

ゲームグラフィックス特論

構築: Doxygen 1.10.0

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]	23
7.1.3.9 ggCross() [2/2]	24
7.1.3.10 ggDistance3() [1/2]	24
7.1.3.11 ggDistance3() [2/2]	25
7.1.3.12 ggDistance4() [1/2]	26
7.1.3.13 ggDistance4() [2/2]	26

7.1.3.14 ggDot3() [1/2]	27
7.1.3.15 ggDot3() [2/2]	28
7.1.3.16 ggDot4() [1/2]	28
7.1.3.17 ggDot4() [2/2]	29
7.1.3.18 ggElementsMesh()	30
7.1.3.19 ggElementsObj()	31
7.1.3.20 ggElementsSphere()	32
7.1.3.21 ggEllipse()	32
7.1.3.22 ggEulerQuaternion() [1/3]	33
7.1.3.23 ggEulerQuaternion() [2/3]	33
7.1.3.24 ggEulerQuaternion() [3/3]	34
7.1.3.25 ggFrustum()	35
7.1.3.26 ggIdentity()	36
7.1.3.27 ggIdentityQuaternion()	36
7.1.3.28 ggInit()	37
7.1.3.29 ggInvert() [1/2]	37
7.1.3.30 ggInvert() [2/2]	38
7.1.3.31 ggLength3() [1/2]	38
7.1.3.32 ggLength3() [2/2]	39
7.1.3.33 ggLength4() [1/2]	40
7.1.3.34 ggLength4() [2/2]	40
7.1.3.35 ggLoadComputeShader()	41
7.1.3.36 ggLoadHeight()	42
7.1.3.37 ggLoadImage()	43
7.1.3.38 ggLoadShader() [1/2]	43
7.1.3.39 ggLoadShader() [2/2]	44
7.1.3.40 ggLoadSimpleObj() [1/2]	45
7.1.3.41 ggLoadSimpleObj() [2/2]	46
7.1.3.42 ggLoadTexture()	46
7.1.3.43 ggLookat() [1/3]	47
7.1.3.44 ggLookat() [2/3]	48
7.1.3.45 ggLookat() [3/3]	49
7.1.3.46 ggMatrixQuaternion() [1/2]	50
7.1.3.47 ggMatrixQuaternion() [2/2]	51
7.1.3.48 ggNorm()	51
7.1.3.49 ggNormal()	52
7.1.3.50 ggNormalize()	52
7.1.3.51 ggNormalize3() [1/4]	53
7.1.3.52 ggNormalize3() [2/4]	54
7.1.3.53 ggNormalize3() [3/4]	54
7.1.3.54 ggNormalize3() [4/4]	55
7.1.3.55 ggNormalize4() [1/4]	55

7.1.3.56 ggNormalize4() [2/4]	56
7.1.3.57 ggNormalize4() [3/4]	57
7.1.3.58 ggNormalize4() [4/4]	57
7.1.3.59 ggOrthogonal()	58
7.1.3.60 ggPerspective()	59
7.1.3.61 ggPointsCube()	60
7.1.3.62 ggPointsSphere()	60
7.1.3.63 ggQuaternionMatrix()	61
7.1.3.64 ggQuaternionTransposeMatrix()	61
7.1.3.65 ggReadImage()	62
7.1.3.66 ggRectangle()	63
7.1.3.67 ggRotate() [1/5]	63
7.1.3.68 ggRotate() [2/5]	64
7.1.3.69 ggRotate() [3/5]	65
7.1.3.70 ggRotate() [4/5]	66
7.1.3.71 ggRotate() [5/5]	67
7.1.3.72 ggRotateQuaternion() [1/3]	68
7.1.3.73 ggRotateQuaternion() [2/3]	68
7.1.3.74 ggRotateQuaternion() [3/3]	69
7.1.3.75 ggRotateX()	70
7.1.3.76 ggRotateY()	70
7.1.3.77 ggRotateZ()	71
7.1.3.78 ggSaveColor()	72
7.1.3.79 ggSaveDepth()	72
7.1.3.80 ggSaveTga()	73
7.1.3.81 ggScale() [1/3]	74
7.1.3.82 ggScale() [2/3]	74
7.1.3.83 ggScale() [3/3]	75
7.1.3.84 ggSlerp() [1/4]	76
7.1.3.85 ggSlerp() [2/4]	76
7.1.3.86 ggSlerp() [3/4]	77
7.1.3.87 ggSlerp() [4/4]	78
7.1.3.88 ggTranslate() [1/3]	79
7.1.3.89 ggTranslate() [2/3]	79
7.1.3.90 ggTranslate() [3/3]	80
7.1.3.91 ggTranspose()	81
7.1.3.92 operator*()	81
7.1.3.93 operator+() [1/2]	82
7.1.3.94 operator+() [2/2]	82
7.1.3.95 operator-() [1/2]	83
7.1.3.96 operator-() [2/2]	83
7.1.3.97 operator/()	83

7.1.4 変数詳解	84
7.1.4.1 ggBufferAlignment	84
8 クラス詳解	85
8.1 Config クラス	85
8.1.1 詳解	85
8.1.2 構築子と解体子	85
8.1.2.1 Config() [1/2]	85
8.1.2.2 Config() [2/2]	85
8.1.2.3 ~Config()	86
8.1.3 関数詳解	86
8.1.3.1 getHeight()	86
8.1.3.2 getWidth()	87
8.1.3.3 load()	87
8.1.3.4 save()	88
8.1.4 フレンドと関連関数の詳解	89
8.1.4.1 Menu	89
8.2 GgApp クラス	89
8.2.1 詳解	89
8.2.2 構築子と解体子	90
8.2.2.1 GgApp() [1/3]	90
8.2.2.2 GgApp() [2/3]	91
8.2.2.3 GgApp() [3/3]	91
8.2.2.4 ~GgApp()	91
8.2.3 関数詳解	91
8.2.3.1 main()	91
8.2.3.2 operator=() [1/2]	92
8.2.3.3 operator=() [2/2]	93
8.3 gg::GgBuffer< T > クラステンプレート	93
8.3.1 詳解	94
8.3.2 構築子と解体子	94
8.3.2.1 GgBuffer() [1/3]	94
8.3.2.2 GgBuffer() [2/3]	95
8.3.2.3 GgBuffer() [3/3]	95
8.3.2.4 ~GgBuffer()	95
8.3.3 関数詳解	95
8.3.3.1 bind()	95
8.3.3.2 copy()	96
8.3.3.3 getBuffer()	97
8.3.3.4 getCount()	97
8.3.3.5 getStride()	97
8.3.3.6 getTarget()	98

8.3.3.7 map() [1/2]	98
8.3.3.8 map() [2/2]	98
8.3.3.9 operator=() [1/2]	99
8.3.3.10 operator=() [2/2]	99
8.3.3.11 read()	99
8.3.3.12 send()	100
8.3.3.13 unbind()	100
8.3.3.14 unmap()	100
8.4 gg::GgColorTexture クラス	101
8.4.1 詳解	101
8.4.2 構築子と解体子	101
8.4.2.1 GgColorTexture() [1/3]	101
8.4.2.2 GgColorTexture() [2/3]	101
8.4.2.3 GgColorTexture() [3/3]	102
8.4.2.4 ~GgColorTexture()	103
8.4.3 関数詳解	103
8.4.3.1 load() [1/2]	103
8.4.3.2 load() [2/2]	104
8.5 gg::GgElements クラス	105
8.5.1 詳解	106
8.5.2 構築子と解体子	106
8.5.2.1 GgElements() [1/2]	106
8.5.2.2 GgElements() [2/2]	107
8.5.2.3 ~GgElements()	107
8.5.3 関数詳解	107
8.5.3.1 draw()	107
8.5.3.2 getIndexBuffer()	108
8.5.3.3 getIndexCount()	108
8.5.3.4 load()	108
8.5.3.5 send()	109
8.6 gg::GgMatrix クラス	109
8.6.1 詳解	112
8.6.2 構築子と解体子	112
8.6.2.1 GgMatrix() [1/6]	112
8.6.2.2 GgMatrix() [2/6]	113
8.6.2.3 GgMatrix() [3/6]	113
8.6.2.4 GgMatrix() [4/6]	114
8.6.2.5 GgMatrix() [5/6]	114
8.6.2.6 GgMatrix() [6/6]	114
8.6.2.7 ~GgMatrix()	114
8.6.3 関数詳解	115
8.6.3.1 frustum()	115

8.6.3.2 get() [1/2]	115
8.6.3.3 get() [2/2]	117
8.6.3.4 invert()	117
8.6.3.5 loadFrustum()	118
8.6.3.6 loadIdentity()	118
8.6.3.7 loadInvert() [1/2]	119
8.6.3.8 loadInvert() [2/2]	119
8.6.3.9 loadLookat() [1/3]	120
8.6.3.10 loadLookat() [2/3]	121
8.6.3.11 loadLookat() [3/3]	121
8.6.3.12 loadNormal() [1/2]	122
8.6.3.13 loadNormal() [2/2]	123
8.6.3.14 loadOrthogonal()	124
8.6.3.15 loadPerspective()	125
8.6.3.16 loadRotate() [1/5]	126
8.6.3.17 loadRotate() [2/5]	126
8.6.3.18 loadRotate() [3/5]	127
8.6.3.19 loadRotate() [4/5]	127
8.6.3.20 loadRotate() [5/5]	128
8.6.3.21 loadRotateX()	129
8.6.3.22 loadRotateY()	130
8.6.3.23 loadRotateZ()	130
8.6.3.24 loadScale() [1/3]	131
8.6.3.25 loadScale() [2/3]	132
8.6.3.26 loadScale() [3/3]	133
8.6.3.27 loadTranslate() [1/3]	133
8.6.3.28 loadTranslate() [2/3]	134
8.6.3.29 loadTranslate() [3/3]	135
8.6.3.30 loadTranspose() [1/2]	135
8.6.3.31 loadTranspose() [2/2]	136
8.6.3.32 lookat() [1/3]	137
8.6.3.33 lookat() [2/3]	137
8.6.3.34 lookat() [3/3]	138
8.6.3.35 normal()	139
8.6.3.36 operator*() [1/3]	140
8.6.3.37 operator*() [2/3]	141
8.6.3.38 operator*() [3/3]	141
8.6.3.39 operator*=() [1/2]	142
8.6.3.40 operator*=() [2/2]	143
8.6.3.41 operator+() [1/2]	144
8.6.3.42 operator+() [2/2]	144
8.6.3.43 operator+=() [1/2]	145

8.6.3.44 operator+=() [2/2]	145
8.6.3.45 operator-() [1/2]	146
8.6.3.46 operator-() [2/2]	147
8.6.3.47 operator-=(()) [1/2]	147
8.6.3.48 operator-=(()) [2/2]	148
8.6.3.49 operator/() [1/2]	148
8.6.3.50 operator/() [2/2]	149
8.6.3.51 operator/=(()) [1/2]	150
8.6.3.52 operator/=(()) [2/2]	150
8.6.3.53 operator=() [1/3]	151
8.6.3.54 operator=() [2/3]	151
8.6.3.55 operator=() [3/3]	152
8.6.3.56 orthogonal()	152
8.6.3.57 perspective()	153
8.6.3.58 projection() [1/4]	154
8.6.3.59 projection() [2/4]	154
8.6.3.60 projection() [3/4]	154
8.6.3.61 projection() [4/4]	155
8.6.3.62 rotate() [1/5]	155
8.6.3.63 rotate() [2/5]	155
8.6.3.64 rotate() [3/5]	156
8.6.3.65 rotate() [4/5]	157
8.6.3.66 rotate() [5/5]	157
8.6.3.67 rotateX()	158
8.6.3.68 rotateY()	159
8.6.3.69 rotateZ()	160
8.6.3.70 scale() [1/3]	160
8.6.3.71 scale() [2/3]	161
8.6.3.72 scale() [3/3]	161
8.6.3.73 translate() [1/3]	162
8.6.3.74 translate() [2/3]	163
8.6.3.75 translate() [3/3]	164
8.6.3.76 transpose()	165
8.7 gg::GgNormalTexture クラス	165
8.7.1 詳解	166
8.7.2 構築子と解体子	166
8.7.2.1 GgNormalTexture() [1/3]	166
8.7.2.2 GgNormalTexture() [2/3]	166
8.7.2.3 GgNormalTexture() [3/3]	167
8.7.2.4 ~GgNormalTexture()	167
8.7.3 関数詳解	167
8.7.3.1 load() [1/2]	167

8.7.3.2 load() [2/2]	168
8.8 gg::GgPoints クラス	169
8.8.1 詳解	170
8.8.2 構築子と解体子	170
8.8.2.1 GgPoints() [1/2]	170
8.8.2.2 GgPoints() [2/2]	170
8.8.2.3 ~GgPoints()	171
8.8.3 関数詳解	171
8.8.3.1 draw()	171
8.8.3.2 getBuffer()	172
8.8.3.3 getCount()	172
8.8.3.4 load()	172
8.8.3.5 operator bool()	172
8.8.3.6 operator"!"()	173
8.8.3.7 send()	173
8.9 gg::GgPointShader クラス	173
8.9.1 詳解	174
8.9.2 構築子と解体子	174
8.9.2.1 GgPointShader() [1/3]	174
8.9.2.2 GgPointShader() [2/3]	175
8.9.2.3 GgPointShader() [3/3]	175
8.9.2.4 ~GgPointShader()	175
8.9.3 関数詳解	176
8.9.3.1 get()	176
8.9.3.2 load() [1/2]	176
8.9.3.3 load() [2/2]	176
8.9.3.4 loadMatrix() [1/2]	177
8.9.3.5 loadMatrix() [2/2]	178
8.9.3.6 loadModelviewMatrix() [1/2]	178
8.9.3.7 loadModelviewMatrix() [2/2]	179
8.9.3.8 loadProjectionMatrix() [1/2]	179
8.9.3.9 loadProjectionMatrix() [2/2]	180
8.9.3.10 unuse()	180
8.9.3.11 use() [1/5]	180
8.9.3.12 use() [2/5]	180
8.9.3.13 use() [3/5]	181
8.9.3.14 use() [4/5]	181
8.9.3.15 use() [5/5]	182
8.10 gg::GgQuaternion クラス	182
8.10.1 詳解	186
8.10.2 構築子と解体子	186
8.10.2.1 GgQuaternion() [1/7]	186

8.10.2.2 GgQuaternion() [2/7]	187
8.10.2.3 GgQuaternion() [3/7]	187
8.10.2.4 GgQuaternion() [4/7]	187
8.10.2.5 GgQuaternion() [5/7]	188
8.10.2.6 GgQuaternion() [6/7]	188
8.10.2.7 GgQuaternion() [7/7]	188
8.10.2.8 ~GgQuaternion()	189
8.10.3 関数詳解	189
8.10.3.1 add() [1/4]	189
8.10.3.2 add() [2/4]	189
8.10.3.3 add() [3/4]	190
8.10.3.4 add() [4/4]	190
8.10.3.5 conjugate()	191
8.10.3.6 divide() [1/4]	192
8.10.3.7 divide() [2/4]	193
8.10.3.8 divide() [3/4]	193
8.10.3.9 divide() [4/4]	194
8.10.3.10 euler() [1/3]	195
8.10.3.11 euler() [2/3]	196
8.10.3.12 euler() [3/3]	196
8.10.3.13 get()	197
8.10.3.14 getConjugateMatrix() [1/3]	198
8.10.3.15 getConjugateMatrix() [2/3]	198
8.10.3.16 getConjugateMatrix() [3/3]	199
8.10.3.17 getMatrix() [1/3]	199
8.10.3.18 getMatrix() [2/3]	200
8.10.3.19 getMatrix() [3/3]	201
8.10.3.20 invert()	201
8.10.3.21 loadAdd() [1/4]	202
8.10.3.22 loadAdd() [2/4]	203
8.10.3.23 loadAdd() [3/4]	203
8.10.3.24 loadAdd() [4/4]	204
8.10.3.25 loadConjugate() [1/2]	205
8.10.3.26 loadConjugate() [2/2]	206
8.10.3.27 loadDivide() [1/4]	207
8.10.3.28 loadDivide() [2/4]	208
8.10.3.29 loadDivide() [3/4]	209
8.10.3.30 loadDivide() [4/4]	210
8.10.3.31 loadEuler() [1/2]	211
8.10.3.32 loadEuler() [2/2]	212
8.10.3.33 loadIdentity()	213
8.10.3.34 loadInvert() [1/2]	213

8.10.3.35 loadInvert() [2/2]	214
8.10.3.36 loadMatrix() [1/2]	215
8.10.3.37 loadMatrix() [2/2]	215
8.10.3.38 loadMultiply() [1/4]	216
8.10.3.39 loadMultiply() [2/4]	217
8.10.3.40 loadMultiply() [3/4]	217
8.10.3.41 loadMultiply() [4/4]	217
8.10.3.42 loadNormalize() [1/2]	218
8.10.3.43 loadNormalize() [2/2]	219
8.10.3.44 loadRotate() [1/3]	220
8.10.3.45 loadRotate() [2/3]	220
8.10.3.46 loadRotate() [3/3]	221
8.10.3.47 loadRotateX()	222
8.10.3.48 loadRotateY()	223
8.10.3.49 loadRotateZ()	223
8.10.3.50 loadSlerp() [1/4]	224
8.10.3.51 loadSlerp() [2/4]	224
8.10.3.52 loadSlerp() [3/4]	225
8.10.3.53 loadSlerp() [4/4]	226
8.10.3.54 loadSubtract() [1/4]	227
8.10.3.55 loadSubtract() [2/4]	227
8.10.3.56 loadSubtract() [3/4]	228
8.10.3.57 loadSubtract() [4/4]	228
8.10.3.58 multiply() [1/4]	229
8.10.3.59 multiply() [2/4]	230
8.10.3.60 multiply() [3/4]	230
8.10.3.61 multiply() [4/4]	230
8.10.3.62 norm()	231
8.10.3.63 normalize()	231
8.10.3.64 operator*() [1/3]	232
8.10.3.65 operator*() [2/3]	232
8.10.3.66 operator*() [3/3]	233
8.10.3.67 operator*=() [1/3]	233
8.10.3.68 operator*=() [2/3]	233
8.10.3.69 operator*=() [3/3]	234
8.10.3.70 operator+() [1/3]	234
8.10.3.71 operator+() [2/3]	235
8.10.3.72 operator+() [3/3]	235
8.10.3.73 operator+=() [1/3]	236
8.10.3.74 operator+=() [2/3]	236
8.10.3.75 operator+=() [3/3]	236
8.10.3.76 operator-() [1/3]	237

8.10.3.77 operator-() [2/3]	237
8.10.3.78 operator-() [3/3]	237
8.10.3.79 operator-=() [1/3]	238
8.10.3.80 operator-=() [2/3]	238
8.10.3.81 operator-=() [3/3]	238
8.10.3.82 operator/() [1/3]	239
8.10.3.83 operator/() [2/3]	239
8.10.3.84 operator/() [3/3]	240
8.10.3.85 operator/() [1/3]	240
8.10.3.86 operator/() [2/3]	241
8.10.3.87 operator/() [3/3]	241
8.10.3.88 operator=() [1/4]	241
8.10.3.89 operator=() [2/4]	242
8.10.3.90 operator=() [3/4]	242
8.10.3.91 operator=() [4/4]	243
8.10.3.92 rotate() [1/3]	243
8.10.3.93 rotate() [2/3]	243
8.10.3.94 rotate() [3/3]	244
8.10.3.95 rotateX()	245
8.10.3.96 rotateY()	246
8.10.3.97 rotateZ()	246
8.10.3.98 slerp() [1/2]	247
8.10.3.99 slerp() [2/2]	247
8.10.3.100 subtract() [1/4]	248
8.10.3.101 subtract() [2/4]	248
8.10.3.102 subtract() [3/4]	249
8.10.3.103 subtract() [4/4]	249
8.11 gg::GgShader クラス	250
8.11.1 詳解	251
8.11.2 構築子と解体子	251
8.11.2.1 GgShader() [1/4]	251
8.11.2.2 GgShader() [2/4]	251
8.11.2.3 GgShader() [3/4]	253
8.11.2.4 GgShader() [4/4]	253
8.11.2.5 ~GgShader()	253
8.11.3 関数詳解	253
8.11.3.1 get()	253
8.11.3.2 operator=() [1/2]	254
8.11.3.3 operator=() [2/2]	254
8.11.3.4 unuse()	254
8.11.3.5 use()	255
8.12 gg::GgShape クラス	255

8.12.1 詳解	256
8.12.2 構築子と解体子	256
8.12.2.1 GgShape()	256
8.12.2.2 ~GgShape()	256
8.12.3 関数詳解	256
8.12.3.1 draw()	256
8.12.3.2 get()	257
8.12.3.3 getMode()	257
8.12.3.4 operator bool()	258
8.12.3.5 operator"!"()	258
8.12.3.6 setMode()	258
8.13 gg::GgSimpleObj クラス	258
8.13.1 詳解	259
8.13.2 構築子と解体子	259
8.13.2.1 GgSimpleObj()	259
8.13.2.2 ~GgSimpleObj()	259
8.13.3 関数詳解	260
8.13.3.1 draw()	260
8.13.3.2 get()	260
8.13.3.3 operator bool()	260
8.13.3.4 operator"!"()	261
8.14 gg::GgSimpleShader クラス	261
8.14.1 詳解	263
8.14.2 構築子と解体子	263
8.14.2.1 GgSimpleShader() [1/4]	263
8.14.2.2 GgSimpleShader() [2/4]	263
8.14.2.3 GgSimpleShader() [3/4]	264
8.14.2.4 GgSimpleShader() [4/4]	264
8.14.2.5 ~GgSimpleShader()	264
8.14.3 関数詳解	264
8.14.3.1 load() [1/2]	264
8.14.3.2 load() [2/2]	265
8.14.3.3 loadMatrix() [1/4]	265
8.14.3.4 loadMatrix() [2/4]	266
8.14.3.5 loadMatrix() [3/4]	267
8.14.3.6 loadMatrix() [4/4]	267
8.14.3.7 loadModelviewMatrix() [1/4]	267
8.14.3.8 loadModelviewMatrix() [2/4]	268
8.14.3.9 loadModelviewMatrix() [3/4]	268
8.14.3.10 loadModelviewMatrix() [4/4]	269
8.14.3.11 operator=()	269
8.14.3.12 use() [1/13]	269

8.14.3.13 use() [2/13]	270
8.14.3.14 use() [3/13]	270
8.14.3.15 use() [4/13]	271
8.14.3.16 use() [5/13]	271
8.14.3.17 use() [6/13]	272
8.14.3.18 use() [7/13]	273
8.14.3.19 use() [8/13]	273
8.14.3.20 use() [9/13]	273
8.14.3.21 use() [10/13]	274
8.14.3.22 use() [11/13]	274
8.14.3.23 use() [12/13]	275
8.14.3.24 use() [13/13]	275
8.15 gg::GgTexture クラス	276
8.15.1 詳解	276
8.15.2 構築子と解体子	277
8.15.2.1 GgTexture() [1/3]	277
8.15.2.2 GgTexture() [2/3]	277
8.15.2.3 GgTexture() [3/3]	277
8.15.2.4 ~GgTexture()	278
8.15.3 関数詳解	278
8.15.3.1 bind()	278
8.15.3.2 getHeight()	278
8.15.3.3 getSize() [1/2]	279
8.15.3.4 getSize() [2/2]	279
8.15.3.5 getTexture()	279
8.15.3.6 getWidth()	280
8.15.3.7 operator=() [1/2]	280
8.15.3.8 operator=() [2/2]	280
8.15.3.9 swapRandB()	281
8.15.3.10 unbind()	281
8.16 gg::GgTrackball クラス	281
8.16.1 詳解	286
8.16.2 構築子と解体子	286
8.16.2.1 GgTrackball()	286
8.16.2.2 ~GgTrackball()	286
8.16.3 関数詳解	287
8.16.3.1 begin()	287
8.16.3.2 end()	288
8.16.3.3 get()	288
8.16.3.4 getMatrix()	289
8.16.3.5 getQuaternion()	289
8.16.3.6 getScale() [1/3]	289

8.16.3.7 <code>getScale()</code> [2/3]	289
8.16.3.8 <code>getScale()</code> [3/3]	290
8.16.3.9 <code>getStart()</code> [1/3]	290
8.16.3.10 <code>getStart()</code> [2/3]	290
8.16.3.11 <code>getStart()</code> [3/3]	290
8.16.3.12 <code>motion()</code>	291
8.16.3.13 <code>operator=()</code>	291
8.16.3.14 <code>region()</code> [1/2]	292
8.16.3.15 <code>region()</code> [2/2]	292
8.16.3.16 <code>reset()</code>	293
8.16.3.17 <code>rotate()</code>	294
8.17 <code>gg::GgTriangles</code> クラス	294
8.17.1 詳解	296
8.17.2 構築子と解体子	296
8.17.2.1 <code>GgTriangles()</code> [1/2]	296
8.17.2.2 <code>GgTriangles()</code> [2/2]	296
8.17.2.3 <code>~GgTriangles()</code>	297
8.17.3 関数詳解	297
8.17.3.1 <code>draw()</code>	297
8.17.3.2 <code>getBuffer()</code>	297
8.17.3.3 <code>getCount()</code>	298
8.17.3.4 <code>load()</code>	298
8.17.3.5 <code>send()</code>	298
8.18 <code>gg::GgUniformBuffer< T ></code> クラステンプレート	299
8.18.1 詳解	299
8.18.2 構築子と解体子	299
8.18.2.1 <code>GgUniformBuffer()</code> [1/3]	299
8.18.2.2 <code>GgUniformBuffer()</code> [2/3]	300
8.18.2.3 <code>GgUniformBuffer()</code> [3/3]	300
8.18.2.4 <code>~GgUniformBuffer()</code>	300
8.18.3 関数詳解	301
8.18.3.1 <code>bind()</code>	301
8.18.3.2 <code>copy()</code>	301
8.18.3.3 <code>fill()</code>	301
8.18.3.4 <code>getBuffer()</code>	302
8.18.3.5 <code>getCount()</code>	302
8.18.3.6 <code>getStride()</code>	302
8.18.3.7 <code>getTarget()</code>	302
8.18.3.8 <code>load()</code> [1/2]	303
8.18.3.9 <code>load()</code> [2/2]	303
8.18.3.10 <code>map()</code> [1/2]	303
8.18.3.11 <code>map()</code> [2/2]	304

8.18.3.12 <code>read()</code>	304
8.18.3.13 <code>send()</code>	304
8.18.3.14 <code>unbind()</code>	305
8.18.3.15 <code>unmap()</code>	305
8.19 <code>gg::GgVector</code> クラス	306
8.19.1 詳解	307
8.19.2 構築子と解体子	307
8.19.2.1 <code>GgVector()</code> [1/6]	307
8.19.2.2 <code>GgVector()</code> [2/6]	308
8.19.2.3 <code>GgVector()</code> [3/6]	308
8.19.2.4 <code>GgVector()</code> [4/6]	309
8.19.2.5 <code>GgVector()</code> [5/6]	309
8.19.2.6 <code>GgVector()</code> [6/6]	309
8.19.2.7 <code>~GgVector()</code>	309
8.19.3 関数詳解	310
8.19.3.1 <code>distance3()</code>	310
8.19.3.2 <code>distance4()</code>	310
8.19.3.3 <code>dot3()</code>	311
8.19.3.4 <code>dot4()</code>	311
8.19.3.5 <code>length3()</code>	312
8.19.3.6 <code>length4()</code>	312
8.19.3.7 <code>normalize3()</code>	313
8.19.3.8 <code>normalize4()</code>	313
8.19.3.9 <code>operator*()</code> [1/2]	314
8.19.3.10 <code>operator*()</code> [2/2]	314
8.19.3.11 <code>operator*=(())</code> [1/2]	315
8.19.3.12 <code>operator*=(())</code> [2/2]	316
8.19.3.13 <code>operator+()</code> [1/2]	316
8.19.3.14 <code>operator+()</code> [2/2]	317
8.19.3.15 <code>operator+=()</code> [1/2]	317
8.19.3.16 <code>operator+=()</code> [2/2]	317
8.19.3.17 <code>operator-()</code> [1/2]	318
8.19.3.18 <code>operator-()</code> [2/2]	318
8.19.3.19 <code>operator-=(())</code> [1/2]	319
8.19.3.20 <code>operator-=(())</code> [2/2]	319
8.19.3.21 <code>operator/()</code> [1/2]	320
8.19.3.22 <code>operator/()</code> [2/2]	320
8.19.3.23 <code>operator/=(())</code> [1/2]	321
8.19.3.24 <code>operator/=(())</code> [2/2]	322
8.19.3.25 <code>operator=(())</code> [1/2]	322
8.19.3.26 <code>operator=(())</code> [2/2]	322
8.20 <code>gg::GgVertex</code> 構造体	323

8.20.1 詳解	323
8.20.2 構築子と解体子	324
8.20.2.1 GgVertex() [1/4]	324
8.20.2.2 GgVertex() [2/4]	324
8.20.2.3 GgVertex() [3/4]	324
8.20.2.4 GgVertex() [4/4]	325
8.20.3 メンバ詳解	325
8.20.3.1 normal	325
8.20.3.2 position	325
8.21 gg::GgVertexArray クラス	325
8.21.1 詳解	326
8.21.2 構築子と解体子	326
8.21.2.1 GgVertexArray() [1/3]	326
8.21.2.2 GgVertexArray() [2/3]	326
8.21.2.3 GgVertexArray() [3/3]	326
8.21.2.4 ~GgVertexArray()	327
8.21.3 関数詳解	327
8.21.3.1 bind()	327
8.21.3.2 get()	327
8.21.3.3 operator=() [1/2]	327
8.21.3.4 operator=() [2/2]	328
8.22 gg::GgSimpleShader::Light 構造体	328
8.22.1 詳解	329
8.22.2 メンバ詳解	329
8.22.2.1 ambient	329
8.22.2.2 diffuse	329
8.22.2.3 position	329
8.22.2.4 specular	330
8.23 gg::GgSimpleShader::LightBuffer クラス	330
8.23.1 詳解	331
8.23.2 構築子と解体子	332
8.23.2.1 LightBuffer() [1/3]	332
8.23.2.2 LightBuffer() [2/3]	332
8.23.2.3 LightBuffer() [3/3]	332
8.23.2.4 ~LightBuffer()	333
8.23.3 関数詳解	333
8.23.3.1 load() [1/2]	333
8.23.3.2 load() [2/2]	333
8.23.3.3 loadAmbient() [1/3]	334
8.23.3.4 loadAmbient() [2/3]	334
8.23.3.5 loadAmbient() [3/3]	334
8.23.3.6 loadColor()	335

8.23.3.7 loadDiffuse() [1/3]	335
8.23.3.8 loadDiffuse() [2/3]	336
8.23.3.9 loadDiffuse() [3/3]	336
8.23.3.10 loadPosition() [1/4]	336
8.23.3.11 loadPosition() [2/4]	337
8.23.3.12 loadPosition() [3/4]	337
8.23.3.13 loadPosition() [4/4]	338
8.23.3.14 loadSpecular() [1/3]	338
8.23.3.15 loadSpecular() [2/3]	338
8.23.3.16 loadSpecular() [3/3]	339
8.23.3.17 select()	339
8.24 gg::GgSimpleShader::Material 構造体	340
8.24.1 詳解	341
8.24.2 メンバ詳解	341
8.24.2.1 ambient	341
8.24.2.2 diffuse	341
8.24.2.3 shininess	341
8.24.2.4 specular	342
8.25 gg::GgSimpleShader::MaterialBuffer クラス	342
8.25.1 詳解	343
8.25.2 構築子と解体子	344
8.25.2.1 MaterialBuffer() [1/3]	344
8.25.2.2 MaterialBuffer() [2/3]	344
8.25.2.3 MaterialBuffer() [3/3]	344
8.25.2.4 ~MaterialBuffer()	345
8.25.3 関数詳解	345
8.25.3.1 load() [1/2]	345
8.25.3.2 load() [2/2]	345
8.25.3.3 loadAmbient() [1/3]	346
8.25.3.4 loadAmbient() [2/3]	346
8.25.3.5 loadAmbient() [3/3]	346
8.25.3.6 loadAmbientAndDiffuse() [1/3]	347
8.25.3.7 loadAmbientAndDiffuse() [2/3]	347
8.25.3.8 loadAmbientAndDiffuse() [3/3]	348
8.25.3.9 loadDiffuse() [1/3]	348
8.25.3.10 loadDiffuse() [2/3]	348
8.25.3.11 loadDiffuse() [3/3]	349
8.25.3.12 loadShininess() [1/2]	349
8.25.3.13 loadShininess() [2/2]	350
8.25.3.14 loadSpecular() [1/3]	350
8.25.3.15 loadSpecular() [2/3]	350
8.25.3.16 loadSpecular() [3/3]	351

8.25.3.17 select()	351
8.26 Menu クラス	351
8.26.1 詳解	352
8.26.2 構築子と解体子	352
8.26.2.1 Menu() [1/3]	352
8.26.2.2 Menu() [2/3]	352
8.26.2.3 Menu() [3/3]	352
8.26.2.4 ~Menu()	353
8.26.3 関数詳解	353
8.26.3.1 draw()	353
8.26.3.2 getLight()	353
8.26.3.3 getModel()	353
8.26.3.4 getShader()	353
8.26.3.5 operator=() [1/2]	353
8.26.3.6 operator=() [2/2]	354
8.26.4 フレンドと関連関数の詳解	354
8.26.4.1 Scene	354
8.27 GgApp::Window クラス	354
8.27.1 詳解	355
8.27.2 構築子と解体子	356
8.27.2.1 Window() [1/3]	356
8.27.2.2 Window() [2/3]	356
8.27.2.3 Window() [3/3]	357
8.27.2.4 ~Window()	357
8.27.3 関数詳解	357
8.27.3.1 get()	357
8.27.3.2 getAltArrowX()	357
8.27.3.3 getAltArrowY()	358
8.27.3.4 getAltArrow()	359
8.27.3.5 getArrow() [1/2]	359
8.27.3.6 getArrow() [2/2]	360
8.27.3.7 getArrowX()	361
8.27.3.8 getArrowY()	361
8.27.3.9 getAspect()	362
8.27.3.10 getControlArrow()	362
8.27.3.11 getControlArrowX()	363
8.27.3.12 getControlArrowY()	364
8.27.3.13 getFboHeight()	364
8.27.3.14 getFboSize() [1/2]	365
8.27.3.15 getFboSize() [2/2]	365
8.27.3.16 getFboWidth()	365
8.27.3.17 getHeight()	366

8.27.3.18 getKey()	366
8.27.3.19 getLastKey()	367
8.27.3.20 getMouse() [1/3]	367
8.27.3.21 getMouse() [2/3]	367
8.27.3.22 getMouse() [3/3]	367
8.27.3.23 getMouseX()	368
8.27.3.24 getMouseY()	368
8.27.3.25 getRotation()	368
8.27.3.26 getRotationMatrix()	369
8.27.3.27 getScrollMatrix()	369
8.27.3.28 getShiftArrow()	369
8.27.3.29 getShiftArrowX()	370
8.27.3.30 getShiftArrowY()	371
8.27.3.31 getSize() [1/2]	371
8.27.3.32 getSize() [2/2]	371
8.27.3.33 getTranslation()	372
8.27.3.34 getTranslationMatrix()	372
8.27.3.35 getUserPointer()	373
8.27.3.36 getWheel() [1/3]	373
8.27.3.37 getWheel() [2/3]	373
8.27.3.38 getWheel() [3/3]	373
8.27.3.39 getWheelX()	374
8.27.3.40 getWheelY()	374
8.27.3.41 getWidth()	374
8.27.3.42 operator bool()	375
8.27.3.43 operator=() [1/2]	375
8.27.3.44 operator=() [2/2]	375
8.27.3.45 reset()	375
8.27.3.46 resetRotation()	376
8.27.3.47 resetTranslation()	376
8.27.3.48 restoreViewport()	377
8.27.3.49 selectInterface()	377
8.27.3.50 setClose()	377
8.27.3.51 setKeyboardFunc()	377
8.27.3.52 setMouseFunc()	378
8.27.3.53 setResizeFunc()	378
8.27.3.54 setUserPointer()	378
8.27.3.55 setVelocity()	378
8.27.3.56 setWheelFunc()	379
8.27.3.57 shouldClose()	379
8.27.3.58 swapBuffers()	379
8.27.3.59 updateViewport()	379

9 ファイル詳解	381
9.1 Config.cpp ファイル	381
9.1.1 詳解	382
9.1.2 変数詳解	382
9.1.2.1 defaultLight	382
9.2 Config.cpp	382
9.3 Config.h ファイル	384
9.3.1 詳解	385
9.4 Config.h	385
9.5 gg.cpp ファイル	386
9.5.1 詳解	386
9.6 gg.cpp	387
9.7 gg.h ファイル	463
9.7.1 詳解	467
9.7.2 マクロ定義詳解	468
9.7.2.1 ggError	468
9.7.2.2 ggFBOError	468
9.7.3 型定義詳解	468
9.7.3.1 pathChar	468
9.7.3.2 pathString	468
9.7.4 関数詳解	468
9.7.4.1 TCHARToUtf8()	468
9.7.4.2 Utf8ToTCHAR()	469
9.8 gg.h	469
9.9 GgApp.cpp ファイル	524
9.9.1 詳解	524
9.10 GgApp.cpp	525
9.11 GgApp.h ファイル	538
9.11.1 詳解	539
9.11.2 マクロ定義詳解	539
9.11.2.1 GG_BUTTON_COUNT	539
9.11.2.2 GG_INTERFACE_COUNT	540
9.11.2.3 GG_USE_IMGUI	540
9.12 GgApp.h	540
9.13 ggsample01.cpp ファイル	548
9.13.1 マクロ定義詳解	548
9.13.1.1 CONFIG_FILE	548
9.13.1.2 PROJECT_NAME	548
9.14 ggsample01.cpp	549
9.15 main.cpp ファイル	550
9.15.1 詳解	550
9.15.2 マクロ定義詳解	550

9.15.2.1 HEADER_STR	550
9.15.3 関数詳解	551
9.15.3.1 main()	551
9.16 main.cpp	551
9.17 Menu.cpp ファイル	552
9.17.1 詳解	552
9.18 Menu.cpp	553
9.19 Menu.h ファイル	554
9.19.1 詳解	555
9.20 Menu.h	555
9.21 parseconfig.h ファイル	556
9.21.1 詳解	557
9.21.2 関数詳解	557
9.21.2.1 getString() [1/3]	557
9.21.2.2 getString() [2/3]	558
9.21.2.3 getString() [3/3]	558
9.21.2.4 getValue() [1/2]	559
9.21.2.5 getValue() [2/2]	559
9.21.2.6 getVector()	560
9.21.2.7 setString() [1/3]	561
9.21.2.8 setString() [2/3]	561
9.21.2.9 setString() [3/3]	562
9.21.2.10 setValue() [1/2]	562
9.21.2.11 setValue() [2/2]	562
9.21.2.12 setVector()	563
9.22 parseconfig.h	564
9.23 README.md ファイル	566
Index	567

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	109
gg::GgMatrix	109
gg::GgVector	306
gg::GgQuaternion	182
gg::GgTrackball	281
Config	85
GgApp	89
gg::GgBuffer< T >	93
gg::GgColorTexture	101
gg::GgNormalTexture	165
gg::GgPointShader	173
gg::GgSimpleShader	261
gg::GgShader	250
gg::GgShape	255
gg::GgPoints	169
gg::GgTriangles	294
gg::GgElements	105
gg::GgSimpleObj	258
gg::GgTexture	276
gg::GgUniformBuffer< T >	299
gg::GgUniformBuffer< Light >	299
gg::GgSimpleShader::LightBuffer	330
gg::GgUniformBuffer< Material >	299
gg::GgSimpleShader::MaterialBuffer	342
gg::GgVertex	323
gg::GgVertexArray	325
gg::GgSimpleShader::Light	328
gg::GgSimpleShader::Material	340
Menu	351
GgApp::Window	354

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	85
GgApp	89
gg::GgBuffer< T >	93
gg::GgColorTexture	101
gg::GgElements	105
gg::GgMatrix	109
gg::GgNormalTexture	165
gg::GgPoints	169
gg::GgPointShader	173
gg::GgQuaternion	182
gg::GgShader	250
gg::GgShape	255
gg::GgSimpleObj	258
gg::GgSimpleShader	261
gg::GgTexture	276
gg::GgTrackball	281
gg::GgTriangles	294
gg::GgUniformBuffer< T >	299
gg::GgVector	306
gg::GgVertex	323
gg::GgVertexArray	325
gg::GgSimpleShader::Light	328
gg::GgSimpleShader::LightBuffer	330
gg::GgSimpleShader::Material	340
gg::GgSimpleShader::MaterialBuffer	342
Menu	351
GgApp::Window	354

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	381
Config.h	384
gg.cpp	386
gg.h	463
GgApp.cpp	524
GgApp.h	538
ggsample01.cpp	548
main.cpp	550
Menu.cpp	552
Menu.h	554
parseconfig.h	556

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 ggEulerQuaternion (const GgVector &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=false)`
- `GLuint ggLoadImage (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA, 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 &crt="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 5275 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

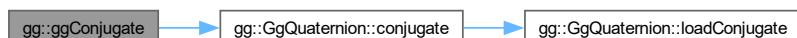
<i>q</i>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

四元数 *q* の共役四元数.

gg.h の 4756 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 5048 行目に定義があります。

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

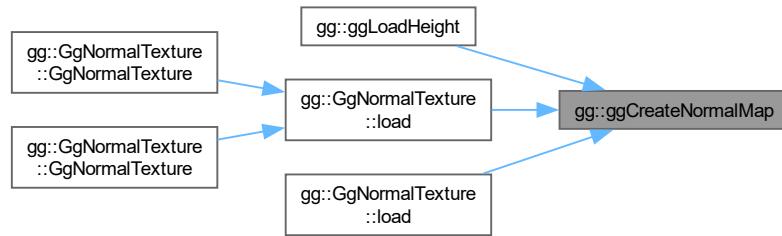
<i>hmap</i>	グレースケール画像のデータ.
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数.
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数.
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_RGBA).
<i>nz</i>	法線の z 成分の割合.
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット.
<i>nmap</i>	法線マップを格納する vector.

gg.cpp の 3875 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 <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).
<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 4872 行目に定義があります。

呼び出し関係図:



7.1.3.8 ggCross() [1/2]

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

`GgVector` 型の 3 要素の外積.

引数

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

戻り値

`a` と `b` の外積.

覚え書き

戻り値の `w` (第4) 要素は 0.

`gg.h` の 1994 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 2031 行目に定義がります。

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

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

戻り値

2 つの GgVector *a*, *b* の距離.

gg.h の 2095 行目に定義があります。

呼び出し関係図:



7.1.3.13 ggDistance4() [2/2]

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

GLfloat 型の 4 要素の距離.

引数

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

戻り値

a と *b* のそれぞれの 4 要素の距離.

gg.h の 1531 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.14 ggDot3() [1/2]

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

GgVector 型の 3 要素の内積.

引数

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

戻り値

a と *b* のそれぞれの 3 要素の内積.

gg.h の 2008 行目に定義がります。

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

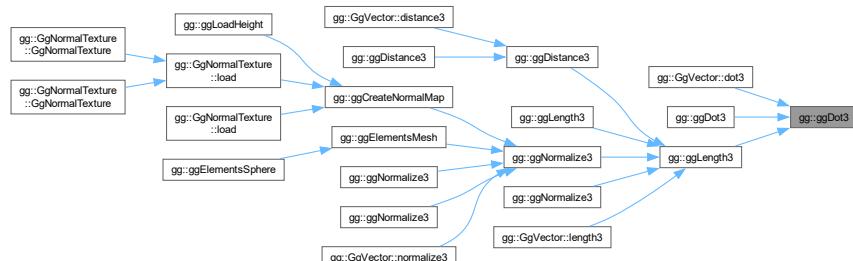
<i>a</i>	GLfloat 型の 3 要素の配列変数.
<i>b</i>	GLfloat 型の 3 要素の配列変数.

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。](#)

被呼び出し関係図:



7.1.3.16 ggDot4() [1/2]

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

GgVector 型の 4 要素の内積.

引数

a	GgVector 型の変数.
b	GgVector 型の変数.

戻り値

a と b のそれぞれの 4 要素の内積.

gg.h の 2072 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

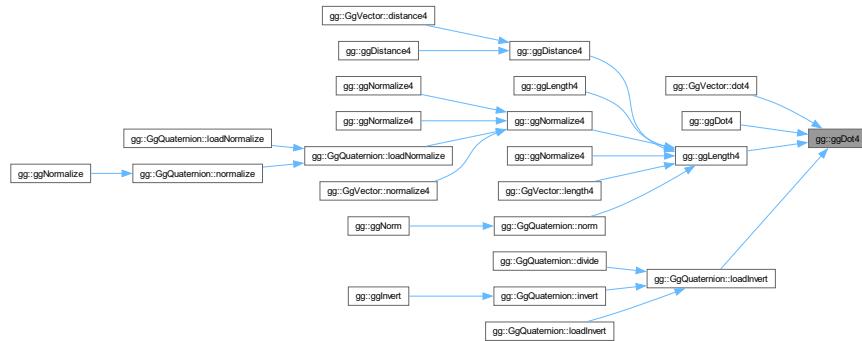
a	GLfloat 型の 4 要素の配列変数.
b	GLfloat 型の 4 要素の配列変数.

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, <code>nullptr</code> なら頂点の位置から算出する.

戻り値

`GgElements` 型のポインタ.

覚え書き

メッシュ状に `GgElements` 形式の三角形データを生成する.

`gg.cpp` の 5309 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.19 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

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

戻り値

GgElements 型のポインタ.

覚え書き

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

gg.cpp の 5291 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 5404 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]
```

橭円状に三角形を生成する.

引数

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

戻り値

`GgTriangles` 型のポインタ.

`gg.cpp` の 5250 行目に定義があります。

7.1.3.22 `ggEulerQuaternion()` [1/3]

```
GgQuaternion gg::ggEulerQuaternion (
    const GgVector & 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

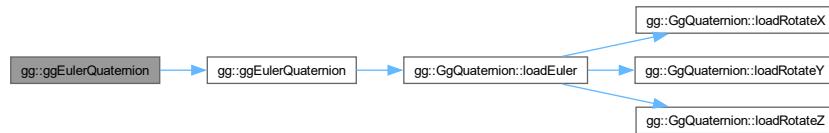
<code>e</code>	オイラー角を表す <code>GgVector</code> 型の変数 (heading, pitch, roll) の参照.
----------------	---

戻り値

回転を表す四元数.

`gg.h` の 4670 行目に定義があります。

呼び出し関係図:



7.1.3.23 `ggEulerQuaternion()` [2/3]

```
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

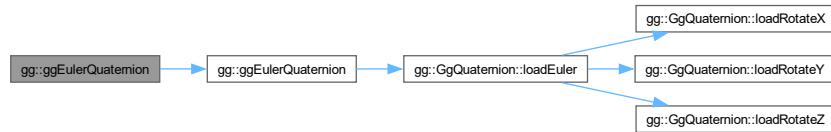
<code>e</code>	オイラー角を表す <code>GLfloat</code> 型の 3 要素の配列変数 (heading, pitch, roll).
----------------	--

戻り値

回転を表す四元数.

[gg.h](#) の 4659 行目に定義があります。

呼び出し関係図:



7.1.3.24 ggEulerQuaternion() [3/3]

```
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

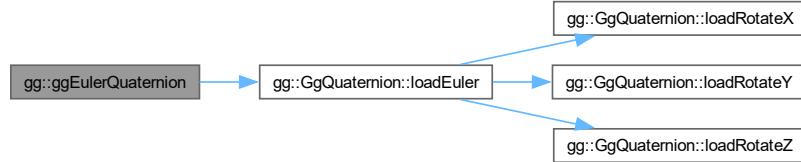
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

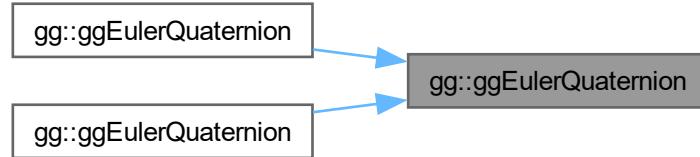
回転を表す四元数.

[gg.h](#) の 4647 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.25 `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 3388 行目に定義があります。

呼び出し関係図:



7.1.3.26 ggIdentity()

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

単位行列を返す。

戻り値

単位行列。

gg.h の 3113 行目に定義があります。

呼び出し関係図:



7.1.3.27 ggIdentityQuaternion()

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

単位四元数を返す。

戻り値

単位四元数。

gg.h の 4545 行目に定義があります。

呼び出し関係図:



7.1.3.28 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。

呼び出し関係図:



7.1.3.29 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 3433 行目に定義があります。

呼び出し関係図:



7.1.3.30 ggInvert() [2/2]

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

四元数の逆元を求める。

引数

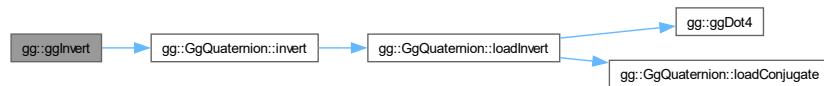
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

四元数 *q* の逆元。

gg.h の 4767 行目に定義があります。

呼び出し関係図:



7.1.3.31 ggLength3() [1/2]

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

GgVector 型の 3 要素の長さ。

引数

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

戻り値

a の 3 要素の長さ。

gg.h の 2019 行目に定義があります。

呼び出し関係図:



7.1.3.32 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

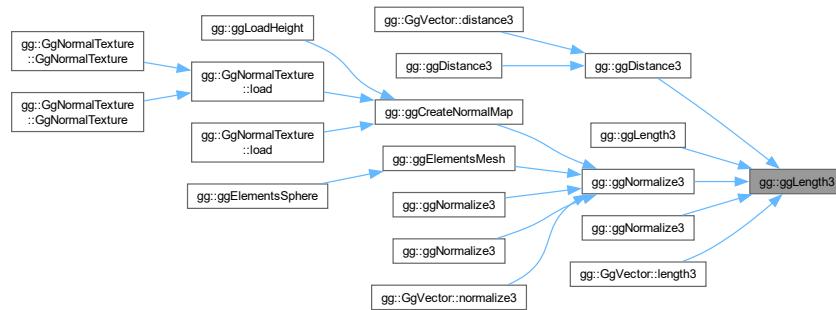
a の 3 要素の長さ.

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:



7.1.3.33 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 2083 行目に定義があります。

呼び出し関係図:



7.1.3.34 ggLength4() [2/2]

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

GLfloat 型の 4 要素の長さ.

引数

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

戻り値

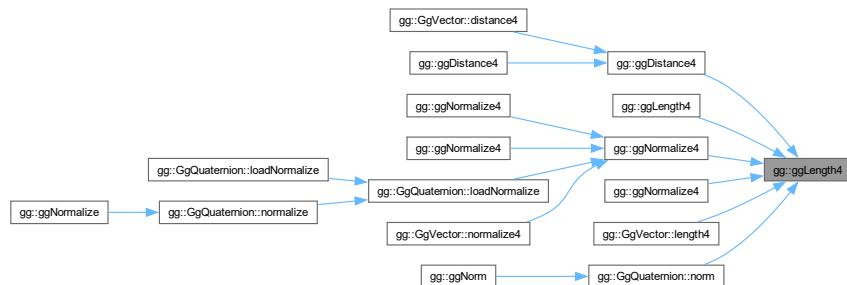
a の 4 要素の長さ.

gg.h の 1519 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.35 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

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

[gg.cpp](#) の 5089 行目に定義があります。

呼び出し関係図:



7.1.3.36 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

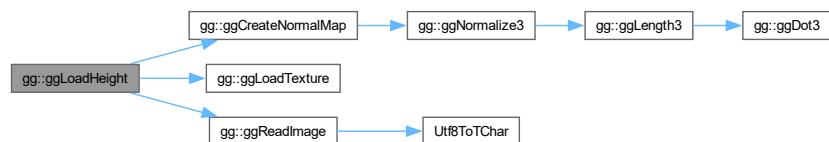
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

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 3960 行目に定義があります。

呼び出し関係図:



7.1.3.37 ggLoadImage()

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

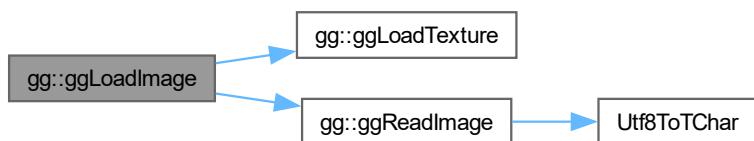
<i>name</i>	読み込むファイル名。
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>internal</i>	テクスチャの内部フォーマット, デフォルトは GL_RGBA. 0 なら外部フォーマットに合わせる。
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

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 3826 行目に定義があります。

呼び出し関係図:



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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

[gg.h](#) の 5156 行目に定義があります。

呼び出し関係図:



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

戻り値

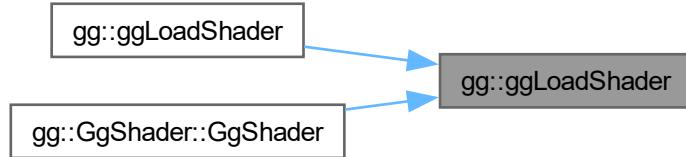
プログラムオブジェクトのプログラム名(作成できなければ 0).

[gg.cpp](#) の 5015 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:



7.1.3.40 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

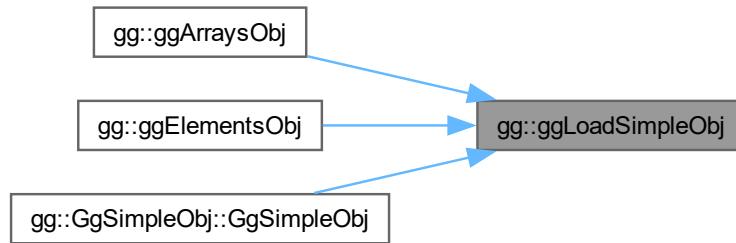
<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの GgSimpleShader::Material 型の材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>normalize</i>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

[gg.cpp](#) の 4624 行目に定義があります。

被呼び出し関係図:



7.1.3.41 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 4713 行目に定義があります。

7.1.3.42 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 = false ) [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

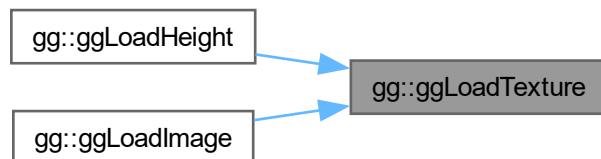
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャの作成のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <i>image</i> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <i>image</i> の縦の画素数.
<i>format</i>	<i>image</i> のフォーマット.
<i>type</i>	<i>image</i> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは <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>false</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 3775 行目に定義があります。

被呼び出し関係図:



7.1.3.43 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 3348 行目に定義があります。

呼び出し関係図:



7.1.3.44 [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 3330 行目に定義があります。

呼び出し関係図:



7.1.3.45 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

ex	視点の位置の x 座標値.
ey	視点の位置の y 座標値.
ez	視点の位置の z 座標値.
tx	目標点の位置の x 座標値.
ty	目標点の位置の y 座標値.
tz	目標点の位置の z 座標値.
ux	上方向のベクトルの x 成分.
uy	上方向のベクトルの y 成分.
uz	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 3312 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:



7.1.3.46 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

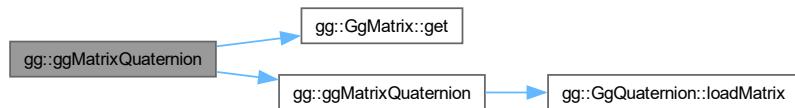
m	GgMatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4569 行目に定義があります。

呼び出し関係図:



7.1.3.47 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 4557 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:



7.1.3.48 ggNorm()

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

四元数のノルムを返す。

引数

q	GgQuaternion 型の四元数.
-----	---------------------

戻り値

四元数 q のノルム.

`gg.h` の 4734 行目に定義があります。

呼び出し関係図:



7.1.3.49 ggNormal()

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

法線変換行列を返す.

引数

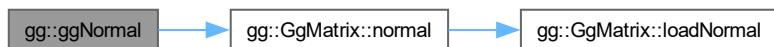
m	元の変換行列.
-----	---------

戻り値

m の法線変換行列.

`gg.h` の 3444 行目に定義があります。

呼び出し関係図:



7.1.3.50 ggNormalize()

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

正規化した四元数を返す.

引数

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

戻り値

四元数 `q` を正規化した四元数.

`gg.h` の 4745 行目に定義があります。

呼び出し関係図:



7.1.3.51 ggNormalize3() [1/4]

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

`GgVector` 型の 3 要素の正規化.

引数

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

戻り値

`a` の 3 要素を正規化したもの.

覚え書き

戻り値の `w` (第4) 要素は操作しない.

`gg.h` の 2045 行目に定義があります。

呼び出し関係図:



7.1.3.52 ggNormalize3() [2/4]

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

3要素の正規化。

引数

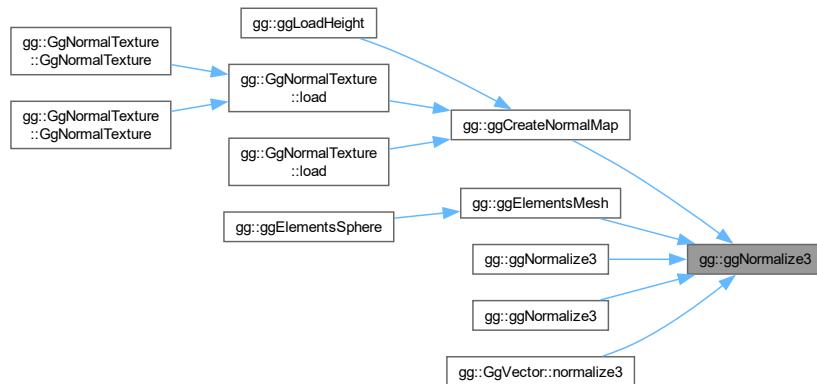
<i>a</i>	GLfloat型の3要素の配列変数。
<i>b</i>	<i>a</i> の3要素を正規化した結果を格納するGLfloat型の3要素の配列変数。

gg.h の 1478 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.53 ggNormalize3() [3/4]

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

GgVector型の3要素の正規化。

引数

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

覚え書き

a の w(第4) 要素は操作しない。

gg.h の 2060 行目に定義があります。

呼び出し関係図:



7.1.3.54 ggNormalize3() [4/4]

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

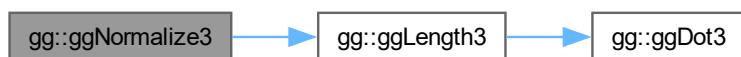
3要素の正規化。

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:



7.1.3.55 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 2106 行目に定義があります。

呼び出し関係図:



7.1.3.56 ggNormalize4() [2/4]

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

GLfloat 型の 4 要素の正規化.

引数

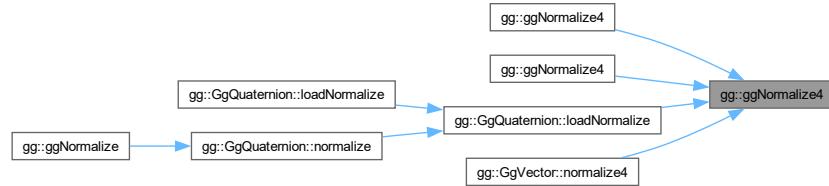
a	GLfloat 型の 4 要素の配列変数.
b	a を正規化した結果を格納する GLfloat 型の 4 要素の配列変数.

gg.h の 1543 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.57 ggNormalize4() [3/4]

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

GgVector 型の 4 要素の正規化。

引数

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

gg.h の 2118 行目に定義があります。

呼び出し関係図:



7.1.3.58 ggNormalize4() [4/4]

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

GLfloat 型の 4 要素の正規化。

引数

a	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:



7.1.3.59 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 3369 行目に定義があります。

呼び出し関係図:



7.1.3.60 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 3407 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:



7.1.3.61 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 5176 行目に定義があります。

7.1.3.62 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 5202 行目に定義があります。

7.1.3.63 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

q	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 4580 行目に定義があります。

呼び出し関係図:



7.1.3.64 ggQuaternionTransposeMatrix()

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

四元数 q の回転の転置した変換行列を返す。

引数

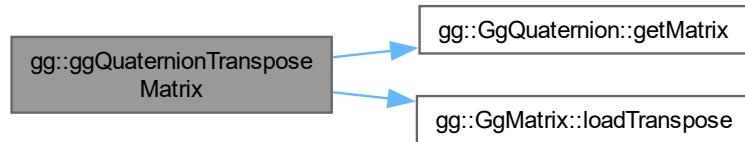
q	元の四元数.
-----	--------

戻り値

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

[gg.h](#) の 4593 行目に定義があります。

呼び出し関係図:



7.1.3.65 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

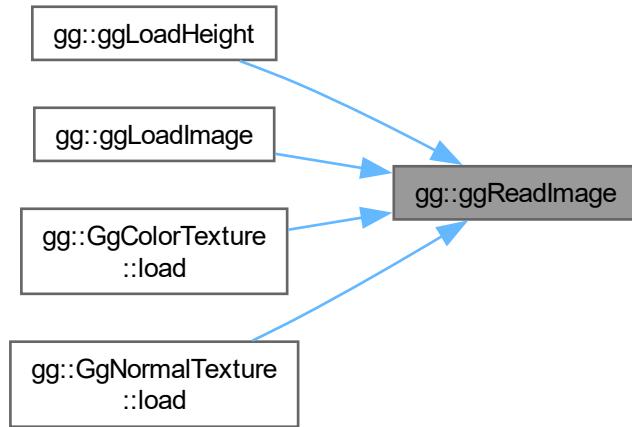
読み込みに成功すれば `true`, 失敗すれば `false`.

