310
03311 //
03312 // x 軸中心に角度 a 回転する四元数を格納する
03313 //
03314 gg::GgQuaternion& gg::GgQuaternion::loadRotateX(GLfloat a)
03315 {
03316     const auto t{ a * 0.5f };
03317
03318     *this = GgQuaternion{ sinf(t), 0.0f, 0.0f, cosf(t) };
03319
03320     return *this;
03321 }
03322
03323 //
03324 // y 軸中心に角度 a 回転する四元数を格納する

```

```

03325 //
03326 gg::GgQuaternion& gg::GgQuaternion::loadRotateY(GLfloat a)
03327 {
03328     const auto t{ a * 0.5f };
03329
03330     *this = GgQuaternion{ 0.0f, sinf(t), 0.0f, cosf(t) };
03331
03332     return *this;
03333 }
03334
03335 //
03336 // z 軸を中心に角度 a 回転する四元数を格納する
03337 //
03338 gg::GgQuaternion& gg::GgQuaternion::loadRotateZ(GLfloat a)
03339 {
03340     const auto t{ a * 0.5f };
03341
03342     *this = GgQuaternion{ 0.0f, 0.0f, sinf(t), cosf(t) };
03343
03344     return *this;
03345 }
03346
03347 //
03348 // 四元数：オイラー角 (heading, pitch, roll) にもとづいて四元数を求める
03349 //
03350 gg::GgQuaternion& gg::GgQuaternion::loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll)
03351 {
03352     GgQuaternion h, p, r;
03353
03354     h.loadRotateY(heading);
03355     p.loadRotateX(pitch);
03356     r.loadRotateZ(roll);
03357
03358     *this = h * p * r;
03359
03360     return *this;
03361 }
03362
03363 //
03364 // 四元数：正規化して格納する
03365 //
03366 gg::GgQuaternion& gg::GgQuaternion::loadNormalize(const GLfloat* a)
03367 {
03368     *this = a;
03369
03370     ggNormalize4(data());
03371
03372     return *this;
03373 }
03374
03375 //
03376 // 四元数：共役四元数を格納する
03377 //
03378 gg::GgQuaternion& gg::GgQuaternion::loadConjugate(const GLfloat* a)
03379 {
03380     // w 要素を反転する
03381     data()[0] = a[0];
03382     data()[1] = a[1];
03383     data()[2] = a[2];
03384     data()[3] = -a[3];
03385
03386     return *this;
03387 }
03388
03389 //
03390 // 四元数：逆元を格納する
03391 //
03392 gg::GgQuaternion& gg::GgQuaternion::loadInvert(const GLfloat* a)
03393 {
03394     // ノルムの二乗を求める
03395     const auto l(ggDot4(a, a));
03396
03397     if (l > 0.0f)
03398     {
03399         // 共役四元数を求める
03400         GgQuaternion r;
03401         r.loadConjugate(a);
03402
03403         // ノルムの二乗で割る
03404         data()[0] = r[0] / l;
03405         data()[1] = r[1] / l;
03406         data()[2] = r[2] / l;
03407         data()[3] = r[3] / l;
03408     }
03409
03410     return *this;
03411 }

```

```
03412 //  
03413 // 簡易トラックボール処理：リセット  
03414 //  
03415 //  
03416 void gg::GgTrackball::reset(const GgQuaternion& q)  
03417 {  
03418     // ドラッグ中ではない  
03419     drag = false;  
03420  
03421     // 単位クォーテーニオンに初期値を与える  
03422     *this = cq = q;  
03423  
03424     // 回転行列を初期化する  
03425     GgQuaternion::getMatrix(rt);  
03426 }  
03427  
03428 //  
03429 // 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  
03430 //  
03431 //    Reshape コールバック (resize) の中に実行する  
03432 //    (w, h) ウィンドウサイズ  
03433 //  
03434 void gg::GgTrackball::region(GLfloat w, GLfloat h)  
03435 {  
03436     // 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  
03437     scale[0] = 2.0f / w;  
03438     scale[1] = 2.0f / h;  
03439 }  
03440  
03441 //  
03442 // 簡易トラックボール処理：ドラッグ開始時の処理  
03443 //  
03444 //    マウスボタンを押したときに実行する  
03445 //    (x, y) 現在のマウス位置  
03446 //  
03447 void gg::GgTrackball::begin(GLfloat x, GLfloat y)  
03448 {  
03449     // ドラッグ開始  
03450     drag = true;  
03451  
03452     // ドラッグ開始点を記録する  
03453     start[0] = x;  
03454     start[1] = y;  
03455 }  
03456  
03457 //  
03458 // 簡易トラックボール処理：ドラッグ中の処理  
03459 //  
03460 //    マウスのドラッグ中に実行する  
03461 //    (x, y) 現在のマウス位置  
03462 //  
03463 void gg::GgTrackball::motion(GLfloat x, GLfloat y)  
03464 {  
03465     // ドラッグ中でなければ何もしない  
03466     if (!drag) return;  
03467  
03468     // マウスボインタの位置のドラッグ開始位置からの変位  
03469     const auto dx{ (x - start[0]) * scale[0] };  
03470     const auto dy{ (y - start[1]) * scale[1] };  
03471  
03472     // マウスボインタの位置のドラッグ開始位置からの距離  
03473     const auto a{ hypot(dx, dy) };  
03474  
03475     // 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  
03476     if (a > 0.0)  
03477     {  
03478         // 現在の回転の四元数に作った四元数を掛けて合成する  
03479         *this = ggRotateQuaternion(dy, dx, 0.0f, a * 3.1415926536f) * cq;  
03480  
03481         // 合成した四元数から回転の変換行列を求める  
03482         GgQuaternion::getMatrix(rt);  
03483     }  
03484 }  
03485  
03486 //  
03487 // 簡易トラックボール処理：回転角の修正  
03488 //  
03489 //    現在の回転角を修正する  
03490 //    q 修正分の回転角を表す四元数  
03491 //  
03492 void gg::GgTrackball::rotate(const GgQuaternion& q)  
03493 {  
03494     // ドラッグ中なら何もしない  
03495     if (drag) return;  
03496  
03497     // 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  
03498     *this = q * cq;
```

```

03499
03500 // 合成した四元数から回軸の変換行列を求める
03501 GgQuaternion::getMatrix(rt);
03502
03503 // 誤差を吸収するために正規化して保存する
03504 cq = normalize();
03505 }
03506
03507 //
03508 // 簡易トラックボール処理：停止時の処理
03509 //
03510 // マウスボタンを離したときに実行する
03511 // (x, y) 現在のマウス位置
03512 //
03513 void gg::GgTrackball::end(GLfloat x, GLfloat y)
03514 {
03515 // ドラッグ終了点における回軸を求める
03516 motion(x, y);
03517
03518 // 誤差を吸収するために正規化して保存する
03519 cq = normalize();
03520
03521 // ドラッグ終了
03522 drag = false;
03523 }
03524
03525 //
03526 // 配列に格納された画像の内容を TGA ファイルに保存する
03527 //
03528 // name ファイル名
03529 // buffer 画像データ
03530 // width 画像の横の画素数
03531 // height 画像の縦の画素数
03532 // depth 画像の 1 画素のバイト数
03533 // 戻り値 保存に成功すれば true, 失敗すれば false
03534 //
03535 bool gg::ggSaveTga(
03536 const std::string& name,
03537 const void* buffer,
03538 unsigned int width,
03539 unsigned int height,
03540 unsigned int depth
03541 )
03542 {
03543 // ファイルを開く
03544 std::ofstream file{ Utf8ToTChar(name), std::ios::binary };
03545
03546 // ファイルが開けなかったら戻る
03547 if (file.fail()) return false;
03548
03549 // 画像のヘッダ
03550 const unsigned char type{ static_cast<unsigned char>(depth == 0 ? 0 : depth < 3 ? 3 : 2) };
03551 const unsigned char alpha{ static_cast<unsigned char>(depth == 2 || depth == 4 ? 8 : 0) };
03552 const unsigned char header[18]
03553 {
03554 0,           // ID length
03555 0,           // Color map type (none)
03556 type,        // Image Type (2:RGB, 3:Grayscale)
03557 0, 0,         // Offset into the color map table
03558 0, 0,         // Number of color map entries
03559 0,           // Number of a color map entry bits per pixel
03560 0, 0,         // Horizontal image position
03561 0, 0,         // Vertical image position
03562 static_cast<unsigned char>(width & 0xff),
03563 static_cast<unsigned char>(width >> 8),
03564 static_cast<unsigned char>(height & 0xff),
03565 static_cast<unsigned char>(height >> 8),
03566 static_cast<unsigned char>(depth * 8),
03567 alpha         // Image descriptor
03568 };
03569
03570 // ヘッダを書き込む
03571 file.write(reinterpret_cast<const char*>(header), sizeof(header));
03572
03573 // ヘッダの書き込みに失敗したら戻る
03574 if (file.bad())
03575 {
03576 file.close();
03577 return false;
03578 }
03579
03580 // データを書き込む
03581 const unsigned int size{ width * height * depth };
03582 if (type == 2)
03583 {
03584 // フルカラー
03585 std::vector<char> temp(size);

```

```

03586     for (GLuint i = 0; i < size; i += depth)
03587     {
03588         temp[i + 2] = static_cast<const char*>(buffer)[i + 0];
03589         temp[i + 1] = static_cast<const char*>(buffer)[i + 1];
03590         temp[i + 0] = static_cast<const char*>(buffer)[i + 2];
03591         if (depth == 4) temp[i + 3] = static_cast<const char*>(buffer)[i + 3];
03592     }
03593     file.write(temp.data(), size);
03594 }
03595 else if (type == 3)
03596 {
03597     // グレースケール
03598     file.write(static_cast<const char*>(buffer), size);
03599 }
03600
03601 // フッタを書き込む
03602 constexpr char footer[] = "\0\0\0\0\0\0\0\0TRUEVISION-XFILE.";
03603 file.write(footer, sizeof footer);
03604
03605 // データの書き込みに失敗したら戻る
03606 if (file.bad())
03607 {
03608     file.close();
03609     return false;
03610 }
03611
03612 // ファイルを閉じる
03613 file.close();
03614
03615 // データの書き込みに成功した
03616 return true;
03617 }
03618
03619 //
03620 // カラーバッファの内容を TGA ファイルに保存する
03621 //
03622 // name 保存するファイル名
03623 // 戻り値 保存に成功すれば true, 失敗すれば false
03624 //
03625 bool gg::ggSaveColor(const std::string& name)
03626 {
03627     // 現在のビューポートのサイズを得る
03628     GLint viewport[4];
03629     glGetIntegerv(GL_VIEWPORT, viewport);
03630
03631 // ビューポートのサイズ分のメモリを確保する
03632 std::vector<GLubyte> buffer(viewport[2] * viewport[3] * 3);
03633
03634 // 画面表示の完了を待つ
03635 glFinish();
03636
03637 // カラーバッファを読み込む
03638 glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03639     GL_RGB, GL_UNSIGNED_BYTE, buffer.data());
03640
03641 // 読み込んだデータをファイルに書き込む
03642 return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 3);
03643 }
03644
03645 //
03646 // デプスバッファの内容を TGA ファイルに保存する
03647 //
03648 // name 保存するファイル名
03649 // 戻り値 保存に成功すれば true, 失敗すれば false
03650 //
03651 bool gg::ggSaveDepth(const std::string& name)
03652 {
03653     // 現在のビューポートのサイズを得る
03654     GLint viewport[4];
03655     glGetIntegerv(GL_VIEWPORT, viewport);
03656
03657 // ビューポートのサイズ分のメモリを確保する
03658 std::vector<GLubyte> buffer(viewport[2] * viewport[3]);
03659
03660 // 画面表示の完了を待つ
03661 glFinish();
03662
03663 // デプスバッファを読み込む
03664 glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03665     GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, buffer.data());
03666
03667 // 読み込んだデータをファイルに書き込む
03668 return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 1);
03669 }
03670
03671 //
03672 // TGA ファイル (8/16/24/32bit) を読み込む

```

```

03673 //
03674 // name 読み込むファイル名
03675 // 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)
03676 // pHight 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)
03677 // 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)
03678 // image 読み込んだ画像を格納する vector
03679 // 戻り値 読み込みに成功すれば true, 失敗すれば false
03680 //
03681 bool gg::ggReadImage(
03682     const std::string& name,
03683     std::vector<GLubyte>& image,
03684     GLsizei* pWidth,
03685     GLsizei* pHight,
03686     GLenum* pFormat
03687 )
03688 {
03689     // ファイルを開く
03690     std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
03691
03692     // ファイルが開けなかったら戻る
03693     if (file.fail()) return false;
03694
03695     // ヘッダを読み込む
03696     unsigned char header[18];
03697     file.read(reinterpret_cast<char*>(header), sizeof(header));
03698
03699     // ヘッダの読み込みに失敗したら戻る
03700     if (file.bad())
03701     {
03702         file.close();
03703         return false;
03704     }
03705
03706     // 深度
03707     const auto depth{ header[16] / 8 };
03708     switch (depth)
03709     {
03710         case 1:
03711             *pFormat = GL_RED;
03712             break;
03713         case 2:
03714             *pFormat = GL_RG;
03715             break;
03716         case 3:
03717             *pFormat = GL_BGR;
03718             break;
03719         case 4:
03720             *pFormat = GL_BGRA;
03721             break;
03722         default:
03723             // 取り扱えないフォーマットだったら戻る
03724             file.close();
03725             return false;
03726     }
03727
03728     // 画像の縦横の画素数
03729     *pWidth = header[13] << 8 | header[12];
03730     *pHeight = header[15] << 8 | header[14];
03731
03732     // データサイズ
03733     const auto size{ *pWidth * *pHeight * depth };
03734     if (size < 2) return false;
03735
03736     // 読み込みに使うメモリを確保する
03737     image.resize(size);
03738
03739     // データを読み込む
03740     if (header[2] & 8)
03741     {
03742         // RLE
03743         int p{ 0 };
03744         char c;
03745         while (file.get(c))
03746         {
03747             if (c & 0x80)
03748             {
03749                 // run-length packet
03750                 const auto count{ (c & 0x7f) + 1 };
03751                 if (p + depth * count > size) break;
03752                 char temp[4];
03753                 file.read(temp, depth);
03754                 for (int i = 0; i < count; ++i)
03755                 {
03756                     for (int j = 0; j < depth; ) image[p++] = temp[j++];
03757                 }
03758             }
03759         }
03760     }

```

```

03760     {
03761         // raw packet
03762         const auto count{ (c + 1) * depth };
03763         if (p + count > size) break;
03764         file.read(reinterpret_cast<char*>(image.data() + p), count);
03765         p += count;
03766     }
03767 }
03769 else
03770 {
03771     // 非圧縮
03772     file.read(reinterpret_cast<char*>(image.data()), size);
03773 }
03774
03775 // 読み込みに失敗したら戻る
03776 if (file.bad())
03777 {
03778     file.close();
03779     return false;
03780 }
03781
03782 // ファイルを閉じる
03783 file.close();
03784
03785 // ファイルの読み込みに成功した
03786 return true;
03787 }
03788
03789 //
03790 // テクスチャを作成して画像を読み込む
03791 //
03792 //    image 画像データ, nullptr ならメモリの確保だけを行う
03793 //    width 画像の横の画素数
03794 //    height 画像の縦の画素数
03795 //    format 画像データのフォーマット
03796 //    type 画像のデータ型
03797 //    internal テクスチャの内部フォーマット
03798 //    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
03799 //    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
03800 //    戻り値 テクスチャ名
03801 //
03802 GLuint gg::ggLoadTexture(
03803     const GLvoid* image,
03804     GLsizei width,
03805     GLsizei height,
03806     GLenum format,
03807     GLenum type,
03808     GLenum internal,
03809     GLenum wrap,
03810     bool swizzle
03811 )
03812 {
03813     // テクスチャオブジェクト
03814     const auto tex{ [] { GLuint tex; glGenTextures(1, &tex); return tex; } () };
03815     glBindTexture(GL_TEXTURE_2D, tex);
03816
03817     // 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
03818     glPixelStorei(GL_UNPACK_ALIGNMENT, format == GL_RGBA ? 4 : 1);
03819
03820     // テクスチャを割り当てる
03821     glTexImage2D(GL_TEXTURE_2D, 0, internal, width, height, 0, format, type, image);
03822
03823     // パイリニア（ミップマップなし）, エッジでクランプ
03824     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
03825     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
03826     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap);
03827     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);
03828
03829     if (swizzle)
03830     {
03831         // テクスチャのサンプリング時に赤と青を入れ替える
03832         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, GL_BLUE);
03833         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, GL_RED);
03834     }
03835
03836     // テクスチャ名を返す
03837     return tex;
03838 }
03839
03840 //
03841 // 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
03842 //
03843 //    name TGA ファイル名
03844 //    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)
03845 //    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)
03846 //    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。

```

```

03847 // 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
03848 // 戻り値 テクスチャ名
03849 //
03850 GLuint gg::ggLoadImage(
03851     const std::string& name,
03852     GLsizei* pWidth,
03853     GLsizei* pHeight,
03854     GLenum internal,
03855     GLenum wrap
03856 )
03857 {
03858     // 画像データ
03859     std::vector<GLubyte> image;
03860
03861     // 画像サイズ
03862     GLsizei width, height;
03863
03864     // 画像フォーマット
03865     GLenum format;
03866
03867     // 画像を読み込む
03868     ggReadImage(name, image, &width, &height, &format);
03869
03870     // 画像が読み込めなかったら戻る
03871     if (image.empty()) return 0;
03872
03873     // 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
03874     if (internal == 0) internal = format;
03875
03876     // テクスチャに読み込む
03877     const auto tex{ ggLoadTexture(image.data(), width, height,
03878         format, GL_UNSIGNED_BYTE, internal, wrap, true) };
03879
03880     // 画像サイズを返す
03881     if (pWidth) *pWidth = width;
03882     if (pHeight) *pHeight = height;
03883
03884     // テクスチャ名を返す
03885     return tex;
03886 }
03887
03888 //
03889 // 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
03890 //
03891 // width 高さマップのグレースケール画像 hmap の横の画素数
03892 // height 高さマップのグレースケール画像のデータ hmap の縦の画素数
03893 // stride データの間隔
03894 // hmap グレースケール画像のデータ
03895 // nz 法線の z 成分の割合
03896 // internal テクスチャの内部フォーマット
03897 // nmap 法線マップを格納する vector
03898 //
03899 void gg::ggCreateNormalMap(
03900     const GLubyte* hmap,
03901     GLsizei width,
03902     GLsizei height,
03903     GLenum format,
03904     GLfloat nz,
03905     GLenum internal,
03906     std::vector<GGVector>& nmap
03907 )
03908 {
03909     // メモリサイズ
03910     const GLsizei size{ width * height };
03911
03912     // 法線マップのメモリを確保する
03913     nmap.resize(size);
03914
03915     // 画素のバイト数
03916     GLint stride;
03917     switch (format)
03918     {
03919         case GL_RED:
03920             stride = 1;
03921             break;
03922         case GL_RGB:
03923             stride = 2;
03924             break;
03925         case GL_RGBA:
03926             stride = 3;
03927             break;
03928         case GL_RGBA:
03929             stride = 4;
03930             break;
03931         default:
03932             stride = 1;
03933             break;

```

```

03934 }
03935
03936 // 法線マップの作成
03937 for (GLsizei i = 0; i < size; ++i)
03938 {
03939     const auto x{ i % width };
03940     const auto y{ i - x };
03941     const auto u0{ (y + (x - 1 + width) % width) * stride };
03942     const auto u1{ (y + (x + 1) % width) * stride };
03943     const auto v0{ ((y - width + size) % size + x) * stride };
03944     const auto v1{ ((y + width) % size + x) * stride };
03945
03946     // 隣接する画素との値の差を法線の成分に用いる
03947     nmap[i][0] = static_cast<GLfloat>(hmap[u1] - hmap[u0]);
03948     nmap[i][1] = static_cast<GLfloat>(hmap[v1] - hmap[v0]);
03949     nmap[i][2] = nz;
03950     nmap[i][3] = hmap[i * stride];
03951
03952     // 法線ベクトルを正規化する
03953     ggNormalize3(nmap[i]);
03954 }
03955
03956 // 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
03957 if (
03958     internal != GL_RGB16F &&
03959     internal != GL_RGBA16F &&
03960     internal != GL_RGB32F &&
03961     internal != GL_RGBA32F
03962 )
03963 {
03964     for (GLsizei i = 0; i < size; ++i)
03965     {
03966         nmap[i][0] = nmap[i][0] * 0.5f + 0.5f;
03967         nmap[i][1] = nmap[i][1] * 0.5f + 0.5f;
03968         nmap[i][2] = nmap[i][2] * 0.5f + 0.5f;
03969         nmap[i][3] *= 0.0039215686f; // == 1/255
03970     }
03971 }
03972 }
03973
03974 //
03975 // 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
03976 //
03977 // name TGA ファイル名
03978 // nz 作成した法線の z 成分の割合
03979 // 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)
03980 // 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)
03981 // internal テクスチャの内部フォーマット
03982 // 戻り値 テクスチャ名
03983 //
03984 GLuint gg::ggLoadHeight(
03985     const std::string& name,
03986     GLfloat nz,
03987     GLsizei* pWidth,
03988     GLsizei* pHeight,
03989     GLenum internal
03990 )
03991 {
03992     // 画像データ
03993     std::vector<GLubyte> hmap;
03994
03995     // 画像サイズ
03996     GLsizei width, height;
03997
03998     // 画像フォーマット
03999     GLenum format;
04000
04001     // 高さマップの画像を読み込む
04002     ggReadImage(name, hmap, &width, &height, &format);
04003
04004     // 画像が読み込めなかつたら戻る
04005     if (hmap.empty()) return 0;
04006
04007     // 法線マップ
04008     std::vector<GgVector> nmap;
04009
04010     // 法線マップを作成する
04011     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
04012
04013     // 画像サイズを返す
04014     if (pWidth) *pWidth = width;
04015     if (pHeight) *pHeight = height;
04016
04017     // テクスチャを作成して返す
04018     return ggLoadTexture(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal, GL_REPEAT);
04019 }
04020

```

```

04021 //
04022 // テクスチャの赤と青を交換する
04023 //
04024 void gg::GgTexture::swapRandB(bool swap) const
04025 {
04026     const auto swizzle
04027     {
04028         swap ?
04029             std::pair<GLint, GLint>{ GL_BLUE, GL_RED } :
04030             std::pair<GLint, GLint>{ GL_RED, GL_BLUE }
04031     };
04032
04033     glBindTexture(GL_TEXTURE_2D, texture);
04034     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, swizzle.first);
04035     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, swizzle.second);
04036     glBindTexture(GL_TEXTURE_2D, 0);
04037 }
04038
04039 //
04040 // 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
04041 //
04042 //   name 読み込むファイル名
04043 //   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
04044 //   戻り値 テクスチャの作成に成功すれば true, 失敗すれば false
04045 //
04046 void gg::GgColorTexture::load(
04047     const std::string& name,
04048     GLenum internal,
04049     GLenum wrap
04050 )
04051 {
04052     // 画像データ
04053     std::vector<GLubyte> image;
04054
04055     // 画像サイズ
04056     GLsizei width, height;
04057
04058     // 画像フォーマット
04059     GLenum format;
04060
04061     // 画像を読み込む
04062     ggReadImage(name, image, &width, &height, &format);
04063
04064     // 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
04065     if (internal == 0) internal = format;
04066
04067     // テクスチャを作成する
04068     texture = std::make_shared<GgTexture>(image.data(), width, height,
04069         format, GL_UNSIGNED_BYTE, internal, wrap, true);
04070 }
04071
04072 //
04073 // 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
04074 //
04075 //   name 画像ファイル名
04076 //   width テクスチャとして用いる画像データの横幅
04077 //   height テクスチャとして用いる画像データの高さ
04078 //   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)
04079 //   nz 法線マップの z 成分の値
04080 //   internal テクスチャの内部フォーマット
04081 //
04082 void gg::GgNormalTexture::load(
04083     const std::string& name,
04084     GLfloat nz,
04085     GLenum internal
04086 )
04087 {
04088     // 高さマップ
04089     std::vector<GLubyte> hmap;
04090
04091     // 画像サイズ
04092     GLsizei width, height;
04093
04094     // 画像フォーマット
04095     GLenum format;
04096
04097     // 高さマップの画像を読み込む
04098     ggReadImage(name, hmap, &width, &height, &format);
04099
04100     // 法線マップ
04101     std::vector<GgVector> nmap;
04102
04103     // 法線マップを作成する
04104     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
04105 }
04106
04108

```

```

04109 // 
04110 // OBJ ファイルの読み込みに使うデータ型と関数
04111 //
04112 namespace gg
04113 {
04114     // GLfloat 型の 2 要素のベクトル
04115     using vec2 = std::array<GLfloat, 2>;
04116
04117     // GLfloat 型の 3 要素のベクトル
04118     using vec3 = std::array<GLfloat, 3>;
04119
04120     // 三角形データ
04121     struct fidx
04122     {
04123         GLuint p[3];           // 頂点座標番号
04124         GLuint n[3];           // 頂点法線番号
04125         GLuint t[3];           // テクスチャ座標番号
04126         bool smooth;          // スムーズシェーディングの有無
04127     };
04128
04129     // ポリゴン グループ
04130     struct fgrp
04131     {
04132         GLuint nextgroup;    // 次のポリゴン グループの最初の三角形番号
04133         GLuint mtlno;        // このポリゴン グループの材質番号
04134
04135         // コンストラクタ
04136         fgrp(GLuint nextgroup, GLuint mtlno) :
04137             nextgroup(nextgroup),
04138             mtlno(mtlno)
04139     {
04140     }
04141 };
04142
04143     // デフォルトの材質
04144     constexpr GgSimpleShader::Material defaultMaterial
04145     {
04146         { 0.1f, 0.1f, 0.1f, 1.0f },
04147         { 0.6f, 0.6f, 0.6f, 0.0f },
04148         { 0.3f, 0.3f, 0.3f, 0.0f },
04149         60.0f
04150     };
04151
04152     // デフォルトの材質名
04153     constexpr char defaultMaterialName[] = "_default_";
04154
04155     //
04156     // Alias OBJ 形式の MTL ファイルを読み込む
04157     //
04158     // mtlpath MTL ファイルのパス名
04159     // mtl 読み込んだ材質名をキー、材質番号を値にした map
04160     // material 材質データ
04161     //
04162     static bool ggLoadMtl(const std::string& mtlpath,
04163         std::map<std::string, GLuint>& mtl,
04164         std::vector<GgSimpleShader::Material>& material)
04165     {
04166         // MTL ファイルがなければ戻る
04167         std::ifstream mtlfile{ Utf8ToTChar(mtlpath), std::ios::binary };
04168         if (!mtlfile)
04169         {
04170             #if defined(DEBUG)
04171                 std::cerr << "Warning: Can't open MTL file: " << mtlpath << std::endl;
04172             #endif
04173             return false;
04174         }
04175
04176         // 一行読み込み用のバッファ
04177         std::string mtlline;
04178
04179         // 材質名 (ループの外に置く)
04180         std::string mtlname{ defaultMaterialName };
04181
04182         // 現在の材質番号を登録する
04183         mtl[mtlname] = static_cast<GLuint>(material.size());
04184
04185         // 現在の材質にデフォルトの材質を設定する
04186         material.emplace_back(defaultMaterial);
04187
04188         // 材質データを読み込む
04189         while (std::getline(mtlfile, mtlline))
04190         {
04191             // 空行は読み飛ばす
04192             if (mtlline == "") continue;
04193
04194             // 最後の文字が '\r' なら
04195             if ((*mtlline.end() - 1) == '\r')

```

```

04196 {
04197     // 最後の文字を削除する
04198     mtlline.erase(mtlline.end() - 1, mtlline.end());
04199
04200     // 空行になったら読み飛ばす
04201     if (mtlline == "") continue;
04202 }
04203
04204     // 読み込んだ行を文字列ストリームにする
04205     std::istringstream mtlstr{ mtlline };
04206
04207     // オペレータ
04208     std::string mtlop;
04209
04210     // 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
04211     mtlstr >> mtlop;
04212
04213     // '#' で始まる場合はコメントとして行末まで読み飛ばす
04214     if (mtlop[0] == '#') continue;
04215
04216     if (mtlop == "newmtl")
04217     {
04218         // 新規作成する材質名を取り出す
04219         mtlstr >> mtlname;
04220
04221         // 材質名が既に存在するかどうか調べる
04222         const auto m{ mt1.find(mtlname) };
04223         if (m == mt1.end())
04224         {
04225             // 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
04226             mt1[mtlname] = static_cast<GLuint>(material.size());
04227
04228             // 新規作成する材質にデフォルトの材質を設定しておく
04229             material.emplace_back(defaultMaterial);
04230         }
04231
04232 #if defined(DEBUG)
04233     std::cerr << "newmtl: " << mtlname << std::endl;
04234 #endif
04235 }
04236 else if (mtlop == "Ka")
04237 {
04238     // 環境光の反射係数を登録する
04239     mtlstr
04240         >> material.back().ambient[0]
04241         >> material.back().ambient[1]
04242         >> material.back().ambient[2];
04243 }
04244 else if (mtlop == "Kd")
04245 {
04246     // 拡散反射係数を登録する
04247     mtlstr
04248         >> material.back().diffuse[0]
04249         >> material.back().diffuse[1]
04250         >> material.back().diffuse[2];
04251 }
04252 else if (mtlop == "Ks")
04253 {
04254     // 鏡面反射係数を登録する
04255     mtlstr
04256         >> material.back().specular[0]
04257         >> material.back().specular[1]
04258         >> material.back().specular[2];
04259 }
04260 else if (mtlop == "Ns")
04261 {
04262     // 輝き係数を登録する
04263     GLfloat shininess;
04264     mtlstr >> shininess;
04265     material.back().shininess = shininess * 0.1f;
04266 }
04267 else if (mtlop == "d")
04268 {
04269     // 不透明度を登録する
04270     mtlstr >> material.back().ambient[3];
04271 }
04272 }
04273
04274     // MTL ファイルの読み込みに失敗したら戻る
04275     if (mtlfile.bad())
04276     {
04277 #if defined(DEBUG)
04278     std::cerr << "Warning: Can't read MTL file: " << mtlpath << std::endl;
04279 #endif
04280
04281     // MTL ファイルを閉じる
04282     mtlfile.close();

```

```
04283     // MTL ファイルが読み込みに失敗したので戻る
04284     return false;
04285 }
04286
04287 // MTL ファイルを閉じる
04288 mtlfile.close();
04289
04290 // MTL ファイルの読み込みに成功した
04291 return true;
04292 }
04293
04294 //
04295 // Alias OBJ 形式のファイルを解析する
04296 //
04297 //
04298 //   name Alias OBJ 形式のファイル名
04299 //   group 同じ材質を割り当てるポリゴングループ
04300 //   mtl 読み込んだ材質名をキーにした map
04301 //   pos 頂点の位置
04302 //   norm 頂点の法線
04303 //   tex 頂点のテクスチャ座標
04304 //   face 三角形のデータ
04305 //
04306 static bool ggParseObj(
04307     const std::string& name,
04308     std::vector<fgrp>& group,
04309     std::vector<GgSimpleShader::Material>& material,
04310     std::vector<vec3>& pos,
04311     std::vector<vec3>& norm,
04312     std::vector<vec2>& tex,
04313     std::vector<fidx>& face,
04314     bool normalize
04315 )
04316 {
04317     // ファイルパスからディレクトリ名を取り出す
04318     const std::string path{ name };
04319     const size_t base{ path.find_last_of("//\\") };
04320     const std::string dirname{ (base == std::string::npos) ? "" : path.substr(0, base + 1) };
04321
04322     // OBJ ファイルを読み込む
04323     std::ifstream file{ Utf8ToTChar(path) };
04324
04325     // 読み込みに失敗したら戻る
04326     if (file.fail())
04327     {
04328 #if defined(DEBUG)
04329         std::cerr << "Error: Can't open OBJ file: " << path << std::endl;
04330 #endif
04331         return false;
04332     }
04333
04334     // ポリゴングループの最初の三角形番号
04335     GLsizei startgroup(static_cast<GLsizei>(group.size()));
04336
04337     // スムーズシェーディングのスイッチ
04338     bool smooth{ false };
04339
04340     // 材質のテーブル
04341     std::map<std::string, GLuint> mtl;
04342
04343     // 現在の材質名（ループの外で宣言する）
04344     std::string mtlname;
04345
04346     // 座標値の最小値・最大値
04347     vec3 bmin{ FLT_MAX }, bmax{ -FLT_MAX };
04348
04349     // 一行読み込み用のバッファ
04350     std::string line;
04351
04352     // データの読み込み
04353     while (std::getline(file, line))
04354     {
04355         // 空行は読み飛ばす
04356         if (line == "") continue;
04357
04358         // 最後の文字が '\r' なら
04359         if (*(line.end() - 1) == '\r')
04360         {
04361             // 最後の文字を削除する
04362             line.erase(line.end() - 1, line.end());
04363
04364             // 空行になったら読み飛ばす
04365             if (line == "") continue;
04366         }
04367
04368         // 一行を文字列ストリームに入れる
04369         std::istringstream str(line);
```

```

04370
04371 // 最初のトークンを命令 (op) とみなす
04372 std::string op;
04373 str >> op;
04374
04375 if (op[0] == '#') continue;
04376
04377 if (op == "v")
{
    // 頂点位置
    vec3 v;

    // 頂点位置はスペースで区切られている
    str >> v[0] >> v[1] >> v[2];
}

// 頂点位置を記録する
pos.emplace_back(v);

// 頂点位置の最小値と最大値を求める (AABB)
for (int i = 0; i < 3; ++i)
{
    bmin[i] = std::min(bmin[i], v[i]);
    bmax[i] = std::max(bmax[i], v[i]);
}

else if (op == "vt")
{
    // テクスチャ座標
    vec2 t;

    // 頂点位置はスペースで区切られている
    str >> t[0] >> t[1];

    // テクスチャ座標を記録する
    tex.emplace_back(t);
}

else if (op == "vn")
{
    // 頂点法線
    vec3 n;

    // 頂点法線はスペースで区切られている
    str >> n[0] >> n[1] >> n[2];

    // 頂点法線を記録する
    norm.emplace_back(n);
}

else if (op == "f")
{
    // 三角形データ
    fidx f;

    // スムースシェーディング
    f.smooth = smooth;

    // 三頂点のそれぞれについて
    for (int i = 0; i < 3; ++i)
    {
        // 1項目取り出す
        std::string s;
        str >> s;

        // 文字の位置
        auto c{ s.begin() };

        // 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
        f.p[i] = f.t[i] = f.n[i] = 0;

        // 項目の最初の要素は頂点座標番号
        while (c != s.end() && isdigit(*c)) f.p[i] = f.p[i] * 10 + *c++ - '0';
        if (c == s.end() || *c++ != '/') continue;

        // 二つ目の項目はテクスチャ座標
        while (c != s.end() && isdigit(*c)) f.t[i] = f.t[i] * 10 + *c++ - '0';
        if (c == s.end() || *c++ != '/') continue;

        // 三つ目の項目は法線番号
        while (c != s.end() && isdigit(*c)) f.n[i] = f.n[i] * 10 + *c++ - '0';
    }

    // 三角形データを登録する
    face.emplace_back(f);
}

else if (op == "s")
{
    // '1' だったらスムースシェーディング有効
    std::string s;
}

```

```
04457         str >> s;
04458         smooth = s == "1";
04459     }
04460     else if (op == "usemtl")
04461     {
04462         // 次のポリゴングループの最初の三角形番号
04463         const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04464
04465         // ポリゴングループに三角形が存在すれば
04466         if (nextgroup > startgroup)
04467         {
04468             // ポリゴングループの三角形数と材質番号を記録する
04469             group.emplace_back(nextgroup, mtl[mtlname]);
04470
04471             // 次のポリゴングループの開始番号を保存しておく
04472             startgroup = nextgroup;
04473         }
04474
04475         // 次に usemtl が来るまで材質名を保持する
04476         str >> mtlname;
04477
04478         // 材質の存在チェック
04479         if (mtl.find(mtlname) == mtl.end())
04480         {
04481 #if defined(DEBUG)
04482             std::cerr << "Warning: Undefined material: " << mtlname << std::endl;
04483 #endif
04484
04485             // デフォルトの材質を割り当てておく
04486             mtlname = defaultMaterialName;
04487         }
04488 #if defined(DEBUG)
04489         else std::cerr << "usemtl: " << mtlname << std::endl;
04490 #endif
04491     }
04492     else if (op == "mtllib")
04493     {
04494         // MTL ファイルのバス名を作る
04495         str >> std::ws;
04496         std::string mtllibpath;
04497         std::getline(str, mtllibpath);
04498
04499         // MTL ファイルを読み込む
04500         ggLoadMtl(dirname + mtllibpath, mtl, material);
04501     }
04502 }
04503
04504     // OBJ ファイルの読み込みに失敗したら戻る
04505     if (file.bad())
04506     {
04507 #if defined(DEBUG)
04508         std::cerr << "Error: Can't read OBJ file: " << path << std::endl;
04509 #endif
04510         file.close();
04511         return false;
04512     }
04513
04514     // ファイルを閉じる
04515     file.close();
04516
04517     // 最後のポリゴングループの次の三角形番号
04518     const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04519     if (nextgroup > startgroup)
04520     {
04521         // 最後のポリゴングループの三角形数と材質を記録する
04522         group.emplace_back(nextgroup, static_cast<GLuint>(mtl[mtlname]));
04523     }
04524
04525     // 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
04526     for (auto& f : face)
04527     {
04528         if (!f.smooth)
04529         {
04530             // 三頂点のそれぞれについて
04531             for (int i = 0; i < 3; ++i)
04532             {
04533                 // 新しい頂点座標を生成する (std::array の要素は emplace_back できない)
04534                 pos.emplace_back(pos[f.p[i] - 1]);
04535                 f.p[i] = static_cast<GLuint>(pos.size());
04536
04537                 if (f.t[i] > 0)
04538                 {
04539                     // 新しいテクスチャ座標を生成する
04540                     tex.emplace_back(tex[f.t[i] - 1]);
04541                     f.t[i] = static_cast<GLuint>(tex.size());
04542                 }
04543         }
```

```

04544     if (f.n[i] > 0)
04545     {
04546         // 新しい法線を生成する
04547         norm.emplace_back(norm[f.n[i] - 1]);
04548         f.n[i] = static_cast<GLuint>(norm.size());
04549     }
04550 }
04551 }
04552 }
04553
04554 // 法線データがなければ算出しておく
04555 if (norm.empty())
04556 {
04557     // 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
04558     norm.resize(pos.size(), { 0.0f, 0.0f, 0.0f });
04559
04560     // 面の法線の算出と頂点法線の算出
04561     for (auto& f : face)
04562     {
04563         // 頂点座標番号
04564         const auto v0{ f.p[0] - 1 };
04565         const auto v1{ f.p[1] - 1 };
04566         const auto v2{ f.p[2] - 1 };
04567
04568         // v1 - v0, v2 - v0 を求める
04569         const GLfloat d1[] { pos[v1][0] - pos[v0][0], pos[v1][1] - pos[v0][1], pos[v1][2] - pos[v0][2] };
04570         const GLfloat d2[] { pos[v2][0] - pos[v0][0], pos[v2][1] - pos[v0][1], pos[v2][2] - pos[v0][2] };
04571
04572         // 外積により面法線を求める
04573         vec3 n;
04574         ggCross(n.data(), d1, d2);
04575
04576         if (f.smooth)
04577         {
04578             // スムースシェーディングを行うときは
04579             for (int i = 0; i < 3; ++i)
04580             {
04581                 // 面法線を頂点法線に積算する
04582                 norm[v0][i] += n[i];
04583                 norm[v1][i] += n[i];
04584                 norm[v2][i] += n[i];
04585
04586                 // 面の各頂点の法線番号は頂点番号と同じにする
04587                 f.n[i] = f.p[i];
04588             }
04589         }
04590         else
04591         {
04592             // 面法線を最初の頂点に保存する
04593             norm[v0] = n;
04594             f.n[0] = f.p[0];
04595
04596             // 2 頂点追加
04597             for (int i = 1; i < 3; ++i)
04598             {
04599                 norm.emplace_back(n);
04600                 f.n[i] = static_cast<GLuint>(norm.size());
04601             }
04602         }
04603     }
04604
04605     // 頂点の法線ベクトルを正規化する
04606     for (auto& n : norm) ggNormalize3(n.data());
04607 }
04608
04609 // 図形の正規化
04610 if (normalize)
04611 {
04612     // 図形の大きさ
04613     const auto sx{ bmax[0] - bmin[0] };
04614     const auto sy{ bmax[1] - bmin[1] };
04615     const auto sz{ bmax[2] - bmin[2] };
04616
04617     // 図形のスケール
04618     GLfloat s{ sx };
04619     if (sy > s) s = sy;
04620     if (sz > s) s = sz;
04621     const auto scale{ s != 0.0f ? 2.0f / s : 1.0f };
04622
04623     // 図形の中心位置
04624     const auto cx{ (bmax[0] + bmin[0]) * 0.5f };
04625     const auto cy{ (bmax[1] + bmin[1]) * 0.5f };
04626     const auto cz{ (bmax[2] + bmin[2]) * 0.5f };
04627
04628     // 図形の大きさと位置を正規化する

```