`gg.cpp` の 3667 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:



7.1.3.66 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

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

戻り値

GgTriangles 型のポインタ。

gg.cpp の 5229 行目に定義があります。

7.1.3.67 ggRotate() [1/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す。

引数

<i>r</i>	回転軸のベクトルと回転角を表す <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 3292 行目に定義があります。

呼び出し関係図:



7.1.3.68 `ggRotate()` [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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

戻り値

`r` を軸に `a` だけ回転する変換行列.

`gg.h` の 3268 行目に定義があります。

呼び出し関係図:



7.1.3.69 `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 3280 行目に定義があります。

呼び出し関係図:



7.1.3.70 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 GLfloat 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 3255 行目に定義があります。

呼び出し関係図:



7.1.3.71 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 3242 行目に定義があります。

呼び出し関係図:



7.1.3.72 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

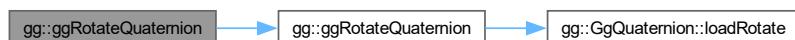
v	軸ベクトルを表す GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

回転を表す四元数.

gg.h の 4634 行目に定義があります。

呼び出し関係図:



7.1.3.73 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

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

回転を表す四元数.

gg.h の 4623 行目に定義があります。

呼び出し関係図:



7.1.3.74 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

x	軸ベクトルの x 成分.
y	軸ベクトルの y 成分.
z	軸ベクトルの z 成分.
a	回転角.

戻り値

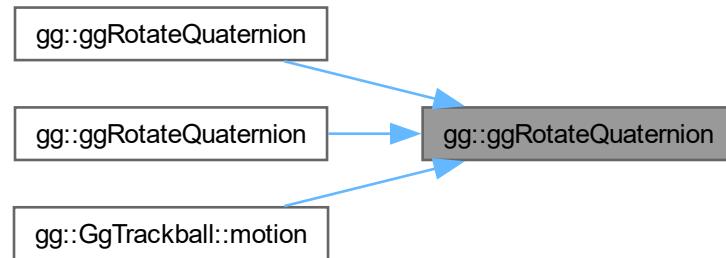
回転を表す四元数.

gg.h の 4610 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.75 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 3203 行目に定義があります。

呼び出し関係図:



7.1.3.76 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 3215 行目に定義があります。

呼び出し関係図:



7.1.3.77 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 3227 行目に定義があります。

呼び出し関係図:



7.1.3.78 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 3611 行目に定義があります。

呼び出し関係図:



7.1.3.79 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 3637 行目に定義があります。

呼び出し関係図:



7.1.3.80 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

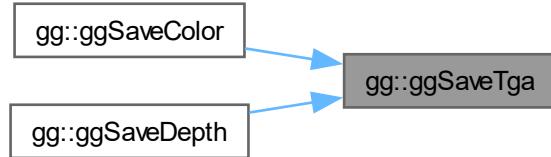
保存に成功すれば true, 失敗すれば false.

gg.cpp の 3535 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.81 ggScale() [1/3]

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

拡大縮小の変換行列を返す。

引数

<i>s</i>	拡大率の GgVector 型の変数。
----------	---------------------

戻り値

拡大縮小の変換行列。

gg.h の 3191 行目に定義があります。

呼び出し関係図:



7.1.3.82 ggScale() [2/3]

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

拡大縮小の変換行列を返す。

引数

<code>s</code>	拡大率の 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 3179 行目に定義がります。

呼び出し関係図:



7.1.3.83 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

<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>	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 3167 行目に定義がります。

呼び出し関係図:



7.1.3.84 ggSlerp() [1/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

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

戻り値

q, r を *t* で内分した四元数。

gg.h の 4697 行目に定義があります。

呼び出し関係図:



7.1.3.85 ggSlerp() [2/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

<i>q</i>	<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

q, a を *t* で内分した四元数.

`gg.h` の 4710 行目に定義があります。

呼び出し関係図:



7.1.3.86 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 4723 行目に定義があります。

呼び出し関係図:



7.1.3.87 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 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

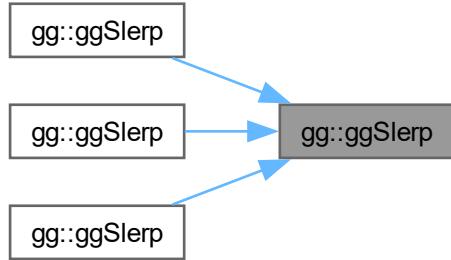
a, *b* を *t* で内分した四元数.

gg.h の 4683 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.88 ggTranslate() [1/3]

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

平行移動の変換行列を返す.

引数

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

戻り値

平行移動の変換行列.

gg.h の 3152 行目に定義があります。

呼び出し関係図:



7.1.3.89 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 (<i>x</i> , <i>y</i> , <i>z</i>).
----------	---

戻り値

平行移動の変換行列.

[gg.h](#) の 3140 行目に定義がります。

呼び出し関係図:



7.1.3.90 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 3128 行目に定義がります。

呼び出し関係図:



7.1.3.91 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 3422 行目に定義があります。

呼び出し関係図:



7.1.3.92 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

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

戻り値

a に *b* の各要素を乗じた積.

gg.h の 1967 行目に定義があります。

7.1.3.93 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

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

戻り値

v の値.

gg.h の 1920 行目に定義があります。

7.1.3.94 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

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

戻り値

a に *b* の各要素を足した和.

gg.h の 1932 行目に定義があります。

7.1.3.95 operator-() [1/2]

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

符号の反転.

引数

<i>v</i>	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 1943 行目に定義があります。

7.1.3.96 operator-() [2/2]

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

スカラーから GgVector 型の各要素を引いた差を返す.

引数

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

戻り値

a から *b* の各要素を引いた差.

gg.h の 1955 行目に定義があります。

7.1.3.97 operator/()

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

スカラーを GgVector 型の各要素で割った商を返す.

引数

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

戻り値

`a` を `b` の各要素で割った商.

`gg.h` の 1979 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 )
```

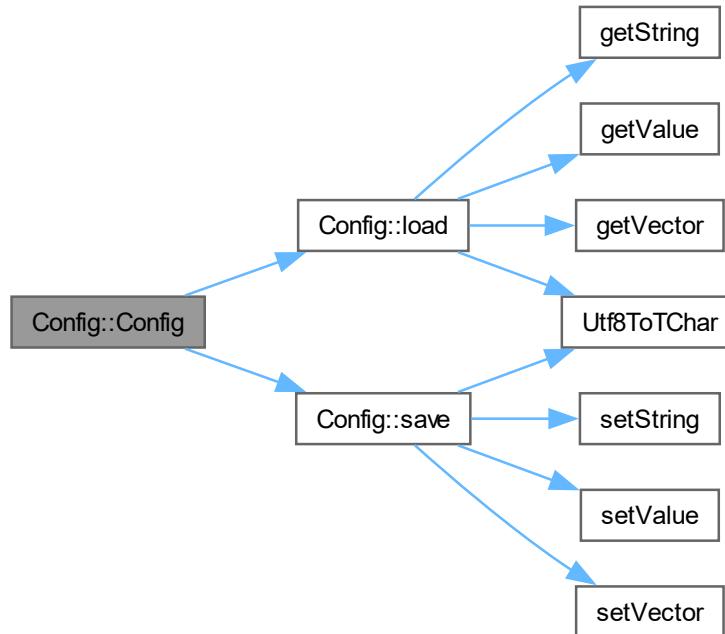
ファイルから構成データを読み込むコンストラクタ

引数

<i>filename</i>	読み込む構成ファイル名
-----------------	-------------

[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]`

ウィンドウの高さを得る

戻り値

ウィンドウの高さ

[Config.h](#) の 78 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

戻り値

 ウィンドウの横幅

[Config.h](#) の 68 行目に定義があります。

8.1.3.3 load()

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

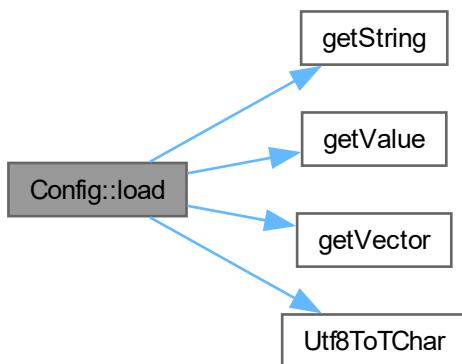
構成ファイルを読み込む

引数

<i>filename</i>	読み込む構成ファイル名
-----------------	-------------

[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
```

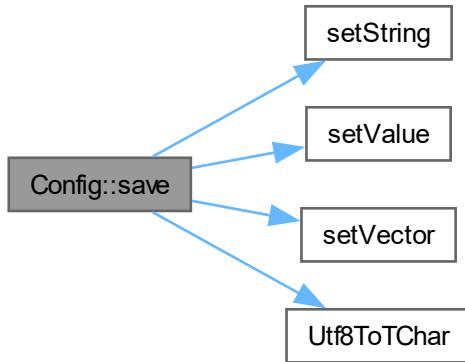
構成ファイルを書き出す

引数

書き出す構成ファイル名	<input type="text"/>
-------------	----------------------

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 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)`

8.2.1 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 行目に定義があります。

8.2.2 構築子と解体子

8.2.2.1 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 行目に定義があります。

8.2.2.2 GgApp() [2/3]

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

コピー構造子は使用しない。

引数

<i>w</i>	コピー元のオブジェクト.
----------	--------------

8.2.2.3 GgApp() [3/3]

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

ムーブ構造子。

引数

<i>w</i>	ムーブ元のオブジェクト.
----------	--------------

8.2.2.4 ~GgApp()

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

デストラクタ。

GgApp.cpp の 95 行目に定義があります。

8.2.3 関数詳解

8.2.3.1 main()

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

アプリケーション本体。

引数

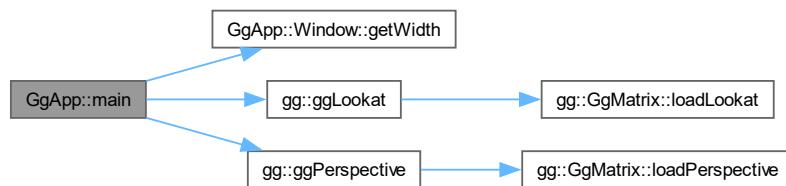
<i>argc</i>	コマンドライン引数の数.
<i>argv</i>	コマンドライン引数の配列.

戻り値

アプリケーションの終了ステータス.

[ggsample01.cpp](#) の 27 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:



8.2.3.2 operator=() [1/2]

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

代入演算子は使用しない.

引数

<i>w</i>	代入元のオブジェクト.
----------	-------------

戻り値

代入後のこのオブジェクトの参照.

8.2.3.3 operator=() [2/2]

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

ムーブ代入演算子.

引数

w	ムーブ代入元のオブジェクト.
---	----------------

戻り値

ムーブ代入後のこのオブジェクトの参照.

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

8.3 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

8.3.1 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 5571 行目に定義があります。

8.3.2 構築子と解体子

8.3.2.1 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 5583 行目に定義があります。

呼び出し関係図:



8.3.2.2 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。

引数

buffer	コピー元のバッファ。
--------	------------

8.3.2.3 GgBuffer() [3/3]

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

ムーブコンストラクタ。

引数

buffer	ムーブ元のバッファ。
--------	------------

8.3.2.4 ~GgBuffer()

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

デストラクタ。

gg.h の 5583 行目に定義があります。

8.3.3 関数詳解

8.3.3.1 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 5696 行目に定義があります。

8.3.3.2 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 5801 行目に定義がります。

8.3.3.3 **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 5688 行目に定義がります。

8.3.3.4 **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 5678 行目に定義がります。

8.3.3.5 **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 5668 行目に定義がります。

8.3.3.6 `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 5658 行目に定義があります。

8.3.3.7 `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 5714 行目に定義があります。

被呼び出し関係図:



8.3.3.8 `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 5731 行目に定義があります。

8.3.3.9 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.

引数

<i>buffer</i>	代入元のバッファ.
---------------	-----------

戻り値

代入後のこのバッファの参照.

8.3.3.10 operator=() [2/2]

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

ムーブ代入演算子.

引数

<i>buffer</i>	ムーブ代入元のバッファ.
---------------	--------------

戻り値

ムーブ代入後のこのバッファの参照.

8.3.3.11 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 5774 行目に定義があります。

8.3.3.12 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 5756 行目に定義があります。

8.3.3.13 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 5704 行目に定義があります。

8.3.3.14 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 5744 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](#)

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

8.4.1 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 5350 行目](#)に定義があります。

8.4.2 構築子と解体子

8.4.2.1 GgColorTexture() [1/3]

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

コンストラクタ.

[gg.h の 5360 行目](#)に定義があります。

8.4.2.2 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]
```

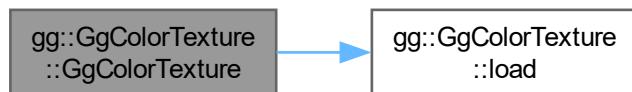
メモリ上のデータからカラーのテクスチャを作成するコンストラクタ.

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 5376 行目に定義があります。

呼び出し関係図:



8.4.2.3 GgColorTexture() [3/3]

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

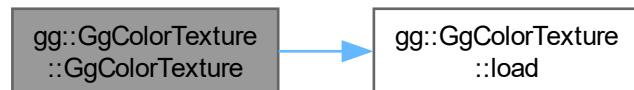
TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成するコンストラクタ.

引数

<i>name</i>	読み込むファイル名.
<i>internal</i>	<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, <code>GL_TEXTURE_WRAP_S</code> および <code>GL_TEXTURE_WRAP_T</code> に設定する値.

[gg.h](#) の 5397 行目に定義があります。

呼び出し関係図:



8.4.2.4 ~GgColorTexture()

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

デストラクタ.

gg.h の 5409 行目に定義があります。

8.4.3 関数詳解

8.4.3.1 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]
```

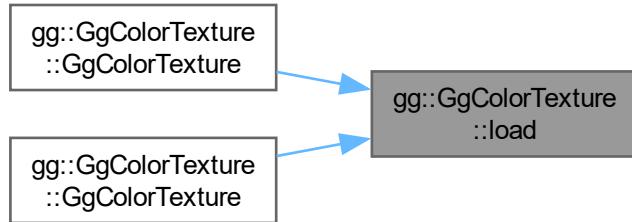
メモリ上のデータを読み込んでテクスチャを作成する.

引数

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

[gg.h](#) の 5425 行目に定義がります。

被呼び出し関係図:



8.4.3.2 load() [2/2]

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

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

引数

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

[gg.cpp](#) の 4022 行目に定義がります。

呼び出し関係図:



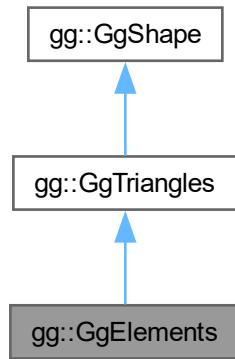
このクラス詳解は次のファイルから抽出されました:

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

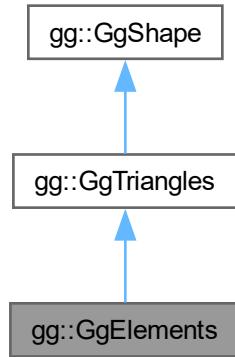
8.5 gg::GgElements クラス

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

gg::GgElements の継承関係図



gg::GgElements 連携図



公開 メンバ関数

- [GgElements](#) (GLenum mode=GL_TRIANGLES)
- [GgElements](#) (const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum mode=GL_TRIANGLES, GLenum usage=GL_STATIC_DRAW)
- virtual [~GgElements](#) ()
- const GLsizei & [getIndexCount](#) () const

- const GLuint & [getIndexBuffer \(\) const](#)
- void [send \(const GgVertex *vert, GLuint firstv, GLsizei countv, const GLuint *face=nullptr, GLuint firstf=0, GLsizei countf=0\) const](#)
- void [load \(const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum usage=GL_STATIC_DRAW\)](#)
- virtual void [draw \(GLint first=0, GLsizei count=0\) const](#)

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 [operator! \(\) const noexcept](#)
- const GLuint & [get \(\) const](#)
- void [setMode \(GLenum mode\)](#)
- const GLenum & [getMode \(\) const](#)

8.5.1 詳解

三角形で表した形状データ (Elements 形式).

[gg.h](#) の 6606 行目に定義があります。

8.5.2 構築子と解体子

8.5.2.1 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 6619 行目に定義があります。

8.5.2.2 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 6634 行目に定義があります。

8.5.2.3 ~GgElements()

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

デストラクタ.

gg.h の 6650 行目に定義があります。

8.5.3 関数詳解

8.5.3.1 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 5163 行目に定義がります。

呼び出し関係図:



8.5.3.2 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 6668 行目に定義がります。

8.5.3.3 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 6658 行目に定義がります。

8.5.3.4 load()

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

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

引数

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

[gg.h](#) の 6705 行目に定義があります。

8.5.3.5 send()

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

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

引数

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

[gg.h](#) の 6683 行目に定義があります。

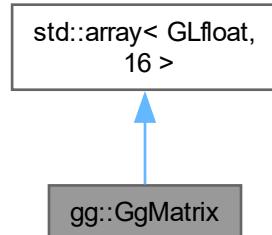
このクラス詳解は次のファイルから抽出されました:

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

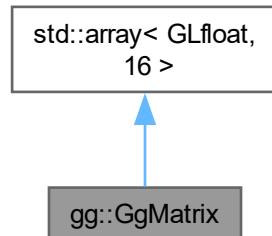
8.6 gg::GgMatrix クラス

```
#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`

8.6.1 詳解

変換行列.

`gg.h` の 2126 行目に定義があります。

8.6.2 構築子と解体子

8.6.2.1 `GgMatrix()` [1/6]

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

コンストラクタ. 被呼び出し関係図:



8.6.2.2 GgMatrix() [2/6]

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

コンストラクタ.

引数

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

gg.h の 2161 行目に定義があります。

8.6.2.3 GgMatrix() [3/6]

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

コンストラクタ.

引数

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

gg.h の 2176 行目に定義があります。

8.6.2.4 GgMatrix() [4/6]

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

コンストラクタ.

引数

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

gg.h の 2186 行目に定義があります。

8.6.2.5 GgMatrix() [5/6]

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

コピーコンストラクタ.

引数

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

8.6.2.6 GgMatrix() [6/6]

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

ムーブコンストラクタ.

引数

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

8.6.2.7 ~GgMatrix()

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

デストラクタ.

8.6.3 関数詳解

8.6.3.1 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

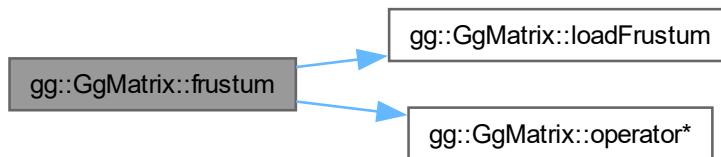
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2969 行目に定義がります。

呼び出し関係図:



8.6.3.2 get() [1/2]

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

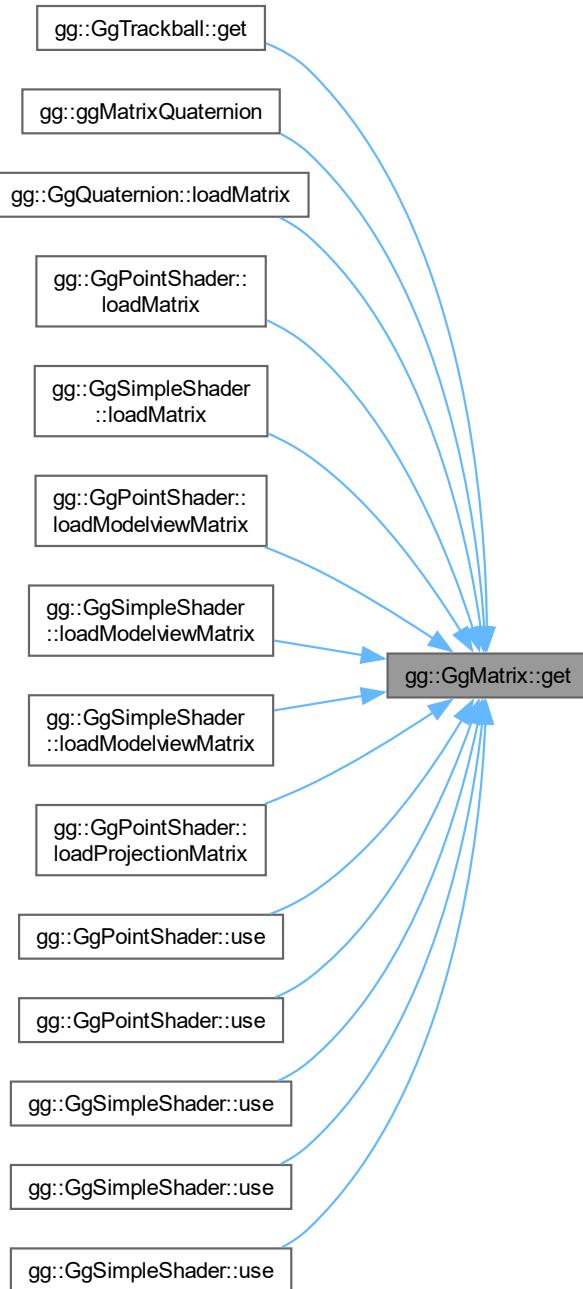
変換行列を取り出す.

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 3092 行目に定義があります。

被呼び出し関係図:



8.6.3.3 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 3102 行目に定義があります。

8.6.3.4 invert()

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

逆行列を返す。

戻り値

逆行列。

gg.h の 3013 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.5 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

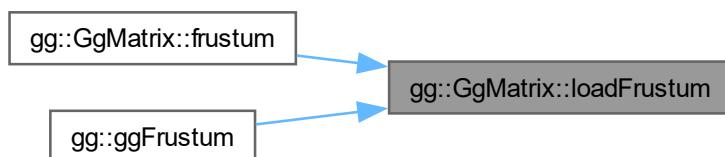
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

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:



8.6.3.6 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。

呼び出し関係図:



8.6.3.7 loadInvert() [1/2]

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

逆行列を格納する。

引数

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

戻り値

設定した *m* の逆行列。

gg.h の 2692 行目に定義があります。

呼び出し関係図:



8.6.3.8 loadInvert() [2/2]

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

逆行列を格納する。

引数

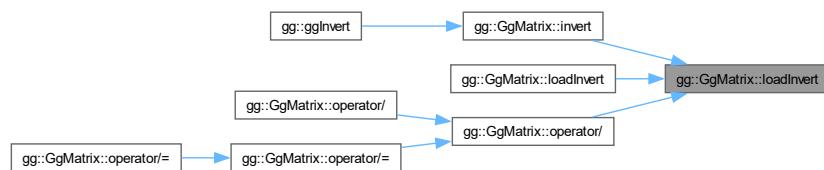
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

設定した 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:



8.6.3.9 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

e	観点の位置の GgVector 型の変数.
t	目標点の位置の GgVector 型の変数.
u	上方向のベクトルの GgVector 型の変数.

戻り値

設定したビュー変換行列.

gg.h の 2612 行目に定義があります。

呼び出し関係図:



8.6.3.10 loadLookat() [2/3]

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

ビュー変換行列を格納する。

引数

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

戻り値

設定したビュー変換行列。

gg.h の 2599 行目に定義があります。

呼び出し関係図:



8.6.3.11 loadLookat() [3/3]

```
ggl::GgMatrix & gg::GgMatrix::loadLookat (
    GLfloat ex,
```

```
GLfloat ey,
GLfloat ez,
GLfloat tx,
GLfloat ty,
GLfloat tz,
GLfloat ux,
GLfloat uy,
GLfloat uz )
```

ビュー変換行列を格納する。

引数

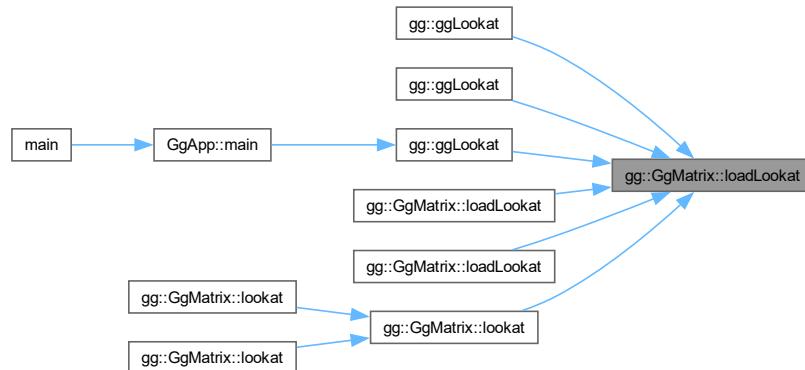
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列。

`gg.cpp` の 3023 行目に定義があります。

被呼び出し関係図:



8.6.3.12 `loadNormal()` [1/2]

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

法線変換行列を格納する。

引数

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

戻り値

設定した `m` の法線変換行列.

`gg.h` の 2711 行目に定義がります。

呼び出し関係図:



8.6.3.13 loadNormal() [2/2]

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

法線変換行列を格納する.

引数

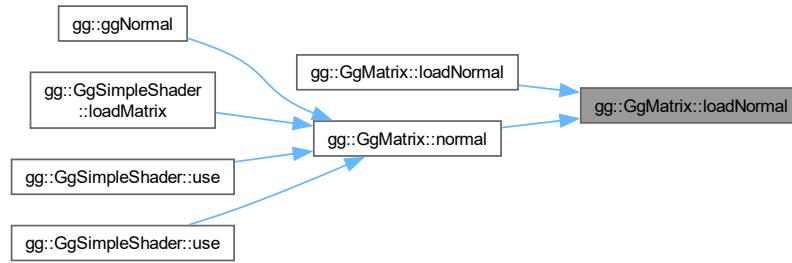
<code>a</code>	GLfloat 型の 16 要素の変換行列.
----------------	------------------------

戻り値

設定した `m` の法線変換行列.

`gg.cpp` の 2992 行目に定義がります。

被呼び出し関係図:



8.6.3.14 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

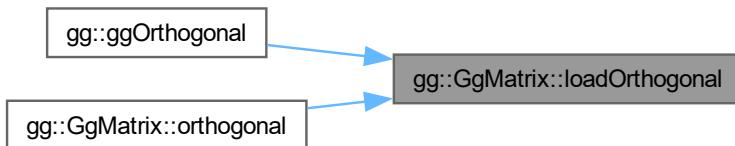
<i>left</i>	ウィンドウの左端の位置。
<i>right</i>	ウィンドウの右端の位置。
<i>bottom</i>	ウィンドウの下端の位置。
<i>top</i>	ウィンドウの上端の位置。
<i>zNear</i>	視点から前方面までの位置。
<i>zFar</i>	視点から後方面までの位置。

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:



8.6.3.15 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

<code>fovy</code>	y 方向の画角.
<code>aspect</code>	縦横比.
<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

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:



8.6.3.16 `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

<code>r</code>	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 2568 行目に定義があります。

呼び出し関係図:



8.6.3.17 `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

<code>r</code>	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.
<code>a</code>	回転角.

戻り値

設定した変換行列.

`gg.h` の 2546 行目に定義があります。

呼び出し関係図:



8.6.3.18 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

r	回転軸の方向ベクトルと回転角を格納した <code>GLfloat</code> 型の 4 要素の配列変数, 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 2557 行目に定義があります。

呼び出し関係図:



8.6.3.19 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 <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

設定した変換行列.

`gg.h` の 2534 行目に定義があります。

呼び出し関係図:



8.6.3.20 `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

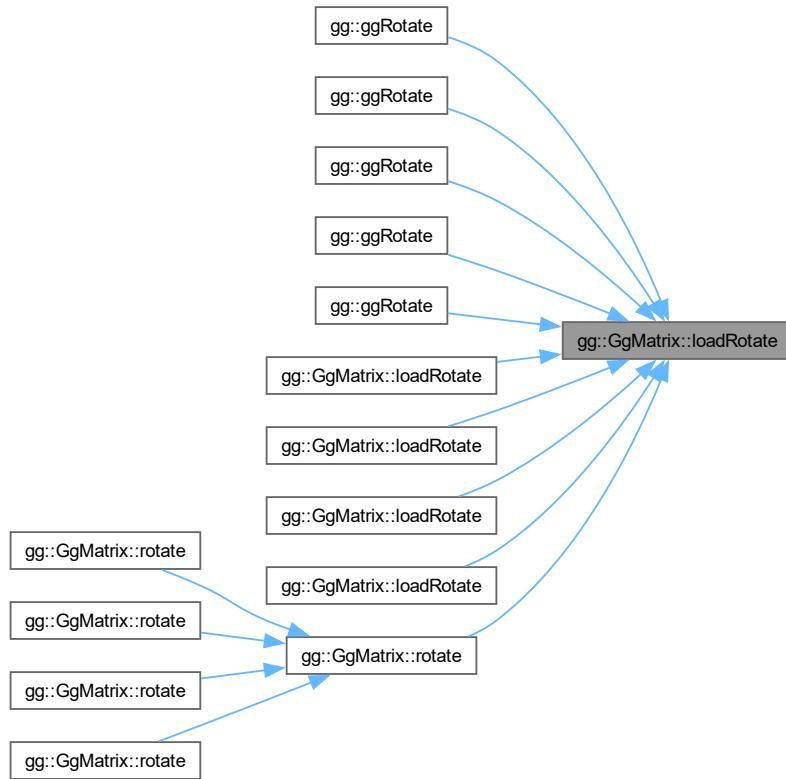
<i>x</i>	回転軸の <i>x</i> 成分.
<i>y</i>	回転軸の <i>y</i> 成分.
<i>z</i>	回転軸の <i>z</i> 成分.
<i>a</i>	回転角.

戻り値

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:



8.6.3.21 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

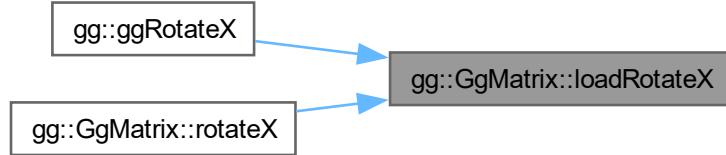
a	回転角.
---	------

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:



8.6.3.22 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

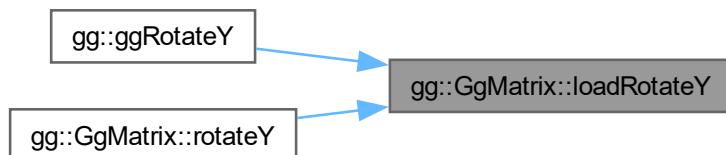
a	回転角.
---	------

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:



8.6.3.23 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

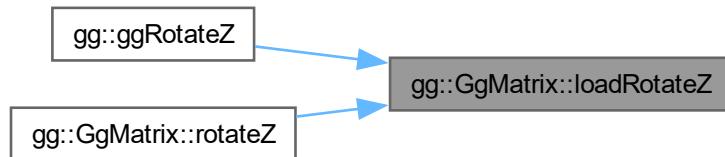
a	回転角.
---	------

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。

被呼び出し関係図:



8.6.3.24 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 2487 行目に定義があります。

呼び出し関係図:



8.6.3.25 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 GLfloat 型の配列 (x, y, z).
----------------	------------------------------

戻り値

設定した変換行列.

gg.h の 2476 行目に定義があります。

呼び出し関係図:



8.6.3.26 loadScale() [3/3]

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

拡大縮小の変換行列を格納する。

引数

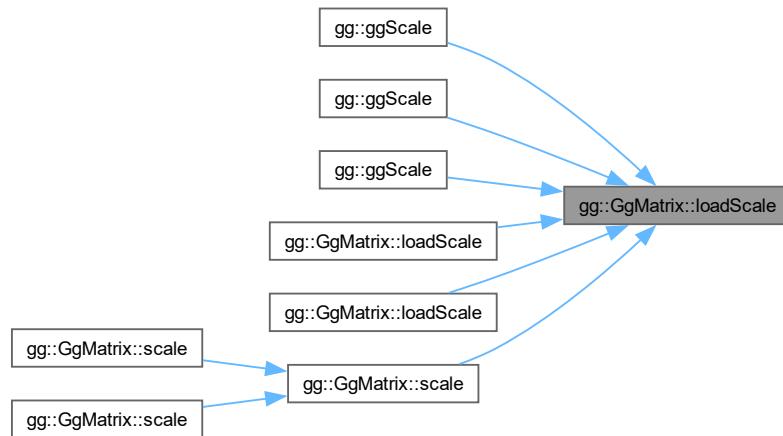
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 拡大率のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2766 行目に定義があります。

被呼び出し関係図:



8.6.3.27 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

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

戻り値

設定した変換行列.

gg.h の 2454 行目に定義がります。

呼び出し関係図:



8.6.3.28 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 2443 行目に定義がります。

呼び出し関係図:



8.6.3.29 loadTranslate() [3/3]

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

平行移動の変換行列を格納する。

引数

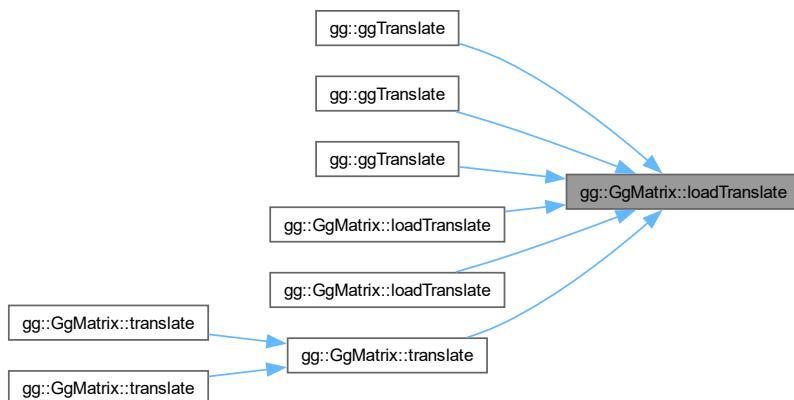
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2750 行目に定義があります。

被呼び出し関係図:



8.6.3.30 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 2673 行目に定義があります。

呼び出し関係図:



8.6.3.31 `loadTranspose()` [2/2]

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

転置行列を格納する.

引数

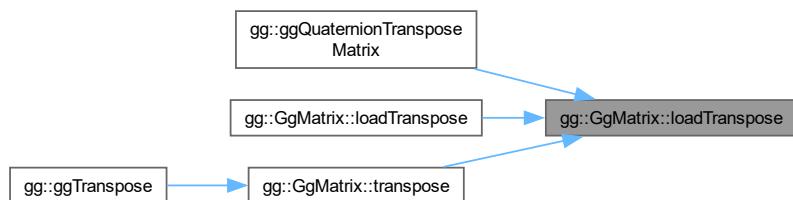
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列.
----------------	-------------------------------------

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:



8.6.3.32 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 2932 行目に定義があります。

呼び出し関係図:



8.6.3.33 lookat() [2/3]

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

ビュー変換を乗じた結果を返す.

引数

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

戻り値

ビュー変換行列を乗じた変換行列.

[gg.h](#) の 2919 行目に定義があります。

呼び出し関係図:



8.6.3.34 `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

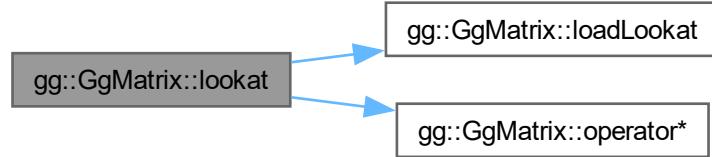
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

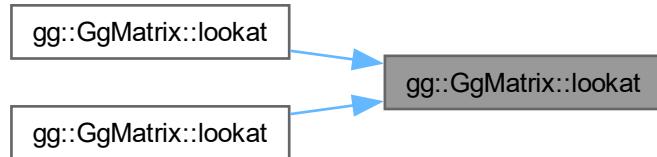
ビュー変換行列を乗じた変換行列.

[gg.h](#) の 2901 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.35 normal()

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

法線変換行列を返す。

戻り値

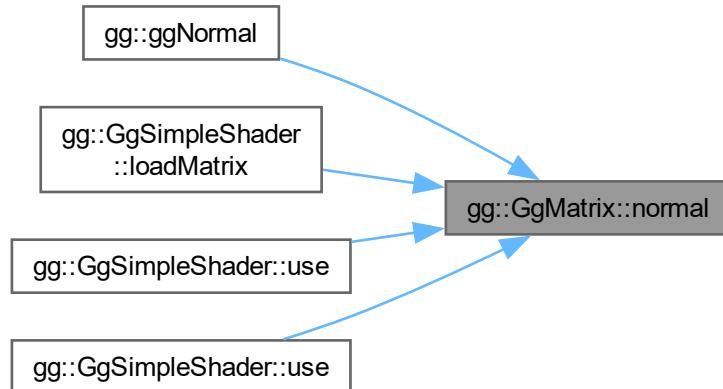
法線変換行列。

`gg.h` の 3024 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.36 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 2345 行目に定義があります。

呼び出し関係図:



8.6.3.37 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 GgVector 型の変数。
---	------------------------

戻り値

v 変換結果の GgVector 型のベクトル。

gg.h の 3080 行目に定義があります。

8.6.3.38 operator*() [3/3]

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

変換行列に配列に格納した変換行列を乗算した値を返す。

引数

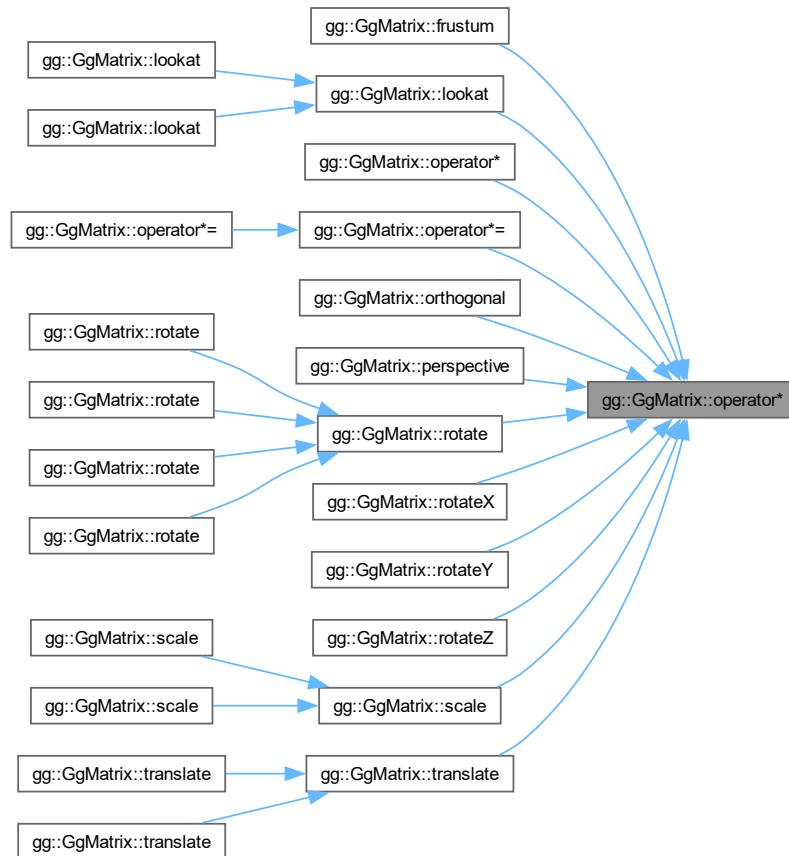
a	変換行列を格納した GLfloat 型の 16 要素の配列変数。
---	----------------------------------

戻り値

変換行列に a を乗じた GgMatrix 型の変換行列。

gg.h の 2332 行目に定義があります。

被呼び出し関係図:



8.6.3.39 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

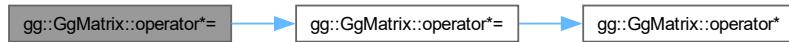
<i>m</i>	<i>GgMatrix</i> 型の変換行列。
----------	-------------------------

戻り値

m を掛けた変換行列の参照。

gg.h の 2368 行目に定義があります。

呼び出し関係図:



8.6.3.40 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 2356 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.41 operator+() [1/2]

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

変換行列に別の変換行列を加算した値を返す.

引数

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

戻り値

変換行列に *a* を加えた GgMatrix 型の変換行列.

gg.h の 2251 行目に定義があります。

呼び出し関係図:



8.6.3.42 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

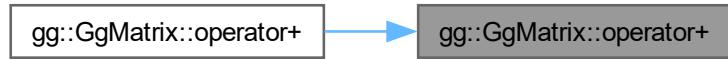
<i>a</i>	変換行列を格納した GLfloat 型の 16 要素の配列変数.
----------	----------------------------------

戻り値

変換行列に *a* を加えた GgMatrix 型の変換行列.

gg.h の 2238 行目に定義があります。

呼び出し関係図:



8.6.3.43 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

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

戻り値

m を加えた変換行列の参照。

gg.h の 2274 行目に定義があります。

呼び出し関係図:



8.6.3.44 operator+=() [2/2]

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

変換行列に配列に格納した変換行列を加算した結果を格納する。

引数

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

戻り値

a を加えた変換行列の参照.

gg.h の 2262 行目に定義があります。

呼び出し関係図:



8.6.3.45 operator-() [1/2]

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

変換行列から別の変換行列を減算した値を返す.

引数

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

戻り値

変換行列から m を引いた GgMatrix 型の変換行列.

gg.h の 2298 行目に定義があります。

呼び出し関係図:



8.6.3.46 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

a	変換行列を格納した GLfloat 型の 16 要素の配列変数。
---	----------------------------------

戻り値

変換行列から a を引いた GgMatrix 型の変換行列。

gg.h の 2285 行目に定義があります。

被呼び出し関係図:



8.6.3.47 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

m	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 2321 行目に定義があります。

呼び出し関係図:



8.6.3.48 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

<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

a を引いた変換行列の参照。

gg.h の 2309 行目に定義があります。

被呼び出し関係図:



8.6.3.49 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

<code>m</code>	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 2393 行目に定義があります。

呼び出し関係図:



8.6.3.50 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

<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

変換行列を `a` で割った GgMatrix 型の変換行列.

gg.h の 2379 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.51 operator/() [1/2]

```
GgMatrix & gg::GgMatrix::operator/=
    const GgMatrix & m) [inline]
```

変換行列を別の変換行列で除算した結果を格納する。

引数

<i>m</i>	<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 2416 行目に定義があります。

呼び出し関係図:



8.6.3.52 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

<i>a</i>	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 2404 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.53 operator=() [1/3]

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

代入演算子.

8.6.3.54 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 2226 行目に定義がります。

呼び出し関係図:



8.6.3.55 operator=() [3/3]

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

ムーブ代入演算子.

8.6.3.56 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

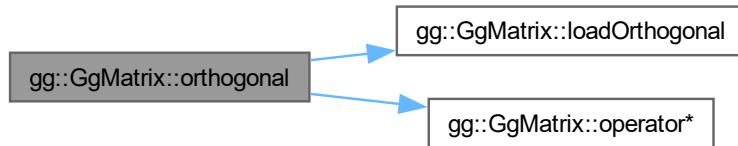
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

直交投影変換行列を乗じた変換行列.

gg.h の 2948 行目に定義がります。

呼び出し関係図:



8.6.3.57 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

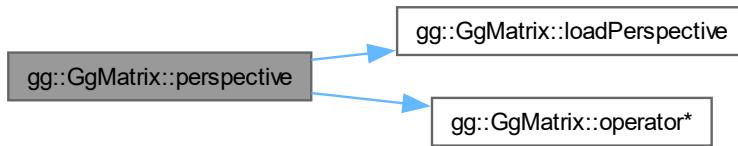
<code>fovy</code>	y 方向の画角.
<code>aspect</code>	縦横比.
<code>zNear</code>	視点から前方面までの位置.
<code>zFar</code>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

`gg.h` の 2988 行目に定義があります。

呼び出し関係図:



8.6.3.58 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 <code>GgVector</code> 型の変数.
<i>v</i>	元のベクトルの <code>GgVector</code> 型の変数.

`gg.h` の 3069 行目に定義があります。

8.6.3.59 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 <code>GgVector</code> 型の変数.
<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 3058 行目に定義があります。

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

`gg.h` の 3047 行目に定義があります。

8.6.3.61 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

c	変換結果を格納する GLfloat 型の 4 要素の配列変数.
v	元のベクトルの GLfloat 型の 4 要素の配列変数.

gg.h の 3036 行目に定義があります。

8.6.3.62 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

r	回転軸の方向ベクトルと回転角を格納した GgVector 型の変数).
----------	-------------------------------------

戻り値

(r[0], r[1], r[2]) を軸にさらに r[3] 回転した変換行列。

gg.h の 2882 行目に定義があります。

呼び出し関係図:



8.6.3.63 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 2860 行目に定義があります。

呼び出し関係図:



8.6.3.64 `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

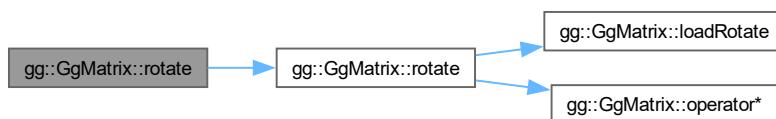
<i>r</i>	回転軸の方向ベクトルと回転角を格納した <code>GLfloat</code> 型の 4 要素の配列変数 (<i>x, y, z, a</i>).
----------	--

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 2871 行目に定義があります。

呼び出し関係図:



8.6.3.65 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

r	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).
a	回転角.

戻り値

$(r[0], r[1], r[2])$ を軸にさらに a 回転した変換行列.

gg.h の 2848 行目に定義があります。

呼び出し関係図:



8.6.3.66 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

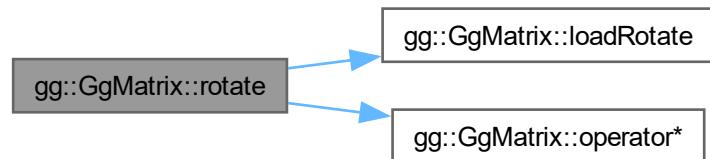
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

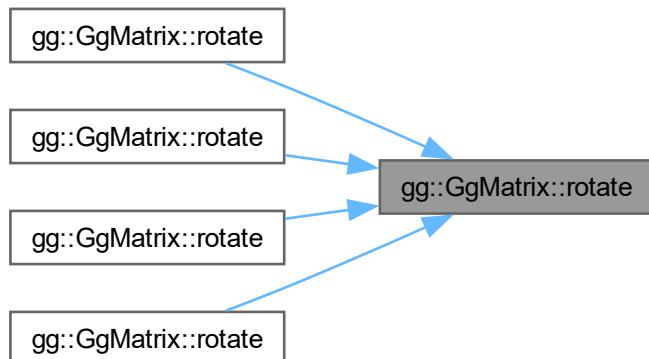
(x, y, z) を軸にさらに a 回転した変換行列.

gg.h の 2835 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.67 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

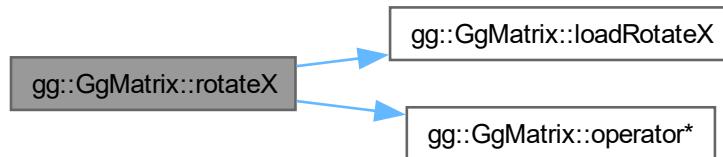
a	回転角.
-----	------

戻り値

x 軸中心にさらに a 回転した変換行列.

gg.h の 2796 行目に定義があります。

呼び出し関係図:



8.6.3.68 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

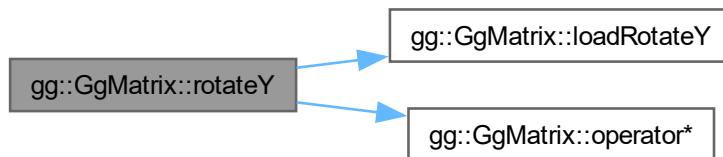
a	回転角.
---	------

戻り値

y 軸中心にさらに a 回転した変換行列.

gg.h の 2808 行目に定義があります。

呼び出し関係図:



8.6.3.69 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

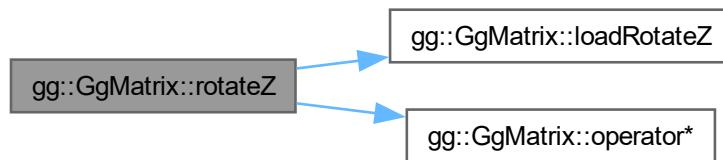
a	回転角.
---	------

戻り値

z 軸中心にさらに a 回転した変換行列。

gg.h の 2820 行目に定義があります。

呼び出し関係図:



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

戻り値

拡大縮小した結果の変換行列。

gg.h の 2785 行目に定義があります。

呼び出し関係図:



8.6.3.71 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

<i>s</i>	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 2774 行目に定義があります。

呼び出し関係図:



8.6.3.72 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

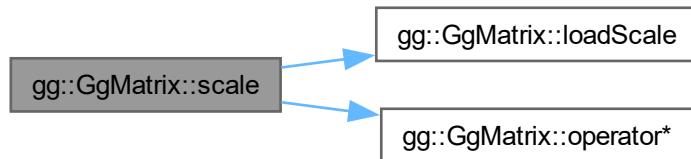
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

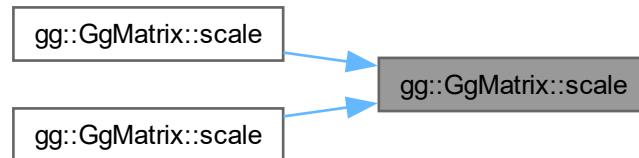
拡大縮小した結果の変換行列.

`gg.h` の 2762 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.73 `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

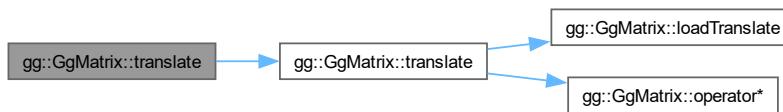
<i>t</i>	移動量の GgVector 型の変数.
----------	---------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2748 行目に定義があります。

呼び出し関係図:



8.6.3.74 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

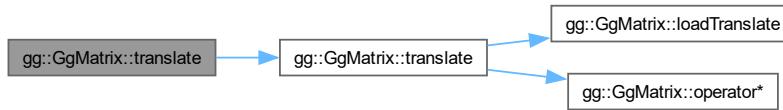
<i>t</i>	移動量の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2737 行目に定義があります。

呼び出し関係図:



8.6.3.75 `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

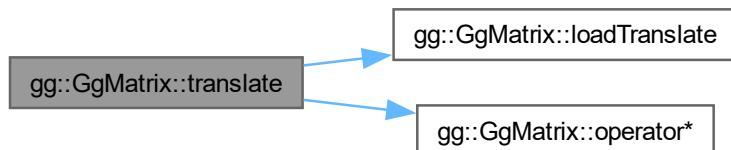
<code>x</code>	<code>x</code> 方向の移動量.
<code>y</code>	<code>y</code> 方向の移動量.
<code>z</code>	<code>z</code> 方向の移動量.
<code>w</code>	<code>w</code> 移動量のスケールファクタ (= 1.0f).

戻り値

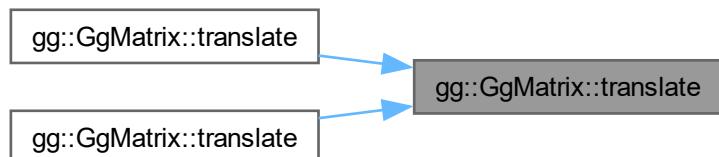
平行移動した結果の変換行列.

`gg.h` の 2725 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.6.3.76 transpose()

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

転置行列を返す。

戻り値

転置行列。

gg.h の 3002 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](#)

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

8.7.1 詳解

法線マップ.

覚え書き

高さマップ（グレースケール画像）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 5460 行目に定義があります。

8.7.2 構築子と解体子

8.7.2.1 GgNormalTexture() [1/3]

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

コンストラクタ.

[gg.h](#) の 5470 行目に定義があります。

8.7.2.2 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 5484 行目に定義があります。

呼び出し関係図:



8.7.2.3 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 5504 行目に定義があります。

呼び出し関係図:



8.7.2.4 ~GgNormalTexture()

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

デストラクタ.

gg.h の 5517 行目に定義があります。

8.7.3 関数詳解

8.7.3.1 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

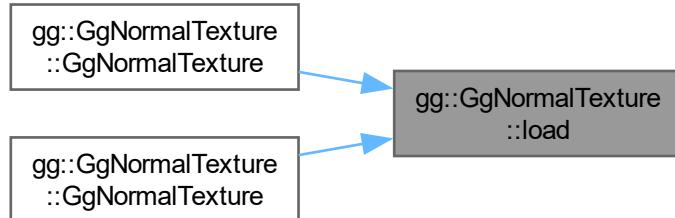
<i>hmap</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット (<code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code> , <code>GL_RGBA</code>).
<i>nz</i>	法線マップの <i>z</i> 成分の値.
<i>internal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

[gg.h](#) の 5531 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:



8.7.3.2 `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 4058 行目に定義があります。

呼び出し関係図:



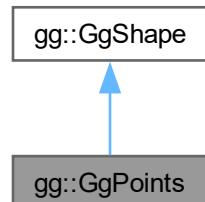
このクラス詳解は次のファイルから抽出されました:

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

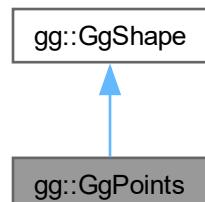
8.8 gg::GgPoints クラス

```
#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`

8.8.1 詳解

点。

`gg.h` の 6333 行目に定義がります。

8.8.2 構築子と解体子

8.8.2.1 `GgPoints()` [1/2]

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

コンストラクタ。

`gg.h` の 6344 行目に定義がります。

8.8.2.2 `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 6357 行目に定義がります。

8.8.2.3 ~GgPoints()

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

デストラクタ.

gg.h の 6371 行目に定義がります。

8.8.3 関数詳解

8.8.3.1 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 5120 行目に定義がります。

呼び出し関係図:



8.8.3.2 `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 6410 行目に定義があります。

8.8.3.3 `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 6400 行目に定義があります。

8.8.3.4 `load()`

```
void gg::GgPoints::load (
    const GgVector * pos,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW )
```

バッファオブジェクトを確保して頂点の位置データを格納する.

引数

<i>pos</i>	頂点の位置データが格納されている領域の先頭のポインタ.
<i>count</i>	頂点のデータの数(頂点数).
<i>usage</i>	バッファオブジェクトの使い方.

[gg.cpp](#) の 5107 行目に定義があります。

8.8.3.5 `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 6380 行目に定義があります。

8.8.3.6 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 6390 行目に定義があります。

8.8.3.7 send()

```
void gg::GgPoints::send (
    const GgVector * pos,
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

既存のバッファオブジェクトに頂点の位置データを転送する。

引数

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

[gg.h](#) の 6422 行目に定義があります。

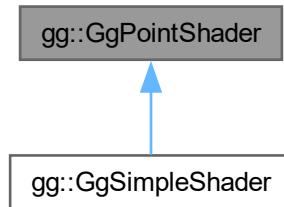
このクラス詳解は次のファイルから抽出されました:

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

8.9 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`

8.9.1 詳解

点のシェーダ.

`gg.h` の 6977 行目に定義があります。

8.9.2 構築子と解体子

8.9.2.1 GgPointShader() [1/3]

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

コンストラクタ.

`gg.h` の 6993 行目に定義があります。

8.9.2.2 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 7008 行目に定義がります。

8.9.2.3 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 7027 行目に定義がります。

8.9.2.4 ~GgPointShader()

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

デストラクタ.

gg.h の 7039 行目に定義がります。

8.9.3 関数詳解

8.9.3.1 `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 7223 行目に定義があります。

8.9.3.2 `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

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

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 7086 行目に定義があります。

8.9.3.3 `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

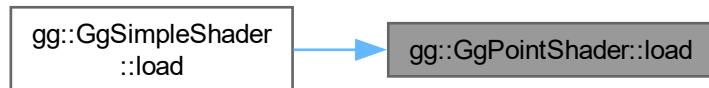
<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

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 7053 行目に定義があります。

被呼び出し関係図:



8.9.3.4 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 7153 行目に定義があります。

呼び出し関係図:



8.9.3.5 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 7141 行目に定義があります。

8.9.3.6 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 7130 行目に定義があります。

呼び出し関係図:



8.9.3.7 loadModelviewMatrix() [2/2]

```
virtual void gg::GgPointShader::loadModelviewMatrix (\n    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 7120 行目に定義があります。

8.9.3.8 loadProjectionMatrix() [1/2]

```
virtual void gg::GgPointShader::loadProjectionMatrix (\n    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 7110 行目に定義があります。

呼び出し関係図:



8.9.3.9 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 7100 行目に定義があります。

8.9.3.10 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 7213 行目に定義があります。

8.9.3.11 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 7161 行目に定義があります。

8.9.3.12 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 7182 行目に定義があります。

呼び出し関係図:



8.9.3.13 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 7205 行目に定義があります。

呼び出し関係図:



8.9.3.14 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 7171 行目に定義があります。

8.9.3.15 `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 7193 行目に定義があります。

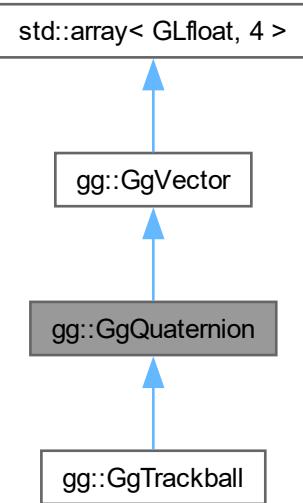
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

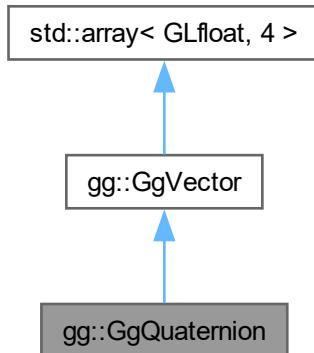
8.10 gg::GgQuaternion クラス

```
#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 euler (const GgVector &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](#)

8.10.1 詳解

四元数.

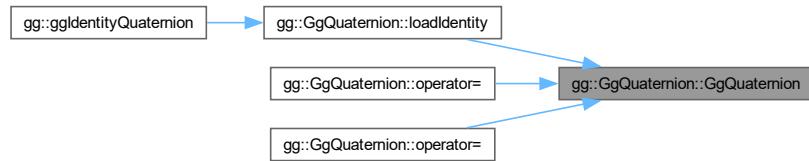
[gg.h](#) の 3452 行目に定義があります。

8.10.2 構築子と解体子

8.10.2.1 GgQuaternion() [1/7]

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

コンストラクタ、被呼び出し関係図:



8.10.2.2 GgQuaternion() [2/7]

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

コンストラクタ.

引数

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

gg.h の 3481 行目に定義があります。

8.10.2.3 GgQuaternion() [3/7]

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

コンストラクタ.

引数

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

gg.h の 3491 行目に定義があります。

8.10.2.4 GgQuaternion() [4/7]

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

コンストラクタ.

引数

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

gg.h の 3501 行目に定義があります。

8.10.2.5 GgQuaternion() [5/7]

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

コンストラクタ.

引数

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

gg.h の 3511 行目に定義があります。

8.10.2.6 GgQuaternion() [6/7]

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

コピーコンストラクタ.

引数

<i>q</i>	コピー元の四元数.
----------	-----------

8.10.2.7 GgQuaternion() [7/7]

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

ムーブコンストラクタ.

引数

<i>q</i>	ムーブ元の四元数.
----------	-----------

8.10.2.8 ~GgQuaternion()

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

デストラクタ.

8.10.3 関数詳解

8.10.3.1 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 3804 行目に定義があります。

呼び出し関係図:



8.10.3.2 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 3793 行目に定義があります。

呼び出し関係図:



8.10.3.3 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 3782 行目に定義があります。

呼び出し関係図:



8.10.3.4 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

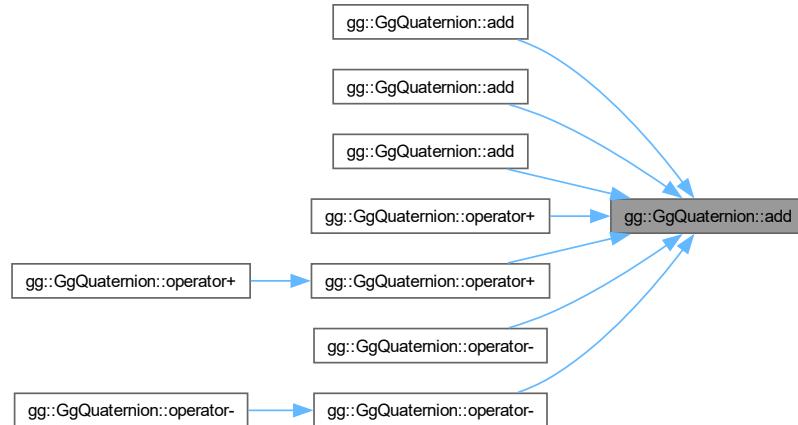
(*x*, *y*, *z*, *w*) を加えた四元数.

gg.h の 3770 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.5 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 4441 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.6 `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

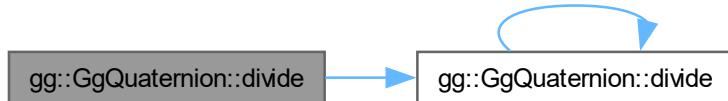
<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

`q` で割った四元数.

`gg.h` の 3953 行目に定義があります。

呼び出し関係図:



8.10.3.7 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

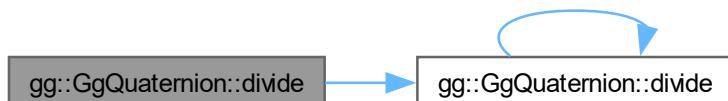
v	四元数を格納した <code>GgVector</code> 型の変数。
---	--------------------------------------

戻り値

v で割った四元数。

`gg.h` の 3942 行目に定義があります。

呼び出し関係図:



8.10.3.8 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 3928 行目に定義がります。

呼び出し関係図:



8.10.3.9 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

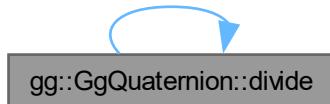
x	割る四元数の x 要素.
y	割る四元数の y 要素.
z	割る四元数の z 要素.
w	割る四元数の w 要素.

戻り値

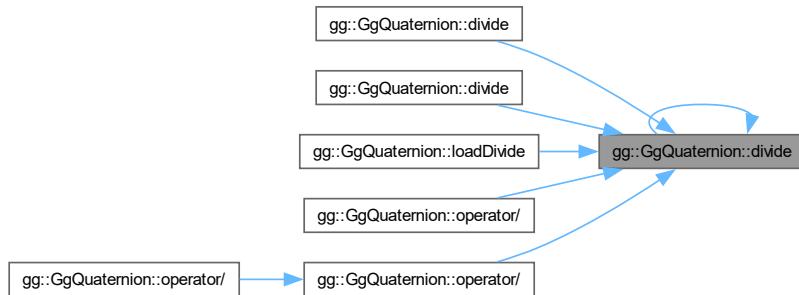
(x, y, z, w) を割った四元数.

gg.h の 3916 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.10 euler() [1/3]

```
GgQuaternion gg::GgQuaternion::euler (
    const GgVector & 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

<code>e</code>	オイラー角を表す <code>GgVector</code> 型の変数 (heading, pitch, roll) の参照.
----------------	---

戻り値

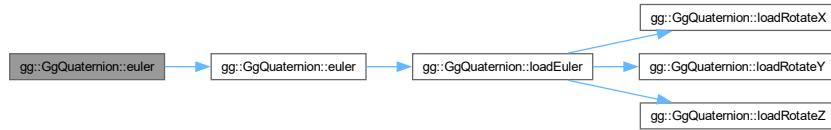
回転した四元数.

覚え書き

第 4 要素は無視する.

gg.h の 4281 行目に定義がります。

呼び出し関係図:



8.10.3.11 euler() [2/3]

```
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

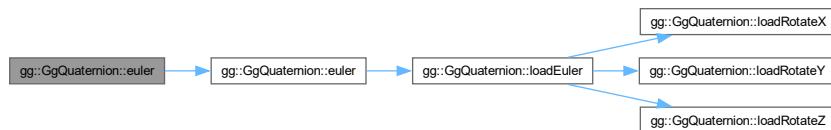
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転した四元数.

gg.h の 4267 行目に定義がります。

呼び出し関係図:



8.10.3.12 euler() [3/3]

```
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

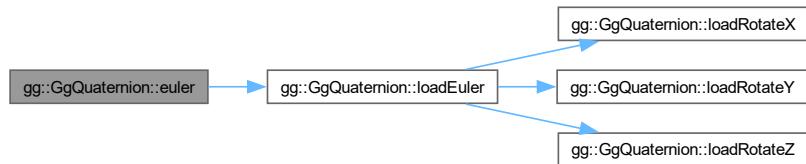
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

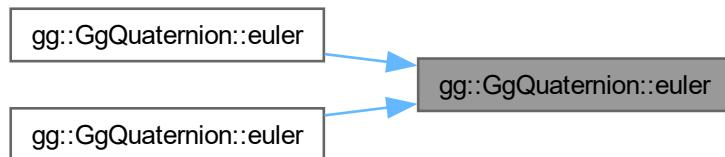
回転した四元数.

gg.h の 4255 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.13 get()

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

四元数を取り出す.

引数

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

gg.h の 4465 行目に定義があります。

8.10.3.14 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 4532 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.15 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 4522 行目に定義があります。

呼び出し関係図:



8.10.3.16 getConjugateMatrix() [3/3]

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

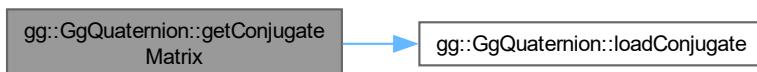
四元数の共役が表す回転の変換行列を *a* に求める。

引数

<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 4510 行目に定義があります。

呼び出し関係図:



8.10.3.17 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

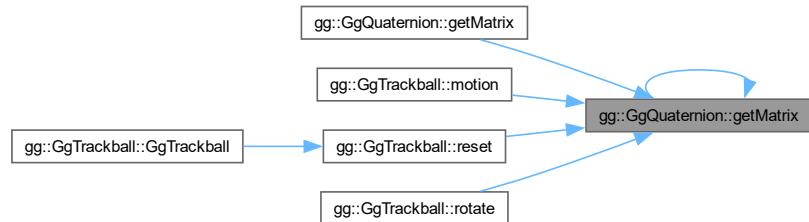
回転の変換を表す [GgMatrix](#) 型の変換行列.

[gg.h](#) の 4498 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.18 `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 4488 行目に定義があります。

呼び出し関係図:



8.10.3.19 getMatrix() [3/3]

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

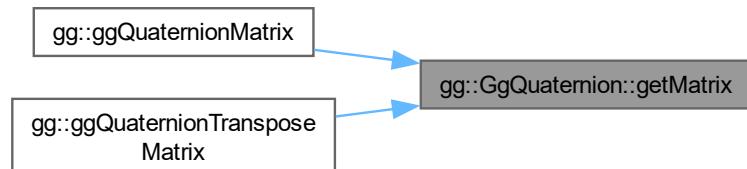
四元数が表す回転の変換行列を a に求める。

引数

a	回転の変換行列を格納する GLfloat 型の 16 要素の配列変数。
---	-------------------------------------

gg.h の 4478 行目に定義があります。

被呼び出し関係図:



8.10.3.20 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 4453 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.21 `loadAdd()` [1/4]

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

四元数に別の四元数を加算した結果を格納する.

引数

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

戻り値

`q` を加えた四元数.

`gg.h` の 3608 行目に定義があります。

呼び出し関係図:



8.10.3.22 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

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

戻り値

v を加えた四元数。

gg.h の 3597 行目に定義があります。

呼び出し関係図:



8.10.3.23 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

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

戻り値

a を加えた四元数.

[gg.h](#) の 3586 行目に定義がります。

呼び出し関係図:



8.10.3.24 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

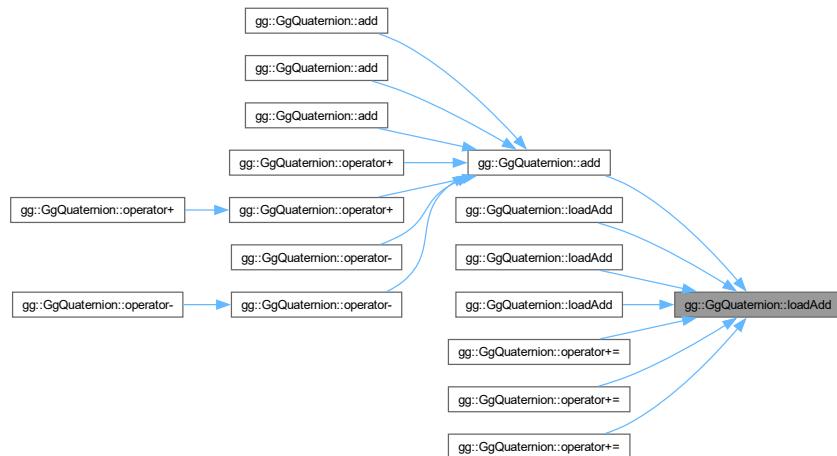
x	加える四元数の x 要素.
y	加える四元数の y 要素.
z	加える四元数の z 要素.
w	加える四元数の w 要素.

戻り値

(x, y, z, w) を加えた四元数.

[gg.h](#) の 3570 行目に定義がります。

被呼び出し関係図:



8.10.3.25 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

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

戻り値

共役四元数。

gg.h の 4372 行目に定義があります。

呼び出し関係図:



8.10.3.26 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

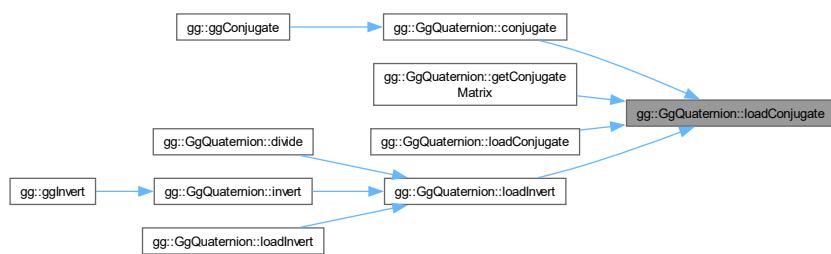
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

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:



8.10.3.27 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

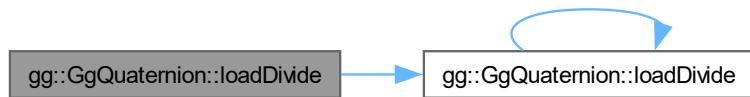
q	GgQuaternion 型の四元数.
---	---------------------

戻り値

\mathbf{q} で割った四元数.

gg.h の 3756 行目に定義があります。

呼び出し関係図:



8.10.3.28 loadDivide() [2/4]

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

四元を別の四元数で除算した結果を格納する.

引数

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

戻り値

\mathbf{v} で割った四元数.

gg.h の 3745 行目に定義があります。

呼び出し関係図:



8.10.3.29 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 3734 行目に定義があります。

呼び出し関係図:



8.10.3.30 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

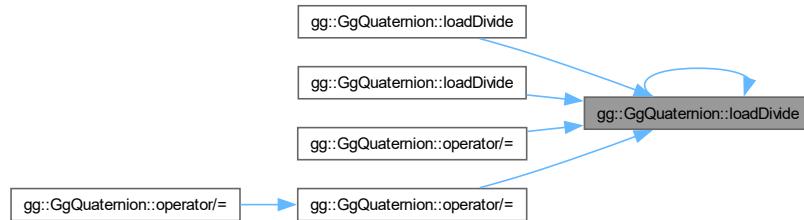
(x, y, z, w) を割った四元数。

gg.h の 3722 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.31 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

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

戻り値

格納した回転を表す四元数。

gg.h の 4242 行目に定義があります。

呼び出し関係図:



8.10.3.32 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

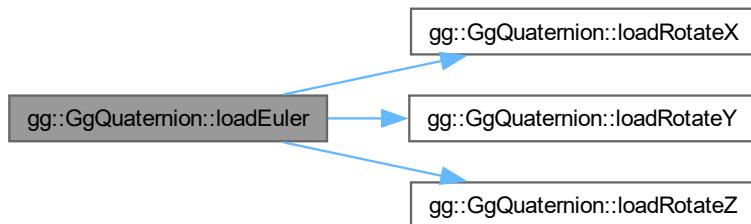
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

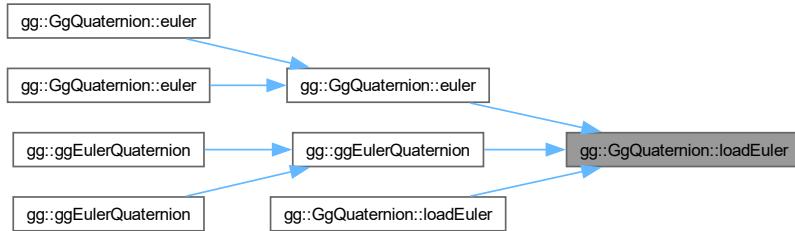
格納した回転を表す四元数。

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:



8.10.3.33 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 4092 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.34 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 4391 行目に定義があります。

呼び出し関係図:



8.10.3.35 `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

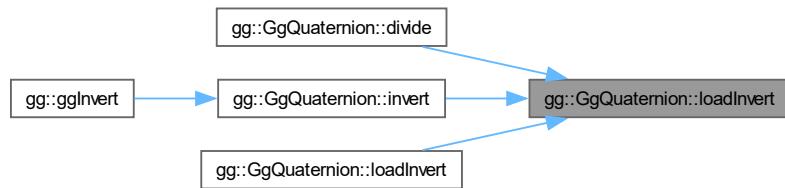
四元数の逆元.

`gg.cpp` の 3392 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



8.10.3.36 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

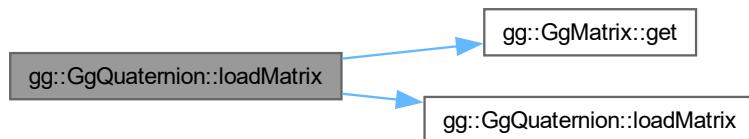
m	Ggmatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4082 行目に定義があります。

呼び出し関係図:



8.10.3.37 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

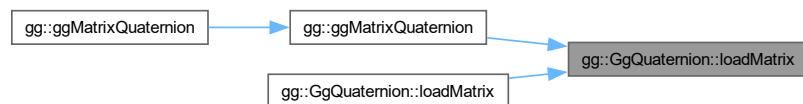
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 4070 行目に定義があります。

呼び出し関係図:



8.10.3.38 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 3708 行目に定義があります。

呼び出し関係図:



8.10.3.39 loadMultiply() [2/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

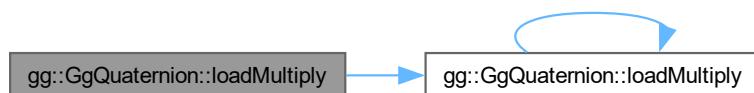
v	四元数を格納した GgVector 型の変数.
----------	-------------------------

戻り値

v を乗じた四元数.

gg.h の 3697 行目に定義があります。

呼び出し関係図:



8.10.3.40 loadMultiply() [3/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

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

戻り値

a を乗じた四元数.

gg.h の 3686 行目に定義があります。

8.10.3.41 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

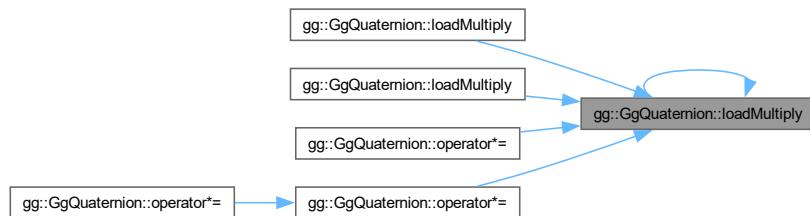
(*x, y, z, w*) を掛けた四元数.

gg.h の 3674 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.42 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 4353 行目に定義がります。

呼び出し関係図:



8.10.3.43 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

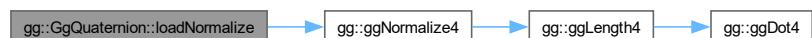
<code>a</code>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------------	--------------------------------

戻り値

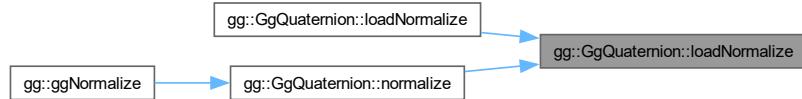
正規化された四元数.

gg.cpp の 3366 行目に定義がります。

呼び出し関係図:



呼び出し関係図:



8.10.3.44 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

v	軸ベクトルと回転角を格納した GLfloat 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 4126 行目に定義があります。

呼び出し関係図:



8.10.3.45 loadRotate() [2/3]

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

(v[0], v[1], v[2]) を軸として角度 a 回転する四元数を格納する。

引数

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

戻り値

格納された回転を表す四元数.

gg.h の 4115 行目に定義があります。

呼び出し関係図:



8.10.3.46 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

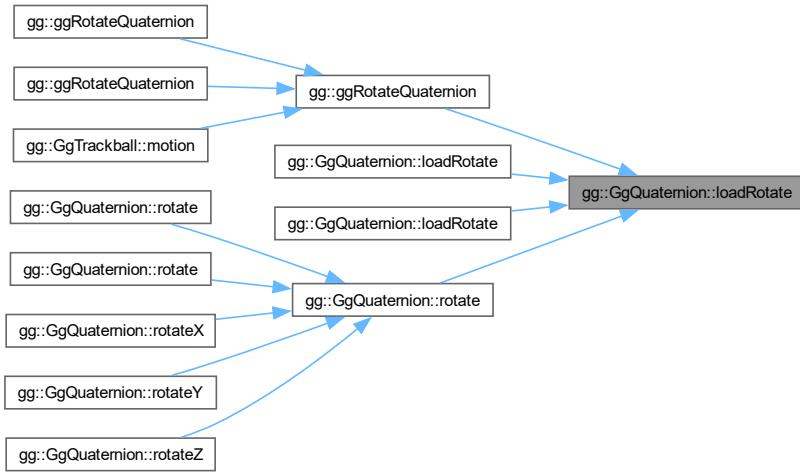
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

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:



8.10.3.47 `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

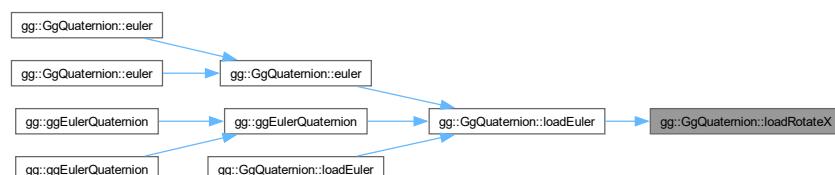
a	回転角。
-----	------

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:



8.10.3.48 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

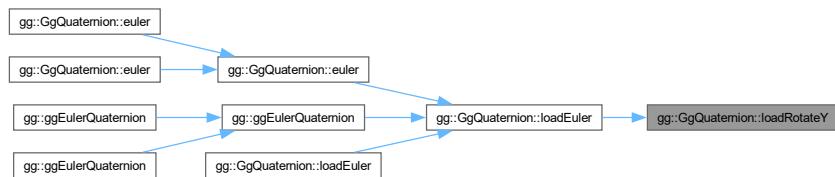
a	回転角.
----------	------

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:



8.10.3.49 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

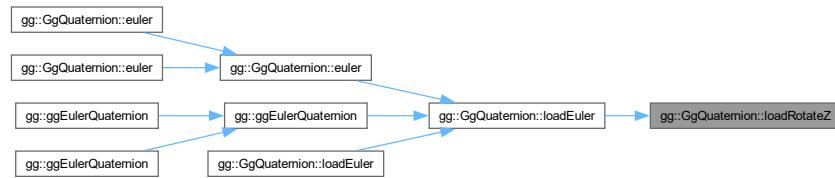
a	回転角.
----------	------

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。

被呼び出し関係図:



8.10.3.50 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

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

戻り値

格納した *q*, *r* を *t* で内分した四元数。

gg.h の 4308 行目に定義があります。

呼び出し関係図:



8.10.3.51 loadSlerp() [2/4]

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

球面線形補間の結果を格納する。

引数

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

戻り値

格納した *q*, *a* を *t* で内分した四元数.

gg.h の 4321 行目に定義があります。

呼び出し関係図:



8.10.3.52 loadSlerp() [3/4]

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

球面線形補間の結果を格納する.

引数

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

戻り値

格納した *a*, *q* を *t* で内分した四元数.

gg.h の 4334 行目に定義があります。

呼び出し関係図:



8.10.3.53 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

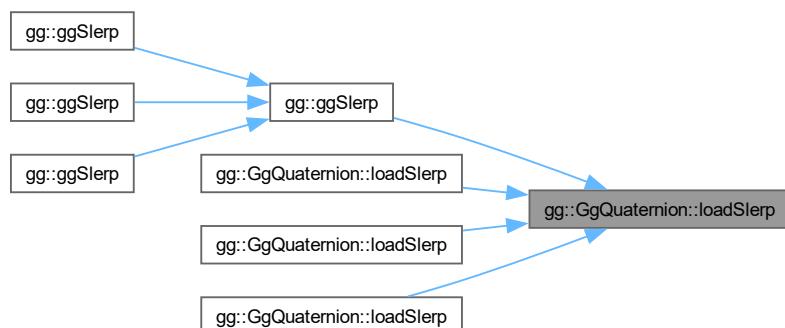
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
<i>t</i>	補間パラメータ。

戻り値

格納した *a*, *b* を *t* で内分した四元数。

gg.h の 4294 行目に定義があります。

被呼び出し関係図:



8.10.3.54 loadSubtract() [1/4]

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

四元数から別の四元数を減算した結果を格納する.

引数

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

戻り値

`q` を引いた四元数.

`gg.h` の 3660 行目に定義がります。

呼び出し関係図:



8.10.3.55 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

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

戻り値

`v` を引いた四元数.

`gg.h` の 3649 行目に定義がります。

呼び出し関係図:



8.10.3.56 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 3638 行目に定義があります。

呼び出し関係図:



8.10.3.57 loadSubtract() [4/4]

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

四元数から別の四元数を減算した結果を格納する。

引数

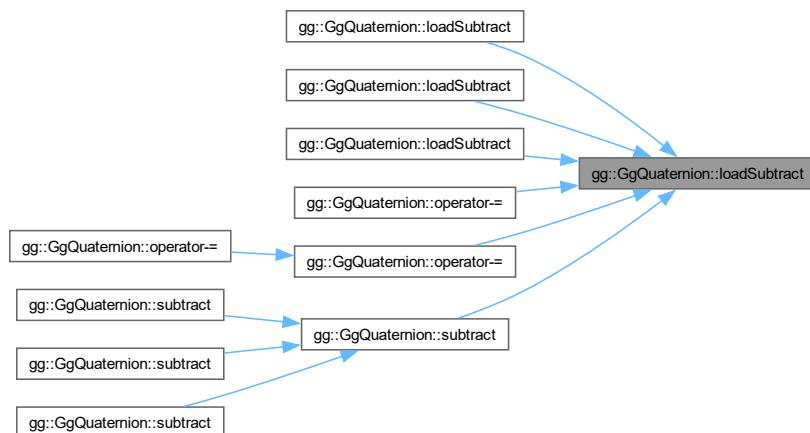
<i>x</i>	引く四元数の <i>x</i> 要素.
<i>y</i>	引く四元数の <i>y</i> 要素.
<i>z</i>	引く四元数の <i>z</i> 要素.
<i>w</i>	引く四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 3622 行目に定義があります。

被呼び出し関係図:



8.10.3.58 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 3902 行目に定義があります。

8.10.3.59 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 3891 行目に定義があります。

8.10.3.60 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 3878 行目に定義があります。

8.10.3.61 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 3866 行目に定義があります。

8.10.3.62 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 3556 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.63 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 4429 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.64 `operator*()` [1/3]

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

`gg.h` の 4047 行目に定義があります。

呼び出し関係図:



8.10.3.65 `operator*()` [2/3]

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

`gg.h` の 4043 行目に定義があります。

8.10.3.66 operator*() [3/3]

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

gg.h の 4039 行目に定義があります。

呼び出し関係図:



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

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

gg.h の 3999 行目に定義があります。

呼び出し関係図:

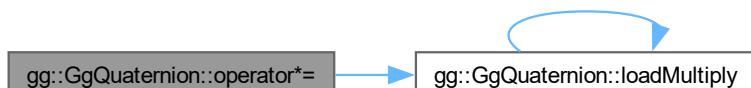


8.10.3.68 operator*=(()) [2/3]

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

gg.h の 3995 行目に定義があります。

呼び出し関係図:

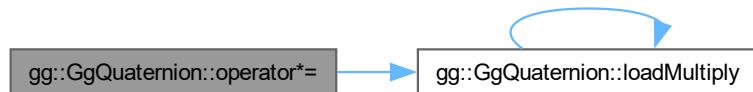


8.10.3.69 operator*=() [3/3]

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

gg.h の 3991 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

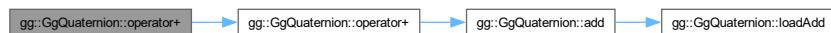


8.10.3.70 operator+() [1/3]

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

gg.h の 4023 行目に定義があります。

呼び出し関係図:



8.10.3.71 operator+() [2/3]

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

gg.h の 4019 行目に定義があります。

呼び出し関係図:



8.10.3.72 operator+() [3/3]

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

gg.h の 4015 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.73 operator+=() [1/3]

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

gg.h の 3975 行目に定義があります。

呼び出し関係図:



8.10.3.74 operator+=() [2/3]

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

gg.h の 3971 行目に定義があります。

呼び出し関係図:



8.10.3.75 operator+=() [3/3]

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

gg.h の 3967 行目に定義があります。

呼び出し関係図:



8.10.3.76 operator-() [1/3]

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

gg.h の 4035 行目に定義があります。

呼び出し関係図:

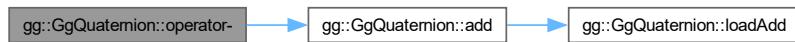


8.10.3.77 operator-() [2/3]

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

gg.h の 4031 行目に定義があります。

呼び出し関係図:

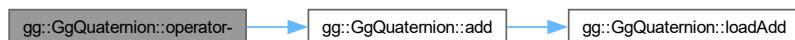


8.10.3.78 operator-() [3/3]

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

gg.h の 4027 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

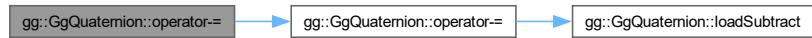


8.10.3.79 operator=() [1/3]

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

[gg.h](#) の 3987 行目に定義がります。

呼び出し関係図:



8.10.3.80 operator=() [2/3]

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

[gg.h](#) の 3983 行目に定義がります。

呼び出し関係図:



8.10.3.81 operator=() [3/3]

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

[gg.h](#) の 3979 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:

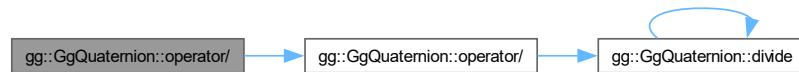


8.10.3.82 operator/() [1/3]

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

gg.h の 4059 行目に定義があります。

呼び出し関係図:



8.10.3.83 operator/() [2/3]

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

gg.h の 4055 行目に定義があります。

呼び出し関係図:



8.10.3.84 operator/() [3/3]

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

gg.h の 4051 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.85 operator/() [1/3]

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

gg.h の 4011 行目に定義があります。

呼び出し関係図:



8.10.3.86 operator/() [2/3]

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

gg.h の 4007 行目に定義があります。

呼び出し関係図:

**8.10.3.87 operator/() [3/3]**

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

gg.h の 4003 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

**8.10.3.88 operator=() [1/4]**

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

代入演算子.

引数

<code>q</code>	代入元の四元数。
----------------	----------

戻り値

代入後のこの四元数の参照。

被呼び出し関係図:



8.10.3.89 operator=() [2/4]

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

gg.h の 3963 行目に定義があります。

呼び出し関係図:



8.10.3.90 operator=() [3/4]

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

gg.h の 3959 行目に定義があります。

呼び出し関係図:



8.10.3.91 operator=() [4/4]

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

ムーブ代入演算子.

引数

<i>q</i>	ムーブ代入元の四元数.
----------	-------------

戻り値

ムーブ代入後のこの四元数の参照.

8.10.3.92 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

<i>v</i>	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 4188 行目に定義があります。

呼び出し関係図:



8.10.3.93 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

<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

回転した四元数.

gg.h の 4177 行目に定義があります。

呼び出し関係図:



8.10.3.94 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

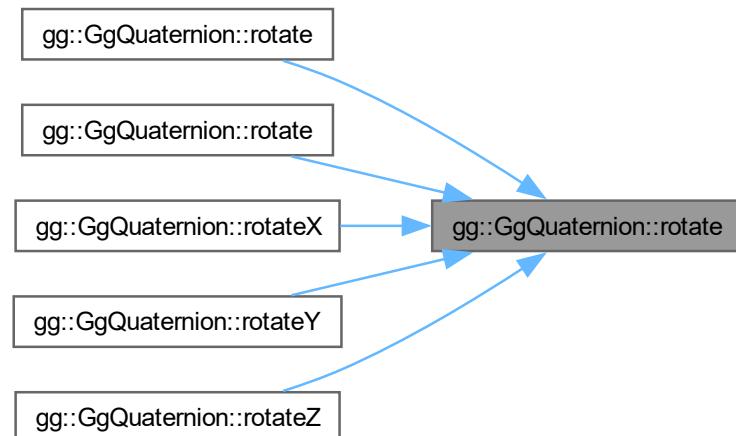
回転した四元数.

gg.h の 4164 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.10.3.95 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

<i>a</i>	回転角.
----------	------

戻り値

回転した四元数.

gg.h の 4199 行目に定義があります。

呼び出し関係図:



8.10.3.96 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 4210 行目に定義があります。

呼び出し関係図:



8.10.3.97 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

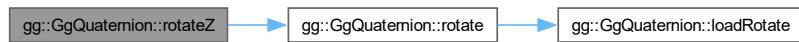
a	回転角.
----------	------

戻り値

回転した四元数.

[gg.h](#) の 4221 行目に定義があります。

呼び出し関係図:



8.10.3.98 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 4417 行目に定義があります。

8.10.3.99 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 4403 行目に定義があります。

8.10.3.100 `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

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

戻り値

q を引いた四元数.

[gg.h](#) の 3852 行目に定義があります。

呼び出し関係図:



8.10.3.101 `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

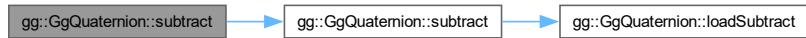
<code>v</code>	四元数を格納した <code>GgVector</code> 型の変数.
----------------	--------------------------------------

戻り値

v を引いた四元数.

[gg.h](#) の 3841 行目に定義があります。

呼び出し関係図:



8.10.3.102 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 3830 行目に定義があります。

呼び出し関係図:



8.10.3.103 subtract() [4/4]

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

四元数から別の四元数を減算した結果を返す.

引数

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

構造: [Doxygen](#)

戻り値

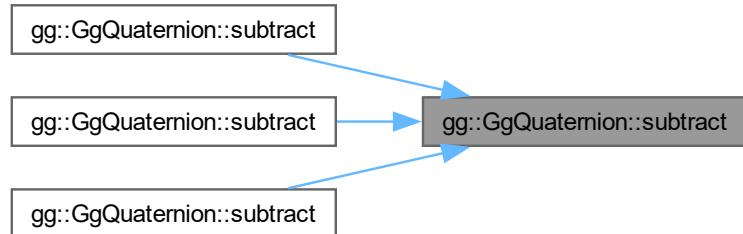
(x, y, z, w) を引いた四元数.

gg.h の 3818 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](#)

8.11 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](#)

8.11.1 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 6864 行目](#)に定義があります。

8.11.2 構築子と解体子

8.11.2.1 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

<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 6880 行目](#)に定義があります。

呼び出し関係図:



8.11.2.2 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 6898 行目に定義がります。

8.11.2.3 GgShader() [3/4]

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

コピーコンストラクタは使用しない。

引数

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

8.11.2.4 GgShader() [4/4]

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

ムーブコンストラクタ.

引数

<i>shader</i>	ムーブ元のシェーダ.
---------------	------------

8.11.2.5 ~GgShader()

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

デストラクタ.

gg.h の 6924 行目に定義がります。

8.11.3 関数詳解

8.11.3.1 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 6968 行目に定義があります。

8.11.3.2 operator=() [1/2]

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

代入演算子は使用しない.

引数

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

戻り値

代入後のこのシェーダの参照.

8.11.3.3 operator=() [2/2]

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

ムーブ代入演算子.

引数

<i>shader</i>	ムーブ代入元のシェーダ.
---------------	--------------

戻り値

ムーブ代入後のこのシェーダの参照.

8.11.3.4 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 6958 行目に定義があります。

8.11.3.5 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 6950 行目に定義があります。

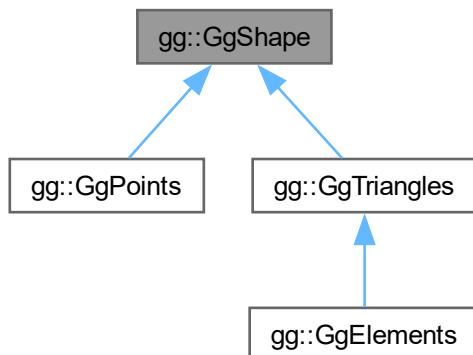
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.12 gg::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](#)

8.12.1 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 6240 行目に定義があります。

8.12.2 構築子と解体子

8.12.2.1 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 6255 行目に定義があります。

8.12.2.2 ~GgShape()

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

デストラクタ.

[gg.h](#) の 6264 行目に定義があります。

8.12.3 関数詳解

8.12.3.1 draw()

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

図形の描画, 派生クラスでこの手続きをオーバーライドする.

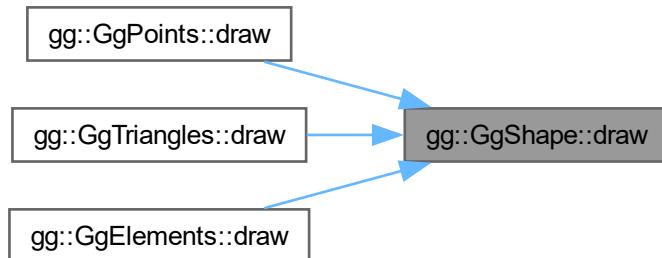
引数

<i>first</i>	描画する最初のアイテム.
<i>count</i>	描画するアイテムの数, 0 なら全部のアイテムを描画する.

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

`gg.h` の 6324 行目に定義があります。

被呼び出し関係図:



8.12.3.2 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 6293 行目に定義があります。

被呼び出し関係図:



8.12.3.3 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 6313 行目に定義があります。

8.12.3.4 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 6273 行目に定義があります。

8.12.3.5 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 6283 行目に定義があります。

8.12.3.6 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 6303 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::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`

8.13.1 詳解

Wavefront OBJ 形式のファイル (Arrays 形式)。

gg.h の 8244 行目に定義があります。

8.13.2 構築子と解体子

8.13.2.1 GgSimpleObj()

```
gg::GgSimpleObj::GgSimpleObj (
    const std::string & name,
    bool normalize = false )
```

コンストラクタ。

引数

<code>name</code>	三角形分割された Alias OBJ 形式のファイルのファイル名。
<code>normalize</code>	true なら図形のサイズを [-1, 1] に正規化する。

gg.cpp の 5984 行目に定義があります。

呼び出し関係図:



8.13.2.2 ~GgSimpleObj()

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

デストラクタ。

gg.h の 8267 行目に定義があります。

8.13.3 関数詳解

8.13.3.1 draw()

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

Wavefront OBJ 形式のデータを描画する手続き.

引数

<i>first</i>	描画する最初のverts番号.
<i>count</i>	描画するvertsの数, 0なら全部のvertsを描く.

gg.cpp の 6012 行目に定義があります。

8.13.3.2 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 8296 行目に定義があります。

呼び出し関係図:



8.13.3.3 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 8276 行目に定義があります。

8.13.3.4 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 8286 行目に定義があります。

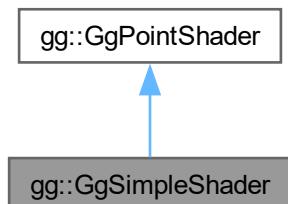
このクラス詳解は次のファイルから抽出されました:

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

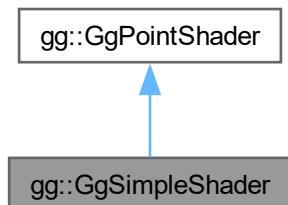
8.14 gg::GgSimpleShader クラス

```
#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

8.14.1 詳解

三角形に単純な陰影付けを行うシェーダ.

`gg.h` の 7232 行目に定義があります。

8.14.2 構築子と解体子

8.14.2.1 GgSimpleShader() [1/4]

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

コンストラクタ.

`gg.h` の 7243 行目に定義があります。

8.14.2.2 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)
```

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 7258 行目に定義があります。

8.14.2.3 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 7276 行目に定義があります。

8.14.2.4 GgSimpleShader() [4/4]

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

コピー コンストラクタ.

[gg.h](#) の 7288 行目に定義があります。

8.14.2.5 ~GgSimpleShader()

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

デストラクタ.

[gg.h](#) の 7297 行目に定義があります。

8.14.3 関数詳解

8.14.3.1 load() [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.

引数

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

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 7344 行目に定義があります。

8.14.3.2 load() [2/2]

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

シェーダのソースプログラムの文字列からプログラムオブジェクトを作成する。

引数

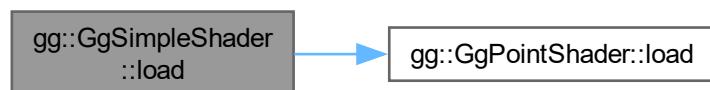
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列。
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列（空文字列なら不使用）。
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列（空文字列なら不使用）。
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用)。

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.cpp の 5967 行目に定義があります。

呼び出し関係図:



8.14.3.3 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> 型のモデルビューチェンジ行列.

`gg::GgPointShader`を再実装しています。

`gg.h` の 7438 行目に定義があります。

呼び出し関係図:



8.14.3.4 `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チェンジ行列と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> 型のモデルビューチェンジ行列.
<i>mn</i>	<code>GgMatrix</code> 型のモデルビューチェンジ行列の法線変換行列.

`gg.h` の 7416 行目に定義があります。

呼び出し関係図:



8.14.3.5 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 7427 行目に定義があります。

8.14.3.6 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 7403 行目に定義があります。

8.14.3.7 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 7391 行目に定義があります。

呼び出し関係図:



8.14.3.8 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 7371 行目に定義があります。

呼び出し関係図:



8.14.3.9 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 7381 行目に定義があります。

8.14.3.10 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 7359 行目に定義があります。

8.14.3.11 operator=(*)

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

代入演算子。

引数

<i>o</i>	代入する GgSimpleShader 型のオブジェクト.
----------	---

戻り値

代入後の [GgSimpleShader](#) 型のオブジェクトの参照。

[gg.h](#) の 7307 行目に定義があります。

8.14.3.12 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 8011 行目に定義があります。

8.14.3.13 `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>	<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.h` の 8062 行目に定義がります。

呼び出し関係図:



8.14.3.14 `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>	<code>GgMatrix</code> 型の投影変換行列.
<i>mv</i>	<code>GgMatrix</code> 型のモデルビュー変換行列.
<i>mn</i>	<code>GgMatrix</code> 型のモデルビュー変換行列の法線変換行列.

`gg.h` の 8040 行目に定義がります。

呼び出し関係図:



8.14.3.15 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 8126 行目に定義があります。

呼び出し関係図:



8.14.3.16 use() [5/13]

```

void gg::GgSimpleShader::use (
    const GgMatrix & mp,
  
```

```
const GgMatrix & mp,
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 8163 行目に定義があります。

呼び出し関係図:



8.14.3.17 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 8196 行目に定義があります。

呼び出し関係図:



8.14.3.18 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 8051 行目に定義がります。

8.14.3.19 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 8024 行目に定義がります。

8.14.3.20 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 8102 行目に定義があります。

8.14.3.21 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 8145 行目に定義があります。

8.14.3.22 use() [11/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>light</i>	光源の特性の gg::LightBuffer 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 8180 行目に定義があります。

8.14.3.23 use() [12/13]

```
void gg::GgSimpleShader::use (
    const LightBuffer & light,
    GLint i = 0 ) const [inline]
```

光源を指定してシェーダプログラムの使用を開始する。

引数

<i>light</i>	光源の特性の gg::LightBuffer 構造体.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 8088 行目に定義があります。

8.14.3.24 use() [13/13]

```
void gg::GgSimpleShader::use (
    const LightBuffer * light,
    GLint i = 0 ) const [inline]
```

光源を指定してシェーダプログラムの使用を開始する。

引数

<i>light</i>	光源の特性の gg::LightBuffer 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 8073 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](#)

8.15 gg::GgTexture クラス

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

公開メンバ関数

- [GgTexture \(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE, bool swizzle=false\)](#)
- [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 GLsizei * getSize \(\) const](#)
- [const GLuint & getTexture \(\) const](#)

8.15.1 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 5193 行目に定義があります。

8.15.2 構築子と解体子

8.15.2.1 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 = false ) [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>false</code> .

`gg.h` の 5215 行目に定義があります。

8.15.2.2 GgTexture() [2/3]

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

コピーコンストラクタは使用しない。

引数

<i>texture</i>	コピー元のテクスチャ.
----------------	-------------

8.15.2.3 GgTexture() [3/3]

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

ムーブコンストラクタ。

引数

<i>texture</i>	ムーブ元のテクスチャ。
----------------	-------------

8.15.2.4 ~GgTexture()

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

デストラクタ。

[gg.h](#) の 5247 行目に定義があります。

8.15.3 関数詳解

8.15.3.1 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 5272 行目に定義があります。

8.15.3.2 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 5307 行目に定義があります。

被呼び出し関係図:



8.15.3.3 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 5328 行目に定義があります。

8.15.3.4 getSize() [2/2]

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

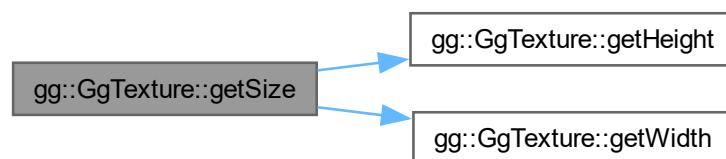
使用しているテクスチャのサイズを取り出す。

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 5317 行目に定義があります。

呼び出し関係図:



8.15.3.5 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 5338 行目に定義があります。

8.15.3.6 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 5297 行目に定義があります。

被呼び出し関係図:



8.15.3.7 operator=() [1/2]

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

代入演算子は使用しない。

引数

<i>texture</i>	代入元のテクスチャ。
----------------	------------

戻り値

代入後のこのテクスチャの参照。

8.15.3.8 operator=() [2/2]

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

ムーブ代入演算子。

引数

<i>texture</i>	ムーブ代入元のテクスチャ.
----------------	---------------

戻り値

ムーブ代入後のこのテクスチャの参照.

8.15.3.9 swapRandB()

```
void gg::GgTexture::swapRandB ( bool swizzle ) const
```

テクスチャの赤と青を交換する

引数

<i>swizzle</i>	赤と青を交換するなら true
----------------	-----------------

gg.cpp の 4000 行目に定義があります。

8.15.3.10 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 5280 行目に定義があります。

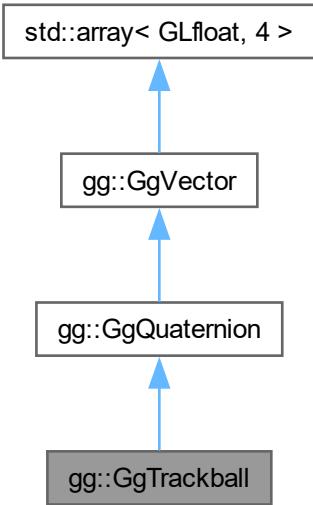
このクラス詳解は次のファイルから抽出されました:

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

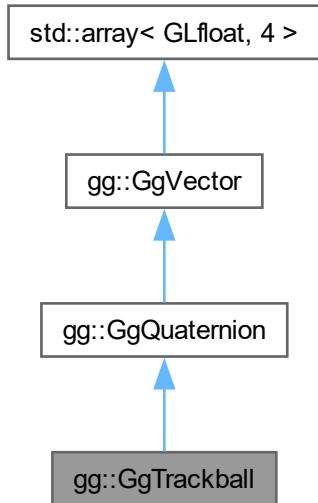
8.16 gg::GgTrackball クラス

```
#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=ggetIdentityQuaternion())`
- `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 euler (const GgVector &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`

8.16.1 詳解

簡易トラックボール処理。

`gg.h` の 4775 行目に定義があります。

8.16.2 構築子と解体子

8.16.2.1 `GgTrackball()`

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

コンストラクタ。

引数

<code>q</code>	トラックボールの回転の初期値の四元数。
----------------	---------------------

`gg.h` の 4790 行目に定義があります。

呼び出し関係図:



8.16.2.2 `~GgTrackball()`

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

デストラクタ。

8.16.3 関数詳解

8.16.3.1 begin()

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

トラックボール処理を開始する。

引数

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

覚え書き

マウスのドラッグ開始時(マウスボタンを押したとき)に呼び出す。

[gg.cpp](#) の 3447 行目に定義があります。

8.16.3.2 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。

8.16.3.3 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 4968 行目に定義があります。

呼び出し関係図:



8.16.3.4 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 4958 行目に定義があります。

8.16.3.5 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 4948 行目に定義があります。

8.16.3.6 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 4917 行目に定義があります。

8.16.3.7 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

<code>factor</code>	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 4937 行目に定義がります。

8.16.3.8 `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 4927 行目に定義がります。

8.16.3.9 `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 4888 行目に定義がります。

8.16.3.10 `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 4906 行目に定義がります。

8.16.3.11 `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 4896 行目に定義があります。

8.16.3.12 motion()

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

回転の変換行列を計算する。

引数

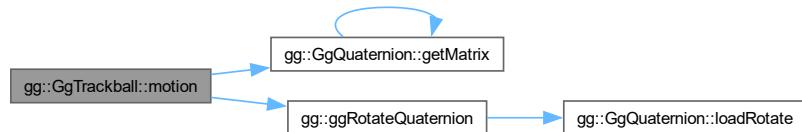
<i>x</i>	現在のマウスの x 座標.
<i>y</i>	現在のマウスの y 座標.

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:



8.16.3.13 operator=()

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

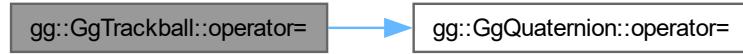
代入。

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 4805 行目に定義があります。

呼び出し関係図:



8.16.3.15 region() [2/2]

```
void gg::GgTrackball::region (
    int w,
    int h ) [inline]
```

トラックボール処理するマウスの移動範囲を指定する。

引数

w	領域の横幅.
h	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す.

gg.h の 4831 行目に定義があります。

呼び出し関係図:



8.16.3.16 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

q	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:



8.16.3.17 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:



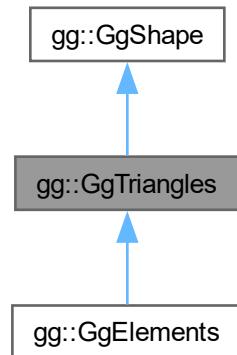
このクラス詳解は次のファイルから抽出されました:

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

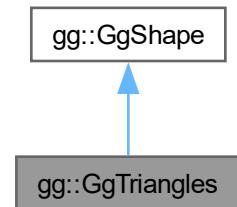
8.17 gg::GgTriangles クラス

```
#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`

8.17.1 詳解

三角形で表した形状データ (Arrays 形式).

`gg.h` の 6509 行目に定義があります。

8.17.2 構築子と解体子

8.17.2.1 GgTriangles() [1/2]

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

コンストラクタ.

引数

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

`gg.h` の 6522 行目に定義があります。

8.17.2.2 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

<code>vert</code>	この図形の頂点属性の配列 (<code>nullptr</code> ならデータを転送しない).
<code>count</code>	頂点数.
<code>mode</code>	描画する基本図形の種類.
<code>usage</code>	バッファオブジェクトの使い方.

`gg.h` の 6535 行目に定義があります。

8.17.2.3 ~GgTriangles()

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

デストラクタ。

[gg.h](#) の 6549 行目に定義があります。

8.17.3 関数詳解

8.17.3.1 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 5151 行目に定義があります。

呼び出し関係図:



8.17.3.2 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

この图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 6568 行目に定義があります。

8.17.3.3 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 6558 行目に定義があります。

8.17.3.4 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 5132 行目に定義があります。

8.17.3.5 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 6580 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](#)

8.18 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](#)

8.18.1 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 5827 行目に定義があります。

8.18.2 構築子と解体子

8.18.2.1 GgUniformBuffer() [1/3]

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

コンストラクタ。

[gg.h](#) の 5830 行目に定義があります。

8.18.2.2 GgUniformBuffer() [2/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer (
    const T * data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトのブロックごとにデータを転送するコンストラクタ。

引数

<i>data</i>	データが格納されている領域の先頭のポインタ (nullptr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 5830 行目に定義があります。

8.18.2.3 GgUniformBuffer() [3/3]

```
template<typename T >
gg::GgUniformBuffer< T >::GgUniformBuffer (
    const T & data,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

ユニフォームバッファオブジェクトの全ブロックに同じデータを格納するコンストラクタ。

引数

<i>data</i>	格納するデータ.
<i>count</i>	格納する数.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 5830 行目に定義があります。

8.18.2.4 ~GgUniformBuffer()

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

デストラクタ。

gg.h の 5830 行目に定義があります。

8.18.3 関数詳解

8.18.3.1 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 5915 行目に定義があります。

8.18.3.2 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 6126 行目に定義があります。

8.18.3.3 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と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 6043 行目に定義がります。

8.18.3.4 `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 5907 行目に定義がります。

8.18.3.5 `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 5897 行目に定義がります。

8.18.3.6 `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 5887 行目に定義がります。

8.18.3.7 `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 5877 行目に定義がります。

8.18.3.8 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 5984 行目に定義があります。

8.18.3.9 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 (nullptr 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 5965 行目に定義があります。

8.18.3.10 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 5933 行目に定義があります。

8.18.3.11 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 5945 行目に定義があります。

8.18.3.12 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 6078 行目に定義があります。

8.18.3.13 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 6005 行目に定義があります。

8.18.3.14 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 5923 行目に定義があります。

8.18.3.15 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 5953 行目に定義があります。

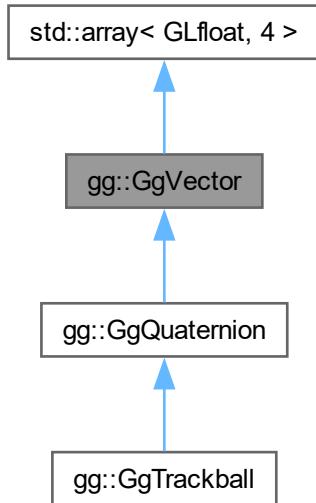
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

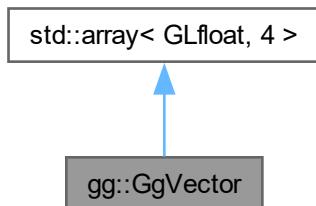
8.19 gg::GgVector クラス

```
#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`

8.19.1 詳解

単精度実数の 4 要素のベクトル。

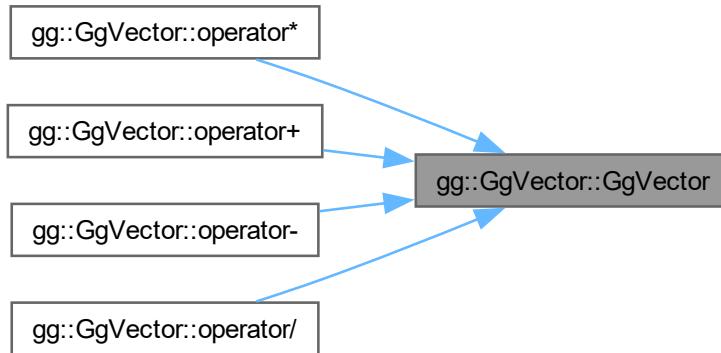
gg.h の 1571 行目に定義があります。

8.19.2 構築子と解体子

8.19.2.1 GgVector() [1/6]

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

コンストラクタ、被呼び出し関係図:



8.19.2.2 GgVector() [2/6]

```
constexpr 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。

8.19.2.3 GgVector() [3/6]

```
constexpr 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。

8.19.2.4 GgVector() [4/6]

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

コンストラクタ.

引数

c	GLfloat 型の値.
---	--------------

gg.h の 1608 行目に定義があります。

8.19.2.5 GgVector() [5/6]

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

コピーコンストラクタ.

引数

v	コピー元のベクトル.
---	------------

8.19.2.6 GgVector() [6/6]

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

ムーブコンストラクタ.

引数

v	ムーブ元のベクトル.
---	------------

8.19.2.7 ~GgVector()

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

デストラクタ.

8.19.3 関数詳解

8.19.3.1 distance3()

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

GgVector 型の 3 要素の距離.

引数

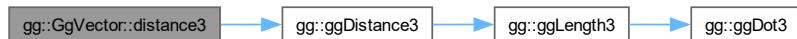
v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v の 3 要素の距離.

gg.h の 1852 行目に定義があります。

呼び出し関係図:



8.19.3.2 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 1896 行目に定義があります。

呼び出し関係図:



8.19.3.3 dot3()

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

GgVector 型の 3 要素の内積.

引数

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

戻り値

オブジェクトと v のそれぞれの 3 要素の内積.

gg.h の 1831 行目に定義があります。

呼び出し関係図:



8.19.3.4 dot4()

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

GgVector 型の 4 要素の内積.

引数

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

戻り値

オブジェクトと v のそれぞれの 4 要素の内積.

`gg.h` の 1875 行目に定義があります。

呼び出し関係図:



8.19.3.5 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 1841 行目に定義があります。

呼び出し関係図:



8.19.3.6 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 1885 行目に定義があります。

呼び出し関係図:



8.19.3.7 normalize3()

`GgVector gg::GgVector::normalize3 () const [inline]`

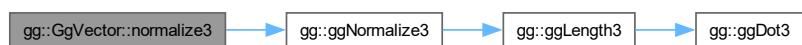
`GgVector` 型の 4 要素の正規化.

戻り値

`GLfloat` 型の 4 要素の配列変数.

gg.h の 1862 行目に定義があります。

呼び出し関係図:



8.19.3.8 normalize4()

`GgVector gg::GgVector::normalize4 () const [inline]`

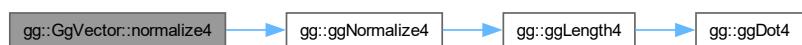
`GgVector` 型の 4 要素の正規化.

戻り値

`GLfloat` 型の 4 要素の配列変数.

gg.h の 1906 行目に定義があります。

呼び出し関係図:



8.19.3.9 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 1740 行目に定義があります。

8.19.3.10 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 1751 行目に定義があります。

呼び出し関係図:



8.19.3.11 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 1761 行目に定義があります。

8.19.3.12 `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 1773 行目に定義があります。

8.19.3.13 `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 1648 行目に定義があります。

8.19.3.14 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 1659 行目に定義があります。

呼び出し関係図:

**8.19.3.15 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 1670 行目に定義があります。

8.19.3.16 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

<i>c</i>	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 1682 行目に定義がります。

8.19.3.17 operator-() [1/2]

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

GgVector 型の差を返す.

引数

<i>v</i>	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 1694 行目に定義がります。

8.19.3.18 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

<i>c</i>	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 1705 行目に定義がります。

呼び出し関係図:



8.19.3.19 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 1716 行目に定義があります。

8.19.3.20 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 1728 行目に定義があります。

8.19.3.21 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 1785 行目に定義があります。

8.19.3.22 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

c	GLfloat 型の変数.
---	---------------

戻り値

オブジェクトの各要素を c で割ったオブジェクト.

gg.h の 1796 行目に定義があります。

呼び出し関係図:



8.19.3.23 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 1807 行目に定義があります。

8.19.3.24 `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>	<code>GLfloat</code> 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 1819 行目に定義があります。

8.19.3.25 `operator=()` [1/2]

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

代入演算子.

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

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

ムーブ代入演算子.

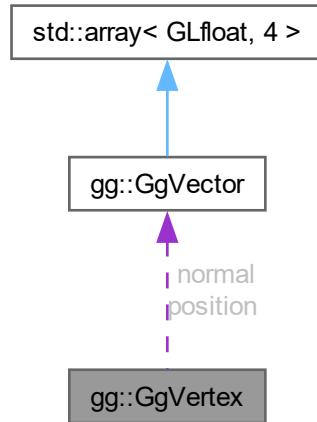
このクラス詳解は次のファイルから抽出されました:

- `gg.h`

8.20 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.

8.20.1 詳解

三角形の頂点データ.

gg.h の 6448 行目に定義があります。

8.20.2 構築子と解体子

8.20.2.1 GgVertex() [1/4]

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

コンストラクタ.

[gg.h](#) の 6459 行目に定義がります。

8.20.2.2 GgVertex() [2/4]

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

コンストラクタ.

引数

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

[gg.h](#) の 6469 行目に定義がります。

8.20.2.3 GgVertex() [3/4]

```
gg::GgVertex::GgVertex (
    GLfloat px,
    GLfloat py,
    GLfloat pz,
    GLfloat nx,
    GLfloat ny,
    GLfloat nz ) [inline]
```

コンストラクタ.

引数

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

[gg.h](#) の 6485 行目に定義がります。

8.20.2.4 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 GLfloat 型の位置データのポインタ.
<i>norm</i>	3 要素の GLfloat 型の法線データのポインタ.

gg.h の 6500 行目に定義があります。

8.20.3 メンバ詳解

8.20.3.1 normal

`GgVector gg::GgVertex::normal`

法線.

gg.h の 6454 行目に定義があります。

8.20.3.2 position

`GgVector gg::GgVertex::position`

位置.

gg.h の 6451 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`

8.21 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`

8.21.1 詳解

頂点配列 クラス.