```

04629     for (auto& p : pos)
04630     {
04631         p[0] = (p[0] - cx) * scale;
04632         p[1] = (p[1] - cy) * scale;
04633         p[2] = (p[2] - cz) * scale;
04634     }
04635 }
04636
04637 #if defined(DEBUG)
04638     std::cerr
04639     << "[" << name << "]\nParsed Group: " << group.size() << ", Material: " << mtl.size()
04640     << ", Pos: " << pos.size() << ", Norm: " << norm.size() << ", Tex: " << tex.size()
04641     << ", Face: " << face.size() << "\n";
04642 #endif
04643
04644     // OBJ ファイルの読み込み成功
04645     return true;
04646 }
04647 }
04649
04650 //
04651 // 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)
04652 //
04653 // name 読み込むOBJ ファイル名
04654 // 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
04655 // material 読み込んだデータのポリゴン グループごとの材質
04656 // vert 読み込んだデータの頂点属性
04657 // normalize true ならサイズを正規化する
04658 // 戻り値 読み込みに成功したら true
04659 //
04660 bool gg::ggLoadSimpleObj(const std::string& name,
04661     std::vector<std::array<GLuint, 3>>& group,
04662     std::vector<GgSimpleShader::Material>& material,
04663     std::vector<GgVertex>& vert,
04664     bool normalize)
04665 {
04666     // 読み込み用の一時記憶領域
04667     std::vector<fgrp> tgroup;
04668     std::vector<vec3> tpos;
04669     std::vector<vec3> tnorm;
04670     std::vector<vec2> ttex;
04671     std::vector<fidx> tface;
04672
04673     // OBJ ファイルを解析する
04674     if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04675
04676     // 頂点属性データのメモリを確保する
04677     vert.reserve(vert.size() + tface.size() * 3);
04678
04679     // ポリゴン グループデータのメモリを確保する
04680     group.reserve(group.size() + tgroup.size());
04681     material.reserve(material.size() + tgroup.size());
04682
04683     // ポリゴン グループの最初の三角形番号
04684     GLuint startgroup{ 0 };
04685
04686     // ポリゴン グループデータの作成
04687     for (auto& g : tgroup)
04688     {
04689         // 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
04690         std::array<GLuint, 3> v;
04691
04692         // このポリゴン グループの最初の頂点番号を保存する
04693         v[0] = static_cast<GLuint>(vert.size());
04694
04695         // 三角形ごとの頂点データの作成
04696         for (GLuint j = startgroup; j < g.nextgroup; ++j)
04697         {
04698             // 処理対象の三角形
04699             auto& f = tface[j];
04700
04701             // 三頂点のそれぞれについて
04702             for (int i = 0; i < 3; ++i)
04703             {
04704                 // テクスチャ座標
04705                 vec2 tex{ 0.0f, 0.0f };
04706                 if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04707
04708                 // 頂点法線
04709                 vec3 norm{ 0.0f, 0.0f, 0.0f };
04710                 if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04711
04712                 // 頂点属性の追加
04713                 vert.emplace_back(tpos[f.p[i] - 1].data(), norm.data());
04714             }
04715         }
04716

```

```

04717 // このポリゴングループの頂点数を保存する
04718 v[1] = static_cast<GLuint>(vert.size()) - v[0];
04719 v[2] = g.mtlno;
04720
04721 // 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
04722 group.emplace_back(v);
04723
04724 // 次のポリゴングループの最初の三角形番号を求める
04725 startgroup = g.nextgroup;
04726 }
04727
04728 #if defined(DEBUG)
04729 std::cerr
04730 << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04731 << ", Vertex: " << vert.size() << "\n";
04732 #endif
04733
04734 // OBJ ファイルの読み込み成功
04735 return true;
04736 }
04737
04738 //
04739 // 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)
04740 //
04741 // name 読み込むOBJ ファイル名
04742 // 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
04743 // material 読み込んだデータのポリゴングループごとの材質
04744 // vert 読み込んだデータの頂点属性
04745 // face 読み込んだデータの三角形の頂点インデックス
04746 // normalize true ならサイズを正規化する
04747 // 戻り値 読み込みに成功したら true
04748 //
04749 bool gg::ggLoadSimpleObj(const std::string& name,
04750 std::vector<std::array<GLuint, 3>>& group,
04751 std::vector<GgSimpleShader::Material>& material,
04752 std::vector<GgVertex>& vert,
04753 std::vector<GLuint>& face,
04754 bool normalize)
04755 {
04756 // 読み込み用の一時記憶領域
04757 std::vector<fgrp> tgroup;
04758 std::vector<vec3> tpos;
04759 std::vector<vec3> tnorm;
04760 std::vector<vec2> ttex;
04761 std::vector<fidx> tface;
04762
04763 // OBJ ファイルを解析する
04764 if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04765
04766 // 頂点属性データの最初の頂点番号
04767 const auto vertbase{ static_cast<GLuint>(vert.size()) };
04768
04769 // 頂点属性データのメモリを確保する
04770 vert.resize(vertbase + tpos.size());
04771
04772 // 三角形データのメモリを確保する
04773 face.reserve(face.size() + tface.size());
04774
04775 // ポリゴングループデータのメモリを確保する
04776 group.reserve(group.size() + tgroup.size());
04777 material.reserve(material.size() + tgroup.size());
04778
04779 // ポリゴングループの最初の三角形番号
04780 GLuint startgroup{ 0 };
04781
04782 // ポリゴングループデータの作成
04783 for (auto& g : tgroup)
04784 {
04785 // このポリゴングループの最初の頂点番号
04786 const auto first{ static_cast<GLuint>(face.size()) };
04787
04788 // 三角形ごとの頂点データの作成
04789 for (GLuint j = startgroup; j < g.nextgroup; ++j)
04790 {
04791 // 処理対象の三角形
04792 auto& f{ tface[j] };
04793
04794 // 三頂点のそれぞれについて
04795 for (int i = 0; i < 3; ++i)
04796 {
04797 // 追加する三角形データの頂点番号
04798 const auto q{ f.p[i] - 1 + vertbase };
04799
04800 // 三角形データの追加
04801 face.emplace_back(q);
04802
04803 // テクスチャ座標番号

```

```

04804     vec2 tex{ 0.0f, 0.0f };
04805     if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04806
04807     // 頂点法線番号
04808     vec3 norm{ 0.0f, 0.0f, 0.0f };
04809     if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04810
04811     // 頂点の格納
04812     vert[q] = GgVertex(tpos[f.p[i] - 1].data(), norm.data());
04813 }
04814 }
04815
04816     // 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
04817     group.emplace_back(std::array<GLuint, 3>{ first, static_cast<GLuint>(face.size()) - first, g.mtlno });
04818 }
04819     // 次のポリゴングループの最初の三角形番号を求める
04820     startgroup = g.nextgroup;
04821 }
04822
04823 #if defined(DEBUG)
04824     std::cerr
04825         << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04826         << ", Vertex: " << vert.size() << ", Face: " << face.size() << "\n";
04827 #endif
04828
04829     // OBJ ファイルの読み込み成功
04830     return true;
04831 }
04832
04833 //
04834 // シェーダオブジェクトのコンパイル結果を表示する
04835 //
04836 static GLboolean printShaderInfoLog(GLuint shader, const std::string& str)
04837 {
04838     // コンパイル結果を取得する
04839     GLint status;
04840     glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
04841 #if defined(DEBUG)
04842     if (status == GL_FALSE) std::cerr << "Compile Error in " << str << std::endl;
04843 #endif
04844
04845     // シェーダのコンパイル時のログの長さを取得する
04846     GLsizei bufSize;
04847     glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &bufSize);
04848
04849     if (bufSize > 1)
04850     {
04851         // シェーダのコンパイル時のログの内容を取得する
04852         std::vector<GLchar> infoLog(bufSize);
04853         GLsizei length;
04854         glGetShaderInfoLog(shader, bufSize, &length, infoLog.data());
04855 #if defined(DEBUG)
04856         std::cerr << infoLog.data();
04857 #endif
04858     }
04859
04860     // コンパイル結果を返す
04861     return static_cast<GLboolean>(status);
04862 }
04863
04864 //
04865 // プログラムオブジェクトのリンク結果を表示する
04866 //
04867 static GLboolean printProgramInfoLog(GLuint program)
04868 {
04869     // リンク結果を取得する
04870     GLint status;
04871     glGetProgramiv(program, GL_LINK_STATUS, &status);
04872 #if defined(DEBUG)
04873     if (status == GL_FALSE) std::cerr << "Link Error." << std::endl;
04874 #endif
04875
04876     // シェーダのリンク時のログの長さを取得する
04877     GLsizei bufSize;
04878     glGetProgramiv(program, GL_INFO_LOG_LENGTH, &bufSize);
04879
04880     // シェーダのリンク時のログの内容を取得する
04881     if (bufSize > 1)
04882     {
04883         std::vector<GLchar> infoLog(bufSize);
04884         GLsizei length;
04885         glGetProgramInfoLog(program, bufSize, &length, infoLog.data());
04886 #if defined(DEBUG)
04887         std::cerr << infoLog.data() << std::endl;
04888 #endif
04889     }

```

```

04890
04891 // リンク結果を返す
04892     return static_cast<GLboolean>(status);
04893 }
04894
04895 //
04896 // 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
04897 //
04898 //    vsrc パーテックスシェーダのソースプログラムの文字列
04899 //    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)
04900 //    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)
04901 //    nvarying フィードバックする varying 変数の数 (0 なら不使用)
04902 //    varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04903 //    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
04904 //    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
04905 //    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
04906 //    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)
04907 //
04908 GLuint gg::ggCreateShader(
04909     const std::string& vsrc,
04910     const std::string& fsrc,
04911     const std::string& gsrc,
04912     GLint nvarying,
04913     const char* const* varyings,
04914     const std::string& vtext,
04915     const std::string& ftext,
04916     const std::string& gtext)
04917 {
04918 // シェーダプログラムの作成
04919     const auto program{ glCreateProgram() };
04920
04921     if (program > 0)
04922     {
04923         bool status = true;
04924
04925         if (!vsrc.empty())
04926         {
04927             // 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
04928             const auto vertShader{ glCreateShader(GL_VERTEX_SHADER) };
04929             const auto* vsrcp{ vsrc.c_str() };
04930             glShaderSource(vertShader, 1, &vsrcp, nullptr);
04931             glCompileShader(vertShader);
04932
04933             // 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
04934             if (printShaderInfoLog(vertShader, vtext))
04935                 glAttachShader(program, vertShader);
04936             else
04937                 status = false;
04938             glDeleteShader(vertShader);
04939         }
04940
04941         if (!fsrc.empty())
04942         {
04943             // 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
04944             const auto fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
04945             const auto* fsrcp{ fsrc.c_str() };
04946             glShaderSource(fragShader, 1, &fsrcp, nullptr);
04947             glCompileShader(fragShader);
04948
04949             // 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
04950             if (printShaderInfoLog(fragShader, ftext))
04951                 glAttachShader(program, fragShader);
04952             else
04953                 status = false;
04954             glDeleteShader(fragShader);
04955         }
04956
04957         if (!gsrc.empty())
04958         {
04959 #if defined(GL_GLES_PROTOTYPES)
04960 # if defined(DEBUG)
04961             std::cerr << gtext << ": The geometry shader is not supported." << std::endl;
04962             status = false;
04963 # endif
04964 #else
04965             // 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
04966             const auto geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
04967             const auto* gsrcp{ gsrc.c_str() };
04968             glShaderSource(geomShader, 1, &gsrcp, nullptr);
04969             glCompileShader(geomShader);
04970
04971             // 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
04972             if (printShaderInfoLog(geomShader, gtext))
04973                 glAttachShader(program, geomShader);
04974             else
04975                 status = false;
04976             glDeleteShader(geomShader);

```

```
04977 #endif
04978 }
04979
04980 // 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
04981 if (status)
04982 {
04983 // feedback に使う varying 変数を指定する
04984 if (nvarying > 0)
04985 glTransformFeedbackVaryings(program, nvarying, varyings, GL_SEPARATE_ATTRIBS);
04986
04987 // シェーダプログラムをリンクする
04988 glLinkProgram(program);
04989
04990 // リンクに成功したらプログラムオブジェクト名を返す
04991 if (printProgramInfoLog(program) != GL_FALSE) return program;
04992 }
04993 }
04994
04995 // プログラムオブジェクトが作成できなかった
04996 glDeleteProgram(program);
04997 return 0;
04998 }
04999
05000 //
05001 // シェーダのソースファイルを読み込んだ vector を返す
05002 //
05003 // name ソースファイル名
05004 // src 読み込んだソースファイルの文字列
05005 // 戻り値 読み込みの成功すれば true, 失敗したら false
05006 //
05007 static bool readShaderSource(const std::string& name, std::string& src)
05008 {
05009 // ファイル名が nullptr ならそのまま戻る
05010 if (name.empty()) return true;
05011
05012 // ソースファイルを開く
05013 std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
05014 if (file.fail())
05015 {
05016 // ファイルが開けなければエラーで戻る
05017 #if defined(DEBUG)
05018 std::cerr << "Error: Can't open source file: " << name << std::endl;
05019 #endif
05020 return false;
05021 }
05022
05023 // ファイル全体を文字列として読み込む
05024 std::istreambuf_iterator<char> it{ file };
05025 std::istreambuf_iterator<char> last;
05026 src = std::string(it, last);
05027
05028 // ファイルがうまく読み込めなければ戻る
05029 if (file.bad())
05030 {
05031 #if defined(DEBUG)
05032 std::cerr << "Error: Could not read souce file: " << name << std::endl;
05033 #endif
05034 file.close();
05035 return false;
05036 }
05037
05038 // ファイルを閉じて戻る
05039 file.close();
05040 return true;
05041 }
05042
05043 //
05044 // 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
05045 //
05046 // vert パーテックスシェーダのソースファイル名
05047 // 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)
05048 // 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)
05049 // nvarying フィードバックする varying 変数の数 (0 なら不使用)
05050 // varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
05051 // 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)
05052 //
05053 GLuint gg::ggLoadShader(
05054 const std::string& vert,
05055 const std::string& frag,
05056 const std::string& geom,
05057 GLint nvarying,
05058 const char* const* varyings
05059 )
05060 {
05061 // 読み込んだシェーダのソースプログラム
05062 std::string vsrc, fsrc, gsrc;
05063
```

```

05064 // 読み込んだ結果
05065 bool status;
05066
05067 // シェーダのソースファイルを読み込む
05068 status = readShaderSource(vert, vsrc);
05069 status = readShaderSource(frag, fsrc) && status;
05070 status = readShaderSource(geom, gsrc) && status;
05071
05072 // 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
05073 if (!status) return 0;
05074
05075 // プログラムオブジェクトを作成する
05076 return ggCreateShader(vsrc, fsrc, gsrc, nvarying, varyings, vert, frag, geom);
05077 }
05078
05079 #if !defined(__APPLE__) && !defined(GL_GLES_PROTOTYPES)
05080 //
05081 // 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
05082 //
05083 // csrc コンピュートシェーダのソースプログラムの文字列
05084 // 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)
05085 //
05086 GLuint gg::ggCreateComputeShader(const std::string& csrc, const std::string& ctext)
05087 {
05088 // シェーダプログラムの作成
05089 const auto program{ glCreateProgram() };
05090
05091 if (program > 0)
05092 {
05093 // ソースプログラムの文字列が空だったら 0 を返す
05094 if (csrc.empty()) return 0;
05095
05096 // 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
05097 const auto compShader{ glCreateShader(GL_COMPUTE_SHADER) };
05098 const auto* csrcp{ csrc.c_str() };
05099 glShaderSource(compShader, 1, &csrcp, nullptr);
05100 glCompileShader(compShader);
05101
05102 // 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
05103 if (printShaderInfoLog(compShader, ctext)) glAttachShader(program, compShader);
05104 glDeleteShader(compShader);
05105
05106 // シェーダプログラムをリンクする
05107 glLinkProgram(program);
05108
05109 // プログラムオブジェクトが作成できなければ 0 を返す
05110 if (printProgramInfoLog(program) == GL_FALSE)
05111 {
05112 glDeleteProgram(program);
05113 return 0;
05114 }
05115 }
05116
05117 // プログラムオブジェクトを返す
05118 return program;
05119 }
05120
05121 //
05122 // 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
05123 //
05124 // comp コンピュートシェーダのソースファイル名
05125 // 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)
05126 //
05127 GLuint gg::ggLoadComputeShader(const std::string& comp)
05128 {
05129 // シェーダのソースファイルを読み込む
05130 std::string csrc;
05131 if (readShaderSource(comp, csrc))
05132 {
05133 // プログラムオブジェクトを作成する
05134 return ggCreateComputeShader(csrc.data(), comp);
05135 }
05136
05137 // プログラムオブジェクト作成失敗
05138 return 0;
05139 }
05140 #endif
05141
05142 //
05143 // 点: データ作成
05144 //
05145 void gg::GgPoints::load(const GgVector* pos, GLsizei count, GLenum usage)
05146 {
05147 // 頂点バッファオブジェクトを作成する
05148 position = std::make_shared<GgBuffer<GgVector>>(GL_ARRAY_BUFFER, pos,
static_cast<GLsizei>(sizeof(GgVector)), count, usage);
05149

```

```

05150 // 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
05151 glVertexAttribPointer(0, static_cast<GLint>(pos->size()), GL_FLOAT, GL_FALSE, 0, 0);
05152 glEnableVertexAttribArray(0);
05153 }
05154
05155 //
05156 // 点：描画
05157 //
05158 void gg::GgPoints::draw(GLint first, GLsizei count) const
05159 {
05160     // 頂点配列オブジェクトを指定する
05161     GgShape::draw(first, count);
05162
05163     // 図形を描画する
05164     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05165 }
05166
05167 //
05168 // 三角形：データ作成
05169 //
05170 void gg::GgTriangles::load(const GgVertex* vert, GLsizei count, GLenum usage)
05171 {
05172     // 頂点バッファオブジェクトを作成する
05173     vertex = std::make_shared<GgBuffer<GgVertex>>(GL_ARRAY_BUFFER, vert,
05174         static_cast<GLsizei>(sizeof(GgVertex)), count, usage);
05175
05176     // 頂点の位置は index == 0 の in 変数から入力する
05177     glVertexAttribPointer(0, static_cast<GLint>(vert->position.size()), GL_FLOAT, GL_FALSE,
05178         sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, position));
05179     glEnableVertexAttribArray(0);
05180
05181     // 頂点の法線は index == 1 の in 変数から入力する
05182     glVertexAttribPointer(1, static_cast<GLint>(vert->normal.size()), GL_FLOAT, GL_FALSE,
05183         sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, normal));
05184     glEnableVertexAttribArray(1);
05185
05186 //
05187 // 三角形：描画
05188 //
05189 void gg::GgTriangles::draw(GLint first, GLsizei count) const
05190 {
05191     // 頂点配列オブジェクトを指定する
05192     GgShape::draw(first, count);
05193
05194     // 図形を描画する
05195     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05196 }
05197
05198 //
05199 // オブジェクト：描画
05200 //
05201 void gg::GgElements::draw(GLint first, GLsizei count) const
05202 {
05203     // 頂点配列オブジェクトを指定する
05204     GgShape::draw(first, count);
05205
05206     // 図形を描画する
05207     glDrawElements(getMode(), count > 0 ? count : getIndexCount() - first,
05208         GL_UNSIGNED_INT, static_cast<GLuint*>(0) + first);
05209 }
05210
05211 //
05212 // 点群を立方体状に生成する
05213 //
05214 std::shared_ptr<gg::GgPoints> gg::ggPointsCube(GLsizei count, GLfloat length, GLfloat cx, GLfloat cy,
05215     GLfloat cz)
05216 {
05217     // メモリを確保する
05218     std::vector<GgVector> pos(count);
05219
05220     // 点を生成する
05221     for (GLsizei v = 0; v < count; ++v)
05222     {
05223         const GgVector p
05224         {
05225             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cx,
05226             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cy,
05227             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cz,
05228             1.0f
05229         };
05230         pos.emplace_back(p);
05231     }
05232
05233     // 点データの GgPoints オブジェクトを作成して返す
05234     return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);

```

```

05235 }
05236
05237 // 点群を球状に生成する
05238 // 05239 //
05240 std::shared_ptr<gg::GgPoints> gg::ggPointsSphere(GLsizei count, GLfloat radius,
05241   GLfloat cx, GLfloat cy, GLfloat cz)
05242 {
05243   // メモリを確保する
05244   std::vector<GgVector> pos(count);
05245
05246   // 点を生成する
05247   for (GLsizei v = 0; v < count; ++v)
05248   {
05249     const auto r{ radius * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) };
05250     const auto t{ 6.28318530718f * static_cast<GLfloat>(rand()) / (static_cast<GLfloat>(RAND_MAX) +
05251       1.0f) };
05252     const auto cp{ 2.0f * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 1.0f };
05253     const auto sp{ sqrtf(1.0f - cp * cp) };
05254     const auto ct{ cosf(t) };
05255     const auto st{ sinf(t) };
05256     const GgVector p{ r * sp * ct + cx, r * sp * st + cy, r * cp + cz, 1.0f };
05257
05258     pos.emplace_back(p);
05259   }
05260   // 点データの GgPoints オブジェクトを作成して返す
05261   return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05262 }
05263
05264 //
05265 // 矩形状に 2 枚の三角形を生成する
05266 //
05267 std::shared_ptr<gg::GgTriangles> gg::ggRectangle(GLfloat width, GLfloat height)
05268 {
05269   // 頂点属性
05270   std::array<gg::GgVertex, 4> vert;
05271
05272   // 頂点位置と法線を求める
05273   for (int v = 0; v < 4; ++v)
05274   {
05275     const auto x{ ((v & 1) * 2 - 1) * width };
05276     const auto y{ ((v & 2) - 1) * height };
05277
05278     vert[v] = gg::GgVertex{ x, y, 0.0f, 0.0f, 0.0f, 1.0f };
05279   }
05280
05281   // 矩形の GgTriangles オブジェクトを作成する
05282   return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05283     GL_TRIANGLE_STRIP);
05284
05285 //
05286 // 楕円状に三角形を生成する
05287 //
05288 std::shared_ptr<gg::GgTriangles> gg::ggEllipse(GLfloat width, GLfloat height, GLuint slices)
05289 {
05290   // 楕円のスケール
05291   constexpr GLfloat scale{ 0.5f };
05292
05293   // 作業用のメモリ
05294   std::vector<gg::GgVertex> vert;
05295
05296   // 頂点位置と法線を求める
05297   for (GLuint v = 0; v < slices; ++v)
05298   {
05299     const auto t{ 6.28318530717f * static_cast<GLfloat>(v) / static_cast<GLfloat>(slices) };
05300     const auto x{ cos(t) * width * scale };
05301     const auto y{ sin(t) * height * scale };
05302
05303     vert.emplace_back(x, y, 0.0f, 0.0f, 0.0f, 1.0f);
05304   }
05305
05306   // GgTriangles オブジェクトを作成する
05307   return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05308     GL_TRIANGLE_FAN);
05309
05310 //
05311 // Wavefront OBJ ファイルを読み込む (Arrays 形式)
05312 //
05313 std::shared_ptr<gg::GgTriangles> gg::ggArraysObj(const std::string& name, bool normalize)
05314 {
05315   std::vector<std::array<GLuint, 3>> group;
05316   std::vector<gg::GgSimpleShader::Material> material;
05317   std::vector<gg::GgVertex> vert;
05318

```

```
05319 // ファイルを読み込む
05320 if (!ggLoadSimpleObj(name, group, material, vert, normalize)) return nullptr;
05321
05322 // GgTriangles オブジェクトを作成する
05323 return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLES);
05324 }
05325
05326 //
05327 // Wavefront OBJ ファイル を読み込む (Elements 形式)
05328 //
05329 std::shared_ptr<gg::GgElements> gg::ggElementsObj(const std::string& name, bool normalize)
05330 {
05331     std::vector<std::array<GLuint, 3>> group;
05332     std::vector<gg::GgSimpleShader::Material> material;
05333     std::vector<gg::GgVertex> vert;
05334     std::vector<GLuint> face;
05335
05336 // ファイルを読み込む
05337 if (!ggLoadSimpleObj(name, group, material, vert, face, normalize)) return nullptr;
05338
05339 // GgElements オブジェクトを作成する
05340 return std::make_shared<gg::GgElements>(vert.data(), static_cast<GLsizei>(vert.size()), face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05341 }
05342 }
05343
05344 //
05345 // メッシュ形状を作成する (Elements 形式)
05346 //
05347 std::shared_ptr<gg::GgElements> gg::ggElementsMesh(GLuint slices, GLuint stacks, const
05348 GLfloat (*pos)[3], const GLfloat (*norm)[3])
05349 {
05350     // 頂点属性
05351     std::vector<gg::GgVertex> vert((slices + 1) * (stacks + 1));
05352
05353     // 頂点の法線を求める
05354     for (GLuint j = 0; j <= stacks; ++j)
05355     {
05356         for (GLuint i = 0; i <= slices; ++i)
05357         {
05358             // 処理対象の頂点番号
05359             const auto k{ j * (slices + 1) + i };
05360
05361             // 頂点の法線
05362             gg::GgVector tnorm;
05363             tnorm[3] = 0.0f;
05364
05365             if (norm)
05366             {
05367                 tnorm[0] = norm[k][0];
05368                 tnorm[1] = norm[k][1];
05369                 tnorm[2] = norm[k][2];
05370             }
05371             else
05372             {
05373                 // 処理対象の頂点の周囲の頂点番号
05374                 const auto kim{ i > 0 ? k - 1 : k };
05375                 const auto kip{ i < slices ? k + 1 : k };
05376                 const auto kjm{ j > 0 ? k - slices - 1 : k };
05377                 const auto kjp{ j < stacks ? k + slices + 1 : k };
05378
05379                 // 接線ベクトル
05380                 const std::array<GLfloat, 3> t
05381                 {
05382                     pos[kip][0] - pos[kim][0],
05383                     pos[kip][1] - pos[kim][1],
05384                     pos[kip][2] - pos[kim][2]
05385                 };
05386
05387                 // 従接線ベクトル
05388                 const std::array<GLfloat, 3> b
05389                 {
05390                     pos[kjp][0] - pos[kjm][0],
05391                     pos[kjp][1] - pos[kjm][1],
05392                     pos[kjp][2] - pos[kjm][2]
05393                 };
05394
05395                 // 法線
05396                 tnorm[0] = t[1] * b[2] - t[2] * b[1];
05397                 tnorm[1] = t[2] * b[0] - t[0] * b[2];
05398                 tnorm[2] = t[0] * b[1] - t[1] * b[0];
05399
05400                 // 法線の正規化
05401                 ggNormalize3(tnorm);
05402             }
05403             // 頂点の位置
```

```

05404     const gg::GgVector tpos{ pos[k][0], pos[k][1], pos[k][2], 1.0f };
05405
05406     // 頂点属性の保存
05407     vert.emplace_back(tpos, tnorm);
05408 }
05409
05410
05411 // 頂点のインデックス (三角形データ)
05412 std::vector<GLuint> face;
05413
05414 // 頂点のインデックスを求める
05415 for (GLuint j = 0; j < stacks; ++j)
05416 {
05417     for (GLuint i = 0; i < slices; ++i)
05418     {
05419         // 処理対象のマス
05420         const auto k{ (slices + 1) * j + i };
05421
05422         // マスの上半分の三角形
05423         face.emplace_back(k);
05424         face.emplace_back(k + slices + 2);
05425         face.emplace_back(k + 1);
05426
05427         // マスのお下半分の三角形
05428         face.emplace_back(k);
05429         face.emplace_back(k + slices + 1);
05430         face.emplace_back(k + slices + 2);
05431     }
05432 }
05433
05434 // GgElements オブジェクトを作成する
05435 return std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05436     face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05437 }
05438
05439 //
05440 // 球状に三角形データを生成する (Elements 形式)
05441 //
05442 std::shared_ptr<gg::GgElements> gg::ggElementsSphere(GLfloat radius, int slices, int stacks)
05443 {
05444     // 頂点の位置と法線
05445     std::vector<GLfloat> p, n;
05446
05447     // 頂点の位置と法線を求める
05448     for (int j = 0; j <= stacks; ++j)
05449     {
05450         const auto t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };
05451         const auto ph{ 3.1415926536f * t };
05452         const auto y{ cos(ph) };
05453         const auto r{ sin(ph) };
05454
05455         for (int i = 0; i <= slices; ++i)
05456         {
05457             const auto s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };
05458             const auto th{ -2.0f * 3.1415926536f * s };
05459             const auto x{ r * cos(th) };
05460             const auto z{ r * sin(th) };
05461
05462             // 頂点の座標値
05463             p.emplace_back(x * radius);
05464             p.emplace_back(y * radius);
05465             p.emplace_back(z * radius);
05466
05467             // 頂点の法線
05468             n.emplace_back(x);
05469             n.emplace_back(y);
05470             n.emplace_back(z);
05471         }
05472     }
05473
05474     // GgElements オブジェクトを作成する
05475     return ggElementsMesh(slices, stacks, reinterpret_cast<GLfloat(*)[3]>(p.data()),
05476         reinterpret_cast<GLfloat(*)[3]>(n.data()));
05477 }
05478
05479 //
05480 // 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
05481 //
05482 // r 光源の強度の環境光成分の赤成分
05483 // g 光源の強度の環境光成分の緑成分
05484 // b 光源の強度の環境光成分の青成分
05485 // a 光源の強度の環境光成分の不透明度, デフォルトは 1
05486 // 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
05487 // count 値を設定する光源データの数, デフォルトは 1
05488 //
05489 void gg::GgSimpleShader::LightBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05490     GLint first, GLsizei count) const

```

```

05491 {
05492     // 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
05493     const auto start{ static_cast<char*>(map(first, count)) };
05494     for (GLsizei i = 0; i < count; ++i)
05495     {
05496         // バッファオブジェクトの i 番目のブロックのポインタ
05497         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05498
05499         // 光源の環境光成分を設定する
05500         light->ambient[0] = r;
05501         light->ambient[1] = g;
05502         light->ambient[2] = b;
05503         light->ambient[3] = a;
05504     }
05505     unmap();
05506 }
05507
05508 //
05509 // 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
05510 //
05511 //    ambient 光源の強度の環境光成分
05512 //    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
05513 //    count 値を設定する光源データの数, デフォルトは 1
05514 //
05515 void gg::GgSimpleShader::LightBuffer::loadAmbient(const GgVector& ambient,
05516     GLint first, GLsizei count) const
05517 {
05518     // 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
05519     const auto start{ static_cast<char*>(map(first, count)) };
05520     for (GLsizei i = 0; i < count; ++i)
05521     {
05522         // バッファオブジェクトの i 番目のブロックのポインタ
05523         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05524
05525         // 光源の強度の環境光成分を設定する
05526         light->ambient = ambient;
05527     }
05528     unmap();
05529 }
05530
05531 //
05532 // 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
05533 //
05534 //    r 光源の強度の拡散反射光成分の赤成分
05535 //    g 光源の強度の拡散反射光成分の緑成分
05536 //    b 光源の強度の拡散反射光成分の青成分
05537 //    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
05538 //    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
05539 //    count 値を設定する光源データの数, デフォルトは 1
05540 //
05541 void gg::GgSimpleShader::LightBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05542     GLint first, GLsizei count) const
05543 {
05544     // 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
05545     const auto start{ static_cast<char*>(map(first, count)) };
05546     for (GLsizei i = 0; i < count; ++i)
05547     {
05548         // バッファオブジェクトの i 番目のブロックのポインタ
05549         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05550
05551         // 光源の拡散反射光成分を設定する
05552         light->diffuse[0] = r;
05553         light->diffuse[1] = g;
05554         light->diffuse[2] = b;
05555         light->diffuse[3] = a;
05556     }
05557     unmap();
05558 }
05559
05560 //
05561 // 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
05562 //
05563 //    ambient 光源の強度の拡散反射光成分
05564 //    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
05565 //    count 値を設定する光源データの数, デフォルトは 1
05566 //
05567 void gg::GgSimpleShader::LightBuffer::loadDiffuse(const GgVector& diffuse,
05568     GLint first, GLsizei count) const
05569 {
05570     // 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
05571     const auto start{ static_cast<char*>(map(first, count)) };
05572     for (GLsizei i = 0; i < count; ++i)
05573     {
05574         // バッファオブジェクトの i 番目のブロックのポインタ
05575         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05576
05577         // 光源の強度の拡散反射光成分を設定する

```

```

05578     light->diffuse = diffuse;
05579 }
05580 unmap();
05581 }
05582
05583 //
05584 // 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
05585 //
05586 // r 光源の強度の鏡面反射光成分の赤成分
05587 // g 光源の強度の鏡面反射光成分の緑成分
05588 // b 光源の強度の鏡面反射光成分の青成分
05589 // 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
05590 // 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
05591 // count 値を設定する光源データの数，デフォルトは 1
05592 //
05593 void gg::GgSimpleShader::LightBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05594 GLint first, GLsizei count) const
05595 {
05596 // 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
05597 const auto start{ static_cast<char*>(map(first, count)) };
05598 for (GLsizei i = 0; i < count; ++i)
05599 {
05600 // バッファオブジェクトの i 番目のブロックのポインタ
05601 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05602
05603 // 光源の鏡面反射光成分を設定する
05604 light->specular[0] = r;
05605 light->specular[1] = g;
05606 light->specular[2] = b;
05607 light->specular[3] = a;
05608 }
05609 unmap();
05610 }
05611
05612 //
05613 // 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
05614 //
05615 // ambient 光源の強度の鏡面反射光成分
05616 // 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
05617 // count 値を設定する光源データの数，デフォルトは 1
05618 //
05619 void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& specular,
05620 GLint first, GLsizei count) const
05621 {
05622 // 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
05623 const auto start{ static_cast<char*>(map(first, count)) };
05624 for (GLsizei i = 0; i < count; ++i)
05625 {
05626 // バッファオブジェクトの i 番目のブロックのポインタ
05627 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05628
05629 // 光源の強度の鏡面反射光成分を設定する
05630 light->specular = specular;
05631 }
05632 unmap();
05633 }
05634
05635 //
05636 // 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
05637 //
05638 // material 光源の特性の GgSimpleShader::Light 構造体
05639 // 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
05640 // count 値を設定する光源データの数，デフォルトは 1
05641 //
05642 void gg::GgSimpleShader::LightBuffer::loadColor(const Light& color,
05643 GLint first, GLsizei count) const
05644 {
05645 // 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
05646 const auto start{ static_cast<char*>(map(first, count)) };
05647 for (GLsizei i = 0; i < count; ++i)
05648 {
05649 // バッファオブジェクトの i 番目のブロックのポインタ
05650 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05651
05652 // 光源の色を設定する
05653 light->ambient = color.ambient;
05654 light->diffuse = color.diffuse;
05655 light->specular = color.specular;
05656 }
05657 unmap();
05658 }
05659
05660 //
05661 // 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
05662 //
05663 // x 光源の位置の x 座標
05664 // y 光源の位置の y 座標

```

```

05665 // z 光源の位置の z 座標
05666 // w 光源の位置の w 座標, デフォルトは 1
05667 // 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
05668 // count 値を設定する光源データの数, デフォルトは 1
05669 //
05670 void gg::GgSimpleShader::LightBuffer::loadPosition(GLfloat x, GLfloat y, GLfloat z, GLfloat w,
05671 GLint first, GLsizei count) const
05672 {
05673 // 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
05674 const auto start{ static_cast<char*>(map(first, count)) };
05675 for (GLsizei i = 0; i < count; ++i)
05676 {
05677 // バッファオブジェクトの i 番目のブロックのポインタ
05678 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05679
05680 // 光源の位置を設定する
05681 light->position[0] = x;
05682 light->position[1] = y;
05683 light->position[2] = z;
05684 light->position[3] = w;
05685 }
05686 unmap();
05687 }
05688
05689 //
05690 // 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
05691 //
05692 // position 光源の位置
05693 // 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
05694 // count 値を設定する光源データの数, デフォルトは 1
05695 //
05696 void gg::GgSimpleShader::LightBuffer::loadPosition(const GgVector& position,
05697 GLint first, GLsizei count) const
05698 {
05699 // 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
05700 const auto start{ static_cast<char*>(map(first, count)) };
05701 for (GLsizei i = 0; i < count; ++i)
05702 {
05703 // バッファオブジェクトの i 番目のブロックのポインタ
05704 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05705
05706 // 光源の位置を設定する
05707 light->position = position;
05708 }
05709 unmap();
05710 }
05711
05712 //
05713 // 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
05714 //
05715 // r 環境光に対する反射係数の赤成分
05716 // g 環境光に対する反射係数の緑成分
05717 // b 環境光に対する反射係数の青成分
05718 // a 環境光に対する反射係数の不透明度, デフォルトは 1
05719 // 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
05720 // count 値を設定する材質データの数, デフォルトは 1
05721 //
05722 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05723 GLint first, GLsizei count) const
05724 {
05725 // 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
05726 const auto start{ static_cast<char*>(map(first, count)) };
05727 for (GLsizei i = 0; i < count; ++i)
05728 {
05729 // バッファオブジェクトの i 番目のブロックのポインタ
05730 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05731
05732 // 環境光に対する反射係数を設定する
05733 material->ambient[0] = r;
05734 material->ambient[1] = g;
05735 material->ambient[2] = b;
05736 material->ambient[3] = a;
05737 }
05738 unmap();
05739 }
05740
05741 //
05742 // 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
05743 //
05744 // ambient 環境光に対する反射係数
05745 // 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
05746 // count 値を設定する光源データの数, デフォルトは 1
05747 //
05748 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(const GgVector& ambient,
05749 GLint first, GLsizei count) const
05750 {
05751 // 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

```

```

05752 const auto start{ static_cast<char*>(map(first, count)) };
05753 for (GLsizei i = 0; i < count; ++i)
05754 {
05755     // バッファオブジェクトの i 番目のブロックのポインタ
05756     const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05757
05758     // 光源の強度の環境光成分を設定する
05759     material->ambient = ambient;
05760 }
05761 unmap();
05762 }
05763
05764 //
05765 // 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
05766 //
05767 //    r 拡散反射係数の赤成分
05768 //    g 拡散反射係数の緑成分
05769 //    b 拡散反射係数の青成分
05770 //    a 拡散反射係数の不透明度、デフォルトは 1
05771 //    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
05772 //    count 値を設定する材質データの数、デフォルトは 1
05773 //
05774 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, 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     const auto 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         const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05783
05784         // 拡散反射係数を設定する
05785         material->diffuse[0] = r;
05786         material->diffuse[1] = g;
05787         material->diffuse[2] = b;
05788         material->diffuse[3] = a;
05789     }
05790     unmap();
05791 }
05792
05793 //
05794 // 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
05795 //
05796 //    diffuse 拡散反射係数
05797 //    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
05798 //    count 値を設定する光源データの数、デフォルトは 1
05799 //
05800 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(const GgVector& diffuse,
05801 GLint first, GLsizei count) const
05802 {
05803     // 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
05804     const auto start{ static_cast<char*>(map(first, count)) };
05805     for (GLsizei i = 0; i < count; ++i)
05806     {
05807         // バッファオブジェクトの i 番目のブロックのポインタ
05808         const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05809
05810         // 光源の強度の環境光成分を設定する
05811         material->diffuse = diffuse;
05812     }
05813     unmap();
05814 }
05815
05816 //
05817 // 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
05818 //
05819 //    r 環境光に対する反射係数と拡散反射係数の赤成分
05820 //    g 環境光に対する反射係数と拡散反射係数の緑成分
05821 //    b 環境光に対する反射係数と拡散反射係数の青成分
05822 //    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
05823 //    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
05824 //    count 値を設定する材質データの数、デフォルトは 1
05825 //
05826 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(GLfloat r, GLfloat g, GLfloat b,
05827 GLfloat a,
05828 GLint first, GLsizei count) const
05829 {
05830     // 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
05831     const auto start{ static_cast<char*>(map(first, count)) };
05832     for (GLsizei i = 0; i < count; ++i)
05833     {
05834         // バッファオブジェクトの i 番目のブロックのポインタ
05835         const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05836
05837         // 環境光に対する反射係数と拡散反射係数を設定する
05838         material->ambient[0] = material->diffuse[0] = r;

```

```
05838     material->ambient[1] = material->diffuse[1] = g;
05839     material->ambient[2] = material->diffuse[2] = b;
05840     material->ambient[3] = material->diffuse[3] = a;
05841 }
05842 unmapped();
05843 }
05844
05845 //
05846 // 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
05847 //
05848 //   color 環境光に対する反射係数と拡散反射係数
05849 //   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
05850 //   count 値を設定する光源データの数、デフォルトは 1
05851 //
05852 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GgVector& color,
05853     GLint first, GLsizei count) const
05854 {
05855     // 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
05856     const auto start{ static_cast<char*>(map(first, count)) };
05857     for (GLsizei i = 0; i < count; ++i)
05858     {
05859         // バッファオブジェクトの i 番目のブロックのポインタ
05860         const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05861
05862         // 光源の強度の環境光成分を設定する
05863         material->ambient = material->diffuse = color;
05864     }
05865     unmapped();
05866 }
05867
05868 //
05869 // 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
05870 //
05871 //   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
05872 //   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
05873 //   count 値を設定する材質データの数、デフォルトは 1
05874 //
05875 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GLfloat* color,
05876     GLint first, GLsizei count) const
05877 {
05878     // ambient 要素のバイトオフセット
05879     constexpr auto ambientOffset{ offsetof(Material, ambient) };
05880
05881     // ambient 要素のバイト数
05882     constexpr auto ambientSize{ sizeof(Material::diffuse) };
05883
05884     // diffuse 要素のバイトオフセット
05885     constexpr auto diffuseOffset{ offsetof(Material, diffuse) };
05886
05887     // diffuse 要素のバイト数
05888     constexpr auto diffuseSize{ sizeof(Material::diffuse) };
05889
05890     // 元のデータの先頭
05891     const auto* source{ reinterpret_cast<const char*>(color) };
05892
05893     // 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
05894     bind();
05895     for (GLsizei i = 0; i < count; ++i)
05896     {
05897         // 格納先
05898         const auto destination{ getStride() * (first + i) };
05899
05900         // first + i 番目のブロックの ambient 要素に値を設定する
05901         glBufferSubData(getTarget(), destination + ambientOffset, ambientSize, source + i * ambientSize);
05902
05903         // first + i 番目のブロックの diffuse 要素に値を設定する
05904         glBufferSubData(getTarget(), destination + diffuseOffset, diffuseSize, source + i * diffuseSize);
05905     }
05906 }
05907
05908 //
05909 // 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
05910 //
05911 //   r 鏡面反射係数の赤成分
05912 //   g 鏡面反射係数の緑成分
05913 //   b 鏡面反射係数の青成分
05914 //   a 鏡面反射係数の不透明度、デフォルトは 1
05915 //   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
05916 //   count 値を設定する材質データの数、デフォルトは 1
05917 //
05918 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05919     GLint first, GLsizei count) const
05920 {
05921     // 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
05922     const auto start{ static_cast<char*>(map(first, count)) };
05923     for (GLsizei i = 0; i < count; ++i)
05924     {
```