[gg.h](#) の 6157 行目に定義があります。

8.21.2 構築子と解体子

8.21.2.1 GgVertexArray() [1/3]

```
gg::GgVertexArray::GgVertexArray (
    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 6169 行目に定義があります。

8.21.2.2 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.

引数

<i>array</i>	コピー元の頂点配列オブジェクト.
--------------	------------------

8.21.2.3 GgVertexArray() [3/3]

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

ムーブ コンストラクタ.

引数

<i>array</i>	ムーブ元の頂点配列オブジェクト.
--------------	------------------

8.21.2.4 ~GgVertexArray()

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

デストラクタ。

gg.h の 6192 行目に定義があります。

8.21.3 関数詳解

8.21.3.1 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 6227 行目に定義があります。

8.21.3.2 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 6219 行目に定義があります。

8.21.3.3 operator=() [1/2]

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

代入演算子は使用しない。

引数

<code>array</code>	代入元の頂点配列オブジェクト。
--------------------	-----------------

戻り値

代入後のこの頂点配列オブジェクトの参照。

8.21.3.4 operator=() [2/2]

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

ムーブ代入演算子.

引数

<i>array</i>	ムーブ代入元の頂点配列オブジェクト.
--------------	--------------------

戻り値

ムーブ代入後のこの頂点配列オブジェクトの参照.

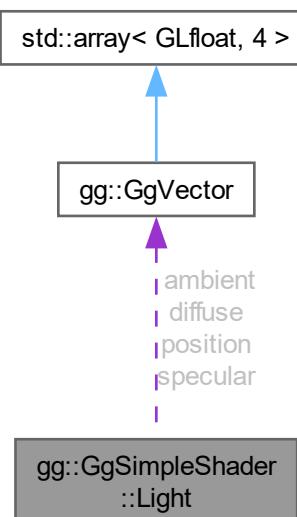
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.22 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.

8.22.1 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 7446 行目に定義があります。

8.22.2 メンバ詳解

8.22.2.1 ambient

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

光源強度の環境光成分.

gg.h の 7448 行目に定義があります。

8.22.2.2 diffuse

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

光源強度の拡散反射光成分.

gg.h の 7449 行目に定義があります。

8.22.2.3 position

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

光源の位置.

gg.h の 7451 行目に定義があります。

8.22.2.4 specular

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

光源強度の鏡面反射光成分。

[gg.h](#) の 7450 行目に定義があります。

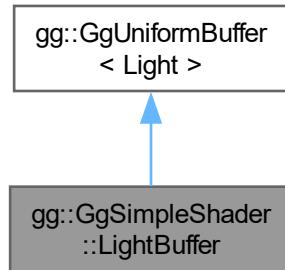
この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

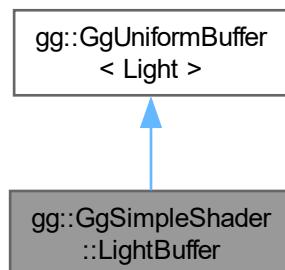
8.23 gg::GgSimpleShader::LightBuffer クラス

```
#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)
- LightBuffer (GgVector ambient, GgVector diffuse, GgVector specular, GgVector position, 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

8.23.1 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のユニフォームバッファオブジェクト。

gg.h の 7457 行目に定義があります。

8.23.2 構築子と解体子

8.23.2.1 LightBuffer() [1/3]

```
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>	GgSimpleShader::Light 型の光源データのポインタ。
<i>count</i>	バッファ中の GgSimpleShader::Light 型の光源データの数。
<i>usage</i>	バッファの使い方のパターン, glBufferData() の第 4 引数の <i>usage</i> に渡される。

gg.h の 7469 行目に定義がります。

8.23.2.2 LightBuffer() [2/3]

```
gg::GgSimpleShader::LightBuffer::LightBuffer (
    const Light & light,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じデータで埋めるコンストラクタ。

引数

<i>light</i>	GgSimpleShader::Light 型の光源データ。
<i>count</i>	バッファ中の GgSimpleShader::Light 型の光源データの数。
<i>usage</i>	バッファの使い方のパターン, glBufferData() の第 4 引数の <i>usage</i> に渡される。

gg.h の 7485 行目に定義がります。

8.23.2.3 LightBuffer() [3/3]

```
gg::GgSimpleShader::LightBuffer::LightBuffer (
    GgVector ambient,
    GgVector diffuse,
    GgVector specular,
    GgVector position,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じ引数で埋めるコンストラクタ。

引数

<i>ambient</i>	光源強度の環境光成分.
<i>diffuse</i>	光源強度の拡散反射光成分.
<i>specular</i>	光源強度の鏡面反射光成分.
<i>position</i>	光源の位置.
<i>count</i>	バッファ中の GgSimpleShader::Light 型の光源データの数.
<i>usage</i>	バッファの使い方のパターン, glBufferData() の第 4 引数の <i>usage</i> に渡される.

[gg.h](#) の 7504 行目に定義があります。

8.23.2.4 ~LightBuffer()

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

デストラクタ.

[gg.h](#) の 7519 行目に定義があります。

8.23.3 関数詳解

8.23.3.1 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 7711 行目に定義があります。

8.23.3.2 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 7699 行目に定義があります。

8.23.3.3 `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 5477 行目に定義があります。

8.23.3.4 `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 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 7554 行目に定義があります。

8.23.3.5 `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 5451 行目に定義がります。

8.23.3.6 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 5604 行目に定義がります。

8.23.3.7 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 <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.
構造体 <code>GgVector</code>	値を設定する光源データの数, デフォルトは 1.

[gg.cpp](#) の 5529 行目に定義がります。

8.23.3.8 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 7591 行目に定義がります。

8.23.3.9 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 5503 行目に定義がります。

8.23.3.10 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 5658 行目に定義がります。

8.23.3.11 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 7687 行目に定義がります。

8.23.3.12 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 7674 行目に定義がります。

8.23.3.13 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 5632 行目に定義がります。

8.23.3.14 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 5581 行目に定義がります。

8.23.3.15 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 7628 行目に定義があります。

8.23.3.16 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 5555 行目に定義があります。

8.23.3.17 select()

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

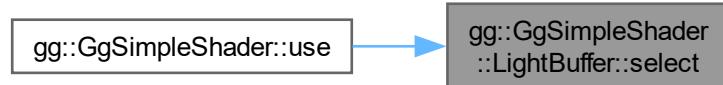
光源を選択する.

引数

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

[gg.h](#) の 7721 行目に定義があります。

被呼び出し関係図:



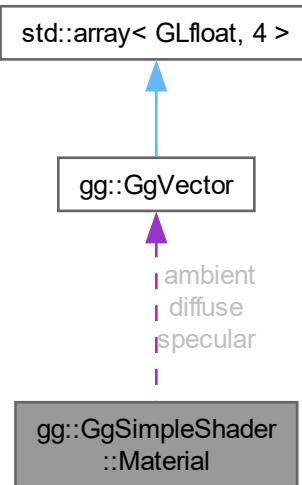
このクラス詳解は次のファイルから抽出されました:

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

8.24 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.

8.24.1 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材質データ.

gg.h の 7732 行目に定義があります。

8.24.2 メンバ詳解

8.24.2.1 ambient

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

環境光に対する反射係数.

gg.h の 7734 行目に定義があります。

8.24.2.2 diffuse

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

拡散反射係数.

gg.h の 7735 行目に定義があります。

8.24.2.3 shininess

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

輝き係数.

gg.h の 7737 行目に定義があります。

8.24.2.4 specular

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

鏡面反射係数。

[gg.h](#) の 7736 行目に定義があります。

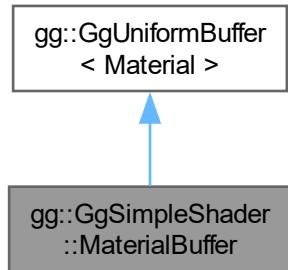
この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

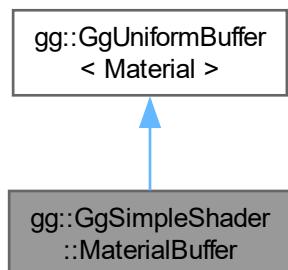
8.25 gg::GgSimpleShader::MaterialBuffer クラス

```
#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)`
- `MaterialBuffer (GgVector ambient, GgVector diffuse, GgVector specular, GLfloat shininess, 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`

8.25.1 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のユニフォームバッファオブジェクト。

`gg.h` の 7743 行目に定義があります。

8.25.2 構築子と解体子

8.25.2.1 MaterialBuffer() [1/3]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material * material = nullptr,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

デフォルトコンストラクタ.

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データのポインタ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

`gg.h` の 7755 行目に定義がります。

8.25.2.2 MaterialBuffer() [2/3]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material & material,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じデータで埋めるコンストラクタ.

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

`gg.h` の 7771 行目に定義がります。

8.25.2.3 MaterialBuffer() [3/3]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    GgVector ambient,
    GgVector diffuse,
    GgVector specular,
    GLfloat shininess,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じ引数で埋めるコンストラクタ.

引数

<i>ambient</i>	環境光に対する反射係数.
<i>diffuse</i>	拡散反射係数.
<i>specular</i>	鏡面反射係数.
<i>shininess</i>	輝き係数.
<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 7790 行目に定義があります。

8.25.2.4 ~MaterialBuffer()

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

デストラクタ.

gg.h の 7805 行目に定義があります。

8.25.3 関数詳解

8.25.3.1 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 7990 行目に定義があります。

8.25.3.2 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 7978 行目に定義があります。

8.25.3.3 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 5710 行目に定義があります。

8.25.3.4 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 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 7840 行目に定義があります。

8.25.3.5 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 5684 行目に定義があります。

8.25.3.6 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 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 5814 行目に定義があります。

8.25.3.7 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 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.cpp](#) の 5837 行目に定義がります。

8.25.3.8 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 5788 行目に定義がります。

8.25.3.9 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 5762 行目に定義がります。

8.25.3.10 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 7877 行目に定義がります。

8.25.3.11 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 5736 行目に定義がります。

8.25.3.12 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 5950 行目に定義がります。

8.25.3.13 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 5929 行目に定義がります。

8.25.3.14 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 5906 行目に定義がります。

8.25.3.15 loadSpecular() [2/3]

```
void gg::GgSimpleShader::MaterialBuffer::loadSpecular (
    const GLfloat * specular,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

鏡面反射係数を設定する。

引数

<i>specular</i>	鏡面反射係数を格納した GLfloat 型の 4 要素の配列変数。
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7947 行目に定義があります。

8.25.3.16 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 5880 行目に定義があります。

8.25.3.17 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 8000 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

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

8.26 Menu クラス

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

公開メンバ関数

- `Menu (const Config &config)`
- `Menu (const Menu &menu)=delete`
- `Menu (Menu &&menu)=default`
- `virtual ~Menu ()`
- `Menu & operator= (const Menu &menu)=delete`
- `Menu & operator= (Menu &&menu)=default`
- `const auto & getLight () const`
- `const auto & getShader () const`
- `const auto & getModel () const`
- `void draw ()`

フレンド

- class `Scene`

8.26.1 詳解

メニューの描画

`Menu.h` の 23 行目に定義がります。

8.26.2 構築子と解体子

8.26.2.1 `Menu()` [1/3]

```
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。

8.26.2.2 `Menu()` [2/3]

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

コピーコンストラクタは使用しない

8.26.2.3 `Menu()` [3/3]

```
Menu::Menu (
    Menu && menu ) [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

8.26.2.4 ~Menu()

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

デストラクタ。

[Menu.cpp](#) の 49 行目に定義がります。

8.26.3 関数詳解

8.26.3.1 draw()

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

描画する

[Menu.cpp](#) の 74 行目に定義がります。

8.26.3.2 getLight()

```
const auto & Menu::getLight ( ) const [inline]
```

光源データを取り出す

[Menu.h](#) の 81 行目に定義がります。

8.26.3.3 getModel()

```
const auto & Menu::getModel ( ) const [inline]
```

モデルデータを取り出す

[Menu.h](#) の 97 行目に定義がります。

8.26.3.4 getShader()

```
const auto & Menu::getShader ( ) const [inline]
```

シェーダを取り出す

[Menu.h](#) の 89 行目に定義がります。

8.26.3.5 operator=() [1/2]

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

代入演算子は使用しない

8.26.3.6 operator=() [2/2]

```
Menu & Menu::operator= (
    Menu && menu ) [default]
```

ムーブ代入演算子はデフォルトのものを使用する

8.26.4 フレンドと関連関数の詳解

8.26.4.1 Scene

```
friend class Scene [friend]
```

[Menu.h](#) の 26 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](#)

8.27 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))

8.27.1 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 150 行目に定義があります。

8.27.2 構築子と解体子

8.27.2.1 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

<i>title</i>	ウィンドウタイトルの文字列.
<i>width</i>	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.
<i>height</i>	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.
<i>fullscreen</i>	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.
<i>share</i>	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:



8.27.2.2 Window() [2/3]

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

コピーコンストラクタは使用しない.

引数

<i>w</i>	コピー元のウィンドウ.
----------	-------------

8.27.2.3 Window() [3/3]

```
GgApp::Window::Window (
```

Window && w)	[default]
---------------	-----------

ムーブコンストラクタ.

引数

w	ムーブ代入元のウィンドウ.
----------	---------------

8.27.2.4 ~Window()

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

デストラクタ.

[GgApp.h](#) の 284 行目に定義があります。

8.27.3 関数詳解

8.27.3.1 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 314 行目に定義があります。

8.27.3.2 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 638 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:



8.27.3.3 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 648 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:



8.27.3.4 getAltArrow()

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

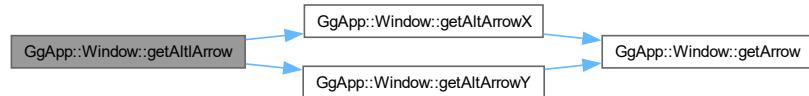
ALT キーを押しながら矢印キーを押したときの現在の値を得る。

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 658 行目に定義があります。

呼び出し関係図:



8.27.3.5 getArrow() [1/2]

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

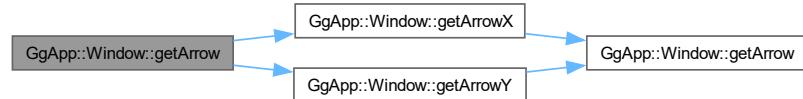
矢印キーの現在の値を得る。

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 565 行目に定義があります。

呼び出し関係図:



8.27.3.6 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

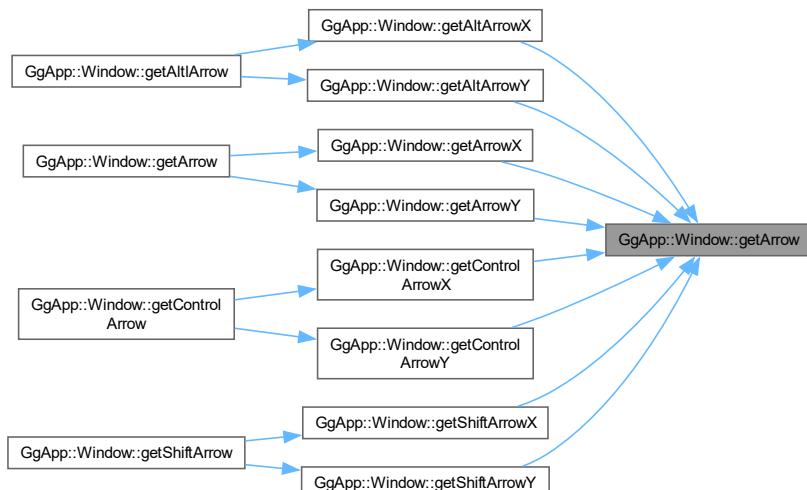
<i>direction</i>	方向 (0: X, 1:Y).
<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).

戻り値

矢印キーの値。

GgApp.h の 531 行目に定義があります。

被呼び出し関係図:



8.27.3.7 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 543 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:



8.27.3.8 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 554 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:



8.27.3.9 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 468 行目に定義があります。

8.27.3.10 getControlArrow()

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

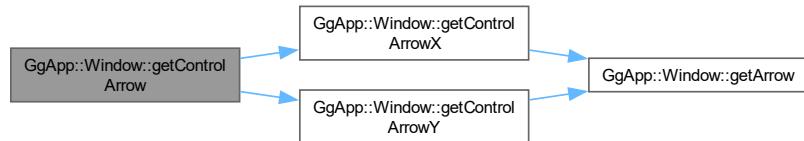
CTRL キーを押しながら矢印キーを押したときの現在の値を得る.

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 627 行目に定義があります。

呼び出し関係図:



8.27.3.11 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

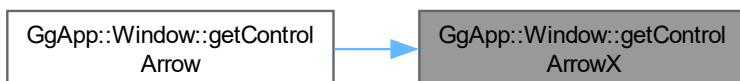
CTRL キーを押しながら矢印キーを押したときの現在の X 値.

GgApp.h の 607 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:



8.27.3.12 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 617 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:



8.27.3.13 getFboHeight()

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

FBO の高さを得る.

戻り値

FBO の高さ.

[GgApp.h](#) の 416 行目に定義があります。

被呼び出し関係図:



8.27.3.14 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 447 行目に定義があります。

8.27.3.15 getFboSize() [2/2]

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

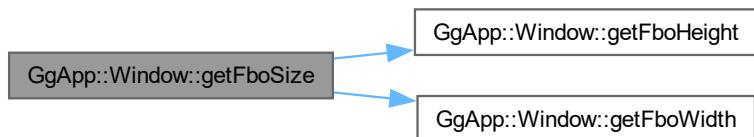
FBO のサイズを得る。

引数

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

[GgApp.h](#) の 457 行目に定義があります。

呼び出し関係図:



8.27.3.16 getFboWidth()

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

FBO の横幅を得る。

戻り値

FBO の横幅.

[GgApp.h](#) の 406 行目に定義があります。

被呼び出し関係図:



8.27.3.17 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 396 行目に定義があります。

被呼び出し関係図:



8.27.3.18 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 478 行目に定義があります。

8.27.3.19 getLastKey()

```
int GgApp::Window::getLastKey ( ) const [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 516 行目に定義があります。

8.27.3.20 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 669 行目に定義があります。

8.27.3.21 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 680 行目に定義があります。

8.27.3.22 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 693 行目に定義があります。

8.27.3.23 `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 704 行目に定義があります。

8.27.3.24 `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 715 行目に定義があります。

8.27.3.25 `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

<code>button</code>	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 843 行目に定義があります。

8.27.3.26 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 856 行目に定義があります。

8.27.3.27 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

<i>button</i>	平行移動量を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	---

戻り値

クリッピング座標系で平行移動を行う GgMatrix 型の変換行列。

GgApp.h の 820 行目に定義があります。

8.27.3.28 getShiftArrow()

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

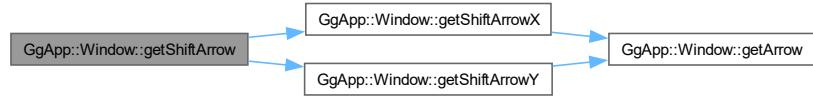
SHIFT キーを押しながら矢印キーを押したときの現在の値を得る。

引数

<i>shift_arrow</i>	SHIFT キーを押しながら矢印キーを押したときの値を格納する GLfloat 型の 2 要素の配列。
--------------------	---

GgApp.h の 596 行目に定義があります。

呼び出し関係図:



8.27.3.29 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 576 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:



8.27.3.30 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 586 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:



8.27.3.31 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 426 行目に定義があります。

8.27.3.32 getSize() [2/2]

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

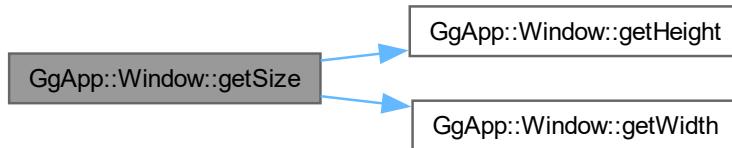
ウィンドウのサイズを得る.

引数

size	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 436 行目に定義があります。

呼び出し関係図:



8.27.3.33 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

button	平行移動量を取得するマウスボタン (<code>GLFW_MOUSE_BUTTON_[1,2]</code>).
---------------	--

戻り値

平行移動量を格納した `GLfloat[3]` の配列のポインタ.

[GgApp.h](#) の 784 行目に定義があります。

8.27.3.34 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

button	平行移動量を取得するマウスボタン (<code>GLFW_MOUSE_BUTTON_[1,2]</code>).
---------------	--

戻り値

視点座標系で平行移動を行う GgMatrix 型の変換行列.

GgApp.h の 797 行目に定義がります。

8.27.3.35 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 898 行目に定義がります。

8.27.3.36 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 726 行目に定義がります。

8.27.3.37 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 737 行目に定義がります。

8.27.3.38 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 750 行目に定義があります。

8.27.3.39 `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 761 行目に定義があります。

8.27.3.40 `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 772 行目に定義があります。

8.27.3.41 `getWidth()`

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

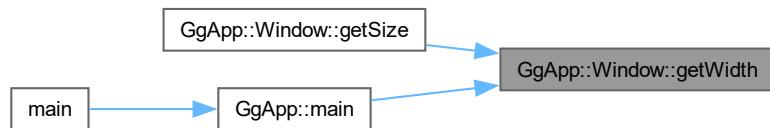
ウィンドウの横幅を得る。

戻り値

ウィンドウの横幅。

[GgApp.h](#) の 386 行目に定義があります。

被呼び出し関係図:



8.27.3.42 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。

8.27.3.43 operator=() [1/2]

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

代入演算子は使用しない。

引数

w	代入元のウィンドウ。
---	------------

戻り値

代入後のこのオブジェクトの参照。

8.27.3.44 operator=() [2/2]

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

ムーブ代入演算子。

引数

w	ムーブ代入元のウィンドウ。
---	---------------

戻り値

ムーブ代入後のこのオブジェクトの参照。

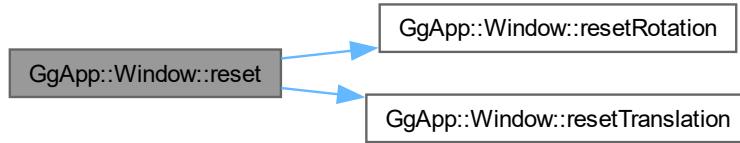
8.27.3.45 reset()

```
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 884 行目に定義があります。

呼び出し関係図:



8.27.3.46 resetRotation()

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

トラックボール処理を初期化する。

[GgApp.h](#) の 866 行目に定義があります。

被呼び出し関係図:



8.27.3.47 resetTranslation()

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

平行移動量を初期化する。

[GgApp.h](#) の 875 行目に定義があります。

被呼び出し関係図:



8.27.3.48 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 355 行目に定義があります。

8.27.3.49 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 493 行目に定義があります。

8.27.3.50 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 324 行目に定義があります。

8.27.3.51 setKeyboardFunc()

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

ユーザ定義の keyboard 関数を設定する。

引数

<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 928 行目に定義があります。

8.27.3.52 setMouseFunc()

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

ユーザ定義の `mouse` 関数を設定する。

引数

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

[GgApp.h](#) の 938 行目に定義があります。

8.27.3.53 setResizeFunc()

```
void GgApp::Window::setResizeFunc (
    void(*)(const Window *window, int width, int height) func ) [inline]
```

ユーザ定義の `resize` 関数を設定する。

引数

<i>func</i>	ユーザ定義の <code>resize</code> 関数、ウィンドウのサイズ変更時に呼び出される。
-------------	--

[GgApp.h](#) の 918 行目に定義があります。

8.27.3.54 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 908 行目に定義があります。

8.27.3.55 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 506 行目に定義がります。

8.27.3.56 setWheelFunc()

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

ユーザ定義の wheel 関数を設定する.

引数

<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 948 行目に定義がります。

8.27.3.57 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 334 行目に定義がります。

8.27.3.58 swapBuffers()

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

カラーバッファを入れ替える.

GgApp.cpp の 492 行目に定義がります。

8.27.3.59 updateViewport()

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

ビューポートのサイズを更新する.

GgApp.cpp の 511 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:

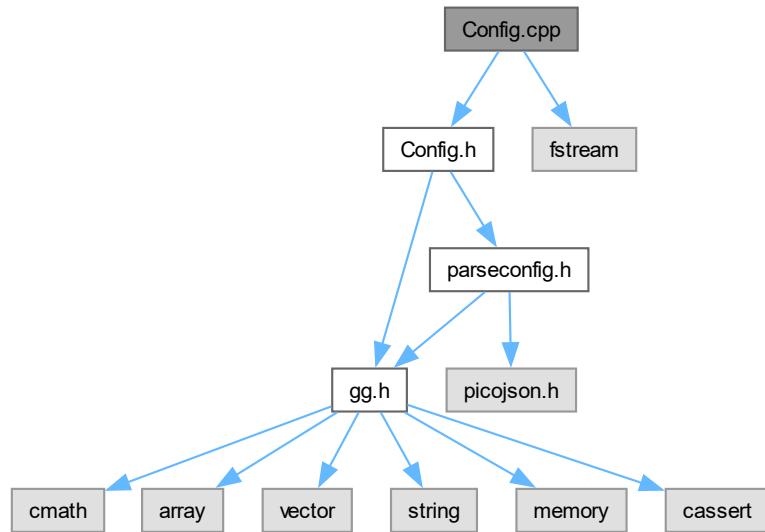
- [GgApp.h](#)
- [GgApp.cpp](#)

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](#) に定義がります。

9.1.2 変数詳解

9.1.2.1 defaultLight

```
constexpr 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 }
```

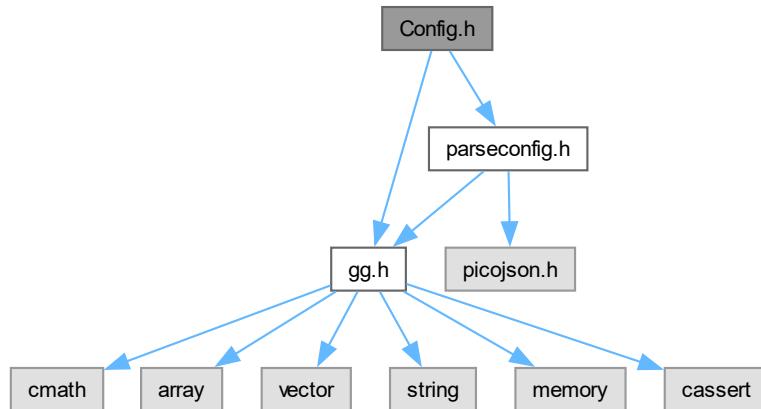
```

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アル化して JSON で保存
00146 picojson::value v{ object };
00147 file << v.serialize(true);
00148
00149 return true;
00150 }

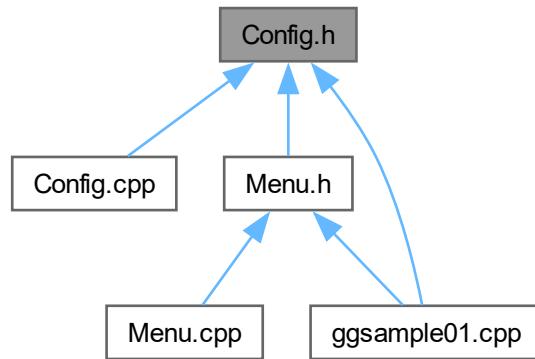
```

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 {
```

```

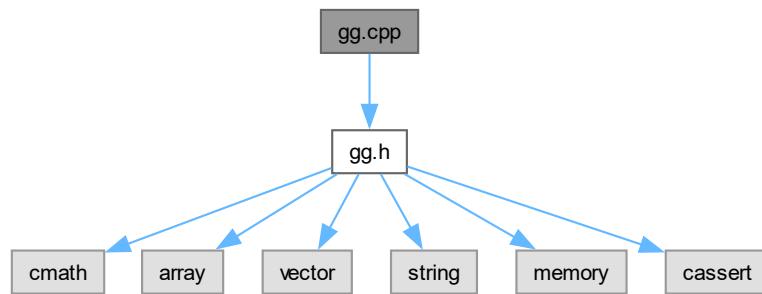
00023 // メニュークラスから参照する
00024 friend class Menu;
00025
00026 // ウィンドウサイズ
00027 std::array<GLsizei, 2> winSize;
00028
00029 // メニューフォント名
00030 std::string menuFont;
00031
00032 // メニューフォントサイズ
00033 float menuFontSize;
00034
00035 // 光源
00036 GgSimpleShader::Light light;
00037
00038 // 形状ファイル名
00039 std::string model;
00040
00041 // シエーダのソースファイル名
00042 std::array<std::string, 3> shader;
00043
00044 public:
00045
00046 Config();
00047 Config(const std::string& filename);
00048
00049 virtual ~Config();
00050
00051 auto getWidth() const
00052 {
00053     return winSize[0];
00054 }
00055
00056 auto getHeight() const
00057 {
00058     return winSize[1];
00059 }
00060
00061 bool load(const std::string& filename);
00062
00063 bool save(const std::string& filename) const;
00064 };
00065

```

9.5 gg.cpp ファイル

#include "gg.h"

gg.cpp の依存先関係図:



9.5.1 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。

9.6 gg.cpp

[詳解]

```
00001 /*
00002 ゲームグラフィックス特論用補助プログラム GLFW3 版
00004
00005 Copyright (c) 2011-2025 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
00025 #include "gg.h"
00026
00027 // 標準ライブラリ
00028 #include <cmath>
00029 #include <cstdlib>
00030 #include <iostream>
00031 #include <fstream>
00032 #include <sstream>
00033 #include <limits>
00034 #include <map>
00035
00036 // Windows (Visual Studio) のとき
00037 #if defined(_MSC_VER)
00038 // リリースビルドではコンソールを出さない
00039 # if !defined(_DEBUG)
00040 # pragma comment(linker, "/subsystem:\\"windows\\" /entry:\\"mainCRTStartup\\")
00041 # endif
00042 // リンクするライブラリ
00043 # pragma comment(lib, "lib\\\" GLFW3_PLATFORM \"\\\" GLFW3_CONFIGURATION \"\\\" glfw3.lib")
00044
00045 // 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
00046 // Original author: mt-u
00047 // Copyright (c) 2013 mt-u
00048 // https://gist.github.com/mt-u/6878251
00049 // Modified by: Kohe Tokoi
00050 //
00051 //
00052 //
00053 //
00054 //
00055 //
00056 //
00057 //
00058 //
00059 //
00060 //
00061 //
00062 //
00063 //
00064 //
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 PFNGLACTIVETEXTUREPROC glActiveTexture;
00112 PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
00113 PFNGLATTACHSHADERPROC glAttachShader;
00114 PFNGLBEGINCNDITRANSLATEPROC glBeginConditionalRenderNV;
00115 PFNGLBEGINCNDITRANSLATEPROC 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 PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
00132 PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00133 PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00134 PFNGLBINDRENDERBUFFERPROC 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 PFNGLBLENDEQUATIONNIPROC glBindEquationi;
00149 PFNGLBLENDEQUATIONPROC glBindEquation;
00150 PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00151 PFNGLBLENDEQUATIONSEPARATEIIPROC glBindEquationSeparatei;
00152 PFNGLBLENDEQUATIONSEPARATEPROC glBindEquationSeparate;
00153 PFNGLBLENDFUNCARBPROC glBindFunciARB;
00154 PFNGLBLENDFUNCIPROC glBindFunci;
00155 PFNGLBLENDFUNCPROC glBindFunc;
00156 PFNGLBLENDFUNCSEPARATEIARBPROC glBindFuncSeparateiARB;
00157 PFNGLBLENDFUNCSEPARATEIIPROC glBindFuncSeparatei;
00158 PFNGLBLENDFUNCSEPARATEPROC glBindFuncSeparate;

```

```
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 glClearBufferuiv;
00178 PFNGLCLEARCOLORPROC glClearColor;
00179 PFNGLCLEARDEPTHFPROC glClearDepthf;
00180 PFNGLCLEARDEPTHPROC glClearDepth;
00181 PFNGLCLEARNAMEDFRAMEBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
00182 PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
00183 PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferDataEXT;
00184 PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferData;
00185 PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
00186 PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
00187 PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
00188 PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferuiv;
00189 PFNGLCLEARPROC glClear;
00190 PFNGLCLEARSTENCILPROC glClearStencil;
00191 PFNGLCLEARTEXIMAGEPROC glClearTexImage;
00192 PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
00193 PFNGLCLIENTATTRIBDEFAULTTEXTPROC 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 glCompressedTextureSubImage1DEXT;
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 PFNGLCOPYTEXTURESUBIMAGE1DPROC 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 PFNGLCREATESAMPLERSRCPROC 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 PFNGLDELETESAMPLERSRCPROC 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 PFNGLDEPTHRANGEARRAYPROC 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 PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC 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 PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstanced;
00335 PFNGLDRAWELEMENTSPROC glDrawElements;
00336 PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00337 PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00338 PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROCPROC glDrawTransformFeedbackInstanced;
00339 PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00340 PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROCPROC glDrawTransformFeedbackStreamInstanced;
00341 PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00342 PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00343 PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00344 PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00345 PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00346 PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00347 PFNGLENABLEIPROC glEnablei;
00348 PFNGLENABLEPROCP glEnable;
00349 PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexAttribArrayAttribEXT;
00350 PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00351 PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00352 PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArrayAttribArray;
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 PFNGLFENCESYNCPROC 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 PFNGLFRAMEBUFFERSAMPLELOCATIONSVNPROC glFramebufferSampleLocationsfvNV;
00376 PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00377 PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00378 PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00379 PFNGLFRAMEBUFFERTEXTUREARBPROC 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 PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC 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 PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
00424 PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
00425 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00426 PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00427 PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00428 PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00429 PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00430 PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00431 PFNGLGETDOUBLEL_VEXTPROC glGetDoublei_vEXT;
00432 PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00433 PFNGLGETDOUBLEVPROC glGetDoublev;
00434 PFNGLGETTERRORPROC 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 PFNGLGETNAMEDBUFFERPARAMETERUI64NVPROC 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 PFNGLGETNAMEDPROGRAMFVEXTPROC glGetNamedProgramfvEXT;
00485 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00486 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00487 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIiivEXT;
00488 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00489 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00490 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00491 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00492 PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00493 PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00494 PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
00495 PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
00496 PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00497 PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00498 PFNGLGETNTEXIMAGEPROC glGetnTexImage;
00499 PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
00500 PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
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 glGetProgramResourcefv;
00541 PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00542 PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00543 PFNGLGETPROGRAMRESOURCELLOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00544 PFNGLGETPROGRAMRESOURCELLOCATIONPROC glGetProgramResourceLocation;
00545 PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00546 PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00547 PFNGLGETQUERYBUFFEROBJECTI16VPROC glGetQueryBufferObjecti16v;
00548 PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00549 PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00550 PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00551 PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00552 PFNGLGETQUERYIVPROC glGetQueryiv;
00553 PFNGLGETQUERYOBJECTI16VPROC glGetQueryObjecti16v;
00554 PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
00555 PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00556 PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00557 PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00558 PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00559 PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00560 PFNGLGETSAMPLERPARAMETERIUIUVPROC 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 PFNGLGETSYNCCIVPROC glGetSynciv;
00572 PFNGLGETTEXIMAGEPROC glGetTexImage;
00573 PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00574 PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00575 PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00576 PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiiv;
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 PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
```

```

00594 PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00595 PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00596 PFNGLGETTEXTURESAMPLERHANDLEENVPROC glGetTextureSamplerHandleNV;
00597 PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00598 PFNGLGETTRANSFORMFEEDBACKI64_VPROC glGetTransformFeedbacki64_v;
00599 PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00600 PFNGLGETTRANSFORMFEEDBACKVPROC glGetTransformFeedbackv;
00601 PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00602 PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00603 PFNGLGETUNIFORMDPROC glGetUniformd;
00604 PFNGLGETUNIFORMFPROC glGetUniformf;
00605 PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00606 PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
00607 PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
00608 PFNGLGETUNIFORMIVPROC glGetUniformiv;
00609 PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
00610 PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
00611 PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
00612 PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
00613 PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
00614 PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00615 PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
00616 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00617 PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00618 PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00619 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00620 PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00621 PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribbdv;
00622 PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribfv;
00623 PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribIiiv;
00624 PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribIuiv;
00625 PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLbdv;
00626 PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLdv;
00627 PFNGLGETVERTEXATTRIBLI64VNVPROC glGetVertexAttribLi64vNV;
00628 PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribLui64vARB;
00629 PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribLui64vNV;
00630 PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribPointerv;
00631 PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrNV;
00632 PFNGLHINTPROC glHint;
00633 PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00634 PFNGLINSETEVENTMARKEREXTPROC 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 PFNGLISCOMMANDLISTNVPROC 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 PFNGLISTTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00667 PFNGLISTTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00668 PFNGLISTTEXTUREPROC glIsTexture;
00669 PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00670 PFNGLISVERTEXARRAYPROC glIsVertexArray;
00671 PFNGLLABELOBJECTEXTPROC 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 PFNGLMAPNAMEDBUFFERPROC glMapNamedBufferRange;
00694 PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00695 PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
00696 PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
00697 PFNGLMATRIXLOADDEXTPROC glMatrixLoaddEXT;
00698 PFNGLMATRIXLOADFEXTPROC glMatrixLoadfEXT;
00699 PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00700 PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
00701 PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedEXT;
00702 PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
00703 PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
00704 PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
00705 PFNGLMATRIXMULTDEXTPROC glMatrixMultdEXT;
00706 PFNGLMATRIXMULTFEXTPROC glMatrixMultfEXT;
00707 PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
00708 PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
00709 PFNGLMATRIXMULTTRANSPPOSEFEXTPROC glMatrixMultTransposefEXT;
00710 PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
00711 PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
00712 PFNGLMATRIXPUSHEXTPROC 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 PFNGLMULTIDRAWARRAYSINDIRECTCOUNTRBPROC glMultiDrawArraysIndirectCountARB;
00727 PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00728 PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
00729 PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00730 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00731 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00732 PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTRBPROC glMultiDrawElementsIndirectCountARB;
00733 PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00734 PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00735 PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00736 PFNGLMULTITEXCOORDINTEREXTPROC glMultiTexCoordPointerEXT;
00737 PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00738 PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00739 PFNGLMULTITEXENVVIEXTPROC glMultiTexEnviEXT;
00740 PFNGLMULTITEXENVIVEXTPROC glMultiTexEnvivEXT;
00741 PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00742 PFNGLMULTITEXGENDEXTPROC 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 PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameterfEXT;
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 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSVARBPROC glNamedFramebufferSampleLocationsfvARB;
00777 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNPROC glNamedFramebufferSampleLocationsfvNV;
00778 PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00779 PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00780 PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00781 PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00782 PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00783 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00784 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00785 PFNGLNAMEDFRAMEBUFFERTEXTUREREPROC glNamedFramebufferTexture;
00786 PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00787 PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00788 PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
00789 PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00790 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
00791 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00792 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00793 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UVEEXTPROC glNamedProgramLocalParameterI4uveEXT;
00794 PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00795 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
00796 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEEXTPROC glNamedProgramLocalParametersI4uvEXT;
00797 PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00798 PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00799 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERGEEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00800 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00801 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00802 PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00803 PFNGLNAMEDESTRINGARBPROC glNamedStringARB;
00804 PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00805 PFNGLOBJECTLABELPROC glBindObjectLabel;
00806 PFNGLOBJECTPTRLABELPROC glBindObjectPtrLabel;
00807 PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
00808 PFNGLPATCHPARAMETERIPROC glPatchParameteri;
00809 PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00810 PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00811 PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00812 PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00813 PFNGLPATHGLYPHINDEXEXARRAYNVPROC glPathGlyphIndexArrayNV;
00814 PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
00815 PFNGLPATHGLYPHRANGEENVPROC glPathGlyphRangeNV;
00816 PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
00817 PFNGLPATHMEMORYGLYPHINDEXEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
00818 PFNGLPATHPARAMETERFNVPROC glPathParameterfvNV;
00819 PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00820 PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
00821 PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
00822 PFNGLPATHSTENCILDEPTHOFFSETNVPROC 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 PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00834 PFNGLPOINTSIZEPROC glPointSize;
00835 PFNGLPOINTSIZEXPROC 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 glProgramUniform1d;
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 glProgramUniformMatrix2dEXT;
00948 PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2d;
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 PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
00985 PFNGLPROVOKINGVERTEXEXTPROC glProvokingVertex;
00986 PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC 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 PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERGENVPROC glRenderbufferStorageMultisampleCoverageNV;
00997 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
00998 PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
00999 PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
01000 PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
01001 PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
01002 PFNGLSAMPLEMASKIPROC glSampleMaski;
01003 PFNGLSAMPLERPARAMETERFPROC glSamplerParameterf;
01004 PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
01005 PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIi;
01006 PFNGLSAMPLERPARAMETERIPROC glSamplerParameteri;
01007 PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIu;
01008 PFNGLSAMPLERPARAMETERIVPROC glSamplerParameteriv;
01009 PFNGLSCISSORARRAYVPROC glScissorArray;
01010 PFNGLSCISSORINDEXEDPROC glScissorIndexed;
01011 PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
01012 PFNGLSCISSORPROC glScissor;
01013 PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
01014 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
01015 PFNGLSHADERBINARYPROC glShaderBinary;
01016 PFNGLSHADERSOURCEPROC glShaderSource;
01017 PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
01018 PFNGLSIGNALVKFENCENVPROC glSignalVkFenceNV;
01019 PFNGLSIGNALVKSEMAPHORENVPROC glSignalVkSemaphoreNV;
01020 PFNGLSPECIALIZE_SHADERARBPROC 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 PFNGLSTENCILSTROKEPATHINSTANCENVPROC glStencilStrokePathInstancedNV;
01031 PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
01032 PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01033 PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01034 PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCENVPROC glStencilThenCoverStrokePathInstancedNV;
01035 PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
01036 PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
01037 PFNGLTEXBUFFERARBPROC glTexBufferARB;
01038 PFNGLTEXBUFFERPROC glTexBuffer;
01039 PFNGLTEXBUFFERRANGEPROC glTexBufferRange;
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 PFNGLTEXPARAMETERIPROC glTexParameterI;
01051 PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01052 PFNGLTEXPARAMETERIVPROC glTexParameteriv;
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 PFNGLTEXTUREBUFFEREXTPROC 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 PFNGLTEXTUREPARAMETERIEVTPROC glTextureParameteriEXT;
01076 PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIivEXT;
01077 PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiv;
01078 PFNGLTEXTUREPARAMETERIPROC glTextureParameteri;
01079 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01080 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01081 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameteriuvEXT;
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 PFNGLUNIFORM2UIPROC glUniform2ui;
01136 PFNGLUNIFORM2UIVPROC glUniform2uiv;
01137 PFNGLUNIFORM3DPROC glUniform3d;
01138 PFNGLUNIFORM3DVPROC glUniform3dv;
01139 PFNGLUNIFORM3FPROC 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 PFNGLUNIFORM4FFPROC 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 PFNGLUNIFORMMATRIX2FPROC glUniformMatrix2fv;
01176 PFNGLUNIFORMMATRIX2X3DPROC glUniformMatrix2x3dv;
01177 PFNGLUNIFORMMATRIX2X3FPROC glUniformMatrix2x3fv;
01178 PFNGLUNIFORMMATRIX2X4DPROC glUniformMatrix2x4dv;
01179 PFNGLUNIFORMMATRIX2X4FPROC glUniformMatrix2x4fv;
01180 PFNGLUNIFORMMATRIX3DPROC glUniformMatrix3dv;
01181 PFNGLUNIFORMMATRIX3FPROC glUniformMatrix3fv;
01182 PFNGLUNIFORMMATRIX3X2DPROC glUniformMatrix3x2dv;
01183 PFNGLUNIFORMMATRIX3X2FPROC glUniformMatrix3x2fv;
01184 PFNGLUNIFORMMATRIX3X4DPROC glUniformMatrix3x4dv;
01185 PFNGLUNIFORMMATRIX3X4FPROC glUniformMatrix3x4fv;
01186 PFNGLUNIFORMMATRIX4DPROC glUniformMatrix4dv;
01187 PFNGLUNIFORMMATRIX4FPROC glUniformMatrix4fv;
01188 PFNGLUNIFORMMATRIX4X2DPROC glUniformMatrix4x2dv;
01189 PFNGLUNIFORMMATRIX4X2FPROC glUniformMatrix4x2fv;
01190 PFNGLUNIFORMMATRIX4X3DPROC glUniformMatrix4x3dv;
01191 PFNGLUNIFORMMATRIX4X3FPROC 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 PFNGLVERTEXARRAYATTRIBIFORMATPROC glVertexArrayAttribIFormat;
01206 PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
01207 PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01208 PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01209 PFNGLVERTEXARRAYCOLOROFFSETSETEXTPROC glVertexArrayColorOffsetEXT;
01210 PFNGLVERTEXARRAYEDGEFLAGOFFSETSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01211 PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01212 PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01213 PFNGLVERTEXARRAYINDEXOFFSETSETEXTPROC glVertexArrayIndexOffsetEXT;
01214 PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01215 PFNGLVERTEXARRAYNORMALOFFSETSETEXTPROC glVertexArrayNormalOffsetEXT;
01216 PFNGLVERTEXARRAYSECONDARYCOLOROFFSETSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01217 PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
01218 PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
01219 PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01220 PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01221 PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01222 PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01223 PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
01224 PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
01225 PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01226 PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01227 PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
01228 PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01229 PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
01230 PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
01231 PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
01232 PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
01233 PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
01234 PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
01235 PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
01236 PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
01237 PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
01238 PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
01239 PFNGLVERTEXATTRIB2FPROC glVertexAttrib2fv;
01240 PFNGLVERTEXATTRIB2SPROC glVertexAttrib2s;
01241 PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
01242 PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
01243 PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
01244 PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
01245 PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
01246 PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
01247 PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
01248 PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
01249 PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
01250 PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
01251 PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
01252 PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
01253 PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
01254 PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4nbv;
01255 PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4niv;
01256 PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4nsv;
01257 PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4nub;
01258 PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4nubv;
01259 PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4nui;
01260 PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4nusv;
01261 PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
01262 PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
01263 PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
01264 PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
01265 PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
01266 PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01267 PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01268 PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01269 PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01270 PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01271 PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01272 PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
01273 PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
01274 PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
01275 PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01276 PFNGLVERTEXATTRIBI2IVPROC 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 glVertexAttribLlui64ARB;
01299 PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribLlui64NV;
01300 PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribLlui64vARB;
01301 PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribLlui64vNV;
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 glViewportArrayv;
01335 PFNGLVIEWPORTINDEXEDFPROC glViewportIndexeddf;
01336 PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;
01337 PFNGLVIEWPORTPOSITIONWSCALENVPROC glviewportPositionWScaleNV;
01338 PFNGLVIEWPORTPROC glviewport;
01339 PFNGLVIEWPORTSWIZZLENVPROC glviewportSwizzleNV;
01340 PFNGLWAITSYNCNVPROC glWaitSync;
01341 PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01342 PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01343 PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01344 #endif
01345
01347
01349 GLint gg::ggBufferAlignment(0);
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 =
01370         PFNGLBEGINCONDITIONALRENDERNVPROC(glfwGetProcAddress("glBeginConditionalRenderNV"));
01371     glBeginPerfMonitorAMD = PFNGLBEGINPERFMONITORAMDPROC(glfwGetProcAddress("glBeginPerfMonitorAMD"));
01372     glBeginPerfQueryINTEL = PFNGLBEGINPERFQUERYINTELPROC(glfwGetProcAddress("glBeginPerfQueryINTEL"));
01373     glBeginQuery = PFNGLBEGINQUERYPROC(glfwGetProcAddress("glBeginQuery"));
01374     glBeginQueryIndexed = PFNGLBEGINQUERYINDEXEDPROC(glfwGetProcAddress("glBeginQueryIndexed"));
01375     glBeginTransformFeedback =
01376         PFNGLBEGINTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBeginTransformFeedback"));
01377     glBindAttribLocation = PFNGLBINDATTRIBLOCATIONPROC(glfwGetProcAddress("glBindAttribLocation"));

```

```

01374 glBindBuffer = PFNGLBINDBUFFERPROC(glfwGetProcAddress("glBindBuffer"));
01375 glBindBufferBase = PFNGLBINDBUFFERBASEPROC(glfwGetProcAddress("glBindBufferBase"));
01376 glBindBufferRange = PFNGLBINDBUFFERRANGEPROC(glfwGetProcAddress("glBindBufferRange"));
01377 glBindBuffersBase = PFNGLBINDBUFFERSBASEPROC(glfwGetProcAddress("glBindBuffersBase"));
01378 glBindBuffersRange = PFNGLBINDBUFFERSRANGEPROC(glfwGetProcAddress("glBindBuffersRange"));
01379 glBindFragDataLocation =
    PFNGLBINDFRAGDATALOCATIONPROC(glfwGetProcAddress("glBindFragDataLocation"));
01380 glBindFragDataLocationIndexed =
    PFNGLBINDFRAGDATALOCATIONINDEXEDPROC(glfwGetProcAddress("glBindFragDataLocationIndexed"));
01381 glBindFramebuffer = PFNGLBINDFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
01382 glBindImageTexture = PFNGLBINDIMAGETEXTUREPROC(glfwGetProcAddress("glBindImageTexture"));
01383 glBindImageTextures = PFNGLBINDIMAGETEXTURESPROC(glfwGetProcAddress("glBindImageTextures"));
01384 glBindMultiTextureEXT = PFNGLBINDMULTITEXTUREEXTPROC(glfwGetProcAddress("glBindMultiTextureEXT"));
01385 glBindProgramPipeline = PFNGLBINDPROGRAMPIPELINEPROC(glfwGetProcAddress("glBindProgramPipeline"));
01386 glBindRenderbuffer = PFNGLBINDRENDERBUFFERPROC(glfwGetProcAddress("glBindRenderbuffer"));
01387 glBindSampler = PFNGLBINDSAMPLERPROC(glfwGetProcAddress("glBindSampler"));
01388 glBindSamplers = PFNGLBINDSAMPLERSPROC(glfwGetProcAddress("glBindSamplers"));
01389 glBindTexture = PFNGLBINDTEXTUREPROC(glfwGetProcAddress("glBindTexture"));
01390 glBindTextureUnit = PFNGLBINDTEXTUREUNITPROC(glfwGetProcAddress("glBindTextureUnit"));
01391 glBindTextures = PFNGLBINDTEXTURESPROC(glfwGetProcAddress("glBindTextures"));
01392 glBindTransformFeedback =
    PFNGLBINDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBindTransformFeedback"));
01393 glBindVertexArray = PFNGLBINDVERTEXARRAYPROC(glfwGetProcAddress("glBindVertexArray"));
01394 glBindVertexBuffer = PFNGLBINDVERTEXBUFFERPROC(glfwGetProcAddress("glBindVertexBuffer"));
01395 glBindVertexBuffers = PFNGLBINDVERTEXBUFFERSPROC(glfwGetProcAddress("glBindVertexBuffers"));
01396 glBindBarrierKHR = PFNGLBLENDBARRIERKHRPROC(glfwGetProcAddress("glBindBarrierKHR"));
01397 glBindBarrierNV = PFNGLBLENDBARRIERNVPROC(glfwGetProcAddress("glBindBarrierNV"));
01398 glBindColor = PFNGLBLENDCOLORPROC(glfwGetProcAddress("glBindColor"));
01399 glBindEquation = PFNGLBLENEQUATIONPROC(glfwGetProcAddress("glBindEquation"));
01400 glBindEquationSeparate =
    PFNGLBLENEQUATIONSEPARATEPROC(glfwGetProcAddress("glBindEquationSeparate"));
01401 glBindEquationSeparatei =
    PFNGLBLENEQUATIONSEPARATEIPROC(glfwGetProcAddress("glBindEquationSeparatei"));
01402 glBindEquationSeparateiARB =
    PFNGLBLENEQUATIONSEPARATEIARBPROC(glfwGetProcAddress("glBindEquationSeparateiARB"));
01403 glBindEquationi = PFNGLBLENEQUATIONIPROC(glfwGetProcAddress("glBindEquationi"));
01404 glBindEquationiARB = PFNGLBLENEQUATIONIARBPROC(glfwGetProcAddress("glBindEquationiARB"));
01405 glBindFunc = PFNGLBLENDFUNCPROC(glfwGetProcAddress("glBindFunc"));
01406 glBindFuncSeparate = PFNGLBLENDFUNCSEPARATEPROC(glfwGetProcAddress("glBindFuncSeparate"));
01407 glBindFuncSeparatei = PFNGLBLENDFUNCSEPARATEIPROC(glfwGetProcAddress("glBindFuncSeparatei"));
01408 glBindFuncSeparateiARB =
    PFNGLBLENDFUNCSEPARATEIARBPROC(glfwGetProcAddress("glBindFuncSeparateiARB"));
01409 glBindFunci = PFNGLBLENDFUNCIPROC(glfwGetProcAddress("glBindFunci"));
01410 glBindFunciARB = PFNGLBLENDFUNCIARBPROC(glfwGetProcAddress("glBindFunciARB"));
01411 glBindParameteriNV = PFNGLBLENDPARAMETERINVPROC(glfwGetProcAddress("glBindParameteriNV"));
01412 glBindFramebuffer = PFNGLBLITFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
01413 glBindNamedFramebuffer =
    PFNGLBLITNAMEDFRAMEBUFFERPROC(glfwGetProcAddress("glBindNamedFramebuffer"));
01414 glBindBufferAddressRangeNV =
    PFNGLBUFFERADDRESSRANGENVPROC(glfwGetProcAddress("glBufferAddressRangeNV"));
01415 glBindBufferData = PFNGLBUFFERDATAPROC(glfwGetProcAddress("glBufferData"));
01416 glBindPageCommitmentARB =
    PFNGLBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glBufferPageCommitmentARB"));
01417 glBindBufferStorage = PFNGLBUFFERSTORAGEPROC(glfwGetProcAddress("glBufferStorage"));
01418 glBindBufferSubData = PFNGLBUFFERSUBDATAPROC(glfwGetProcAddress("glBufferSubData"));
01419 glBindCommandListNV = PFNGLCALLCOMMANDLISTNVPROC(glfwGetProcAddress("glCallCommandListNV"));
01420 glBindCheckFramebufferStatus =
    PFNGLCHECKFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckFramebufferStatus"));
01421 glBindCheckNamedFramebufferStatus =
    PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckNamedFramebufferStatus"));
01422 glBindCheckNamedFramebufferStatusEXT =
    PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC(glfwGetProcAddress("glCheckNamedFramebufferStatusEXT"));
01423 glBindClampColor = PFNGLCLAMPCOLORPROC(glfwGetProcAddress("glClampColor"));
01424 glBindClear = PFNGLCLEARPROC(glfwGetProcAddress("glClear"));
01425 glBindClearBufferData = PFNGLCLEARBUFFERDATAPROC(glfwGetProcAddress("glClearBufferData"));
01426 glBindClearBufferSubData = PFNGLCLEARBUFFERSUBDATAPROC(glfwGetProcAddress("glClearBufferSubData"));
01427 glBindClearBufferfi = PFNGLCLEARBUFFERFIPROC(glfwGetProcAddress("glClearBufferfi"));
01428 glBindClearBufferfv = PFNGLCLEARBUFFERFVPROC(glfwGetProcAddress("glClearBufferfv"));
01429 glBindClearBufferiv = PFNGLCLEARBUFFERIVPROC(glfwGetProcAddress("glClearBufferiv"));
01430 glBindClearBufferuiv = PFNGLCLEARBUFFERUIVPROC(glfwGetProcAddress("glClearBufferuiv"));
01431 glBindClearColor = PFNGLCLEARCOLORPROC(glfwGetProcAddress("glClearColor"));
01432 glBindClearDepth = PFNGLCLEARDEPTHPROC(glfwGetProcAddress("glClearDepth"));
01433 glBindClearDepthf = PFNGLCLEARDEPTHFPROC(glfwGetProcAddress("glClearDepthf"));
01434 glBindClearNamedBufferData =
    PFNGLCLEARNAMEDBUFFERDATAPROC(glfwGetProcAddress("glClearNamedBufferData"));
01435 glBindClearNamedBufferDataEXT =
    PFNGLCLEARNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataEXT"));
01436 glBindClearNamedBufferSubData =
    PFNGLCLEARNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glClearNamedBufferSubData"));
01437 glBindClearNamedBufferSubDataEXT =
    PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferSubDataEXT"));
01438 glBindClearNamedFramebuffifi =
    PFNGLCLEARNAMEDFRAMEBUFFERFIPROC(glfwGetProcAddress("glClearNamedFramebuffifi"));
01439 glBindClearNamedFramebufferv =
    PFNGLCLEARNAMEDFRAMEBUFFERFVPROC(glfwGetProcAddress("glClearNamedFramebufferv"));
01440 glBindClearNamedFramebufferviv =
    PFNGLCLEARNAMEDFRAMEBUFFERIVPROC(glfwGetProcAddress("glClearNamedFramebufferviv"));

```

```

01441     glClearNamedFramebufferuiv =
01442         PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC(glfwGetProcAddress("glClearNamedFramebufferuiv"));
01443     glClearStencil = PFNGLCLEARSTENCILPROC(glfwGetProcAddress("glClearStencil"));
01444     glClearTexImage = PFNGLCLEARTEXIMAGEPROC(glfwGetProcAddress("glClearTexImage"));
01445     glClearTexSubImage = PFNGLCLEARTEXSUBIMAGEPROC(glfwGetProcAddress("glClearTexSubImage"));
01446     glClientAttribDefaultEXT =
01447         PFNGLCLIENTATTRIBDEFAULTTEXTPROC(glfwGetProcAddress("glClientAttribDefaultEXT"));
01448     glClientWaitSync = PFNGLCLIENTWAITSYNCPROC(glfwGetProcAddress("glClientWaitSync"));
01449     glClipControl = PFNGLCLIPCONTROLPROC(glfwGetProcAddress("glClipControl"));
01450     glColorFormatNV = PFNGLCOLORFORMATNVPROC(glfwGetProcAddress("glColorFormatNV"));
01451     glColorMask = PFNGLCOLORMASKPROC(glfwGetProcAddress("glColorMask"));
01452     glColorMaski = PFNGLCOLORMASKIPROC(glfwGetProcAddress("glColorMaski"));
01453     glCommandListSegmentsNV =
01454         PFNGLCOMMANDLISTSEGMENTSNVPROC(glfwGetProcAddress("glCommandListSegmentsNV"));
01455     glCompileCommandListNV =
01456         PFNGLCOMPILECOMMANDLISTNVPROC(glfwGetProcAddress("glCompileCommandListNV"));
01457     glCompileShader = PFNGLCOMPILESHADERPROC(glfwGetProcAddress("glCompileShader"));
01458     glCompileShaderIncludeARB =
01459         PFNGLCOMPILESHADERINCLUDEARBPROC(glfwGetProcAddress("glCompileShaderIncludeARB"));
01460     glCompressedMultiTexImage1DEXT =
01461         PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage1DEXT"));
01462     glCompressedMultiTexImage2DEXT =
01463         PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage2DEXT"));
01464     glCompressedMultiTexImage3DEXT =
01465         PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage3DEXT"));
01466     glCompressedMultiTexSubImage1DEXT =
01467         PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage1DEXT"));
01468     glCompressedMultiTexSubImage2DEXT =
01469         PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage2DEXT"));
01470     glCompressedMultiTexSubImage3DEXT =
01471         PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage3DEXT"));
01472     glCompressedTexImage1D =
01473         PFNGLCOMPRESSEDTEXIMAGE1DPROC(glfwGetProcAddress("glCompressedTexImage1D"));
01474     glCompressedTexImage2D =
01475         PFNGLCOMPRESSEDTEXIMAGE2DPROC(glfwGetProcAddress("glCompressedTexImage2D"));
01476     glCompressedTexImage3D =
01477         PFNGLCOMPRESSEDTEXIMAGE3DPROC(glfwGetProcAddress("glCompressedTexImage3D"));
01478     glCompressedTexSubImage1D =
01479         PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTexSubImage1D"));
01480     glCompressedTexSubImage2D =
01481         PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTexSubImage2D"));
01482     glCompressedTexSubImage3D =
01483         PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTexSubImage3D"));
01484     glCompressedTextureImage1DEXT =
01485         PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureImage1DEXT"));
01486     glCompressedTextureImage2DEXT =
01487         PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureImage2DEXT"));
01488     glCompressedTextureImage3DEXT =
01489         PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureImage3DEXT"));
01490     glCompressedTextureSubImage1D =
01491         PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTextureSubImage1D"));
01492     glCompressedTextureSubImage2D =
01493         PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTextureSubImage2D"));
01494     glCompressedTextureSubImage3D =
01495         PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTextureSubImage3D"));
01496     glCompressedTextureSubImage3DEXT =
01497         PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage3DEXT"));
01498     glConservativeRasterParameterfNV =
01499         PFNGLCONSERVATIVERASTERPARAMETERFNVPROC(glfwGetProcAddress("glConservativeRasterParameterfNV"));
01500     glConservativeRasterParameteriINV =
01501         PFNGLCONSERVATIVERASTERPARAMETERINVPROC(glfwGetProcAddress("glConservativeRasterParameteriINV"));
01502     glCopyBufferSubData = PFNGLCOPYBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyBufferSubData"));
01503     glCopyImageSubData = PFNGLCOPYIMAGESUBDATAPROC(glfwGetProcAddress("glCopyImageSubData"));
01504     glCopyMultiTexImage1DEXT =
01505         PFNGLCOPYMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage1DEXT"));
01506     glCopyMultiTexImage2DEXT =
01507         PFNGLCOPYMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage2DEXT"));
01508     glCopyMultiTexSubImage1DEXT =
01509         PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage1DEXT"));
01510     glCopyMultiTexSubImage2DEXT =
01511         PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage2DEXT"));
01512     glCopyMultiTexSubImage3DEXT =
01513         PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage3DEXT"));
01514     glCopyNamedBufferSubData =
01515         PFNGLCOPYNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyNamedBufferSubData"));
01516     glCopyPathNV = PFNGLCOPYPATHNVPROC(glfwGetProcAddress("glCopyPathNV"));
01517     glCopyTexImage1D = PFNGLCOPYTEXIMAGE1DPROC(glfwGetProcAddress("glCopyTexImage1D"));
01518     glCopyTexImage2D = PFNGLCOPYTEXIMAGE2DPROC(glfwGetProcAddress("glCopyTexImage2D"));
01519     glCopyTexSubImage1D = PFNGLCOPYTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCopyTexSubImage1D"));
01520     glCopyTexSubImage2D = PFNGLCOPYTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCopyTexSubImage2D"));
01521     glCopyTexSubImage3D = PFNGLCOPYTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCopyTexSubImage3D"));
01522     glCopyTextureImage1DEXT =
01523         PFNGLCOPYTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureImage1DEXT"));

```

```

0143     glCopyTextureImage2DEXT =
0144     PFNGLCOPYTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureImage2DEXT"));
0145     glCopyTextureSubImage1D =
0146     PFNGLCOPYTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCopyTextureSubImage1D"));
0147     glCopyTextureSubImage1DEXT =
0148     PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage1DEXT"));
0149     glCopyTextureSubImage2D =
0150     PFNGLCOPYTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCopyTextureSubImage2D"));
0151     glCopyTextureSubImage2DEXT =
0152     PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage2DEXT"));
0153     glCopyTextureSubImage3D =
0154     PFNGLCOPYTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCopyTextureSubImage3D"));
0155     glCopyTextureSubImage3DEXT =
0156     PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage3DEXT"));
0157     glCoverFillPathInstancedNV =
0158     PFNGLCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverFillPathInstancedNV"));
0159     glCoverFillPathNV = PFNGLCOVERFILLPATHNVPROC(glfwGetProcAddress("glCoverFillPathNV"));
0160     glCoverStrokePathInstancedNV =
0161     PFNGLCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverStrokePathInstancedNV"));
0162     glCoverStrokePathNV = PFNGLCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glCoverStrokePathNV"));
0163     glCoverageModulationNV =
0164     PFNGLCOVERAGEMODULATIONNVPROC(glfwGetProcAddress("glCoverageModulationNV"));
0165     glCoverageModulationTableNV =
0166     PFNGLCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glCoverageModulationTableNV"));
0167     glCreateBuffers = PFNGLCREATEBUFFERSPROC(glfwGetProcAddress("glCreateBuffers"));
0168     glCreateCommandListsNV =
0169     PFNGLCREATECOMMANDLISTSNVPROC(glfwGetProcAddress("glCreateCommandListsNV"));
0170     glCreateFramebuffers = PFNGLCREATEFRAMEBUFFERSPROC(glfwGetProcAddress("glCreateFramebuffers"));
0171     glCreatePerfQueryINTEL =
0172     PFNGLCREATEPERFQUERYINTELPROC(glfwGetProcAddress("glCreatePerfQueryINTEL"));
0173     glCreateProgram = PFNGLCREATEPROGRAMPROC(glfwGetProcAddress("glCreateProgram"));
0174     glCreateProgramPipelines =
0175     PFNGLCREATEPROGRAMPIPELINESPROC(glfwGetProcAddress("glCreateProgramPipelines"));
0176     glCreateQueries = PFNGLCREATEQUERIESPROC(glfwGetProcAddress("glCreateQueries"));
0177     glCreateRenderbuffers = PFNGLCREATERENDERBUFFERSPROC(glfwGetProcAddress("glCreateRenderbuffers"));
0178     glCreateSamplers = PFNGLCREATESAMPLERSPROC(glfwGetProcAddress("glCreateSamplers"));
0179     glCreateShader = PFNGLCREATESHADERPROC(glfwGetProcAddress("glCreateShader"));
0180     glCreateShaderProgramEXT =
0181     PFNGLCREATESHADERPROGRAMEXTPROC(glfwGetProcAddress("glCreateShaderProgramEXT"));
0182     glCreateShaderProgramv =
0183     PFNGLCREATESHADERPROGRAMVPROC(glfwGetProcAddress("glCreateShaderProgramv"));
0184     glCreateStatesNV = PFNGLCREATESTATESNVPROC(glfwGetProcAddress("glCreateStatesNV"));
0185     glCreateSyncFromCleventARB =
0186     PFNGLCREATESYNCFROMCLEVENTARBPROC(glfwGetProcAddress("glCreateSyncFromCleventARB"));
0187     glCreateTextures = PFNGLCREATETEXTURESPROC(glfwGetProcAddress("glCreateTextures"));
0188     glCreateTransformFeedbacks =
0189     PFNGLCREATETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glCreateTransformFeedbacks"));
0190     glCreateVertexArrays = PFNGLCREATEVERTEXARRAYSPROC(glfwGetProcAddress("glCreateVertexArrays"));
0191     glCullFace = PFNGLCULLFACEPROC(glfwGetProcAddress("glCullFace"));
0192     glDebugMessageCallback =
0193     PFNGLDEBUGMESSAGECALLBACKPROC(glfwGetProcAddress("glDebugMessageCallback"));
0194     glDebugMessageCallbackARB =
0195     PFNGLDEBUGMESSAGECALLBACKARBPROC(glfwGetProcAddress("glDebugMessageCallbackARB"));
0196     glDebugMessageControl =
0197     PFNGLDEBUGMESSAGECONTROLPROC(glfwGetProcAddress("glDebugMessageControl"));
0198     glDebugMessageControlARB =
0199     PFNGLDEBUGMESSAGECONTROLARBPROC(glfwGetProcAddress("glDebugMessageControlARB"));
0200     glDebugMessageInsert =
0201     PFNGLDEBUGMESSAGEINSERTPROC(glfwGetProcAddress("glDebugMessageInsert"));
0202     glDebugMessageInsertARB =
0203     PFNGLDEBUGMESSAGEINSERTARBPROC(glfwGetProcAddress("glDebugMessageInsertARB"));
0204     glDeleteBuffers = PFNGLDELETEBUFFERSPROC(glfwGetProcAddress("glDeleteBuffers"));
0205     glDeleteCommandListsNV =
0206     PFNGLDELETECOMMANDLISTSNVPROC(glfwGetProcAddress("glDeleteCommandListsNV"));
0207     glDeleteFramebuffers = PFNGLDELETEFRAMEBUFFERSPROC(glfwGetProcAddress("glDeleteFramebuffers"));
0208     glDeleteNamedStringARB =
0209     PFNGLDELETENAMEDSTRINGARBPROC(glfwGetProcAddress("glDeleteNamedStringARB"));
0210     glDeletePathsNV = PFNGLDELETEPATHSNVPROC(glfwGetProcAddress("glDeletePathsNV"));
0211     glDeletePerfMonitorsAMD =
0212     PFNGLDELETEPERFMONITORSAMDPROC(glfwGetProcAddress("glDeletePerfMonitorsAMD"));
0213     glDeletePerfQueryINTEL =
0214     PFNGLDELETEPERFQUERYINTELPROC(glfwGetProcAddress("glDeletePerfQueryINTEL"));
0215     glDeleteProgram = PFNGLDELETEPROGRAMPROC(glfwGetProcAddress("glDeleteProgram"));
0216     glDeleteProgramPipelines =
0217     PFNGLDELETEPROGRAMPIPELINESPROC(glfwGetProcAddress("glDeleteProgramPipelines"));
0218     glDeleteQueries = PFNGLDELETEQUERIESPROC(glfwGetProcAddress("glDeleteQueries"));
0219     glDeleteRenderbuffers = PFNGLDELETERENDERBUFFERSPROC(glfwGetProcAddress("glDeleteRenderbuffers"));
0220     glDeleteSamplers = PFNGLDELETESAMPLERSPROC(glfwGetProcAddress("glDeleteSamplers"));
0221     glDeleteShader = PFNGLDELETESHADERPROC(glfwGetProcAddress("glDeleteShader"));
0222     glDeleteStatesNV = PFNGLDELETETESTATESNVPROC(glfwGetProcAddress("glDeleteStatesNV"));
0223     glDeleteSync = PFNGLDELETESYNCPROC(glfwGetProcAddress("glDeleteSync"));
0224     glDeleteTextures = PFNGLDELETETEXTURESPROC(glfwGetProcAddress("glDeleteTextures"));
0225     glDeleteTransformFeedbacks =
0226     PFNGLDELETETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glDeleteTransformFeedbacks"));
0227     glDeleteVertexArrays = PFNGLDELETEVERTEXARRAYSPROC(glfwGetProcAddress("glDeleteVertexArrays"));
0228     glDepthFunc = PFNGLDEPTHFUNCPROC(glfwGetProcAddress("glDepthFunc"));
0229     glDepthMask = PFNGLDEPTHMASKPROC(glfwGetProcAddress("glDepthMask"));
0230     glDepthRange = PFNGLDEPTH RANGEPROC(glfwGetProcAddress("glDepthRange"));
0231     glDepthRangeArrayv = PFNGLDEPTH RANGEARRAYVPROC(glfwGetProcAddress("glDepthRangeArrayv"));

```

```

01552     glDepthRangeIndexed = PFNGLDEPTHRANGEINDEXEDPROC(glfwGetProcAddress("glDepthRangeIndexed"));
01553     glDepthRangef = PFNGLDEPTHRANGEFFPROC(glfwGetProcAddress("glDepthRangef"));
01554     glDetachShader = PFNGLDETACHSHADERPROC(glfwGetProcAddress("glDetachShader"));
01555     glDisable = PFNGLDISABLEPROC(glfwGetProcAddress("glDisable"));
01556     glDisableClientStateIndexedEXT =
01557         PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glDisableClientStateIndexedEXT"));
01558     glDisableClientStateEXT =
01559         PFNGLDISABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glDisableClientStateiEXT"));
01560     glDisableIndexedEXT = PFNGLDISABLEINDEXEDEXTPROC(glfwGetProcAddress("glDisableIndexedEXT"));
01561     glDisableVertexArrayAttrib =
01562         PFNGLDISABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glDisableVertexArrayAttrib"));
01563     glDisableVertexAttribArrayAttribEXT =
01564         PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttribEXT"));
01565     glDisableVertexAttribArrayEXT =
01566         PFNGLDISABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayEXT"));
01567     glDispatchComputeGroupSizeARB =
01568         PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC(glfwGetProcAddress("glDispatchComputeGroupSizeARB"));
01569     glDispatchComputeIndirect =
01570         PFNGLDISPATCHCOMPUTEDIRECTPROC(glfwGetProcAddress("glDispatchComputeIndirect"));
01571     glDrawArrays = PFNGLDRAWARRAYSPROC(glfwGetProcAddress("glDrawArrays"));
01572     glDrawArraysIndirect = PFNGLDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glDrawArraysIndirect"));
01573     glDrawArraysInstanced = PFNGLDRAWARRAYSINSTANCEDPROC(glfwGetProcAddress("glDrawArraysInstanced"));
01574     glDrawArraysInstancedARB =
01575         PFNGLDRAWARRAYSINSTANCEDARBPROC(glfwGetProcAddress("glDrawArraysInstancedARB"));
01576     glDrawArraysInstancedBaseInstance =
01577         PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawArraysInstancedBaseInstance"));
01578     glDrawArraysInstancedEXT =
01579         PFNGLDRAWARRAYSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawArraysInstancedEXT"));
01580     glDrawBuffer = PFNGLDRAWBUFFERPROC(glfwGetProcAddress("glDrawBuffer"));
01581     glDrawBuffers = PFNGLDRAWBUFFERSPROC(glfwGetProcAddress("glDrawBuffers"));
01582     glDrawCommandsAddressNV =
01583         PFNGLDRAWCOMMANDSADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsAddressNV"));
01584     glDrawCommandsNV = PFNGLDRAWCOMMANDSNVPROC(glfwGetProcAddress("glDrawCommandsNV"));
01585     glDrawCommandsStatesAddressNV =
01586         PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsStatesAddressNV"));
01587     glDrawCommandsStatesNV =
01588         PFNGLDRAWCOMMANDSSTATESNVPROC(glfwGetProcAddress("glDrawCommandsStatesNV"));
01589     glDrawElements = PFNGLDRAWELEMENTSPROC(glfwGetProcAddress("glDrawElements"));
01590     glDrawElementsBaseVertex =
01591         PFNGLDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsBaseVertex"));
01592     glDrawElementsIndirect =
01593         PFNGLDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glDrawElementsIndirect"));
01594     glDrawElementsInstanced =
01595         PFNGLDRAWELEMENTSINSTANCEDPROC(glfwGetProcAddress("glDrawElementsInstanced"));
01596     glDrawElementsInstancedARB =
01597         PFNGLDRAWELEMENTSINSTANCEDARBPROC(glfwGetProcAddress("glDrawElementsInstancedARB"));
01598     glDrawElementsInstancedBaseInstance =
01599         PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseInstance"));
01600     glDrawElementsInstancedBaseVertex =
01601         PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertex"));
01602     glDrawElementsInstancedBaseVertexBuffer =
01603         PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertexBuffer"));
01604     glDrawElementsInstancedEXT =
01605         PFNGLDRAWELEMENTSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawElementsInstancedEXT"));
01606     glDrawRangeElements = PFNGLDRAWRANGEELEMENTSPROC(glfwGetProcAddress("glDrawRangeElements"));
01607     glDrawRangeElementsBaseVertex =
01608         PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawRangeElementsBaseVertex"));
01609     glDrawTransformFeedback =
01610         PFNGLDRAWTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glDrawTransformFeedback"));
01611     glDrawTransformFeedbackInstanced =
01612         PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackInstanced"));
01613     glDrawTransformFeedbackStream =
01614         PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC(glfwGetProcAddress("glDrawTransformFeedbackStream"));
01615     glDrawTransformFeedbackStreamInstanced =
01616         PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackStreamInstanced"));
01617     glDrawVkImageNV = PFNGLDRAWVKIMAGENVPROC(glfwGetProcAddress("glDrawVkImageNV"));
01618     glEdgeFlagFormatNV = PFNGLEDGEFLAGFORMATNVPROC(glfwGetProcAddress("glEdgeFlagFormatNV"));
01619     glEnable = PFNGLENABLEPROC(glfwGetProcAddress("glEnable"));
01620     glEnableClientStateIndexedEXT =
01621         PFNGLENABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glEnableClientStateIndexedEXT"));
01622     glEnableClientStateiEXT =
01623         PFNGLENABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glEnableClientStateiEXT"));
01624     glEnableIndexedEXT = PFNGLENABLEINDEXEDEXTPROC(glfwGetProcAddress("glEnableIndexedEXT"));
01625     glEnableVertexAttribArrayAttrib =
01626         PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttrib"));
01627     glEnableVertexAttribArrayAttribEXT =
01628         PFNGLENABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttribEXT"));
01629     glEnableVertexAttribArrayEXT =
01630         PFNGLENABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayEXT"));
01631     glEnableVertexAttribArrayAttribArray =
01632         PFNGLENABLEVERTEXARRAYATTRIBARRAYPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttribArray"));
01633     glEnablei = PFNGLENABLEIIPROC(glfwGetProcAddress("glEnablei"));
01634     glEndConditionalRender =

```

```

01606 PFNGLENDCONDITIONALRENDERPROC(glfwGetProcAddress("glEndConditionalRender"));
01607     glEndConditionalRenderNV =
01608 PFNGLENDCONDITIONALRENDERNVPROC(glfwGetProcAddress("glEndConditionalRenderNV"));
01609     glEndPerfMonitorAMD = PFNGLENDPERFMONITORAMDPROC(glfwGetProcAddress("glEndPerfMonitorAMD"));
01610     glEndPerfQueryINTEL = PFNGLENDPERFQUERYINTELPROC(glfwGetProcAddress("glEndPerfQueryINTEL"));
01611     glEndQuery = PFNGLENDQUERYPROC(glfwGetProcAddress("glEndQuery"));
01612     glEndQueryIndexed = PFNGLENDQUERYINDEXEDPROC(glfwGetProcAddress("glEndQueryIndexed"));
01613     glEndTransformFeedback =
01614 PFNGLENDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glEndTransformFeedback"));
01615     glEvaluateDepthValuesARB =
01616 PFNGLEVALUATEDEPTHVALUESARBPROC(glfwGetProcAddress("glEvaluateDepthValuesARB"));
01617     glFenceSync = PFNGLFENCESYNCPROC(glfwGetProcAddress("glFenceSync"));
01618     glFinish = PFNGLFINISHPROC(glfwGetProcAddress("glFinish"));
01619     glFlush = PFNGLFLUSHPROC(glfwGetProcAddress("glFlush"));
01620     glFlushMappedBufferRange =
01621 PFNGLFLUSHMAPPEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedBufferRange"));
01622     glFlushMappedNamedBufferRange =
01623 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedNamedBufferRange"));
01624     glFlushMappedNamedBufferRangeEXT =
01625 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glFlushMappedNamedBufferRangeEXT"));
01626     glFogCoordFormatNV = PFNGLFOGCOORDFORMATNVPROC(glfwGetProcAddress("glFogCoordFormatNV"));
01627     glFragmentCoverageColorNV =
01628 PFNGLFRAGMENTCOVERAGECOLORNVPROC(glfwGetProcAddress("glFragmentCoverageColorNV"));
01629     glFramebufferDrawBufferEXT =
01630 PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC(glfwGetProcAddress("glFramebufferDrawBufferEXT"));
01631     glFramebufferDrawBuffersEXT =
01632 PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC(glfwGetProcAddress("glFramebufferDrawBuffersEXT"));
01633     glFramebufferParameteri =
01634 PFNGLFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glFramebufferParameteri"));
01635     glFramebufferReadBufferEXT =
01636 PFNGLFRAMEBUFERREADBUFFEREXTPROC(glfwGetProcAddress("glFramebufferReadBufferEXT"));
01637     glFramebufferRenderbuffer =
01638 PFNGLFRAMEBUFERRENDERBUFFERPROC(glfwGetProcAddress("glFramebufferRenderbuffer"));
01639     glFramebufferSampleLocationsfvARB =
01640 PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvARB"));
01641     glFramebufferSampleLocationsfvNV =
01642 PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvNV"));
01643     glFramebufferTexture = PFNGLFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glFramebufferTexture"));
01644     glFramebufferTexture1D =
01645 PFNGLFRAMEBUFFERTEXTURE1DPROC(glfwGetProcAddress("glFramebufferTexture1D"));
01646     glFramebufferTexture2D =
01647 PFNGLFRAMEBUFFERTTEXTURE2DPROC(glfwGetProcAddress("glFramebufferTexture2D"));
01648     glFramebufferTexture3D =
01649 PFNGLFRAMEBUFFERTTEXTURE3DPROC(glfwGetProcAddress("glFramebufferTexture3D"));
01650     glFramebufferTextureARB =
01651 PFNGLFRAMEBUFFERTTEXTUREARBPROC(glfwGetProcAddress("glFramebufferTextureARB"));
01652     glFramebufferTextureFaceARB =
01653 PFNGLFRAMEBUFFERTTEXTUREFACEARBPROC(glfwGetProcAddress("glFramebufferTextureFaceARB"));
01654     glFramebufferTextureLayer =
01655 PFNGLFRAMEBUFFERTTEXTURELAYERPROC(glfwGetProcAddress("glFramebufferTextureLayer"));
01656     glFramebufferTextureLayerARB =
01657 PFNGLFRAMEBUFFERTTEXTURELAYERARBPROC(glfwGetProcAddress("glFramebufferTextureLayerARB"));
01658     glFramebufferTextureMultiviewOVR =
01659 PFNGLFRAMEBUFFERTTEXTUREMULTIVIEWOVRPROC(glfwGetProcAddress("glFramebufferTextureMultiviewOVR"));
01660     glFrontFace = PFNGLFRONTFACEPROC(glfwGetProcAddress("glFrontFace"));
01661     glGenBuffers = PFNGLGENBUFFERSPROC(glfwGetProcAddress("glGenBuffers"));
01662     glGenFramebuffers = PFNGLGENFRAMEBUFFERSPROC(glfwGetProcAddress("glGenFramebuffers"));
01663     glGenPathsNV = PFNGLGENPATHSNVPROC(glfwGetProcAddress("glGenPathsNV"));
01664     glGenPerfMonitorsAMD = PFNGLGENPERFMONITORSAMDPROC(glfwGetProcAddress("glGenPerfMonitorsAMD"));
01665     glGenProgramPipelines = PFNGLGENPROGRAMPIPELINESPROC(glfwGetProcAddress("glGenProgramPipelines"));
01666     glGenQueries = PFNGLGENQUERIESPROC(glfwGetProcAddress("glGenQueries"));
01667     glGenRenderbuffers = PFNGLGENRENDERBUFFERSPROC(glfwGetProcAddress("glGenRenderbuffers"));
01668     glGenSamplers = PFNGLGENSAMPLERSPROC(glfwGetProcAddress("glGenSamplers"));
01669     glGenTextures = PFNGLGENTEXTURESPROC(glfwGetProcAddress("glGenTextures"));
01670     glGenTransformFeedbacks =
01671 PFNGLGENTRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glGenTransformFeedbacks"));
01672     glGenVertexArrays = PFNGLGENVERTEXARRAYSPROC(glfwGetProcAddress("glGenVertexArrays"));
01673     glGenerateMipmap = PFNGLGENERATEMIPMAPPROC(glfwGetProcAddress("glGenerateMipmap"));
01674     glGenerateMultiTexMipmapEXT =
01675 PFNGLGENERATEMULTITEXMIPMAPEXTPROC(glfwGetProcAddress("glGenerateMultiTexMipmapEXT"));
01676     glGenerateTextureMipmap =
01677 PFNGLGENERATETEXTUREMIPMAPPROC(glfwGetProcAddress("glGenerateTextureMipmap"));
01678     glGenerateTextureMipmapEXT =
01679 PFNGLGENERATETEXTUREMIPMAPEXTPROC(glfwGetProcAddress("glGenerateTextureMipmapEXT"));
01680     glGetActiveAtomicCounterBufferiv =
01681 PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC(glfwGetProcAddress("glGetActiveAtomicCounterBufferiv"));
01682     glGetActiveAttrib = PFNGLGETACTIVEATTRIBPROC(glfwGetProcAddress("glGetActiveAttrib"));
01683     glGetActiveSubroutineName =
01684 PFNGLGETACTIVESUBROUTINENAMEPROC(glfwGetProcAddress("glGetActiveSubroutineName"));
01685     glGetActiveSubroutineUniformName =
01686 PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveSubroutineUniformName"));
01687     glGetActiveSubroutineUniformiv =
01688 PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC(glfwGetProcAddress("glGetActiveSubroutineUniformiv"));
01689     glGetActiveUniform = PFNGLGETACTIVEUNIFORMPROC(glfwGetProcAddress("glGetActiveUniform"));
01690     glGetActiveUniformBlockName =
01691 PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC(glfwGetProcAddress("glGetActiveUniformBlockName"));
01692     glGetActiveUniformBlockiv =

```

```

0161     PFNGLGETACTIVEUNIFORMBLOCKIVPROC(glfwGetProcAddress("glGetActiveUniformBlockiv"));
0162     glGetActiveUniformName =
0163     PFNGLGETACTIVEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveUniformName"));
0164     glGetAttachedShaders = PFNGLGETATTACHEDSHADERSPROC(glfwGetProcAddress("glGetAttachedShaders"));
0165     glGetAttribLocation = PFNGLGETATTRIBLOCATIONPROC(glfwGetProcAddress("glGetAttribLocation"));
0166     glGetBooleanIndexedvEXT =
0167     PFNGLGETBOOLEANINDEXEDVEXTPROC(glfwGetProcAddress("glGetBooleanIndexedvEXT"));
0168     glGetBooleani_v = PFNGLGETBOOLEANI_VPROC(glfwGetProcAddress("glGetBooleani_v"));
0169     glGetBooleanv = PFNGLGETBOOLEANVPROC(glfwGetProcAddress("glGetBooleanv"));
0170     glGetBufferParameteri64v =
0171     PFNGLGETBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetBufferParameteri64v"));
0172     glGetBufferParameteriv =
0173     PFNGLGETBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetBufferParameteriv"));
0174     glGetBufferParameterui64NV =
0175     PFNGLGETBUFFERPARAMETERUI64NVPROC(glfwGetProcAddress("glGetBufferParameterui64NV"));
0176     glGetBufferPointerv = PFNGLGETBUFFERPOINTERVPROC(glfwGetProcAddress("glGetBufferPointerv"));
0177     glGetBufferData = PFNGLGETBUFFERSUBDATAPROC(glfwGetProcAddress("glGetBufferData"));
0178     glGetCommandHeaderNV = PFNGLGETCOMMANDHEADERNVPROC(glfwGetProcAddress("glGetCommandHeaderNV"));
0179     glGetCompressedMultiTexImageEXT =
0180     PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedMultiTexImageEXT"));
0181     glGetCompressedTexImage =
0182     PFNGLGETCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetCompressedTexImage"));
0183     glGetCompressedTextureImage =
0184     PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureImage"));
0185     glGetCompressedTextureImageEXT =
0186     PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedTextureImageEXT"));
0187     glGetCompressedTextureSubImage =
0188     PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureSubImage"));
0189     glGetCoverageModulationTableNV =
0190     PFNGLGETCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glGetCoverageModulationTableNV"));
0191     glGetDebugMessageLog = PFNGLGETDEBUGMESSAGELOGPROC(glfwGetProcAddress("glGetDebugMessageLog"));
0192     glGetDebugMessageLogARB =
0193     PFNGLGETDEBUGMESSAGELOGARBPROC(glfwGetProcAddress("glGetDebugMessageLogARB"));
0194     glGetDoubleIndexedvEXT =
0195     PFNGLGETDOUBLEINDEXEDVEXTPROC(glfwGetProcAddress("glGetDoubleIndexedvEXT"));
0196     glGetDoublei_v = PFNGLGETDOUBLEI_VPROC(glfwGetProcAddress("glGetDoublei_v"));
0197     glGetDoublei_vEXT = PFNGLGETDOUBLEI_VEXTPROC(glfwGetProcAddress("glGetDoublei_vEXT"));
0198     glGetDoublev = PFNGLGETDOUBLEVPROC(glfwGetProcAddress("glGetDoublev"));
0199     glGetError = PFNGLGETERRORPROC(glfwGetProcAddress("glGetError"));
0200     glGetFirstPerfQueryIdINTEL =
0201     PFNGLGETFIRSTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetFirstPerfQueryIdINTEL"));
0202     glGetFloatIndexedvEXT = PFNGLGETFLOATINDEXEDVEXTPROC(glfwGetProcAddress("glGetFloatIndexedvEXT"));
0203     glGetFloati_v = PFNGLGETFLOATI_VPROC(glfwGetProcAddress("glGetFloati_v"));
0204     glGetFloati_vEXT = PFNGLGETFLOATI_VEXTPROC(glfwGetProcAddress("glGetFloati_vEXT"));
0205     glGetFloatv = PFNGLGETFLOATVPROC(glfwGetProcAddress("glGetFloatv"));
0206     glGetFragDataIndex = PFNGLGETFRAGDATAINDEXPROC(glfwGetProcAddress("glGetFragDataIndex"));
0207     glGetFragDataLocation = PFNGLGETFRAGDATALOCATIONPROC(glfwGetProcAddress("glGetFragDataLocation"));
0208     glGetFramebufferAttachmentParameteri =
0209     PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferAttachmentParameteri"));
0210     glGetFramebufferParameteri =
0211     PFNGLGETFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferParameteri"));
0212     glGetFramebufferParameteriEXT =
0213     PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetFramebufferParameteriEXT"));
0214     glGetGraphicsResetStatus =
0215     PFNGLGETGRAPHICSRESETSTATUSPROC(glfwGetProcAddress("glGetGraphicsResetStatus"));
0216     glGetGraphicsResetStatusARB =
0217     PFNGLGETGRAPHICSRESETSTATUSARBPROC(glfwGetProcAddress("glGetGraphicsResetStatusARB"));
0218     glGetImageHandleARB = PFNGLGETIMAGEHANDLEARBPROC(glfwGetProcAddress("glGetImageHandleARB"));
0219     glGetImageHandleNV = PFNGLGETIMAGEHANDLENVPROC(glfwGetProcAddress("glGetImageHandleNV"));
0220     glGetInteger64i_v = PFNGLGETINTEGER64I_VPROC(glfwGetProcAddress("glGetInteger64i_v"));
0221     glGetInteger64v = PFNGLGETINTEGER64VPROC(glfwGetProcAddress("glGetInteger64v"));
0222     glGetIntegerIndexedvEXT =
0223     PFNGLGETINTEGERINDEXEDVEXTPROC(glfwGetProcAddress("glGetIntegerIndexedvEXT"));
0224     glGetIntegeri_v = PFNGLGETINTEGERI_VPROC(glfwGetProcAddress("glGetIntegeri_v"));
0225     glGetIntegerui64i_vNV = PFNGLGETINTEGERUI64I_NVNPROC(glfwGetProcAddress("glGetIntegerui64i_vNV"));
0226     glGetIntegerui64vNV = PFNGLGETINTEGERUI64VNVPROC(glfwGetProcAddress("glGetIntegerui64vNV"));
0227     glGetIntegerv = PFNGLGETINTEGERVPROC(glfwGetProcAddress("glGetIntegerv"));
0228     glGetInternalformatSampleivNV =
0229     PFNGLGETINTERNALFORMATSAMPLEIVNPROC(glfwGetProcAddress("glGetInternalformatSampleivNV"));
0230     glGetInternalformati64v =
0231     PFNGLGETINTERNALFORMATI64VPROC(glfwGetProcAddress("glGetInternalformati64v"));
0232     glGetInternalformativ = PFNGLGETINTERNALFORMATIVPROC(glfwGetProcAddress("glGetInternalformativ"));
0233     glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
0234     glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
0235     glGetMultiTexGendvEXT = PFNGLGETMULTITEXGENDEXTPROC(glfwGetProcAddress("glGetMultiTexGendvEXT"));
0236     glGetMultiTexGenfvEXT = PFNGLGETMULTITEXGENFVEXTPROC(glfwGetProcAddress("glGetMultiTexGenfvEXT"));
0237     glGetMultiTexGenivEXT = PFNGLGETMULTITEXGENIVEXTPROC(glfwGetProcAddress("glGetMultiTexGenivEXT"));
0238     glGetMultiTexImageEXT = PFNGLGETMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetMultiTexImageEXT"));
0239     glGetMultiTexLevelParameterfvEXT =
0240     PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterfvEXT"));
0241     glGetMultiTexLevelParameterivEXT =
0242     PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterivEXT"));
0243     glGetMultiTexParameteriivEXT =
0244     PFNGLGETMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIivEXT"));
0245     glGetMultiTexParameterIiivEXT =
0246     PFNGLGETMULTITEXPARAMETERIIIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIiivEXT"));

```

```

01721     glGetMultiTexParameterfvEXT =
01722     PFNGLGETMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterfvEXT"));
01723     glGetMultiTexParameterivEXT =
01724     PFNGLGETMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterivEXT"));
01725     glGetMultisamplefv = PFNGLGETMULTISAMPLEFVPROC(glfwGetProcAddress("glGetMultisamplefv"));
01726     glGetNamedBufferParameteri64v =
01727     PFNGLGETNAMEDBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetNamedBufferParameteri64v"));
01728     glGetNamedBufferParameteriv =
01729     PFNGLGETNAMEDBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedBufferParameteriv"));
01730     glGetNamedBufferParameterEXT =
01731     PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedBufferParameterEXT"));
01732     glGetNamedBufferParameterui64NV =
01733     PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC(glfwGetProcAddress("glGetNamedBufferParameterui64NV"));
01734     glGetNamedBufferPointerv =
01735     PFNGLGETNAMEDBUFFERPOINTERVPROC(glfwGetProcAddress("glGetNamedBufferPointerv"));
01736     glGetNamedBufferPointervEXT =
01737     PFNGLGETNAMEDBUFFERPOINTERVEXTPROC(glfwGetProcAddress("glGetNamedBufferPointervEXT"));
01738     glGetNamedBufferSubData =
01739     PFNGLGETNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glGetNamedBufferSubData"));
01740     glGetNamedBufferSubDataEXT =
01741     PFNGLGETNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glGetNamedBufferSubDataEXT"));
01742     glGetNamedFramebufferAttachmentParameteriv =
01743     PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameteriv"));
01744     glGetNamedFramebufferAttachmentParameterivEXT =
01745     PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameterivEXT"));
01746     glGetNamedFramebufferParameteriv =
01747     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriv"));
01748     glGetNamedFramebufferParameterivEXT =
01749     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterivEXT"));
01750     glGetNamedFramebufferParameteriIivEXT =
01751     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriIivEXT"));
01752     glGetNamedFramebufferParameteriIuivEXT =
01753     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriIuivEXT"));
01754     glGetNamedFramebufferParameterdvEXT =
01755     PFNGLGETNAMEDFRAMEBUFFERPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterdvEXT"));
01756     glGetNamedFramebufferParameterfvEXT =
01757     PFNGLGETNAMEDFRAMEBUFFERPARAMETERFVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterfvEXT"));
01758     glGetNamedFramebufferParameterivEXT =
01759     PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterivEXT"));
01760     glGetNamedProgramLocalParameterIiivEXT =
01761     PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIiivEXT"));
01762     glGetNamedProgramLocalParameterIiuvEXT =
01763     PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIiuvEXT"));
01764     glGetNamedProgramLocalParameterIdINTEL =
01765     PFNGLGETNEXTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetNextPerfQueryIdINTEL"));
01766     glGetObjectLabel = PFNGLGETOBJECTLABELPROC(glfwGetProcAddress("glGetObjectLabel"));
01767     glGetObjectLabelEXT = PFNGLGETOBJECTLABELEXTPROC(glfwGetProcAddress("glGetObjectLabelEXT"));
01768     glGetObjectPtrLabel = PFNGLGETOBJECTPTRLABELPROC(glfwGetProcAddress("glGetObjectPtrLabel"));
01769     glGetPathCommandsNV = PFNGLGETPATHCOMMANDSNVPROC(glfwGetProcAddress("glGetPathCommandsNV"));
01770     glGetPathCoordsNV = PFNGLGETPATHCOORDSNVPROC(glfwGetProcAddress("glGetPathCoordsNV"));
01771     glGetPathDashArrayNV = PFNGLGETPATHDASHARRAYNVPROC(glfwGetProcAddress("glGetPathDashArrayNV"));
01772     glGetPathLengthNV = PFNGLGETPATHLENGTHNVPROC(glfwGetProcAddress("glGetPathLengthNV"));
01773     glGetPathMetricRangeNV =
01774     PFNGLGETPATHMETRICRANGEPROC(glfwGetProcAddress("glGetPathMetricRangeNV"));
01775     glGetPathMetricsNV = PFNGLGETPATHMETRICSNVPROC(glfwGetProcAddress("glGetPathMetricsNV"));
01776     glGetPathParameterfvNV =
01777     PFNGLGETPATHPARAMETERFVNVPROC(glfwGetProcAddress("glGetPathParameterfvNV"));
01778     glGetPathParameterivNV =
01779     PFNGLGETPATHPARAMETERIVNVPROC(glfwGetProcAddress("glGetPathParameterivNV"));
01780     glGetPathSpacingNV = PFNGLGETPATHSPACINGNVPROC(glfwGetProcAddress("glGetPathSpacingNV"));
01781     glGetPerfCounterInfoINTEL =
01782     PFNGLGETPERFCOUNTERINFOINTELPROC(glfwGetProcAddress("glGetPerfCounterInfoINTEL"));
01783     glGetPerfMonitorCounterDataAMD =
01784     PFNGLGETPERFMONITORCOUNTERDATAAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterDataAMD"));
01785     glGetPerfMonitorCounterInfoAMD =
01786     PFNGLGETPERFMONITORCOUNTERINFOAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterInfoAMD"));
01787     glGetPerfMonitorCounterStringAMD =
01788     PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterStringAMD"));
01789     glGetPerfMonitorCountersAMD =
01790     PFNGLGETPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glGetPerfMonitorCountersAMD"));
01791     glGetPerfMonitorGroupStringAMD =
01792     PFNGLGETPERFMONITORGROUPSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupStringAMD"));
01793     glGetPerfMonitorGroupsAMD =
01794     PFNGLGETPERFMONITORGROUPSAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupsAMD"));
01795     glGetPerfQueryDataINTEL =
01796     PFNGLGETPERFQUERYDATAINTELPROC(glfwGetProcAddress("glGetPerfQueryDataINTEL"));
01797     glGetPerfQueryIdByNameINTEL =
01798     PFNGLGETPERFQUERYIDBYNAMEINTELPROC(glfwGetProcAddress("glGetPerfQueryIdByNameINTEL"));
01799     glGetPerfQueryInfoINTEL =
01800     PFNGLGETPERFQUERYINFOINTELPROC(glfwGetProcAddress("glGetPerfQueryInfoINTEL"));
01801     glGetPointerIndexedvEXT =
01802     PFNGLGETPOINTERINDEXEDVEXTPROC(glfwGetProcAddress("glGetPointerIndexedvEXT"));
01803     glGetPointeri_vEXT = PFNGLGETPOINTERI_VEXTPROC(glfwGetProcAddress("glGetPointeri_vEXT"));

```

```

01771     glGetPointerv = PFNGLGETPOINTERVPROC(glfwGetProcAddress("glGetPointerv"));
01772     glGetProgramBinary = PFNGLGETPROGRAMBINARYPROC(glfwGetProcAddress("glGetProgramBinary"));
01773     glGetProgramInfoLog = PFNGLGETPROGRAMINFOLOGPROC(glfwGetProcAddress("glGetProgramInfoLog"));
01774     glGetProgramInterfaceiv =
01775         PFNGLGETPROGRAMINTERFACEIVPROC(glfwGetProcAddress("glGetProgramInterfaceiv"));
01776     glGetProgramPipelineInfoLog =
01777         PFNGLGETPROGRAMPIPELINEINFOLOGPROC(glfwGetProcAddress("glGetProgramPipelineInfoLog"));
01778     glGetProgramPipelineiv =
01779         PFNGLGETPROGRAMPIPELINEIVPROC(glfwGetProcAddress("glGetProgramPipelineiv"));
01780     glGetProgramResourceIndex =
01781         PFNGLGETPROGRAMRESOURCEINDEXPROC(glfwGetProcAddress("glGetProgramResourceIndex"));
01782     glGetProgramResourceLocation =
01783         PFNGLGETPROGRAMRESOURCELOCATIONPROC(glfwGetProcAddress("glGetProgramResourceLocation"));
01784     glGetProgramResourceLocationIndex =
01785         PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC(glfwGetProcAddress("glGetProgramResourceLocationIndex"));
01786     glGetProgramResourceName =
01787         PFNGLGETPROGRAMRESOURCENAMEPROC(glfwGetProcAddress("glGetProgramResourceName"));
01788     glGetProgramResourcefvNV =
01789         PFNGLGETPROGRAMRESOURCEFVNVPROC(glfwGetProcAddress("glGetProgramResourcefvNV"));
01790     glGetProgramResourceiv =
01791         PFNGLGETPROGRAMRESOURCEIVPROC(glfwGetProcAddress("glGetProgramResourceiv"));
01792     glGetProgramStageiv =
01793         PFNGLGETPROGRAMSTAGEIVPROC(glfwGetProcAddress("glGetProgramStageiv"));
01794     glGetQueryBufferObjecti64v =
01795         PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
01796     glGetQueryBufferObjectiv =
01797         PFNGLGETQUERYBUFFEROBJECTIVPROC(glfwGetProcAddress("glGetQueryBufferObjectiv"));
01798     glGetQueryBufferObjectui64v =
01799         PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
01800     glGetQueryBufferObjectuiv =
01801         PFNGLGETQUERYBUFFEROBJECTUIVPROC(glfwGetProcAddress("glGetQueryBufferObjectuiv"));
01802     glGetQueryIndexediv =
01803         PFNGLGETQUERYINDEXEDIVPROC(glfwGetProcAddress("glGetQueryIndexediv"));
01804     glGetQueryObjecti64v =
01805         PFNGLGETQUERYOBJECTI64VPROC(glfwGetProcAddress("glGetQueryObjecti64v"));
01806     glGetQueryObjectiv =
01807         PFNGLGETQUERYOBJECTIVPROC(glfwGetProcAddress("glGetQueryObjectiv"));
01808     glGetQueryObjectui64v =
01809         PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
01810     glGetQueryObjectuiv =
01811         PFNGLGETQUERYOBJECTUIVPROC(glfwGetProcAddress("glGetQueryObjectuiv"));
01812     glGetRenderbufferParameteriv =
01813         PFNGLGETRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetRenderbufferParameteriv"));
01814     glGetSamplerParameterIiiv =
01815         PFNGLGETSAMPLERPARAMETERIIIVPROC(glfwGetProcAddress("glGetSamplerParameterIiiv"));
01816     glGetSamplerParameterIuiv =
01817         PFNGLGETSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glGetSamplerParameterIuiv"));
01818     glGetSamplerParameterfv =
01819         PFNGLGETSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glGetSamplerParameterfv"));
01820     glGetSamplerParameteriv =
01821         PFNGLGETSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glGetSamplerParameteriv"));
01822     glGetShaderInfoLog =
01823         PFNGLGETSHADERINFOLOGPROC(glfwGetProcAddress("glGetShaderInfoLog"));
01824     glGetShaderPrecisionFormat =
01825         PFNGLGETSHADERPRECISIONFORMATPROC(glfwGetProcAddress("glGetShaderPrecisionFormat"));
01826     glGetShaderSource =
01827         PFNGLGETSHADERSOURCEPROC(glfwGetProcAddress("glGetShaderSource"));
01828     glGetShaderiv =
01829         PFNGLGETSHADERIVPROC(glfwGetProcAddress("glGetShaderiv"));
01830     glGetStageIndexNV =
01831         PFNGLGETSTAGEINDEXNVPROC(glfwGetProcAddress("glGetStageIndexNV"));
01832     glGetString =
01833         PFNGLGETSTRINGPROC(glfwGetProcAddress("glGetString"));
01834     glGetStringi =
01835         PFNGLGETSTRINGIPROC(glfwGetProcAddress("glGetStringi"));
01836     glGetSubroutineIndex =
01837         PFNGLGETSUBROUTINEINDEXPROC(glfwGetProcAddress("glGetSubroutineIndex"));
01838     glGetSubroutineUniform =
01839         PFNGLGETSUBROUTINEUNIFORMPROC(glfwGetProcAddress("glGetSubroutineUniform"));
01840     glGetSynciv =
01841         PFNGLGETSYNCIVPROC(glfwGetProcAddress("glGetSynciv"));
01842     glGetTexImage =
01843         PFNGLGETTEXIMAGEPROC(glfwGetProcAddress("glGetTexImage"));
01844     glGetTexLevelParameterfv =
01845         PFNGLGETTEXLEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTexLevelParameterfv"));
01846     glGetTexLevelParameteriv =
01847         PFNGLGETTEXLEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTexLevelParameteriv"));
01848     glGetTexParameterIiiv =
01849         PFNGLGETTEXPARAMETERIIIVPROC(glfwGetProcAddress("glGetTexParameterIiiv"));
01850     glGetTexParameterIuiv =
01851         PFNGLGETTEXPARAMETERIUIVPROC(glfwGetProcAddress("glGetTexParameterIuiv"));
01852     glGetTexParameterfv =
01853         PFNGLGETTEXPARAMETERFVPROC(glfwGetProcAddress("glGetTexParameterfv"));
01854     glGetTexParameteriv =
01855         PFNGLGETTEXPARAMETERIVPROC(glfwGetProcAddress("glGetTexParameteriv"));
01856     glGetTextureHandleARB =
01857         PFNGLGETTEXTUREHANDLEARBPROC(glfwGetProcAddress("glGetTextureHandleARB"));
01858     glGetTextureHandleNV =
01859         PFNGLGETTEXTUREHANDLENVPROC(glfwGetProcAddress("glGetTextureHandleNV"));
01860     glGetTextureImage =
01861         PFNGLGETTEXTUREIMAGEPROC(glfwGetProcAddress("glGetTextureImage"));
01862     glGetTextureImageEXT =
01863         PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTextureImageEXT"));
01864     glGetTextureLevelParameterfv =
01865         PFNGLGETTEXTURELEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTextureLevelParameterfv"));
01866     glGetTextureLevelParameterfvEXT =
01867         PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterfvEXT"));
01868     glGetTextureLevelParameteriv =
01869         PFNGLGETTEXTURELEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTextureLevelParameteriv"));
01870     glGetTextureLevelParameterivEXT =
01871         PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterivEXT"));
01872     glGetTextureParameterIiiv =
01873         PFNGLGETTEXTUREPARAMETERIIIVPROC(glfwGetProcAddress("glGetTextureParameterIiiv"));
01874     glGetTextureParameterIiivEXT =
01875         PFNGLGETTEXTUREPARAMETERIIIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIiivEXT"));
01876     glGetTextureParameterIuiv =
01877         PFNGLGETTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glGetTextureParameterIuiv"));
01878     glGetTextureParameterIuivEXT =
01879         PFNGLGETTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIuivEXT"));

```

```

01829     PFNGLGETTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIuivEXT"));
01830     glGetTextureParameterfv =
01831     PFNGLGETTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glGetTextureParameterfv"));
01832     glGetTextureParameterfvEXT =
01833     PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
01834     glGetTextureParameteriv =
01835     PFNGLGETTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glGetTextureParameteriv"));
01836     glGetTextureParameterivEXT =
01837     PFNGLGETTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureParameterivEXT"));
01838     glGetTextureSamplerHandleARB =
01839     PFNGLGETTEXTURESAMPLERHANDLEARBPROC(glfwGetProcAddress("glGetTextureSamplerHandleARB"));
01840     glGetTextureSamplerHandleEnv =
01841     PFNGLGETTEXTURESAMPLERHANDLENVPROC(glfwGetProcAddress("glGetTextureSamplerHandleNV"));
01842     glGetTextureSubImage = PFNGLGETTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetTextureSubImage"));
01843     glGetTransformFeedbackVarying =
01844     PFNGLGETTRANSFORMFEEDBACKVARYINGPROC(glfwGetProcAddress("glGetTransformFeedbackVarying"));
01845     glGetTransformFeedbacki64_v =
01846     PFNGLGETTRANSFORMFEEDBACKI64_VPROC(glfwGetProcAddress("glGetTransformFeedbacki64_v"));
01847     glGetTransformFeedbacki_v =
01848     PFNGLGETTRANSFORMFEEDBACKI_VPROC(glfwGetProcAddress("glGetTransformFeedbacki_v"));
01849     glGetUniformBlockIndex =
01850     PFNGLGETUNIFORMBLOCKINDEXPROC(glfwGetProcAddress("glGetUniformBlockIndex"));
01851     glGetUniformLocation = PFNGLGETUNIFORMINDICESPROC(glfwGetProcAddress("glGetUniformIndices"));
01852     glGetUniformLocation = PFNGLGETUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetUniformLocation"));
01853     glGetUniformLocation = PFNGLGETUNIFORMSUBROUTINEUIVPROC(glfwGetProcAddress("glGetUniformSubroutineuiv"));
01854     glGetUniformLocation = PFNGLGETUNIFORMMDVPROC(glfwGetProcAddress("glGetUniformmdv"));
01855     glGetUniformLocation = PFNGLGETUNIFORMMFVPROC(glfwGetProcAddress("glGetUniformfv"));
01856     glGetUniformLocation = PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64vARB"));
01857     glGetUniformLocation = PFNGLGETUNIFORMI64NVPROC(glfwGetProcAddress("glGetUniformi64vNV"));
01858     glGetUniformLocation = PFNGLGETUNIFORMIVPROC(glfwGetProcAddress("glGetUniformiv"));
01859     glGetUniformLocation = PFNGLGETUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetUniformui64vARB"));
01860     glGetUniformLocation = PFNGLGETUNIFORMUI64NVPROC(glfwGetProcAddress("glGetUniformui64vNV"));
01861     glGetUniformLocation = PFNGLGETUNIFORMUIVPROC(glfwGetProcAddress("glGetUniformuiv"));
01862     glGetUniformLocation = PFNGLGETVERTEXARRAYINDEXED64IVPROC(glfwGetProcAddress("glGetVertexArrayIndexed64iv"));
01863     glGetUniformLocation = PFNGLGETVERTEXARRAYINDEXEDIVPROC(glfwGetProcAddress("glGetVertexArrayIndexediv"));
01864     glGetUniformLocation = PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegeri_vEXT"));
01865     glGetUniformLocation = PFNGLGETVERTEXARRAYINTEGERVEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegervEXT"));
01866     glGetUniformLocation = PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayPointeri_vEXT"));
01867     glGetUniformLocation = PFNGLGETVERTEXARRAYPOINTERVEXTPROC(glfwGetProcAddress("glGetVertexArrayPointervEXT"));
01868     glGetUniformLocation = PFNGLGETVERTEXATTRIBL64VNVPROC(glfwGetProcAddress("glGetVertexAttribArrayL64vNV"));
01869     glGetUniformLocation = PFNGLGETVERTEXATTRIBLUI64VNVPROC(glfwGetProcAddress("glGetVertexAttribArrayLui64vNV"));
01870     glGetUniformLocation = PFNGLGETVERTEXATTRIBLUI64VNVPROC(glfwGetProcAddress("glGetVertexAttribArrayLui64vNV"));
01871     glGetUniformLocation = PFNGLGETCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetnCompressedTexImage"));
01872     glGetUniformLocation = PFNGLGETCOMPRESSEDTEXIMAGEARBPROC(glfwGetProcAddress("glGetnCompressedTexImageARB"));
01873     glGetUniformLocation = PFNGLGETNTEXIMAGEPROC(glfwGetProcAddress("glGetnTexImage"));
01874     glGetUniformLocation = PFNGLGETNUNIFORMDVPROC(glfwGetProcAddress("glGetnUniformdv"));
01875     glGetUniformLocation = PFNGLGETNUNIFORMDVARBPROC(glfwGetProcAddress("glGetnUniformdvarb"));
01876     glGetUniformLocation = PFNGLGETNUNIFORMFVPROC(glfwGetProcAddress("glGetnUniformfv"));
01877     glGetUniformLocation = PFNGLGETNUNIFORMFVARBPROC(glfwGetProcAddress("glGetnUniformfvARB"));
01878     glGetUniformLocation = PFNGLGETNUNIFORMI64VARBPROC(glfwGetProcAddress("glGetnUniformi64vARB"));
01879     glGetUniformLocation = PFNGLGETNUNIFORMIVPROC(glfwGetProcAddress("glGetnUniformiv"));
01880     glGetUniformLocation = PFNGLGETNUNIFORMIVARBPROC(glfwGetProcAddress("glGetnUniformivARB"));
01881     glGetUniformLocation = PFNGLGETNUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetnUniformui64vARB"));
01882     glGetUniformLocation = PFNGLGETNUNIFORMUIVPROC(glfwGetProcAddress("glGetnUniformuiv"));
01883     glGetUniformLocation = 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"));

```

```

01889     glInvalidateBufferSubData =
01890     PFNGLINVALIDATEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateBufferSubData"));
01891     glInvalidateFramebuffer =
01892     PFNGLINVALIDATEFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateFramebuffer"));
01893     glInvalidateNamedFramebufferData =
01894     PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferData"));
01895     glInvalidateNamedFramebufferSubData =
01896     PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferSubData"));
01897     glInvalidateSubFramebuffer =
01898     PFNGLINVALIDATESUBFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateSubFramebuffer"));
01899     glInvalidateTexImage = PFNGLINVALIDATETEXIMAGEPROC(glfwGetProcAddress("glInvalidateTexImage"));
01900     glInvalidateTexSubImage =
01901     PFNGLINVALIDATETEXSUBIMAGEPROC(glfwGetProcAddress("glInvalidateTexSubImage"));
01902     gllsBuffer = PFNGLISBUFFERPROC(glfwGetProcAddress("gllsBuffer"));
01903     gllsBufferResidentNV = PFNGLISBUFFERRESIDENTNVPROC(glfwGetProcAddress("gllsBufferResidentNV"));
01904     gllsCommandListNV = PFNGLCOMMANDLISTNVPROC(glfwGetProcAddress("gllsCommandListNV"));
01905     gllsEnabled = PFNGLISENABLEDPROC(glfwGetProcAddress("gllsEnabled"));
01906     gllsEnabledIndexedEXT = PFNGLISENABLEDINDEXEDEXTPROC(glfwGetProcAddress("gllsEnabledIndexedEXT"));
01907     gllsEnablededi = PFNGLISENABLEDIPROC(glfwGetProcAddress("gllsEnablededi"));
01908     gllsFramebuffer = PFNGLISFRAMEBUFFERPROC(glfwGetProcAddress("gllsFramebuffer"));
01909     gllsImageHandleResidentARB =
01910     PFNGLISIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("gllsImageHandleResidentARB"));
01911     gllsImageHandleResidentNV =
01912     PFNGLISIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("gllsImageHandleResidentNV"));
01913     gllsNamedBufferResidentNV =
01914     PFNGLISNAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("gllsNamedBufferResidentNV"));
01915     gllsNamedStringARB = PFNGLISNAMEDSTRINGARBPROC(glfwGetProcAddress("gllsNamedStringARB"));
01916     gllsPathNV = PFNGLISPATHNVPROC(glfwGetProcAddress("gllsPathNV"));
01917     gllsPointInFillPathNV = PFNGLISPOINTINFILLPATHNVPROC(glfwGetProcAddress("gllsPointInFillPathNV"));
01918     gllsPointInStrokePathNV =
01919     PFNGLISPOINTINSTROKEPATHNVPROC(glfwGetProcAddress("gllsPointInStrokePathNV"));
01920     gllsProgram = PFNGLISPROGRAMPROC(glfwGetProcAddress("gllsProgram"));
01921     gllsProgramPipeline = PFNGLISPROGRAMPIPELINEPROC(glfwGetProcAddress("gllsProgramPipeline"));
01922     gllsQuery = PFNGLISQUERYPROC(glfwGetProcAddress("gllsQuery"));
01923     gllsRenderbuffer = PFNGLISRENDERBUFFERPROC(glfwGetProcAddress("gllsRenderbuffer"));
01924     gllsSampler = PFNGLISSAMPLERPROC(glfwGetProcAddress("gllsSampler"));
01925     gllsShader = PFNGLISSHADERPROC(glfwGetProcAddress("gllsShader"));
01926     gllsStateNV = PFNGLISSTATEENVPROC(glfwGetProcAddress("gllsStateNV"));
01927     gllsSync = PFNGLISSYNCPROC(glfwGetProcAddress("gllsSync"));
01928     gllsTexture = PFNGLISTEXTUREPROC(glfwGetProcAddress("gllsTexture"));
01929     gllsTextureHandleResidentARB =
01930     PFNGLISTTEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("gllsTextureHandleResidentARB"));
01931     gllsTextureHandleResidentNV =
01932     PFNGLISTTEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("gllsTextureHandleResidentNV"));
01933     gllsTransformFeedback = PFNGLISTRANSFORMFEEDBACKPROC(glfwGetProcAddress("gllsTransformFeedback"));
01934     gllsVertexArray = PFNGLISVERTEXARRAYPROC(glfwGetProcAddress("gllsVertexArray"));
01935     gllLabelObjectEXT = PFNGLLABELOBJECTEXTPROC(glfwGetProcAddress("gllLabelObjectEXT"));
01936     gllLineWidth = PFNGLLINEWIDTHPROC(glfwGetProcAddress("gllLineWidth"));
01937     gllLinkProgram = PFNGLLINKPROGRAMPROC(glfwGetProcAddress("gllLinkProgram"));
01938     gllListDrawCommandsStatesClientNV =
01939     PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC(glfwGetProcAddress("gllListDrawCommandsStatesClientNV"));
01940     gllLogicOp = PFNGLLOGICOPPROC(glfwGetProcAddress("gllLogicOp"));
01941     gllMakeBufferNonResidentNV =
01942     PFNGLMAKEBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("gllMakeBufferNonResidentNV"));
01943     gllMakeBufferResidentNV =
01944     PFNGLMAKEBUFFERRESIDENTNVPROC(glfwGetProcAddress("gllMakeBufferResidentNV"));
01945     gllMakeImageHandleNonResidentARB =
01946     PFNGLMAKEIMAGENONRESIDENTARBPROC(glfwGetProcAddress("gllMakeImageHandleNonResidentARB"));
01947     gllMakeImageHandleNonResidentNV =
01948     PFNGLMAKEIMAGENONRESIDENTNVPROC(glfwGetProcAddress("gllMakeImageHandleNonResidentNV"));
01949     gllMakeImageHandleResidentARB =
01950     PFNGLMAKEIMAGERESIDENTARBPROC(glfwGetProcAddress("gllMakeImageHandleResidentARB"));
01951     gllMakeImageHandleResidentNV =
01952     PFNGLMAKEIMAGERESIDENTNVPROC(glfwGetProcAddress("gllMakeImageHandleResidentNV"));
01953     gllMakeNamedBufferNonResidentNV =
01954     PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("gllMakeNamedBufferNonResidentNV"));
01955     gllMakeNamedBufferResidentNV =
01956     PFNGLMAKENAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("gllMakeNamedBufferResidentNV"));
01957     gllMakeTextureHandleNonResidentARB =
01958     PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC(glfwGetProcAddress("gllMakeTextureHandleNonResidentARB"));
01959     gllMakeTextureHandleNonResidentNV =
01960     PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC(glfwGetProcAddress("gllMakeTextureHandleNonResidentNV"));
01961     gllMakeTextureHandleResidentARB =
01962     PFNGLMAKETEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("gllMakeTextureHandleResidentARB"));
01963     gllMakeTextureHandleResidentNV =
01964     PFNGLMAKETEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("gllMakeTextureHandleResidentNV"));
01965     gllMapBuffer = PFNGLMAPBUFFERPROC(glfwGetProcAddress("gllMapBuffer"));
01966     gllMapBufferRange = PFNGLMAPBUFFERRANGEPROC(glfwGetProcAddress("gllMapBufferRange"));
01967     gllMapNamedBuffer = PFNGLMAPNAMEDBUFFERPROC(glfwGetProcAddress("gllMapNamedBuffer"));
01968     gllMapNamedBufferEXT = PFNGLMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("gllMapNamedBufferEXT"));
01969     gllMapNamedBufferRange = PFNGLMAPNAMEDBUFFERRANGEPROC(glfwGetProcAddress("gllMapNamedBufferRange"));
01970     gllMapNamedBufferRangeEXT =
01971     PFNGLMAPNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("gllMapNamedBufferRangeEXT"));
01972     gllMatrixFrustumEXT = PFNGLMATRIXFRUSTUMEXTPROC(glfwGetProcAddress("gllMatrixFrustumEXT"));
01973     gllMatrixLoad3x2fNV = PFNGLMATRIXLOAD3X2FNVPROC(glfwGetProcAddress("gllMatrixLoad3x2fNV"));
01974     gllMatrixLoad3x3fNV = PFNGLMATRIXLOAD3X3FNVPROC(glfwGetProcAddress("gllMatrixLoad3x3fNV"));
01975     gllMatrixLoadIdentityEXT =