```

05925 // バッファオブジェクトの i 番目のブロックのポインタ
05926 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05927
05928 // 鏡面反射係数を設定する
05929 material->specular[0] = r;
05930 material->specular[1] = g;
05931 material->specular[2] = b;
05932 material->specular[3] = a;
05933 }
05934 unmap();
05935 }
05936
05937 //
05938 // 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
05939 //
05940 // specular 鏡面反射係数
05941 // 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
05942 // count 値を設定する光源データの数, デフォルトは 1
05943 //
05944 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(const GgVector& specular,
05945 GLint first, GLsizei count) const
05946 {
05947 // 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
05948 const auto start{ static_cast<char*>(map(first, count)) };
05949 for (GLsizei i = 0; i < count; ++i)
05950 {
05951 // バッファオブジェクトの i 番目のブロックのポインタ
05952 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05953
05954 // 光源の強度の環境光成分を設定する
05955 material->specular = specular;
05956 }
05957 unmap();
05958 }
05959
05960 //
05961 // 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
05962 //
05963 // shininess 輝き係数
05964 // 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
05965 // count 値を設定する材質データの数, デフォルトは 1
05966 //
05967 void gg::GgSimpleShader::MaterialBuffer::loadShininess(GLfloat shininess,
05968 GLint first, GLsizei count) const
05969 {
05970 // 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
05971 const auto start{ static_cast<char*>(map(first, count)) };
05972 for (GLsizei i = 0; i < count; ++i)
05973 {
05974 // バッファオブジェクトの i 番目のブロックのポインタ
05975 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05976 material->shininess = shininess;
05977 }
05978 unmap();
05979 }
05980
05981 //
05982 // 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
05983 //
05984 // shininess 輝き係数
05985 // 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
05986 // count 値を設定する材質データの数, デフォルトは 1
05987 //
05988 void gg::GgSimpleShader::MaterialBuffer::loadShininess(const GLfloat* shininess,
05989 GLint first, GLsizei count) const
05990 {
05991 // 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
05992 const auto start{ static_cast<char*>(map(first, count)) };
05993 for (GLsizei i = 0; i < count; ++i)
05994 {
05995 // バッファオブジェクトの i 番目のブロックのポインタ
05996 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05997 material->shininess = shininess[i];
05998 }
05999 unmap();
06000 }
06001
06002 //
06003 // 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
06004 //
06005 bool gg::GgSimpleShader::load(const std::string& vert, const std::string& frag, const std::string&
geom,
06006 GLint nvarying, const char* const* varyings)
06007 {
06008 if (!GgPointShader::load(vert, frag, geom, nvarying, varyings)) return false;
06009 mnLoc = glGetUniformLocation(get(), "mn");

```

```

06011    lightIndex = glGetUniformBlockIndex(get(), "Light");
06012    glUniformBlockBinding(get(), lightIndex, 0);
06013    materialIndex = glGetUniformBlockIndex(get(), "Material");
06014    glUniformBlockBinding(get(), materialIndex, 1);
06015
06016    return true;
06017 }
06018
06019 //
06020 // Wavefront OBJ 形式のデータ：コンストラクタ
06021 //
06022 gg::GgSimpleObj::GgSimpleObj(const std::string& name, bool normalize)
06023 {
06024     // 作業用のメモリ
06025     std::vector<GgSimpleShader::Material> mat;
06026     std::vector<GgVertex> vert;
06027     std::vector<GLuint> face;
06028
06029     // グループのデータのメモリを確保する
06030     group = std::make_shared<std::vector<std::array<GLuint, 3>>>();
06031
06032     // ファイルを読み込む
06033     if (ggLoadSimpleObj(name, *group, mat, vert, face, normalize))
06034     {
06035         // 頂点バッファオブジェクトを作成する
06036         data = std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
06037             face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
06038
06039         // 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
06040         glBindVertexArray(0);
06041
06042         // 材質データを設定する
06043         material = std::make_shared<GgSimpleShader::MaterialBuffer>(mat.data(),
06044             static_cast<GLsizei>(mat.size()));
06045     }
06046
06047 //
06048 // Wavefront OBJ 形式のデータ：図形の描画
06049 //
06050 void gg::GgSimpleObj::draw(GLint first, GLsizei count) const
06051 {
06052     // 保持しているグループの数
06053     const auto ng{ static_cast<GLsizei>(group->size()) };
06054
06055     // 描画する最後のグループの次
06056     auto last{ count <= 0 ? ng : first + count };
06057     if (last > ng) last = ng;
06058
06059     for (GLsizei i = first; i < last; ++i)
06060     {
06061         // グループのデータ
06062         const auto& g{ (*group)[i] };
06063
06064         // 材質を設定する
06065         material->select(g[2]);
06066
06067         // 図形を描画する
06068         data->draw(g[0], g[1]);
06069     }
06070 }

```

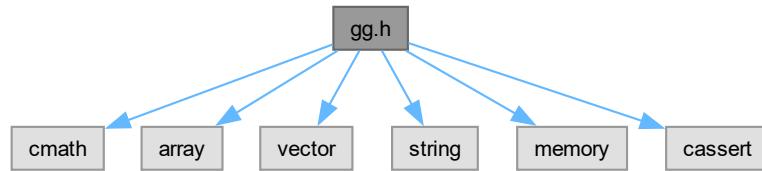
9.11 gg.h ファイル

```

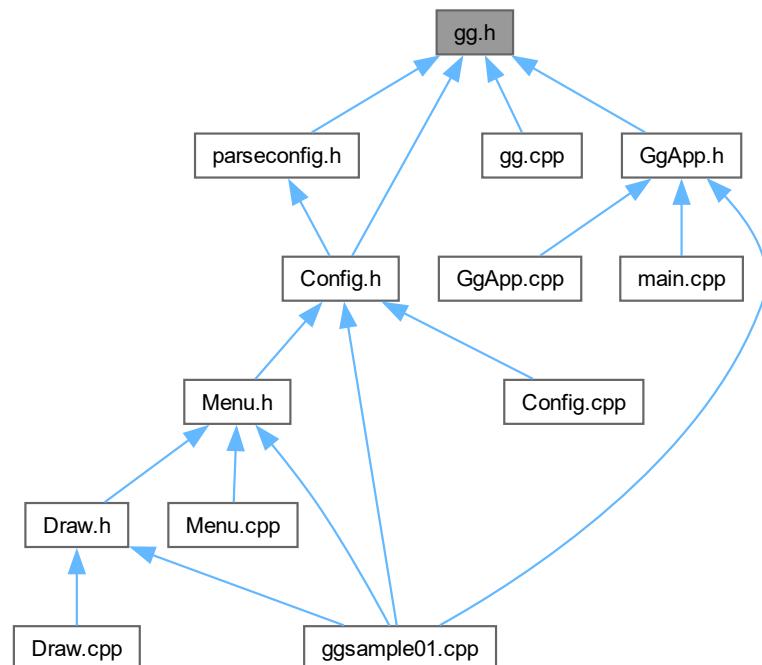
#include <cmath>
#include <array>
#include <vector>
#include <string>
#include <memory>
#include <cassert>

```

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::ggInit ()`
- `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::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 &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 >ext="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 &csrc, 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(*pos)[3], const GLfloat(*norm)[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。

9.11.1 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

July 17, 2025

[gg.h](#) に定義があります。

9.11.2 マクロ定義詳解

9.11.2.1 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。

9.11.2.2 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。

9.11.3 型定義詳解

9.11.3.1 pathChar

```
using pathChar = char
```

[gg.h](#) の 78 行目に定義があります。

9.11.3.2 pathString

```
using pathString = std::string
```

gg.h の 77 行目に定義があります。

9.11.4 関数詳解

9.11.4.1 TCharToUtf8()

```
std::string TCharToUtf8 (
    const pathString & cstring) [inline]
```

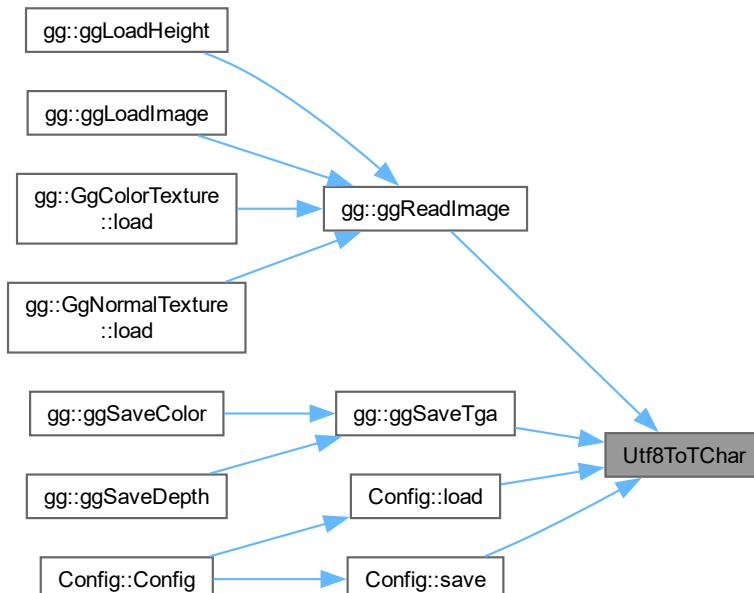
gg.h の 80 行目に定義があります。

9.11.4.2 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:



9.12 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 PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC glFramebufferSampleLocationsfvARB;
00368 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFNVPROC 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 glGetnUniformdvARB;
00493 extern PFNGLGETNUNIFORMDVPROC glGetnUniformdv;
00494 extern PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
00495 extern PFNGLGETNUNIFORMFPROC 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 PFNGLISTATENVPROC glIsStateNV;
00658 extern PFNGLISSYNCPROC glIsSync;
00659 extern PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00660 extern PFNGLISTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00661 extern PFNGLISTEXTUREPROC 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 PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC 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 PFNGLPATCHPARAMETERIPROC 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 PFNGLPATHPARAMETERINVNPROC glPathParameterivNV;
00815 extern PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
00816 extern PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00817 extern PFNGLPATHSTRINGNPROC 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 PFNGLPRIMITIVEBOXARBPROC 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 PFNGLREADNPPIXELSARBPROC glReadnPixelsARB;
00986 extern PFNGLREADNPPIXELSPROC 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 PFNGLSCISSORINDEXEDDVPROC glScissorIndexeddv;
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 PFNGLSTENCILCILOPPROC glStencilOp;
01022 extern PFNGLSTENCILCILOPSEPARATEPROC glStencilOpSeparate;
01023 extern PFNGLSTENCILCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
01024 extern PFNGLSTENCILCILSTROKEPATHNVPROC glStencilStrokePathNV;
01025 extern PFNGLSTENCILCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01026 extern PFNGLSTENCILCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01027 extern PFNGLSTENCILCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01028 extern PFNGLSTENCILCILTHENCOVERSTROKEPATHNVPROC 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 PFNGLTEXTUREBUFFERRANGEXTPROC 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() = default;
01558
01559     constexpr GgVector(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3) :
01560         std::array<GLfloat, 4>{ v0, v1, v2, v3 }
01561     {
01562     }
01563
01564     constexpr GgVector(const GLfloat* a) :
01565         GgVector{ a[0], a[1], a[2], a[3] }
01566     {
01567     }
01568
01569     constexpr GgVector(GLfloat c) :
01570         GgVector{ c, c, c, c }
01571     {
01572     }
01573
01574     GgVector(const GgVector& v) = default;
01575
01576     GgVector(GgVector&& v) = default;
01577
01578     ~GgVector() = default;
01579
01580     GgVector& operator=(const GgVector& v) = default;
01581
01582     GgVector& operator=(GgVector&& v) = default;
01583
01584     inline GgVector operator+(const GgVector& v) const
01585     {
01586         return GgVector{ (*this)[0] + v[0], (*this)[1] + v[1], (*this)[2] + v[2], (*this)[3] + v[3] };
01587     }
01588
01589     inline GgVector operator+(GLfloat c) const
01590     {
01591         return *this + GgVector(c);
01592     }
01593
01594     inline GgVector& operator+=(const GgVector& v)
01595     {
01596         *this = *this + v;
01597         return *this;
01598     }
01599
01600     inline GgVector& operator+=(GLfloat c)
01601     {
01602         *this = *this + c;
01603         return *this;
01604     }
01605
01606     inline GgVector operator-(<b>const</b> GgVector& v) const
01607     {
01608         return GgVector{ (*this)[0] - v[0], (*this)[1] - v[1], (*this)[2] - v[2], (*this)[3] - v[3] };
01609     }
01610
01611     inline GgVector operator-(<b>GLfloat</b> c) const

```

```
01702     {
01703         return *this - GgVector(c);
01704     }
01705
01712     inline GgVector& operator+=(const GgVector& v)
01713     {
01714         *this = *this + v;
01715         return *this;
01716     }
01717
01724     inline GgVector& operator+=(GLfloat c)
01725     {
01726         *this = *this + c;
01727         return *this;
01728     }
01729
01736     inline GgVector operator*(const GgVector& v) const
01737     {
01738         return GgVector{ (*this)[0] * v[0], (*this)[1] * v[1], (*this)[2] * v[2], (*this)[3] * v[3] };
01739     }
01740
01747     inline GgVector operator*(GLfloat c) const
01748     {
01749         return *this * GgVector(c);
01750     }
01751
01757     inline GgVector& operator*=(const GgVector& v)
01758     {
01759         *this = *this * v;
01760         return *this;
01761     }
01762
01769     inline GgVector& operator*=(GLfloat c)
01770     {
01771         *this = *this * c;
01772         return *this;
01773     }
01774
01781     inline GgVector operator/(const GgVector& v) const
01782     {
01783         return GgVector{ (*this)[0] / v[0], (*this)[1] / v[1], (*this)[2] / v[2], (*this)[3] / v[3] };
01784     }
01785
01792     inline GgVector operator/(GLfloat c) const
01793     {
01794         return *this / GgVector(c);
01795     }
01796
01803     inline GgVector& operator/=(GgVector& v)
01804     {
01805         *this = *this / v;
01806         return *this;
01807     }
01808
01815     inline GgVector& operator/=(GLfloat c)
01816     {
01817         *this = *this / c;
01818         return *this;
01819     }
01820
01827     inline GLfloat dot3(const GgVector& v) const
01828     {
01829         return ggDot3(data(), v.data());
01830     }
01831
01837     inline GLfloat length3() const
01838     {
01839         return ggLength3(data());
01840     }
01841
01848     inline GLfloat distance3(const GgVector& v) const
01849     {
01850         return ggDistance3(data(), v.data());
01851     }
01852
01858     inline GgVector normalize3() const
01859     {
01860         GgVector b;
01861         ggNormalize3(data(), b.data());
01862         return b;
01863     }
01864
01871     inline GLfloat dot4(const GgVector& v) const
01872     {
01873         return ggDot4(data(), v.data());
01874     }
01875
```

```

01881     inline GLfloat length4() const
01882     {
01883         return ggLength4(data());
01884     }
01885
01892     inline GLfloat distance4(const GgVector& v) const
01893     {
01894         return ggDistance4(data(), v.data());
01895     }
01896
01902     inline GgVector normalize4() const
01903     {
01904         GgVector b;
01905         ggNormalize4(data(), b.data());
01906         return b;
01907     }
01908 };
01909
01916     inline const GgVector& operator+(const GgVector& v)
01917     {
01918         return v;
01919     }
01920
01928     inline GgVector operator+(GLfloat a, const GgVector& b)
01929     {
01930         return GgVector{ a + b[0], a + b[1], a + b[2], a + b[3] };
01931     }
01932
01939     inline const GgVector operator-(const GgVector& v)
01940     {
01941         return GgVector{ -v[0], -v[1], -v[2], -v[3] };
01942     }
01943
01951     inline GgVector operator-(GLfloat a, const GgVector& b)
01952     {
01953         return GgVector{ a - b[0], a - b[1], a - b[2], a - b[3] };
01954     }
01955
01963     inline GgVector operator*(GLfloat a, const GgVector& b)
01964     {
01965         return GgVector{ a * b[0], a * b[1], a * b[2], a * b[3] };
01966     }
01967
01975     inline GgVector operator/(GLfloat a, const GgVector& b)
01976     {
01977         return GgVector{ a / b[0], a / b[1], a / b[2], a / b[3] };
01978     }
01979
01990     inline GgVector ggCross(const GgVector& a, const GgVector& b)
01991     {
01992         GgVector c;
01993         ggCross(c.data(), a.data(), b.data());
01994         return c;
01995     }
01996
02004     inline GLfloat ggDot3(const GgVector& a, const GgVector& b)
02005     {
02006         return ggDot3(a.data(), b.data());
02007     }
02008
02015     inline GLfloat ggLength3(const GgVector& a)
02016     {
02017         return ggLength3(a.data());
02018     }
02019
02027     inline GLfloat ggDistance3(const GgVector& a, const GgVector& b)
02028     {
02029         return ggDistance3(a.data(), b.data());
02030     }
02031
02041     inline GgVector ggNormalize3(const GgVector& a)
02042     {
02043         GgVector b;
02044         ggNormalize3(a.data(), b.data());
02045         b.data()[3] = 0.0f;
02046         return b;
02047     }
02048
02057     inline void ggNormalize3(GgVector* a)
02058     {
02059         ggNormalize3(a->data());
02060         a->data()[3] = 0.0f;
02061     }
02062
02070     inline GLfloat ggDot4(const GgVector& a, const GgVector& b)
02071     {
02072         return ggDot4(a.data(), b.data());
02073     }

```

```
02073 }
02074
02081 inline GLfloat ggLength4(const GgVector& a)
02082 {
02083     return ggLength4(a.data());
02084 }
02085
02093 inline GLfloat ggDistance4(const GgVector& a, const GgVector& b)
02094 {
02095     return ggDistance4(a.data(), b.data());
02096 }
02097
02104 inline GgVector ggNormalize4(const GgVector& a)
02105 {
02106     GgVector b;
02107     ggNormalize4(a.data(), b.data());
02108     return b;
02109 }
02110
02116 inline void ggNormalize4(GgVector* a)
02117 {
02118     ggNormalize4(a->data());
02119 }
02120
02124 class GgMatrix : public std::array<GLfloat, 16>
02125 {
02126     // 行列 a とベクトル b の積をベクトル c に格納する
02127     void projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02128
02129     // 行列 a と行列 b の積を行列 c に格納する
02130     void multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02131
02132 public:
02133
02137     GgMatrix() = default;
02138
02159     constexpr GgMatrix(
02160         GLfloat m00, GLfloat m01, GLfloat m02, GLfloat m03,
02161         GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13,
02162         GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23,
02163         GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33
02164     ) :
02165         std::array<GLfloat, 16>{ m00, m01, m02, m03, m10, m11, m12, m13, m20, m21, m22, m23, m30, m31,
02166         m32, m33 }
02166     {
02167     }
02168
02174     constexpr GgMatrix(const GLfloat* a) :
02175         GgMatrix{ a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9], a[10], a[11], a[12],
02176         a[13], a[14], a[15] }
02177     {
02178
02184     constexpr GgMatrix(GLfloat c) :
02185         GgMatrix{ c, c }
02186     {
02187     }
02188
02192     GgMatrix(const GgMatrix& m) = default;
02193
02197     GgMatrix(GgMatrix&& m) = default;
02198
02202     ~GgMatrix() = default;
02203
02207     GgMatrix& operator=(const GgMatrix& m) = default;
02208
02212     GgMatrix& operator=(GgMatrix&& m) = default;
02213
02220     GgMatrix& operator=(const GLfloat* a)
02221     {
02222         *this = GgMatrix(a);
02223         return *this;
02224     }
02225
02232     GgMatrix operator+(const GLfloat* a) const
02233     {
02234         GgMatrix t;
02235         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] + a[i];
02236         return t;
02237     }
02238
02245     GgMatrix operator+(const GgMatrix& m) const
02246     {
02247         return operator+(m.data());
02248     }
02249
02256     GgMatrix& operator+=(const GLfloat* a)
```

```

02257
02258     {
02259         for (std::size_t i = 0; i < size(); ++i) data()[i] += a[i];
02260     }
02261
02262     GgMatrix& operator+=(const GgMatrix& m)
02263     {
02264         return operator+=(m.data());
02265     }
02266
02267     GgMatrix operator-(const GLfloat* a) const
02268     {
02269         GgMatrix t;
02270         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] - a[i];
02271         return t;
02272     }
02273
02274     GgMatrix operator-(const GgMatrix& m) const
02275     {
02276         return operator-(m.data());
02277     }
02278
02279     GgMatrix& operator-=(const GLfloat* a)
02280     {
02281         for (std::size_t i = 0; i < size(); ++i) data()[i] -= a[i];
02282         return *this;
02283     }
02284
02285
02286     GgMatrix& operator-=(const GgMatrix& m) const
02287     {
02288         return operator-(m.data());
02289     }
02290
02291     GgMatrix& operator-=(const GgMatrix& m)
02292     {
02293         for (std::size_t i = 0; i < size(); ++i) data()[i] -= m.data()[i];
02294         return *this;
02295     }
02296
02297     GgMatrix& operator-=(const GLfloat* a)
02298     {
02299         for (std::size_t i = 0; i < size(); ++i) data()[i] -= a[i];
02300         return *this;
02301     }
02302
02303     GgMatrix& operator-=(const GgMatrix& m)
02304     {
02305         for (std::size_t i = 0; i < size(); ++i) data()[i] -= m.data()[i];
02306         return *this;
02307     }
02308
02309
02310     GgMatrix operator*(const GLfloat* a) const
02311     {
02312         GgMatrix t;
02313         multiply(t.data(), data(), a);
02314         return t;
02315     }
02316
02317     GgMatrix operator*(const GgMatrix& m) const
02318     {
02319         return operator*(m.data());
02320     }
02321
02322
02323     GgMatrix operator*(const GLfloat* a) const
02324     {
02325         GgMatrix t;
02326         multiply(t.data(), data(), a);
02327         return t;
02328     }
02329
02330     GgMatrix operator*(const GgMatrix& m) const
02331     {
02332         return operator*(m.data());
02333     }
02334
02335
02336     GgMatrix& operator*=(const GLfloat* a)
02337     {
02338         *this = operator*(a);
02339         return *this;
02340     }
02341
02342
02343     GgMatrix& operator*=(const GgMatrix& m)
02344     {
02345         return operator*(m.data());
02346     }
02347
02348
02349     GgMatrix& operator/=(const GLfloat* a)
02350     {
02351         *this = operator/(a);
02352         return *this;
02353     }
02354
02355
02356     GgMatrix& operator/=(const GgMatrix& m)
02357     {
02358         return operator/(m.data());
02359     }
02360
02361
02362     GgMatrix operator/(const GLfloat* a) const
02363     {
02364         GgMatrix t;
02365         i.loadInvert(a);
02366         multiply(t.data(), data(), i.data());
02367         return t;
02368     }
02369
02370
02371     GgMatrix operator/(const GgMatrix& m) const
02372     {
02373         return operator/(m.data());
02374     }
02375
02376
02377     GgMatrix& operator/=(const GLfloat* a)
02378     {
02379         *this = operator/(a);
02380         return *this;
02381     }
02382
02383
02384     GgMatrix& operator/=(const GgMatrix& m)
02385     {
02386         return operator/(m.data());
02387     }
02388
02389
02390     GgMatrix& operator/=(const GLfloat* a)
02391     {
02392         *this = operator/(a);
02393         return *this;
02394     }
02395
02396
02397     GgMatrix& operator/=(const GgMatrix& m)
02398     {
02399         return operator/(m.data());
02400     }
02401
02402
02403     GgMatrix& loadIdentity()
02404     {
02405         GgMatrix t;
02406         t[0] = 1.0f;
02407         t[1] = 0.0f;
02408         t[2] = 0.0f;
02409         t[3] = 0.0f;
02410         t[4] = 0.0f;
02411         t[5] = 1.0f;
02412         t[6] = 0.0f;
02413         t[7] = 0.0f;
02414         t[8] = 0.0f;
02415         t[9] = 0.0f;
02416         t[10] = 0.0f;
02417         t[11] = 1.0f;
02418         t[12] = 0.0f;
02419         t[13] = 0.0f;
02420         t[14] = 0.0f;
02421         t[15] = 0.0f;
02422         return t;
02423     }
02424
02425     GgMatrix& loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02426
02427     GgMatrix& loadTranslate(const GLfloat* t)
02428     {
02429         return loadTranslate(t[0], t[1], t[2]);
02430     }
02431
02432
02433     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z)
02434     {
02435         GgMatrix t;
02436         t[0] = x;
02437         t[1] = 0.0f;
02438         t[2] = 0.0f;
02439         t[3] = 0.0f;
02440         t[4] = 0.0f;
02441         t[5] = y;
02442         t[6] = 0.0f;
02443         t[7] = 0.0f;
02444         t[8] = 0.0f;
02445         t[9] = 0.0f;
02446         t[10] = 0.0f;
02447         t[11] = z;
02448         t[12] = 0.0f;
02449         t[13] = 0.0f;
02450         t[14] = 0.0f;
02451         t[15] = 1.0f;
02452         return t;
02453     }
02454
02455
02456     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02457     {
02458         GgMatrix t;
02459         t[0] = x;
02460         t[1] = 0.0f;
02461         t[2] = 0.0f;
02462         t[3] = 0.0f;
02463         t[4] = 0.0f;
02464         t[5] = y;
02465         t[6] = 0.0f;
02466         t[7] = 0.0f;
02467         t[8] = 0.0f;
02468         t[9] = 0.0f;
02469         t[10] = 0.0f;
02470         t[11] = z;
02471         t[12] = 0.0f;
02472         t[13] = 0.0f;
02473         t[14] = 0.0f;
02474         t[15] = w;
02475         return t;
02476     }
02477
02478
02479     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v)
02480     {
02481         GgMatrix t;
02482         t[0] = x;
02483         t[1] = 0.0f;
02484         t[2] = 0.0f;
02485         t[3] = 0.0f;
02486         t[4] = 0.0f;
02487         t[5] = y;
02488         t[6] = 0.0f;
02489         t[7] = 0.0f;
02490         t[8] = 0.0f;
02491         t[9] = 0.0f;
02492         t[10] = 0.0f;
02493         t[11] = z;
02494         t[12] = 0.0f;
02495         t[13] = 0.0f;
02496         t[14] = 0.0f;
02497         t[15] = w;
02498         t[16] = u;
02499         t[17] = v;
02500         return t;
02501     }
02502
02503
02504     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2)
02505     {
02506         GgMatrix t;
02507         t[0] = x;
02508         t[1] = 0.0f;
02509         t[2] = 0.0f;
02510         t[3] = 0.0f;
02511         t[4] = 0.0f;
02512         t[5] = y;
02513         t[6] = 0.0f;
02514         t[7] = 0.0f;
02515         t[8] = 0.0f;
02516         t[9] = 0.0f;
02517         t[10] = 0.0f;
02518         t[11] = z;
02519         t[12] = 0.0f;
02520         t[13] = 0.0f;
02521         t[14] = 0.0f;
02522         t[15] = w;
02523         t[16] = u;
02524         t[17] = v;
02525         t[18] = w2;
02526         return t;
02527     }
02528
02529
02530     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3)
02531     {
02532         GgMatrix t;
02533         t[0] = x;
02534         t[1] = 0.0f;
02535         t[2] = 0.0f;
02536         t[3] = 0.0f;
02537         t[4] = 0.0f;
02538         t[5] = y;
02539         t[6] = 0.0f;
02540         t[7] = 0.0f;
02541         t[8] = 0.0f;
02542         t[9] = 0.0f;
02543         t[10] = 0.0f;
02544         t[11] = z;
02545         t[12] = 0.0f;
02546         t[13] = 0.0f;
02547         t[14] = 0.0f;
02548         t[15] = w;
02549         t[16] = u;
02550         t[17] = v;
02551         t[18] = w2;
02552         t[19] = w3;
02553         return t;
02554     }
02555
02556
02557     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4)
02558     {
02559         GgMatrix t;
02560         t[0] = x;
02561         t[1] = 0.0f;
02562         t[2] = 0.0f;
02563         t[3] = 0.0f;
02564         t[4] = 0.0f;
02565         t[5] = y;
02566         t[6] = 0.0f;
02567         t[7] = 0.0f;
02568         t[8] = 0.0f;
02569         t[9] = 0.0f;
02570         t[10] = 0.0f;
02571         t[11] = z;
02572         t[12] = 0.0f;
02573         t[13] = 0.0f;
02574         t[14] = 0.0f;
02575         t[15] = w;
02576         t[16] = u;
02577         t[17] = v;
02578         t[18] = w2;
02579         t[19] = w3;
02580         t[20] = w4;
02581         return t;
02582     }
02583
02584
02585     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5)
02586     {
02587         GgMatrix t;
02588         t[0] = x;
02589         t[1] = 0.0f;
02590         t[2] = 0.0f;
02591         t[3] = 0.0f;
02592         t[4] = 0.0f;
02593         t[5] = y;
02594         t[6] = 0.0f;
02595         t[7] = 0.0f;
02596         t[8] = 0.0f;
02597         t[9] = 0.0f;
02598         t[10] = 0.0f;
02599         t[11] = z;
02600         t[12] = 0.0f;
02601         t[13] = 0.0f;
02602         t[14] = 0.0f;
02603         t[15] = w;
02604         t[16] = u;
02605         t[17] = v;
02606         t[18] = w2;
02607         t[19] = w3;
02608         t[20] = w4;
02609         t[21] = w5;
02610         return t;
02611     }
02612
02613
02614     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6)
02615     {
02616         GgMatrix t;
02617         t[0] = x;
02618         t[1] = 0.0f;
02619         t[2] = 0.0f;
02620         t[3] = 0.0f;
02621         t[4] = 0.0f;
02622         t[5] = y;
02623         t[6] = 0.0f;
02624         t[7] = 0.0f;
02625         t[8] = 0.0f;
02626         t[9] = 0.0f;
02627         t[10] = 0.0f;
02628         t[11] = z;
02629         t[12] = 0.0f;
02630         t[13] = 0.0f;
02631         t[14] = 0.0f;
02632         t[15] = w;
02633         t[16] = u;
02634         t[17] = v;
02635         t[18] = w2;
02636         t[19] = w3;
02637         t[20] = w4;
02638         t[21] = w5;
02639         t[22] = w6;
02640         return t;
02641     }
02642
02643
02644     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7)
02645     {
02646         GgMatrix t;
02647         t[0] = x;
02648         t[1] = 0.0f;
02649         t[2] = 0.0f;
02650         t[3] = 0.0f;
02651         t[4] = 0.0f;
02652         t[5] = y;
02653         t[6] = 0.0f;
02654         t[7] = 0.0f;
02655         t[8] = 0.0f;
02656         t[9] = 0.0f;
02657         t[10] = 0.0f;
02658         t[11] = z;
02659         t[12] = 0.0f;
02660         t[13] = 0.0f;
02661         t[14] = 0.0f;
02662         t[15] = w;
02663         t[16] = u;
02664         t[17] = v;
02665         t[18] = w2;
02666         t[19] = w3;
02667         t[20] = w4;
02668         t[21] = w5;
02669         t[22] = w6;
02670         t[23] = w7;
02671         return t;
02672     }
02673
02674
02675     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8)
02676     {
02677         GgMatrix t;
02678         t[0] = x;
02679         t[1] = 0.0f;
02680         t[2] = 0.0f;
02681         t[3] = 0.0f;
02682         t[4] = 0.0f;
02683         t[5] = y;
02684         t[6] = 0.0f;
02685         t[7] = 0.0f;
02686         t[8] = 0.0f;
02687         t[9] = 0.0f;
02688         t[10] = 0.0f;
02689         t[11] = z;
02690         t[12] = 0.0f;
02691         t[13] = 0.0f;
02692         t[14] = 0.0f;
02693         t[15] = w;
02694         t[16] = u;
02695         t[17] = v;
02696         t[18] = w2;
02697         t[19] = w3;
02698         t[20] = w4;
02699         t[21] = w5;
02700         t[22] = w6;
02701         t[23] = w7;
02702         t[24] = w8;
02703         return t;
02704     }
02705
02706
02707     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9)
02708     {
02709         GgMatrix t;
02710         t[0] = x;
02711         t[1] = 0.0f;
02712         t[2] = 0.0f;
02713         t[3] = 0.0f;
02714         t[4] = 0.0f;
02715         t[5] = y;
02716         t[6] = 0.0f;
02717         t[7] = 0.0f;
02718         t[8] = 0.0f;
02719         t[9] = 0.0f;
02720         t[10] = 0.0f;
02721         t[11] = z;
02722         t[12] = 0.0f;
02723         t[13] = 0.0f;
02724         t[14] = 0.0f;
02725         t[15] = w;
02726         t[16] = u;
02727         t[17] = v;
02728         t[18] = w2;
02729         t[19] = w3;
02730         t[20] = w4;
02731         t[21] = w5;
02732         t[22] = w6;
02733         t[23] = w7;
02734         t[24] = w8;
02735         t[25] = w9;
02736         return t;
02737     }
02738
02739
02740     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10)
02741     {
02742         GgMatrix t;
02743         t[0] = x;
02744         t[1] = 0.0f;
02745         t[2] = 0.0f;
02746         t[3] = 0.0f;
02747         t[4] = 0.0f;
02748         t[5] = y;
02749         t[6] = 0.0f;
02750         t[7] = 0.0f;
02751         t[8] = 0.0f;
02752         t[9] = 0.0f;
02753         t[10] = 0.0f;
02754         t[11] = z;
02755         t[12] = 0.0f;
02756         t[13] = 0.0f;
02757         t[14] = 0.0f;
02758         t[15] = w;
02759         t[16] = u;
02760         t[17] = v;
02761         t[18] = w2;
02762         t[19] = w3;
02763         t[20] = w4;
02764         t[21] = w5;
02765         t[22] = w6;
02766         t[23] = w7;
02767         t[24] = w8;
02768         t[25] = w9;
02769         t[26] = w10;
02770         return t;
02771     }
02772
02773
02774     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11)
02775     {
02776         GgMatrix t;
02777         t[0] = x;
02778         t[1] = 0.0f;
02779         t[2] = 0.0f;
02780         t[3] = 0.0f;
02781         t[4] = 0.0f;
02782         t[5] = y;
02783         t[6] = 0.0f;
02784         t[7] = 0.0f;
02785         t[8] = 0.0f;
02786         t[9] = 0.0f;
02787         t[10] = 0.0f;
02788         t[11] = z;
02789         t[12] = 0.0f;
02790         t[13] = 0.0f;
02791         t[14] = 0.0f;
02792         t[15] = w;
02793         t[16] = u;
02794         t[17] = v;
02795         t[18] = w2;
02796         t[19] = w3;
02797         t[20] = w4;
02798         t[21] = w5;
02799         t[22] = w6;
02800         t[23] = w7;
02801         t[24] = w8;
02802         t[25] = w9;
02803         t[26] = w10;
02804         t[27] = w11;
02805         return t;
02806     }
02807
02808
02809     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12)
02810     {
02811         GgMatrix t;
02812         t[0] = x;
02813         t[1] = 0.0f;
02814         t[2] = 0.0f;
02815         t[3] = 0.0f;
02816         t[4] = 0.0f;
02817         t[5] = y;
02818         t[6] = 0.0f;
02819         t[7] = 0.0f;
02820         t[8] = 0.0f;
02821         t[9] = 0.0f;
02822         t[10] = 0.0f;
02823         t[11] = z;
02824         t[12] = 0.0f;
02825         t[13] = 0.0f;
02826         t[14] = 0.0f;
02827         t[15] = w;
02828         t[16] = u;
02829         t[17] = v;
02830         t[18] = w2;
02831         t[19] = w3;
02832         t[20] = w4;
02833         t[21] = w5;
02834         t[22] = w6;
02835         t[23] = w7;
02836         t[24] = w8;
02837         t[25] = w9;
02838         t[26] = w10;
02839         t[27] = w11;
02840         t[28] = w12;
02841         return t;
02842     }
02843
02844
02845     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13)
02846     {
02847         GgMatrix t;
02848         t[0] = x;
02849         t[1] = 0.0f;
02850         t[2] = 0.0f;
02851         t[3] = 0.0f;
02852         t[4] = 0.0f;
02853         t[5] = y;
02854         t[6] = 0.0f;
02855         t[7] = 0.0f;
02856         t[8] = 0.0f;
02857         t[9] = 0.0f;
02858         t[10] = 0.0f;
02859         t[11] = z;
02860         t[12] = 0.0f;
02861         t[13] = 0.0f;
02862         t[14] = 0.0f;
02863         t[15] = w;
02864         t[16] = u;
02865         t[17] = v;
02866         t[18] = w2;
02867         t[19] = w3;
02868         t[20] = w4;
02869         t[21] = w5;
02870         t[22] = w6;
02871         t[23] = w7;
02872         t[24] = w8;
02873         t[25] = w9;
02874         t[26] = w10;
02875         t[27] = w11;
02876         t[28] = w12;
02877         t[29] = w13;
02878         return t;
02879     }
02880
02881
02882     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13, GLfloat w14)
02883     {
02884         GgMatrix t;
02885         t[0] = x;
02886         t[1] = 0.0f;
02887         t[2] = 0.0f;
02888         t[3] = 0.0f;
02889         t[4] = 0.0f;
02890         t[5] = y;
02891         t[6] = 0.0f;
02892         t[7] = 0.0f;
02893         t[8] = 0.0f;
02894         t[9] = 0.0f;
02895         t[10] = 0.0f;
02896         t[11] = z;
02897         t[12] = 0.0f;
02898         t[13] = 0.0f;
02899         t[14] = 0.0f;
02900         t[15] = w;
02901         t[16] = u;
02902         t[17] = v;
02903         t[18] = w2;
02904         t[19] = w3;
02905         t[20] = w4;
02906         t[21] = w5;
02907         t[22] = w6;
02908         t[23] = w7;
02909         t[24] = w8;
02910         t[25] = w9;
02911         t[26] = w10;
02912         t[27] = w11;
02913         t[28] = w12;
02914         t[29] = w13;
02915         t[30] = w14;
02916         return t;
02917     }
02918
02919
02920     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13, GLfloat w14, GLfloat w15)
02921     {
02922         GgMatrix t;
02923         t[0] = x;
02924         t[1] = 0.0f;
02925         t[2] = 0.0f;
02926         t[3] = 0.0f;
02927         t[4] = 0.0f;
02928         t[5] = y;
02929         t[6] = 0.0f;
02930         t[7] = 0.0f;
02931         t[8] = 0.0f;
02932         t[9] = 0.0f;
02933         t[10] = 0.0f;
02934         t[11] = z;
02935         t[12] = 0.0f;
02936         t[13] = 0.0f;
02937         t[14] = 0.0f;
02938         t[15] = w;
02939         t[16] = u;
02940         t[17] = v;
02941         t[18] = w2;
02942         t[19] = w3;
02943         t[20] = w4;
02944         t[21] = w5;
02945         t[22] = w6;
02946         t[23] = w7;
02947         t[24] = w8;
02948         t[25] = w9;
02949         t[26] = w10;
02950         t[27] = w11;
02951         t[28] = w12;
02952         t[29] = w13;
02953         t[30] = w14;
02954         t[31] = w15;
02955         return t;
02956     }
02957
02958
02959     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13, GLfloat w14, GLfloat w15, GLfloat w16)
02960     {
02961         GgMatrix t;
02962         t[0] = x;
02963         t[1] = 0.0f;
02964         t[2] = 0.0f;
02965         t[3] = 0.0f;
02966         t[4] = 0.0f;
02967         t[5] = y;
02968         t[6] = 0.0f;
02969         t[7] = 0.0f;
02970         t[8] = 0.0f;
02971         t[9] = 0.0f;
02972         t[10] = 0.0f;
02973         t[11] = z;
02974         t[12] = 0.0f;
02975         t[13] = 0.0f;
02976         t[14] = 0.0f;
02977         t[15] = w;
02978         t[16] = u;
02979         t[17] = v;
02980         t[18] = w2;
02981         t[19] = w3;
02982         t[20] = w4;
02983         t[21] = w5;
02984         t[22] = w6;
02985         t[23] = w7;
02986         t[24] = w8;
02987         t[25] = w9;
02988         t[26] = w10;
02989         t[27] = w11;
02990         t[28] = w12;
02991         t[29] = w13;
02992         t[30] = w14;
02993         t[31] = w15;
02994         t[32] = w16;
02995         return t;
02996     }
02997
02998
02999     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13, GLfloat w14, GLfloat w15, GLfloat w16, GLfloat w17)
03000     {
03001         GgMatrix t;
03002         t[0] = x;
03003         t[1] = 0.0f;
03004         t[2] = 0.0f;
03005         t[3] = 0.0f;
03006         t[4] = 0.0f;
03007         t[5] = y;
03008         t[6] = 0.0f;
03009         t[7] = 0.0f;
03010         t[8] = 0.0f;
03011         t[9] = 0.0f;
03012         t[10] = 0.0f;
03013         t[11] = z;
03014         t[12] = 0.0f;
03015         t[13] = 0.0f;
03016         t[14] = 0.0f;
03017         t[15] = w;
03018         t[16] = u;
03019         t[17] = v;
03020         t[18] = w2;
03021         t[19] = w3;
03022         t[20] = w4;
03023         t[21] = w5;
03024         t[22] = w6;
03025         t[23] = w7;
03026         t[24] = w8;
03027         t[25] = w9;
03028         t[26] = w10;
03029         t[27] = w11;
03030         t[28] = w12;
03031         t[29] = w13;
03032         t[30] = w14;
03033         t[31] = w15;
03034         t[32] = w16;
03035         t[33] = w17;
03036         return t;
03037     }
03038
03039
03040     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2, GLfloat w3, GLfloat w4, GLfloat w5, GLfloat w6, GLfloat w7, GLfloat w8, GLfloat w9, GLfloat w10, GLfloat w11, GLfloat w12, GLfloat w13, GLfloat w14, GLfloat w15, GLfloat w16, GLfloat w17, GLfloat w18)
03041     {
03042         GgMatrix t;
03043         t[0] = x;
03044         t[1] = 0.0f;
03045         t[2] = 0.0f;
03046         t[3] = 0.0f;
03047         t[4] = 0.0f;
03048         t[5] = y;
03049         t[6] = 0.0f;
03050         t[7] = 0.0f;
03051         t[8] = 0.0f;
03052         t[9] = 0.0f;
03053         t[10] = 0.0f;
03054         t[11] = z;
03055         t[12] = 0.0f;
03056         t[13] = 0.0f;
03057         t[14] = 0.0f;
03058         t[15] = w;
03059         t[16] = u;
03060         t[17] = v;
03061         t[18] = w2;
03062         t[19] = w3;
03063         t[20] = w4;
03064         t[21] = w5;
03065         t[22] = w6;
03066         t[23] = w7;
03067         t[24] = w8;
03068         t[25] = w9;
03069         t[26] = w10;
03070         t[27] = w11;
03071         t[28] = w12;
03072         t[29] = w13;
03073         t[30] = w14;
03074         t[31] = w15;
03075         t[32] = w16;
03076         t[33] = w17;
03077         t[34] = w18;
03078         return t;
03079     }
03080
03081
03082     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w, GLfloat u, GLfloat v, GLfloat w2
```

```
02440     }
02441
02448     GgMatrix& loadTranslate(const GgVector& t)
02449     {
02450         return loadTranslate(t[0], t[1], t[2], t[3]);
02451     }
02452
02462     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02463
02470     GgMatrix& loadScale(const GLfloat* s)
02471     {
02472         return loadScale(s[0], s[1], s[2]);
02473     }
02474
02481     GgMatrix& loadScale(const GgVector& s)
02482     {
02483         return loadScale(s[0], s[1], s[2], s[3]);
02484     }
02485
02492     GgMatrix& loadRotateX(GLfloat a);
02493
02500     GgMatrix& loadRotateY(GLfloat a);
02501
02508     GgMatrix& loadRotateZ(GLfloat a);
02509
02519     GgMatrix& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
02520
02528     GgMatrix& loadRotate(const GLfloat* r, GLfloat a)
02529     {
02530         return loadRotate(r[0], r[1], r[2], a);
02531     }
02532
02540     GgMatrix& loadRotate(const GgVector& r, GLfloat a)
02541     {
02542         return loadRotate(r[0], r[1], r[2], a);
02543     }
02544
02551     GgMatrix& loadRotate(const GLfloat* r)
02552     {
02553         return loadRotate(r[0], r[1], r[2], r[3]);
02554     }
02555
02562     GgMatrix& loadRotate(const GgVector& r)
02563     {
02564         return loadRotate(r[0], r[1], r[2], r[3]);
02565     }
02566
02581     GgMatrix& loadLookat(GLfloat ex, GLfloat ey, GLfloat ez,
02582         GLfloat tx, GLfloat ty, GLfloat tz,
02583         GLfloat ux, GLfloat uy, GLfloat uz);
02584
02593     GgMatrix& loadLookat(const GLfloat* e, const GLfloat* t, const GLfloat* u)
02594     {
02595         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02596     }
02597
02606     GgMatrix& loadLookat(const GgVector& e, const GgVector& t, const GgVector& u)
02607     {
02608         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02609     }
02610
02622     GgMatrix& loadOrthogonal(GLfloat left, GLfloat right,
02623         GLfloat bottom, GLfloat top,
02624         GLfloat zNear, GLfloat zFar);
02625
02637     GgMatrix& loadFrustum(GLfloat left, GLfloat right,
02638         GLfloat bottom, GLfloat top,
02639         GLfloat zNear, GLfloat zFar);
02640
02650     GgMatrix& loadPerspective(GLfloat fovy, GLfloat aspect,
02651         GLfloat zNear, GLfloat zFar);
02652
02659     GgMatrix& loadTranspose(const GLfloat* a);
02660
02667     GgMatrix& loadTranspose(const GgMatrix& m)
02668     {
02669         return loadTranspose(m.data());
02670     }
02671
02678     GgMatrix& loadInvert(const GLfloat* a);
02679
02686     GgMatrix& loadInvert(const GgMatrix& m)
02687     {
02688         return loadInvert(m.data());
02689     }
02690
02697     GgMatrix& loadNormal(const GLfloat* a);
```

```
02698
02705     GgMatrix& loadNormal(const GgMatrix& m)
02706     {
02707         return loadNormal(m.data());
02708     }
02709
02719     GgMatrix translate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02720     {
02721         GgMatrix m;
02722         return operator*(m.loadTranslate(x, y, z, w));
02723     }
02724
02731     GgMatrix translate(const GLfloat* t) const
02732     {
02733         return translate(t[0], t[1], t[2]);
02734     }
02735
02742     GgMatrix translate(const GgVector& t) const
02743     {
02744         return translate(t[0], t[1], t[2], t[3]);
02745     }
02746
02756     GgMatrix scale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02757     {
02758         GgMatrix m;
02759         return operator*(m.loadScale(x, y, z, w));
02760     }
02761
02768     GgMatrix scale(const GLfloat* s) const
02769     {
02770         return scale(s[0], s[1], s[2]);
02771     }
02772
02779     GgMatrix scale(const GgVector& s) const
02780     {
02781         return scale(s[0], s[1], s[2], s[3]);
02782     }
02783
02790     GgMatrix rotateX(GLfloat a) const
02791     {
02792         GgMatrix m;
02793         return operator*(m.loadRotateX(a));
02794     }
02795
02802     GgMatrix rotateY(GLfloat a) const
02803     {
02804         GgMatrix m;
02805         return operator*(m.loadRotateY(a));
02806     }
02807
02814     GgMatrix rotateZ(GLfloat a) const
02815     {
02816         GgMatrix m;
02817         return operator*(m.loadRotateZ(a));
02818     }
02819
02829     GgMatrix rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
02830     {
02831         GgMatrix m;
02832         return operator*(m.loadRotate(x, y, z, a));
02833     }
02834
02842     GgMatrix rotate(const GLfloat* r, GLfloat a) const
02843     {
02844         return rotate(r[0], r[1], r[2], a);
02845     }
02846
02854     GgMatrix rotate(const GgVector& r, GLfloat a) const
02855     {
02856         return rotate(r[0], r[1], r[2], a);
02857     }
02858
02865     GgMatrix rotate(const GLfloat* r) const
02866     {
02867         return rotate(r[0], r[1], r[2], r[3]);
02868     }
02869
02876     GgMatrix rotate(const GgVector& r) const
02877     {
02878         return rotate(r[0], r[1], r[2], r[3]);
02879     }
02880
02895     GgMatrix lookat(
02896         GLfloat ex, GLfloat ey, GLfloat ez,
02897         GLfloat tx, GLfloat ty, GLfloat tz,
02898         GLfloat ux, GLfloat uy, GLfloat uz
02899     ) const
```

```
02900  {
02901      GgMatrix m;
02902      return operator*(m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz));
02903  }
02904
02913  GgMatrix lookat(const GLfloat* e, const GLfloat* t, const GLfloat* u) const
02914  {
02915      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02916  }
02917
02926  GgMatrix lookat(const GgVector& e, const GgVector& t, const GgVector& u) const
02927  {
02928      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02929  }
02930
02942  GgMatrix orthogonal(
02943      GLfloat left, GLfloat right,
02944      GLfloat bottom, GLfloat top,
02945      GLfloat zNear, GLfloat zFar
02946  ) const
02947  {
02948      GgMatrix m;
02949      return operator*(m.loadOrthogonal(left, right, bottom, top, zNear, zFar));
02950  }
02951
02963  GgMatrix frustum(
02964      GLfloat left, GLfloat right,
02965      GLfloat bottom, GLfloat top,
02966      GLfloat zNear, GLfloat zFar
02967  ) const
02968  {
02969      GgMatrix m;
02970      return operator*(m.loadFrustum(left, right, bottom, top, zNear, zFar));
02971  }
02972
02982  GgMatrix perspective(
02983      GLfloat fovy, GLfloat aspect,
02984      GLfloat zNear, GLfloat zFar
02985  ) const
02986  {
02987      GgMatrix m;
02988      return operator*(m.loadPerspective(fovy, aspect, zNear, zFar));
02989  }
02990
02996  GgMatrix transpose() const
02997  {
02998      GgMatrix t;
02999      return t.loadTranspose(*this);
03000  }
03001
03007  GgMatrix invert() const
03008  {
03009      GgMatrix t;
03010      return t.loadInvert(*this);
03011  }
03012
03018  GgMatrix normal() const
03019  {
03020      GgMatrix t;
03021      return t.loadNormal(*this);
03022  }
03023
03030  void projection(GLfloat* c, const GLfloat* v) const
03031  {
03032      projection(c, data(), v);
03033  }
03034
03041  void projection(GLfloat* c, const GgVector& v) const
03042  {
03043      projection(c, v.data());
03044  }
03045
03052  void projection(GgVector& c, const GLfloat* v) const
03053  {
03054      projection(c.data(), v);
03055  }
03056
03063  void projection(GgVector& c, const GgVector& v) const
03064  {
03065      projection(c.data(), v.data());
03066  }
03067
03074  GgVector operator*(const GgVector& v) const
03075  {
03076      GgVector c;
03077      projection(c, v);
03078      return c;
```

```
03079 }
03080
03086 const GLfloat* get() const
03087 {
03088     return data();
03089 }
03090
03096 void get(GLfloat* a) const
03097 {
03098     std::copy(data(), data() + size(), a);
03099 }
03100 };
03101
03107 inline GgMatrix ggIdentity()
03108 {
03109     GgMatrix t;
03110     return t.loadIdentity();
03111 };
03112
03122 inline GgMatrix ggTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03123 {
03124     GgMatrix m;
03125     return m.loadTranslate(x, y, z, w);
03126 };
03127
03134 inline GgMatrix ggTranslate(const GLfloat* t)
03135 {
03136     GgMatrix m;
03137     return m.loadTranslate(t[0], t[1], t[2]);
03138 };
03139
03146 inline GgMatrix ggTranslate(const GgVector& t)
03147 {
03148     GgMatrix m;
03149     return m.loadTranslate(t[0], t[1], t[2], t[3]);
03150 };
03151
03161 inline GgMatrix ggScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03162 {
03163     GgMatrix m;
03164     return m.loadScale(x, y, z, w);
03165 };
03166
03173 inline GgMatrix ggScale(const GLfloat* s)
03174 {
03175     GgMatrix m;
03176     return m.loadScale(s[0], s[1], s[2]);
03177 };
03178
03185 inline GgMatrix ggScale(const GgVector& s)
03186 {
03187     GgMatrix m;
03188     return m.loadScale(s[0], s[1], s[2], s[3]);
03189 };
03190
03197 inline GgMatrix ggRotateX(GLfloat a)
03198 {
03199     GgMatrix m;
03200     return m.loadRotateX(a);
03201 };
03202
03209 inline GgMatrix ggRotateY(GLfloat a)
03210 {
03211     GgMatrix m;
03212     return m.loadRotateY(a);
03213 };
03214
03221 inline GgMatrix ggRotateZ(GLfloat a)
03222 {
03223     GgMatrix m;
03224     return m.loadRotateZ(a);
03225 };
03226
03236 inline GgMatrix ggRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03237 {
03238     GgMatrix m;
03239     return m.loadRotate(x, y, z, a);
03240 };
03241
03249 inline GgMatrix ggRotate(const GLfloat* r, GLfloat a)
03250 {
03251     GgMatrix m;
03252     return m.loadRotate(r[0], r[1], r[2], a);
03253 };
03254
03262 inline GgMatrix ggRotate(const GgVector& r, GLfloat a)
03263 {
```

```
03264     GgMatrix m;
03265     return m.loadRotate(r[0], r[1], r[2], a);
03266 }
03267
03274 inline GgMatrix ggRotate(const GLfloat* r)
03275 {
03276     GgMatrix m;
03277     return m.loadRotate(r[0], r[1], r[2], r[3]);
03278 }
03279
03286 inline GgMatrix ggRotate(const GgVector& r)
03287 {
03288     GgMatrix m;
03289     return m.loadRotate(r[0], r[1], r[2], r[3]);
03290 }
03291
03306 inline GgMatrix ggLookat(
03307     GLfloat ex, GLfloat ey, GLfloat ez,           // 視点の位置
03308     GLfloat tx, GLfloat ty, GLfloat tz,           // 目標点の位置
03309     GLfloat ux, GLfloat uy, GLfloat uz           // 上方向のベクトル
03310 )
03311 {
03312     GgMatrix m;
03313     return m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz);
03314 }
03315
03324 inline GgMatrix ggLookat(
03325     const GLfloat* e,                           // 視点の位置
03326     const GLfloat* t,                           // 目標点の位置
03327     const GLfloat* u                           // 上方向のベクトル
03328 )
03329 {
03330     GgMatrix m;
03331     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03332 }
03333
03342 inline GgMatrix ggLookat(
03343     const GgVector& e,                         // 視点の位置
03344     const GgVector& t,                         // 目標点の位置
03345     const GgVector& u                         // 上方向のベクトル
03346 )
03347 {
03348     GgMatrix m;
03349     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03350 }
03351
03363 inline GgMatrix ggOrthogonal(GLfloat left, GLfloat right,
03364     GLfloat bottom, GLfloat top,
03365     GLfloat zNear, GLfloat zFar)
03366 {
03367     GgMatrix m;
03368     return m.loadOrthogonal(left, right, bottom, top, zNear, zFar);
03369 }
03370
03382 inline GgMatrix ggFrustum(
03383     GLfloat left, GLfloat right,
03384     GLfloat bottom, GLfloat top,
03385     GLfloat zNear, GLfloat zFar
03386 )
03387 {
03388     GgMatrix m;
03389     return m.loadFrustum(left, right, bottom, top, zNear, zFar);
03390 }
03391
03401 inline GgMatrix ggPerspective(
03402     GLfloat fovy, GLfloat aspect,
03403     GLfloat zNear, GLfloat zFar
03404 )
03405 {
03406     GgMatrix m;
03407     return m.loadPerspective(fovy, aspect, zNear, zFar);
03408 }
03409
03416 inline GgMatrix ggTranspose(const GgMatrix& m)
03417 {
03418     return m.transpose();
03419 }
03420
03427 inline GgMatrix ggInvert(const GgMatrix& m)
03428 {
03429     return m.invert();
03430 }
03431
03438 inline GgMatrix ggNormal(const GgMatrix& m)
03439 {
03440     return m.normal();
03441 }
```

```

03442
03446 class GgQuaternion : public GgVector
03447 {
03448     // GgQuaternion 型の四元数 p と四元数 q の積を四元数 r に求める
03449     void multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const;
03450
03451     // GgQuaternion 型の四元数 q が表す回転の変換行列を m に求める
03452     void toMatrix(GLfloat* m, const GLfloat* q) const;
03453
03454     // 回転の変換行列 m が表す四元数を q に求める
03455     void toQuaternion(GLfloat* q, const GLfloat* m) const;
03456
03457     // 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
03458     void slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const;
03459
03460 public:
03461
03465     GgQuaternion() = default;
03466
03475     constexpr GgQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w) :
03476         GgVector{ x, y, z, w }
03477     {
03478     }
03479
03485     constexpr GgQuaternion(GLfloat c) :
03486         GgQuaternion{ c, c, c, c }
03487     {
03488     }
03489
03495     GgQuaternion(const GLfloat* a) :
03496         GgQuaternion{ a[0], a[1], a[2], a[3] }
03497     {
03498     }
03499
03505     GgQuaternion(const GgVector& v) :
03506         GgQuaternion{ v[0], v[1], v[2], v[3] }
03507     {
03508     }
03509
03513     GgQuaternion(const GgQuaternion& q) = default;
03514
03518     GgQuaternion(GgQuaternion&& q) = default;
03519
03523     ~GgQuaternion() = default;
03524
03528     GgQuaternion& operator=(const GgQuaternion& q) = default;
03529
03533     GgQuaternion& operator=(GgQuaternion&& q) = default;
03534
03540     GLfloat norm() const
03541     {
03542         return ggLength4(*this);
03543     }
03544
03554     GgQuaternion& loadAdd(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03555     {
03556         (*this)[0] += x;
03557         (*this)[1] += y;
03558         (*this)[2] += z;
03559         (*this)[3] += w;
03560
03561         return *this;
03562     }
03563
03570     GgQuaternion& loadAdd(const GLfloat* a)
03571     {
03572         return loadAdd(a[0], a[1], a[2], a[3]);
03573     }
03574
03581     GgQuaternion& loadAdd(const GgVector& v)
03582     {
03583         return loadAdd(v[0], v[1], v[2], v[3]);
03584     }
03585
03592     GgQuaternion& loadAdd(const GgQuaternion& q)
03593     {
03594         return loadAdd(q[0], q[1], q[2], q[3]);
03595     }
03596
03606     GgQuaternion& loadSubtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03607     {
03608         (*this)[0] -= x;
03609         (*this)[1] -= y;
03610         (*this)[2] -= z;
03611         (*this)[3] -= w;
03612
03613         return *this;

```

```
03614     }
03615
03622     GgQuaternion& loadSubtract(const GLfloat* a)
03623     {
03624         return loadSubtract(a[0], a[1], a[2], a[3]);
03625     }
03626
03633     GgQuaternion& loadSubtract(const GgVector& v)
03634     {
03635         return loadSubtract(v[0], v[1], v[2], v[3]);
03636     }
03637
03644     GgQuaternion& loadSubtract(const GgQuaternion& q)
03645     {
03646         return loadSubtract(q[0], q[1], q[2], q[3]);
03647     }
03648
03655     GgQuaternion& loadMultiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03656     {
03657         const GLfloat a[] { x, y, z, w };
03658         return loadMultiply(a);
03659     }
03663
03670     GgQuaternion& loadMultiply(const GLfloat* a)
03671     {
03672         return *this = multiply(a);
03673     }
03674
03681     GgQuaternion& loadMultiply(const GgVector& v)
03682     {
03683         return loadMultiply(v.data());
03684     }
03685
03692     GgQuaternion& loadMultiply(const GgQuaternion& q)
03693     {
03694         return loadMultiply(q.data());
03695     }
03696
03706     GgQuaternion& loadDivide(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03707     {
03708         const GLfloat a[] { x, y, z, w };
03709         return loadDivide(a);
03710     }
03711
03718     GgQuaternion& loadDivide(const GLfloat* a)
03719     {
03720         return *this = divide(a);
03721     }
03722
03729     GgQuaternion& loadDivide(const GgVector& v)
03730     {
03731         return loadDivide(v.data());
03732     }
03733
03740     GgQuaternion& loadDivide(const GgQuaternion& q)
03741     {
03742         return loadDivide(q.data());
03743     }
03744
03754     GgQuaternion add(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03755     {
03756         GgQuaternion s{ *this };
03757         return s.loadAdd(x, y, z, w);
03758     }
03759
03766     GgQuaternion add(const GLfloat* a) const
03767     {
03768         return add(a[0], a[1], a[2], a[3]);
03769     }
03770
03777     GgQuaternion add(const GgVector& v) const
03778     {
03779         return add(v[0], v[1], v[2], v[3]);
03780     }
03781
03788     GgQuaternion add(const GgQuaternion& q) const
03789     {
03790         return add(q[0], q[1], q[2], q[3]);
03791     }
03792
03802     GgQuaternion subtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03803     {
03804         GgQuaternion s{ *this };
03805         return s.loadSubtract(x, y, z, w);
03806     }
03807
03814     GgQuaternion subtract(const GLfloat* a) const
```

```

03815    {
03816        return subtract(a[0], a[1], a[2], a[3]);
03817    }
03818
03825    GgQuaternion subtract(const GgVector& v) const
03826    {
03827        return subtract(v[0], v[1], v[2], v[3]);
03828    }
03829
03836    GgQuaternion subtract(const GgQuaternion& q) const
03837    {
03838        return subtract(q[0], q[1], q[2], q[3]);
03839    }
03840
03850    GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03851    {
03852        const GLfloat a[] { x, y, z, w };
03853        return multiply(a);
03854    }
03855
03862    GgQuaternion multiply(const GLfloat* a) const
03863    {
03864        GgQuaternion s;
03865        multiply(s.data(), data(), a);
03866        return s;
03867    }
03868
03875    GgQuaternion multiply(const GgVector& v) const
03876    {
03877        return multiply(v.data());
03878    }
03879
03886    GgQuaternion multiply(const GgQuaternion& q) const
03887    {
03888        return multiply(q.data());
03889    }
03890
03900    GgQuaternion divide(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03901    {
03902        const GLfloat a[] { x, y, z, w };
03903        return divide(a);
03904    }
03905
03912    GgQuaternion divide(const GLfloat* a) const
03913    {
03914        GgQuaternion s, ia;
03915        ia.loadInvert(a);
03916        multiply(s.data(), data(), ia.data());
03917        return s;
03918    }
03919
03926    GgQuaternion divide(const GgVector& v) const
03927    {
03928        return divide(v.data());
03929    }
03930
03937    GgQuaternion divide(const GgQuaternion& q) const
03938    {
03939        return divide(q.data());
03940    }
03941
03942 // 演算子
03943 GgQuaternion& operator=(const GLfloat* a)
03944 {
03945     return *this = GgQuaternion(a);
03946 }
03947 GgQuaternion& operator=(const GgVector& v)
03948 {
03949     return *this = GgQuaternion(v);
03950 }
03951 GgQuaternion& operator+=(const GLfloat* a)
03952 {
03953     return loadAdd(a);
03954 }
03955 GgQuaternion& operator+=(const GgVector& v)
03956 {
03957     return loadAdd(v);
03958 }
03959 GgQuaternion& operator+=(const GgQuaternion& q)
03960 {
03961     return loadAdd(q);
03962 }
03963 GgQuaternion& operator-=(const GLfloat* a)
03964 {
03965     return loadSubtract(a);
03966 }
03967 GgQuaternion& operator-=(const GgVector& v)

```

```
03968     {
03969         return loadSubtract(v);
03970     }
03971     GgQuaternion& operator==(const GgQuaternion& q)
03972     {
03973         return operator==(q.data());
03974     }
03975     GgQuaternion& operator*=(const GLfloat* a)
03976     {
03977         return loadMultiply(a);
03978     }
03979     GgQuaternion& operator*=(const GgVector& v)
03980     {
03981         return loadMultiply(v);
03982     }
03983     GgQuaternion& operator*=(const GgQuaternion& q)
03984     {
03985         return operator*=(q.data());
03986     }
03987     GgQuaternion& operator/=(const GLfloat* a)
03988     {
03989         return loadDivide(a);
03990     }
03991     GgQuaternion& operator/=(const GgVector& v)
03992     {
03993         return loadDivide(v);
03994     }
03995     GgQuaternion& operator/=(const GgQuaternion& q)
03996     {
03997         return operator/=(q.data());
03998     }
03999     GgQuaternion operator+(const GLfloat* a) const
04000     {
04001         return add(a);
04002     }
04003     GgQuaternion operator+(const GgVector& v) const
04004     {
04005         return add(v);
04006     }
04007     GgQuaternion operator+(const GgQuaternion& q) const
04008     {
04009         return operator+(q.data());
04010     }
04011     GgQuaternion operator-(const GLfloat* a) const
04012     {
04013         return add(a);
04014     }
04015     GgQuaternion operator-(const GgVector& v) const
04016     {
04017         return add(v);
04018     }
04019     GgQuaternion operator-(const GgQuaternion& q) const
04020     {
04021         return operator-(q.data());
04022     }
04023     GgQuaternion operator*(const GLfloat* a) const
04024     {
04025         return multiply(a);
04026     }
04027     GgQuaternion operator*(const GgVector& v) const
04028     {
04029         return multiply(v);
04030     }
04031     GgQuaternion operator*(const GgQuaternion& q) const
04032     {
04033         return operator*(q.data());
04034     }
04035     GgQuaternion operator/(const GLfloat* a) const
04036     {
04037         return divide(a);
04038     }
04039     GgQuaternion operator/(const GgVector& v) const
04040     {
04041         return divide(v);
04042     }
04043     GgQuaternion operator/(const GgQuaternion& q) const
04044     {
04045         return operator/(q.data());
04046     }
04047
04048     GgQuaternion& loadMatrix(const GLfloat* a)
04049     {
04050         toQuaternion(data(), a);
04051         return *this;
04052     }
04053
04054     GgQuaternion& loadMatrix(const GgMatrix& m)
```

```

04067  {
04068      return loadMatrix(m.get());
04069  }
04070
04076  GgQuaternion& loadIdentity()
04077  {
04078     return *this = GgQuaternion(0.0f, 0.0f, 0.0f, 1.0f);
04079  }
04080
04090  GgQuaternion& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
04091
04099  GgQuaternion& loadRotate(const GLfloat* v, GLfloat a)
04100  {
04101     return loadRotate(v[0], v[1], v[2], a);
04102  }
04103
04110  GgQuaternion& loadRotate(const GLfloat* v)
04111  {
04112     return loadRotate(v[0], v[1], v[2], v[3]);
04113  }
04114
04121  GgQuaternion& loadRotateX(GLfloat a);
04122
04129  GgQuaternion& loadRotateY(GLfloat a);
04130
04137  GgQuaternion& loadRotateZ(GLfloat a);
04138
04148  GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
04149  {
04150     GgQuaternion q;
04151     return multiply(q.loadRotate(x, y, z, a));
04152  }
04153
04161  GgQuaternion rotate(const GLfloat* v, GLfloat a) const
04162  {
04163     return rotate(v[0], v[1], v[2], a);
04164  }
04165
04172  GgQuaternion rotate(const GLfloat* v) const
04173  {
04174     return rotate(v[0], v[1], v[2], v[3]);
04175  }
04176
04183  GgQuaternion rotateX(GLfloat a) const
04184  {
04185     return rotate(1.0f, 0.0f, 0.0f, a);
04186  }
04187
04194  GgQuaternion rotateY(GLfloat a) const
04195  {
04196     return rotate(0.0f, 1.0f, 0.0f, a);
04197  }
04198
04205  GgQuaternion rotateZ(GLfloat a) const
04206  {
04207     return rotate(0.0f, 0.0f, 1.0f, a);
04208  }
04209
04218  GgQuaternion& loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll);
04219
04226  GgQuaternion& loadEuler(const GLfloat* e)
04227  {
04228     return loadEuler(e[0], e[1], e[2]);
04229  }
04230
04239  GgQuaternion euler(GLfloat heading, GLfloat pitch, GLfloat roll) const
04240  {
04241     GgQuaternion r;
04242     return multiply(r.loadEuler(heading, pitch, roll));
04243  }
04244
04251  GgQuaternion euler(const GLfloat* e) const
04252  {
04253     return euler(e[0], e[1], e[2]);
04254  }
04255
04264  GgQuaternion& loadSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04265  {
04266     slerp(data(), a, b, t);
04267     return *this;
04268  }
04269
04278  GgQuaternion& loadSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04279  {
04280     return loadSlerp(q.data(), r.data(), t);
04281  }
04282

```

```
04291     GgQuaternion& loadSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04292     {
04293         return loadSlerp(q.data(), a, t);
04294     }
04295
04304     GgQuaternion& loadSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04305     {
04306         return loadSlerp(a, q.data(), t);
04307     }
04308
04315     GgQuaternion& loadNormalize(const GLfloat* a);
04316
04323     GgQuaternion& loadNormalize(const GgQuaternion& q)
04324     {
04325         return loadNormalize(q.data());
04326     }
04327
04334     GgQuaternion& loadConjugate(const GLfloat* a);
04335
04342     GgQuaternion& loadConjugate(const GgQuaternion& q)
04343     {
04344         return loadConjugate(q.data());
04345     }
04346
04353     GgQuaternion& loadInvert(const GLfloat* a);
04354
04361     GgQuaternion& loadInvert(const GgQuaternion& q)
04362     {
04363         return loadInvert(q.data());
04364     }
04365
04373     GgQuaternion slerp(GLfloat* a, GLfloat t) const
04374     {
04375         GgQuaternion p;
04376         slerp(p.data(), data(), a, t);
04377         return p;
04378     }
04379
04387     GgQuaternion slerp(const GgQuaternion& q, GLfloat t) const
04388     {
04389         GgQuaternion p;
04390         slerp(p.data(), data(), q.data(), t);
04391         return p;
04392     }
04393
04399     GgQuaternion normalize() const
04400     {
04401         GgQuaternion q;
04402         q.loadNormalize(data());
04403         return q;
04404     }
04405
04411     GgQuaternion conjugate() const
04412     {
04413         GgQuaternion q;
04414         q.loadConjugate(data());
04415         return q;
04416     }
04417
04423     GgQuaternion invert() const
04424     {
04425         GgQuaternion q;
04426         q.loadInvert(data());
04427         return q;
04428     }
04429
04435     void get(GLfloat* a) const
04436     {
04437         a[0] = data()[0];
04438         a[1] = data()[1];
04439         a[2] = data()[2];
04440         a[3] = data()[3];
04441     }
04442
04448     void getMatrix(GLfloat* a) const
04449     {
04450         toMatrix(a, data());
04451     }
04452
04458     void getMatrix(GgMatrix& m) const
04459     {
04460         getMatrix(m.data());
04461     }
04462
04468     GgMatrix getMatrix() const
04469     {
04470         GgMatrix m;
```

```
04471     getMatrix(m);
04472     return m;
04473 }
04474
04475 void getConjugateMatrix(GLfloat* a) const
04476 {
04477     GgQuaternion c;
04478     c.loadConjugate(data());
04479     toMatrix(a, c.data());
04480 }
04481
04482 void getConjugateMatrix(GgMatrix& m) const
04483 {
04484     getConjugateMatrix(m.data());
04485 }
04486
04487 GgMatrix getConjugateMatrix() const
04488 {
04489     GgMatrix m;
04490     getConjugateMatrix(m);
04491     return m;
04492 }
04493
04494 inline GgQuaternion ggIdentityQuaternion()
04495 {
04496     GgQuaternion q;
04497     return q.loadIdentity();
04498 }
04499
04500 inline GgQuaternion ggMatrixQuaternion(const GLfloat* a)
04501 {
04502     GgQuaternion q;
04503     return q.loadMatrix(a);
04504 }
04505
04506 inline GgQuaternion ggMatrixQuaternion(const GgMatrix& m)
04507 {
04508     return ggMatrixQuaternion(m.get());
04509 }
04510
04511 inline GgMatrix ggQuaternionMatrix(const GgQuaternion& q)
04512 {
04513     GLfloat m[16];
04514     q.getMatrix(m);
04515     return GgMatrix{ m };
04516 }
04517
04518 inline GgMatrix ggQuaternionTransposeMatrix(const GgQuaternion& q)
04519 {
04520     GLfloat m[16];
04521     q.getMatrix(m);
04522     GgMatrix t;
04523     return t.loadTranspose(m);
04524 }
04525
04526 inline GgQuaternion ggRotateQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
04527 {
04528     GgQuaternion q;
04529     return q.loadRotate(x, y, z, a);
04530 }
04531
04532 inline GgQuaternion ggRotateQuaternion(const GLfloat* v, GLfloat a)
04533 {
04534     return ggRotateQuaternion(v[0], v[1], v[2], a);
04535 }
04536
04537 inline GgQuaternion ggRotateQuaternion(const GLfloat* v)
04538 {
04539     return ggRotateQuaternion(v[0], v[1], v[2], v[3]);
04540 }
04541
04542 inline GgQuaternion ggEulerQuaternion(GLfloat heading, GLfloat pitch, GLfloat roll)
04543 {
04544     GgQuaternion q;
04545     return q.loadEuler(heading, pitch, roll);
04546 }
04547
04548 inline GgQuaternion ggEulerQuaternion(const GLfloat* e)
04549 {
04550     return ggEulerQuaternion(e[0], e[1], e[2]);
04551 }
04552
04553 inline GgQuaternion ggSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04554 {
04555     GgQuaternion r;
04556     return r.loadSlerp(a, b, t);
```

```
04646 }
04647
04656 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04657 {
04658     return ggSlerp(q.data(), r.data(), t);
04659 }
04660
04669 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04670 {
04671     return ggSlerp(q.data(), a, t);
04672 }
04673
04682 inline GgQuaternion ggSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04683 {
04684     return ggSlerp(a, q.data(), t);
04685 }
04686
04693 inline GLfloat ggNorm(const GgQuaternion& q)
04694 {
04695     return q.norm();
04696 }
04697
04704 inline GgQuaternion ggNormalize(const GgQuaternion& q)
04705 {
04706     return q.normalize();
04707 }
04708
04715 inline GgQuaternion ggConjugate(const GgQuaternion& q)
04716 {
04717     return q.conjugate();
04718 }
04719
04726 inline GgQuaternion ggInvert(const GgQuaternion& q)
04727 {
04728     return q.invert();
04729 }
04730
04734 class GgTrackball : public GgQuaternion
04735 {
04736     bool drag;           // ドラッグ中か否か
04737     GLfloat start[2];   // ドラッグ開始位置
04738     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
04739     GgQuaternion cq;    // 回転の初期値（四元数）
04740     GgMatrix rt;        // 回転の変換行列
04741
04742 public:
04743
04749     GgTrackball(const GgQuaternion& q = ggIdentityQuaternion())
04750     {
04751         reset(q);
04752     }
04753
04757     ~GgTrackball() = default;
04758
04764     GgTrackball& operator=(const GgQuaternion& q)
04765     {
04766         GgQuaternion::operator=(q);
04767         return *this;
04768     }
04769
04779     void region(GLfloat w, GLfloat h);
04780
04790     void region(int w, int h)
04791     {
04792         region(static_cast<GLfloat>(w), static_cast<GLfloat>(h));
04793     }
04794
04804     void begin(GLfloat x, GLfloat y);
04805
04815     void motion(GLfloat x, GLfloat y);
04816
04822     void rotate(const GgQuaternion& q);
04823
04833     void end(GLfloat x, GLfloat y);
04834
04840     void reset(const GgQuaternion& q = ggIdentityQuaternion());
04841
04847     const GLfloat* getStart() const
04848     {
04849         return start;
04850     }
04851
04855     const GLfloat& getStart(int direction) const
04856     {
04857         return start[direction];
04858     }
04859
```

```
04865 void getStart(GLfloat* position) const
04866 {
04867     position[0] = start[0];
04868     position[1] = start[1];
04869 }
04870
04876 const GLfloat* getScale() const
04877 {
04878     return scale;
04879 }
04880
04886 const GLfloat getScale(int direction) const
04887 {
04888     return scale[direction];
04889 }
04890
04896 void getScale(GLfloat* factor) const
04897 {
04898     factor[0] = scale[0];
04899     factor[1] = scale[1];
04900 }
04901
04907 const GgQuaternion& getQuaternion() const
04908 {
04909     return *this;
04910 }
04911
04917 const GgMatrix& getMatrix() const
04918 {
04919     return rt;
04920 }
04921
04927 const GLfloat* get() const
04928 {
04929     return rt.get();
04930 }
04931 };
04932
04943 extern bool ggSaveTga(
04944     const std::string& name,
04945     const void* buffer,
04946     unsigned int width,
04947     unsigned int height,
04948     unsigned int depth
04949 );
04950
04955 extern bool ggSaveColor(const std::string& name);
04958
04965 extern bool ggSaveDepth(const std::string& name);
04966
04977 extern bool ggReadImage(
04978     const std::string& name,
04979     std::vector<GLubyte>& image,
04980     GLsizei* pWidth,
04981     GLsizei* pHeight,
04982     GLenum* pFormat
04983 );
04984
04998 extern GLuint ggLoadTexture(
05000     const GLvoid* image,
05001     GLsizei width,
05002     GLsizei height,
05003     GLenum format = GL_RGB,
05004     GLenum type = GL_UNSIGNED_BYTE,
05005     GLenum internal = GL_RGB,
05006     GLenum wrap = GL_CLAMP_TO_EDGE,
05007     bool swizzle = true
05008 );
05009
05019 extern GLuint ggLoadImage(
05020     const std::string& name,
05021     GLsizei* pWidth = nullptr,
05022     GLsizei* pHeight = nullptr,
05023     GLenum internal = 0,
05024     GLenum wrap = GL_CLAMP_TO_EDGE
05025 );
05026
05038 extern void ggCreateNormalMap(
05039     const GLubyte* hmap,
05040     GLsizei width,
05041     GLsizei height,
05042     GLenum format,
05043     GLfloat nz,
05044     GLenum internal,
05045     std::vector<GgVector>& nmap
05046 );
05047
```

```
05058     extern GLuint ggLoadHeight(
05059         const std::string& name,
05060         GLfloat nz,
05061         GLsizei* pWidth = nullptr,
05062         GLsizei* pHeight = nullptr,
05063         GLenum internal = GL_RGBA
05064     );
05065
05079     extern GLuint ggCreateShader(
05080         const std::string& vsrc,
05081         const std::string& fsrc = "",
05082         const std::string& gsrc = "",
05083         GLint nvarying = 0,
05084         const char* const* varyings = nullptr,
05085         const std::string& vtext = "vertex shader",
05086         const std::string& ftext = "fragment shader",
05087         const std::string& gtext = "geometry shader");
05088
05099     extern GLuint ggLoadShader(
05100         const std::string& vert,
05101         const std::string& frag = "",
05102         const std::string& geom = "",
05103         GLint nvarying = 0,
05104         const char* const* varyings = nullptr
05105     );
05106
05115     inline GLuint ggLoadShader(
05116         const std::array<std::string, 3>& files,
05117         GLint nvarying = 0,
05118         const char* const* varyings = nullptr
05119     )
05120     {
05121         return ggLoadShader(files[0], files[1], files[2], nvarying, varyings);
05122     }
05123
05124 #if !defined(_APPLE_)
05132     extern GLuint ggCreateComputeShader(
05133         const std::string& csrc,
05134         const std::string& ctext = "compute shader"
05135     );
05136
05143     extern GLuint ggLoadComputeShader(const std::string& comp);
05144 #endif
05145
05152     class GgTexture
05153     {
05154         // テクスチャ名
05155         GLuint texture;
05156
05157         // テクスチャの縦横の画素数
05158         GLsizei size[2];
05159
05160     public:
05161
05174         GgTexture(
05175             const GLvoid* image,
05176             GLsizei width,
05177             GLsizei height,
05178             GLenum format = GL_RGB,
05179             GLenum type = GL_UNSIGNED_BYTE,
05180             GLenum internal = GL_RGBA,
05181             GLenum wrap = GL_CLAMP_TO_EDGE,
05182             bool swizzle = true
05183         ) :
05184             texture{ ggLoadTexture(image, width, height, format, type, internal, wrap, swizzle) },
05185             size{ width, height }
05186         {
05187     }
05188
05192     GgTexture(const GgTexture& texture) = delete;
05193
05197     GgTexture(GgTexture&& texture) = default;
05198
05202     virtual ~GgTexture()
05203     {
05204         glBindTexture(GL_TEXTURE_2D, 0);
05205         glDeleteTextures(1, &texture);
05206     }
05207
05211     GgTexture& operator=(const GgTexture& texture) = delete;
05212
05216     GgTexture& operator=(GgTexture&& texture) = default;
05217
05221     void bind() const
05222     {
05223         glBindTexture(GL_TEXTURE_2D, texture);
05224     }
```

```

05225
05229     void unbind() const
05230     {
05231         glBindTexture(GL_TEXTURE_2D, 0);
05232     }
05233
05239     void swapRandB(bool swizzle) const;
05240
05246     const GLsizei& getWidth() const
05247     {
05248         return size[0];
05249     }
05250
05256     const GLsizei& getHeight() const
05257     {
05258         return size[1];
05259     }
05260
05266     void getSize(GLsizei* size) const
05267     {
05268         size[0] = getWidth();
05269         size[1] = getHeight();
05270     }
05271
05277     const GLsizei* getSize() const
05278     {
05279         return size;
05280     }
05281
05287     const GLuint& getTexture() const
05288     {
05289         return texture;
05290     }
05291 };
05292
05299 class GgColorTexture
05300 {
05301     // テクスチャ
05302     std::shared_ptr<GgTexture> texture;
05303
05304 public:
05305
05309     GgColorTexture()
05310     {
05311     }
05312
05325     GgColorTexture(
05326         const GLvoid* image,
05327         GLsizei width,
05328         GLsizei height,
05329         GLenum format = GL_RGB,
05330         GLenum type = GL_UNSIGNED_BYTE,
05331         GLenum internal = GL_RGB,
05332         GLenum wrap = GL_CLAMP_TO_EDGE,
05333         bool swizzle = true
05334     )
05335     {
05336         load(image, width, height, format, type, internal, wrap, swizzle);
05337     }
05338
05346     GgColorTexture(
05347         const std::string& name,
05348         GLenum internal = 0,
05349         GLenum wrap = GL_CLAMP_TO_EDGE
05350     )
05351     {
05352         load(name, internal, wrap);
05353     }
05354
05358     virtual ~GgColorTexture()
05359     {
05360     }
05361
05374     void load(
05375         const GLvoid* image,
05376         GLsizei width,
05377         GLsizei height,
05378         GLenum format = GL_RGB,
05379         GLenum type = GL_UNSIGNED_BYTE,
05380         GLenum internal = GL_RGB,
05381         GLenum wrap = GL_CLAMP_TO_EDGE,
05382         bool swizzle = true
05383     )
05384     {
05385         // テクスチャを作成する
05386         texture = std::make_shared<GgTexture>(image, width, height, format, type, internal, wrap,
05387             swizzle);