```

```

PFNGLMATRIXLOADIDENTITYEXTPROC glfwGetProcAddress("glMatrixLoadIdentityEXT"));
01950    glMatrixLoadTranspose3x3fNV =
PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glfwGetProcAddress("glMatrixLoadTranspose3x3fNV"));
01951    glMatrixLoadTransposedEXT =
PFNGLMATRIXLOADTRANSPOSEDEXTPROC glfwGetProcAddress("glMatrixLoadTransposedEXT"));
01952    glMatrixLoadTransposefEXT =
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 =
PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glfwGetProcAddress("glMatrixMultTranspose3x3fNV"));
01958    glMatrixMultTransposedEXT =
PFNGLMATRIXMULTTRANSPOSEDEXTPROC glfwGetProcAddress("glMatrixMultTransposedEXT"));
01959    glMatrixMultTransposefEXT =
PFNGLMATRIXMULTTRANSPOSEFEXTPROC glfwGetProcAddress("glMatrixMultTransposefEXT"));
01960    glMatrixMultdEXT = PFNGLMATRIXMULTDEXTPROC glfwGetProcAddress("glMatrixMultdEXT"));
01961    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 =
PFNGLMAXSHADERCOMPILERTHREADSARBPROC glfwGetProcAddress("glMaxShaderCompilerThreadsARB"));
01972    glMemoryBarrier = PFNGLMEMORYBARRIERPROC glfwGetProcAddress("glMemoryBarrier"));
01973    glMemoryBarrierByRegion =
PFNGLMEMORYBARRIERBYREGIONPROC glfwGetProcAddress("glMemoryBarrierByRegion"));
01974    glMinSampleShading = PFNGLMINSAMPLESHADINGPROC glfwGetProcAddress("glMinSampleShading"));
01975    glMinSampleShadingARB = PFNGLMINSAMPLESHADINGARBPROC glfwGetProcAddress("glMinSampleShadingARB"));
01976    glMultiDrawArrays = PFNGLMULTIDRAWARRAYSPROC glfwGetProcAddress("glMultiDrawArrays"));
01977    glMultiDrawArraysIndirect =
PFNGLMULTIDRAWARRAYSINDIRECTPROC glfwGetProcAddress("glMultiDrawArraysIndirect"));
01978    glMultiDrawArraysIndirectBindlessCountNV =
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glfwGetProcAddress("glMultiDrawArraysIndirectBindlessCountNV"));
01979    glMultiDrawArraysIndirectBindlessNV =
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glfwGetProcAddress("glMultiDrawArraysIndirectBindlessNV"));
01980    glMultiDrawArraysIndirectCountARB =
PFNGLMULTIDRAWARRAYSINDIRECTCOUNTRBPROC glfwGetProcAddress("glMultiDrawArraysIndirectCountARB"));
01981    glMultiDrawElements = PFNGLMULTIDRAWELEMENTSPROC glfwGetProcAddress("glMultiDrawElements"));
01982    glMultiDrawElementsBaseVertex =
PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glfwGetProcAddress("glMultiDrawElementsBaseVertex"));
01983    glMultiDrawElementsIndirect =
PFNGLMULTIDRAWELEMENTSINDIRECTPROC glfwGetProcAddress("glMultiDrawElementsIndirect"));
01984    glMultiDrawElementsIndirectBindlessCountNV =
PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glfwGetProcAddress("glMultiDrawElementsIndirectBindlessCountNV"));
01985    glMultiDrawElementsIndirectBindlessNV =
PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glfwGetProcAddress("glMultiDrawElementsIndirectBindlessNV"));
01986    glMultiDrawElementsIndirectCountARB =
PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTRBPROC glfwGetProcAddress("glMultiDrawElementsIndirectCountARB"));
01987    glMultiTexBufferEXT = PFNGLMULTITEXBUFFEREXTPROC glfwGetProcAddress("glMultiTexBufferEXT"));
01988    glMultiTexCoordPointerEXT =
PFNGLMULTITEXCOORDPOINTEREEXTPROC glfwGetProcAddress("glMultiTexCoordPointerEXT"));
01989    glMultiTexEnvfEXT = PFNGLMULTITEXENVFEXTPROC glfwGetProcAddress("glMultiTexEnvfEXT"));
01990    glMultiTexEnvfVEXT = PFNGLMULTITEXENVFVEXTPROC glfwGetProcAddress("glMultiTexEnvfVEXT"));
01991    glMultiTexEnviEXT = PFNGLMULTITEXENVIENTRPROC glfwGetProcAddress("glMultiTexEnviEXT"));
01992    glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC glfwGetProcAddress("glMultiTexEnvivEXT"));
01993    glMultiTexGendEXT = PFNGLMULTITEXGENDEXTPROC glfwGetProcAddress("glMultiTexGendEXT"));
01994    glMultiTexGenfEXT = PFNGLMULTITEXGENDEXTPROC glfwGetProcAddress("glMultiTexGenfEXT"));
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    glMultiTexParameterIiivEXT =
PFNGLMULTITEXPARAMETERIIVEXTPROC glfwGetProcAddress("glMultiTexParameterIiivEXT"));
02003    glMultiTexParameterIiivEXT =
PFNGLMULTITEXPARAMETERIUIVEXTPROC glfwGetProcAddress("glMultiTexParameterIuiVEXT"));
02004    glMultiTexParameterfEXT =
PFNGLMULTITEXPARAMETERFEXTPROC glfwGetProcAddress("glMultiTexParameterfEXT"));
02005    glMultiTexParameterfvEXT =
PFNGLMULTITEXPARAMETERFVEXTPROC glfwGetProcAddress("glMultiTexParameterfvEXT"));
02006    glMultiTexParameterivEXT =
PFNGLMULTITEXPARAMETERIEXTPROC glfwGetProcAddress("glMultiTexParameterivEXT"));
02007    glMultiTexParameterivEXT =
PFNGLMULTITEXPARAMETERIVEXTPROC glfwGetProcAddress("glMultiTexParameterivEXT"));
02008    glMultiTexRenderbufferEXT =
PFNGLMULTITEXRENDERBUFFEREXTPROC glfwGetProcAddress("glMultiTexRenderbufferEXT"));
02009    glMultiTexSubImage1DEXT =
PFNGLMULTITEXSUBIMAGE1DEXTPROC glfwGetProcAddress("glMultiTexSubImage1DEXT"));

```

```

02010     glMultiTexSubImage2DEXT =
02011     PFNGLMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexSubImage2DEXT"));
02012     glMultiTexSubImage3DEXT =
02013     PFNGLMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexSubImage3DEXT"));
02014     glNamedBufferData = PFNGLNAMEDBUFFERDATAPROC(glfwGetProcAddress("glNamedBufferData"));
02015     glNamedBufferDataEXT = PFNGLNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glNamedBufferDataEXT"));
02016     glNamedBufferPageCommitmentARB =
02017     PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glNamedBufferPageCommitmentARB"));
02018     glNamedBufferPageCommitmentEXT =
02019     PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glNamedBufferPageCommitmentEXT"));
02020     glNamedBufferStorage = PFNGLNAMEDBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedBufferStorage"));
02021     glNamedBufferStorageEXT =
02022     PFNGLNAMEDBUFFERSTORAGEXTPROC(glfwGetProcAddress("glNamedBufferStorageEXT"));
02023     glNamedBufferSubData = PFNGLNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glNamedBufferSubData"));
02024     glNamedBufferSubDataEXT =
02025     PFNGLNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedBufferSubDataEXT"));
02026     glNamedCopyBufferSubDataEXT =
02027     PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedCopyBufferSubDataEXT"));
02028     glNamedFramebufferDrawBuffer =
02029     PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffer"));
02030     glNamedFramebufferDrawBuffers =
02031     PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffers"));
02032     glNamedFramebufferParameteri =
02033     PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glNamedFramebufferParameteri"));
02034     glNamedFramebufferParameteriEXT =
02035     PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC(glfwGetProcAddress("glNamedFramebufferParameteriEXT"));
02036     glNamedFramebufferReadBuffer =
02037     PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC(glfwGetProcAddress("glNamedFramebufferReadBuffer"));
02038     glNamedFramebufferRenderbuffer =
02039     PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glNamedFramebufferRenderbuffer"));
02040     glNamedFramebufferRenderbufferEXT =
02041     PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC(glfwGetProcAddress("glNamedFramebufferRenderbufferEXT"));
02042     glNamedFramebufferSampleLocationsfvARB =
02043     PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvARB"));
02044     glNamedFramebufferSampleLocationsfvNV =
02045     PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvNV"));
02046     glNamedFramebufferTexture =
02047     PFNGLNAMEDFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glNamedFramebufferTexture"));
02048     glNamedFramebufferTexture1DEXT =
02049     PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture1DEXT"));
02050     glNamedFramebufferTexture2DEXT =
02051     PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture2DEXT"));
02052     glNamedFramebufferTexture3DEXT =
02053     PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture3DEXT"));
02054     glNamedFramebufferTextureEXT =
02055     PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureEXT"));
02056     glNamedFramebufferTextureFaceEXT =
02057     PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureFaceEXT"));
02058     glNamedFramebufferTextureLayer =
02059     PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glNamedFramebufferTextureLayer"));
02060     glNamedFramebufferTextureLayerEXT =
02061     PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC(glfwGetProcAddress("glNamedFramebufferTextureLayerEXT"));
02062     glNamedProgramLocalParameter4dEXT =
02063     PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dEXT"));
02064     glNamedProgramLocalParameter4dvEXT =
02065     PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dvEXT"));
02066     glNamedProgramLocalParameter4fEXT =
02067     PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fEXT"));
02068     glNamedProgramLocalParameter4fvEXT =
02069     PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fvEXT"));
02070     glNamedProgramLocalParameterI4iEXT =
02071     PFNGLNAMEDPROGRAMLOCALPARAMETERI4IBEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4iEXT"));
02072     glNamedProgramLocalParameterI4ivEXT =
02073     PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4ivEXT"));
02074     glNamedProgramLocalParameterI4uiEXT =
02075     PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uiEXT"));
02076     glNamedProgramLocalParameterI4uvEXT =
02077     PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uvEXT"));
02078     glNamedProgramLocalParameters4fvEXT =
02079     PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameters4fvEXT"));
02080     glNamedProgramLocalParametersI4ivEXT =
02081     PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4ivEXT"));
02082     glNamedProgramLocalParametersI4uvEXT =
02083     PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4uvEXT"));
02084     glNamedProgramStringEXT =
02085     PFNGLNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glNamedProgramStringEXT"));
02086     glNamedRenderbufferStorage =
02087     PFNGLNAMEDRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedRenderbufferStorage"));
02088     glNamedRenderbufferStorageEXT =
02089     PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageEXT"));
02090     glNamedRenderbufferStorageMultisample =
02091     PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisample"));
02092     glNamedRenderbufferStorageMultisampleCoverageEXT =
02093     PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleCoverageEXT"));
02094     glNamedRenderbufferStorageMultisampleEXT =
02095     PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleEXT"));
02096     glNamedStringARB = PFNGLNAMEDSTRINGARBPROC(glfwGetProcAddress("glNamedStringARB"));

```

```

02056     glNormalFormatNV = PFNGLNORMALFORMATNVPROC(glfwGetProcAddress("glNormalFormatNV"));
02057     glObjectLabel = PFNGLOBJECTLABELPROC(glfwGetProcAddress("glObjectLabel"));
02058     glObjectPtrLabel = PFNGLOBJECTPTRLABELPROC(glfwGetProcAddress("glObjectPtrLabel"));
02059     glPatchParameterfv = PFNGLPATCHPARAMETERFVPROC(glfwGetProcAddress("glPatchParameterfv"));
02060     glPatchParameteri = PFNGLPATCHPARAMETERIPROC(glfwGetProcAddress("glPatchParameteri"));
02061     glPathCommandsNV = PFNGLPATHCOMMANDSNVPROC(glfwGetProcAddress("glPathCommandsNV"));
02062     glPathCoordsNV = PFNGLPATHCOORDSNVPROC(glfwGetProcAddress("glPathCoordsNV"));
02063     glPathCoverDepthFuncNV =
02064         PFNGLPATHCOVERDEPTHFHUNCNVPROC(glfwGetProcAddress("glPathCoverDepthFuncNV"));
02065     glPathDashArrayNV = PFNGLPATHDASHARRAYNVPROC(glfwGetProcAddress("glPathDashArrayNV"));
02066     glPathGlyphIndexArrayNV =
02067         PFNGLPATHGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathGlyphIndexArrayNV"));
02068     glPathGlyphsNV = PFNGLPATHGLYPHSNVPROC(glfwGetProcAddress("glPathGlyphsNV"));
02069     glPathMemoryGlyphIndexArrayNV =
02070         PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathMemoryGlyphIndexArrayNV"));
02071     glPathParameterfNV = PFNGLPATHPARAMETERFNVPROC(glfwGetProcAddress("glPathParameterfNV"));
02072     glPathParameterfvNV = PFNGLPATHPARAMETERFVNVPROC(glfwGetProcAddress("glPathParameterfvNV"));
02073     glPathParameteriNV = PFNGLPATHPARAMETERINVPROC(glfwGetProcAddress("glPathParameteriNV"));
02074     glPathParameterivNV = PFNGLPATHPARAMETERIVNVPROC(glfwGetProcAddress("glPathParameterivNV"));
02075     glPathStencilDepthOffsetNV =
02076         PFNGLPATHSTENCILDEPTHOFFSETNVPROC(glfwGetProcAddress("glPathStencilDepthOffsetNV"));
02077     glPathStencilFuncNV = PFNGLPATHSTENCILFUNCNVPROC(glfwGetProcAddress("glPathStencilFuncNV"));
02078     glPathStringNV = PFNGLPATHSTRINGNVPROC(glfwGetProcAddress("glPathStringNV"));
02079     glPathSubCommandsNV = PFNGLPATHSUBCOMMANDSNVPROC(glfwGetProcAddress("glPathSubCommandsNV"));
02080     glPathSubCoordsNV = PFNGLPATHSUBCOORDSNVPROC(glfwGetProcAddress("glPathSubCoordsNV"));
02081     glPauseTransformFeedback =
02082         PFNGLPAUSETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glPauseTransformFeedback"));
02083     glPixelStoref = PFNGLPIXELSTOREFFPROC(glfwGetProcAddress("glPixelStoref"));
02084     glPixelStorei = PFNGLPIXELSTOREIPROC(glfwGetProcAddress("glPixelStorei"));
02085     glPointAlongPathNV = PFNGLPOINTALONGPATHNVPROC(glfwGetProcAddress("glPointAlongPathNV"));
02086     glPointParameterf = PFNGLPOINTPARAMETERFPROC(glfwGetProcAddress("glPointParameterf"));
02087     glPointParameterfv = PFNGLPOINTPARAMETERFVPROC(glfwGetProcAddress("glPointParameterfv"));
02088     glPointParameteri = PFNGLPOINTPARAMETERIPROC(glfwGetProcAddress("glPointParameteri"));
02089     glPointParameteriv = PFNGLPOINTPARAMETERIVPROC(glfwGetProcAddress("glPointParameteriv"));
02090     glPointSize = PFNGLPOINTSIZEPROC(glfwGetProcAddress("glPointSize"));
02091     glPolygonMode = PFNGLPOLYGMODEPROC(glfwGetProcAddress("glPolygonMode"));
02092     glPolygonOffset = PFNGLPOLYGONOFFSETPROC(glfwGetProcAddress("glPolygonOffset"));
02093     glPolygonOffsetClampEXT =
02094         PFNGLPOLYGONOFFSETCLAMPEXTPROC(glfwGetProcAddress("glPolygonOffsetClampEXT"));
02095     glPopDebugGroup = PFNGLPOPOBJECTDEBBUGGROUPPROC(glfwGetProcAddress("glPopDebugGroup"));
02096     glPopGroupMarkerEXT = PFNGLPOPGROUPMARKEREXTPROC(glfwGetProcAddress("glPopGroupMarkerEXT"));
02097     glPrimitiveBoundingBoxARB =
02098         PFNGLPRIMITIVEBOUNDINGBOXARBPROC(glfwGetProcAddress("glPrimitiveBoundingBoxARB"));
02099     glPrimitiveRestartIndex =
02100         PFNGLPRIMITIVERESTARTINDEXPROC(glfwGetProcAddress("glPrimitiveRestartIndex"));
02101     glProgramBinary = PFNGLPROGRAMBINARYPROC(glfwGetProcAddress("glProgramBinary"));
02102     glProgramParameteri = PFNGLPROGRAMPARAMETERIPROC(glfwGetProcAddress("glProgramParameteri"));
02103     glProgramParameteriARB =
02104         PFNGLPROGRAMPARAMETERIARBPROC(glfwGetProcAddress("glProgramParameteriARB"));
02105     glProgramPathFragmentInputGenNV =
02106         PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC(glfwGetProcAddress("glProgramPathFragmentInputGenNV"));
02107     glProgramUniform1d = PFNGLPROGRAMUNIFORM1DPROC(glfwGetProcAddress("glProgramUniform1d"));
02108     glProgramUniform1dEXT = PFNGLPROGRAMUNIFORM1DEXTPROC(glfwGetProcAddress("glProgramUniform1dEXT"));
02109     glProgramUniform1dv = PFNGLPROGRAMUNIFORM1DVPROC(glfwGetProcAddress("glProgramUniform1dv"));
02110     glProgramUniform1dvEXT =
02111         PFNGLPROGRAMUNIFORM1DVEXTPROC(glfwGetProcAddress("glProgramUniform1dvEXT"));
02112     glProgramUniform1f = PFNGLPROGRAMUNIFORM1FPROC(glfwGetProcAddress("glProgramUniform1f"));
02113     glProgramUniform1fEXT = PFNGLPROGRAMUNIFORM1FEXTPROC(glfwGetProcAddress("glProgramUniform1fEXT"));
02114     glProgramUniform1fv = PFNGLPROGRAMUNIFORM1FVPROC(glfwGetProcAddress("glProgramUniform1fv"));
02115     glProgramUniform1fvEXT =
02116         PFNGLPROGRAMUNIFORM1FVEXTPROC(glfwGetProcAddress("glProgramUniform1fvEXT"));
02117     glProgramUniform1i = PFNGLPROGRAMUNIFORM1IPROC(glfwGetProcAddress("glProgramUniform1i"));
02118     glProgramUniform1i64ARB =
02119         PFNGLPROGRAMUNIFORM1I64ARBPROC(glfwGetProcAddress("glProgramUniform1i64ARB"));
02120     glProgramUniform1i64NV =
02121         PFNGLPROGRAMUNIFORM1I64NVPROC(glfwGetProcAddress("glProgramUniform1i64NV"));
02122     glProgramUniform1iEXT = PFNGLPROGRAMUNIFORM1IEXTPROC(glfwGetProcAddress("glProgramUniform1iEXT"));
02123     glProgramUniform1iv = PFNGLPROGRAMUNIFORM1IVPROC(glfwGetProcAddress("glProgramUniform1iv"));
02124     glProgramUniform1ivEXT =
02125         PFNGLPROGRAMUNIFORM1IVEXTPROC(glfwGetProcAddress("glProgramUniform1ivEXT"));
02126     glProgramUniform1ui = PFNGLPROGRAMUNIFORM1UIPROC(glfwGetProcAddress("glProgramUniform1ui"));
02127     glProgramUniform1ui64ARB =
02128         PFNGLPROGRAMUNIFORM1UI64ARBPROC(glfwGetProcAddress("glProgramUniform1ui64ARB"));
02129     glProgramUniform1ui64NV =
02130         PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniform1ui64NV"));
02131     glProgramUniform1ui64vARB =
02132         PFNGLPROGRAMUNIFORM1UI64VARBPROC(glfwGetProcAddress("glProgramUniform1ui64vARB"));
02133     glProgramUniform1ui64vNV =
02134         PFNGLPROGRAMUNIFORM1UI64VNPROC(glfwGetProcAddress("glProgramUniform1ui64vNV"));
02135     glProgramUniform1uiEXT = PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniform1uiEXT"));
02136     glProgramUniform1iv = PFNGLPROGRAMUNIFORM1IVPROC(glfwGetProcAddress("glProgramUniform1iv"));
02137     glProgramUniform1ivEXT =
02138         PFNGLPROGRAMUNIFORM1IVEXTPROC(glfwGetProcAddress("glProgramUniform1ivEXT"));
02139     glProgramUniform1ui = PFNGLPROGRAMUNIFORM1UIPROC(glfwGetProcAddress("glProgramUniform1ui"));
02140     glProgramUniform1ui64ARB =
02141         PFNGLPROGRAMUNIFORM1UI64ARBPROC(glfwGetProcAddress("glProgramUniform1ui64ARB"));
02142     glProgramUniform1ui64NV =
02143         PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniform1ui64NV"));
02144     glProgramUniform1ui64vARB =
02145         PFNGLPROGRAMUNIFORM1UI64VARBPROC(glfwGetProcAddress("glProgramUniform1ui64vARB"));
02146     glProgramUniform1ui64vNV =
02147         PFNGLPROGRAMUNIFORM1UI64VNPROC(glfwGetProcAddress("glProgramUniform1ui64vNV"));
02148     glProgramUniform1uiEXT =

```

```

PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniform1uiEXT"));
02111    glProgramUniform1ui = PFNGLPROGRAMUNIFORM1UIVPROC(glfwGetProcAddress("glProgramUniform1ui"));
02112    glProgramUniform1uiEXT =
PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniform1uiEXT"));
02113    glProgramUniform2d = PFNGLPROGRAMUNIFORM2DPROC(glfwGetProcAddress("glProgramUniform2d"));
02114    glProgramUniform2dEXT = PFNGLPROGRAMUNIFORM2DEXTPROC(glfwGetProcAddress("glProgramUniform2dEXT"));
02115    glProgramUniform2dv = PFNGLPROGRAMUNIFORM2DVPROC(glfwGetProcAddress("glProgramUniform2dv"));
02116    glProgramUniform2dvEXT =
PFNGLPROGRAMUNIFORM2DVEXTPROC(glfwGetProcAddress("glProgramUniform2dvEXT"));
02117    glProgramUniform2f = PFNGLPROGRAMUNIFORM2FPROC(glfwGetProcAddress("glProgramUniform2f"));
02118    glProgramUniform2fEXT = PFNGLPROGRAMUNIFORM2FEXTPROC(glfwGetProcAddress("glProgramUniform2fEXT"));
02119    glProgramUniform2fv = PFNGLPROGRAMUNIFORM2FVPROC(glfwGetProcAddress("glProgramUniform2fv"));
02120    glProgramUniform2fvEXT =
PFNGLPROGRAMUNIFORM2FVEXTPROC(glfwGetProcAddress("glProgramUniform2fvEXT"));
02121    glProgramUniform2i = PFNGLPROGRAMUNIFORM2IPROC(glfwGetProcAddress("glProgramUniform2i"));
02122    glProgramUniform2iARB =
PFNGLPROGRAMUNIFORM2I64ARBPROC(glfwGetProcAddress("glProgramUniform2i64ARB"));
02123    glProgramUniform2i64NV =
PFNGLPROGRAMUNIFORM2I64NVPROC(glfwGetProcAddress("glProgramUniform2i64NV"));
02124    glProgramUniform2i64vARB =
PFNGLPROGRAMUNIFORM2I64VARBPROC(glfwGetProcAddress("glProgramUniform2i64vARB"));
02125    glProgramUniform2i64vNV =
PFNGLPROGRAMUNIFORM2I64VNVPROC(glfwGetProcAddress("glProgramUniform2i64vNV"));
02126    glProgramUniform2iEXT = PFNGLPROGRAMUNIFORM2IEXTPROC(glfwGetProcAddress("glProgramUniform2iEXT"));
02127    glProgramUniform2iv = PFNGLPROGRAMUNIFORM2IVPROC(glfwGetProcAddress("glProgramUniform2iv"));
02128    glProgramUniform2ivEXT =
PFNGLPROGRAMUNIFORM2IVEXTPROC(glfwGetProcAddress("glProgramUniform2ivEXT"));
02129    glProgramUniform2ui = PFNGLPROGRAMUNIFORM2UIPROC(glfwGetProcAddress("glProgramUniform2ui"));
02130    glProgramUniform2uiARB =
PFNGLPROGRAMUNIFORM2UI64ARBPROC(glfwGetProcAddress("glProgramUniform2ui64ARB"));
02131    glProgramUniform2ui64NV =
PFNGLPROGRAMUNIFORM2UI64NVPROC(glfwGetProcAddress("glProgramUniform2ui64NV"));
02132    glProgramUniform2ui64vARB =
PFNGLPROGRAMUNIFORM2UI64VARBPROC(glfwGetProcAddress("glProgramUniform2ui64vARB"));
02133    glProgramUniform2ui64vNV =
PFNGLPROGRAMUNIFORM2UI64VNVPROC(glfwGetProcAddress("glProgramUniform2ui64vNV"));
02134    glProgramUniform2i1EXT = PFNGLPROGRAMUNIFORM2I1PROC(glfwGetProcAddress("glProgramUniform2i1EXT"));
02135    glProgramUniform2i1iv =
PFNGLPROGRAMUNIFORM2I1IVPROC(glfwGetProcAddress("glProgramUniform2i1iv"));
02136    glProgramUniform2i1ui =
PFNGLPROGRAMUNIFORM2I1UIPROC(glfwGetProcAddress("glProgramUniform2i1ui"));
02137    glProgramUniform2i1uiEXT =
PFNGLPROGRAMUNIFORM2I1UIEXTPROC(glfwGetProcAddress("glProgramUniform2i1uiEXT"));
02138    glProgramUniform2i1ui64ARB =
PFNGLPROGRAMUNIFORM2I1UI64ARBPROC(glfwGetProcAddress("glProgramUniform2i1ui64ARB"));
02139    glProgramUniform2i1ui64NV =
PFNGLPROGRAMUNIFORM2I1UI64NVPROC(glfwGetProcAddress("glProgramUniform2i1ui64NV"));
02140    glProgramUniform2i1ui64vARB =
PFNGLPROGRAMUNIFORM2I1UI64VARBPROC(glfwGetProcAddress("glProgramUniform2i1ui64vARB"));
02141    glProgramUniform2i1ui64vNV =
PFNGLPROGRAMUNIFORM2I1UI64VNVPROC(glfwGetProcAddress("glProgramUniform2i1ui64vNV"));
02142    glProgramUniform2i1uiEXT =
PFNGLPROGRAMUNIFORM2I1UIEXTPROC(glfwGetProcAddress("glProgramUniform2i1uiEXT"));
02143    glProgramUniform2i1ui64vNV =
PFNGLPROGRAMUNIFORM2I1UI64VNVPROC(glfwGetProcAddress("glProgramUniform2i1ui64vNV"));
02144    glProgramUniform2i1uiEXT =
PFNGLPROGRAMUNIFORM2I1UIEXTPROC(glfwGetProcAddress("glProgramUniform2i1uiEXT"));
02145    glProgramUniform2i1ui64NV =
PFNGLPROGRAMUNIFORM2I1UI64NVPROC(glfwGetProcAddress("glProgramUniform2i1ui64NV"));
02146    glProgramUniform2i1uiEXT =
PFNGLPROGRAMUNIFORM2I1UIEXTPROC(glfwGetProcAddress("glProgramUniform2i1uiEXT"));
02147    glProgramUniform3d = PFNGLPROGRAMUNIFORM3DPROC(glfwGetProcAddress("glProgramUniform3d"));
02148    glProgramUniform3dEXT = PFNGLPROGRAMUNIFORM3DEXTPROC(glfwGetProcAddress("glProgramUniform3dEXT"));
02149    glProgramUniform3dv = PFNGLPROGRAMUNIFORM3DVPROC(glfwGetProcAddress("glProgramUniform3dv"));
02150    glProgramUniform3dvEXT =
PFNGLPROGRAMUNIFORM3DVEXTPROC(glfwGetProcAddress("glProgramUniform3dvEXT"));
02151    glProgramUniform3f = PFNGLPROGRAMUNIFORM3FPROC(glfwGetProcAddress("glProgramUniform3f"));
02152    glProgramUniform3fEXT = PFNGLPROGRAMUNIFORM3FEXTPROC(glfwGetProcAddress("glProgramUniform3fEXT"));
02153    glProgramUniform3fv = PFNGLPROGRAMUNIFORM3FVPROC(glfwGetProcAddress("glProgramUniform3fv"));
02154    glProgramUniform3fvEXT =
PFNGLPROGRAMUNIFORM3FVEXTPROC(glfwGetProcAddress("glProgramUniform3fvEXT"));
02155    glProgramUniform3i = PFNGLPROGRAMUNIFORM3IPROC(glfwGetProcAddress("glProgramUniform3i"));
02156    glProgramUniform3i164ARB =
PFNGLPROGRAMUNIFORM3I164ARBPROC(glfwGetProcAddress("glProgramUniform3i164ARB"));
02157    glProgramUniform3i164NV =
PFNGLPROGRAMUNIFORM3I164NVPROC(glfwGetProcAddress("glProgramUniform3i164NV"));
02158    glProgramUniform3i164vARB =
PFNGLPROGRAMUNIFORM3I164VARBPROC(glfwGetProcAddress("glProgramUniform3i164vARB"));
02159    glProgramUniform3i164vNV =
PFNGLPROGRAMUNIFORM3I164VNVPROC(glfwGetProcAddress("glProgramUniform3i164vNV"));
02160    glProgramUniform3iEXT = PFNGLPROGRAMUNIFORM3IEXTPROC(glfwGetProcAddress("glProgramUniform3iEXT"));
02161    glProgramUniform3iv = PFNGLPROGRAMUNIFORM3IVPROC(glfwGetProcAddress("glProgramUniform3iv"));
02162    glProgramUniform3ivEXT =
PFNGLPROGRAMUNIFORM3IVEXTPROC(glfwGetProcAddress("glProgramUniform3ivEXT"));
02163    glProgramUniform3ui = PFNGLPROGRAMUNIFORM3UIPROC(glfwGetProcAddress("glProgramUniform3ui"));
02164    glProgramUniform3ui164ARB =
PFNGLPROGRAMUNIFORM3UI164ARBPROC(glfwGetProcAddress("glProgramUniform3ui164ARB"));
02165    glProgramUniform3ui164NV =
PFNGLPROGRAMUNIFORM3UI164NVPROC(glfwGetProcAddress("glProgramUniform3ui164NV"));
02166    glProgramUniform3ui164vARB =
PFNGLPROGRAMUNIFORM3UI164VARBPROC(glfwGetProcAddress("glProgramUniform3ui164vARB"));
02167    glProgramUniform3ui164vNV =
PFNGLPROGRAMUNIFORM3UI164VNVPROC(glfwGetProcAddress("glProgramUniform3ui164vNV"));
02168    glProgramUniform3uiEXT =
PFNGLPROGRAMUNIFORM3UIEXTPROC(glfwGetProcAddress("glProgramUniform3uiEXT"));
02169    glProgramUniform3uiv = PFNGLPROGRAMUNIFORM3UIVPROC(glfwGetProcAddress("glProgramUniform3uiv"));
02170    glProgramUniform3uiEXT =
PFNGLPROGRAMUNIFORM3UIVEXTPROC(glfwGetProcAddress("glProgramUniform3uiEXT"));
02171    glProgramUniform4d = PFNGLPROGRAMUNIFORM4DPROC(glfwGetProcAddress("glProgramUniform4d"));
02172    glProgramUniform4dEXT = PFNGLPROGRAMUNIFORM4DEXTPROC(glfwGetProcAddress("glProgramUniform4dEXT"));
02173    glProgramUniform4dv = PFNGLPROGRAMUNIFORM4DVPROC(glfwGetProcAddress("glProgramUniform4dv"));
02174    glProgramUniform4dvEXT =
PFNGLPROGRAMUNIFORM4DVEXTPROC(glfwGetProcAddress("glProgramUniform4dvEXT"));
02175    glProgramUniform4f = PFNGLPROGRAMUNIFORM4FFPROC(glfwGetProcAddress("glProgramUniform4f"));
02176    glProgramUniform4fEXT = PFNGLPROGRAMUNIFORM4FEXTPROC(glfwGetProcAddress("glProgramUniform4fEXT"));
02177    glProgramUniform4fv = PFNGLPROGRAMUNIFORM4FVPROC(glfwGetProcAddress("glProgramUniform4fv"));
02178    glProgramUniform4fvEXT =

```

```
02179 PFNGLPROGRAMUNIFORM4FVEXTPROC(glfwGetProcAddress("glProgramUniform4fvEXT"));
02180 g1ProgramUniform4i = PFNGLPROGRAMUNIFORM4IPROC(glfwGetProcAddress("glProgramUniform4i"));
02181 PFNGLPROGRAMUNIFORM4I64ARBPROC(glfwGetProcAddress("glProgramUniform4i64ARB"));
02182 g1ProgramUniform4i64NV =
02183 PFNGLPROGRAMUNIFORM4I64NVPROC(glfwGetProcAddress("glProgramUniform4i64NV"));
02184 g1ProgramUniform4i64vARB =
02185 PFNGLPROGRAMUNIFORM4I64VARBPROC(glfwGetProcAddress("glProgramUniform4i64vARB"));
02186 g1ProgramUniform4i64vNV =
02187 PFNGLPROGRAMUNIFORM4I64VNVPROC(glfwGetProcAddress("glProgramUniform4i64vNV"));
02188 g1ProgramUniform4iEXT = PFNGLPROGRAMUNIFORM4IEXTPROC(glfwGetProcAddress("glProgramUniform4iEXT"));
02189 g1ProgramUniform4iv = PFNGLPROGRAMUNIFORM4IVPROC(glfwGetProcAddress("glProgramUniform4iv"));
02190 g1ProgramUniform4ivEXT =
02191 PFNGLPROGRAMUNIFORM4IVEXTPROC(glfwGetProcAddress("glProgramUniform4ivEXT"));
02192 g1ProgramUniform4ui = PFNGLPROGRAMUNIFORM4UIPROC(glfwGetProcAddress("glProgramUniform4ui"));
02193 g1ProgramUniform4ui64ARB =
02194 PFNGLPROGRAMUNIFORM4UI64ARBPROC(glfwGetProcAddress("glProgramUniform4ui64ARB"));
02195 g1ProgramUniform4ui64NV =
02196 PFNGLPROGRAMUNIFORM4UI64NVPROC(glfwGetProcAddress("glProgramUniform4ui64NV"));
02197 g1ProgramUniform4ui64vARB =
02198 PFNGLPROGRAMUNIFORM4UI64VARBPROC(glfwGetProcAddress("glProgramUniform4ui64vARB"));
02199 g1ProgramUniform4ui64vNV =
02200 PFNGLPROGRAMUNIFORM4UI64VNVPROC(glfwGetProcAddress("glProgramUniform4ui64vNV"));
02201 g1ProgramUniform4uiEXT =
02202 PFNGLPROGRAMUNIFORM4UIEXTPROC(glfwGetProcAddress("glProgramUniform4uiEXT"));
02203 g1ProgramUniform4uiv = PFNGLPROGRAMUNIFORM4UIVPROC(glfwGetProcAddress("glProgramUniform4uiv"));
02204 g1ProgramUniform4uivEXT =
02205 PFNGLPROGRAMUNIFORM4UIVEXTPROC(glfwGetProcAddress("glProgramUniform4uivEXT"));
02206 g1ProgramUniformHandleui64ARB =
02207 PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glProgramUniformHandleui64ARB"));
02208 g1ProgramUniformHandleui64NV =
02209 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glProgramUniformHandleui64NV"));
02210 g1ProgramUniformHandleui64vARB =
02211 PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glProgramUniformHandleui64vARB"));
02212 g1ProgramUniformHandleui64vNV =
02213 PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glProgramUniformHandleui64vNV"));
02214 g1ProgramUniformMatrix2dv =
02215 PFNGLPROGRAMUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glProgramUniformMatrix2dv"));
02216 g1ProgramUniformMatrix2dVEXT =
02217 PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2dVEXT"));
02218 g1ProgramUniformMatrix2fv =
02219 PFNGLPROGRAMUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glProgramUniformMatrix2fv"));
02220 g1ProgramUniformMatrix2fVEXT =
02221 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2fVEXT"));
02222 g1ProgramUniformMatrix2x3dv =
02223 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dv"));
02224 g1ProgramUniformMatrix2x3dvEXT =
02225 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dvEXT"));
02226 g1ProgramUniformMatrix2x3fv =
02227 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fv"));
02228 g1ProgramUniformMatrix2x3fvEXT =
02229 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fvEXT"));
02230 g1ProgramUniformMatrix2x4dv =
02231 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dv"));
02232 g1ProgramUniformMatrix2x4dVEXT =
02233 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dVEXT"));
02234 g1ProgramUniformMatrix2x4fv =
02235 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fv"));
02236 g1ProgramUniformMatrix2x4fvEXT =
02237 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fvEXT"));
02238 g1ProgramUniformMatrix3dv =
02239 PFNGLPROGRAMUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glProgramUniformMatrix3dv"));
02240 g1ProgramUniformMatrix3dVEXT =
02241 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3dVEXT"));
02242 g1ProgramUniformMatrix3fv =
02243 PFNGLPROGRAMUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glProgramUniformMatrix3fv"));
02244 g1ProgramUniformMatrix3fvEXT =
02245 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3fvEXT"));
02246 g1ProgramUniformMatrix3x2dv =
02247 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dv"));
02248 g1ProgramUniformMatrix3x2dVEXT =
02249 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dVEXT"));
02250 g1ProgramUniformMatrix3x2fv =
02251 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fv"));
02252 g1ProgramUniformMatrix3x2fvEXT =
02253 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fvEXT"));
02254 g1ProgramUniformMatrix3x4dv =
02255 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dv"));
02256 g1ProgramUniformMatrix3x4dVEXT =
02257 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dVEXT"));
02258 g1ProgramUniformMatrix3x4fv =
02259 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fv"));
02260 g1ProgramUniformMatrix3x4fvEXT =
02261 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fvEXT"));
02262 g1ProgramUniformMatrix4dv =
02263 PFNGLPROGRAMUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glProgramUniformMatrix4dv"));
02264 g1ProgramUniformMatrix4dVEXT =
```

```

02225 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4dvEXT"));
02226     glProgramUniformMatrix4fv =
02227 PFNGLPROGRAMUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));
02228     glProgramUniformMatrix4fvEXT =
02229 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fvEXT"));
02230     glProgramUniformMatrix4x2dv =
02231 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dv"));
02232     glProgramUniformMatrix4x2dvEXT =
02233 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dvEXT"));
02234     glProgramUniformMatrix4x2fv =
02235 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fv"));
02236     glProgramUniformMatrix4x2fvEXT =
02237 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fvEXT"));
02238     glProgramUniformMatrix4x3dv =
02239 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dv"));
02240     glProgramUniformMatrix4x3dvEXT =
02241 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dvEXT"));
02242     glProgramUniformMatrix4x3fv =
02243 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fv"));
02244     glProgramUniformMatrix4x3fvEXT =
02245 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fvEXT"));
02246     glProgramUniformui64NV =
02247 PFNGLPROGRAMUNIFORMUI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
02248     glProgramUniformui64vNV =
02249 PFNGLPROGRAMUNIFORMI64VNVPROC(glfwGetProcAddress("glProgramUniformui64vNV"));
02250     glProvokingVertex = PFNGLPROVOKINGVERTEXPROC(glfwGetProcAddress("glProvokingVertex"));
02251     glPushClientAttribDefaultEXT =
02252 PFNGLPUSHCLIENTATTRIBDEFAULTTEXTPROC(glfwGetProcAddress("glPushClientAttribDefaultEXT"));
02253     glPushDebugGroup = PFNGLPUSHDEBUGGROUPPROC(glfwGetProcAddress("glPushDebugGroup"));
02254     glPushGroupMarkerEXT = PFNGLPUSHGROUPMARKEREXTPROC(glfwGetProcAddress("glPushGroupMarkerEXT"));
02255     glQueryCounter = PFNGLQUERYCOUNTERPROC(glfwGetProcAddress("glQueryCounter"));
02256     glRasterSamplesEXT = PFNGLRASTERSAMPLESEXTPROC(glfwGetProcAddress("glRasterSamplesEXT"));
02257     glReadBuffer = PFNGLREADBUFFERPROC(glfwGetProcAddress("glReadBuffer"));
02258     glReadPixels = PFNGLREADPIXELSPROC(glfwGetProcAddress("glReadPixels"));
02259     glReadnPixels = PFNGLREADNPPIXELSPROC(glfwGetProcAddress("glReadnPixels"));
02260     glReadnPixelsARB = PFNGLREADNPPIXELSARBPROC(glfwGetProcAddress("glReadnPixelsARB"));
02261     glReleaseShaderCompiler =
02262 PFNGLRELEASESHADERCOMPILERPROC(glfwGetProcAddress("glReleaseShaderCompiler"));
02263     glRenderbufferStorage = PFNGLRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glRenderbufferStorage"));
02264     glRenderbufferStorageMultisample =
02265 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glRenderbufferStorageMultisample"));
02266     glRenderbufferStorageMultisampleCoverageNV =
02267 PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC(glfwGetProcAddress("glRenderbufferStorageMultisampleCoverageNV"));
02268     glResolveDepthValuesNV =
02269 PFNGLRESOLVEDEPTHVALUESNVPROC(glfwGetProcAddress("glResolveDepthValuesNV"));
02270     glResumeTransformFeedback =
02271 PFNGLRESUMETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glResumeTransformFeedback"));
02272     glSampleCoverage = PFNGLSAMPLECOVERAGEPROC(glfwGetProcAddress("glSampleCoverage"));
02273     glSampleMaski = PFNGLSAMPLEMASKIPROC(glfwGetProcAddress("glSampleMaski"));
02274     glSamplerParameterIiv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameterIiv"));
02275     glSamplerParameterIuiv =
02276 PFNGLSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glSamplerParameterIuiv"));
02277     glSamplerParameterf = PFNGLSAMPLERPARAMETERFPROC(glfwGetProcAddress("glSamplerParameterf"));
02278     glSamplerParameterfv = PFNGLSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glSamplerParameterfv"));
02279     glSamplerParameteri = PFNGLSAMPLERPARAMETERIPROC(glfwGetProcAddress("glSamplerParameteri"));
02280     glSamplerParameteriv = PFNGLSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glSamplerParameteriv"));
02281     glScissor = PFNGLSCISSORPROC(glfwGetProcAddress("glScissor"));
02282     glScissorArrayv = PFNGLSCISSORARRAYVPROC(glfwGetProcAddress("glScissorArrayv"));
02283     glScissorIndexed = PFNGLSCISSORINDEXEDPROC(glfwGetProcAddress("glScissorIndexed"));
02284     glScissorIndexedv = PFNGLSCISSORINDEXEDVPROC(glfwGetProcAddress("glScissorIndexedv"));
02285     glSecondaryColorFormatNV =
02286 PFNGLSECONDARYCOLORFORMATNVPROC(glfwGetProcAddress("glSecondaryColorFormatNV"));
02287     glSelectPerfMonitorCountersAMD =
02288 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glSelectPerfMonitorCountersAMD"));
02289     glShaderBinary = PFNGLSHADERBINARYPROC(glfwGetProcAddress("glShaderBinary"));
02290     glShaderSource = PFNGLSHADERSOURCEPROC(glfwGetProcAddress("glShaderSource"));
02291     glShaderStorageBlockBinding =
02292 PFNGLSHADERSTORAGEBLOCKBINDINGPROC(glfwGetProcAddress("glShaderStorageBlockBinding"));
02293     glSignalVkFenceNV = PFNGLSIGNALKVFENCENVPROC(glfwGetProcAddress("glSignalVkFenceNV"));
02294     glSignalVkSemaphoreNV = PFNGLSIGNALKSEMAPHORENVPROC(glfwGetProcAddress("glSignalVkSemaphoreNV"));
02295     glSpecializeShaderARB = PFNGLSPECIALIZESHADERARBPROC(glfwGetProcAddress("glSpecializeShaderARB"));
02296     glStateCaptureNV = PFNGLSTATECAPTURENVPROC(glfwGetProcAddress("glStateCaptureNV"));
02297     glStencilFillPathInstancedNV =
02298 PFNGLSTENCILFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilFillPathInstancedNV"));
02299     glStencilFillPathNV = PFNGLSTENCILFILLPATHNVPROC(glfwGetProcAddress("glStencilFillPathNV"));
02300     glStencilFunc = PFNGLSTENCILFUNCPROC(glfwGetProcAddress("glStencilFunc"));
02301     glStencilFuncSeparate = PFNGLSTENCILFUNCSEPARATEPROC(glfwGetProcAddress("glStencilFuncSeparate"));
02302     glStencilMask = PFNGLSTENCILMASKPROC(glfwGetProcAddress("glStencilMask"));
02303     glStencilMaskSeparate = PFNGLSTENCILMASKSEPARATEPROC(glfwGetProcAddress("glStencilMaskSeparate"));
02304     glStencilOp = PFNGLSTENCILOPPROC(glfwGetProcAddress("glStencilOp"));
02305     glStencilOpSeparate = PFNGLSTENCILOPSEPARATEPROC(glfwGetProcAddress("glStencilOpSeparate"));
02306     glStencilStrokePathInstancedNV =
02307 PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilStrokePathInstancedNV"));
02308     glStencilStrokePathNV = PFNGLSTENCILSTROKEPATHNVPROC(glfwGetProcAddress("glStencilStrokePathNV"));
02309     glStencilThenCoverFillPathInstancedNV =
02310 PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathInstancedNV"));
02311     glStencilThenCoverFillPathNV =

```

```

02286 PFNGLSTENCILTHENCOVERFILLPATHNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathNV"));
02287     g1StencilThenCoverStrokePathInstancedNV =
02288 PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathInstancedNV"));
02289     g1StencilThenCoverStrokePathNV =
02290 PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathNV"));
02291     g1SubpixelPrecisionBiasNV =
02292 PFNGLSUBPIXELPRECISIONBIASNVPROC(glfwGetProcAddress("glSubpixelPrecisionBiasNV"));
02293     g1TexBuffer = PFNGLTEXBUFFERPROC(glfwGetProcAddress("glTexBuffer"));
02294     g1TexBufferARB = PFNGLTEXBUFFERARBPROC(glfwGetProcAddress("glTexBufferARB"));
02295     g1TexBufferRange = PFNGLTEXBUFFERRANGEPROC(glfwGetProcAddress("glTexBufferRange"));
02296     g1TexCoordFormatNV = PFNGLTEXCOORDFORMATNVPROC(glfwGetProcAddress("glTexCoordFormatNV"));
02297     g1TexImage1D = PFNGLTEXIMAGE1DPROC(glfwGetProcAddress("glTexImage1D"));
02298     g1TexImage2D = PFNGLTEXIMAGE2DPROC(glfwGetProcAddress("glTexImage2D"));
02299     g1TexImage2DMultisample =
02300 PFNGLTEXIMAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage2DMultisample"));
02301     g1TexImage3D = PFNGLTEXIMAGE3DPROC(glfwGetProcAddress("glTexImage3D"));
02302     g1TexImage3DMultisample =
02303 PFNGLTEXIMAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage3DMultisample"));
02304     g1TexPageCommitmentARB =
02305 PFNGLTEXPAGECOMMITMENTARBPROC(glfwGetProcAddress("glTexPageCommitmentARB"));
02306     g1TexParameterIiv = PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameterIiv"));
02307     g1TexParameterIuiv = PFNGLTEXPARAMETERUIVPROC(glfwGetProcAddress("glTexParameterIuiv"));
02308     g1TexParameterIfv = PFNGLTEXPARAMETERIFPROC(glfwGetProcAddress("glTexParameterIfv"));
02309     g1TexParameterFv = PFNGLTEXPARAMETERFVPROC(glfwGetProcAddress("glTexParameterFv"));
02310     g1TexParameterIri = PFNGLTEXPARAMETERIPROC(glfwGetProcAddress("glTexParameterIri"));
02311     g1TexParameterIrv = PFNGLTEXPARAMETERIRVPROC(glfwGetProcAddress("glTexParameterIrv"));
02312     g1TexStorage1D = PFNGLTEXTORGE1DPROC(glfwGetProcAddress("glTexStorage1D"));
02313     g1TexStorage2D = PFNGLTEXTORGE2DPROC(glfwGetProcAddress("glTexStorage2D"));
02314     g1TexStorage2DMultisample =
02315 PFNGLTEXTORGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage2DMultisample"));
02316     g1TexStorage3D = PFNGLTEXTORGE3DPROC(glfwGetProcAddress("glTexStorage3D"));
02317     g1TexStorage3DMultisample =
02318 PFNGLTEXTORGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage3DMultisample"));
02319     g1TexSubImage1D = PFNGLTEXSUBIMAGE1DPROC(glfwGetProcAddress("glTexSubImage1D"));
02320     g1TexSubImage2D = PFNGLTEXSUBIMAGE2DPROC(glfwGetProcAddress("glTexSubImage2D"));
02321     g1TexSubImage3D = PFNGLTEXSUBIMAGE3DPROC(glfwGetProcAddress("glTexSubImage3D"));
02322     g1TextureBarrier = PFNGLTEXTUREBARRIERPROC(glfwGetProcAddress("glTextureBarrier"));
02323     g1TextureBarrierNV = PFNGLTEXTUREBARRIERNVPROC(glfwGetProcAddress("glTextureBarrierNV"));
02324     g1TextureBuffer = PFNGLTEXTUREBUFFERPROC(glfwGetProcAddress("glTextureBuffer"));
02325     g1TextureBufferEXT = PFNGLTEXTUREBUFFEREXTPROC(glfwGetProcAddress("glTextureBufferEXT"));
02326     g1TextureBufferRange = PFNGLTEXTUREBUFFERRANGEPROC(glfwGetProcAddress("glTextureBufferRange"));
02327     g1TextureBufferRangeEXT =
02328 PFNGLTEXTUREBUFFERRANGEEXTPROC(glfwGetProcAddress("glTextureBufferRangeEXT"));
02329     g1TextureImage1DEXT = PFNGLTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glTextureImage1DEXT"));
02330     g1TextureImage2DEXT = PFNGLTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glTextureImage2DEXT"));
02331     g1TextureImage3DEXT = PFNGLTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glTextureImage3DEXT"));
02332     g1TexturePageCommitmentEXT =
02333 PFNGLTEXTUREPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glTexturePageCommitmentEXT"));
02334     g1TextureParameterIiv = PFNGLTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glTextureParameterIiv"));
02335     g1TextureParameterIiivEXT =
02336 PFNGLTEXTUREPARAMETERIIIVEXTPROC(glfwGetProcAddress("glTextureParameterIiivEXT"));
02337     g1TextureParameterIuiv =
02338 PFNGLTEXTUREPARAMETERUIVPROC(glfwGetProcAddress("glTextureParameterIuiv"));
02339     g1TextureParameterIuivEXT =
02340 PFNGLTEXTUREPARAMETERUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02341     g1TextureParameterIfv = PFNGLTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glTextureParameterIfv"));
02342     g1TextureParameterFvEXT =
02343 PFNGLTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glTextureParameterFvEXT"));
02344     g1TextureParameterIri = PFNGLTEXTUREPARAMETERIPROC(glfwGetProcAddress("glTextureParameterIri"));
02345     g1TextureParameterIriEXT =
02346 PFNGLTEXTUREPARAMETERIEXTPROC(glfwGetProcAddress("glTextureParameterIriEXT"));
02347     g1TextureParameterIrv = PFNGLTEXTUREPARAMETERIRVPROC(glfwGetProcAddress("glTextureParameterIrv"));
02348     g1TextureParameterIrvEXT =
02349 PFNGLTEXTUREPARAMETERIREXTPROC(glfwGetProcAddress("glTextureParameterIrvEXT"));
02350     g1TextureRenderbufferEXT =
02351 PFNGLTEXTURERENDERBUFFEREXTPROC(glfwGetProcAddress("glTextureRenderbufferEXT"));
02352     g1TextureStorage1D = PFNGLTEXTURERESTORAGE1DPROC(glfwGetProcAddress("glTextureStorage1D"));
02353     g1TextureStorage1DEXT = PFNGLTEXTURERESTORAGE1DEXTPROC(glfwGetProcAddress("glTextureStorage1DEXT"));
02354     g1TextureStorage2D = PFNGLTEXTURERESTORAGE2DPROC(glfwGetProcAddress("glTextureStorage2D"));
02355     g1TextureStorage2DEXT = PFNGLTEXTURERESTORAGE2DEXTPROC(glfwGetProcAddress("glTextureStorage2DEXT"));
02356     g1TextureStorage2DMultisample =
02357 PFNGLTEXTURERESTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage2DMultisample"));
02358     g1TextureStorage2DMultisampleEXT =
02359 PFNGLTEXTURERESTORAGE2DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage2DMultisampleEXT"));
02360     g1TextureStorage3D = PFNGLTEXTURERESTORAGE3DPROC(glfwGetProcAddress("glTextureStorage3D"));
02361     g1TextureStorage3DEXT = PFNGLTEXTURERESTORAGE3DEXTPROC(glfwGetProcAddress("glTextureStorage3DEXT"));
02362     g1TextureStorage3DMultisample =
02363 PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage3DMultisample"));
02364     g1TextureStorage3DMultisampleEXT =
02365 PFNGLTEXTURERESTORAGE3DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage3DMultisampleEXT"));
02366     g1TextureSubImage1D = PFNGLTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glTextureSubImage1D"));
02367     g1TextureSubImage1DEXT =
02368 PFNGLTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glTextureSubImage1DEXT"));
02369     g1TextureSubImage2D = PFNGLTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glTextureSubImage2D"));

```

```

02349    glTextureSubImage2DEXT =
02350        PFNGLTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glTextureSubImage2DEXT"));
02351    glTextureSubImage3D =
02352        PFNGLTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glTextureSubImage3D"));
02353    glTextureSubImage3DEXT =
02354        PFNGLTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glTextureSubImage3DEXT"));
02355    glTextureView =
02356        PFNGLTEXTUREVIEWPROC(glfwGetProcAddress("glTextureView"));
02357    glTransformFeedbackBufferBase =
02358        PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC(glfwGetProcAddress("glTransformFeedbackBufferBase"));
02359    glTransformFeedbackBufferRange =
02360        PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC(glfwGetProcAddress("glTransformFeedbackBufferRange"));
02361    glTransformFeedbackVaryings =
02362        PFNGLTRANSFORMFEEDBACKVARYINGSPROC(glfwGetProcAddress("glTransformFeedbackVaryings"));
02363    glTransformPathNV =
02364        PFNGLTRANSFORMPATHNVPROC(glfwGetProcAddress("glTransformPathNV"));
02365    glUniformId =
02366        PFNGLUNIFORM1DPROC(glfwGetProcAddress("glUniformId"));
02367    glUniformIdfv =
02368        PFNGLUNIFORM1FPROC(glfwGetProcAddress("glUniformIdfv"));
02369    glUniformIdfv =
02370        PFNGLUNIFORM1FVPROC(glfwGetProcAddress("glUniformIdfv"));
02371    glUniformIdfv =
02372        PFNGLUNIFORM1IPROC(glfwGetProcAddress("glUniformIdfv"));
02373    glUniformIdfv =
02374        PFNGLUNIFORM1IVPROC(glfwGetProcAddress("glUniformIdfv"));
02375    glUniformIdfv =
02376        PFNGLUNIFORM1UIPROC(glfwGetProcAddress("glUniformIdfv"));
02377    glUniformIdfv =
02378        PFNGLUNIFORM1UI64ARBPROC(glfwGetProcAddress("glUniformIdfv"));
02379    glUniformIdfv =
02380        PFNGLUNIFORM1UI64NVPROC(glfwGetProcAddress("glUniformIdfv"));
02381    glUniformIdfv =
02382        PFNGLUNIFORM1UI64VARBPROC(glfwGetProcAddress("glUniformIdfv"));
02383    glUniformIdfv =
02384        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02385    glUniformIdfv =
02386        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02387    glUniformIdfv =
02388        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02389    glUniformIdfv =
02390        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02391    glUniformIdfv =
02392        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02393    glUniformIdfv =
02394        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02395    glUniformIdfv =
02396        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02397    glUniformIdfv =
02398        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02399    glUniformIdfv =
02400        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02401    glUniformIdfv =
02402        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02403    glUniformIdfv =
02404        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02405    glUniformIdfv =
02406        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02407    glUniformIdfv =
02408        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02409    glUniformIdfv =
02410        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02411    glUniformIdfv =
02412        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02413    glUniformIdfv =
02414        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02415    glUniformIdfv =
02416        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02417    glUniformIdfv =
02418        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02419    glUniformIdfv =
02420        PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniformIdfv"));
02421    glUniformIdfv =
02422        PFNGLUNIFORMBLOCKBINDINGPROC(glfwGetProcAddress("glUniformBlockBinding"));
02423    glUniformHandleui64ARB =
02424        PFNGLUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glUniformHandleui64ARB"));
02425    glUniformHandleui64NV =
02426        PFNGLUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glUniformHandleui64NV"));
02427    glUniformMatrix2dv =
02428        PFNGLUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glUniformMatrix2dv"));
02429    glUniformMatrix2fv =
02430        PFNGLUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glUniformMatrix2fv"));

```

```

02428 glUniformMatrix2x3dv = PFNGLUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glUniformMatrix2x3dv"));
02429 glUniformMatrix2x3fv = PFNGLUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glUniformMatrix2x3fv"));
02430 glUniformMatrix2x4dv = PFNGLUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glUniformMatrix2x4dv"));
02431 glUniformMatrix2x4fv = PFNGLUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glUniformMatrix2x4fv"));
02432 glUniformMatrix3dv = PFNGLUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glUniformMatrix3dv"));
02433 glUniformMatrix3fv = PFNGLUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glUniformMatrix3fv"));
02434 glUniformMatrix3x2dv = PFNGLUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glUniformMatrix3x2dv"));
02435 glUniformMatrix3x2fv = PFNGLUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glUniformMatrix3x2fv"));
02436 glUniformMatrix3x4dv = PFNGLUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glUniformMatrix3x4dv"));
02437 glUniformMatrix3x4fv = PFNGLUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glUniformMatrix3x4fv"));
02438 glUniformMatrix4dv = PFNGLUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glUniformMatrix4dv"));
02439 glUniformMatrix4fv = PFNGLUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glUniformMatrix4fv"));
02440 glUniformMatrix4x2dv = PFNGLUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glUniformMatrix4x2dv"));
02441 glUniformMatrix4x2fv = PFNGLUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glUniformMatrix4x2fv"));
02442 glUniformMatrix4x3dv = PFNGLUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glUniformMatrix4x3dv"));
02443 glUniformMatrix4x3fv = PFNGLUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glUniformMatrix4x3fv"));
02444 glUniformSubroutinesuiv =
    PFNGLUNIFORMSUBROUTINESUIVPROC(glfwGetProcAddress("glUniformSubroutinesuiv"));
02445 glUniformui64NV = PFNGLUNIFORMUI64NVPROC(glfwGetProcAddress("glUniformui64NV"));
02446 glUniformui64NVv = PFNGLUNIFORMUI64NVVPROC(glfwGetProcAddress("glUniformui64vNV"));
02447 glUnmapBuffer = PFNGLUNMAPBUFFERPROC(glfwGetProcAddress("glUnmapBuffer"));
02448 glUnmapNamedBuffer = PFNGLUNMAPNAMEDBUFFERPROC(glfwGetProcAddress("glUnmapNamedBuffer"));
02449 glUnmapNamedBufferEXT = PFNGLUNMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glUnmapNamedBufferEXT"));
02450 glUseProgram = PFNGLUSEPROGRAMPROC(glfwGetProcAddress("glUseProgram"));
02451 glUseProgramStages = PFNGLUSEPROGRAMSTAGESPROC(glfwGetProcAddress("glUseProgramStages"));
02452 glUseShaderProgramEXT = PFNGLUSESHELDERPROGRAMEXTPROC(glfwGetProcAddress("glUseShaderProgramEXT"));
02453 glValidateProgram = PFNGLVALIDATEPROGRAMPROC(glfwGetProcAddress("glValidateProgram"));
02454 glValidateProgramPipeline =
    PFNGLVALIDATEPROGRAMPIPELINEPROC(glfwGetProcAddress("glValidateProgramPipeline"));
02455 glVertexArrayAttribBinding =
    PFNGLVERTEXARRAYATTRIBBINDINGPROC(glfwGetProcAddress("glVertexArrayAttribBinding"));
02456 glVertexArrayAttribFormat =
    PFNGLVERTEXARRAYATTRIBFORMATPROC(glfwGetProcAddress("glVertexArrayAttribFormat"));
02457 glVertexArrayAttribIFormat =
    PFNGLVERTEXARRAYATTRIBIFORMATPROC(glfwGetProcAddress("glVertexArrayAttribIFormat"));
02458 glVertexArrayAttribLFormat =
    PFNGLVERTEXARRAYATTRIBLFORMATPROC(glfwGetProcAddress("glVertexArrayAttribLFormat"));
02459 glVertexArrayBindVertexBufferEXT =
    PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC(glfwGetProcAddress("glVertexArrayBindVertexBufferEXT"));
02460 glVertexArrayBindingDivisor =
    PFNGLVERTEXARRAYBINDINGDIVISORPROC(glfwGetProcAddress("glVertexArrayBindingDivisor"));
02461 glVertexArrayColorOffsetEXT =
    PFNGLVERTEXARRAYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArrayColorOffsetEXT"));
02462 glVertexArrayEdgeFlagOffsetEXT =
    PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayEdgeFlagOffsetEXT"));
02463 glVertexArrayElementBuffer =
    PFNGLVERTEXARRAYELEMENTBUFFERPROC(glfwGetProcAddress("glVertexArrayElementBuffer"));
02464 glVertexArrayFogCoordOffsetEXT =
    PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayFogCoordOffsetEXT"));
02465 glVertexArrayIndexOffsetEXT =
    PFNGLVERTEXARRAYINDEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayIndexOffsetEXT"));
02466 glVertexArrayMultiTexCoordOffsetEXT =
    PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayMultiTexCoordOffsetEXT"));
02467 glVertexArrayNormalOffsetEXT =
    PFNGLVERTEXARRAYNORMALOFFSETPROC(glfwGetProcAddress("glVertexArrayNormalOffsetEXT"));
02468 glVertexArraySecondaryColorOffsetEXT =
    PFNGLVERTEXARRAYSECONDARYCOLOROFFSETPROC(glfwGetProcAddress("glVertexArraySecondaryColorOffsetEXT"));
02469 glVertexArrayTexCoordOffsetEXT =
    PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayTexCoordOffsetEXT"));
02470 glVertexArrayVertexAttribBindingEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribBindingEXT"));
02471 glVertexArrayVertexAttribDivisorEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribDivisorEXT"));
02472 glVertexArrayVertexAttribFormatEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribFormatEXT"));
02473 glVertexArrayVertexAttribIFormatEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIFormatEXT"));
02474 glVertexArrayVertexAttribIOffsetEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIOffsetEXT"));
02475 glVertexArrayVertexAttribLFormatEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLFormatEXT"));
02476 glVertexArrayVertexAttribLOffsetEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLOffsetEXT"));
02477 glVertexArrayVertexAttribOffsetEXT =
    PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribOffsetEXT"));
02478 glVertexArrayVertexBindingDivisorEXT =
    PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexBindingDivisorEXT"));
02479 glVertexArrayVertexBuffer =
    PFNGLVERTEXARRAYVERTEXBUFFERPROC(glfwGetProcAddress("glVertexArrayVertexBuffer"));
02480 glVertexArrayVertexBuffers =
    PFNGLVERTEXARRAYVERTEXBUFFERSPROC(glfwGetProcAddress("glVertexArrayVertexBuffers"));
02481 glVertexArrayVertexOffsetEXT =
    PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexOffsetEXT"));
02482 glVertexArrayAttribId = PFNGLVERTEXATTRIBIDPROC(glfwGetProcAddress("glVertexArrayAttribId"));
02483 glVertexArrayAttribIdv = PFNGLVERTEXATTRIB1DVPROC(glfwGetProcAddress("glVertexArrayAttribIdv"));
02484 glVertexArrayAttribIf = PFNGLVERTEXATTRIB1FPROC(glfwGetProcAddress("glVertexArrayAttribIf"));
02485 glVertexArrayAttribIfv = PFNGLVERTEXATTRIB1FVPROC(glfwGetProcAddress("glVertexArrayAttribIfv"));