```

```
05387     }
05388
05396     void load(
05397         const std::string& name,
05398         GLenum internal = 0,
05399         GLenum wrap = GL_CLAMP_TO_EDGE
05400     );
05401 };
05402
05409 class GgNormalTexture
05410 {
05411     // テクスチャ
05412     std::shared_ptr<GgTexture> texture;
05413
05414 public:
05415
05419     GgNormalTexture()
05420     {
05421     }
05422
05433     GgNormalTexture(
05434         const GLubyte* image,
05435         GLsizei width,
05436         GLsizei height,
05437         GLenum format = GL_RED,
05438         GLfloat nz = 1.0f,
05439         GLenum internal = GL_RGBA
05440     )
05441     {
05442         // 法線マップのテクスチャを作成する
05443         load(image, width, height, format, nz, internal);
05444     }
05445
05453     GgNormalTexture(
05454         const std::string& name,
05455         GLfloat nz = 1.0f,
05456         GLenum internal = GL_RGBA
05457     )
05458     {
05459         // 法線マップのテクスチャを作成する
05460         load(name, nz, internal);
05461     }
05462
05466     virtual ~GgNormalTexture()
05467     {
05468     }
05469
05480     void load(
05481         const GLubyte* hmap,
05482         GLsizei width,
05483         GLsizei height,
05484         GLenum format = GL_RED,
05485         GLfloat nz = 1.0f,
05486         GLenum internal = GL_RGBA
05487     )
05488     {
05489         // 法線マップ
05490         std::vector<GgVector> nmap;
05491
05492         // 法線マップを作成する
05493         ggCreateNormalMap(hmap, width, height, format, nz, internal, nmap);
05494
05495         // テクスチャを作成する
05496         texture = std::make_shared<GgTexture>(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal,
05497             GL_REPEAT);
05498     }
05506     void load(
05507         const std::string& name,
05508         GLfloat nz = 1.0f,
05509         GLenum internal = GL_RGBA
05510     );
05511 };
05512
05519 template <typename T>
05520 class GgBuffer
05521 {
05522     // ターゲット
05523     const GLenum target;
05524
05525     // 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
05526     const GLsizei stride;
05527
05528     // データの数
05529     const GLsizei count;
05530
05531     // バッファオブジェクト
```

```

05532     const GLuint buffer;
05533
05534 public:
05535
05545     GgBuffer<T>(
05546         GLenum target,
05547         const T* data,
05548         GLsizei stride,
05549         GLsizei count,
05550         GLenum usage
05551     ) :
05552         target{ target },
05553         stride{ stride },
05554         count{ count },
05555         buffer{ [] { GLuint buffer; glGenBuffers(1, &buffer); return buffer; } () }
05556     {
05557         // 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
05558         glBindBuffer(target, buffer);
05559         glBufferData(target, getStride() * count, data, usage);
05560     }
05561
05565     GgBuffer<T>(const GgBuffer<T>& buffer) = delete;
05566
05570     GgBuffer<T>(GgBuffer<T>&& buffer) = default;
05571
05575     virtual ~GgBuffer<T>()
05576     {
05577         // バッファオブジェクトを削除する
05578         glBindBuffer(target, 0);
05579         glDeleteBuffers(1, &buffer);
05580     }
05581
05585     GgBuffer<T>& operator=(const GgBuffer<T>& buffer) = delete;
05586
05590     GgBuffer<T>& operator=(GgBuffer<T>&& buffer) = default;
05591
05597     const GLuint& getTarget() const
05598     {
05599         return target;
05600     }
05601
05607     GLsizei* getStride() const
05608     {
05609         return static_cast<GLsizei*>(stride);
05610     }
05611
05617     const GLsizei& getCount() const
05618     {
05619         return count;
05620     }
05621
05627     const GLuint& getBuffer() const
05628     {
05629         return buffer;
05630     }
05631
05635     void bind() const
05636     {
05637         glBindBuffer(target, buffer);
05638     }
05639
05643     void unbind() const
05644     {
05645         glBindBuffer(target, 0);
05646     }
05647
05653     void* map() const
05654     {
05655         glBindBuffer(target, buffer);
05656 #if defined(GL_GLES_PROTOTYPES)
05657         return glMapBufferRange(target, 0, getCount() * count, GL_MAP_WRITE_BIT);
05658 #else
05659         return glMapBuffer(target, GL_WRITE_ONLY);
05660 #endif
05661     }
05662
05670     void* map(GLint first, GLsizei count) const
05671     {
05672         // count が 0 なら全データをマップする
05673         if (count == 0) count = getCount();
05674         if (first + count > getCount()) count = getCount() - first;
05675
05676         glBindBuffer(target, buffer);
05677         return glMapBufferRange(target, getStride() * first, getStride() * count, GL_MAP_WRITE_BIT);
05678     }
05679
05683     void unmap() const

```

```
05684  {
05685     glUnmapBuffer(target);
05686 }
05687
05695 void send(const T* data, GLint first, GLsizei count) const
05696 {
05697     // count が 0 なら全データを転送する
05698     if (count == 0) count = getCount();
05699     if (first + count > getCount()) count = getCount() - first;
05700
05701     // データを既存のバッファオブジェクトに転送する
05702     glBindBuffer(target, buffer);
05703     glBufferSubData(target, getStride() * first, getStride() * count, data);
05704 }
05705
05713 void read(T* data, GLint first, GLsizei count) const
05714 {
05715     // count が 0 なら全データを抽出する
05716     if (count == 0) count = getCount();
05717     if (first + count > getCount()) count = getCount() - first;
05718
05719     // データをバッファオブジェクトから抽出する
05720     glBindBuffer(target, buffer);
05721 #if defined(GL_GLES_PROTOTYPES)
05722     const GLsizeiptr begin{ getStride() * first };
05723     const GLsizeiptr range{ getStride() * count };
05724     T* const source{ glMapBufferRange(target, begin, range, GL_MAP_READ_BIT) };
05725     std::copy(source, source + count, data);
05726     glUnmapBuffer(target);
05727 #else
05728     glGetBufferSubData(target, getStride() * first, getStride() * count, data);
05729 #endif
05730 }
05731
05740 void copy(GLuint src_buffer, GLint src_first = 0, GLint dst_first = 0, GLsizei count = 0) const
05741 {
05742     // count が 0 なら全データを複写する
05743     if (count == 0) count = getCount();
05744     if (src_first + count > getCount()) count = getCount() - src_first;
05745     if (dst_first + count > getCount()) count = getCount() - dst_first;
05746
05747     // データの間隔
05748     const GLsizeiptr stride{ getStride() };
05749
05750     glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
05751     glBindBuffer(GL_COPY_WRITE_BUFFER, buffer);
05752     glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
05753         stride * src_first, stride * dst_first, stride * count);
05754     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
05755     glBindBuffer(GL_COPY_READ_BUFFER, 0);
05756 }
05757
05758
05765 template <typename T>
05766 class GgUniformBuffer
05767 {
05768     // ユニフォームバッファオブジェクト
05769     std::shared_ptr<GgBuffer<T>> uniform;
05770
05771 public:
05772
05773     GgUniformBuffer<T>()
05774     {
05775     }
05776
05777     GgUniformBuffer<T>(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05778     {
05779         load(data, count, usage);
05780     }
05781
05782     GgUniformBuffer<T>(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05783     {
05784         load(data, count, usage);
05785     }
05786
05787     virtual ~GgUniformBuffer<T>()
05788     {
05789     }
05790
05791     const GLuint& getTarget() const
05792     {
05793         return uniform->getTarget();
05794     }
05795
05796     GLsizeiptr getStride() const
05797     {
05798         return uniform->getStride();
05799     }
```

```

05829 }
05830
05831 const GLsizei& getCount() const
05832 {
05833     return uniform->getCount();
05834 }
05835
05836 const GLuint& getBuffer() const
05837 {
05838     return uniform->getBuffer();
05839 }
05840
05841
05842 void bind() const
05843 {
05844     uniform->bind();
05845 }
05846
05847 void unbind() const
05848 {
05849     uniform->unbind();
05850 }
05851
05852 void* map() const
05853 {
05854     return uniform->map();
05855 }
05856
05857 void* map(GLint first, GLsizei count) const
05858 {
05859     return uniform->map(first, count);
05860 }
05861
05862 void unmap() const
05863 {
05864     uniform->unmap();
05865 }
05866
05867
05868 void load(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05869 {
05870     // バッファオブジェクト上のデータの間隔
05871     const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) *
05872         ggBufferAlignment };
05873
05874     // ユニフォームバッファオブジェクトを確保する
05875     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05876
05877     // 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
05878     if (data) send(data, 0, sizeof(T), 0, count);
05879 }
05880
05881
05882 void load(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05883 {
05884     // バッファオブジェクト上のデータの間隔
05885     const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) *
05886         ggBufferAlignment };
05887
05888     // ユニフォームバッファオブジェクトを確保する
05889     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05890
05891     // 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
05892     fill(&data, 0, sizeof(T), 0, count);
05893 }
05894
05895
05896 void send(
05897     const GLvoid* data,
05898     GLint offset = 0,
05899     GLsizei size = sizeof(T),
05900     GLint first = 0,
05901     GLsizei count = 0
05902 ) const
05903 {
05904     // count が 0 なら全データを転送する
05905     if (count == 0) count = getCount();
05906     if (first + count > getCount()) count = getCount() - first;
05907
05908     // 転送元のデータの先頭
05909     const char* source{ reinterpret_cast<const char*>(data) };
05910
05911     // ターゲット
05912     const GLuint target{ getTarget() };
05913
05914     // データの間隔
05915     const GLsizeiptr stride{ getStride() };
05916
05917     // 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
05918     bind();
05919     for (GLsizei i = 0; i < count; ++i)

```

```
05968     {
05969         glBufferSubData(target, stride * (first + i) + offset, size, source + size * i);
05970     }
05971 }
05972
05973 void fill(
05974     const GLvoid* data,
05975     GLint offset = 0,
05976     GLsizei size = sizeof(T),
05977     GLint first = 0,
05978     GLsizei count = 0
05979 ) const
05980 {
05981     // count が 0 なら全データを転送する
05982     if (count == 0) count = getCount();
05983     if (first + count > getCount()) count = getCount() - first;
05984
05985     // ターゲット
05986     const GLuint target{ getTarget() };
05987
05988     // データの間隔
05989     const GLsizeiptr stride{ getStride() };
05990
05991     // 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
05992     bind();
05993     for (GLsizei i = 0; i < count; ++i)
05994     {
05995         glBufferSubData(target, stride * (first + i) + offset, size, data);
05996     }
05997 }
05998
05999 void read(
06000     GLvoid* data,
06001     GLint offset = 0,
06002     GLsizei size = sizeof(T),
06003     GLint first = 0,
06004     GLsizei count = 0
06005 ) const
06006 {
06007     // count が 0 なら全データを転送する
06008     if (count == 0) count = getCount();
06009     if (first + count > getCount()) count = getCount() - first;
06010
06011     // 抽出先のデータの先頭
06012     char* const destination{ reinterpret_cast<char*>(data) };
06013
06014     // ターゲット
06015     const GLuint target{ getTarget() };
06016
06017     // データの間隔
06018     const GLsizeiptr stride{ getStride() };
06019
06020     // 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
06021     bind();
06022 #if defined(GL_GLES_PROTOTYPES)
06023     const char* const source{ glMapBufferRange(target, stride * first, stride * count,
06024         GL_MAP_READ_BIT) };
06025     for (GLsizei i = 0; i < count; ++i)
06026     {
06027         const char* const begin{ source + stride * i + offset };
06028         const char* const end{ begin + size };
06029         std::copy(begin, end, destination + sizeof(T) * i);
06030     }
06031     glUnmapBuffer(target);
06032 #else
06033     for (GLsizei i = 0; i < count; ++i)
06034     {
06035         glGetBufferSubData(target, stride * (first + i) + offset, size, destination + sizeof(T) * i);
06036     }
06037 #endif
06038 }
06039
06040 void copy(
06041     GLuint srcbuffer,
06042     GLint srcFirst = 0,
06043     GLint dstFirst = 0,
06044     GLsizei count = 0
06045 ) const
06046 {
06047     // count が 0 なら全データを複写する
06048     if (count == 0) count = getCount();
06049     if (srcFirst + count > getCount()) count = getCount() - srcFirst;
06050     if (dstFirst + count > getCount()) count = getCount() - dstFirst;
06051
06052     // データの間隔
06053     const GLsizeiptr stride{ getStride() };
06054 }
```

```

06080 // 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
06081 glBindBuffer(GL_COPY_READ_BUFFER, src.buffer);
06082 glBindBuffer(GL_COPY_WRITE_BUFFER, getBuffer());
06083 for (GLsizei i = 0; i < count; ++i)
06084 {
06085     glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
06086                         stride * (src.first + i), stride * (dst.first + i), sizeof(T));
06087 }
06088 glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
06089 glBindBuffer(GL_COPY_READ_BUFFER, 0);
06090 }
06091 };
06092
06093 class GgVertexArray
06094 {
06095     // 頂点配列オブジェクト
06096     const GLuint vao;
06097
06098 public:
06099
06100     GgVertexArray(GLenum mode = 0) :
06101         vao{ [] { GLuint vao; glGenVertexArrays(1, &vao); return vao; } () }
06102     {
06103         glBindVertexArray(vao);
06104     }
06105
06106     GgVertexArray(const GgVertexArray& array) = delete;
06107
06108     GgVertexArray(GgVertexArray&& array) = default;
06109
06110     virtual ~GgVertexArray()
06111     {
06112         glBindVertexArray(0);
06113         glDeleteVertexArrays(1, &vao);
06114     }
06115
06116     GgVertexArray& operator=(const GgVertexArray& array) = delete;
06117
06118     GgVertexArray& operator=(GgVertexArray&& array) = default;
06119
06120     const GLuint& get() const
06121     {
06122         return vao;
06123     }
06124
06125     void bind() const
06126     {
06127         glBindVertexArray(vao);
06128     }
06129 };
06130
06131 class GgShape
06132 {
06133     // 頂点配列オブジェクト
06134     std::shared_ptr<GgVertexArray> object;
06135
06136     // 基本図形の種類
06137     GLenum mode;
06138
06139 public:
06140
06141     GgShape(GLenum mode = 0) :
06142         object{ std::make_shared<GgVertexArray>(),
06143                 mode{ mode } }
06144     {
06145     }
06146
06147     virtual ~GgShape()
06148     {
06149     }
06150
06151     virtual explicit operator bool() const noexcept
06152     {
06153         return object.get() != nullptr;
06154     }
06155
06156     virtual bool operator!() const noexcept
06157     {
06158         return !static_cast<bool>(*this);
06159     }
06160
06161     const GLuint& get() const
06162     {
06163         return object->get();
06164     }
06165
06166     void setMode(GLenum mode)
06167
06168
06169
06170
06171
06172
06173
06174
06175
06176
06177
06178
06179
06180
06181
06182
06183
06184
06185
06186
06187
06188
06189
06190
06191
06192
06193
06194
06195
06196
06197
06198
06199
06200
06201
06202
06203
06204
06205
06206
06207
06208
06209
06210
06211
06212
06213
06214
06215
06216
06217
06218
06219
06220
06221
06222
06223
06224
06225
06226
06227
06228
06229
06230
06231
06232
06233
06234
06235
06236
06237
06238
06239
06240
06241
06242
06243
06244
06245
06246
06247
06248
06249
06250
06251
06252
06253
06254
06255
06256
06257
06258
06259
06260
06261
06262
06263
06264
06265
06266
06267
06268
06269
06270
06271
06272
06273
06274
06275
06276
06277
06278
06279
06280
06281
06282
06283
06284
06285
06286
06287
06288
06289
06290
06291
06292
06293
06294
06295
06296
06297
06298
06299
06300
06301
06302
06303
06304
06305
06306
06307
06308
06309
06310
06311
06312
06313
06314
06315
06316
06317
06318
06319
06320
06321
06322
06323
06324
06325
06326
06327
06328
06329
06330
06331
06332
06333
06334
06335
06336
06337
06338
06339
06340
06341
06342
06343
06344
06345
06346
06347
06348
06349
06350
06351
06352
06353
06354
06355
06356
06357
06358
06359
06360
06361
06362
06363
06364
06365
06366
06367
06368
06369
06370
06371
06372
06373
06374
06375
06376
06377
06378
06379
06380
06381
06382
06383
06384
06385
06386
06387
06388
06389
06390
06391
06392
06393
06394
06395
06396
06397
06398
06399
06400
06401
06402
06403
06404
06405
06406
06407
06408
06409
06410
06411
06412
06413
06414
06415
06416
06417
06418
06419
06420
06421
06422
06423
06424
06425
06426
06427
06428
06429
06430
06431
06432
06433
06434
06435
06436
06437
06438
06439
06440
06441
06442
06443
06444
06445
06446
06447
06448
06449
06450
06451
06452
06453
06454
06455
06456
06457
06458
06459
06460
06461
06462
06463
06464
06465
06466
06467
06468
06469
06470
06471
06472
06473
06474
06475
06476
06477
06478
06479
06480
06481
06482
06483
06484
06485
06486
06487
06488
06489
06490
06491
06492
06493
06494
06495
06496
06497
06498
06499
06500
06501
06502
06503
06504
06505
06506
06507
06508
06509
06510
06511
06512
06513
06514
06515
06516
06517
06518
06519
06520
06521
06522
06523
06524
06525
06526
06527
06528
06529
06530
06531
06532
06533
06534
06535
06536
06537
06538
06539
06540
06541
06542
06543
06544
06545
06546
06547
06548
06549
06550
06551
06552
06553
06554
06555
06556
06557
06558
06559
06560
06561
06562
06563
06564
06565
06566
06567
06568
06569
06570
06571
06572
06573
06574
06575
06576
06577
06578
06579
06580
06581
06582
06583
06584
06585
06586
06587
06588
06589
06590
06591
06592
06593
06594
06595
06596
06597
06598
06599
06600
06601
06602
06603
06604
06605
06606
06607
06608
06609
06610
06611
06612
06613
06614
06615
06616
06617
06618
06619
06620
06621
06622
06623
06624
06625
06626
06627
06628
06629
06630
06631
06632
06633
06634
06635
06636
06637
06638
06639
06640
06641
06642
06643
06644
06645
06646
06647
06648
06649
06650
06651
06652
06653
06654
06655
06656
06657
06658
06659
06660
06661
06662
06663
06664
06665
06666
06667
06668
06669
06670
06671
06672
06673
06674
06675
06676
06677
06678
06679
06680
06681
06682
06683
06684
06685
06686
06687
06688
06689
06690
06691
06692
06693
06694
06695
06696
06697
06698
06699
06700
06701
06702
06703
06704
06705
06706
06707
06708
06709
06710
06711
06712
06713
06714
06715
06716
06717
06718
06719
06720
06721
06722
06723
06724
06725
06726
06727
06728
06729
06730
06731
06732
06733
06734
06735
06736
06737
06738
06739
06740
06741
06742
06743
06744
06745
06746
06747
06748
06749
06750
06751
06752
06753
06754
06755
06756
06757
06758
06759
06760
06761
06762
06763
06764
06765
06766
06767
06768
06769
06770
06771
06772
06773
06774
06775
06776
06777
06778
06779
06780
06781
06782
06783
06784
06785
06786
06787
06788
06789
06790
06791
06792
06793
06794
06795
06796
06797
06798
06799
06800
06801
06802
06803
06804
06805
06806
06807
06808
06809
06810
06811
06812
06813
06814
06815
06816
06817
06818
06819
06820
06821
06822
06823
06824
06825
06826
06827
06828
06829
06830
06831
06832
06833
06834
06835
06836
06837
06838
06839
06840
06841
06842
06843
06844
06845
06846
06847
06848
06849
06850
06851
06852
06853
06854
06855
06856
06857
06858
06859
06860
06861
06862
06863
06864
06865
06866
06867
06868
06869
06870
06871
06872
06873
06874
06875
06876
06877
06878
06879
06880
06881
06882
06883
06884
06885
06886
06887
06888
06889
06890
06891
06892
06893
06894
06895
06896
06897
06898
06899
06900
06901
06902
06903
06904
06905
06906
06907
06908
06909
06910
06911
06912
06913
06914
06915
06916
06917
06918
06919
06920
06921
06922
06923
06924
06925
06926
06927
06928
06929
06930
06931
06932
06933
06934
06935
06936
06937
06938
06939
06940
06941
06942
06943
06944
06945
06946
06947
06948
06949
06950
06951
06952
06953
06954
06955
06956
06957
06958
06959
06960
06961
06962
06963
06964
06965
06966
06967
06968
06969
06970
06971
06972
06973
06974
06975
06976
06977
06978
06979
06980
06981
06982
06983
06984
06985
06986
06987
06988
06989
06990
06991
06992
06993
06994
06995
06996
06997
06998
06999
06999

```

```
06233  {
06234      this->mode = mode;
06235 }
06236
06242 const GLenum& getMode() const
06243 {
06244     return this->mode;
06245 }
06246
06253 virtual void draw(GLint first = 0, GLsizei count = 0) const
06254 {
06255     object->bind();
06256 }
06257 };
06258
06262 class GgPoints
06263 : public GgShape
06264 {
// 頂点バッファオブジェクト
06265     std::shared_ptr<GgBuffer<GgVector>> position;
06266
06268 public:
06269
06273     GgPoints(GLenum mode = GL_POINTS) :
06274         GgShape(mode)
06275     {
06276     }
06277
06286     GgPoints(
06287         const GgVector* pos,
06288         GLsizei countv,
06289         GLenum mode = GL_POINTS,
06290         GLenum usage = GL_STATIC_DRAW
06291     ) :
06292         GgPoints(mode)
06293     {
06294         load(pos, countv, usage);
06295     }
06296
06300     virtual ~GgPoints()
06301     {
06302     }
06303
06309     explicit operator bool() const noexcept
06310     {
06311         return position.get() != nullptr;
06312     }
06313
06319     bool operator!() const noexcept
06320     {
06321         return !static_cast<bool>(*this);
06322     }
06323
06329     const GLsizei& getCount() const
06330     {
06331         return position->getCount();
06332     }
06333
06339     const GLuint& getBuffer() const
06340     {
06341         return position->getBuffer();
06342     }
06343
06351     void send(const GgVector* pos, GLint first = 0, GLsizei count = 0) const
06352     {
06353         position->send(pos, first, count);
06354     }
06355
06363     void load(const GgVector* pos, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06364
06371     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06372 };
06373
06377 struct GgVertex
06378 {
06380     GgVector position;
06381
06383     GgVector normal;
06384
06388     GgVertex()
06389     {
06390     }
06391
06398     GgVertex(const GgVector& pos, const GgVector& norm) :
06399         position(pos),
06400         normal(norm)
06401     {
```

```

06402     }
06403
06414     GgVertex(
06415         GLfloat px, GLfloat py, GLfloat pz,
06416         GLfloat nx, GLfloat ny, GLfloat nz
06417     ) :
06418         position{ px, py, pz, 1.0f },
06419         normal{ nx, ny, nz, 0.0f }
06420     {
06421     }
06422
06429     GgVertex(const GLfloat* pos, const GLfloat* norm) :
06430         GgVertex(pos[0], pos[1], pos[2], norm[0], norm[1], norm[2])
06431     {
06432     }
06433 };
06434
06438     class GgTriangles
06439         : public GgShape
06440     {
06441         // 頂点属性
06442         std::shared_ptr<GgBuffer<GgVertex>> vertex;
06443
06444     public:
06445
06451         GgTriangles(GLenum mode = GL_TRIANGLES) :
06452             GgShape(mode)
06453         {
06455
06464         GgTriangles(
06465             const GgVertex* vert,
06466             GLsizei count,
06467             GLenum mode = GL_TRIANGLES,
06468             GLenum usage = GL_STATIC_DRAW
06469         ) :
06470             GgTriangles(mode)
06471         {
06472             load(vert, count, usage);
06473         }
06474
06478         virtual ~GgTriangles()
06479     {
06480     }
06481
06487         const GLsizei& getCount() const
06488     {
06489         return vertex->getCount();
06490     }
06491
06497         const GLuint& getBuffer() const
06498     {
06499         return vertex->getBuffer();
06500     }
06501
06509         void send(const GgVertex* vert, GLint first = 0, GLsizei count = 0) const
06510     {
06511         vertex->send(vert, first, count);
06512     }
06513
06521         void load(const GgVertex* vert, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06522
06529         virtual void draw(GLint first = 0, GLsizei count = 0) const;
06530     };
06531
06535     class GgElements
06536         : public GgTriangles
06537     {
06538         // インデックスを格納する頂点バッファオブジェクト
06539         std::shared_ptr<GgBuffer<GLuint>> index;
06540
06541     public:
06542
06548         GgElements(GLenum mode = GL_TRIANGLES) :
06549             GgTriangles(mode)
06550         {
06551         }
06552
06563         GgElements(
06564             const GgVertex* vert,
06565             GLsizei countv,
06566             const GLuint* face,
06567             GLsizei countf,
06568             GLenum mode = GL_TRIANGLES,
06569             GLenum usage = GL_STATIC_DRAW
06570         ) :
06571             GgElements(mode)

```

```
06572     {
06573         load(vert, countv, face, countf, usage);
06574     }
06575
06579     virtual ~GgElements()
06580     {
06581     }
06582
06587     const GLsizei& getIndexCount() const
06588     {
06589         return index->getCount();
06590     }
06591
06597     const GLuint& getIndexBuffer() const
06598     {
06599         return index->getBuffer();
06600     }
06601
06612     void send(
06613         const GgVertex* vert,
06614         GLuint firstv,
06615         GLsizei countv,
06616         const GLuint* face = nullptr,
06617         GLuint firstf = 0,
06618         GLsizei countf = 0
06619     ) const
06620     {
06621         GgTriangles::send(vert, firstv, countv);
06622         if (face != nullptr && countf > 0) index->send(face, firstf, countf);
06623     }
06624
06634     void load(
06635         const GgVertex* vert,
06636         GLsizei countv,
06637         const GLuint* face,
06638         GLsizei countf,
06639         GLenum usage = GL_STATIC_DRAW
06640     )
06641     {
06642         // 頂点バッファオブジェクトを作成する
06643         GgTriangles::load(vert, countv, usage);
06644
06645         // インデックスの頂点バッファオブジェクトを作成する
06646         index = std::make_shared<GgBuffer<GLuint>>(GL_ELEMENT_ARRAY_BUFFER, face,
06647             static_cast<GLsizei>(sizeof(GLuint)), countf, usage);
06648     }
06649
06655     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06656 };
06657
06668     extern std::shared_ptr<GgPoints> ggPointsCube(
06669         GLsizei countv,
06670         GLfloat length = 1.0f,
06671         GLfloat cx = 0.0f,
06672         GLfloat cy = 0.0f,
06673         GLfloat cz = 0.0f
06674     );
06675
06686     extern std::shared_ptr<GgPoints> ggPointsSphere(
06687         GLsizei countv,
06688         GLfloat radius = 0.5f,
06689         GLfloat cx = 0.0f,
06690         GLfloat cy = 0.0f,
06691         GLfloat cz = 0.0f
06692     );
06693
06701     extern std::shared_ptr<GgTriangles> ggRectangle(
06702         GLfloat width = 1.0f,
06703         GLfloat height = 1.0f
06704     );
06705
06714     extern std::shared_ptr<GgTriangles> ggEllipse(
06715         GLfloat width = 1.0f,
06716         GLfloat height = 1.0f,
06717         GLuint slices = 16
06718     );
06719
06731     extern std::shared_ptr<GgTriangles> ggArraysObj(
06732         const std::string& name,
06733         bool normalize = false
06734     );
06735
06747     extern std::shared_ptr<GgElements> ggElementsObj(
06748         const std::string& name,
06749         bool normalize = false
06750     );
06751
```

```
06764     extern std::shared_ptr<GgElements> ggElementsMesh(
06765         GLuint slices,
06766         GLuint stacks,
06767         const GLfloat(*pos)[3],
06768         const GLfloat(*norm)[3] = nullptr
06769     );
06770
06781     extern std::shared_ptr<GgElements> ggElementsSphere(
06782         GLfloat radius = 1.0f,
06783         int slices = 16,
06784         int stacks = 8
06785     );
06786
06793     class GgShader
06794     {
06795         // プログラム名
06796         const GLuint program;
06797
06798     public:
06799
06809     GgShader(
06810         const std::string& vert,
06811         const std::string& frag = "",
06812         const std::string& geom = "",
06813         int nvarying = 0,
06814         const char* const* varyings = nullptr
06815     ) :
06816         program(ggLoadShader(vert, frag, geom, nvarying, varyings))
06817     {
06818     }
06819
06827     GgShader(
06828         const std::array<std::string, 3>& files,
06829         int nvarying = 0,
06830         const char* const* varyings = nullptr
06831     ) :
06832         GgShader(files[0], files[1], files[2], nvarying, varyings)
06833     {
06834     }
06835
06839     GgShader(const GgShader& shader) = delete;
06840
06844     GgShader(GgShader&& shader) = default;
06845
06849     virtual ~GgShader()
06850     {
06851         // 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
06852         glUseProgram(0);
06853         glDeleteProgram(program);
06854     }
06855
06859     GgShader& operator=(const GgShader& shader) = delete;
06860
06864     GgShader& operator=(GgShader&& shader) = default;
06865
06869     void use() const
06870     {
06871         glUseProgram(program);
06872     }
06873
06877     void unuse() const
06878     {
06879         glUseProgram(0);
06880     }
06881
06887     GLuint get() const
06888     {
06889         return program;
06890     }
06891 };
06892
06896     class GgPointShader
06897     {
06898         // シエーダー
06899         std::shared_ptr<GgShader> shader;
06900
06901         // 投影変換行列の uniform 変数の場所
06902         GLint mpLoc;
06903
06904         // モデルビュー変換行列の uniform 変数の場所
06905         GLint mvLoc;
06906
06907     public:
06908
06912     GgPointShader() :
06913         mpLoc{ -1 },
06914         mvLoc{ -1 }
```

```
06915  {
06916  }
06917
06927  GgPointShader(
06928      const std::string& vert,
06929      const std::string& frag = "",
06930      const std::string& geom = "",
06931      GLint nvarying = 0,
06932      const char* const* varyings = nullptr
06933  ) :
06934  GgPointShader()
06935  {
06936      load(vert, frag, geom, nvarying, varyings);
06937  }
06938
06946  GgPointShader(
06947      const std::array<std::string, 3>& files,
06948      int nvarying = 0,
06949      const char* const* varyings = nullptr
06950  ) :
06951  GgPointShader(files[0], files[1], files[2], nvarying, varyings)
06952  {
06953  }
06954
06958  virtual ~GgPointShader()
06959  {
06960  }
06961
06972  bool load(
06973      const std::string& vert,
06974      const std::string& frag = "",
06975      const std::string& geom = "",
06976      GLint nvarying = 0,
06977      const char* const* varyings = nullptr
06978  )
06979  {
06980      // シェーダを作成する
06981      shader = std::make_shared<GgShader>(vert, frag, geom, nvarying, varyings);
06982
06983      // プログラム名を取り出す
06984      const GLuint program(shader->get());
06985
06986      // プログラムオブジェクトが作成できていなければ戻る
06987      if (program == 0) return false;
06988
06989      // 変換行列の uniform 変数の場所
06990      mpLoc = glGetUniformLocation(program, "mp");
06991      mvLoc = glGetUniformLocation(program, "mv");
06992
06993      // プログラムオブジェクトの作成に成功した
06994      return true;
06995  }
06996
07005  bool load(
07006      const std::array<std::string, 3>& files,
07007      GLint nvarying = 0,
07008      const char* const* varyings = nullptr
07009  )
07010  {
07011      return load(files[0], files[1], files[2], nvarying, varyings);
07012  }
07013
07019  virtual void loadProjectionMatrix(const GLfloat* mp) const
07020  {
07021      glUniformMatrix4fv(mpLoc, 1, GL_FALSE, mp);
07022  }
07023
07029  virtual void loadModelviewMatrix(const GgMatrix& mp) const
07030  {
07031      loadProjectionMatrix(mp.get());
07032  }
07033
07039  virtual void loadModelviewMatrix(const GLfloat* mv) const
07040  {
07041      glUniformMatrix4fv(mvLoc, 1, GL_FALSE, mv);
07042  }
07043
07049  virtual void loadModelviewMatrix(const GgMatrix& mv) const
07050  {
07051      loadModelviewMatrix(mv.get());
07052  }
07053
07060  virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07061  {
07062      loadProjectionMatrix(mp);
07063      loadModelviewMatrix(mv);
07064  }
```

```
07065
07072     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
07073     {
07074         loadMatrix(mp.get(), mv.get());
07075     }
07076
07080     virtual void use() const
07081     {
07082         shader->use();
07083     }
07084
07089     void use(const GLfloat* mp) const
07090     {
07091         use();
07092         loadProjectionMatrix(mp);
07093     }
07094
07101     void use(const GgMatrix& mp) const
07102     {
07103         use(mp.get());
07104     }
07105
07112     void use(const GLfloat* mp, const GLfloat* mv) const
07113     {
07114         use(mp);
07115         loadModelviewMatrix(mv);
07116     }
07117
07124     void use(const GgMatrix& mp, const GgMatrix& mv) const
07125     {
07126         use(mp.get(), mv.get());
07127     }
07128
07132     void unuse() const
07133     {
07134         shader->unuse();
07135     }
07136
07142     GLuint get() const
07143     {
07144         return shader->get();
07145     }
07146 }
07147
07151 class GgSimpleShader
07152     : public GgPointShader
07153 {
07154     // 材質データの uniform block のインデックス
07155     GLint materialIndex;
07156
07157     // 光源データの uniform block のインデックス
07158     GLint lightIndex;
07159
07160     // モデルビュー変換の法線変換行列の uniform 変数の場所
07161     GLint mnLoc;
07162
07163 public:
07164
07168     GgSimpleShader() :
07169         GgPointShader(),
07170         materialIndex{ -1 },
07171         lightIndex{ -1 },
07172         mnLoc{ -1 }
07173     {}
07174 }
07175
07185     GgSimpleShader(
07186         const std::string& vert,
07187         const std::string& frag = "",
07188         const std::string& geom = "",
07189         GLint nvarying = 0,
07190         const char* const* varyings = nullptr
07191     )
07192     {
07193         load(vert, frag, geom, nvarying, varyings);
07194     }
07195
07203     GgSimpleShader(
07204         const std::array<std::string, 3>& files,
07205         GLint nvarying = 0,
07206         const char* const* varyings = nullptr
07207     ) :
07208         GgSimpleShader(files[0], files[1], files[2], nvarying, varyings)
07209     {}
07210
07211     GgSimpleShader(const GgSimpleShader& o) :
```

```
07216     GgPointShader(o),
07217     materialIndex{ o.materialIndex },
07218     lightIndex{ o.lightIndex },
07219     mnLoc{ o.mnLoc }
07220 {
07221 }
07222
07223     virtual ~GgSimpleShader()
07224 {
07225 }
07226
07227     GgSimpleShader& operator=(const GgSimpleShader& o)
07228 {
07229     if (&o != this)
07230     {
07231         GgPointShader::operator=(o);
07232         materialIndex = o.materialIndex;
07233         lightIndex = o.lightIndex;
07234         mnLoc = o.mnLoc;
07235     }
07236
07237     return *this;
07238 }
07239
07240     bool load(
07241         const std::string& vert,
07242         const std::string& frag = "",
07243         const std::string& geom = "",
07244         GLint nvarying = 0,
07245         const char* const* varyings = nullptr
07246     );
07247
07248     bool load(
07249         const std::array<std::string, 3>& files,
07250         GLint nvarying = 0,
07251         const char* const* varyings = nullptr
07252     );
07253     {
07254         return load(files[0], files[1], files[2], nvarying, varyings);
07255     }
07256
07257     virtual void loadModelviewMatrix(const GLfloat* mv, const GLfloat* mn) const
07258     {
07259         GgPointShader::loadModelviewMatrix(mv);
07260         glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07261     }
07262
07263     virtual void loadModelviewMatrix(const GgMatrix& mv, const GgMatrix& mn) const
07264     {
07265         loadModelviewMatrix(mv.get(), mn.get());
07266     }
07267
07268     virtual void loadModelviewMatrix(const GLfloat* mv) const
07269     {
07270         loadModelviewMatrix(mv, GgMatrix(mv).normal().get());
07271     }
07272
07273     virtual void loadModelviewMatrix(const GgMatrix& mv) const
07274     {
07275         loadModelviewMatrix(mv.get());
07276     }
07277
07278     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07279     {
07280         GgPointShader::loadMatrix(mp, mv);
07281         glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07282     }
07283
07284     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07285     {
07286         loadMatrix(mp.get(), mv.get(), mn.get());
07287     }
07288
07289     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07290     {
07291         loadMatrix(mp, mv, GgMatrix(mv).normal());
07292     }
07293
07294     struct Light
07295     {
07296         GgVector ambient;
07297         GgVector diffuse;
```

```
07378     GgVector specular;
07379     GgVector position;
07380 }
07381
07385 class LightBuffer
07386   : public GgUniformBuffer<Light>
07387 {
07388 public:
07389
07397     LightBuffer(
07398       const Light* light = nullptr,
07399       GLsizei count = 1,
07400       GLenum usage = GL_STATIC_DRAW
07401     ) :
07402     GgUniformBuffer<Light>(light, count, usage)
07403   {
07404   }
07405
07413     LightBuffer(
07414       const Light& light,
07415       GLsizei count = 1,
07416       GLenum usage = GL_STATIC_DRAW
07417     ) :
07418     GgUniformBuffer<Light>(light, count, usage)
07419   {
07420   }
07421
07425     virtual ~LightBuffer()
07426   {
07427   }
07428
07429     void loadAmbient(
07430       GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07431       GLint first = 0, GLsizei count = 1
07432     ) const;
07433
07451     void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07452
07460     void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07461   {
07462     // first 番目のブロックから count 個の ambient 要素に値を設定する
07463     send(ambient, offsetof(Light, ambient), sizeof(Light::ambient), first, count);
07464   }
07465
07476     void loadDiffuse(
07477       GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07478       GLint first = 0, GLsizei count = 1
07479     ) const;
07480
07488     void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07489
07497     void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07498   {
07499     // first 番目のブロックから count 個の diffuse 要素に値を設定する
07500     send(diffuse, offsetof(Light, diffuse), sizeof(Light::diffuse), first, count);
07501   }
07502
07513     void loadSpecular(
07514       GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07515       GLint first = 0, GLsizei count = 1
07516     ) const;
07517
07525     void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07526
07534     void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07535   {
07536     // first 番目のブロックから count 個の specular 要素に値を設定する
07537     send(specular, offsetof(Light, specular), sizeof(Light::specular), first, count);
07538   }
07539
07547     void loadColor(const Light& color, GLint first = 0, GLsizei count = 1) const;
07548
07559     void loadPosition(
07560       GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f,
07561       GLint first = 0, GLsizei count = 1
07562     ) const;
07563
07571     void loadPosition(const GgVector& position, GLint first = 0, GLsizei count = 1) const;
07572
07580     void loadPosition(const GLfloat* position, GLint first = 0, GLsizei count = 1) const
07581   {
07582     // first 番目のブロックから count 個の position 要素に値を設定する
07583     send(position, offsetof(Light, position), sizeof(Light::position), first, count);
07584   }
07585
07593     void loadPosition(const GgVector* position, GLint first = 0, GLsizei count = 1) const
07594   {
```

```
07595     loadPosition(position->data(), first, count);
07596 }
07597
07605 void load(const Light* light, GLint first = 0, GLsizei count = 1) const
07606 {
07607     send(light, 0, sizeof(Light), first, count);
07608 }
07609
07617 void load(const Light& light, GLint first = 0, GLsizei count = 1) const
07618 {
07619     load(&light, first, count);
07620 }
07621
07627 void select(GLint i = 0) const
07628 {
07629     // バッファオブジェクトの i 番目のブロックの位置
07630     const GLintptr offset(static_cast<GLintptr>(getStride()) * i);
07631     glBindBufferRange(getTarget(), LightBindingPoint, getBuffer(), offset, sizeof(Light));
07632 }
07633
07634
07638 struct Material
07639 {
07640     GgVector ambient;
07641     GgVector diffuse;
07642     GgVector specular;
07643     GLfloat shininess;
07644 };
07645
07649 class MaterialBuffer
07650     : public GgUniformBuffer<Material>
07651 {
07652     public:
07653
07661     MaterialBuffer(
07662         const Material* material = nullptr,
07663         GLsizei count = 1,
07664         GLenum usage = GL_STATIC_DRAW
07665     ) :
07666         GgUniformBuffer<Material>(material, count, usage)
07667     {}
07668
07669
07677     MaterialBuffer(
07678         const Material& material,
07679         GLsizei count = 1,
07680         GLenum usage = GL_STATIC_DRAW
07681     ) :
07682         GgUniformBuffer<Material>(material, count, usage)
07683     {}
07684
07685
07689     virtual ~MaterialBuffer()
07690     {}
07691
07692
07703     void loadAmbient(
07704         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07705         GLint first = 0, GLsizei count = 1
07706     ) const;
07707
07715     void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07716
07724 void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07725 {
07726     // 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
07727     send(ambient, offsetof(Material, ambient), sizeof(Material::ambient), first, count);
07728 }
07729
07740     void loadDiffuse(
07741         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07742         GLint first = 0, GLsizei count = 1
07743     ) const;
07744
07752     void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07753
07761 void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07762 {
07763     // first 番目のブロックから count 個の diffuse 要素に値を設定する
07764     send(diffuse, offsetof(Material, diffuse), sizeof(Material::diffuse), first, count);
07765 }
07766
07777     void loadAmbientAndDiffuse(
07778         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07779         GLint first = 0, GLsizei count = 1
07780     ) const;
```

```

07789 void loadAmbientAndDiffuse(const GgVector& color, GLint first = 0, GLsizei count = 1) const;
07790
07791 void loadAmbientAndDiffuse(const GLfloat* color, GLint first = 0, GLsizei count = 1) const;
07792
07793 void loadSpecular(
07794     GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07795     GLint first = 0, GLsizei count = 1
07796 ) const;
07797
07798 void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07799
07800 void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07801 {
07802     // first 番目のブロックから count 個の specular 要素に値を設定する
07803     send(specular, offsetof(Material, specular), sizeof(Material::specular), first, count);
07804 }
07805
07806 void loadShininess(GLfloat shininess, GLint first = 0, GLsizei count = 1) const;
07807
07808 void loadShininess(const GLfloat* shininess, GLint first = 0, GLsizei count = 1) const;
07809
07810 void load(const Material* material, GLint first = 0, GLsizei count = 1) const
07811 {
07812     send(material, 0, sizeof(Material), first, count);
07813 }
07814
07815 void load(const Material& material, GLint first = 0, GLsizei count = 1) const
07816 {
07817     load(&material, first, count);
07818 }
07819
07820 void select(GLint i = 0) const
07821 {
07822     // バッファオブジェクトの i 番目のブロックの位置
07823     const GLintptr offset{ static_cast<GLintptr>(getStride()) * i };
07824     glBindBufferRange(getTarget(), MaterialBindingPoint, getBuffer(), offset, sizeof(Material));
07825 }
07826
07827 void use() const
07828 {
07829     // プログラムオブジェクトは基底クラスで指定する
07830     GgPointShader::use();
07831 }
07832
07833 void use(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07834 {
07835     // プログラムオブジェクトを指定する
07836     use();
07837
07838     // 変換行列を設定する
07839     loadMatrix(mp, mv, mn);
07840 }
07841
07842 void use(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07843 {
07844     use(mp.get(), mv.get(), mn.get());
07845 }
07846
07847 void use(const GLfloat* mp, const GLfloat* mv) const
07848 {
07849     use(mp, mv, GgMatrix(mv).normal().get());
07850 }
07851
07852 void use(const GgMatrix& mp, const GgMatrix& mv) const
07853 {
07854     use(mp, mv, mv.normal());
07855 }
07856
07857 void use(const LightBuffer* light, GLint i = 0) const
07858 {
07859     // プログラムオブジェクトを指定する
07860     use();
07861
07862     // 光源を設定する
07863     light->select(i);
07864 }
07865
07866 void use(const LightBuffer& light, GLint i = 0) const
07867 {
07868     use(&light, i);
07869 }
07870
07871 void use(
07872     const GLfloat* mp,
07873     const GLfloat* mv,
07874     const GLfloat* mn,
07875

```

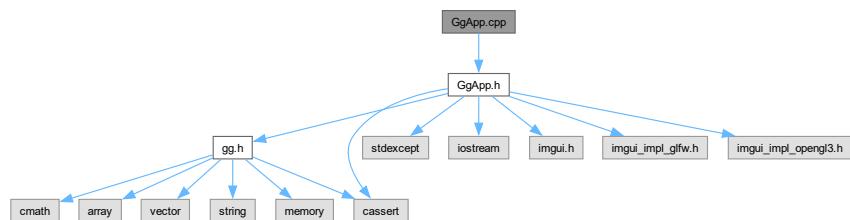
```
07990     const LightBuffer* light,
07991     GLint i = 0
07992 } const
07993 {
07994     // 光源を指定してプログラムオブジェクトを指定する
07995     use(light, i);
07996
07997     // 変換行列を設定する
07998     loadMatrix(mp, mv, mn);
07999 }
08000
08010 void use(
08011     const GgMatrix& mp,
08012     const GgMatrix& mv,
08013     const GgMatrix& mn,
08014     const LightBuffer& light,
08015     GLint i = 0
08016 ) const
08017 {
08018     use(mp.get(), mv.get(), mn.get(), &light, i);
08019 }
08020
08029 void use(
08030     const GLfloat* mp,
08031     const GLfloat* mv,
08032     const LightBuffer* light,
08033     GLint i = 0
08034 ) const
08035 {
08036     use(mp, mv, GgMatrix(mv).normal().get(), light, i);
08037 }
08038
08047 void use(
08048     const GgMatrix& mp,
08049     const GgMatrix& mv,
08050     const LightBuffer& light,
08051     GLint i = 0
08052 ) const
08053 {
08054     use(mp, mv, mv.normal(), light, i);
08055 }
08056
08064 void use(const GLfloat* mp, const LightBuffer* light, GLint i = 0) const
08065 {
08066     // 光源を指定してプログラムオブジェクトを指定する
08067     use(light, i);
08068
08069     // 投影変換行列を設定する
08070     loadProjectionMatrix(mp);
08071 }
08072
08080 void use(const GgMatrix& mp, const LightBuffer& light, GLint i = 0) const
08081 {
08082     // 光源を指定してプログラムオブジェクトを指定する
08083     use(mp.get(), &light, i);
08084 }
08085 };
08086
08097 extern bool ggLoadSimpleObj(
08098     const std::string& name,
08099     std::vector<std::array<GLuint, 3>>& group,
08100     std::vector<GgSimpleShader::Material>& material,
08101     std::vector<GgVertex>& vert,
08102     bool normalize = false
08103 );
08104
08116 extern bool ggLoadSimpleObj(
08117     const std::string& name,
08118     std::vector<std::array<GLuint, 3>>& group,
08119     std::vector<GgSimpleShader::Material>& material,
08120     std::vector<GgVertex>& vert,
08121     std::vector<GLuint>& face,
08122     bool normalize = false
08123 );
08124
08128 class GgSimpleObj
08129 {
08130     // 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
08131     std::shared_ptr<std::vector<std::array<GLuint, 3>>> group;
08132
08133     // ポリゴングループごとの材質のユニフォームバッファ
08134     std::shared_ptr<GgSimpleShader::MaterialBuffer> material;
08135
08136     // この図形の形状データ
08137     std::shared_ptr<GgElements> data;
08138
08139 public:
```

```

08140
08147     GgSimpleObj(const std::string& name, bool normalize = false);
08148
08151     virtual ~GgSimpleObj()
08152     {
08153     }
08154
08160     explicit operator bool() const noexcept
08161     {
08162         return data.get() != nullptr;
08163     }
08164
08170     bool operator!() const noexcept
08171     {
08172         return !static_cast<bool>(*this);
08173     }
08174
08180     const GgTriangles* get() const
08181     {
08182         return data.get();
08183     }
08184
08191     virtual void draw(GLint first = 0, GLsizei count = 0) const;
08192 };
08193 }
```

9.13 GgApp.cpp ファイル

#include "GgApp.h"
GgApp.cpp の依存先関係図:



9.13.1 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

July 27, 2025

GgApp.cpp に定義があります。

9.14 GgApp.cpp

[詳解]

```
[GLFW]
00001 /*
00002
00003 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) 2011-2025 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
00010 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00011 copies or substantial portions of the Software.
00012
00013 The above copyright notice and this permission notice shall be included in
00014 all copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
00019 KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
00020 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
00021 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00022
00023 */
00024
00032 #include "GgApp.h"
00033
00034 //
00035 // GLFW のエラー表示
00036 //
00037 static void glfwErrorCallback(int error, const char* description)
00038 {
00039 #if defined(__aarch64__)
00040     if (error == 65544) return;
00041 #endif
00042     throw std::runtime_error(description);
00043 }
00044
00045 //
00046 // GgApp クラスのコンストラクタ
00047 //
00048 GgApp::GgApp(int major, int minor)
00049 {
00050     // GLFW のエラー処理関数を登録する
00051     glfwSetErrorCallback(glfwErrorCallback);
00052
00053     // GLFW を初期化する
00054     if (glfwInit() == GL_FALSE) throw std::runtime_error("Can't initialize GLFW");
00055
00056     // OpenGL の major 番号が指定されていれば
00057     if (major > 0)
00058     {
00059         // OpenGL のバージョンを指定する
00060         glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, major);
00061         glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, minor);
00062
00063 #if defined(GL_GLES_PROTOTYPES)
00064     // OpenGL ES 3 のコンテキストを指定する
00065     glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
00066     glfwWindowHint(GLFW_CONTEXT_CREATION_API, GLFW_EGL_CONTEXT_API);
00067 #else
00068     // OpenGL Version 3.2 以降なら
00069     if (major * 10 + minor >= 32)
00070     {
00071         // Core Profile を選択する (macOS の都合)
00072         glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00073         glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00074     }
00075 #endif
00076 }
00077
00078 #if defined(GG_USE_OOCULUS_RIFT)
00079     // Oculus Rift では SRGB でレンダリングする
00080     glfwWindowHint(GLFW_SRGB_CAPABLE, GL_TRUE);
00081 #endif
00082
00083 #if defined(IMGUI_VERSION)
00084     // ImGui のバージョンをチェックする
00085     IMGUI_CHECKVERSION();
00086
00087     // ImGui のコンテキストを作成する
00088     ImGui::CreateContext();
00089 #endif
```

```

00090 }
00091 //
00092 // デストラクタ
00093 //
00094 //
00095 GgApp::~GgApp()
00096 {
00097 #if defined(IMGUI_VERSION)
00098 // Shutdown Platform/Renderer bindings
00099 ImGui_ImplOpenGL3_Shutdown();
00100 ImGui_ImplGlfw_Shutdown();
00101 ImGui::DestroyContext();
00102 #endif
00103
00104 // プログラム終了時に GLFW を終了する
00105 glfwTerminate();
00106 }
00107
00108 //
00109 // マウスや矢印キーによる平行移動量を初期化する
00110 //
00111 void GgApp::Window::HumanInterface::resetTranslation()
00112 {
00113 // 平行移動量を初期化する
00114 for (auto& t : translation)
00115 {
00116 std::fill(t.begin(), t.end(), GgVector{ 0.0f, 0.0f, 0.0f, 1.0f });
00117 }
00118
00119 // 矢印キーの設定値を初期化する
00120 std::fill(arrow.begin(), arrow.end(), std::array<int, 2>{ 0, 0 });
00121
00122 // マウスホイールの回転量を初期化する
00123 std::fill(wheel.begin(), wheel.end(), 0.0f);
00124 }
00125
00126 //
00127 // 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)
00128 //
00129 void GgApp::Window::HumanInterface::calcTranslation(int button, const std::array<GLfloat, 3>& velocity)
00130 {
00131 // マウスの相対変位
00132 assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00133 const auto dx{ (mouse[0] - rotation[button].getStart(0)) * rotation[button].getScale(0) };
00134 const auto dy{ (rotation[button].getStart(1) - mouse[1]) * rotation[button].getScale(1) };
00135
00136 // 平行移動量
00137 auto& t{ translation[button] };
00138
00139 // 平行移動量の更新
00140 t[1][0] = dx * velocity[0] + t[0][0];
00141 t[1][1] = dy * velocity[1] + t[0][1];
00142
00143 // 回転量の更新
00144 rotation[button].motion(mouse[0], mouse[1]);
00145 }
00146
00147 //
00148 // ウィンドウのサイズ変更時の処理
00149 //
00150 void GgApp::Window::resize(GLFWwindow* window, int width, int height)
00151 {
00152 // このインスタンスの this ポインタを得る
00153 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00154
00155 if (instance)
00156 {
00157 // ウィンドウのサイズを保存する
00158 instance->size[0] = width;
00159 instance->size[1] = height;
00160
00161 // トラックボール処理の範囲を設定する
00162 for (auto& current_if : instance->interfaceData)
00163 {
00164 for (auto& t : current_if.rotation)
00165 {
00166 t.region(width, height);
00167 }
00168 }
00169
00170 // ビューポートを更新する
00171 instance->updateViewport();
00172
00173 // ユーザー定義のコールバック関数の呼び出し
00174 if (instance->resizeFunc) (*instance->resizeFunc)(instance, width, height);
00175 }

```

```
00176 }
00177
00178 // キーボードをタイプした時の処理
00179 // キーボードをタイプした時の処理
00180 //
00181 void GgApp::Window::keyboard(GLFWwindow* window, int key, int scancode, int action, int mods)
00182 {
00183 #if defined(IMGUI_VERSION)
00184     // 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
00185     if (ImGui::GetIO().WantCaptureKeyboard) return;
00186 #endif
00187
00188     // このインスタンスの this ポインタを得る
00189     auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00190
00191     if (instance && action)
00192     {
00193         // ユーザー定義のコールバック関数の呼び出し
00194         if (instance->keyboardFunc) (*instance->keyboardFunc)(instance, key, scancode, action, mods);
00195
00196         // 対象のユーザインターフェース
00197         auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00198
00199         switch (key)
00200         {
00201             case GLFW_KEY_HOME:
00202
00203                 // トラックボールを初期化する
00204                 instance->resetRotation();
00205                 [[fallthrough]];
00206
00207             case GLFW_KEY_END:
00208
00209                 // 平行移動量を初期化する
00210                 instance->resetTranslation();
00211                 break;
00212
00213             case GLFW_KEY_UP:
00214
00215                 if (mods & GLFW_MOD_SHIFT)
00216                     current_if.arrow[1][1]++;
00217                 else if (mods & GLFW_MOD_CONTROL)
00218                     current_if.arrow[2][1]++;
00219                 else if (mods & GLFW_MOD_ALT)
00220                     current_if.arrow[3][1]++;
00221                 else
00222                     current_if.arrow[0][1]++;
00223                 break;
00224
00225             case GLFW_KEY_DOWN:
00226
00227                 if (mods & GLFW_MOD_SHIFT)
00228                     current_if.arrow[1][1]--;
00229                 else if (mods & GLFW_MOD_CONTROL)
00230                     current_if.arrow[2][1]--;
00231                 else if (mods & GLFW_MOD_ALT)
00232                     current_if.arrow[3][1]--;
00233                 else
00234                     current_if.arrow[0][1]--;
00235                 break;
00236
00237             case GLFW_KEY_RIGHT:
00238
00239                 if (mods & GLFW_MOD_SHIFT)
00240                     current_if.arrow[1][0]++;
00241                 else if (mods & GLFW_MOD_CONTROL)
00242                     current_if.arrow[2][0]++;
00243                 else if (mods & GLFW_MOD_ALT)
00244                     current_if.arrow[3][0]++;
00245                 else
00246                     current_if.arrow[0][0]++;
00247                 break;
00248
00249             case GLFW_KEY_LEFT:
00250
00251                 if (mods & GLFW_MOD_SHIFT)
00252                     current_if.arrow[1][0]--;
00253                 else if (mods & GLFW_MOD_CONTROL)
00254                     current_if.arrow[2][0]--;
00255                 else if (mods & GLFW_MOD_ALT)
00256                     current_if.arrow[3][0]--;
00257                 else
00258                     current_if.arrow[0][0]--;
00259                 break;
00260
00261             default:
00262                 break;
00263 }
```

```

00263     }
00264     current_if.lastKey = key;
00265   }
00266 }
00267 }
00268
00269 // マウスボタンを操作したときの処理
00270 //
00271 void GgApp::Window::mouse(GLFWwindow* window, int button, int action, int mods)
00272 {
00273 #if defined(IMGUI_VERSION)
00274   // 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
00275   if (ImGui::GetIO().WantCaptureMouse) return;
00276 #endif
00277
00278 // このインスタンスの this ポインタを得る
00279 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00280
00281 // マウスボタンの状態を記録する
00282 assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00283 instance->status[button] = action != GLFW_RELEASE;
00284
00285 if (instance)
00286 {
00287   // ユーザー定義のコールバック関数の呼び出し
00288   if (instance->mouseFunc) (*instance->mouseFunc)(instance, button, action, mods);
00289
00290   // 対象のユーザインターフェース
00291   auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00292
00293   // マウスの現在位置を得る
00294   const auto x{ current_if.mouse[0] };
00295   const auto y{ current_if.mouse[1] };
00296
00297   if (x < 0 || x >= instance->size[0] || y < 0 || y >= instance->size[1]) return;
00298
00299   if (action)
00300   {
00301     // ドラッグ開始
00302     current_if.rotation[button].begin(x, y);
00303   }
00304   else
00305   {
00306     // ドラッグ終了
00307     current_if.translation[button][0] = current_if.translation[button][1];
00308     current_if.rotation[button].end(x, y);
00309   }
00310 }
00311 }
00312 }
00313
00314 // マウスホイールを操作した時の処理
00315 //
00316 void GgApp::Window::wheel(GLFWwindow* window, double x, double y)
00317 {
00318 #if defined(IMGUI_VERSION)
00319   // 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
00320   if (ImGui::GetIO().WantCaptureMouse) return;
00321 #endif
00322
00323 // このインスタンスの this ポインタを得る
00324 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00325
00326 if (instance)
00327 {
00328   // ユーザー定義のコールバック関数の呼び出し
00329   if (instance->wheelFunc) (*instance->wheelFunc)(instance, x, y);
00330
00331   // 対象のユーザインターフェース
00332   auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00333
00334   // マウスホイールの回転量の保存
00335   current_if.wheel[0] += static_cast<GLfloat>(x);
00336   current_if.wheel[1] += static_cast<GLfloat>(y);
00337
00338   // マウスによる平行移動量の z 値の更新
00339   const auto z{ current_if.wheel[1] * instance->velocity[2] };
00340   for (auto& t : current_if.translation) t[1][2] = z;
00341 }
00342 }
00343 }
00344
00345 //
00346 // Window クラスのコンストラクタ
00347 //
00348 GgApp::Window::Window(const std::string& title, int width, int height, int fullscreen, GLFWwindow* share) :

```

```
00349     window{ nullptr },
00350     size{ width, height },
00351     fboSize{ width, height },
00352 #if defined(IMGUI_VERSION)
00353     menubarHeight{ 0 },
00354 #endif
00355     aspect{ 1.0f },
00356     velocity{ 1.0f, 1.0f, 0.1f },
00357     status{ false },
00358     interfaceNo{ 0 },
00359     userPointer{ nullptr },
00360     resizeFunc{ nullptr },
00361     keyboardFunc{ nullptr },
00362     mouseFunc{ nullptr },
00363     wheelFunc{ nullptr }
00364 {
00365     // ディスプレイの情報
00366     GLFWmonitor* monitor{ nullptr };
00367
00368     // フルスクリーン表示
00369     if (fullscreen > 0)
00370     {
00371         // 接続されているモニタの数を数える
00372         int mcount;
00373         auto** const monitors{ glfwGetMonitors(&mcount) };
00374
00375         // セカンダリモニタがあればそれを使う
00376         if (fullscreen > mcount) fullscreen = mcount;
00377         monitor = monitors[fullscreen - 1];
00378
00379         // モニタのモードを調べる
00380         const auto* mode{ glfwGetVideoMode(monitor) };
00381
00382         // ウィンドウのサイズをディスプレイのサイズにする
00383         width = mode->width;
00384         height = mode->height;
00385     }
00386
00387     // GLFW のウィンドウを作成する
00388     window = glfwCreateWindow(width, height, title.c_str(), monitor, share);
00389
00390     // ウィンドウが作成できなければエラー
00391     if (!window) throw std::runtime_error("Unable to open the GLFW window.");
00392
00393     // 現在のウィンドウを処理対象にする
00394     glfwMakeContextCurrent(window);
00395
00396     // ゲームグラフィックス特論の都合による初期化を行う
00397     ggInit();
00398
00399     // このインスタンスの this ポインタを記録しておく
00400     glfwSetWindowUserPointer(window, this);
00401
00402     // キーボードを操作した時の処理を登録する
00403     glfwSetKeyCallback(window, keyboard);
00404
00405     // マウスボタンを操作したときの処理を登録する
00406     glfwSetMouseButtonCallback(window, mouse);
00407
00408     // マウスホイール操作時に呼び出す処理を登録する
00409     glfwSetScrollCallback(window, wheel);
00410
00411     // ウィンドウのサイズ変更時に呼び出す処理を登録する
00412     glfwSetFramebufferSizeCallback(window, resize);
00413
00414     // 垂直同期タイミングに合わせる
00415     glfwSwapInterval(1);
00416
00417     // 実際のフレームバッファのサイズを取得する
00418     glfwGetFramebufferSize(window, &width, &height);
00419
00420     // ビューポートと投影変換行列を初期化する
00421     resize(window, width, height);
00422
00423 #if defined(IMGUI_VERSION)
00424     // 最初のウィンドウを開いたとき
00425     static bool firstTime{ true };
00426     if (firstTime)
00427     {
00428         // Setup Platform/Renderer bindings
00429         ImGui_ImplGlfw_InitForOpenGL(window, true);
00430         ImGui_ImplOpenGL3_Init(nullptr);
00431
00432         // 実行済みであることを記録する
00433         firstTime = false;
00434     }
00435 #endif
```

```

00436 }
00437
00438 // 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
00439 // イベントを取り出す
00440 {
00441 GgApp::Window::operator bool()
00442 {
00443 // ウィンドウを閉じるべきなら false を返す
00444 if (shouldClose()) return false;
00445
00446 // 対象のユーザインターフェース
00447 auto& current_if{ interfaceData[interfaceNo] };
00448
00449 // ImGui の状態を取り出す
00450 const ImGuiIO& io{ ImGui::GetIO() };
00451
00452 #if defined(IMGUI_VERSION)
00453 // ImGui の新規フレームを作成する
00454 ImGui_ImplOpenGL3_NewFrame();
00455 ImGui_ImplGlfw_NewFrame();
00456
00457 // ImGui の状態を取り出す
00458 // 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
00459 if (io.WantCaptureMouse) return true;
00460
00461 // マウスの位置を更新する
00462 current_if.mouse = std::array<GLfloat, 2>{ io.MousePos.x, io.MousePos.y };
00463
00464 // マウスの現在位置を調べる
00465 double x, y;
00466 glfwGetCursorPos(window, &x, &y);
00467
00468 // マウスの位置を更新する
00469 current_if.mouse = std::array<GLfloat, 2>{ static_cast<GLfloat>(x), static_cast<GLfloat>(y) };
00470
00471 #endif
00472
00473 // マウスドラッグ
00474 for (int button = GLFW_MOUSE_BUTTON_1; button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT; ++button)
00475 {
00476 // マウスポタンを押していたら
00477 if (status[button])
00478 {
00479 // 現在位置と平行移動量を更新する
00480 current_if.calcTranslation(button, velocity);
00481 }
00482 }
00483
00484 return true;
00485 }
00486
00487
00488 // カラーバッファを入れ替える
00489 // カラーバッファを入れ替える
00490 //
00491 void GgApp::Window::swapBuffers() const
00492 {
00493 #if defined(IMGUI_VERSION)
00494 // ImGui の描画データがあればフレームをレンダリングする
00495 ImDrawData* data{ ImGui::GetDrawData() };
00496 if (data) ImGui_ImplOpenGL3_RenderDrawData(data);
00497 #endif
00498
00499 // エラーチェック
00500 ggError();
00501
00502 // カラーバッファを入れ替える
00503 glfwSwapBuffers(window);
00504 }
00505
00506 //
00507 // ビューポートのサイズを更新する
00508 //
00509 void GgApp::Window::updateViewport()
00510 {
00511 // フレームバッファの大きさを求める
00512 glfwGetFramebufferSize(window, &fboSize[0], &fboSize[1]);
00513
00514 #if defined(IMGUI_VERSION)
00515 // 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
00516 fboSize[1] -= menubarHeight;
00517 #endif
00518
00519 // ウィンドウの縦横比を保存する
00520 aspect = static_cast<GLfloat>(fboSize[0]) / static_cast<GLfloat>(fboSize[1]);
00521
00522 // ビューポートを設定する

```

```
00523     restoreViewport();
00524 }
00525
00526 #if defined(GG_USE_OCUlus_RIFT)
00527 # if OVR_PRODUCT_VERSION > 0
00528 //
00529 // グラフィックスカードのデフォルトの LUID を得る
00530 //
00531 ovrGraphicsLuid GgApp::Oculus::GetDefaultAdapterLuid()
00532 {
00533     ovrGraphicsLuid luid = ovrGraphicsLuid();
00534
00535 # if defined(_MSC_VER)
00536     IDXGIFactory* factory{ nullptr };
00537
00538     if (SUCCEEDED(CreateDXGIFactory(IID_PPV_ARGS(&factory))))
00539     {
00540         IDXGIAdapter* adapter{ nullptr };
00541
00542         if (SUCCEEDED(factory->EnumAdapters(0, &adapter)))
00543         {
00544             DXGI_ADAPTER_DESC desc;
00545
00546             adapter->GetDesc(&desc);
00547             memcpy(&luid, &desc.AdapterLuid, sizeof luid);
00548             adapter->Release();
00549         }
00550
00551         factory->Release();
00552     }
00553 # endif
00554
00555     return luid;
00556 }
00557
00558 //
00559 // グラフィックスカードの LUID の比較
00560 //
00561 int GgApp::Oculus::Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs)
00562 {
00563     return memcmp(&lhs, &rhs, sizeof(ovrGraphicsLuid));
00564 }
00565 # endif
00566
00567 //
00568 // コンストラクタ
00569 //
00570 GgApp::Oculus::Oculus() :
00571     session{ nullptr },
00572     oculusFbo{ 0 },
00573     screen{ -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, },
00574     mirrorFbo{ 0 },
00575     window{ nullptr },
00576 # if OVR_PRODUCT_VERSION > 0
00577     frameIndex{ 0 },
00578     oculusDepth{ 0 },
00579     mirrorWidth{ 1280 },
00580     mirrorHeight{ 640 },
00581 # endif
00582     mirrorTexture{ nullptr }
00583 {
00584 }
00585
00586 //
00587 // Oculus Rift のセッションを作成する
00588 //
00589 GgApp::Oculus& GgApp::Oculus::initialize(const Window& window)
00590 {
00591     // Oculus Rift のコンテキスト
00592     static Oculus oculus;
00593
00594     // 既に Oculus Rift のセッションが作成されていたら参照を返す
00595     if (oculus.session) return oculus;
00596
00597     // 最初に呼び出したときだけ実行する
00598     static bool firstTime{ true };
00599     if (firstTime)
00600     {
00601         // Oculus Rift (LibOVR) を初期化する
00602         ovrInitParams initParams{ ovrInit_RequestVersion, OVR_MINOR_VERSION, NULL, 0, 0 };
00603         if (OVR_FAILURE(ovr_Initialize(&initParams)))
00604             throw std::runtime_error("Can't initialize LibOVR");
00605
00606         // アプリケーションの終了時に LibOVR を終了する
00607         atexit(ovr_Shutdown);
00608
00609         // 実行済みであることを記録する
00610     }
```

```

00610     firstTime = false;
00611 }
00612
00613 // Oculus Rift のセッションを作成する
00614 ovrGraphicsLuid luid;
00615 if (OVR_FAILURE(ovr.Create(&oculus.session, &luid)))
00616     throw std::runtime_error("Can't create Oculus Rift session");
00617
00618 # if OVR_PRODUCT_VERSION > 0
00619 // 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
00620 if (Compare(luid, GetDefaultAdapterLuid()))
00621     throw std::runtime_error("Graphics adapter is not default");
00622 # endif
00623
00624 // session が無効ならエラー
00625 if (!oculus.session) std::runtime_error("Unable to use the Oculus Rift.");
00626
00627 // ミラー表示を行うウィンドウを設定する
00628 oculus.window = &window;
00629
00630 // Oculus Rift の情報を取り出す
00631 oculus.hmdDesc = ovr.GetHmdDesc(oculus.session);
00632
00633 # if defined(_DEBUG)
00634 // Oculus Rift の情報を表示する
00635 std::cerr
00636     << "\nProduct name: " << oculus.hmdDesc.ProductName
00637     << "\nResolution: " << oculus.hmdDesc.Resolution.w << " x " << oculus.hmdDesc.Resolution.h
00638     << "\nDefault Fov: (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].LeftTan
00639     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].DownTan
00640     << ")" - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].RightTan
00641     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].UpTan
00642     << ")\\n        (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].LeftTan
00643     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].DownTan
00644     << ")" - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].RightTan
00645     << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].UpTan
00646     << ")\\nMaximum Fov: (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].LeftTan
00647     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].DownTan
00648     << ")" - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].RightTan
00649     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].UpTan
00650     << ")\\n        (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].LeftTan
00651     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].DownTan
00652     << ")" - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].RightTan
00653     << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].UpTan
00654     << ")\\n" << std::endl;
00655 # endif
00656
00657 // Oculus Rift に転送する描画データを作成する
00658 # if OVR_PRODUCT_VERSION > 0
00659 oculus.layerData.Header.Type = ovrLayerType_EyeFov;
00660 # else
00661 oculus.layerData.Header.Type = ovrLayerType_EyeFovDepth;
00662 # endif
00663
00664 // OpenGL なので左下が原点
00665 oculus.layerData.Header.Flags = ovrLayerFlag_TextureOriginAtBottomLeft;
00666
00667 // Oculus Rift のレンダリングに使う FBO を作成する
00668 glGenFramebuffers(ovrEye_Count, oculus.oculusFbo);
00669
00670 // 全ての目について
00671 for (int eye = 0; eye < ovrEye_Count; ++eye)
00672 {
00673     // Oculus Rift の視野を取得する
00674     const auto& fov{ oculus.hmdDesc.DefaultEyeFov[ovrEyeType(eye)] };
00675
00676     // Oculus Rift 用の FBO のサイズを求める
00677     const auto textureSize{ ovr.GetFovTextureSize(oculus.session, ovrEyeType(eye), fov, 1.0f) };
00678
00679     // Oculus Rift のスクリーンのサイズを保存する
00680     oculus.screen[eye][0] = -fov.LeftTan;
00681     oculus.screen[eye][1] = fov.RightTan;
00682     oculus.screen[eye][2] = -fov.DownTan;
00683     oculus.screen[eye][3] = fov.UpTan;
00684
00685 # if OVR_PRODUCT_VERSION > 0
00686
00687     // 描画データに視野を設定する
00688     oculus.layerData.Fov[eye] = fov;
00689
00690     // 描画データにビューポートを設定する
00691     oculus.layerData.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00692     oculus.layerData.Viewport[eye].Size = textureSize;
00693
00694     // 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
00695     const ovrTextureSwapChainDesc colorDesc
00696     {

```

```

00697     ovrTexture_2D,
00698     OVR_FORMAT_R8G8B8A8_UNORM_SRGB,           // Type
00699     1,                                         // Format
00700     textureSize.w,                          // ArraySize
00701     textureSize.h,                          // Width
00702     1,                                         // Height
00703     1,                                         // MipLevels
00704     1,                                         // SampleCount
00705     ovrFalse,                                // StaticImage
00706     0, 0
00707 };
00708
00709 // 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
00710 oculus.layerData.ColorTexture[eye] = nullptr;
00711 if (OVR_SUCCESS(ovr_CreateTextureSwapChainGL(oculus.session, &colorDesc,
00712 &oculus.layerData.ColorTexture[eye])))
00713 {
00714     // 作成したテクスチャチェインの長さを取得する
00715     int length(0);
00716     if (OVR_SUCCESS(ovr_GetTextureSwapChainLength(oculus.session, oculus.layerData.ColorTexture[eye],
00717 &length)))
00718     {
00719         // テクスチャチェインの個々の要素について
00720         for (int i = 0; i < length; ++i)
00721         {
00722             // テクスチャのパラメータを設定する
00723             GLuint texId{ 0 };
00724             ovr_GetTextureSwapChainBufferGL(oculus.session, oculus.layerData.ColorTexture[eye], i,
00725 &texId);
00726             glBindTexture(GL_TEXTURE_2D, texId);
00727             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00728             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00729             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00730             glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00731         }
00732     }
00733
00734 // 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
00735 glGenTextures(1, oculus.oculusDepth + eye);
00736 glBindTexture(GL_TEXTURE_2D, oculus.oculusDepth[eye]);
00737 glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h, 0,
00738 GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
00739 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00740 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00741 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00742 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00743
00744 # else
00745     // 描画データに視野を設定する
00746     oculus.layerData.EyeFov.Fov[eye] = fov;
00747
00748     // 描画データにビューポートを設定する
00749     oculus.layerData.EyeFov.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00750     oculus.layerData.EyeFov.Viewport[eye].Size = textureSize;
00751
00752     // 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
00753     ovrSwapTextureSet* colorTexture{ nullptr };
00754     ovr_CreateSwapTextureSetGL(oculus.session, GL_RGB8_ALPHA8, textureSize.w, textureSize.h,
00755 &colorTexture);
00756     oculus.layerData.EyeFov.ColorTexture[eye] = colorTexture;
00757
00758     // 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
00759     ovrSwapTextureSet* depthTexture{ nullptr };
00760     ovr_CreateSwapTextureSetGL(oculus.session, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h,
00761 &depthTexture);
00762     oculus.layerData.EyeFov.Depth.DepthTexture[eye] = depthTexture;
00763
00764     // Oculus Rift のレンズ補正等の設定値を取得する
00765     oculus.eyeRenderDesc[eye] = ovr_GetRenderDesc(oculus.session, ovrEyeType(e眼), fov);
00766
00767 # endif
00768 }
00769
00770 // ミラー表示用の FBO を作成する
00771 const GLsizei* size{ oculus.window->GetSize() };
00772 const ovrMirrorTextureDesc mirrorDesc
00773 {
00774     OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00775     oculus.mirrorWidth = size[0], // Width
00776     oculus.mirrorHeight = size[1], // Height
00777     0                           // Flags

```

```

00778    };
00779
00780    // 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
00781    if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, &mirrorDesc, &oculus.mirrorTexture)))
00782    {
00783        // 作成したテクスチャのテクスチャ名を得る
00784        GLuint texId{ 0 };
00785        if (OVR_SUCCESS(ovr_GetMirrorTextureBufferGL(oculus.session, oculus.mirrorTexture, &texId)))
00786        {
00787            // 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
00788            glGenFramebuffers(1, &oculus.mirrorFbo);
00789            glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00790            glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texId, 0);
00791            glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
00792            glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00793        }
00794    }
00795
00796 # else
00797
00798 // 作成したテクスチャのテクスチャ名を得る
00799 if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, GL_SRGB8_ALPHA8, width, height,
reinterpret_cast<ovrTexture*>(&mirrorTexture)))
00800 {
00801     // 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
00802     oculus.mirrorFbo = 0;
00803     glGenFramebuffers(1, &oculus.mirrorFbo);
00804     glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00805     glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
mirrorTexture->OGL.TexId, 0);
00806     glFramebufferRenderbuffer(GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
00807     glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00808 }
00809
00810 # endif
00811
00812 // 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
00813 glEnable(GL_FRAMEBUFFER_SRGB);
00814
00815 // フロントバッファに描く
00816 glDrawBuffer(GL_FRONT);
00817
00818 // Oculus Rift への表示では垂直同期タイミングに合わせない
00819 glfwSwapInterval(0);
00820
00821 return oculus;
00822 }
00823
00824 //
00825 // Oculus Rift のセッションを破棄する
00826 //
00827 void GgApp::Oculus::terminate()
00828 {
00829     // session が無効なら何もしない
00830     if (!session) return;
00831
00832     // ミラー表示用の FBO を作っていたら削除する
00833     if (mirrorFbo)
00834     {
00835         glDeleteFramebuffers(1, &mirrorFbo);
00836         mirrorFbo = 0;
00837     }
00838
00839     // 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
00840     if (mirrorTexture)
00841     {
00842 # if OVR_PRODUCT_VERSION > 0
00843         ovr_DestroyMirrorTexture(session, mirrorTexture);
00844 # else
00845         glDeleteTextures(1, &mirrorTexture->OGL.TexId);
00846         ovr_DestroyMirrorTexture(session, reinterpret_cast<ovrTexture*>(mirrorTexture));
00847 # endif
00848         mirrorTexture = nullptr;
00849     }
00850
00851     // 全ての目について
00852     for (int eye = 0; eye < ovrEye_Count; ++eye)
00853     {
00854         // Oculus Rift へのレンダリング用の FBO を削除する
00855         glDeleteFramebuffers(1, oculusFbo + eye);
00856         oculusFbo[eye] = 0;
00857
00858 # if OVR_PRODUCT_VERSION > 0
00859
00860         // 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
00861         if (layerData.ColorTexture[eye])
00862         {

```

```
00863     ovr_DestroyTextureSwapChain(session, layerData.ColorTexture[eye]);
00864     layerData.ColorTexture[eye] = nullptr;
00865 }
00866
00867 // デブスバッファとして使ったテクスチャを削除する
00868 glDeleteTextures(1, oculusDepth + eye);
00869 oculusDepth[eye] = 0;
00870
00871 # else
00872
00873 // 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
00874 auto* const colorTexture(layerData.EyeFov.ColorTexture[eye]);
00875 for (int i = 0; i < colorTexture->TextureCount; ++i)
00876 {
00877     const auto* const ctex(reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[i]));
00878     glDeleteTextures(1, &ctex->OGL.TexId);
00879 }
00880 ovr_DestroySwapTextureSet(session, colorTexture);
00881
00882 // デブスバッファとして使ったテクスチャを削除する
00883 auto* const depthTexture(layerData.EyeFovDepth.DepthTexture[eye]);
00884 for (int i = 0; i < depthTexture->TextureCount; ++i)
00885 {
00886     const auto* const dtex(reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[i]));
00887     glDeleteTextures(1, &dtex->OGL.TexId);
00888 }
00889 ovr_DestroySwapTextureSet(session, depthTexture);
00890
00891 # endif
00892 }
00893
00894 // Oculus Rift のセッションを破棄する
00895 ovrDestroy(session);
00896 session = nullptr;
00897
00898 // カラースペースを元に戻す
00899 glDisable(GL_FRAMEBUFFER_SRGB);
00900
00901 // バックバッファに描く
00902 glDrawBuffer(GL_BACK);
00903
00904 // 垂直同期タイミングに合わせる
00905 glfwSwapInterval(1);
00906 }
00907
00908 //
00909 // Oculus Rift による描画開始
00910 //
00911 bool GgApp::Oculus::begin()
00912 {
00913 # if OVR_PRODUCT_VERSION > 0
00914
00915 // セッションの状態を取得する
00916 ovrSessionStatus sessionStatus;
00917 ovr_GetSessionStatus(session, &sessionStatus);
00918
00919 // 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
00920 if (sessionStatus.ShouldQuit) window->setClose(GLFW_TRUE);
00921
00922 // Oculus Rift に表示されていないときは戻る
00923 if (!sessionStatus.IsVisible) return false;
00924
00925 // 現在の状態をトラッキングの原点にする
00926 if (sessionStatus.ShouldRecenter) ovr_RecenterTrackingOrigin(session);
00927
00928 // 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
00929 const ovrEyeRenderDesc eyeRenderDesc[]
00930 {
00931     ovr_GetRenderDesc(session, ovrEyeType(0), hmdDesc.DefaultEyeFov[0]),
00932     ovr_GetRenderDesc(session, ovrEyeType(1), hmdDesc.DefaultEyeFov[1])
00933 };
00934
00935 // 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
00936 const ovrPosef hmdToEyePose[]
00937 {
00938     eyeRenderDesc[0].HmdToEyePose,
00939     eyeRenderDesc[1].HmdToEyePose
00940 };
00941
00942 // 視点の姿勢情報を取得する
00943 ovr_GetEyePoses(session, frameIndex, ovrTrue, hmdToEyePose, layerData.RenderPose,
&layerData.SensorSampleTime);
00944
00945 # else
00946
00947 // フレームのタイミング計測開始
00948 const auto ftiming(ovr_GetPredictedDisplayTime(session, 0));
```

```

00949
00950 // sensorSampleTime の取得は可能な限り ovr_GetTrackingState() の近くで行う
00951 layerData.EyeFov.SensorSampleTime = ovr_GetTimeInSeconds();
00952
00953 // ヘッドトラッキングの状態を取得する
00954 const auto hmdState(ovr_GetTrackingState(session, ftiming, ovrTrue));
00955
00956 // 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
00957 const ovrVector3f hmdToEyeViewOffset[]
00958 {
00959     eyeRenderDesc[0].HmdToEyeViewOffset,
00960     eyeRenderDesc[1].HmdToEyeViewOffset
00961 };
00962
00963 // 視点の姿勢情報を求める
00964 ovr_CalcEyePoses(hmdState.HeadPose.ThePose, hmdToEyeViewOffset, eyePose);
00965
00966 # endif
00967
00968 return true;
00969 }
00970
00971 //
00972 // Oculus Rift の描画する目の指定
00973 //
00974 void GgApp::Oculus::select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation)
00975 {
00976 # if OVR_PRODUCT_VERSION > 0
00977
00978 // Oculus Rift にレンダリングする FBO に切り替える
00979 if (layerData.ColorTexture[eye])
00980 {
00981     // 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
00982     int curIndex;
00983     ovr_GetTextureSwapChainCurrentIndex(session, layerData.ColorTexture[eye], &curIndex);
00984
00985     // FBO のカラー バッファに使うテクスチャを取得する
00986     GLuint curTexId;
00987     ovr_GetTextureSwapChainBufferGL(session, layerData.ColorTexture[eye], curIndex, &curTexId);
00988
00989     // FBO を設定する
00990     glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
00991     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, curTexId, 0);
00992     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, oculusDepth[eye], 0);
00993
00994     // ビューポートを設定する
00995     const auto& vp{ layerData.Viewport[eye] };
00996     glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
00997 }
00998
00999 // Oculus Rift の片目の位置と回転を取得する
01000 const auto& p{ layerData.RenderPose[eye].Position };
01001 const auto& o{ layerData.RenderPose[eye].Orientation };
01002
01003 # else
01004
01005 // 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
01006 auto* const colorTexture{ layerData.EyeFov.ColorTexture[eye] };
01007 colorTexture->CurrentIndex = (colorTexture->CurrentIndex + 1) % colorTexture->TextureCount;
01008 auto* const depthTexture{ layerData.EyeFovDepth.DepthTexture[eye] };
01009 depthTexture->CurrentIndex = (depthTexture->CurrentIndex + 1) % depthTexture->TextureCount;
01010
01011 // レンダーターゲットを切り替える
01012 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
01013 const auto& ctex{
01014     reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[colorTexture->CurrentIndex]);
01015     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ctex->OGL.TexId, 0);
01016     const auto& dtex{
01017         reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[depthTexture->CurrentIndex]);
01018         glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, dtex->OGL.TexId, 0);
01019
01020     // ビューポートを設定する
01021     const auto& vp{ layerData.EyeFov.Viewport[eye] };
01022     glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
01023
01024     // Oculus Rift の片目の位置と回転を取得する
01025     const auto& p{ eyePose[eye].Position };
01026     const auto& o{ eyePose[eye].Orientation };
01027
01028 # endif
01029 // Oculus Rift のスクリーンの大きさを返す
01030 screen[0] = this->screen[eye][0];
01031 screen[1] = this->screen[eye][1];
01032 screen[2] = this->screen[eye][2];
01033 screen[3] = this->screen[eye][3];
01034