```

```

02486 glVertexAttrib1s = PFNGLVERTEXATTRIB1SPROC(glfwGetProcAddress("glVertexAttrib1s"));
02487 glVertexAttrib1sv = PFNGLVERTEXATTRIB1SVPROC(glfwGetProcAddress("glVertexAttrib1sv"));
02488 glVertexAttrib2d = PFNGLVERTEXATTRIB2DPROC(glfwGetProcAddress("glVertexAttrib2d"));
02489 glVertexAttrib2dv = PFNGLVERTEXATTRIB2DVPROC(glfwGetProcAddress("glVertexAttrib2dv"));
02490 glVertexAttrib2f = PFNGLVERTEXATTRIB2FPROC(glfwGetProcAddress("glVertexAttrib2f"));
02491 glVertexAttrib2fv = PFNGLVERTEXATTRIB2FVPROC(glfwGetProcAddress("glVertexAttrib2fv"));
02492 glVertexAttrib2s = PFNGLVERTEXATTRIB2SPROC(glfwGetProcAddress("glVertexAttrib2s"));
02493 glVertexAttrib2sv = PFNGLVERTEXATTRIB2SVPROC(glfwGetProcAddress("glVertexAttrib2sv"));
02494 glVertexAttrib3d = PFNGLVERTEXATTRIB3DPROC(glfwGetProcAddress("glVertexAttrib3d"));
02495 glVertexAttrib3dv = PFNGLVERTEXATTRIB3DVPROC(glfwGetProcAddress("glVertexAttrib3dv"));
02496 glVertexAttrib3f = PFNGLVERTEXATTRIB3FPROC(glfwGetProcAddress("glVertexAttrib3f"));
02497 glVertexAttrib3fv = PFNGLVERTEXATTRIB3FVPROC(glfwGetProcAddress("glVertexAttrib3fv"));
02498 glVertexAttrib3s = PFNGLVERTEXATTRIB3SPROC(glfwGetProcAddress("glVertexAttrib3s"));
02499 glVertexAttrib3sv = PFNGLVERTEXATTRIB3SVPROC(glfwGetProcAddress("glVertexAttrib3sv"));
02500 glVertexAttrib4Nbv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4Nbv"));
02501 glVertexAttrib4Niv = PFNGLVERTEXATTRIB4NIVPROC(glfwGetProcAddress("glVertexAttrib4Niv"));
02502 glVertexAttrib4Nsv = PFNGLVERTEXATTRIB4NSVPROC(glfwGetProcAddress("glVertexAttrib4Nsv"));
02503 glVertexAttrib4Nub = PFNGLVERTEXATTRIB4NUBPROC(glfwGetProcAddress("glVertexAttrib4Nub"));
02504 glVertexAttrib4Nubv = PFNGLVERTEXATTRIB4NUBVPROC(glfwGetProcAddress("glVertexAttrib4Nubv"));
02505 glVertexAttrib4Nuiv = PFNGLVERTEXATTRIB4NUIVPROC(glfwGetProcAddress("glVertexAttrib4Nuiv"));
02506 glVertexAttrib4Nusv = PFNGLVERTEXATTRIB4USVPROC(glfwGetProcAddress("glVertexAttrib4Usv"));
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 =
PFNGLVERTEXATTRIBDIVISORARBPROC(glfwGetProcAddress("glVertexAttribDivisorARB"));
02521 glVertexAttribFormat = PFNGLVERTEXATTRIBFORMATPROC(glfwGetProcAddress("glVertexAttribFormat"));
02522 glVertexAttribFormatNV =
PFNGLVERTEXATTRIBFORMATNVPROC(glfwGetProcAddress("glVertexAttribFormatNV"));
02523 glVertexAttribI1i = PFNGLVERTEXATTRIBI1IPROC(glfwGetProcAddress("glVertexAttribI1i"));
02524 glVertexAttribI1iv = PFNGLVERTEXATTRIBI1IVPROC(glfwGetProcAddress("glVertexAttribI1iv"));
02525 glVertexAttribI1ui = PFNGLVERTEXATTRIBI1UIPROC(glfwGetProcAddress("glVertexAttribI1ui"));
02526 glVertexAttribI1uiv = PFNGLVERTEXATTRIBI1UIVPROC(glfwGetProcAddress("glVertexAttribI1uiv"));
02527 glVertexAttribI2i = PFNGLVERTEXATTRIBI2IPROC(glfwGetProcAddress("glVertexAttribI2i"));
02528 glVertexAttribI2iv = PFNGLVERTEXATTRIBI2IVPROC(glfwGetProcAddress("glVertexAttribI2iv"));
02529 glVertexAttribI2ui = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2ui"));
02530 glVertexAttribI2uiv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2uiv"));
02531 glVertexAttribI3i = PFNGLVERTEXATTRIBI3IPROC(glfwGetProcAddress("glVertexAttribI3i"));
02532 glVertexAttribI3iv = PFNGLVERTEXATTRIBI3IVPROC(glfwGetProcAddress("glVertexAttribI3iv"));
02533 glVertexAttribI3ui = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3ui"));
02534 glVertexAttribI3uiv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3uiv"));
02535 glVertexAttribI4bv = PFNGLVERTEXATTRIBI4BVPROC(glfwGetProcAddress("glVertexAttribI4bv"));
02536 glVertexAttribI4i = PFNGLVERTEXATTRIBI4IPROC(glfwGetProcAddress("glVertexAttribI4i"));
02537 glVertexAttribI4iv = PFNGLVERTEXATTRIBI4IVPROC(glfwGetProcAddress("glVertexAttribI4iv"));
02538 glVertexAttribI4sv = PFNGLVERTEXATTRIBI4SVPROC(glfwGetProcAddress("glVertexAttribI4sv"));
02539 glVertexAttribI4ubv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4ubv"));
02540 glVertexAttribI4ui = PFNGLVERTEXATTRIBI4UIPROC(glfwGetProcAddress("glVertexAttribI4ui"));
02541 glVertexAttribI4uiv = PFNGLVERTEXATTRIBI4UIVPROC(glfwGetProcAddress("glVertexAttribI4uiv"));
02542 glVertexAttribI4usv = PFNGLVERTEXATTRIBI4USVPROC(glfwGetProcAddress("glVertexAttribI4usv"));
02543 glVertexAttribIFormat = PFNGLVERTEXATTRIBIFORMATPROC(glfwGetProcAddress("glVertexAttribIFormat"));
02544 glVertexAttribIFormatNV =
PFNGLVERTEXATTRIBIFORMATNVPROC(glfwGetProcAddress("glVertexAttribIFormatNV"));
02545 glVertexAttribIPointer =
PFNGLVERTEXATTRIBIPOINTERPROC(glfwGetProcAddress("glVertexAttribIPointer"));
02546 glVertexAttribL1d = PFNGLVERTEXATTRIBL1DPROC(glfwGetProcAddress("glVertexAttribL1d"));
02547 glVertexAttribL1dv = PFNGLVERTEXATTRIBL1DVPROC(glfwGetProcAddress("glVertexAttribL1dv"));
02548 glVertexAttribL1i64NV = PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
02549 glVertexAttribL1i64NV =
PFNGLVERTEXATTRIBL1I64NVNPROC(glfwGetProcAddress("glVertexAttribL1i64vNV"));
02550 glVertexAttribL1ui64ARB =
PFNGLVERTEXATTRIBL1UI64ARBPROC(glfwGetProcAddress("glVertexAttribL1ui64ARB"));
02551 glVertexAttribL1ui64NV =
PFNGLVERTEXATTRIBL1UI64NVPROC(glfwGetProcAddress("glVertexAttribL1ui64NV"));
02552 glVertexAttribL1ui64vARB =
PFNGLVERTEXATTRIBL1UI64VARBPROC(glfwGetProcAddress("glVertexAttribL1ui64vARB"));
02553 glVertexAttribL1ui64vNV =
PFNGLVERTEXATTRIBL1UI64VNVPROC(glfwGetProcAddress("glVertexAttribL1ui64vNV"));
02554 glVertexAttribL2d = PFNGLVERTEXATTRIBL2DPROC(glfwGetProcAddress("glVertexAttribL2d"));
02555 glVertexAttribL2dv = PFNGLVERTEXATTRIBL2DVPROC(glfwGetProcAddress("glVertexAttribL2dv"));
02556 glVertexAttribL2i64NV = PFNGLVERTEXATTRIBL2I64NVPROC(glfwGetProcAddress("glVertexAttribL2i64NV"));
02557 glVertexAttribL2i64NV =
PFNGLVERTEXATTRIBL2I64VNVPROC(glfwGetProcAddress("glVertexAttribL2i64vNV"));
02558 glVertexAttribL2ui64NV =
PFNGLVERTEXATTRIBL2UI64NVPROC(glfwGetProcAddress("glVertexAttribL2ui64NV"));
02559 glVertexAttribL2ui64vNV =
PFNGLVERTEXATTRIBL2UI64VNVPROC(glfwGetProcAddress("glVertexAttribL2ui64vNV"));
02560 glVertexAttribL3d = PFNGLVERTEXATTRIBL3DPROC(glfwGetProcAddress("glVertexAttribL3d"));

```

```

02561     glVertexAttribL3dv = PFNGLVERTEXATTRIBL3DVPROC(glfwGetProcAddress("glVertexAttribL3dv"));
02562     glVertexAttribL3i64NV = PFNGLVERTEXATTRIBL3I64NVPROC(glfwGetProcAddress("glVertexAttribL3i64NV"));
02563     glVertexAttribL3i64vNV =
02564         PFNGLVERTEXATTRIBL3I64VNVPROC(glfwGetProcAddress("glVertexAttribL3i64vNV"));
02565     glVertexAttribL3ui64NV =
02566         PFNGLVERTEXATTRIBL3UI64NVPROC(glfwGetProcAddress("glVertexAttribL3ui64NV"));
02567     glVertexAttribL3ui64vNV =
02568         PFNGLVERTEXATTRIBL3UI64VNVPROC(glfwGetProcAddress("glVertexAttribL3ui64vNV"));
02569     glVertexAttribL4i64NV =
02570         PFNGLVERTEXATTRIBL4I64VNVPROC(glfwGetProcAddress("glVertexAttribL4i64vNV"));
02571     glVertexAttribL4ui64NV =
02572         PFNGLVERTEXATTRIBL4UI64NVPROC(glfwGetProcAddress("glVertexAttribL4ui64NV"));
02573     glVertexAttribLFormatNV =
02574         PFNGLVERTEXATTRIBLFORMATNVPROC(glfwGetProcAddress("glVertexAttribLFormatNV"));
02575     glVertexAttribLPointer =
02576         PFNGLVERTEXATTRIBLPOINTERPROC(glfwGetProcAddress("glVertexAttribLPointer"));
02577     glVertexAttribP1ui = PFNGLVERTEXATTRIBP1UIPROC(glfwGetProcAddress("glVertexAttribP1ui"));
02578     glVertexAttribP1uiv = PFNGLVERTEXATTRIBP1UIVPROC(glfwGetProcAddress("glVertexAttribP1uiv"));
02579     glVertexAttribP2ui = PFNGLVERTEXATTRIBP2UIPROC(glfwGetProcAddress("glVertexAttribP2ui"));
02580     glVertexAttribP2uiv = PFNGLVERTEXATTRIBP2UIVPROC(glfwGetProcAddress("glVertexAttribP2uiv"));
02581     glVertexAttribP3ui = PFNGLVERTEXATTRIBP3UIPROC(glfwGetProcAddress("glVertexAttribP3ui"));
02582     glVertexAttribP3uiv = PFNGLVERTEXATTRIBP3UIVPROC(glfwGetProcAddress("glVertexAttribP3uiv"));
02583     glVertexAttribP4ui = PFNGLVERTEXATTRIBP4UIPROC(glfwGetProcAddress("glVertexAttribP4ui"));
02584     glVertexAttribP4uiv = PFNGLVERTEXATTRIBP4UIVPROC(glfwGetProcAddress("glVertexAttribP4uiv"));
02585     glVertexAttribPointer = PFNGLVERTEXATTRIBPOINTERPROC(glfwGetProcAddress("glVertexAttribPointer"));
02586     glVertexBindingDivisor =
02587         PFNGLVERTEXBINDINGDIVISORPROC(glfwGetProcAddress("glVertexBindingDivisor"));
02588     glVertexFormatNV = PFNGLVERTEXFORMATNVPROC(glfwGetProcAddress("glVertexFormatNV"));
02589     glViewport = PFNGLVIEWPORTPROC(glfwGetProcAddress("glViewport"));
02590     glViewportArrayv = PFNGLVIEWPORTARRAYVPROC(glfwGetProcAddress("glViewportArrayv"));
02591     glViewportIndexeddf = PFNGLVIEWPORTINDEXEDDFPROC(glfwGetProcAddress("glViewportIndexeddf"));
02592     glViewportIndexeddfv = PFNGLVIEWPORTINDEXEDDFVPROC(glfwGetProcAddress("glViewportIndexeddfv"));
02593     glViewportPositionWScaleNV =
02594         PFNGLVIEWPORTPOSITIONWSCALENVPROC(glfwGetProcAddress("glViewportPositionWScaleNV"));
02595     glViewportSwizzleNV = PFNGLVIEWPORTSWIZZLENVPROC(glfwGetProcAddress("glViewportSwizzleNV"));
02596 #endif
02597
02598 // 使用している GPU のバッファアライメントを調べる
02599     glGetIntegerv(GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT, &ggBufferAlignment);
02600 }
02601
02602 // OpenGL のエラーをチェックする
02603 // 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
02604 //
02605 // 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
02606 //
02607 // 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
02608 //
02609 void gg::ggError(const std::string& name, unsigned int line)
02610 {
02611     const GLenum error{ glGetError() };
02612
02613     if (error != GL_NO_ERROR)
02614     {
02615         if (!name.empty())
02616         {
02617             std::cerr << name;
02618             if (line > 0) std::cerr << " (" << line << ")";
02619             std::cerr << ": ";
02620         }
02621
02622         switch (error)
02623         {
02624             case GL_INVALID_ENUM:
02625                 std::cerr << "An unacceptable value is specified for an enumerated argument" << std::endl;
02626                 break;
02627             case GL_INVALID_VALUE:
02628                 std::cerr << "A numeric argument is out of range" << std::endl;
02629                 break;
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 occured: " << std::hex << std::showbase << error << std::endl;
02641     break;
02642 }
02643 }
02645
02646 //
02647 // FBO のエラーをチェックする
02648 //
02649 //   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
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 //
02719 // 変換行列：行列と行列の積 c ← a × b
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],
02886         marray[ 4],
02887         marray[ 8],
02888         marray[12],
02889         marray[ 1],
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{ sqrtf(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{ sqrtf(xx * xx + xy * xy + xz * xz) };
03044
03045     // y 軸 = z 軸 × x 軸
03046     const auto yx{ zy * xz - 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{ sqrtf(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
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 //
03241 // 四元数：回転変換行列 a が表す四元数を q に求める
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{ sqrtf(ss) };
03277         const auto ph{ acosf(qr) };
03278         const auto pt{ ph * t };
03279         const auto t1{ sinf(pt) / sp };
03280         const auto t0{ sinf(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 }
```

```

03333 }
03334
03335 // z 軸を中心に角度 a 回転する四元数を格納する
03336 // GgQuaternion& GgQuaternion::loadRotateZ(GLfloat a)
03337 {
03338     const auto t{ a * 0.5f };
03339     *this = GgQuaternion{ 0.0f, 0.0f, sinf(t), cosf(t) };
03340
03341     return *this;
03342
03343
03344
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 GgQuaternion& 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 GgQuaternion& 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 GgQuaternion& 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 GgQuaternion& 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 //
03414 // 簡易トラックボール処理: リセット
03415 //
03416 void GgTrackball::reset(const GgQuaternion& q)
03417 {
03418     // ドラッグ中ではない
03419     drag = false;

```

```
03420 // 単位クォーテーニオンに初期値を与える
03421 *this = cq = q;
03422
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 // マウスポ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 // マウスポ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 // マウスポ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 // マウスポ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転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 // 簡易トラックボール処理：停止時の処理
03508 // マウスボタンを離したときに実行する
03509 //
03510 // (x, y) 現在のマウス位置
03511 //
03512 //
03513 void gg::GgTrackball::end(GLfloat x, GLfloat y)
03514 {
03515     // ドラッグ終了点における回転を求める
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()) return false;
03575
03576     // データを書き込む
03577     const unsigned int size{ width * height * depth };
03578     if (type == 2)
03579     {
03580         // フルカラー
03581         std::vector<char> temp(size);
03582         for (GLuint i = 0; i < size; i += depth)
03583         {
03584             temp[i + 2] = static_cast<const char*>(buffer)[i + 0];
03585             temp[i + 1] = static_cast<const char*>(buffer)[i + 1];
03586             temp[i + 0] = static_cast<const char*>(buffer)[i + 2];
03587             if (depth == 4) temp[i + 3] = static_cast<const char*>(buffer)[i + 3];
03588         }
03589         file.write(temp.data(), size);
03590     }
03591     else if (type == 3)
03592     {
03593         // グレースケール

```

```
03594     file.write(static_cast<const char*>(buffer), size);
03595 }
03596
03597 // フッタを書き込む
03598 constexpr char footer[] = "\0\0\0\0\0\0\0\0\0TRUEVISION-XFILE.";
03599 file.write(footer, sizeof footer);
03600
03601 // データの書き込みに失敗していなければ true を返す
03602 return file.bad() != false;
03603 }
03604
03605 //
03606 // カラーバッファの内容を TGA ファイルに保存する
03607 //
03608 //   name 保存するファイル名
03609 //   戻り値 保存に成功すれば true, 失敗すれば false
03610 //
03611 bool gg::ggSaveColor(const std::string& name)
03612 {
03613     // 現在のビューポートのサイズを得る
03614     GLint viewport[4];
03615     glGetIntegerv(GL_VIEWPORT, viewport);
03616
03617     // ビューポートのサイズ分のメモリを確保する
03618     std::vector<GLubyte> buffer(viewport[2] * viewport[3] * 3);
03619
03620     // 画面表示の完了を待つ
03621     glFinish();
03622
03623     // カラーバッファを読み込む
03624     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03625                 GL_RGB, GL_UNSIGNED_BYTE, buffer.data());
03626
03627     // 読み込んだデータをファイルに書き込む
03628     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 3);
03629 }
03630
03631 //
03632 // デプスバッファの内容を TGA ファイルに保存する
03633 //
03634 //   name 保存するファイル名
03635 //   戻り値 保存に成功すれば true, 失敗すれば false
03636 //
03637 bool gg::ggSaveDepth(const std::string& name)
03638 {
03639     // 現在のビューポートのサイズを得る
03640     GLint viewport[4];
03641     glGetIntegerv(GL_VIEWPORT, viewport);
03642
03643     // ビューポートのサイズ分のメモリを確保する
03644     std::vector<GLubyte> buffer(viewport[2] * viewport[3]);
03645
03646     // 画面表示の完了を待つ
03647     glFinish();
03648
03649     // デプスバッファを読み込む
03650     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03651                 GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, buffer.data());
03652
03653     // 読み込んだデータをファイルに書き込む
03654     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 1);
03655 }
03656
03657 //
03658 // TGA ファイル (8/16/24/32bit) を読み込む
03659 //
03660 //   name 読み込むファイル名
03661 //   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)
03662 //   pHeight 読み込んだファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)
03663 //   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)
03664 //   image 読み込んだ画像を格納する vector
03665 //   戻り値 読み込みに成功すれば true, 失敗すれば false
03666 //
03667 bool gg::ggReadImage(
03668     const std::string& name,
03669     std::vector<GLubyte>& image,
03670     GLsizei* pWidth,
03671     GLsizei* pHeight,
03672     GLenum* pFormat
03673 )
03674 {
03675     // ファイルを開く
03676     std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
03677
03678     // ファイルが開けなかったら戻る
03679     if (file.fail()) return false;
03680 }
```

```

03681 // ヘッダを読み込む
03682 unsigned char header[18];
03683 file.read(reinterpret_cast<char*>(header), sizeof header);
03684
03685 // ヘッダの読み込みに失敗したら戻る
03686 if (file.bad()) return false;
03687
03688 // 深度
03689 const auto depth{ header[16] / 8 };
03690 switch (depth)
03691 {
03692     case 1:
03693         *pFormat = GL_RED;
03694         break;
03695     case 2:
03696         *pFormat = GL_RG;
03697         break;
03698     case 3:
03699         *pFormat = GL_BGR;
03700         break;
03701     case 4:
03702         *pFormat = GL_BGRA;
03703         break;
03704     default:
03705         // 取り扱えないフォーマットだったら戻る
03706         return false;
03707 }
03708
03709 // 画像の縦横の画素数
03710 *pWidth = header[13] << 8 | header[12];
03711 *pHeight = header[15] << 8 | header[14];
03712
03713 // データサイズ
03714 const auto size{ *pWidth * *pHeight * depth };
03715
03716 // サイズが小さすぎたら戻る
03717 if (size < 2) return false;
03718
03719 // 読み込みに使うメモリを確保する
03720 image.resize(size);
03721
03722 // データを読み込む
03723 if (header[2] & 8)
03724 {
03725     // RLE
03726     int p{ 0 };
03727     char c;
03728     while (file.get(c))
03729     {
03730         if (c & 0x80)
03731         {
03732             // run-length packet
03733             const auto count{ (c & 0x7f) + 1 };
03734             if (p + depth * count > size) break;
03735             char temp[4];
03736             file.read(temp, depth);
03737             for (int i = 0; i < count; ++i)
03738             {
03739                 for (int j = 0; j < depth;) image[p++] = temp[j++];
03740             }
03741         }
03742         else
03743         {
03744             // raw packet
03745             const auto count{ (c + 1) * depth };
03746             if (p + count > size) break;
03747             file.read(reinterpret_cast<char*>(image.data() + p), count);
03748             p += count;
03749         }
03750     }
03751 }
03752 else
03753 {
03754     // 非圧縮
03755     file.read(reinterpret_cast<char*>(image.data()), size);
03756 }
03757
03758 // 読み込みに失敗していなければ true を返す
03759 return file.bad() != false;
03760 }
03761
03762 //
03763 // テクスチャを作成して画像を読み込む
03764 //
03765 //    image 画像データ, nullptr ならメモリの確保だけを行う
03766 //    width 画像の横の画素数
03767 //    height 画像の縦の画素数

```

```

03768 // format 画像データのフォーマット
03769 // type 画像のデータ型
03770 // internal テクスチャの内部フォーマット
03771 // wrap テクスチャのラッピングモード
03772 // swizzle true ならテクスチャの赤と青を入れ替える
03773 // 戻り値 テクスチャ名
03774 //
03775 GLuint gg::ggLoadTexture(
03776     const GLvoid* image,
03777     GLsizei width,
03778     GLsizei height,
03779     GLenum format,
03780     GLenum type,
03781     GLenum internal,
03782     GLenum wrap,
03783     bool swizzle
03784 )
03785 {
03786     // テクスチャを作成する
03787     GLuint texture;
03788     glGenTextures(1, &texture);
03789
03790     // 作成したテクスチャを 2D テクスチャとしてバインドする
03791     glBindTexture(GL_TEXTURE_2D, texture);
03792
03793     // 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
03794     glPixelStorei(GL_UNPACK_ALIGNMENT, format == GL_RGBA || format == GL_BGRA ? 4 : 1);
03795
03796     // テクスチャを割り当てる
03797     glTexImage2D(GL_TEXTURE_2D, 0, internal, width, height, 0, format, type, image);
03798
03799     // バイリニア（ミップマップなし）, エッジでクランプ
03800     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
03801     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
03802     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap);
03803     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);
03804
03805     if (swizzle)
03806     {
03807         // テクスチャのサンプリング時に赤と青を入れ替える
03808         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, GL_BLUE);
03809         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, GL_RED);
03810     }
03811
03812     // テクスチャ名を返す
03813     return texture;
03814 }
03815
03816 //
03817 // 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
03818 //
03819 // name TGA ファイル名
03820 // 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)
03821 // 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)
03822 // internal テクスチャの内部フォーマット
03823 // wrap テクスチャのラッピングモード
03824 // 戻り値 テクスチャ名
03825 //
03826 GLuint gg::ggLoadImage(
03827     const std::string& name,
03828     GLsizei* pWidth,
03829     GLsizei* pHeight,
03830     GLenum internal,
03831     GLenum wrap
03832 )
03833 {
03834     // 画像データ
03835     std::vector<GLubyte> image;
03836
03837     // 画像サイズ
03838     GLsizei width, height;
03839
03840     // 画像フォーマット
03841     GLenum format;
03842
03843     // 画像を読み込む
03844     ggReadImage(name, image, &width, &height, &format);
03845
03846     // 画像が読み込めなかつたら戻る
03847     if (image.empty()) return 0;
03848
03849     // 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
03850     if (internal == 0) internal = format;
03851
03852     // テクスチャに読み込む (ggReadImage() で読み込んだ画像は GL_BGR / GL_BGRA)
03853     const auto tex{ ggLoadTexture(image.data(), width, height,
03854         format, GL_UNSIGNED_BYTE, internal, wrap, false) };

```

```

03855 // 画像サイズを返す
03856 if (pWidth) *pWidth = width;
03857 if (pHeight) *pHeight = height;
03858
03859 // テクスチャ名を返す
03860 return tex;
03861 }
03862 }
03863
03864 //
03865 // 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
03866 //
03867 // width 高さマップのグレースケール画像 hmap の横の画素数
03868 // 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
03869 // stride データの間隔
03870 // hmap グレースケール画像のデータ
03871 // nz 法線の z 成分の割合
03872 // internal テクスチャの内部フォーマット
03873 // nmap 法線マップを格納する vector
03874 //
03875 void gg::ggCreateNormalMap(
03876     const GLubyte* hmap,
03877     GLsizei width,
03878     GLsizei height,
03879     GLenum format,
03880     GLfloat nz,
03881     GLenum internal,
03882     std::vector<GgVector>& nmap
03883 )
03884 {
03885 // メモリサイズ
03886 const GLsizei size{ width * height };
03887
03888 // 法線マップのメモリを確保する
03889 nmap.resize(size);
03890
03891 // 画素のバイト数
03892 GLint stride;
03893 switch (format)
03894 {
03895 case GL_RED:
03896     stride = 1;
03897     break;
03898 case GL_RGB:
03899     stride = 2;
03900     break;
03901 case GL_RGB:
03902     stride = 3;
03903     break;
03904 case GL_RGBA:
03905     stride = 4;
03906     break;
03907 default:
03908     stride = 1;
03909     break;
03910 }
03911
03912 // 法線マップの作成
03913 for (GLsizei i = 0; i < size; ++i)
03914 {
03915     const auto x{ i % width };
03916     const auto y{ i - x };
03917     const auto u0{ (y + (x - 1 + width) % width) * stride };
03918     const auto u1{ (y + (x + 1) % width) * stride };
03919     const auto v0{ ((y - width + size) % size + x) * stride };
03920     const auto v1{ ((y + width) % size + x) * stride };
03921
03922 //隣接する画素との値の差を法線の成分に用いる
03923 nmap[i][0] = static_cast<GLfloat>(hmap[u0] - hmap[u1]);
03924 nmap[i][1] = static_cast<GLfloat>(hmap[v0] - hmap[v1]);
03925 nmap[i][2] = nz;
03926 nmap[i][3] = hmap[i * stride];
03927
03928 // 法線ベクトルを正規化する
03929 ggNormalize3(&nmap[i]);
03930 }
03931
03932 // 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
03933 if (
03934     internal != GL_RGB16F &&
03935     internal != GL_RGBA16F &&
03936     internal != GL_RGB32F &&
03937     internal != GL_RGBA32F
03938 )
03939 {
03940     for (GLsizei i = 0; i < size; ++i)
03941     {

```

```

03942     nmap[i][0] = nmap[i][0] * 0.5f + 0.5f;
03943     nmap[i][1] = nmap[i][1] * 0.5f + 0.5f;
03944     nmap[i][2] = nmap[i][2] * 0.5f + 0.5f;
03945     nmap[i][3] *= 0.0039215686f; // == 1/255
03946 }
03947 }
03948 }
03949
03950 //
03951 // 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
03952 //
03953 // name TGA ファイル名
03954 // nz 作成した法線の z 成分の割合
03955 // 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)
03956 // 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)
03957 // internal テクスチャの内部フォーマット
03958 // 戻り値 テクスチャ名
03959 //
03960 GLuint gg::ggLoadHeight(
03961     const std::string& name,
03962     GLfloat nz,
03963     GLsizei* pWidth,
03964     GLsizei* pHeight,
03965     GLenum internal
03966 )
03967 {
03968     // 画像データ
03969     std::vector<GLubyte> hmap;
03970
03971     // 画像サイズ
03972     GLsizei width, height;
03973
03974     // 画像フォーマット
03975     GLenum format;
03976
03977     // 高さマップの画像を読み込む
03978     ggReadImage(name, hmap, &width, &height, &format);
03979
03980     // 画像が読み込めなかったら戻る
03981     if (hmap.empty()) return 0;
03982
03983     // 法線マップ
03984     std::vector<GgVector> nmap;
03985
03986     // 法線マップを作成する
03987     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
03988
03989     // 画像サイズを返す
03990     if (pWidth) *pWidth = width;
03991     if (pHeight) *pHeight = height;
03992
03993     // テクスチャを作成して返す
03994     return ggLoadTexture(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal, GL_REPEAT);
03995 }
03996
03997 //
03998 // テクスチャの赤と青を交換する
03999 //
04000 void gg::GgTexture::swapRnB(bool swap) const
04001 {
04002     const auto swizzle
04003     {
04004         swap ?
04005             std::pair<GLint, GLint>{ GL_BLUE, GL_RED } :
04006             std::pair<GLint, GLint>{ GL_RED, GL_BLUE }
04007     };
04008
04009     glBindTexture(GL_TEXTURE_2D, texture);
04010     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, swizzle.first);
04011     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, swizzle.second);
04012     glBindTexture(GL_TEXTURE_2D, 0);
04013 }
04014
04015 //
04016 // 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
04017 //
04018 // name 読み込むファイル名
04019 // 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
04020 // 戻り値 テクスチャの作成に成功すれば true, 失敗すれば false
04021 //
04022 void gg::GgColorTexture::load(
04023     const std::string& name,
04024     GLenum internal,
04025     GLenum wrap
04026 )
04027 {
04028     // 画像データ

```

```

04029     std::vector<GLubyte> image;
04030
04031     // 画像サイズ
04032     GLsizei width, height;
04033
04034     // 画像フォーマット
04035     GLenum format;
04036
04037     // 画像を読み込む
04038     ggReadImage(name, image, &width, &height, &format);
04039
04040     // 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
04041     if (internal == 0) internal = format;
04042
04043     // テクスチャを作成する (ggReadImage() で読み込んだ画像は GL_BGR / GL_BGRA)
04044     texture = std::make_shared<GgTexture>(image.data(), width, height,
04045         format, GL_UNSIGNED_BYTE, internal, wrap, false);
04046 }
04047
04048 //
04049 // 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
04050 //
04051 //   name 画像ファイル名
04052 //   width テクスチャとして用いる画像データの横幅
04053 //   height テクスチャとして用いる画像データの高さ
04054 //   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)
04055 //   nz 法線マップの z 成分の値
04056 //   internal テクスチャの内部フォーマット
04057 //
04058 void gg::GgNormalTexture::load(
04059     const std::string& name,
04060     GLfloat nz,
04061     GLenum internal
04062 )
04063 {
04064     // 高さマップ
04065     std::vector<GLubyte> hmap;
04066
04067     // 画像サイズ
04068     GLsizei width, height;
04069
04070     // 画像フォーマット
04071     GLenum format;
04072
04073     // 高さマップの画像を読み込む
04074     ggReadImage(name, hmap, &width, &height, &format);
04075
04076     // 法線マップ
04077     std::vector<GgVector> nmap;
04078
04079     // 法線マップを作成する
04080     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
04081 }
04082
04083
04084
04085 //
04086 // OBJ ファイルの読み込みに使うデータ型と関数
04087 //
04088 namespace gg
04089 {
04090     // GLfloat 型の 2 要素のベクトル
04091     using vec2 = std::array<GLfloat, 2>;
04092
04093     // GLfloat 型の 3 要素のベクトル
04094     using vec3 = std::array<GLfloat, 3>;
04095
04096     // 三角形データ
04097     struct fidx
04098     {
04099         GLuint p[3];           // 頂点座標番号
04100         GLuint n[3];           // 頂点法線番号
04101         GLuint t[3];           // テクスチャ座標番号
04102         bool smooth;          // スムーズシェーディングの有無
04103     };
04104
04105     // ポリゴングループ
04106     struct fgrp
04107     {
04108         GLuint nextgroup;    // 次のポリゴングループの最初の三角形番号
04109         GLuint mtlno;         // このポリゴングループの材質番号
04110
04111         // コンストラクタ
04112         fgrp(GLuint nextgroup, GLuint mtlno) :
04113             nextgroup(nextgroup),
04114             mtlno(mtlno)
04115         {
04116     }

```

```
04117    };
04118
04119    // デフォルトの材質
04120    constexpr GgSimpleShader::Material defaultMaterial
04121    {
04122        { 0.1f, 0.1f, 0.1f, 1.0f },
04123        { 0.6f, 0.6f, 0.6f, 0.0f },
04124        { 0.3f, 0.3f, 0.3f, 0.0f },
04125        60.0f
04126    };
04127
04128    // デフォルトの材質名
04129    constexpr char defaultMaterialName[] = ".default.";
04130
04131    //
04132    // Alias OBJ 形式の MTL ファイルを読み込む
04133    //
04134    //     mtlpath MTL ファイルのパス名
04135    //     mtl 読み込んだ材質名をキー、材質番号を値にした map
04136    //     material 材質データ
04137    //
04138    static bool ggLoadMtl(const std::string& mtlpath,
04139        std::map<std::string, GLuint>& mtl,
04140        std::vector<GgSimpleShader::Material>& material)
04141    {
04142        // MTL ファイルが無ければ戻る
04143        std::ifstream mtlfile{ Utf8ToTChar(mtlpath), std::ios::binary };
04144        if (!mtlfile)
04145        {
04146            #if defined(DEBUG)
04147                std::cerr << "Warning: Can't open MTL file: " << mtlpath << std::endl;
04148            #endif
04149            return false;
04150        }
04151
04152        // 一行読み込み用のバッファ
04153        std::string mtlline;
04154
04155        // 材質名（ループの外に置く）
04156        std::string mtlname{ defaultMaterialName };
04157
04158        // 現在の材質番号を登録する
04159        mtl[mtlname] = static_cast<GLuint>(material.size());
04160
04161        // 現在の材質にデフォルトの材質を設定する
04162        material.emplace_back(defaultMaterial);
04163
04164        // 材質データを読み込む
04165        while (std::getline(mtlfile, mtlline))
04166        {
04167            // 空行は読み飛ばす
04168            if (mtlline == "") continue;
04169
04170            // 最後の文字が '\r' なら
04171            if (*(mtlline.end() - 1) == '\r')
04172            {
04173                // 最後の文字を削除する
04174                mtlline.erase(mtlline.end() - 1, mtlline.end());
04175
04176                // 空行になら読み飛ばす
04177                if (mtlline == "") continue;
04178            }
04179
04180            // 読み込んだ行を文字列ストリームにする
04181            std::istringstream mtlstr{ mtlline };
04182
04183            // オペレータ
04184            std::string mtlop;
04185
04186            // 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
04187            mtlstr >> mtlop;
04188
04189            // '#' で始まる場合はコメントとして行末まで読み飛ばす
04190            if (mtlop[0] == '#') continue;
04191
04192            if (mtlop == "newmtl")
04193            {
04194                // 新規作成する材質名を取り出す
04195                mtlstr >> mtlname;
04196
04197                // 材質名が既に存在するかどうか調べる
04198                const auto m{ mtl.find(mtlname) };
04199                if (m == mtl.end())
04200                {
04201                    // 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
04202                    mtl[mtlname] = static_cast<GLuint>(material.size());
04203                }
04204            }
04205        }
04206    }
04207
04208    // マテリアルを返す
04209    Material getMaterial(GLuint index) const
04210    {
04211        return material[index];
04212    }
04213
04214    // MTL ファイルを読み込む
04215    void loadMtl(const std::string& mtlpath)
04216    {
04217        ggLoadMtl(mtlpath, mtl, material);
04218    }
04219
04220    // マテリアルを登録する
04221    void registerMaterial(GLuint index, const Material& material)
04222    {
04223        material[index] = material;
04224    }
04225
04226    // マテリアルを削除する
04227    void removeMaterial(GLuint index)
04228    {
04229        material.erase(index);
04230    }
04231
04232    // マテリアルを更新する
04233    void updateMaterial(GLuint index, const Material& material)
04234    {
04235        material[index] = material;
04236    }
04237
04238    // マテリアルをクリアする
04239    void clearMaterial()
04240    {
04241        material.clear();
04242    }
04243
04244    // マテリアルを初期化する
04245    void initializeMaterial()
04246    {
04247        material.clear();
04248        material.push_back(defaultMaterial);
04249    }
04250
04251    // マテリアルを初期化する
04252    void initializeMaterial(GLuint count)
04253    {
04254        material.clear();
04255        material.reserve(count);
04256        material.push_back(defaultMaterial);
04257    }
04258
04259    // マテリアルを初期化する
04260    void initializeMaterial(GLuint count, const Material& material)
04261    {
04262        material.clear();
04263        material.reserve(count);
04264        material.push_back(material);
04265    }
04266
04267    // マテリアルを初期化する
04268    void initializeMaterial(GLuint count, const std::vector<Material>& materials)
04269    {
04270        material.clear();
04271        material.reserve(count);
04272        material.insert(material.end(), materials.begin(), materials.end());
04273    }
04274
04275    // マテリアルを初期化する
04276    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const Material& defaultMaterial)
04277    {
04278        material.clear();
04279        material.reserve(count);
04280        material.insert(material.end(), materials.begin(), materials.end());
04281        material.push_back(defaultMaterial);
04282    }
04283
04284    // マテリアルを初期化する
04285    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials)
04286    {
04287        material.clear();
04288        material.reserve(count);
04289        material.insert(material.end(), materials.begin(), materials.end());
04290        material.push_back(defaultMaterials[0]);
04291    }
04292
04293    // マテリアルを初期化する
04294    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const Material& newmtl)
04295    {
04296        material.clear();
04297        material.reserve(count);
04298        material.insert(material.end(), materials.begin(), materials.end());
04299        material.push_back(newmtl);
04300    }
04301
04302    // マテリアルを初期化する
04303    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls)
04304    {
04305        material.clear();
04306        material.reserve(count);
04307        material.insert(material.end(), materials.begin(), materials.end());
04308        material.push_back(newmtls[0]);
04309    }
04310
04311    // マテリアルを初期化する
04312    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const Material& newmtl)
04313    {
04314        material.clear();
04315        material.reserve(count);
04316        material.insert(material.end(), materials.begin(), materials.end());
04317        material.push_back(newmtl);
04318    }
04319
04320    // マテリアルを初期化する
04321    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2)
04322    {
04323        material.clear();
04324        material.reserve(count);
04325        material.insert(material.end(), materials.begin(), materials.end());
04326        material.push_back(newmtls2[0]);
04327    }
04328
04329    // マテリアルを初期化する
04330    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const Material& newmtl)
04331    {
04332        material.clear();
04333        material.reserve(count);
04334        material.insert(material.end(), materials.begin(), materials.end());
04335        material.push_back(newmtl);
04336    }
04337
04338    // マテリアルを初期化する
04339    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3)
04340    {
04341        material.clear();
04342        material.reserve(count);
04343        material.insert(material.end(), materials.begin(), materials.end());
04344        material.push_back(newmtls3[0]);
04345    }
04346
04347    // マテリアルを初期化する
04348    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const Material& newmtl)
04349    {
04350        material.clear();
04351        material.reserve(count);
04352        material.insert(material.end(), materials.begin(), materials.end());
04353        material.push_back(newmtl);
04354    }
04355
04356    // マテリアルを初期化する
04357    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4)
04358    {
04359        material.clear();
04360        material.reserve(count);
04361        material.insert(material.end(), materials.begin(), materials.end());
04362        material.push_back(newmtls4[0]);
04363    }
04364
04365    // マテリアルを初期化する
04366    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const Material& newmtl)
04367    {
04368        material.clear();
04369        material.reserve(count);
04370        material.insert(material.end(), materials.begin(), materials.end());
04371        material.push_back(newmtl);
04372    }
04373
04374    // マテリアルを初期化する
04375    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5)
04376    {
04377        material.clear();
04378        material.reserve(count);
04379        material.insert(material.end(), materials.begin(), materials.end());
04380        material.push_back(newmtls5[0]);
04381    }
04382
04383    // マテリアルを初期化する
04384    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const Material& newmtl)
04385    {
04386        material.clear();
04387        material.reserve(count);
04388        material.insert(material.end(), materials.begin(), materials.end());
04389        material.push_back(newmtl);
04390    }
04391
04392    // マテリアルを初期化する
04393    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6)
04394    {
04395        material.clear();
04396        material.reserve(count);
04397        material.insert(material.end(), materials.begin(), materials.end());
04398        material.push_back(newmtls6[0]);
04399    }
04400
04401    // マテリアルを初期化する
04402    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const Material& newmtl)
04403    {
04404        material.clear();
04405        material.reserve(count);
04406        material.insert(material.end(), materials.begin(), materials.end());
04407        material.push_back(newmtl);
04408    }
04409
04410    // マテリアルを初期化する
04411    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7)
04412    {
04413        material.clear();
04414        material.reserve(count);
04415        material.insert(material.end(), materials.begin(), materials.end());
04416        material.push_back(newmtls7[0]);
04417    }
04418
04419    // マテリアルを初期化する
04420    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const Material& newmtl)
04421    {
04422        material.clear();
04423        material.reserve(count);
04424        material.insert(material.end(), materials.begin(), materials.end());
04425        material.push_back(newmtl);
04426    }
04427
04428    // マテリアルを初期化する
04429    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8)
04430    {
04431        material.clear();
04432        material.reserve(count);
04433        material.insert(material.end(), materials.begin(), materials.end());
04434        material.push_back(newmtls8[0]);
04435    }
04436
04437    // マテリアルを初期化する
04438    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const Material& newmtl)
04439    {
04440        material.clear();
04441        material.reserve(count);
04442        material.insert(material.end(), materials.begin(), materials.end());
04443        material.push_back(newmtl);
04444    }
04445
04446    // マテリアルを初期化する
04447    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9)
04448    {
04449        material.clear();
04450        material.reserve(count);
04451        material.insert(material.end(), materials.begin(), materials.end());
04452        material.push_back(newmtls9[0]);
04453    }
04454
04455    // マテリアルを初期化する
04456    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const Material& newmtl)
04457    {
04458        material.clear();
04459        material.reserve(count);
04460        material.insert(material.end(), materials.begin(), materials.end());
04461        material.push_back(newmtl);
04462    }
04463
04464    // マテリアルを初期化する
04465    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10)
04466    {
04467        material.clear();
04468        material.reserve(count);
04469        material.insert(material.end(), materials.begin(), materials.end());
04470        material.push_back(newmtls10[0]);
04471    }
04472
04473    // マテリアルを初期化する
04474    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const Material& newmtl)
04475    {
04476        material.clear();
04477        material.reserve(count);
04478        material.insert(material.end(), materials.begin(), materials.end());
04479        material.push_back(newmtl);
04480    }
04481
04482    // マテリアルを初期化する
04483    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const std::vector<Material>& newmtls11)
04484    {
04485        material.clear();
04486        material.reserve(count);
04487        material.insert(material.end(), materials.begin(), materials.end());
04488        material.push_back(newmtls11[0]);
04489    }
04490
04491    // マテリアルを初期化する
04492    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const std::vector<Material>& newmtls11, const Material& newmtl)
04493    {
04494        material.clear();
04495        material.reserve(count);
04496        material.insert(material.end(), materials.begin(), materials.end());
04497        material.push_back(newmtl);
04498    }
04499
04500    // マテリアルを初期化する
04501    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const std::vector<Material>& newmtls11, const std::vector<Material>& newmtls12)
04502    {
04503        material.clear();
04504        material.reserve(count);
04505        material.insert(material.end(), materials.begin(), materials.end());
04506        material.push_back(newmtls12[0]);
04507    }
04508
04509    // マテリアルを初期化する
04510    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const std::vector<Material>& newmtls11, const std::vector<Material>& newmtls12, const Material& newmtl)
04511    {
04512        material.clear();
04513        material.reserve(count);
04514        material.insert(material.end(), materials.begin(), materials.end());
04515        material.push_back(newmtl);
04516    }
04517
04518    // マテリアルを初期化する
04519    void initializeMaterial(GLuint count, const std::vector<Material>& materials, const std::vector<Material>& defaultMaterials, const std::vector<Material>& newmtls, const std::vector<Material>& newmtls2, const std::vector<Material>& newmtls3, const std::vector<Material>& newmtls4, const std::vector<Material>& newmtls5, const std::vector<Material>& newmtls6, const std::vector<Material>& newmtls7, const std::vector<Material>& newmtls8, const std::vector<Material>& newmtls9, const std::vector<Material>& newmtls10, const std::vector<Material>& newmtls11, const std::vector<Material>& newmtls12, const std::vector<Material>& newmtls13)
04516
```

```

04204     // 新規作成する材質にデフォルトの材質を設定しておく
04205     material.emplace_back(defaultMaterial);
04206 }
04207
04208 #if defined(DEBUG)
04209     std::cerr << "newmtl: " << mtlname << std::endl;
04210#endif
04211 }
04212 else if (mtlop == "Ka")
04213 {
04214     // 環境光の反射係数を登録する
04215     mtlstr
04216         >> material.back().ambient[0]
04217         >> material.back().ambient[1]
04218         >> material.back().ambient[2];
04219 }
04220 else if (mtlop == "Kd")
04221 {
04222     // 拡散反射係数を登録する
04223     mtlstr
04224         >> material.back().diffuse[0]
04225         >> material.back().diffuse[1]
04226         >> material.back().diffuse[2];
04227 }
04228 else if (mtlop == "Ks")
04229 {
04230     // 鏡面反射係数を登録する
04231     mtlstr
04232         >> material.back().specular[0]
04233         >> material.back().specular[1]
04234         >> material.back().specular[2];
04235 }
04236 else if (mtlop == "Ns")
04237 {
04238     // 輝き係数を登録する
04239     GLfloat shininess;
04240     mtlstr >> shininess;
04241     material.back().shininess = shininess * 0.1f;
04242 }
04243 else if (mtlop == "d")
04244 {
04245     // 不透明度を登録する
04246     mtlstr >> material.back().ambient[3];
04247 }
04248 }
04249
04250 // MTL ファイルの読み込みに失敗したら戻る
04251 if (mtlfile.bad())
04252 {
04253 #if defined(DEBUG)
04254     std::cerr << "Warning: Can't read MTL file: " << mtlname << std::endl;
04255#endif
04256     return false;
04257 }
04258
04259 // MTL ファイルの読み込みに成功した
04260 return true;
04261 }
04262
04263 //
04264 // Alias OBJ 形式のファイルを解析する
04265 //
04266 // name Alias OBJ 形式のファイルのファイル名
04267 // group 同じ材質を割り当てるポリゴングループ
04268 // mtl 読み込んだ材質名をキーにした map
04269 // pos 頂点の位置
04270 // norm 頂点の法線
04271 // tex 頂点のテクスチャ座標
04272 // face 三角形のデータ
04273 //
04274 static bool ggParseObj(
04275     const std::string& name,
04276     std::vector<fgrp>& group,
04277     std::vector<GgSimpleShader::Material>& material,
04278     std::vector<vec3>& pos,
04279     std::vector<vec3>& norm,
04280     std::vector<vec2>& tex,
04281     std::vector<fidx>& face,
04282     bool normalize
04283 )
04284 {
04285     // ファイルパスからディレクトリ名を取り出す
04286     const std::string path{ name };
04287     const size_t base{ path.find_last_of("\\\\") };
04288     const std::string dirname{ (base == std::string::npos) ? "" : path.substr(0, base + 1) };
04289
04290     // OBJ ファイルを読み込む

```

```
04291     std::ifstream file{ Utf8ToTChar(path) };
04292
04293     // 読み込みに失敗したら戻る
04294     if (file.fail())
04295     {
04296         #if defined(DEBUG)
04297             std::cerr << "Error: Can't open OBJ file: " << path << std::endl;
04298     #endif
04299     }
04300
04301     // ポリゴングループの最初の三角形番号
04302     GLsizei startgroup(static_cast<GLsizei>(group.size()));
04303
04304     // スムーズシェーディングのスイッチ
04305     bool smooth{ false };
04306
04307     // 材質のテーブル
04308     std::map<std::string, GLuint> mtl;
04309
04310     // 現在の材質名（ループの外で宣言する）
04311     std::string mtlname;
04312
04313     // 座標値の最小値・最大値
04314     vec3 bmin{ FLT_MAX }, bmax{ -FLT_MAX };
04315
04316     // 一行読み込み用のバッファ
04317     std::string line;
04318
04319     // データの読み込み
04320     while (std::getline(file, line))
04321     {
04322         // 空行は読み飛ばす
04323         if (line == "") continue;
04324
04325         // 最後の文字が '\r' なら
04326         if (*(line.end() - 1) == '\r')
04327         {
04328             // 最後の文字を削除する
04329             line.erase(line.end() - 1, line.end());
04330
04331             // 空行になったら読み飛ばす
04332             if (line == "") continue;
04333         }
04334
04335         // 一行を文字列ストリームに入れる
04336         std::istringstream str(line);
04337
04338         // 最初のトークンを命令 (op) とみなす
04339         std::string op;
04340         str >> op;
04341
04342         if (op[0] == '#') continue;
04343
04344         if (op == "v")
04345         {
04346             // 頂点位置
04347             vec3 v;
04348
04349             // 頂点位置はスペースで区切られている
04350             str >> v[0] >> v[1] >> v[2];
04351
04352             // 頂点位置を記録する
04353             pos.emplace_back(v);
04354
04355             // 頂点位置の最小値と最大値を求める (AABB)
04356             for (int i = 0; i < 3; ++i)
04357             {
04358                 bmin[i] = std::min(bmin[i], v[i]);
04359                 bmax[i] = std::max(bmax[i], v[i]);
04360             }
04361         }
04362
04363         else if (op == "vt")
04364         {
04365             // テクスチャ座標
04366             vec2 t;
04367
04368             // 頂点位置はスペースで区切られている
04369             str >> t[0] >> t[1];
04370
04371             // テクスチャ座標を記録する
04372             tex.emplace_back(t);
04373         }
04374         else if (op == "vn")
04375         {
04376             // 頂点法線
04377             vec3 n;
```

```

04378     // 頂点法線はスペースで区切られている
04379     str >> n[0] >> n[1] >> n[2];
04380
04381     // 頂点法線を記録する
04382     norm.emplace_back(n);
04383 }
04384 else if (op == "f")
04385 {
04386     // 三角形データ
04387     fidx f;
04388
04389     // スムースシェーディング
04390     f.smooth = smooth;
04391
04392     // 三頂点のそれぞれについて
04393     for (int i = 0; i < 3; ++i)
04394     {
04395         // 1項目取り出す
04396         std::string s;
04397         str >> s;
04398
04399         // 文字の位置
04400         auto c{ s.begin() };
04401
04402         // 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
04403         f.p[i] = f.t[i] = f.n[i] = 0;
04404
04405         // 項目の最初の要素は頂点座標番号
04406         while (c != s.end() && isdigit(*c)) f.p[i] = f.p[i] * 10 + *c++ - '0';
04407         if (c == s.end() || *c++ != '/') continue;
04408
04409         // 二つ目の項目はテクスチャ座標
04410         while (c != s.end() && isdigit(*c)) f.t[i] = f.t[i] * 10 + *c++ - '0';
04411         if (c == s.end() || *c++ != '/') continue;
04412
04413         // 三つ目の項目は法線番号
04414         while (c != s.end() && isdigit(*c)) f.n[i] = f.n[i] * 10 + *c++ - '0';
04415     }
04416
04417     // 三角形データを登録する
04418     face.emplace_back(f);
04419 }
04420 else if (op == "s")
04421 {
04422     // '1' だったらスムースシェーディング有効
04423     std::string s;
04424     str >> s;
04425     smooth = s == "1";
04426 }
04427 else if (op == "usemtl")
04428 {
04429     // 次のポリゴングループの最初の三角形番号
04430     const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04431
04432     // ポリゴングループに三角形が存在すれば
04433     if (nextgroup > startgroup)
04434     {
04435         // ポリゴングループの三角形数と材質番号を記録する
04436         group.emplace_back(nextgroup, mtl[mtlname]);
04437
04438         // 次のポリゴン グループの開始番号を保存しておく
04439         startgroup = nextgroup;
04440     }
04441
04442     // 次に usemtl が来るまで材質名を保持する
04443     str >> mtlname;
04444
04445     // 材質の存在チェック
04446     if (mtl.find(mtlname) == mtl.end())
04447     {
04448 #if defined(DEBUG)
04449         std::cerr << "Warning: Undefined material: " << mtlname << std::endl;
04450 #endif
04451     }
04452     // デフォルトの材質を割り当てておく
04453     mtlname = defaultMaterialName;
04454
04455 }
04456 #if defined(DEBUG)
04457     else std::cerr << "usemtl: " << mtlname << std::endl;
04458 #endif
04459 }
04460 else if (op == "mtllib")
04461 {
04462     // MTL ファイルのパス名を作る
04463     str >> std::ws;
04464     std::string mtlpath;

```

```
04465     std::getline(str, mt1path);
04466
04467     // MTL ファイルを読み込む
04468     ggLoadMtl(dirname + mt1path, mtl, material);
04469 }
04470
04471 // OBJ ファイルの読み込みに失敗したら戻る
04472 if (file.bad())
04473 {
04475 #if defined(DEBUG)
04476     std::cerr << "Error: Can't read OBJ file: " << path << std::endl;
04477 #endif
04478     return false;
04479 }
04480
04481 // 最後のポリゴングループの次の三角形番号
04482 const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04483 if (nextgroup > startgroup)
04484 {
04485     // 最後のポリゴングループの三角形数と材質を記録する
04486     group.emplace_back(nextgroup, static_cast<GLuint>(mtl[mtlname]));
04487 }
04488
04489 // 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
04490 for (auto& f : face)
04491 {
04492     if (!f.smooth)
04493     {
04494         // 三頂点のそれぞれについて
04495         for (int i = 0; i < 3; ++i)
04496         {
04497             // 新しい頂点座標を生成する (std::array の要素は emplace_back できない)
04498             pos.emplace_back(pos[f.p[i] - 1]);
04499             f.p[i] = static_cast<GLuint>(pos.size());
04500
04501             if (f.t[i] > 0)
04502             {
04503                 // 新しいテクスチャ座標を生成する
04504                 tex.emplace_back(tex[f.t[i] - 1]);
04505                 f.t[i] = static_cast<GLuint>(tex.size());
04506             }
04507
04508             if (f.n[i] > 0)
04509             {
04510                 // 新しい法線を生成する
04511                 norm.emplace_back(norm[f.n[i] - 1]);
04512                 f.n[i] = static_cast<GLuint>(norm.size());
04513             }
04514         }
04515     }
04516 }
04517
04518 // 法線データがなければ算出しておく
04519 if (norm.empty())
04520 {
04521     // 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
04522     norm.resize(pos.size(), { 0.0f, 0.0f, 0.0f });
04523
04524     // 面の法線の算出と頂点法線の算出
04525     for (auto& f : face)
04526     {
04527         // 頂点座標番号
04528         const auto v0{ f.p[0] - 1 };
04529         const auto v1{ f.p[1] - 1 };
04530         const auto v2{ f.p[2] - 1 };
04531
04532         // v1 - v0, v2 - v0 を求める
04533         const GLfloat d1[] { pos[v1][0] - pos[v0][0], pos[v1][1] - pos[v0][1], pos[v1][2] - pos[v0][2] };
04534         const GLfloat d2[] { pos[v2][0] - pos[v0][0], pos[v2][1] - pos[v0][1], pos[v2][2] - pos[v0][2] };
04535
04536         // 外積により面法線を求める
04537         vec3 n;
04538         ggCross(n.data(), d1, d2);
04539
04540         if (f.smooth)
04541         {
04542             // スムースシェーディングを行うときは
04543             for (int i = 0; i < 3; ++i)
04544             {
04545                 // 面法線を頂点法線に積算する
04546                 norm[v0][i] += n[i];
04547                 norm[v1][i] += n[i];
04548                 norm[v2][i] += n[i];
04549             }
04550         }
04551     }
04552 }
```

```

04550      // 面の各頂点の法線番号は頂点番号と同じにする
04551      f.n[i] = f.p[i];
04552  }
04553  }
04554  else
04555  {
04556      // 面法線を最初の頂点に保存する
04557      norm[v0] = n;
04558      f.n[0] = f.p[0];
04559
04560      // 2 頂点追加
04561      for (int i = 1; i < 3; ++i)
04562      {
04563          norm.emplace_back(n);
04564          f.n[i] = static_cast<GLuint>(norm.size());
04565      }
04566  }
04567  }
04568
04569  // 頂点の法線ベクトルを正規化する
04570  for (auto& n : norm) ggNormalize3(n.data());
04571
04572
04573  // 図形の正規化
04574  if (normalize)
04575  {
04576      // 図形の大きさ
04577      const auto sx{ bmax[0] - bmin[0] };
04578      const auto sy{ bmax[1] - bmin[1] };
04579      const auto sz{ bmax[2] - bmin[2] };
04580
04581      // 図形のスケール
04582      GLfloat s{ sx };
04583      if (sy > s) s = sy;
04584      if (sz > s) s = sz;
04585      const auto scale{ s != 0.0f ? 2.0f / s : 1.0f };
04586
04587      // 図形の中心位置
04588      const auto cx{ (bmax[0] + bmin[0]) * 0.5f };
04589      const auto cy{ (bmax[1] + bmin[1]) * 0.5f };
04590      const auto cz{ (bmax[2] + bmin[2]) * 0.5f };
04591
04592      // 図形の大きさと位置を正規化する
04593      for (auto& p : pos)
04594  {
04595          p[0] = (p[0] - cx) * scale;
04596          p[1] = (p[1] - cy) * scale;
04597          p[2] = (p[2] - cz) * scale;
04598      }
04599  }
04600
04601 #if defined(DEBUG)
04602     std::cerr
04603     << "[" << name << "]\nParsed Group: " << group.size() << ", Material: " << mtl.size()
04604     << ", Pos: " << pos.size() << ", Norm: " << norm.size() << ", Tex: " << tex.size()
04605     << ", Face: " << face.size() << "\n";
04606 #endif
04607
04608     // OBJ ファイルの読み込み成功
04609     return true;
04610 }
04611 }
04612
04613
04614 //
04615 // 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)
04616 //
04617 // name 読み込むOBJ ファイル名
04618 // 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
04619 // material 読み込んだデータのポリゴン グループごとの材質
04620 // vert 読み込んだデータの頂点属性
04621 // normalize true ならサイズを正規化する
04622 // 戻り値 読み込みに成功したら true
04623 //
04624 bool gg::ggLoadSimpleObj(const std::string& name,
04625     std::vector<std::array<GLuint, 3>>& group,
04626     std::vector<GgSimpleShader::Material>& material,
04627     std::vector<GgVertex>& vert,
04628     bool normalize)
04629 {
04630     // 読み込み用の一時記憶領域
04631     std::vector<fgrp> tgroup;
04632     std::vector<vec3> tpos;
04633     std::vector<vec3> tnorm;
04634     std::vector<vec2> ttex;
04635     std::vector<fidx> tface;
04636
04637     // OBJ ファイルを解析する