```

```

01034 // Oculus Rift の位置を返す
01035 position[0] = p.x;
01036 position[1] = p.y;
01037 position[2] = p.z;
01038
01039 // Oculus Rift の方向を返す
01040 orientation[0] = o.x;
01041 orientation[1] = o.y;
01042 orientation[2] = o.z;
01043 orientation[3] = o.w;
01044 }
01045
01046 //
01047 // Time Warp 处理に使う投影変換行列の成分の設定 (DK1, DK2)
01048 //
01049 void GgApp::Oculus::timewarp(const GgMatrix& projection)
01050 {
01051 # if OVR_PRODUCT_VERSION < 1
01052 // TimeWarp に使う変換行列の成分を設定する
01053 auto& posTimewarpProjectionDesc{ layerData.EyeFovDepth.ProjectionDesc };
01054 posTimewarpProjectionDesc.Projection22 = (projection.get()[4 * 2 + 2] + projection.get()[4 * 3 + 2])
* 0.5f;
01055 posTimewarpProjectionDesc.Projection23 = projection.get()[4 * 2 + 3] * 0.5f;
01056 posTimewarpProjectionDesc.Projection32 = projection.get()[4 * 3 + 2];
01057 # endif
01058 }
01059
01060 //
01061 // 図形の描画を完了する (CV1 以降)
01062 //
01063 void GgApp::Oculus::commit(int eye)
01064 {
01065 # if OVR_PRODUCT_VERSION > 0
01066 // GL_COLOR_ATTACHMENT0 に割り当てられたテクスチャが wglDXUnlockObjectsNV() によって
01067 // 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
01068 // FBO に結合されるのを避ける
01069 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
01070 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
01071 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, 0, 0);
01072
01073 // 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
01074 ovrCommitTextureSwapChain(session, layerData.ColorTexture[eye]);
01075 # endif
01076 }
01077
01078 //
01079 // フレームを転送する
01080 //
01081 bool GgApp::Oculus::submit(bool mirror)
01082 {
01083 // エラーチェック
01084 ggError();
01085
01086 # if OVR_PRODUCT_VERSION > 0
01087 // 描画データを Oculus Rift に転送する
01088 const auto* const layers{ &layerData.Header };
01089 if (OVR_FAILURE(ovrSubmitFrame(session, frameIndex++, nullptr, &layers, 1))) return false;
01090 # else
01091 // Oculus Rift 上の描画位置と拡大率を求める
01092 ovrViewScaleDesc viewScaleDesc;
01093 viewScaleDesc.HmdSpaceToWorldScaleInMeters = 1.0f;
01094 viewScaleDesc.HmdToEyeViewOffset[0] = eyeRenderDesc[0].HmdToEyeViewOffset;
01095 viewScaleDesc.HmdToEyeViewOffset[1] = eyeRenderDesc[1].HmdToEyeViewOffset;
01096
01097 // 描画データを更新する
01098 layerData.EyeFov.RenderPose[0] = eyePose[0];
01099 layerData.EyeFov.RenderPose[1] = eyePose[1];
01100
01101 // 描画データを Oculus Rift に転送する
01102 const auto* const layers{ &layerData.Header };
01103 if (OVR_FAILURE(ovrSubmitFrame(session, 0, &viewScaleDesc, &layers, 1))) return false;
01104 # endif
01105
01106 // ミラー表示
01107 if (mirror)
01108 {
01109 # if OVR_PRODUCT_VERSION > 0
01110 const auto& sxl{ mirrorWidth };
01111 const auto& syl{ mirrorHeight };
01112 # else
01113 const auto& sxl{ mirrorTexture->OGL.Header.TextureSize.w };
01114 const auto& syl{ mirrorTexture->OGL.Header.TextureSize.h };
01115 # endif
01116
01117 // ミラー表示のウィンドウのサイズ
01118 GLsizei size[2];
01119 window->getSize(size);

```

```

01120 // ミラー表示の表示領域
01121 GLint dx0{ 0 }, dx1{ size[0] }, dy0{ 0 }, dy1{ size[1] };
01123
01124 // ミラー表示がウィンドウからはみ出ないようにする
01125 if ((size[0] *= sy1) < (size[1] *= sx1))
01126 {
01127     const GLint tyl{ size[0] / sx1 };
01128     dy0 = (dy1 - tyl) / 2;
01129     dy1 = dy0 + tyl;
01130 }
01131 else
01132 {
01133     const GLint tx1{ size[1] / sy1 };
01134     dx0 = (dx1 - tx1) / 2;
01135     dx1 = dx0 + tx1;
01136 }
01137
01138 // 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
01139 glBindFramebuffer(GL_READ_FRAMEBUFFER, mirrorFbo);
01140 glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
01141 glBlitFramebuffer(0, sy1, sx1, 0, dx0, dy0, dx1, dy1, GL_COLOR_BUFFER_BIT, GL_NEAREST);
01142 glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
01143
01144 // 残っている OpenGL コマンドを実行する
01145 glFlush();
01146 }
01147
01148 return true;
01149 }
01150 #endif

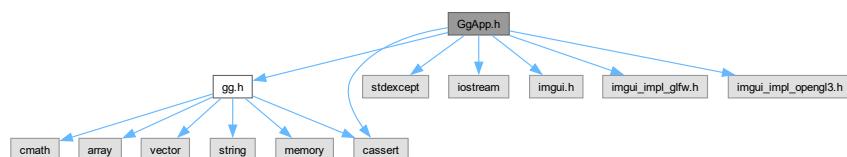
```

9.15 GgApp.h ファイル

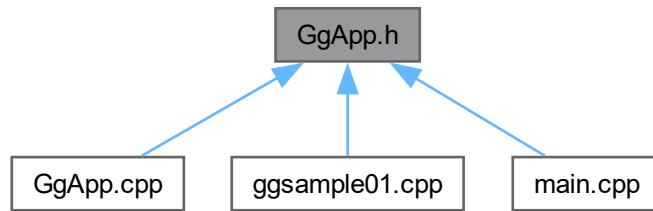
```

#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`

9.15.1 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

July 17, 2025

[GgApp.h](#) に定義があります。

9.15.2 マクロ定義詳解

9.15.2.1 GG_BUTTON_COUNT

```
#define GG_BUTTON_COUNT 3
```

[GgApp.h](#) の 43 行目に定義があります。

9.15.2.2 GG_INTERFACE_COUNT

```
#define GG_INTERFACE_COUNT 5
```

GgApp.h の 48 行目に定義がります。

9.15.2.3 GG_USE_IMGUI

```
#define GG_USE_IMGUI
```

GgApp.h の 36 行目に定義がります。

9.16 GgApp.h

[詳解]

```
00001 #pragma once
00002 /*
00004 ゲームグラフィックス特論用補助プログラム GLFW3 版
00006
00007 Copyright (c) 2011-2025 Kohe Tokoi. All Rights Reserved.
00008
00009 Permission is hereby granted, free of charge, to any person obtaining a copy
00010 of this software and associated documentation files (the "Software"), to deal
00011 in the Software without restriction, including without limitation the rights
00012 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00013 copies or substantial portions of the Software.
00014
00015 The above copyright notice and this permission notice shall be included in
00016 all copies or substantial portions of the Software.
00017
00018 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00019 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00020 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
00021 KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
00022 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
00023 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00024
00025 */
00026
00034
00035 // Dear ImGui を使うなら
00036 #define GG_USE_IMGUI
00037
00038 // Oculus Rift を使うなら
00039 // #define GG_USE_OOCULUS_RIFT
00040
00041 // 使用するマウスのボタン数
00042 #if !defined(GG_BUTTON_COUNT)
00043 # define GG_BUTTON_COUNT 3
00044 #endif
00045
00046 // 使用するユーザインターフェースの数
00047 #if !defined(GG_INTERFACE_COUNT)
00048 # define GG_INTERFACE_COUNT 5
00049 #endif
00050
00051 // 補助プログラム
00052 #include "gg.h"
00053 using namespace gg;
00054
00055 // 標準ライブラリ
00056 #include <cassert>
00057 #include <stdexcept>
00058 #include <iostream>
00059
00060 // ImGui の組み込み
00061 #if defined(GG_USE_IMGUI)
00062 # include "imgui.h"
00063 # include "imgui_impl_glfw.h"
00064 # include "imgui_impl_opengl3.h"
00065 #endif
```

```
00066 // Oculus Rift SDK ライブリ (LibOVR) の組み込み
00067 #if defined(GG_USE_OOCULUS_RIFT)
00068 # if defined(_MSC_VER)
00069 #   define GLFW_EXPOSE_NATIVE_WIN32
00070 #   define GLFW_EXPOSE_NATIVE_WGL
00071 #   include <GLFW/glfw3native.h>
00072 #   define OVR_OS_WIN32
00073 #   undef APIENTRY
00074 #   pragma comment(lib, "LibOVR.lib")
00075 # endif
00076 #endif
00077 # include <OVR_CAPI_GL.h>
00078 # include <Extras/OVR_Math.h>
00079 # if OVR_PRODUCT_VERSION > 0
00080 #   include <dxgi.h> // GetDefaultAdapterLuid のため
00081 #   pragma comment(lib, "dxgi.lib")
00082 # endif
00083 #endif
00084
00085 class GgApp
00086 {
00087 public:
00088     GgApp(int major = 0, int minor = 1);
00089
00090     GgApp(const GgApp& w) = delete;
00091
00092     GgApp(GgApp&& w) = default;
00093
00094     virtual ~GgApp();
00095
00096     GgApp& operator=(const GgApp& w) = delete;
00097
00098     GgApp& operator=(GgApp&& w) = default;
00099
00100     int main(int argc, const char* const* argv);
00101
00102     class Window
00103     {
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             // トラックボール
00146             std::array<GgTrackball, GG_BUTTON_COUNT> rotation;
00147
00148             // コンストラクタ
00149             HumanInterface() :
00150                 lastKey{ 0 },
00151                 arrow{},
```

```

00186     mouse{},
00187     wheel{},
00188     translation{}
00189 {
00190     resetTranslation();
00191 }
00192
00193 /**
00194 // マウスや矢印キーによる平行移動量を初期化する
00195 /**
00196 void resetTranslation();
00197
00198 /**
00199 // 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)
00200 /**
00201 void calcTranslation(int button, const std::array<GLfloat, 3>& velocity);
00202 };
00203
00204 // ヒューマンインターフェースデバイスのデータ
00205 std::array<HumanInterface, GG_INTERFACE_COUNT> interfaceData;
00206
00207 // ヒューマンインターフェースデバイスの番号
00208 int interfaceNo;
00209
00210 /**
00211 // ユーザー定義のコールバック関数へのポインタ
00212 /**
00213 void* userPointer;
00214 void (*resizeFunc)(const Window* window, int width, int height);
00215 void (*keyboardFunc)(const Window* window, int key, int scanCode, int action, int mods);
00216 void (*mouseFunc)(const Window* window, int button, int action, int mods);
00217 void (*wheelFunc)(const Window* window, double x, double y);
00218
00219 /**
00220 // ウィンドウのサイズ変更時の処理
00221 /**
00222 static void resize(GLFWwindow* window, int width, int height);
00223
00224 /**
00225 // キーボードをタイプした時の処理
00226 /**
00227 static void keyboard(GLFWwindow* window, int key, int scanCode, int action, int mods);
00228
00229 /**
00230 // マウスボタンを操作したときの処理
00231 /**
00232 static void mouse(GLFWwindow* window, int button, int action, int mods);
00233
00234 /**
00235 // マウスホイールを操作した時の処理
00236 /**
00237 static void wheel(GLFWwindow* window, double x, double y);
00238
00239 public:
00240
00250 Window(const std::string& title = "GLFW Window", int width = 640, int height = 480,
00251         int fullscreen = 0, GLFWwindow* share = nullptr);
00252
00256 Window(const Window& w) = delete;
00257
00261 Window(Window&& w) = default;
00262
00266 virtual ~Window()
00267 {
00268     // ウィンドウが作成されていなければ戻る
00269     if (!window) return;
00270
00271     // ウィンドウを破棄する
00272     glfwDestroyWindow(window);
00273 }
00274
00278 Window& operator=(const Window& w) = delete;
00279
00283 Window& operator=(Window&& w) = default;
00284
00290 auto* get() const
00291 {
00292     return window;
00293 }
00294
00300 void setClose(int flag = GLFW_TRUE) const
00301 {
00302     glfwSetWindowShouldClose(window, flag);
00303 }
00304
00310 bool shouldClose() const
00311 {

```

```
00312     // ウィンドウを閉じるべきなら true を返す
00313     return glfwWindowShouldClose(window) != GLFW_FALSE;
00314 }
00315
00321 explicit operator bool();
00322
00326 void swapBuffers() const;
00327
00331 void restoreViewport() const
00332 {
00333     if (!glfwGetWindowAttrib(window, GLFW_ICONIFIED)) glViewport(0, 0, fboSize[0], fboSize[1]);
00334 }
00335
00339 void updateViewport();
00340
00341 #if defined(IMGUI_VERSION)
00347 void setMenubarHeight(GLsizei height)
00348 {
00349     // メニューバーの高さを保存する
00350     menubarHeight = height;
00351
00352     // ビューポートを復帰する
00353     updateViewport();
00354 }
00355#endif
00356
00362 auto getWidth() const
00363 {
00364     return size[0];
00365 }
00366
00372 auto getHeight() const
00373 {
00374     return size[1];
00375 }
00376
00382 auto getFboWidth() const
00383 {
00384     return fboSize[0];
00385 }
00386
00392 auto getFboHeight() const
00393 {
00394     return fboSize[1];
00395 }
00396
00402 const auto& getSize() const
00403 {
00404     return size;
00405 }
00406
00412 void getSize(GLsizei* size) const
00413 {
00414     size[0] = getWidth();
00415     size[1] = getHeight();
00416 }
00417
00423 const auto& getFboSize() const
00424 {
00425     return fboSize;
00426 }
00427
00433 void getFboSize(GLsizei* fboSize) const
00434 {
00435     fboSize[0] = getFboWidth();
00436     fboSize[1] = getFboHeight();
00437 }
00438
00444 auto getAspect() const
00445 {
00446     return aspect;
00447 }
00448
00454 bool getKey(int key) const
00455 {
00456 #if defined(IMGUI_VERSION)
00457     // 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
00458     if (ImGui::GetIO().WantCaptureKeyboard) return false;
00459#endif
00460
00461     return glfwGetKey(window, key) != GLFW_RELEASE;
00462 }
00463
00469 void selectInterface(int no)
00470 {
00471     assert(static_cast<size_t>(no) < interfaceData.size());
00472     interfaceNo = no;
```

```
00473     }
00474
00475     void setVelocity(GLfloat vx, GLfloat vy, GLfloat vz = 0.1f)
00476     {
00477         velocity = std::array<GLfloat, 3>{ vx, vy, vz };
00478     }
00479
00480     int getLastKey()
00481     {
00482         auto& current_if{ interfaceData[interfaceNo] };
00483         const int key{ current_if.lastKey };
00484         current_if.lastKey = 0;
00485         return key;
00486     }
00487
00488     auto getArrow(int direction = 0, int mods = 0) const
00489     {
00490         const auto& current_if{ interfaceData[interfaceNo] };
00491         return static_cast<GLfloat>(current_if.arrow[mods & 3][direction & 1]);
00492     }
00493
00494     auto getArrowX(int mods = 0) const
00495     {
00496         return getArrow(0, mods);
00497     }
00498
00499     auto getArrowY(int mods = 0) const
00500     {
00501         return getArrow(1, mods);
00502     }
00503
00504     void getArrow(GLfloat* arrow, int mods = 0) const
00505     {
00506         arrow[0] = getArrowX(mods);
00507         arrow[1] = getArrowY(mods);
00508     }
00509
00510     auto getShiftArrowX() const
00511     {
00512         return getArrow(0, 1);
00513     }
00514
00515     auto getShiftArrowY() const
00516     {
00517         return getArrow(1, 1);
00518     }
00519
00520     void getShiftArrow(GLfloat* shift_arrow) const
00521     {
00522         shift_arrow[0] = getShiftArrowX();
00523         shift_arrow[1] = getShiftArrowY();
00524     }
00525
00526     auto getControlArrowX() const
00527     {
00528         return getArrow(0, 2);
00529     }
00530
00531     auto getControlArrowY() const
00532     {
00533         return getArrow(1, 2);
00534     }
00535
00536     void getControlArrow(GLfloat* control_arrow) const
00537     {
00538         control_arrow[0] = getControlArrowX();
00539         control_arrow[1] = getControlArrowY();
00540     }
00541
00542     auto getAltArrowX() const
00543     {
00544         return getArrow(0, 3);
00545     }
00546
00547     auto getAltArrowY() const
00548     {
00549         return getArrow(1, 3);
00550     }
00551
00552     void getAltArrow(GLfloat* alt_arrow) const
00553     {
00554         alt_arrow[0] = getAltArrowX();
00555         alt_arrow[1] = getAltArrowY();
00556     }
00557
00558     const auto* getMouse() const
00559     {
```

```
00647     const auto& current_if{ interfaceData[interfaceNo] };
00648     return current_if.mouse.data();
00649 }
00650
00651 void getMouse(GLfloat* position) const
00652 {
00653     const auto& current_if{ interfaceData[interfaceNo] };
00654     position[0] = current_if.mouse[0];
00655     position[1] = current_if.mouse[1];
00656 }
00657
00658 auto getMouse(int direction) const
00659 {
00660     const auto& current_if{ interfaceData[interfaceNo] };
00661     return current_if.mouse[direction & 1];
00662 }
00663
00664 auto getMouseX() const
00665 {
00666     const auto& current_if{ interfaceData[interfaceNo] };
00667     return current_if.mouse[0];
00668 }
00669
00670 auto getMouseY() const
00671 {
00672     const auto& current_if{ interfaceData[interfaceNo] };
00673     return current_if.mouse[1];
00674 }
00675
00676 const auto* getWheel() const
00677 {
00678     const auto& current_if{ interfaceData[interfaceNo] };
00679     return current_if.wheel.data();
00680 }
00681
00682 void getWheel(GLfloat* rotation) const
00683 {
00684     const auto& current_if{ interfaceData[interfaceNo] };
00685     rotation[0] = current_if.wheel[0];
00686     rotation[1] = current_if.wheel[1];
00687 }
00688
00689 auto getWheel(int direction) const
00690 {
00691     const auto& current_if{ interfaceData[interfaceNo] };
00692     return current_if.wheel[direction & 1];
00693 }
00694
00695 auto getWheelX() const
00696 {
00697     const auto& current_if{ interfaceData[interfaceNo] };
00698     return current_if.wheel[0];
00699 }
00700
00701 auto getWheelY() const
00702 {
00703     const auto& current_if{ interfaceData[interfaceNo] };
00704     return current_if.wheel[1];
00705 }
00706
00707 const auto& getTranslation(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     return current_if.translation[button][1];
00712 }
00713
00714 auto getTranslationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00715 {
00716     const auto& current_if{ interfaceData[interfaceNo] };
00717     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00718     const auto& t{ current_if.translation[button][1] };
00719
00720     GgMatrix m
00721     {
00722         1.0f, 0.0f, 0.0f, 0.0f,
00723         0.0f, 1.0f, 0.0f, 0.0f,
00724         0.0f, 0.0f, 1.0f, 0.0f,
00725         t[0], t[1], t[2], 1.0f
00726     };
00727
00728     return m;
00729 }
00730
00731 auto getScrollMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00732 {
00733     const auto& current_if{ interfaceData[interfaceNo] };
```

```

00799 assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00800 const auto& t{ current_if.translation[button][1] };
00801
00802 GgMatrix m
00803 {
00804     t[2] + 1.0f, 0.0f, 0.0f, 0.0f,
00805     0.0f, t[2] + 1.0f, 0.0f, 0.0f,
00806     0.0f, 0.0f, 1.0f, 0.0f,
00807     t[0], t[1], 0.0f, 1.0f
00808 };
00809
00810     return m;
00811 }
00812
00813 auto getRotation(int button = GLFW_MOUSE_BUTTON_1) const
00814 {
00815     const auto& current_if{ interfaceData[interfaceNo] };
00816     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00817     return current_if.rotation[button].getQuaternion();
00818 }
00819
00820 auto getRotationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00821 {
00822     const auto& current_if{ interfaceData[interfaceNo] };
00823     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00824     return current_if.rotation[button].getMatrix();
00825 }
00826
00827 void resetRotation()
00828 {
00829     // トラックボールを初期化する
00830     for (auto& tb : interfaceData[interfaceNo].rotation) tb.reset();
00831 }
00832
00833 void resetTranslation()
00834 {
00835     // 現在のインターフェースの平行移動量を初期化する
00836     interfaceData[interfaceNo].resetTranslation();
00837 }
00838
00839 void reset()
00840 {
00841     // トラックボール処理を初期化する
00842     resetRotation();
00843
00844     // 平行移動量を初期化する
00845     resetTranslation();
00846 }
00847
00848 void* getUserPointer() const
00849 {
00850     return userPointer;
00851 }
00852
00853 void setUserPointer(void* pointer)
00854 {
00855     userPointer = pointer;
00856 }
00857
00858 void setResizeFunc(void (*func)(const Window* window, int width, int height))
00859 {
00860     resizeFunc = func;
00861 }
00862
00863 void setKeyboardFunc(void (*func)(const Window* window, int key, int scancode, int action, int
00864 mods))
00865 {
00866     keyboardFunc = func;
00867 }
00868
00869 void setMouseFunc(void (*func)(const Window* window, int button, int action, int mods))
00870 {
00871     mouseFunc = func;
00872 }
00873
00874 void setWheelFunc(void (*func)(const Window* window, double x, double y))
00875 {
00876     wheelFunc = func;
00877 }
00878 };
00879
00880 #if defined(GG_USE_OCU盧S_RIFT)
00881 class Oculus
00882 {
00883     // Oculus Rift のセッション
00884     ovrSession session;
00885
00886     void setupOculus();
00887
00888     void updateOculus();
00889
00890     void handleOculusInput();
00891
00892     void handleOculusEvents();
00893
00894     void handleOculusSession();
00895
00896     void handleOculusTracking();
00897
00898     void handleOculusInput();
00899
00900     void handleOculusEvents();
00901
00902     void handleOculusSession();
00903
00904     void handleOculusTracking();
00905
00906     void handleOculusInput();
00907
00908     void handleOculusEvents();
00909
00910     void handleOculusSession();
00911
00912     void handleOculusTracking();
00913
00914     void handleOculusInput();
00915
00916     void handleOculusEvents();
00917
00918     void handleOculusSession();
00919
00920     void handleOculusTracking();
00921
00922     void handleOculusInput();
00923
00924     void handleOculusEvents();
00925
00926     void handleOculusSession();
00927
00928     void handleOculusTracking();
00929
00930 #endif

```

```
00942 // Oculus Rift の状態
00943 ovrHmdDesc hmdDesc;
00944
00945 // Oculus Rift へのレンダリングに使う FBO
00946 GLuint oculusFbo[ovrEye_Count];
00947
00948 // Oculus Rift のスクリーンのサイズ
00949 GLfloat screen[ovrEye_Count][4];
00950
00951 // ミラー表示用の FBO
00952 GLuint mirrorFbo;
00953
00954 // Oculus Rift のミラー表示を行うウィンドウ
00955 const Window* window;
00956
00957 # if OVR_PRODUCT_VERSION > 0
00958
00959 // Oculus Rift に送る描画データ
00960 ovrLayerEyeFov layerData;
00961
00962 // Oculus Rift にレンダリングするフレームの番号
00963 long long frameIndex;
00964
00965 // Oculus Rift へのレンダリングに使う FBO のデブステクスチャ
00966 GLuint oculusDepth[ovrEye_Count];
00967
00968 // ミラー表示用の FBO のサイズ
00969 int mirrorWidth, mirrorHeight;
00970
00971 // ミラー表示用の FBO のカラーteksture
00972 ovrMirrorTexture mirrorTexture;
00973
00974 //
00975 // グラフィックスカードのデフォルトの LUID を得る
00976 //
00977 static ovrGraphicsLuid GetDefaultAdapterLuid();
00978
00979 //
00980 // グラフィックスカードの LUID の比較
00981 //
00982 static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);
00983
00984 # else
00985
00986 // Oculus Rift に送る描画データ
00987 ovrLayerUnion layerData;
00988
00989 // Oculus Rift のレンダリング情報
00990 ovrEyeRenderDesc eyeRenderDesc[ovrEye_Count];
00991
00992 // Oculus Rift の視点情報
00993 ovrPosef eyePose[ovrEye_Count];
00994
00995 // ミラー表示用の FBO のカラーteksture
00996 ovrGLTexture* mirrorTexture;
00997
00998 # endif
00999
01000 //
01001 // コンストラクタ
01002 //
01003 Oculus();
01004
01005 //
01006 // デストラクタ
01007 //
01008 virtual ~Oculus() = default;
01009
01010 public:
01011
01012 // シングルトンなのでコピー・ムーブ禁止
01013 Oculus(const Oculus&) = delete;
01014 Oculus& operator=(const Oculus&) = delete;
01015 Oculus(Oculus&&) = delete;
01016 Oculus& operator=(Oculus&&) = delete;
01017
01023 static Oculus& initialize(const Window& window);
01024
01028 void terminate();
01029
01035 bool begin();
01036
01045 void select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation);
01046
01052 void timewarp(const GgMatrix& projection);
01053
01059 void commit(int eye);
```

```

01060
01061     bool submit(bool mirror = true);
01062 }
01063 #endif
01070 };

```

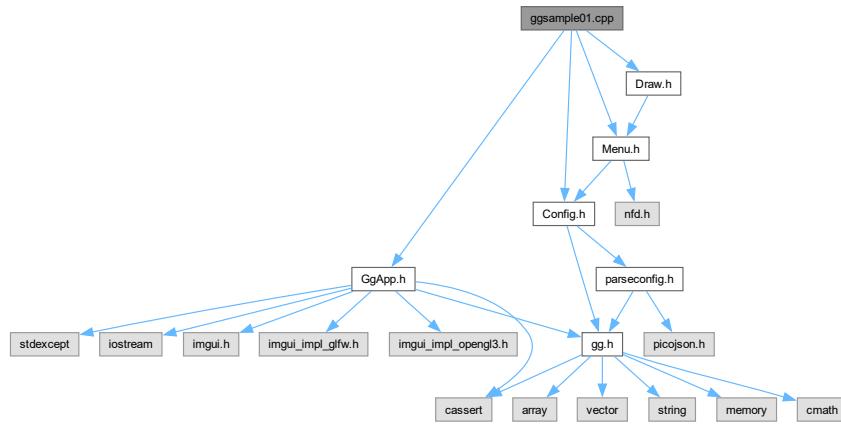
9.17 ggsample01.cpp ファイル

```

#include "GgApp.h"
#include "Config.h"
#include "Menu.h"
#include "Draw.h"

```

ggsample01.cpp の依存先関係図:



マクロ定義

- `#define PROJECT_NAME "ggsample01"`
- `#define CONFIG_FILE PROJECT_NAME "_config.json"`

9.17.1 マクロ定義詳解

9.17.1.1 CONFIG_FILE

```
#define CONFIG_FILE PROJECT_NAME "_config.json"
```

ggsample01.cpp の 13 行目に定義があります。

9.17.1.2 PROJECT_NAME

```
#define PROJECT_NAME "ggsample01"
```

ggsample01.cpp の 8 行目に定義があります。

9.18 ggsample01.cpp

[詳解]

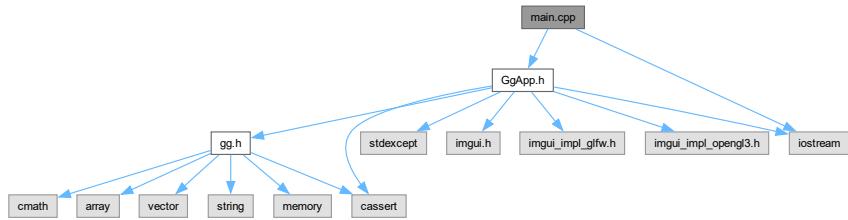
```

00001 //
00002 // ゲームグラフィックス特論宿題アプリケーション
00003 //
00004 #include "GgApp.h"
00005
00006 // プロジェクト名
00007 #if !defined(PROJECT_NAME)
00008 # define PROJECT_NAME "ggsample01"
00009 #endif
00010
00011 // 構成ファイル名
00012 #if !defined(CONFIG_FILE)
00013 # define CONFIG_FILE PROJECT_NAME ".config.json"
00014 #endif
00015
00016 // 構成データ
00017 #include "Config.h"
00018
00019 // メニューの描画
00020 #include "Menu.h"
00021
00022 // 図形の描画
00023 #include "Draw.h"
00024
00025 //
00026 // アプリケーション本体
00027 //
00028 int GgApp::main(int argc, const char* const* argv)
00029 {
00030     // 設定を読み込む
00031     const Config config{ CONFIG_FILE };
00032
00033     // ウィンドウを作成する
00034     Window window{ argc > 1 ? argv[1] : PROJECT_NAME, config.getWidth(), config.getHeight() };
00035
00036     // メニューを初期化する
00037     Menu menu{ config };
00038
00039     // 図形の描画の設定を行う
00040     Draw draw{ menu };
00041
00042     // ピュー変換行列を設定する
00043     const GgMatrix mv{ ggLookat(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f) };
00044
00045     // ウィンドウが開いている間繰り返す
00046     while (window)
00047     {
00048         // ウィンドウを消去する
00049         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00050
00051         // メニューを表示する
00052         menu.draw();
00053
00054         // オブジェクトの回転の変換行列を設定する
00055         const auto& mr{ window.getRotationMatrix(0) };
00056
00057         // 視点の平行移動の変換行列を設定する
00058         const auto& mt{ window.getTranslationMatrix(1) };
00059
00060         // 投影変換行列を設定する
00061         const GgMatrix&& mp{ ggPerspective(0.5f, window.getAspect(), 1.0f, 15.0f) };
00062
00063         // 図形を描画する
00064         draw.draw(mp, mt * mv * mr);
00065
00066         // カラーバッファを入れ替えてイベントを取り出す
00067         window.swapBuffers();
00068     }
00069
00070     return 0;
00071 }
```

9.19 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)`

9.19.1 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。

9.19.2 マクロ定義詳解

9.19.2.1 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。

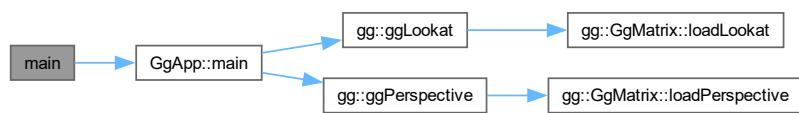
9.19.3 関数詳解

9.19.3.1 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:



9.20 main.cpp

[詳解]

```
[日付]
00001
00002 #include "GgApp.h"
00003
00004 // MessageBox の準備
00005 #if defined(_MSC_VER)
00006 # include <atlstr.h>
00007 #elif defined(__APPLE__)
00008 # include <CoreFoundation/CoreFoundation.h>
00009 #else
00010 # include <iostream>
00011 #endif
00012 #define HEADER_STR "ゲームグラフィックス特論"
00013
00014 //
00015 // メインプログラム
00016 //
00017
00018 int main(int argc, const char* const* argv) try
00019 {
00020     // アプリケーションのオブジェクトを生成する
00021 #if defined(GL_GLES_PROTOTYPES)
00022     GgApp app(3, 1);
00023 #else
00024     GgApp app(4, 1);
00025 #endif
00026
00027     // アプリケーションを実行する
00028     return app.main(argc, argv);
00029 }
00030
00031 catch (const std::runtime_error &e)
00032 {
00033     // エラーメッセージを表示する
00034 #if defined(_MSC_VER)
00035     MessageBox(NULL, CString(e.what()), TEXT(HEADER_STR), MB_ICONERROR);
00036 #elif defined(__APPLE__)
00037     // the following code is copied from
00038     // http://blog.jorgearimany.com/2010/05/messagebox-from-windows-to-mac.html
00039     // convert the strings from char* to CFStringRef
00040     CFStringRef msg_ref = CFStringCreateWithCString(NULL, e.what(), kCFStringEncodingUTF8);
00041
00042     // result code from the message box
00043     CFOptionFlags result;
00044
00045     // launch the message box
00046     CFUserNotificationDisplayAlert(
00047         0, // no timeout
```

```

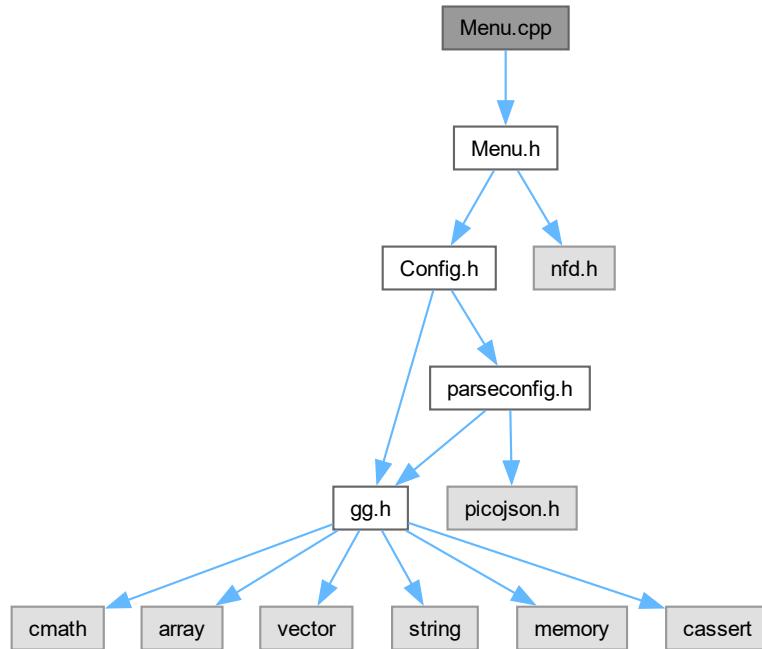
00051     kCFUserNotificationNoteAlertLevel, // change it depending message.type flags ( MB_ICONASTERISK.....
00052     etc.)
00053     NULL,
00054     message_type flags
00055     NULL,
00056     CFSTR(HEADER_STR),
00057     msg.ref,
00058     NULL,
00059     NULL,
00060     &result
00061 );
00062
00063 // 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 }

```

9.21 Menu.cpp ファイル

#include "Menu.h"

Menu.cpp の依存先関係図:



9.21.1 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。

9.22 Menu.cpp

[詳解]

```
00001
00008 #include "Menu.h"
00009
00010 // 
00011 // コンストラクタ
00012 //
00013 Menu::Menu(const Config& config) :
00014     defaults{ config },
00015     settings{ config },
00016     model{ std::make_unique<GgSimpleObj>(config.model, true) },
00017     light{ std::make_unique<GgSimpleShader::LightBuffer>(config.light) },
00018     shader{ std::make_unique<GgSimpleShader>(config.shader) }
00019 {
00020 #if defined(IMGUI_VERSION)
00021     //
00022     // ImGui の初期設定
00023     //
00024
00025     // ファイルダイアログ (Native File Dialog Extended) を初期化する
00026     NFD_Init();
00027
00028     // Dear ImGui の入力デバイス
00029     //io.ConfigFlags |= ImGuiConfigFlags.NavEnableKeyboard;           // キーボードコントロールを使う
00030     //io.ConfigFlags |= ImGuiConfigFlags.NavEnableGamepad;          // ゲームパッドを使う
00031
00032     // Dear ImGui のスタイル
00033     //ImGui::StyleColorsDark();                                     // 暗めのスタイル
00034     //ImGui::StyleColorsClassic();                                  // 以前のスタイル
00035
00036     // 日本語を表示できるメニュー フォントを読み込む
00037     if (!ImGui::GetIO().Fonts->AddFontFromTTF(config.menuFont.c_str(), config.menuFontSize, nullptr,
00038         ImGui::GetIO().Fonts->GetGlyphRangesJapanese()))
00039     {
00040         // メニューフォントが読み込めなかったらエラーにする
00041         throw std::runtime_error("Cannot find any menu fonts.");
00042     }
00043 #endif
00044 }
00045
00046 //
00047 // デストラクタ
00048 //
00049 Menu::~Menu()
00050 {
00051 }
00052
00053 //
00054 // ファイルパスを取得する
00055 //
00056 bool Menu::getFilePath(std::string& path, const nfdfilteritem_t* filter)
00057 {
00058     // ファイルダイアログから得るパス
00059     nfdchar_t* filepath{ nullptr };
00060
00061     // ファイルダイアログを開く
00062     if (NFD_OpenDialog(&filepath, filter, 1, nullptr) == NFD_OKAY)
00063     {
00064         path = TCHARToUtf8(filepath);
00065         return true;
00066     }
00067
00068     return false;
```

```

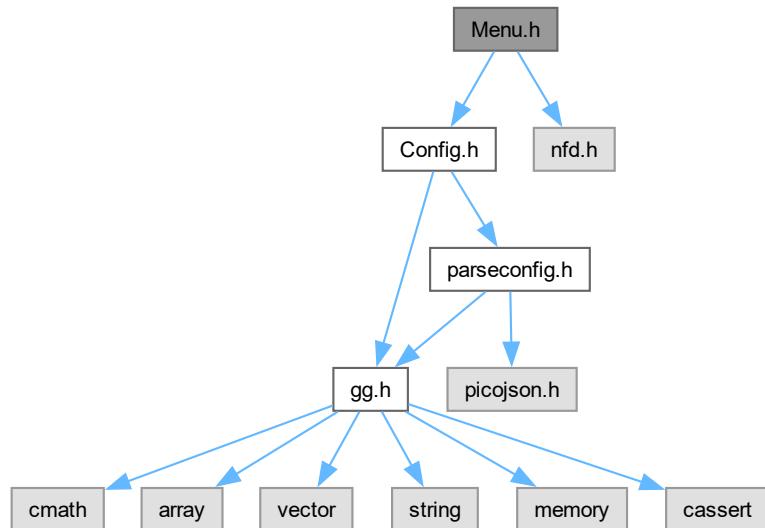
00069 }
00070
00071 // 描画する
00072 // メニューの表示位置を決定する
00073 //
00074 void Menu::draw()
00075 {
00076 #if defined(IMGUI_VERSION)
00077 //
00078 // ImGui によるユーザインターフェース
00079 //
00080 // ImGui のフレームを準備する
00081 ImGui::NewFrame();
00082
00083 // メニュー表示開始
00084 ImGui::Begin(u8"コントロールパネル");
00085
00086 // 光源
00087 if (ImGui::SliderFloat3(u8"光源位置", settings.light.position.data(), -10.0f, 10.0f, "%.2f"))
00088     light->loadPosition(settings.light.position);
00089 if (ImGui::ColorEdit3(u8"光源色", settings.light.diffuse.data(), ImGuiColorEditFlags_Float))
00090     light->loadDiffuse(settings.light.diffuse);
00091
00092 // メニュー表示終了
00093 ImGui::End();
00094
00095 // ImGui のフレームに描画する
00096 ImGui::Render();
00097 #endif
00098 }

```

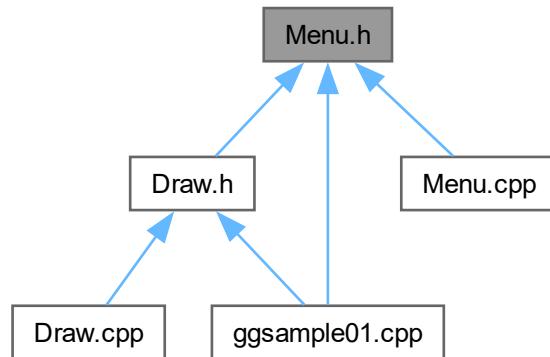
9.23 Menu.h ファイル

```
#include "Config.h"
#include "nfd.h"
```

Menu.h の依存先関係図:



被依存関係図:



クラス

- class [Menu](#)

9.23.1 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。

9.24 Menu.h

[詳解]

```
00001 #pragma once
00002
00010
00011 // 構成データ
00012 #include "Config.h"
00013
00014 // ファイルダイアログ
00015 #include "nfd.h"
00016
00020 class Menu
00021 {
00022     // 図形の描画クラスから参照する
```

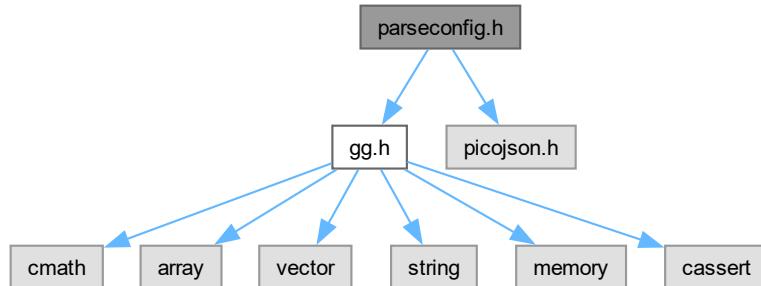
```

00023     friend class Draw;
00024
00025     // オリジナルの構成データ
00026     const Config& defaults;
00027
00028     // 構成データのコピー
00029     Config settings;
00030
00031     // CAD データ
00032     std::unique_ptr<const GgSimpleObj> model;
00033
00034     // 光源データ
00035     std::unique_ptr<const GgSimpleShader::LightBuffer> light;
00036
00037     // シエーダ
00038     std::unique_ptr<const GgSimpleShader> shader;
00039
00040     // ファイルパスを取得する
00041     bool getFilePath(std::string& path, const nfdfilteritem_t* filter);
00042
00043 public:
00044
00045     Menu(const Config& config);
00046
00047     // コピーコンストラクタは封じる
00048     Menu(const Menu& menu) = delete;
00049
00050     virtual ~Menu();
00051
00052     // 代入演算子は封じる
00053     Menu& operator=(const Menu& menu) = delete;
00054
00055     void draw();
00056
00057 };
00058
00059 }
```

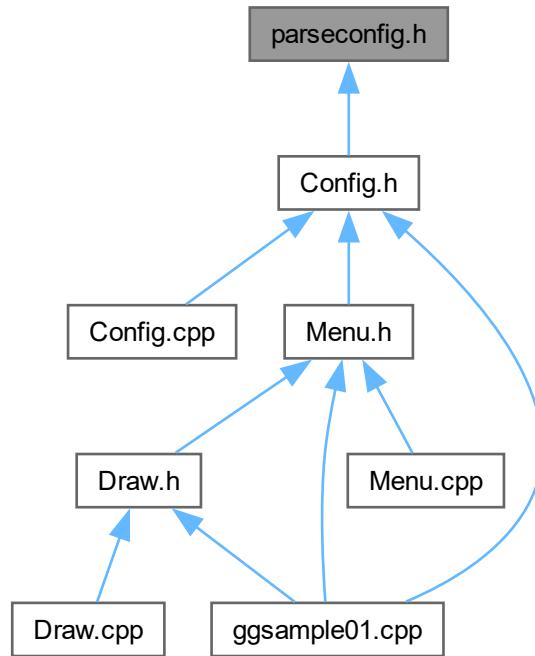
9.25 parseconfig.h ファイル

```
#include "gg.h"
#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 , std::size_t U>
 bool **getValue** (const picojson::object &object, const std::string &key, std::array< T, U > &vector)
- bool **getVector** (const picojson::object &object, const std::string &key, **GgVector** &vector)
- bool **getString** (const picojson::object &object, const std::string &key, std::string &string)
- template<std::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 , std::size_t U>
 void **setValue** (picojson::object &object, const std::string &key, const std::array< T, U > &vector)
- void **setVector** (picojson::object &object, const std::string &key, const **GgVector** &vector)
- void **setString** (picojson::object &object, const std::string &key, const std::string &string)
- template<std::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)

9.25.1 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](#) に定義がります。

9.25.2 関数詳解

9.25.2.1 getString() [1/3]

```
template<std::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 142 行目に定義がります。

9.25.2.2 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 118 行目に定義があります。

被呼び出し関係図:



9.25.2.3 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 175 行目に定義があります。

9.25.2.4 getValue() [1/2]

```
template<typename T , std::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 52 行目に定義がります。

9.25.2.5 `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 27 行目に定義がります。

被呼び出し関係図:



9.25.2.6 `getVector()`

```
bool getVector (
    const picojson::object & object,
    const std::string & key,
    GgVector & vector) [inline]
```

構成ファイルの JSON オブジェクトから 4 要素の数値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>vector</i>	取得した数値の配列を格納する変数

parseconfig.h の 85 行目に定義がります。

被呼び出し関係図:



9.25.2.7 setString() [1/3]

```
template<std::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

<i>U</i>	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 287 行目に定義がります。

9.25.2.8 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 272 行目に定義があります。

被呼び出し関係図:



9.25.2.9 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 312 行目に定義があります。

9.25.2.10 setValue() [1/2]

```
template<typename T , std::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 222 行目に定義があります。

9.25.2.11 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 206 行目に定義があります。

被呼び出し関係図:



9.25.2.12 setVector()

```
void setVector (
    picojson::object & object,
    const std::string & key,
    const GgVector & vector) [inline]
```

構成ファイルの JSON オブジェクトに 4 要素の数値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>vector</i>	設定する数値の配列

parseconfig.h の 247 行目に定義があります。

被呼び出し関係図:



9.26 parseconfig.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 // JSON
00016 #include "picojson.h"
00017
00026 template <typename T>
00027 bool getValue(const picojson::object& object,
00028     const std::string& key, T& scalar)
00029 {
00030     // key に一致するオブジェクトを探す
00031     const auto& value{ object.find(key) };
00032
00033     // オブジェクトが無いか数値でなかったら戻る
00034     if (value == object.end() || !value->second.is<double>()) return false;
00035
00036     // 数値として格納する
00037     scalar = static_cast<T>(value->second.get<double>());
00038
00039     return true;
00040 }
00041
00051 template <typename T, std::size_t U>
00052 bool getValue(const picojson::object& object,
00053     const std::string& key, std::array<T, U>& vector)
00054 {
00055     // key に一致するオブジェクトを探す
```

```
00056     const auto&& value{ object.find(key) };
00057
00058     // オブジェクトが無いか配列でなかったら戻る
00059     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00060
00061     // 配列を取り出す
00062     const auto& array{ value->second.get<picojson::array>() };
00063
00064     // 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
00065     const auto n{ std::min(U, array.size()) };
00066
00067     // 配列の要素について
00068     for (std::size_t i = 0; i < n; ++i)
00069     {
00070         // 要素が数値なら格納する
00071         if (array[i].is<double>()) vector[i] = static_cast<T>(array[i].get<double>());
00072     }
00073
00074     return true;
00075 }
00076
00077
00078 inline
00079 bool getVector(const picojson::object& object,
00080                 const std::string& key, GgVector& vector)
00081 {
00082     // key に一致するオブジェクトを探す
00083     const auto&& value{ object.find(key) };
00084
00085     // オブジェクトが無いか配列でなかったら戻る
00086     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00087
00088     // 配列を取り出す
00089     const auto& array{ value->second.get<picojson::array>() };
00090
00091     // 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
00092     const auto n{ std::min(sizeof(GgVector) / sizeof(GLfloat), array.size()) };
00093
00094     // 配列の要素について
00095     for (std::size_t i = 0; i < n; ++i)
00096     {
00097         // 要素が数値なら格納する
00098         if (array[i].is<double>()) vector[i] = static_cast<GLfloat>(array[i].get<double>());
00099     }
00100
00101     return true;
00102 }
00103
00104
00105 inline
00106 bool getString(const picojson::object& object,
00107                 const std::string& key, std::string& string)
00108 {
00109     // key に一致するオブジェクトを探す
00110     const auto&& value{ object.find(key) };
00111
00112     // オブジェクトが無いか文字列でなかったら戻る
00113     if (value == object.end() || !value->second.is<std::string>()) return false;
00114
00115     // 文字列として格納する
00116     string = value->second.get<std::string>();
00117
00118     return true;
00119 }
00120
00121
00122 template <std::size_t U>
00123 bool getString(const picojson::object& object,
00124                 const std::string& key, std::array<std::string, U>& strings)
00125 {
00126     // key に一致するオブジェクトを探す
00127     const auto&& value{ object.find(key) };
00128
00129     // オブジェクトが無いか配列でなかったら戻る
00130     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00131
00132     // 配列を取り出す
00133     const auto& array{ value->second.get<picojson::array>() };
00134
00135     // 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
00136     const auto n{ std::min(U, array.size()) };
00137
00138     // 配列の要素について
00139     for (std::size_t i = 0; i < n; ++i)
00140     {
00141         // 要素が文字列なら文字列として格納する
00142         strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
00143     }
00144
00145     return true;
00146 }
```

```
00165 }
00166
00174 inline
00175 bool getString(const picojson::object& object,
00176   const std::string& key, std::vector<std::string>& strings)
00177 {
00178   // key に一致するオブジェクトを探す
00179   const auto& value{ object.find(key) };
00180
00181   // オブジェクトが無いか配列でなかったら戻る
00182   if (value == object.end() || !value->second.is<picojson::array>()) return false;
00183
00184   // 配列を取り出す
00185   const auto& array{ value->second.get<picojson::array>() };
00186
00187   // 配列のすべての要素について
00188   for (auto& element : array)
00189   {
00190     // 要素が文字列なら文字列として格納する
00191     strings.emplace_back(element.is<std::string>() ? element.get<std::string>() : "");
00192   }
00193
00194   return true;
00195 }
00196
00205 template <typename T>
00206 void setValue(picojson::object& object,
00207   const std::string& key, const T& scalar)
00208 {
00209   object.emplace(key, picojson::value(static_cast<double>(scalar)));
00210 }
00211
00221 template <typename T, std::size_t U>
00222 void setValue(picojson::object& object,
00223   const std::string& key, const std::array<T, U>& vector)
00224 {
00225   // picojson の配列
00226   picojson::array array;
00227
00228   // 配列のすべての要素について
00229   for (const auto& element : vector)
00230   {
00231     // 要素を picojson::array に追加する
00232     array.emplace_back(picojson::value(static_cast<double>(element)));
00233   }
00234
00235   // オブジェクトに追加する
00236   object.emplace(key, array);
00237 }
00238
00246 inline
00247 void setVector(picojson::object& object,
00248   const std::string& key, const GgVector& vector)
00249 {
00250   // picojson の配列
00251   picojson::array array;
00252
00253   // ベクトルのすべての要素について
00254   for (const auto& element : vector)
00255   {
00256     // 要素を picojson::array に追加する
00257     array.emplace_back(picojson::value(static_cast<double>(element)));
00258   }
00259
00260   // オブジェクトに追加する
00261   object.emplace(key, array);
00262 }
00263
00271 inline
00272 void setString(picojson::object& object,
00273   const std::string& key, const std::string& string)
00274 {
00275   object.emplace(key, picojson::value(string));
00276 }
00277
00286 template <std::size_t U>
00287 void setString(picojson::object& object,
00288   const std::string& key, const std::array<std::string, U>& strings)
00289 {
00290   // picojson の配列
00291   picojson::array array;
00292
00293   // 配列のすべての要素について
00294   for (const auto& string : strings)
00295   {
00296     // 要素を picojson::array に追加する
00297     array.emplace_back(picojson::value(string));
```

```
00298     }
00299
00300     // オブジェクトに追加する
00301     object.emplace(key, array);
00302 }
00303
00311 inline
00312 void setString(picojson::object& object,
00313     const std::string& key, const std::vector<std::string>& strings)
00314 {
00315     // picojson の配列
00316     picojson::array array;
00317
00318     // 配列のすべての要素について
00319     for (auto& string : strings)
00320     {
00321         // 要素を picojson::array に追加する
00322         array.emplace_back(picojson::value(string));
00323     }
00324
00325     // オブジェクトに追加する
00326     object.emplace(key, array);
00327 }
```

9.27 README.md ファイル

Index

-ggError
 gg, 18
-ggFBOError
 gg, 18
~Config
 Config, 82
~Draw
 Draw, 85
~GgApp
 GgApp, 87
~GgBuffer
 gg::GgBuffer< T >, 90
~GgColorTexture
 gg::GgColorTexture, 96
~GgElements
 gg::GgElements, 100
~GgMatrix
 gg::GgMatrix, 107
~GgNormalTexture
 gg::GgNormalTexture, 161
~GgPointShader
 gg::GgPointShader, 169
~GgPoints
 gg::GgPoints, 165
~GgQuaternion
 gg::GgQuaternion, 182
~GgShader
 gg::GgShader, 242
~GgShape
 gg::GgShape, 245
~GgSimpleObj
 gg::GgSimpleObj, 249
~GgSimpleShader
 gg::GgSimpleShader, 254
~GgTexture
 gg::GgTexture, 267
~GgTrackball
 gg::GgTrackball, 275
~GgTriangles
 gg::GgTriangles, 284
~GgUniformBuffer
 gg::GgUniformBuffer< T >, 288
~GgVector
 gg::GgVector, 296
~GgVertexArray
 gg::GgVertexArray, 311
~LightBuffer
 gg::GgSimpleShader::LightBuffer, 316
~MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 327
~Menu
 Menu, 335
~Window
 GgApp::Window, 338
add
 gg::GgQuaternion, 183, 184
ambient
 gg::GgSimpleShader::Light, 313
 gg::GgSimpleShader::Material, 324
begin
 gg::GgTrackball, 275
bind
 gg::GgBuffer< T >, 90
 gg::GgTexture, 267
 gg::GgUniformBuffer< T >, 288
 gg::GgVertexArray, 311
BindingPoints
 gg, 18
Config, 81
 ~Config, 82
 Config, 81
 getHeight, 82
 getWidth, 82
 load, 83
 Menu, 84
 save, 83
Config.cpp, 361
 defaultLight, 362
Config.h, 364
CONFIG.FILE
 ggsample01.cpp, 532
conjugate
 gg::GgQuaternion, 185
copy
 gg::GgBuffer< T >, 90
 gg::GgUniformBuffer< T >, 288
defaultLight
 Config.cpp, 362
diffuse
 gg::GgSimpleShader::Light, 313
 gg::GgSimpleShader::Material, 324
distance3
 gg::GgVector, 296
distance4
 gg::GgVector, 297

divide
 gg::GgQuaternion, 186, 187
 dot3
 gg::GgVector, 297
 dot4
 gg::GgVector, 298
 Draw, 85
 ~Draw, 85
 Draw, 85
 draw, 86
 Menu, 335
 draw
 Draw, 86
 gg::GgElements, 101
 gg::GgPoints, 165
 gg::GgShape, 245
 gg::GgSimpleObj, 249
 gg::GgTriangles, 284
 Menu, 335
 Draw.cpp, 367
 Draw.h, 369

 end
 gg::GgTrackball, 275
 euler
 gg::GgQuaternion, 188, 189

 fill
 gg::GgUniformBuffer< T >, 288
 frustum
 gg::GgMatrix, 107

 get
 gg::GgMatrix, 108, 109
 gg::GgPointShader, 169
 gg::GgQuaternion, 190
 gg::GgShader, 243
 gg::GgShape, 246
 gg::GgSimpleObj, 249
 gg::GgTrackball, 276
 gg::GgVertexArray, 311
 GgApp::Window, 338
 getAltArrowX
 GgApp::Window, 338
 getAltArrowY
 GgApp::Window, 339
 getAltIArrow
 GgApp::Window, 340
 getArrow
 GgApp::Window, 340, 341
 getArrowX
 GgApp::Window, 341
 getArrowY
 GgApp::Window, 342
 getAspect
 GgApp::Window, 343
 getBuffer
 gg::GgBuffer< T >, 91
 gg::GgPoints, 165

 gg::GgTriangles, 285
 gg::GgUniformBuffer< T >, 289
 getConjugateMatrix
 gg::GgQuaternion, 190–192
 getControlArrow
 GgApp::Window, 343
 getControlArrowX
 GgApp::Window, 344
 getControlArrowY
 GgApp::Window, 344
 getCount
 gg::GgBuffer< T >, 91
 gg::GgPoints, 166
 gg::GgTriangles, 285
 gg::GgUniformBuffer< T >, 289
 getFboHeight
 GgApp::Window, 345
 getFboSize
 GgApp::Window, 345, 346
 getFboWidth
 GgApp::Window, 346
 getHeight
 Config, 82
 gg::GgTexture, 267
 GgApp::Window, 346
 getIndexBuffer
 gg::GgElements, 101
 getIndexCount
 gg::GgElements, 101
 getKey
 GgApp::Window, 347
 getLastKey
 GgApp::Window, 347
 getMatrix
 gg::GgQuaternion, 192–194
 gg::GgTrackball, 276
 getMode
 gg::GgShape, 246
 getMouse
 GgApp::Window, 347, 348
 getMouseX
 GgApp::Window, 348
 getMouseY
 GgApp::Window, 348
 getQuaternion
 gg::GgTrackball, 276
 getRotation
 GgApp::Window, 349
 getRotationMatrix
 GgApp::Window, 349
 getScale
 gg::GgTrackball, 277
 getScrollMatrix
 GgApp::Window, 349
 getShiftArrow
 GgApp::Window, 350
 getShiftArrowX
 GgApp::Window, 350

getShiftArrowY
 GgApp::Window, 351
getSize
 gg::GgTexture, 267, 268
 GgApp::Window, 352
getStart
 gg::GgTrackball, 277, 278
getStride
 gg::GgBuffer< T >, 91
 gg::GgUniformBuffer< T >, 289
getString
 parseconfig.h, 542, 543
getTarget
 gg::GgBuffer< T >, 91
 gg::GgUniformBuffer< T >, 290
getTexture
 gg::GgTexture, 268
getTranslation
 GgApp::Window, 352
getTranslationMatrix
 GgApp::Window, 353
getUserPointer
 GgApp::Window, 353
getValue
 parseconfig.h, 543, 544
getVector
 parseconfig.h, 544
getWheel
 GgApp::Window, 353, 354
getWheelX
 GgApp::Window, 354
getWheelY
 GgApp::Window, 354
getWidth
 Config, 82
 gg::GgTexture, 268
 GgApp::Window, 354
gg, 15
 _ggError, 18
 _ggFBOError, 18
 BindingPoints, 18
 ggArraysObj, 19
 ggBufferAlignment, 80
 ggConjugate, 19
 ggCreateComputeShader, 20
 ggCreateNormalMap, 20
 ggCreateShader, 21
 ggCross, 22, 23
 ggDistance3, 23, 24
 ggDistance4, 25, 26
 ggDot3, 27
 ggDot4, 28
 ggElementsMesh, 29
 ggElementsObj, 30
 ggElementsSphere, 31
 ggEllipse, 31
 ggEulerQuaternion, 32
 ggFrustum, 33
 ggIdentity, 34
 ggIdentityQuaternion, 34
 ggInit, 35
 ggInvert, 35, 36
 ggLength3, 36, 37
 ggLength4, 38
 ggLoadComputeShader, 39
 ggLoadHeight, 40
 ggLoadImage, 40
 ggLoadShader, 41, 42
 ggLoadSimpleObj, 43, 44
 ggLoadTexture, 44
 ggLookat, 45, 46
 ggMatrixQuaternion, 48
 ggNorm, 49
 ggNormal, 49
 ggNormalize, 50
 ggNormalize3, 50–52
 ggNormalize4, 53, 54
 ggOrthogonal, 55
 ggPerspective, 56
 ggPointsCube, 57
 ggPointsSphere, 57
 ggQuaternionMatrix, 58
 ggQuaternionTransposeMatrix, 58
 ggReadImage, 59
 ggRectangle, 60
 ggRotate, 61–63
 ggRotateQuaternion, 64, 65
 ggRotateX, 66
 ggRotateY, 66
 ggRotateZ, 67
 ggSaveColor, 67
 ggSaveDepth, 69
 ggSaveTga, 69
 ggScale, 70, 71
 ggSlerp, 72–74
 ggTranslate, 75, 76
 ggTranspose, 77
 LightBindingPoint, 18
 MaterialBindingPoint, 18
 operator+, 78
 operator-, 78, 79
 operator/, 79
 operator*, 77
 gg.cpp, 370
 gg.h, 447
 ggError, 452
 ggFBOError, 452
 pathChar, 452
 pathString, 452
 TCharToUtf8, 453
 Utf8ToTChar, 453
 gg::GgBuffer< T >, 89
 ~GgBuffer, 90
 bind, 90
 copy, 90
 getBuffer, 91

getCount, 91
 getStride, 91
 getTarget, 91
 GgBuffer, 89, 90
 map, 92
 operator=, 92, 93
 read, 93
 send, 93
 unbind, 93
 unmap, 94
 gg::GgColorTexture, 94
 ~GgColorTexture, 96
 GgColorTexture, 95
 load, 96, 97
 gg::GgElements, 98
 ~GgElements, 100
 draw, 101
 getIndexBuffer, 101
 getIndexCount, 101
 GgElements, 100
 load, 101
 send, 102
 gg::GgMatrix, 103
 ~GgMatrix, 107
 frustum, 107
 get, 108, 109
 GgMatrix, 105–107
 invert, 110
 loadFrustum, 110
 loadIdentity, 111
 loadInvert, 111, 112
 loadLookat, 112–114
 loadNormal, 115
 loadOrthogonal, 116
 loadPerspective, 117
 loadRotate, 117–120
 loadRotateX, 121
 loadRotateY, 122
 loadRotateZ, 122
 loadScale, 123, 124
 loadTranslate, 125, 126
 loadTranspose, 127
 lookat, 128, 129
 normal, 131
 operator+, 136, 137
 operator+=, 137, 138
 operator-, 138, 139
 operator-=, 139, 141
 operator/, 141, 142
 operator/=, 143
 operator=, 144, 145
 operator*, 132, 134
 operator*=, 135, 136
 orthogonal, 145
 perspective, 146
 projection, 146, 147
 rotate, 148–150
 rotateX, 151
 rotateY, 152
 rotateZ, 152
 scale, 154, 155
 translate, 156, 157
 transpose, 158
 gg::GgNormalTexture, 159
 ~GgNormalTexture, 161
 GgNormalTexture, 160
 load, 161, 162
 gg::GgPoints, 163
 ~GgPoints, 165
 draw, 165
 getBuffer, 165
 getCount, 166
 GgPoints, 164
 load, 166
 operator bool, 166
 operator!, 166
 send, 167
 gg::GgPointShader, 167
 ~GgPointShader, 169
 get, 169
 GgPointShader, 168, 169
 load, 169, 171
 loadMatrix, 171, 172
 loadModelviewMatrix, 172, 173
 loadProjectionMatrix, 173, 174
 unuse, 174
 use, 174–176
 gg::GgQuaternion, 176
 ~GgQuaternion, 182
 add, 183, 184
 conjugate, 185
 divide, 186, 187
 euler, 188, 189
 get, 190
 getConjugateMatrix, 190–192
 getMatrix, 192–194
 GgQuaternion, 180, 182
 invert, 194
 loadAdd, 195, 196
 loadConjugate, 197, 198
 loadDivide, 198, 200, 202
 loadEuler, 203, 204
 loadIdentity, 205
 loadInvert, 205, 206
 loadMatrix, 207
 loadMultiply, 208, 209
 loadNormalize, 210, 211
 loadRotate, 211, 212
 loadRotateX, 213
 loadRotateY, 214
 loadRotateZ, 214
 loadSlerp, 215–217
 loadSubtract, 217–219
 multiply, 220, 221
 norm, 221
 normalize, 222

operator+, 225, 226
operator+=, 226, 227
operator-, 227, 228
operator-=, 228, 229
operator/, 230
operator/=, 231, 232
operator=, 232, 233
operator*, 223
operator*=, 224
rotate, 234, 235
rotateX, 236
rotateY, 236
rotateZ, 237
slerp, 237, 238
subtract, 238, 239
gg::GgShader, 241
~GgShader, 242
get, 243
GgShader, 241, 242
operator=, 243
unuse, 243
use, 243
gg::GgShape, 244
~GgShape, 245
draw, 245
get, 246
getMode, 246
GgShape, 245
operator bool, 247
operator!, 247
setMode, 247
gg::GgSimpleObj, 248
~GgSimpleObj, 249
draw, 249
get, 249
GgSimpleObj, 248
operator bool, 250
operator!, 250
gg::GgSimpleShader, 251
~GgSimpleShader, 254
GgSimpleShader, 253
load, 254
loadMatrix, 255, 256
loadModelviewMatrix, 257, 258
operator=, 258
use, 259–264
gg::GgSimpleShader::Light, 312
ambient, 313
diffuse, 313
position, 313
specular, 313
gg::GgSimpleShader::LightBuffer, 314
~LightBuffer, 316
LightBuffer, 315, 316
load, 316, 317
loadAmbient, 317, 318
loadColor, 318
loadDiffuse, 318, 319
loadPosition, 320, 321
loadSpecular, 321, 322
select, 322
gg::GgSimpleShader::Material, 323
ambient, 324
diffuse, 324
shininess, 324
specular, 324
gg::GgSimpleShader::MaterialBuffer, 325
~MaterialBuffer, 327
load, 327, 328
loadAmbient, 328, 329
loadAmbientAndDiffuse, 329, 330
loadDiffuse, 330, 331
loadShininess, 331, 332
loadSpecular, 332, 333
MaterialBuffer, 327
select, 333
gg::GgTexture, 265
~GgTexture, 267
bind, 267
getHeight, 267
getSize, 267, 268
getTexture, 268
getWidth, 268
GgTexture, 266
operator=, 269
swapRandB, 269
unbind, 269
gg::GgTrackball, 270
~GgTrackball, 275
begin, 275
end, 275
get, 276
getMatrix, 276
getQuaternion, 276
getScale, 277
getStart, 277, 278
GgTrackball, 274
motion, 278
operator=, 279
region, 279, 280
reset, 280
rotate, 281
gg::GgTriangles, 282
~GgTriangles, 284
draw, 284
getBuffer, 285
getCount, 285
GgTriangles, 283, 284
load, 285
send, 286
gg::GgUniformBuffer< T >, 286
~GgUniformBuffer, 288
bind, 288
copy, 288
fill, 288
getBuffer, 289

getCount, 289
 getStride, 289
 getTarget, 290
 GgUniformBuffer, 287
 load, 290
 map, 291
 read, 291
 send, 292
 unbind, 292
 unmap, 292
gg::GgVector, 293
 ~GgVector, 296
 distance3, 296
 distance4, 297
 dot3, 297
 dot4, 298
 GgVector, 294–296
 length3, 298
 length4, 299
 normalize3, 299
 normalize4, 300
 operator+, 302
 operator+=, 303
 operator-, 304
 operator-=, 304, 305
 operator/, 305
 operator/=, 306
 operator=, 307
 operator*, 300, 301
 operator*+, 301, 302
gg::GgVertex, 307
 GgVertex, 308, 309
 normal, 309
 position, 309
gg::GgVertexArray, 310
 ~GgVertexArray, 311
 bind, 311
 get, 311
 GgVertexArray, 310
 operator=, 311
GG_BUTTON_COUNT
 GgApp.h, 523
GG_INTERFACE_COUNT
 GgApp.h, 523
GG_USE_IMGUI
 GgApp.h, 524
GgApp, 86
 ~GgApp, 87
 GgApp, 87
 main, 88
 operator=, 88
GgApp.cpp, 508
GgApp.h, 522
 GG_BUTTON_COUNT, 523
 GG_INTERFACE_COUNT, 523
 GG_USE_IMGUI, 524
GgApp::Window, 335
 ~Window, 338
 getCount, 338
 getAltArrowX, 339
 getAltArrowY, 339
 getAltIArrow, 340
 getArrow, 340, 341
 getArrowX, 341
 getArrowY, 342
 getAspect, 343
 getControlArrow, 343
 getControlArrowX, 344
 getControlArrowY, 344
 getFboHeight, 345
 getFboSize, 345, 346
 getFboWidth, 346
 getHeight, 346
 getKey, 347
 getLastKey, 347
 getMouse, 347, 348
 getMouseX, 348
 getMouseY, 348
 getRotation, 349
 getRotationMatrix, 349
 getScrollMatrix, 349
 getShiftArrow, 350
 getShiftArrowX, 350
 getShiftArrowY, 351
 getSize, 352
 getTranslation, 352
 getTranslationMatrix, 353
 getUserPointer, 353
 getWheel, 353, 354
 getWheelX, 354
 getWheelY, 354
 getWidth, 354
 operator bool, 355
 operator=, 355
 reset, 355
 resetRotation, 356
 resetTranslation, 356
 restoreViewport, 356
 selectInterface, 357
 setClose, 357
 setKeyboardFunc, 357
 setMouseFunc, 357
 setResizeFunc, 358
 setUserPointer, 358
 setVelocity, 358
 setWheelFunc, 358
 shouldClose, 359
 swapBuffers, 359
 updateViewport, 359
 Window, 337, 338
ggArraysObj
 gg, 19
GgBuffer
 gg::GgBuffer< T >, 89, 90
ggBufferAlignment
 gg, 80

GgColorTexture
 gg::GgColorTexture, 95
ggConjugate
 gg, 19
ggCreateComputeShader
 gg, 20
ggCreateNormalMap
 gg, 20
ggCreateShader
 gg, 21
ggCross
 gg, 22, 23
ggDistance3
 gg, 23, 24
ggDistance4
 gg, 25, 26
ggDot3
 gg, 27
ggDot4
 gg, 28
GgElements
 gg::GgElements, 100
ggElementsMesh
 gg, 29
ggElementsObj
 gg, 30
ggElementsSphere
 gg, 31
ggEllipse
 gg, 31
ggError
 gg.h, 452
ggEulerQuaternion
 gg, 32
ggFBOError
 gg.h, 452
ggFrustum
 gg, 33
ggIdentity
 gg, 34
ggIdentityQuaternion
 gg, 34
ggInit
 gg, 35
ggInvert
 gg, 35, 36
ggLength3
 gg, 36, 37
ggLength4
 gg, 38
ggLoadComputeShader
 gg, 39
ggLoadHeight
 gg, 40
ggLoadImage
 gg, 40
ggLoadShader
 gg, 41, 42
ggLoadSimpleObj
 gg, 43, 44
ggLoadTexture
 gg, 44
ggLookat
 gg, 45, 46
GgMatrix
 gg::GgMatrix, 105–107
ggMatrixQuaternion
 gg, 48
ggNorm
 gg, 49
ggNormal
 gg, 49
ggNormalize
 gg, 50
ggNormalize3
 gg, 50–52
ggNormalize4
 gg, 53, 54
GgNormalTexture
 gg::GgNormalTexture, 160
ggOrthogonal
 gg, 55
ggPerspective
 gg, 56
GgPoints
 gg::GgPoints, 164
ggPointsCube
 gg, 57
GgPointShader
 gg::GgPointShader, 168, 169
ggPointsSphere
 gg, 57
GgQuaternion
 gg::GgQuaternion, 180, 182
ggQuaternionMatrix
 gg, 58
ggQuaternionTransposeMatrix
 gg, 58
ggReadImage
 gg, 59
ggRectangle
 gg, 60
ggRotate
 gg, 61–63
ggRotateQuaternion
 gg, 64, 65
ggRotateX
 gg, 66
ggRotateY
 gg, 66
ggRotateZ
 gg, 67
ggsample01, 3
ggsample01.cpp, 532
 CONFIG_FILE, 532
 PROJECT_NAME, 532

ggSaveColor
 gg, 67
 ggSaveDepth
 gg, 69
 ggSaveTga
 gg, 69
 ggScale
 gg, 70, 71
 GgShader
 gg::GgShader, 241, 242
 GgShape
 gg::GgShape, 245
 GgSimpleObj
 gg::GgSimpleObj, 248
 GgSimpleShader
 gg::GgSimpleShader, 253
 ggSlerp
 gg, 72–74
 GgTexture
 gg::GgTexture, 266
 GgTrackball
 gg::GgTrackball, 274
 ggTranslate
 gg, 75, 76
 ggTranspose
 gg, 77
 GgTriangles
 gg::GgTriangles, 283, 284
 GgUniformBuffer
 gg::GgUniformBuffer< T >, 287
 GgVector
 gg::GgVector, 294–296
 GgVertex
 gg::GgVertex, 308, 309
 GgVertexArray
 gg::GgVertexArray, 310

 HEADER_STR
 main.cpp, 534

 invert
 gg::GgMatrix, 110
 gg::GgQuaternion, 194

 length3
 gg::GgVector, 298
 length4
 gg::GgVector, 299
 LightBindingPoint
 gg, 18
 LightBuffer
 gg::GgSimpleShader::LightBuffer, 315, 316
 load
 Config, 83
 gg::GgColorTexture, 96, 97
 gg::GgElements, 101
 gg::GgNormalTexture, 161, 162
 gg::GgPoints, 166
 gg::GgPointShader, 169, 171
 gg::GgSimpleShader, 254
 gg::GgSimpleShader::LightBuffer, 316, 317
 gg::GgSimpleShader::MaterialBuffer, 327, 328
 gg::GgTriangles, 285
 gg::GgUniformBuffer< T >, 290
 loadAdd
 gg::GgQuaternion, 195, 196
 loadAmbient
 gg::GgSimpleShader::LightBuffer, 317, 318
 gg::GgSimpleShader::MaterialBuffer, 328, 329
 loadAmbientAndDiffuse
 gg::GgSimpleShader::MaterialBuffer, 329, 330
 loadColor
 gg::GgSimpleShader::LightBuffer, 318
 loadConjugate
 gg::GgQuaternion, 197, 198
 loadDiffuse
 gg::GgSimpleShader::LightBuffer, 318, 319
 gg::GgSimpleShader::MaterialBuffer, 330, 331
 loadDivide
 gg::GgQuaternion, 198, 200, 202
 loadEuler
 gg::GgQuaternion, 203, 204
 loadFrustum
 gg::GgMatrix, 110
 loadIdentity
 gg::GgMatrix, 111
 gg::GgQuaternion, 205
 loadInvert
 gg::GgMatrix, 111, 112
 gg::GgQuaternion, 205, 206
 loadLookat
 gg::GgMatrix, 112–114
 loadMatrix
 gg::GgPointShader, 171, 172
 gg::GgQuaternion, 207
 gg::GgSimpleShader, 255, 256
 loadModelviewMatrix
 gg::GgPointShader, 172, 173
 gg::GgSimpleShader, 257, 258
 loadMultiply
 gg::GgQuaternion, 208, 209
 loadNormal
 gg::GgMatrix, 115
 loadNormalize
 gg::GgQuaternion, 210, 211
 loadOrthogonal
 gg::GgMatrix, 116
 loadPerspective
 gg::GgMatrix, 117
 loadPosition
 gg::GgSimpleShader::LightBuffer, 320, 321
 loadProjectionMatrix
 gg::GgPointShader, 173, 174
 loadRotate
 gg::GgMatrix, 117–120
 gg::GgQuaternion, 211, 212
 loadRotateX

gg::GgMatrix, 121
gg::GgQuaternion, 213
loadRotateY
 gg::GgMatrix, 122
 gg::GgQuaternion, 214
loadRotateZ
 gg::GgMatrix, 122
 gg::GgQuaternion, 214
loadScale
 gg::GgMatrix, 123, 124
loadShininess
 gg::GgSimpleShader::MaterialBuffer, 331, 332
loadSlerp
 gg::GgQuaternion, 215–217
loadSpecular
 gg::GgSimpleShader::LightBuffer, 321, 322
 gg::GgSimpleShader::MaterialBuffer, 332, 333
loadSubtract
 gg::GgQuaternion, 217–219
loadTranslate
 gg::GgMatrix, 125, 126
loadTranspose
 gg::GgMatrix, 127
lookat
 gg::GgMatrix, 128, 129

main
 GgApp, 88
 main.cpp, 535
main.cpp, 533
 HEADER_STR, 534
 main, 535
map
 gg::GgBuffer< T >, 92
 gg::GgUniformBuffer< T >, 291
MaterialBindingPoint
 gg, 18
MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 327
Menu, 334
 ~Menu, 335
 Config, 84
 Draw, 335
 draw, 335
 Menu, 334
 operator=, 335
Menu.cpp, 536
Menu.h, 538
motion
 gg::GgTrackball, 278
multiply
 gg::GgQuaternion, 220, 221

norm
 gg::GgQuaternion, 221
normal
 gg::GgMatrix, 131
 gg::GgVertex, 309
normalize
 gg::GgQuaternion, 222
 gg::GgVector, 299
normalize3
 gg::GgVector, 299
normalize4
 gg::GgVector, 300

operator bool
 gg::GgPoints, 166
 gg::GgShape, 247
 gg::GgSimpleObj, 250
 GgApp::Window, 355
operator!
 gg::GgPoints, 166
 gg::GgShape, 247
 gg::GgSimpleObj, 250
operator+
 gg, 78
 gg::GgMatrix, 136, 137
 gg::GgQuaternion, 225, 226
 gg::GgVector, 302
operator+=
 gg::GgMatrix, 137, 138
 gg::GgQuaternion, 226, 227
 gg::GgVector, 303
operator-
 gg, 78, 79
 gg::GgMatrix, 138, 139
 gg::GgQuaternion, 227, 228
 gg::GgVector, 304
operator-=
 gg::GgMatrix, 139, 141
 gg::GgQuaternion, 228, 229
 gg::GgVector, 304, 305
operator/
 gg, 79
 gg::GgMatrix, 141, 142
 gg::GgQuaternion, 230
 gg::GgVector, 305
operator/=
 gg::GgMatrix, 143
 gg::GgQuaternion, 231, 232
 gg::GgVector, 306
operator=
 gg::GgBuffer< T >, 92, 93
 gg::GgMatrix, 144, 145
 gg::GgQuaternion, 232, 233
 gg::GgShader, 243
 gg::GgSimpleShader, 258
 gg::GgTexture, 269
 gg::GgTrackball, 279
 gg::GgVector, 307
 gg::GgVertexArray, 311
 GgApp, 88
 GgApp::Window, 355
 Menu, 335
operator*
 gg, 77
 gg::GgMatrix, 132, 134
 gg::GgQuaternion, 223

gg::GgVector, 300, 301
operator*=
 gg::GgMatrix, 135, 136
 gg::GgQuaternion, 224
 gg::GgVector, 301, 302
orthogonal
 gg::GgMatrix, 145

parseconfig.h, 540
 getString, 542, 543
 getValue, 543, 544
 getVector, 544
 setString, 545, 546
 setValue, 546, 547
 setVector, 547
pathChar
 gg.h, 452
pathString
 gg.h, 452
perspective
 gg::GgMatrix, 146
position
 gg::GgSimpleShader::Light, 313
 gg::GgVertex, 309
PROJECT_NAME
 ggsample01.cpp, 532
projection
 gg::GgMatrix, 146, 147

read
 gg::GgBuffer< T >, 93
 gg::GgUniformBuffer< T >, 291
README.md, 551
region
 gg::GgTrackball, 279, 280
reset
 gg::GgTrackball, 280
 GgApp::Window, 355
resetRotation
 GgApp::Window, 356
resetTranslation
 GgApp::Window, 356
restoreViewport
 GgApp::Window, 356
rotate
 gg::GgMatrix, 148–150
 gg::GgQuaternion, 234, 235
 gg::GgTrackball, 281
rotateX
 gg::GgMatrix, 151
 gg::GgQuaternion, 236
rotateY
 gg::GgMatrix, 152
 gg::GgQuaternion, 236
rotateZ
 gg::GgMatrix, 152
 gg::GgQuaternion, 237

save

Config, 83
scale
 gg::GgMatrix, 154, 155
select
 gg::GgSimpleShader::LightBuffer, 322
 gg::GgSimpleShader::MaterialBuffer, 333
selectInterface
 GgApp::Window, 357
send
 gg::GgBuffer< T >, 93
 gg::GgElements, 102
 gg::GgPoints, 167
 gg::GgTriangles, 286
 gg::GgUniformBuffer< T >, 292
setClose
 GgApp::Window, 357
setKeyboardFunc
 GgApp::Window, 357
setMode
 gg::GgShape, 247
setMouseFunc
 GgApp::Window, 357
setResizeFunc
 GgApp::Window, 358
setString
 parseconfig.h, 545, 546
setUserPointer
 GgApp::Window, 358
setValue
 parseconfig.h, 546, 547
setVector
 parseconfig.h, 547
setVelocity
 GgApp::Window, 358
setWheelFunc
 GgApp::Window, 358
shininess
 gg::GgSimpleShader::Material, 324
shouldClose
 GgApp::Window, 359
slerp
 gg::GgQuaternion, 237, 238
specular
 gg::GgSimpleShader::Light, 313
 gg::GgSimpleShader::Material, 324
subtract
 gg::GgQuaternion, 238, 239
swapBuffers
 GgApp::Window, 359
swapRandB
 gg::GgTexture, 269

TCharToUtf8
 gg.h, 453
translate
 gg::GgMatrix, 156, 157
transpose
 gg::GgMatrix, 158

unbind
 gg::GgBuffer< T >, [93](#)
 gg::GgTexture, [269](#)
 gg::GgUniformBuffer< T >, [292](#)

unmap
 gg::GgBuffer< T >, [94](#)
 gg::GgUniformBuffer< T >, [292](#)

unuse
 gg::GgPointShader, [174](#)
 gg::GgShader, [243](#)

updateViewport
 GgApp::Window, [359](#)

use
 gg::GgPointShader, [174–176](#)
 gg::GgShader, [243](#)
 gg::GgSimpleShader, [259–264](#)

Utf8ToTChar
 gg.h, [453](#)

Window
 GgApp::Window, [337](#), [338](#)

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., [1](#)