```

```
04638 if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, normalize)) return false;
04639 // 頂点属性データのメモリを確保する
04640 vert.reserve(vert.size() + tface.size() * 3);
04641 // ポリゴングループデータのメモリを確保する
04642 group.reserve(group.size() + tgroup.size());
04643 material.reserve(material.size() + tgroup.size());
04644 // ポリゴングループの最初の三角形番号
04645 GLuint startgroup{ 0 };
04646 // ポリゴングループデータの作成
04647 for (auto& g : tgroup)
04648 {
04649     // 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
04650     std::array<GLuint, 3> v;
04651     // このポリゴングループの最初の頂点番号を保存する
04652     v[0] = static_cast<GLuint>(vert.size());
04653     // 三角形ごとの頂点データの作成
04654     for (GLuint j = startgroup; j < g.nextgroup; ++j)
04655     {
04656         // 処理対象の三角形
04657         auto& f = tface[j];
04658         // 三頂点のそれぞれについて
04659         for (int i = 0; i < 3; ++i)
04660         {
04661             // テクスチャ座標
04662             vec2 tex{ 0.0f, 0.0f };
04663             if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04664             // 頂点法線
04665             vec3 norm{ 0.0f, 0.0f, 0.0f };
04666             if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04667             // 頂点属性の追加
04668             vert.emplace_back(tpos[f.p[i] - 1].data(), norm.data());
04669         }
04670     }
04671     // このポリゴングループの頂点数を保存する
04672     v[1] = static_cast<GLuint>(vert.size()) - v[0];
04673     v[2] = g.mtlno;
04674     // 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
04675     group.emplace_back(v);
04676     // 次のポリゴングループの最初の三角形番号を求める
04677     startgroup = g.nextgroup;
04678 }
04679 // #if defined(DEBUG)
04680 #if defined(DEBUG)
04681     std::cerr
04682         << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04683         << ", Vertex: " << vert.size() << "\n";
04684 #endif
04685 // OBJ ファイルの読み込み成功
04686 return true;
04687 }
04688 // 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)
04689 // name 読み込むOBJ ファイル名
04690 // 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
04691 // material 読み込んだデータのポリゴングループごとの材質
04692 // vert 読み込んだデータの頂点属性
04693 // face 読み込んだデータの三角形の頂点インデックス
04694 // normalize true ならサイズを正規化する
04695 // 戻り値 読み込みに成功したら true
04696 //
04697 // bool gg::ggLoadSimpleObj(const std::string& name,
04698 // std::vector<std::array<GLuint, 3>>& group,
04699 // std::vector<GgSimpleShader::Material>& material,
04700 // std::vector<GgVertex>& vert,
04701 // std::vector<GLuint>& face,
04702 // bool normalize)
04703 // {
04704 //     // 読み込み用の一時記憶領域
04705 //     std::vector<fgrp> tgroup;
04706 //     std::vector<vec3> tpos;
04707 //     std::vector<vec3> tnorm;
04708 //     std::vector<vec2> ttex;
```

```

04725     std::vector<fidx> tface;
04726
04727     // OBJ ファイルを解析する
04728     if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04729
04730     // 頂点属性データの最初の頂点番号
04731     const auto vertbase{ static_cast<GLuint>(vert.size()) };
04732
04733     // 頂点属性データのメモリを確保する
04734     vert.resize(vertbase + tpos.size());
04735
04736     // 三角形データのメモリを確保する
04737     face.reserve(face.size() + tface.size());
04738
04739     // ポリゴングループデータのメモリを確保する
04740     group.reserve(group.size() + tgroup.size());
04741     material.reserve(material.size() + tgroup.size());
04742
04743     // ポリゴングループの最初の三角形番号
04744     GLuint startgroup{ 0 };
04745
04746     // ポリゴングループデータの作成
04747     for (auto& g : tgroup)
04748     {
04749         // このポリゴングループの最初の頂点番号
04750         const auto first{ static_cast<GLuint>(face.size()) };
04751
04752         // 三角形ごとの頂点データの作成
04753         for (GLuint j = startgroup; j < g.nextgroup; ++j)
04754         {
04755             // 処理対象の三角形
04756             auto& f{ tface[j] };
04757
04758             // 三頂点のそれぞれについて
04759             for (int i = 0; i < 3; ++i)
04760             {
04761                 // 追加する三角形データの頂点番号
04762                 const auto q{ f.p[i] - 1 + vertbase };
04763
04764                 // 三角形データの追加
04765                 face.emplace_back(q);
04766
04767                 // テクスチャ座標番号
04768                 vec2 tex{ 0.0f, 0.0f };
04769                 if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04770
04771                 // 頂点法線番号
04772                 vec3 norm{ 0.0f, 0.0f, 0.0f };
04773                 if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04774
04775                 // 頂点の格納
04776                 vert[q] = GgVertex(tpos[f.p[i] - 1].data(), norm.data());
04777             }
04778         }
04779
04780         // 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
04781         group.emplace_back(std::array<GLuint, 3>{ first, static_cast<GLuint>(face.size()) - first, g.mtlno });
04782     }
04783     // 次のポリゴングループの最初の三角形番号を求める
04784     startgroup = g.nextgroup;
04785 }
04786
04787 #if defined(DEBUG)
04788     std::cerr
04789         << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04790         << ", Vertex: " << vert.size() << ", Face: " << face.size() << "\n";
04791 #endif
04792
04793     // OBJ ファイルの読み込み成功
04794     return true;
04795 }
04796
04797 //
04798 // シェーダオブジェクトのコンパイル結果を表示する
04799 //
04800 static GLboolean printShaderInfoLog(GLuint shader, const std::string& str)
04801 {
04802     // コンパイル結果を取得する
04803     GLint status;
04804     glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
04805 #if defined(DEBUG)
04806     if (status == GL_FALSE) std::cerr << "Compile Error in " << str << std::endl;
04807 #endif
04808
04809     // シェーダのコンパイル時のログの長さを取得する
04810     GLsizei bufSize;

```

```
04811     glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &bufSize);
04812
04813     if (bufSize > 1)
04814     {
04815         // シェーダのコンパイル時のログの内容を取得する
04816         std::vector<GLchar> infoLog(bufSize);
04817         GLsizei length;
04818         glGetShaderInfoLog(shader, bufSize, &length, infoLog.data());
04819 #if defined(DEBUG)
04820         std::cerr << infoLog.data();
04821 #endif
04822     }
04823
04824     // コンパイル結果を返す
04825     return static_cast<GLboolean>(status);
04826 }
04827
04828 /**
04829 // プログラムオブジェクトのリンク結果を表示する
04830 /**
04831 static GLboolean printProgramInfoLog(GLuint program)
04832 {
04833     // リンク結果を取得する
04834     GLint status;
04835     glGetProgramiv(program, GL_LINK_STATUS, &status);
04836 #if defined(DEBUG)
04837     if (status == GL_FALSE) std::cerr << "Link Error." << std::endl;
04838 #endif
04839
04840     // シェーダのリンク時のログの長さを取得する
04841     GLsizei bufSize;
04842     glGetProgramiv(program, GL_INFO_LOG_LENGTH, &bufSize);
04843
04844     // シェーダのリンク時のログの内容を取得する
04845     if (bufSize > 1)
04846     {
04847         std::vector<GLchar> infoLog(bufSize);
04848         GLsizei length;
04849         glGetProgramInfoLog(program, bufSize, &length, infoLog.data());
04850 #if defined(DEBUG)
04851         std::cerr << infoLog.data() << std::endl;
04852 #endif
04853     }
04854
04855     // リンク結果を返す
04856     return static_cast<GLboolean>(status);
04857 }
04858
04859 /**
04860 // 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
04861 /**
04862 //    vsrc パーテックスシェーダのソースプログラムの文字列
04863 //    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)
04864 //    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)
04865 //    nvarying フィードバックする varying 変数の数 (0 なら不使用)
04866 //    varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04867 //    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
04868 //    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
04869 //    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
04870 // 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)
04871 /**
04872 GLuint gg::ggCreateShader(
04873     const std::string& vsrc,
04874     const std::string& fsrc,
04875     const std::string& gsrc,
04876     GLint nvarying,
04877     const char* const* varyings,
04878     const std::string& vtext,
04879     const std::string& ftext,
04880     const std::string& gtext)
04881 {
04882     // シェーダプログラムの作成
04883     const auto program{ glCreateProgram() };
04884
04885     if (program > 0)
04886     {
04887         bool status = true;
04888
04889         if (!vsrc.empty())
04890         {
04891             // 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
04892             const auto vertShader{ glCreateShader(GL_VERTEX_SHADER) };
04893             const auto* vsrcc{ vsrc.c_str() };
04894             glShaderSource(vertShader, 1, &vsrcc, nullptr);
04895             glCompileShader(vertShader);
04896
04897             // 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
04898         }
04899
04900         if (!fsrc.empty())
04901         {
04902             const auto fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
04903             const auto* fsrcs{ fsrcc };
04904             glShaderSource(fragShader, 1, &fsrcs, nullptr);
04905             glCompileShader(fragShader);
04906
04907             glLinkProgram(program);
04908
04909             GLint linkStatus;
04910             glGetProgramiv(program, GL_LINK_STATUS, &linkStatus);
04911             if (linkStatus != GL_TRUE)
04912             {
04913                 std::cout << "Link Error." << std::endl;
04914                 status = false;
04915             }
04916         }
04917
04918         if (!gsrc.empty())
04919         {
04920             const auto geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
04921             const auto* gsrcc{ gsrc.c_str() };
04922             glShaderSource(geomShader, 1, &gsrcc, nullptr);
04923             glCompileShader(geomShader);
04924
04925             glLinkProgram(program);
04926
04927             GLint linkStatus;
04928             glGetProgramiv(program, GL_LINK_STATUS, &linkStatus);
04929             if (linkStatus != GL_TRUE)
04930             {
04931                 std::cout << "Link Error." << std::endl;
04932                 status = false;
04933             }
04934         }
04935
04936         if (!status)
04937         {
04938             glDeleteProgram(program);
04939             return 0;
04940         }
04941     }
04942
04943     return program;
04944 }
```

```

04898     if (printShaderInfoLog(vertShader, vtext))
04899         glAttachShader(program, vertShader);
04900     else
04901         status = false;
04902     glDeleteShader(vertShader);
04903 }
04904
04905 if (!fsrc.empty())
04906 {
04907     // 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
04908     const auto fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
04909     const auto* fsrcc{ fsrcc.c_str() };
04910     glShaderSource(fragShader, 1, &fsrcc, nullptr);
04911     glCompileShader(fragShader);
04912
04913     // 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
04914     if (printShaderInfoLog(fragShader, ftext))
04915         glAttachShader(program, fragShader);
04916     else
04917         status = false;
04918     glDeleteShader(fragShader);
04919 }
04920
04921 if (!gsrc.empty())
04922 {
04923 #if defined(GL_GLES_PROTOTYPES)
04924 # if defined(DEBUG)
04925     std::cerr << gtext << ": The geometry shader is not supported." << std::endl;
04926     status = false;
04927 # endif
04928 #else
04929     // 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
04930     const auto geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
04931     const auto* gsrcc{ gsrc.c_str() };
04932     glShaderSource(geomShader, 1, &gsrcc, nullptr);
04933     glCompileShader(geomShader);
04934
04935     // 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
04936     if (printShaderInfoLog(geomShader, gtext))
04937         glAttachShader(program, geomShader);
04938     else
04939         status = false;
04940     glDeleteShader(geomShader);
04941 #endif
04942 }
04943
04944 // 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
04945 if (status)
04946 {
04947     // feedback に使う varying 変数を指定する
04948     if (nvarying > 0)
04949         glTransformFeedbackVaryings(program, nvarying, varyings, GL_SEPARATE_ATTRIBS);
04950
04951     // シェーダプログラムをリンクする
04952     glLinkProgram(program);
04953
04954     // リンクに成功したらプログラムオブジェクト名を返す
04955     if (printProgramInfoLog(program) != GL_FALSE) return program;
04956 }
04957 }
04958
04959 // プログラムオブジェクトが作成できなかった
04960 glDeleteProgram(program);
04961 return 0;
04962 }
04963
04964 //
04965 // シェーダのソースファイルを読み込んだ vector を返す
04966 //
04967 //   name ソースファイル名
04968 //   src 読み込んだソースファイルの文字列
04969 //   戻り値 読み込みの成功すれば true, 失敗したら false
04970 //
04971 static bool readShaderSource(const std::string& name, std::string& src)
04972 {
04973     // ファイル名が nullptr ならそのまま戻る
04974     if (name.empty()) return true;
04975
04976     // ソースファイルを開く
04977     std::ifstream file{ Utf8ToTChar(name), std::ios::binary };
04978     if (file.fail())
04979     {
04980         // ファイルが開けなければエラーで戻る
04981 #if defined(DEBUG)
04982         std::cerr << "Error: Can't open source file: " << name << std::endl;
04983 #endif
04984         return false;

```

```
04985    }
04986
04987    // ファイル全体を文字列として読み込む
04988    std::istreambuf_iterator<char> it{ file };
04989    std::istreambuf_iterator<char> last;
04990    src = std::string(it, last);
04991
04992    // ファイルがうまく読み込めなければ戻る
04993    if (file.bad())
04994    {
04995        #if defined(DEBUG)
04996            std::cerr << "Error: Could not read source file: " << name << std::endl;
04997        #endif
04998        return false;
04999    }
05000
05001    // ファイルを閉じて戻る
05002    return true;
05003 }
05004
05005 //
05006 // 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
05007 //
05008 //    vert パーテックスシェーダのソースファイル名
05009 //    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)
05010 //    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)
05011 //    nvarying フィードバックする varying 変数の数 (0 なら不使用)
05012 //    varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
05013 //    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)
05014 //
05015 GLuint gg::ggLoadShader(
05016     const std::string& vert,
05017     const std::string& frag,
05018     const std::string& geom,
05019     GLint nvarying,
05020     const char* const* varyings
05021 )
05022 {
05023     // 読み込んだシェーダのソースプログラム
05024     std::string vsrc, fsrc, gsrc;
05025
05026     // 読み込んだ結果
05027     bool status;
05028
05029     // シェーダのソースファイルを読み込む
05030     status = readShaderSource(vert, vsrc);
05031     status = readShaderSource(frag, fsrc) && status;
05032     status = readShaderSource(geom, gsrc) && status;
05033
05034     // 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
05035     if (!status) return 0;
05036
05037     // プログラムオブジェクトを作成する
05038     return ggCreateShader(vsrc, fsrc, gsrc, nvarying, varyings, vert, frag, geom);
05039 }
05040
05041 #if !defined(__APPLE__) && !defined(GL_GLES_PROTOTYPES)
05042 //
05043 // 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
05044 //
05045 //    csrc コンピュートシェーダのソースプログラムの文字列
05046 //    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)
05047 //
05048 GLuint gg::ggCreateComputeShader(const std::string& csrc, const std::string& ctext)
05049 {
05050     // シェーダプログラムの作成
05051     const auto program{ glCreateProgram() };
05052
05053     if (program > 0)
05054     {
05055         // ソースプログラムの文字列が空だったら 0 を返す
05056         if (csrc.empty()) return 0;
05057
05058         // 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
05059         const auto compShader{ glCreateShader(GL_COMPUTE_SHADER) };
05060         const auto csrcp{ csrc.c_str() };
05061         glShaderSource(compShader, 1, &csrcp, nullptr);
05062         glCompileShader(compShader);
05063
05064         // 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
05065         if (printShaderInfoLog(compShader, ctext)) glAttachShader(program, compShader);
05066         glDeleteShader(compShader);
05067
05068         // シェーダプログラムをリンクする
05069         glLinkProgram(program);
05070
05071         // プログラムオブジェクトが作成できなければ 0 を返す
05072 }
```

```

05072     if (printProgramInfoLog(program) == GL_FALSE)
05073     {
05074         glDeleteProgram(program);
05075         return 0;
05076     }
05077 }
05078
05079 // プログラムオブジェクトを返す
05080 return program;
05081 }
05082
05083 //
05084 // 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
05085 //
05086 //    comp コンピュートシェーダのソースファイル名
05087 //    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)
05088 //
05089 GLuint gg::ggLoadComputeShader(const std::string& comp)
05090 {
05091     // シェーダのソースファイルを読み込む
05092     std::string csrc;
05093     if (readShaderSource(comp, csrc))
05094     {
05095         // プログラムオブジェクトを作成する
05096         return ggCreateComputeShader(csrc.data(), comp);
05097     }
05098
05099     // プログラムオブジェクト作成失敗
05100     return 0;
05101 }
05102 #endif
05103
05104 //
05105 // 点: データ作成
05106 //
05107 void gg::GgPoints::load(const GgVector* pos, GLsizei count, GLenum usage)
05108 {
05109     // 頂点バッファオブジェクトを作成する
05110     position = std::make_shared<GgBuffer<GgVector>>(GL_ARRAY_BUFFER, pos,
05111     static_cast<GLsizei>(sizeof(GgVector)), count, usage);
05112
05113     // 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
05114     glVertexAttribPointer(0, static_cast<GLint>(pos->size()), GL_FLOAT, GL_FALSE, 0, 0);
05115     glEnableVertexAttribArray(0);
05116
05117 //
05118 // 点: 描画
05119 //
05120 void gg::GgPoints::draw(GLint first, GLsizei count) const
05121 {
05122     // 頂点配列オブジェクトを指定する
05123     GgShape::draw(first, count);
05124
05125     // 図形を描画する
05126     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05127 }
05128
05129 //
05130 // 三角形: データ作成
05131 //
05132 void gg::GgTriangles::load(const GgVertex* vert, GLsizei count, GLenum usage)
05133 {
05134     // 頂点バッファオブジェクトを作成する
05135     vertex = std::make_shared<GgBuffer<GgVertex>>(GL_ARRAY_BUFFER, vert,
05136     static_cast<GLsizei>(sizeof(GgVertex)), count, usage);
05137
05138     // 頂点の位置は index == 0 の in 変数から入力する
05139     glVertexAttribPointer(0, static_cast<GLint>(vert->position.size()), GL_FLOAT, GL_FALSE,
05140     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, position));
05141     glEnableVertexAttribArray(0);
05142
05143     // 頂点の法線は index == 1 の in 変数から入力する
05144     glVertexAttribPointer(1, static_cast<GLint>(vert->normal.size()), GL_FLOAT, GL_FALSE,
05145     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, normal));
05146     glEnableVertexAttribArray(1);
05147
05148 //
05149 // 三角形: 描画
05150 //
05151 void gg::GgTriangles::draw(GLint first, GLsizei count) const
05152 {
05153     // 頂点配列オブジェクトを指定する
05154     GgShape::draw(first, count);
05155
05156     // 図形を描画する

```

```
05157     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05158 }
05159
05160 // オブジェクト：描画
05161 // オブジェクトを指定する
05162 // 頂点配列オブジェクトを指定する
05163 void gg::GgElements::draw(GLint first, GLsizei count) const
05164 {
05165     GgShape::draw(first, count);
05166
05167     // 図形を描画する
05168     glDrawElements(getMode(), count > 0 ? count : getIndexCount() - first,
05169                     GL_UNSIGNED_INT, static_cast<GLuint*>(0) + first);
05170
05171 }
05172
05173 // 点群を立方体状に生成する
05174 // メモリを確保する
05175 std::shared_ptr<gg::GgPoints> gg::ggPointsCube(GLsizei count, GLfloat length, GLfloat cx, GLfloat cy,
05176                                                 GLfloat cz)
05177 {
05178     // メモリを確保する
05179     std::vector<GgVector> pos(count);
05180
05181     // 点を生成する
05182     for (GLsizei v = 0; v < count; ++v)
05183     {
05184         const GgVector p
05185         {
05186             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cx,
05187             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cy,
05188             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cz,
05189             1.0f
05190         };
05191
05192         pos.emplace_back(p);
05193     }
05194
05195     // 点データの GgPoints オブジェクトを作成して返す
05196     return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05197 }
05198
05199 //
05200 // 点群を球状に生成する
05201 //
05202 std::shared_ptr<gg::GgPoints> gg::ggPointsSphere(GLsizei count, GLfloat radius,
05203                                                 GLfloat cx, GLfloat cy, GLfloat cz)
05204 {
05205     // メモリを確保する
05206     std::vector<GgVector> pos(count);
05207
05208     // 点を生成する
05209     for (GLsizei v = 0; v < count; ++v)
05210     {
05211         const auto r{ radius * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) };
05212         const auto t{ 6.28318530718f * static_cast<GLfloat>(rand()) / (static_cast<GLfloat>(RAND_MAX) +
05213             1.0f) };
05214         const auto cp{ 2.0f * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 1.0f };
05215         const auto sp{ sqrtf(1.0f - cp * cp) };
05216         const auto ct{ cosf(t) };
05217         const auto st{ sinf(t) };
05218         const GgVector p{ r * sp * ct + cx, r * sp * st + cy, r * cp + cz, 1.0f };
05219
05220         pos.emplace_back(p);
05221     }
05222
05223     // 点データの GgPoints オブジェクトを作成して返す
05224     return std::make_shared<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05225 }
05226
05227 // 矩形状に 2 枚の三角形を生成する
05228 //
05229 std::shared_ptr<gg::GgTriangles> gg::ggRectangle(GLfloat width, GLfloat height)
05230 {
05231     // 頂点属性
05232     std::array<gg::GgVertex, 4> vert;
05233
05234     // 頂点位置と法線を求める
05235     for (int v = 0; v < 4; ++v)
05236     {
05237         const auto x{ ((v & 1) * 2 - 1) * width };
05238         const auto y{ ((v & 2) - 1) * height };
05239
05240         vert[v] = gg::GgVertex{ x, y, 0.0f, 0.0f, 0.0f, 1.0f };
05241     }
}
```

```

05242 // 矩形の GgTriangles オブジェクトを作成する
05243     return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05244         GL_TRIANGLE_STRIP);
05245 }
05246
05247 //
05248 // 構円状に三角形を生成する
05249 //
05250 std::shared_ptr<gg::GgTriangles> gg::ggEllipse(GLfloat width, GLfloat height, GLuint slices)
05251 {
05252     // 構円のスケール
05253     constexpr GLfloat scale{ 0.5f };
05254
05255     // 作業用のメモリ
05256     std::vector<gg::GgVertex> vert;
05257
05258     // 頂点位置と法線を求める
05259     for (GLuint v = 0; v < slices; ++v)
05260     {
05261         const auto t{ 6.28318530717f * static_cast<GLfloat>(v) / static_cast<GLfloat>(slices) };
05262         const auto x{ cosf(t) * width * scale };
05263         const auto y{ sinf(t) * height * scale };
05264
05265         vert.emplace_back(x, y, 0.0f, 0.0f, 0.0f, 1.0f);
05266     }
05267
05268     // GgTriangles オブジェクトを作成する
05269     return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05270         GL_TRIANGLE_FAN);
05271 }
05272 //
05273 // Wavefront OBJ ファイルを読み込む (Arrays 形式)
05274 //
05275 std::shared_ptr<gg::GgTriangles> gg::ggArraysObj(const std::string& name, bool normalize)
05276 {
05277     std::vector<std::array<GLuint, 3>> group;
05278     std::vector<gg::GgSimpleShader::Material> material;
05279     std::vector<gg::GgVertex> vert;
05280
05281     // ファイルを読み込む
05282     if (!ggLoadSimpleObj(name, group, material, vert, normalize)) return nullptr;
05283
05284     // GgTriangles オブジェクトを作成する
05285     return std::make_shared<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
05286         GL_TRIANGLES);
05287 }
05288 //
05289 // Wavefront OBJ ファイル を読み込む (Elements 形式)
05290 //
05291 std::shared_ptr<gg::GgElements> gg::ggElementsObj(const std::string& name, bool normalize)
05292 {
05293     std::vector<std::array<GLuint, 3>> group;
05294     std::vector<gg::GgSimpleShader::Material> material;
05295     std::vector<gg::GgVertex> vert;
05296     std::vector<GLuint> face;
05297
05298     // ファイルを読み込む
05299     if (!ggLoadSimpleObj(name, group, material, vert, face, normalize)) return nullptr;
05300
05301     // GgElements オブジェクトを作成する
05302     return std::make_shared<gg::GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05303         face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05304 }
05305
05306 //
05307 // メッシュ形状を作成する (Elements 形式)
05308 //
05309 std::shared_ptr<gg::GgElements> gg::ggElementsMesh(GLuint slices, GLuint stacks, const
05310     GLfloat(*pos)[3], const GLfloat(*norm)[3])
05311 {
05312     // 頂点属性
05313     std::vector<gg::GgVertex> vert((slices + 1) * (stacks + 1));
05314
05315     // 頂点の法線を求める
05316     for (GLuint j = 0; j <= stacks; ++j)
05317     {
05318         for (GLuint i = 0; i <= slices; ++i)
05319         {
05320             // 処理対象の頂点番号
05321             const auto k{ j * (slices + 1) + i };
05322
05323             // 頂点の法線
05324             gg::GgVector tnorm;
05325             tnorm[3] = 0.0f;

```

```

05325
05326     if (norm)
05327     {
05328         tnorm[0] = norm[k][0];
05329         tnorm[1] = norm[k][1];
05330         tnorm[2] = norm[k][2];
05331     }
05332     else
05333     {
05334         // 処理対象の頂点の周囲の頂点番号
05335         const auto kim{ i > 0 ? k - 1 : k };
05336         const auto kip{ i < slices ? k + 1 : k };
05337         const auto kjm{ j > 0 ? k - slices - 1 : k };
05338         const auto kjp{ j < stacks ? k + slices + 1 : k };
05339
05340         // 接線ベクトル
05341         const std::array<GLfloat, 3> t
05342         {
05343             pos[kip][0] - pos[kim][0],
05344             pos[kip][1] - pos[kim][1],
05345             pos[kip][2] - pos[kim][2]
05346         };
05347
05348         // 従接線ベクトル
05349         const std::array<GLfloat, 3> b
05350         {
05351             pos[kjp][0] - pos[kjm][0],
05352             pos[kjp][1] - pos[kjm][1],
05353             pos[kjp][2] - pos[kjm][2]
05354         };
05355
05356         // 法線
05357         tnorm[0] = t[1] * b[2] - t[2] * b[1];
05358         tnorm[1] = t[2] * b[0] - t[0] * b[2];
05359         tnorm[2] = t[0] * b[1] - t[1] * b[0];
05360
05361         // 法線の正規化
05362         ggNormalize3(tnorm);
05363     }
05364
05365         // 頂点の位置
05366         const gg::GgVector tpos{ pos[k][0], pos[k][1], pos[k][2], 1.0f };
05367
05368         // 頂点属性の保存
05369         vert.emplace_back(tpos, tnorm);
05370     }
05371
05372         // 頂点のインデックス (三角形データ)
05373         std::vector<GLuint> face;
05374
05375         // 頂点のインデックスを求める
05376         for (GLuint j = 0; j < stacks; ++j)
05377     {
05378         for (GLuint i = 0; i < slices; ++i)
05379         {
05380             // 処理対象のマス
05381             const auto k{ (slices + 1) * j + i };
05382
05383             // マスの上半分の三角形
05384             face.emplace_back(k);
05385             face.emplace_back(k + slices + 2);
05386             face.emplace_back(k + 1);
05387
05388             // マスのお下半分の三角形
05389             face.emplace_back(k);
05390             face.emplace_back(k + slices + 1);
05391             face.emplace_back(k + slices + 2);
05392
05393         }
05394     }
05395
05396         // GgElements オブジェクトを作成する
05397         return std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05398             face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05399     }
05400
05401     //
05402     // 球状に三角形データを生成する (Elements 形式)
05403     //
05404     std::shared_ptr<gg::GgElements> gg::ggElementsSphere(GLfloat radius, int slices, int stacks)
05405     {
05406         // 頂点の位置と法線
05407         std::vector<GLfloat> p, n;
05408
05409         // 頂点の位置と法線を求める
05410         for (int j = 0; j <= stacks; ++j)
05411     {

```

```

05412     const auto t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };
05413     const auto ph{ 3.1415926536f * t };
05414     const auto y{ cosf(ph) };
05415     const auto r{ sinf(ph) };
05416
05417     for (int i = 0; i <= slices; ++i)
05418     {
05419         const auto s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };
05420         const auto th{ -2.0f * 3.1415926536f * s };
05421         const auto x{ r * cosf(th) };
05422         const auto z{ r * sinf(th) };
05423
05424         // 頂点の座標値
05425         p.emplace_back(x * radius);
05426         p.emplace_back(y * radius);
05427         p.emplace_back(z * radius);
05428
05429         // 頂点の法線
05430         n.emplace_back(x);
05431         n.emplace_back(y);
05432         n.emplace_back(z);
05433     }
05434 }
05435
05436 // GgElements オブジェクトを作成する
05437 return ggElementsMesh(slices, stacks, reinterpret_cast<GLfloat(*)[3]>(p.data()),
05438   reinterpret_cast<GLfloat(*)[3]>(p.data())));
05439 }
05440
05441 //
05442 // 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
05443 //
05444 //    r 光源の強度の環境光成分の赤成分
05445 //    g 光源の強度の環境光成分の緑成分
05446 //    b 光源の強度の環境光成分の青成分
05447 //    a 光源の強度の環境光成分の不透明度
05448 //    first 値を設定する光源データの最初の番号
05449 //    count 値を設定する光源データの数
05450 //
05451 void gg::GgSimpleShader::LightBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05452 GLint first, GLsizei count) const
05453 {
05454     // 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
05455     const auto start{ static_cast<char*>(map(first, count)) };
05456     for (GLsizei i = 0; i < count; ++i)
05457     {
05458         // バッファオブジェクトの i 番目のブロックのポインタ
05459         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05460
05461         // 光源の環境光成分を設定する
05462         light->ambient[0] = r;
05463         light->ambient[1] = g;
05464         light->ambient[2] = b;
05465         light->ambient[3] = a;
05466     }
05467     unmap();
05468 }
05469
05470 //
05471 // 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
05472 //
05473 //    ambient 光源の強度の環境光成分
05474 //    first 値を設定する光源データの最初の番号
05475 //    count 値を設定する光源データの数
05476 //
05477 void gg::GgSimpleShader::LightBuffer::loadAmbient(const GgVector& ambient,
05478 GLint first, GLsizei count) const
05479 {
05480     // 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
05481     const auto start{ static_cast<char*>(map(first, count)) };
05482     for (GLsizei i = 0; i < count; ++i)
05483     {
05484         // バッファオブジェクトの i 番目のブロックのポインタ
05485         const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05486
05487         // 光源の強度の環境光成分を設定する
05488         light->ambient = ambient;
05489     }
05490     unmap();
05491 }
05492
05493 //
05494 // 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
05495 //
05496 //    r 光源の強度の拡散反射光成分の赤成分
05497 //    g 光源の強度の拡散反射光成分の緑成分
05498 //    b 光源の強度の拡散反射光成分の青成分

```

```
05499 // a 光源の強度の拡散反射光成分の不透明度
05500 // first 値を設定する光源データの最初の番号
05501 // count 値を設定する光源データの数
05502 //
05503 void gg::GgSimpleShader::LightBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05504 GLint first, GLsizei count) const
05505 {
05506 // 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
05507 const auto start{ static_cast<char*>(map(first, count)) };
05508 for (GLsizei i = 0; i < count; ++i)
05509 {
05510 // バッファオブジェクトの i 番目のブロックのポインタ
05511 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05512
05513 // 光源の拡散反射光成分を設定する
05514 light->diffuse[0] = r;
05515 light->diffuse[1] = g;
05516 light->diffuse[2] = b;
05517 light->diffuse[3] = a;
05518 }
05519 unmmap();
05520 }
05521
05522 //
05523 // 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
05524 //
05525 // ambient 光源の強度の拡散反射光成分
05526 // first 値を設定する光源データの最初の番号
05527 // count 値を設定する光源データの数
05528 //
05529 void gg::GgSimpleShader::LightBuffer::loadDiffuse(const GgVector& diffuse,
05530 GLint first, GLsizei count) const
05531 {
05532 // 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
05533 const auto start{ static_cast<char*>(map(first, count)) };
05534 for (GLsizei i = 0; i < count; ++i)
05535 {
05536 // バッファオブジェクトの i 番目のブロックのポインタ
05537 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05538
05539 // 光源の強度の拡散反射光成分を設定する
05540 light->diffuse = diffuse;
05541 }
05542 unmmap();
05543 }
05544
05545 //
05546 // 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
05547 //
05548 // r 光源の強度の鏡面反射光成分の赤成分
05549 // g 光源の強度の鏡面反射光成分の緑成分
05550 // b 光源の強度の鏡面反射光成分の青成分
05551 // a 光源の強度の鏡面反射光成分の不透明度
05552 // first 値を設定する光源データの最初の番号
05553 // count 値を設定する光源データの数
05554 //
05555 void gg::GgSimpleShader::LightBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05556 GLint first, GLsizei count) const
05557 {
05558 // 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
05559 const auto start{ static_cast<char*>(map(first, count)) };
05560 for (GLsizei i = 0; i < count; ++i)
05561 {
05562 // バッファオブジェクトの i 番目のブロックのポインタ
05563 const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05564
05565 // 光源の鏡面反射光成分を設定する
05566 light->specular[0] = r;
05567 light->specular[1] = g;
05568 light->specular[2] = b;
05569 light->specular[3] = a;
05570 }
05571 unmmap();
05572 }
05573
05574 //
05575 // 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
05576 //
05577 // ambient 光源の強度の鏡面反射光成分
05578 // first 値を設定する光源データの最初の番号
05579 // count 値を設定する光源データの数
05580 //
05581 void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& specular,
05582 GLint first, GLsizei count) const
05583 {
05584 // 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
05585 const auto start{ static_cast<char*>(map(first, count)) };
```

```

05586   for (GLsizei i = 0; i < count; ++i)
05587   {
05588     // バッファオブジェクトの i 番目のブロックのポインタ
05589     const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05590
05591     // 光源の強度の鏡面反射光成分を設定する
05592     light->specular = specular;
05593   }
05594   unmap();
05595 }
05596
05597 /**
05598 // 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
05599 /**
05600 // material 光源の特性の GgSimpleShader::Light 構造体
05601 // first 値を設定する光源データの最初の番号
05602 // count 値を設定する光源データの数
05603 /**
05604 void gg::GgSimpleShader::LightBuffer::loadColor(const Light& color,
05605   GLint first, GLsizei count) const
05606 {
05607   // 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
05608   const auto start{ static_cast<char*>(map(first, count)) };
05609   for (GLsizei i = 0; i < count; ++i)
05610   {
05611     // バッファオブジェクトの i 番目のブロックのポインタ
05612     const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05613
05614     // 光源の色を設定する
05615     light->ambient = color.ambient;
05616     light->diffuse = color.diffuse;
05617     light->specular = color.specular;
05618   }
05619   unmap();
05620 }
05621
05622 /**
05623 // 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
05624 /**
05625 // x 光源の位置の x 座標
05626 // y 光源の位置の y 座標
05627 // z 光源の位置の z 座標
05628 // w 光源の位置の w 座標
05629 // first 値を設定する光源データの最初の番号
05630 // count 値を設定する光源データの数
05631 /**
05632 void gg::GgSimpleShader::LightBuffer::loadPosition(GLfloat x, GLfloat y, GLfloat z, GLfloat w,
05633   GLint first, GLsizei count) const
05634 {
05635   // 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
05636   const auto start{ static_cast<char*>(map(first, count)) };
05637   for (GLsizei i = 0; i < count; ++i)
05638   {
05639     // バッファオブジェクトの i 番目のブロックのポインタ
05640     const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05641
05642     // 光源の位置を設定する
05643     light->position[0] = x;
05644     light->position[1] = y;
05645     light->position[2] = z;
05646     light->position[3] = w;
05647   }
05648   unmap();
05649 }
05650
05651 /**
05652 // 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
05653 /**
05654 // position 光源の位置
05655 // first 値を設定する光源データの最初の番号
05656 // count 値を設定する光源データの数
05657 /**
05658 void gg::GgSimpleShader::LightBuffer::loadPosition(const GgVector& position,
05659   GLint first, GLsizei count) const
05660 {
05661   // 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
05662   const auto start{ static_cast<char*>(map(first, count)) };
05663   for (GLsizei i = 0; i < count; ++i)
05664   {
05665     // バッファオブジェクトの i 番目のブロックのポインタ
05666     const auto light{ reinterpret_cast<Light*>(start + getStride() * i) };
05667
05668     // 光源の位置を設定する
05669     light->position = position;
05670   }
05671   unmap();
05672 }

```

```
05673
05674 // 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
05675 //
05676 //
05677 // r 環境光に対する反射係数の赤成分
05678 // g 環境光に対する反射係数の緑成分
05679 // b 環境光に対する反射係数の青成分
05680 // a 環境光に対する反射係数の不透明度
05681 // first 値を設定する材質データの最初の番号
05682 // count 値を設定する材質データの数
05683 //
05684 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05685 GLint first, GLsizei count) const
05686 {
05687 // 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
05688 const auto start{ static_cast<char*>(map(first, count)) };
05689 for (GLsizei i = 0; i < count; ++i)
05690 {
05691 // バッファオブジェクトの i 番目のブロックのポインタ
05692 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05693
05694 // 環境光に対する反射係数を設定する
05695 material->ambient[0] = r;
05696 material->ambient[1] = g;
05697 material->ambient[2] = b;
05698 material->ambient[3] = a;
05699 }
05700 unmap();
05701 }
05702
05703 //
05704 // 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
05705 //
05706 // ambient 環境光に対する反射係数
05707 // first 値を設定する光源データの最初の番号
05708 // count 値を設定する光源データの数
05709 //
05710 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(const GgVector& ambient,
05711 GLint first, GLsizei count) const
05712 {
05713 // 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
05714 const auto start{ static_cast<char*>(map(first, count)) };
05715 for (GLsizei i = 0; i < count; ++i)
05716 {
05717 // バッファオブジェクトの i 番目のブロックのポインタ
05718 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05719
05720 // 光源の強度の環境光成分を設定する
05721 material->ambient = ambient;
05722 }
05723 unmap();
05724 }
05725
05726 //
05727 // 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
05728 //
05729 // r 拡散反射係数の赤成分
05730 // g 拡散反射係数の緑成分
05731 // b 拡散反射係数の青成分
05732 // a 拡散反射係数の不透明度
05733 // first 値を設定する材質データの最初の番号
05734 // count 値を設定する材質データの数
05735 //
05736 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05737 GLint first, GLsizei count) const
05738 {
05739 // 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
05740 const auto start{ static_cast<char*>(map(first, count)) };
05741 for (GLsizei i = 0; i < count; ++i)
05742 {
05743 // バッファオブジェクトの i 番目のブロックのポインタ
05744 const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05745
05746 // 拡散反射係数を設定する
05747 material->diffuse[0] = r;
05748 material->diffuse[1] = g;
05749 material->diffuse[2] = b;
05750 material->diffuse[3] = a;
05751 }
05752 unmap();
05753 }
05754
05755 //
05756 // 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
05757 //
05758 // diffuse 拡散反射係数
05759 // first 値を設定する光源データの最初の番号
```

```

05760 //   count 値を設定する光源データの数
05761 //
05762 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(const GgVector& diffuse,
05763   GLint first, GLsizei count) const
05764 {
05765   // 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
05766   const auto start{ static_cast<char*>(map(first, count)) };
05767   for (GLsizei i = 0; i < count; ++i)
05768   {
05769     // バッファオブジェクトの i 番目のブロックのポインタ
05770     const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05771     // 光源の強度の環境光成分を設定する
05773     material->diffuse = diffuse;
05774   }
05775   unmap();
05776 }
05777
05778 //
05779 // 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
05780 //
05781 //   r 環境光に対する反射係数と拡散反射係数の赤成分
05782 //   g 環境光に対する反射係数と拡散反射係数の緑成分
05783 //   b 環境光に対する反射係数と拡散反射係数の青成分
05784 //   a 環境光に対する反射係数と拡散反射係数の不透明度
05785 //   first 値を設定する材質データの最初の番号
05786 //   count 値を設定する材質データの数
05787 //
05788 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(GLfloat r, GLfloat g, GLfloat b,
05789   GLfloat a,
05790   GLint first, GLsizei count) const
05791 {
05792   // 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
05793   const auto start{ static_cast<char*>(map(first, count)) };
05794   for (GLsizei i = 0; i < count; ++i)
05795   {
05796     // バッファオブジェクトの i 番目のブロックのポインタ
05797     const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05798     // 環境光に対する反射係数と拡散反射係数を設定する
05799     material->ambient[0] = material->diffuse[0] = r;
05800     material->ambient[1] = material->diffuse[1] = g;
05801     material->ambient[2] = material->diffuse[2] = b;
05802     material->ambient[3] = material->diffuse[3] = a;
05803   }
05804   unmap();
05805 }
05806
05807 //
05808 // 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
05809 //
05810 //   color 環境光に対する反射係数と拡散反射係数
05811 //   first 値を設定する光源データの最初の番号
05812 //   count 値を設定する光源データの数
05813 //
05814 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GgVector& color,
05815   GLint first, GLsizei count) const
05816 {
05817   // 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
05818   const auto start{ static_cast<char*>(map(first, count)) };
05819   for (GLsizei i = 0; i < count; ++i)
05820   {
05821     // バッファオブジェクトの i 番目のブロックのポインタ
05822     const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05823     // 光源の強度の環境光成分を設定する
05824     material->ambient = material->diffuse = color;
05825   }
05826 }
05827 unmap();
05828 }
05829
05830 //
05831 // 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
05832 //
05833 //   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
05834 //   first 値を設定する材質データの最初の番号
05835 //   count 値を設定する材質データの数
05836 //
05837 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GLfloat* color,
05838   GLint first, GLsizei count) const
05839 {
05840   // ambient 要素のバイトオフセット
05841   constexpr auto ambientOffset{ offsetof(Material, ambient) };
05842
05843   // ambient 要素のバイト数
05844   constexpr auto ambientSize{ sizeof(Material::diffuse) };
05845

```

```

05846 // diffuse 要素のバイトオフセット
05847 constexpr auto diffuseOffset{ offsetof(Material, diffuse) };
05848
05849 // diffuse 要素のバイト数
05850 constexpr auto diffuseSize{ sizeof(Material::diffuse) };
05851
05852 // 元のデータの先頭
05853 const auto* source{ reinterpret_cast<const char*>(color) };
05854
05855 // 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
05856 bind();
05857 for (GLsizei i = 0; i < count; ++i)
05858 {
    // 格納先
    const auto destination{ getStride() * (first + i) };

    // first + i 番目のブロックの ambient 要素に値を設定する
    glBufferSubData(getTarget(), destination + ambientOffset, ambientSize, source + i * ambientSize);

    // first + i 番目のブロックの diffuse 要素に値を設定する
    glBufferSubData(getTarget(), destination + diffuseOffset, diffuseSize, source + i * diffuseSize);
}
05869
05870 //
05871 // 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
05872 //
05873 // r 鏡面反射係数の赤成分
05874 // g 鏡面反射係数の緑成分
05875 // b 鏡面反射係数の青成分
05876 // a 鏡面反射係数の不透明度
05877 // first 値を設定する材質データの最初の番号
05878 // count 値を設定する材質データの数
05879 //
05880 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05881 GLint first, GLsizei count) const
05882 {
    // 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
    const auto start{ static_cast<char*>(map(first, count)) };
    for (GLsizei i = 0; i < count; ++i)
    {
        // バッファオブジェクトの i 番目のブロックのポインタ
        const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };

        // 鏡面反射係数を設定する
        material->specular[0] = r;
        material->specular[1] = g;
        material->specular[2] = b;
        material->specular[3] = a;
    }
    unmap();
}
05897 }

05898
05899 //
05900 // 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
05901 //
05902 // specular 鏡面反射係数
05903 // first 値を設定する光源データの最初の番号
05904 // count 値を設定する光源データの数
05905 //
05906 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(const GgVector& specular,
05907 GLint first, GLsizei count) const
05908 {
    // 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
    const auto start{ static_cast<char*>(map(first, count)) };
    for (GLsizei i = 0; i < count; ++i)
    {
        // バッファオブジェクトの i 番目のブロックのポインタ
        const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };

        // 光源の強度の環境光成分を設定する
        material->specular = specular;
    }
    unmap();
}
05919 }

05920 }

05921
05922 //
05923 // 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
05924 //
05925 // shininess 輝き係数
05926 // first 値を設定する材質データの最初の番号
05927 // count 値を設定する材質データの数
05928 //
05929 void gg::GgSimpleShader::MaterialBuffer::loadShininess(GLfloat shininess,
05930 GLint first, GLsizei count) const
05931 {
    // 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

```

```

05933 const auto start{ static_cast<char*>(map(first, count)) };
05934 for (GLsizei i = 0; i < count; ++i)
05935 {
05936     // バッファオブジェクトの i 番目のブロックのポインタ
05937     const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05938     material->shininess = shininess;
05939 }
05940 unmap();
05941 }
05942
05943 //
05944 // 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
05945 //
05946 //    shininess 輝き係数
05947 //    first 値を設定する材質データの最初の番号
05948 //    count 値を設定する材質データの数
05949 //
05950 void gg::GgSimpleShader::MaterialBuffer::loadShininess(const GLfloat* shininess,
05951 GLint first, GLsizei count) const
05952 {
05953     // 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
05954     const auto start{ static_cast<char*>(map(first, count)) };
05955     for (GLsizei i = 0; i < count; ++i)
05956     {
05957         // バッファオブジェクトの i 番目のブロックのポインタ
05958         const auto material{ reinterpret_cast<Material*>(start + getStride() * i) };
05959         material->shininess = shininess[i];
05960     }
05961 unmap();
05962 }
05963
05964 //
05965 // 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
05966 //
05967 bool gg::GgSimpleShader::load(const std::string& vert, const std::string& frag, const std::string&
geom,
05968 GLint nvarying, const char* const* varyings)
05969 {
05970     if (!GgPointShader::load(vert, frag, geom, nvarying, varyings)) return false;
05971
05972 mnLoc = glGetUniformLocation(get(), "mn");
05973 const auto lightIndex{ glGetUniformLocation(get(), "Light") };
05974 glUniformBlockBinding(get(), lightIndex, LightBindingPoint);
05975 const auto materialIndex{ glGetUniformLocation(get(), "Material") };
05976 glUniformBlockBinding(get(), materialIndex, MaterialBindingPoint);
05977
05978 return true;
05979 }
05980
05981 //
05982 // Wavefront OBJ 形式のデータ：コンストラクタ
05983 //
05984 gg::GgSimpleObj::GgSimpleObj(const std::string& name, bool normalize)
05985 {
05986     // 作業用のメモリ
05987     std::vector<GgSimpleShader::Material> mat;
05988     std::vector<GgVertex> vert;
05989     std::vector<GLuint> face;
05990
05991     // グループのデータのメモリを確保する
05992     group = std::make_shared<std::vector<std::array<GLuint, 3>>>();
05993
05994     // ファイルを読み込む
05995     if (ggLoadSimpleObj(name, *group, mat, vert, face, normalize))
05996     {
05997         // 頂点バッファオブジェクトを作成する
05998         data = std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05999             face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
06000
06001     // 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
06002     glBindVertexArray(0);
06003
06004     // 材質データを設定する
06005     material = std::make_shared<GgSimpleShader::MaterialBuffer>(mat.data(),
06006         static_cast<GLsizei>(mat.size()));
06007 }
06008
06009 //
06010 // Wavefront OBJ 形式のデータ：図形の描画
06011 //
06012 void gg::GgSimpleObj::draw(GLint first, GLsizei count) const
06013 {
06014     // 保持しているグループの数
06015     const auto ng{ static_cast<GLsizei>(group->size()) };
06016
06017     // 描画する最後のグループの次

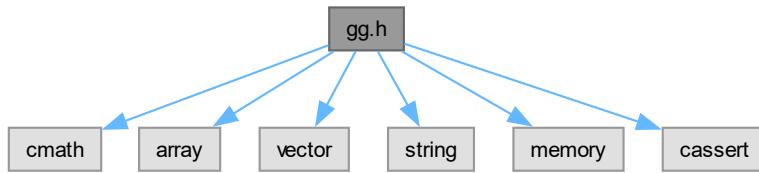
```

```
06018 auto last{ count <= 0 ? ng : first + count };
06019 if (last > ng) last = ng;
06020
06021 for (GLsizei i = first; i < last; ++i)
06022 {
06023     // グループのデータ
06024     const auto& g{ (*group)[i] };
06025
06026     // 材質を設定する
06027     material->select(g[2]);
06028
06029     // 図形を描画する
06030     data->draw(g[0], g[1]);
06031 }
06032 }
```

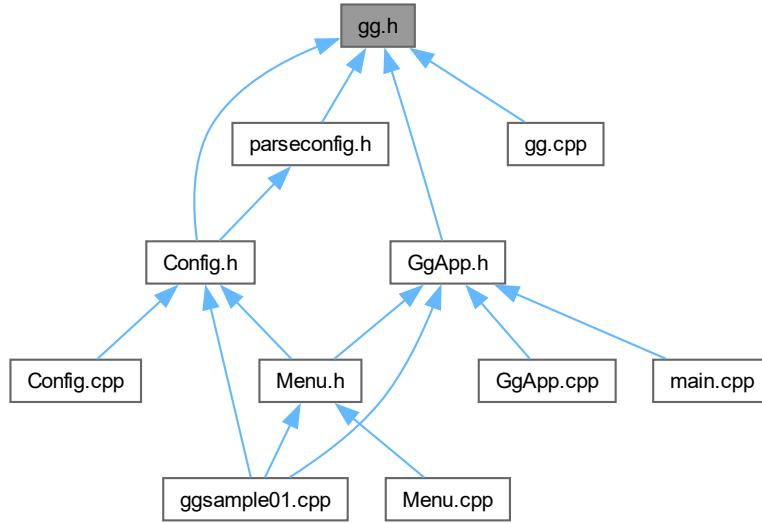
9.7 gg.h ファイル

```
#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::ggiInvert (const GgMatrix &m)`
- `GgMatrix gg::ggNormal (const GgMatrix &m)`
- `GgQuaternion gg::ggiIdentityQuaternion ()`
- `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::ggEulerQuaternion (const GgVector &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::ggiInvert (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=false)`
- `GLuint gg::ggLoadImage (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE)`
- `void gg::ggCreateNormalMap (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector<GgVector> &nmap)`

- GLuint `gg::ggLoadHeight` (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)
- GLuint `gg::ggCreateShader` (const std::string &vsr, const std::string &fsr="", const std::string &gsr="", GLint nvarying=0, const char *const *varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string >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 &csr, const std::string &ctext="compute shader")
- GLuint `gg::ggLoadComputeShader` (const std::string &comp)
- std::shared_ptr< `GgPoints` > `gg::ggPointsCube` (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared_ptr< `GgPoints` > `gg::ggPointsSphere` (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared_ptr< `GgTriangles` > `gg::ggRectangle` (GLfloat width=1.0f, GLfloat height=1.0f)
- std::shared_ptr< `GgTriangles` > `gg::ggEllipse` (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)
- std::shared_ptr< `GgTriangles` > `gg::ggArraysObj` (const std::string &name, bool normalize=false)
- std::shared_ptr< `GgElements` > `gg::ggElementsObj` (const std::string &name, bool normalize=false)
- std::shared_ptr< `GgElements` > `gg::ggElementsMesh` (GLuint slices, GLuint stacks, const GLfloat(*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。

9.7.1 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。

9.7.2 マクロ定義詳解

9.7.2.1 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。

9.7.2.2 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。

9.7.3 型定義詳解

9.7.3.1 pathChar

```
using pathChar = char
```

[gg.h](#) の 78 行目に定義があります。

9.7.3.2 pathString

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

[gg.h](#) の 77 行目に定義があります。

9.7.4 関数詳解

9.7.4.1 TCharToUtf8()

```
std::string TCHARToUtf8 (
    const pathString & cstring ) [inline]
```

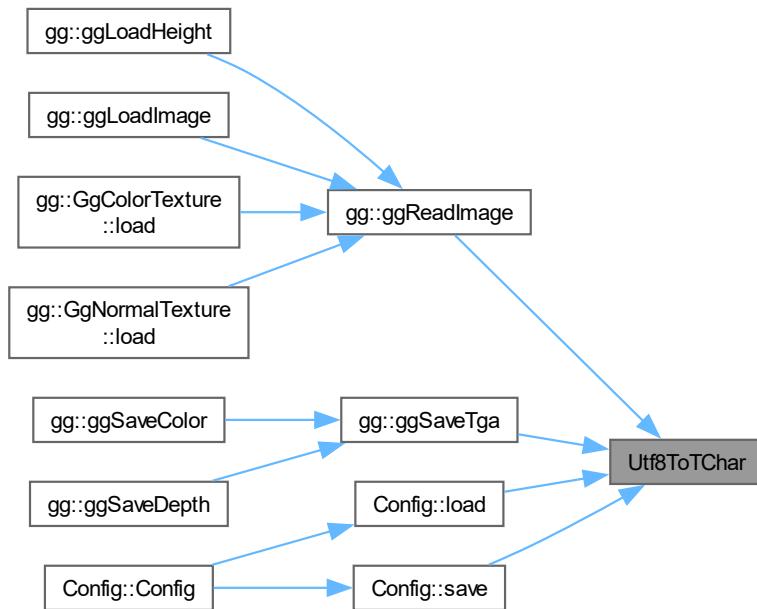
[gg.h](#) の 80 行目に定義があります。

9.7.4.2 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:



9.8 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 = wchart;
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
00083
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 PFNGLBEGINCONDITIONALRENDERNVPROC glBeginConditionalRenderNV;
00108 extern PFNGLBEGINCONDITIONALRENDERPROC glBeginConditionalRender;
00109 extern PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00110 extern PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00111 extern PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00112 extern PFNGLBEGINQUERYPROC glBeginQuery;
00113 extern PFNGLBEGINTRANSFORMFEEDBACKPROC 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 PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
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 PFNGLBLENEQUATIONIARBPROC glBindEquationiARB;
00141 extern PFNGLBLENEQUATIONIPROC glBindEquationi;
00142 extern PFNGLBLENEQUATIONPROC glBindEquation;
00143 extern PFNGLBLENEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00144 extern PFNGLBLENEQUATIONSEPARATEIPROC glBindEquationSeparatei;

```

```
00145 extern PFNGLBLENDEQUATIONSEPARATEPROC glBlendEquationSeparate;
00146 extern PFNGLBLENDFUNCIARBPROC glBlendFunciARB;
00147 extern PFNGLBLENDFUNCIPROC glBlendFunci;
00148 extern PFNGLBLENDFUNCPROC glBlendFunc;
00149 extern PFNGLBLENDFUNCSEPARATEIARBPROC glBlendFuncSeparateiARB;
00150 extern PFNGLBLENDFUNCSEPARATEIPROC glBlendFuncSeparatei;
00151 extern PFNGLBLENDFUNCSEPARATEPROC glBlendFuncSeparate;
00152 extern PFNGLBLENDPARAMETERINVPROC glBlendParameteriNV;
00153 extern PFNGLBLITFRAMEBUFFERPROC glBlitFramebuffer;
00154 extern PFNGLBLITNAMEDFRAMEBUFFERPROC glBlitNamedFramebuffer;
00155 extern PFNGLBUFFERADDRESSRANGEVPROC glBufferAddressRangeNV;
00156 extern PFNGLBUFFERDATAPROC glBufferData;
00157 extern PFNGLBUFFERPAGECOMMITEMNTARBPROC glBufferPageCommitmentARB;
00158 extern PFNGLBUFFERSTORAGEPROC glBufferStorage;
00159 extern PFNGLBUFFERSUBDATAPROC glBufferSubData;
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 PFNGLCLEARBUFFERRIVPROC 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 PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC 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 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
00213 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00214 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00215 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00216 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00217 extern PFNGLCONSERVATERASTERPARAMETERFNVPROC glConservativeRasterParameterfnv;
00218 extern PFNGLCONSERVATERASTERPARAMETERINVPROC 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 PFNGLCREATEESAMPLERSPROC 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 PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexAttribArrayEXT;
00304 extern PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArray;
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 PFNGLENABLEVERTEXARRAYATTRAYEXTPROC 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 PFNGLFENCECSYNCPROC glFenceSync;
00355 extern PFNGLFINISHPROC glFinish;
00356 extern PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00357 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00358 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
00359 extern PFNGLFLUSHPROC glFlush;
00360 extern PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00361 extern PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00362 extern PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00363 extern PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00364 extern PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00365 extern PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00366 extern PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
00367 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glFramebufferSampleLocationsfvARB;
00368 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
00369 extern PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00370 extern PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00371 extern PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00372 extern PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
00373 extern PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00374 extern PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
00375 extern PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00376 extern PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00377 extern PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
00378 extern PFNGLFRONTFACEPROC 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 PFNGLGENSAMPLESPROC 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 glGetBooleanav;
00409 extern PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00410 extern PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00411 extern PFNGLGETBUFFERPARAMETERUI64NVPROC glGetBufferParameterui64vNV;
00412 extern PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00413 extern PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00414 extern PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00415 extern PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
00416 extern PFNGLGETCOMPRESSEDTEXTIMAGEPROC 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 PFNGLGETERRORPROC 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 PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
00443 extern PFNGLGETINTEGER64VPROC glGetInteger64v;
00444 extern PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00445 extern PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00446 extern PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00447 extern PFNGLGETINTEGERUI64NVPROC glGetIntegerui64NV;
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 glGetMultiTexParameterIiivEXT;
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 PFNGLGETNAMEDPROGRAMEXTPROC glGetNamedProgramvEXT;
00478 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00479 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00480 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIiivEXT;
00481 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00482 extern PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00483 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00484 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00485 extern PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00486 extern PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00487 extern PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
00488 extern PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
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 PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00496 extern PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00497 extern PFNGLGETNUNIFORMI64VPROC glGetnUniformivARB;
00498 extern PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
00499 extern PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00500 extern PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
00501 extern PFNGLGETNUNIFORMUINVPROC 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 PFNGLGETPATHMETRICRANGENVPROC glGetPathMetricRangeNV;
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 PFNGLGETPOINTERINDEXDEVEXTPROC glGetPointerIndexedvEXT;
00525 extern PFNGLGETPOINTERI_VEXTPROC glGetPointeri_VEXT;
00526 extern PFNGLGETPOINTERVPROC 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 PFNGLGETPROGRAMRESOURCECFVNVPROC glGetProgramResourcefvNV;
00534 extern PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00535 extern PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00536 extern PFNGLGETPROGRAMRESOURCELLOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00537 extern PFNGLGETPROGRAMRESOURCELLOCATIONPROC glGetProgramResourceLocation;
00538 extern PFNGLGETPROGRAMCENAMENPROC 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 PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00551 extern PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00552 extern PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00553 extern PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
00554 extern PFNGLGETSAMPLERPARAMETERIVPROC 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 PFNGLGETSUBROUTINEINDEXNPROC glGetSubroutineUniformLocation;
00564 extern PFNGLGETSYNCIVPROC glGetSynciv;
00565 extern PFNGLGETTEXIMAGEPROC glGetTexImage;
00566 extern PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00567 extern PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00568 extern PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00569 extern PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00570 extern PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00571 extern PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
00572 extern PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00573 extern PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
00574 extern PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00575 extern PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00576 extern PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00577 extern PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;

```

```

00578 extern PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC 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 PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
00587 extern PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00588 extern PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00589 extern PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
00590 extern PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00591 extern PFNGLGETTRANSFORMFEEDBACKI64_VPROC glGetTransformFeedbacki64_v;
00592 extern PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00593 extern PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
00594 extern PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00595 extern PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformLocation;
00596 extern PFNGLGETUNIFORMDVPROC glGetUniformLocation;
00597 extern PFNGLGETUNIFORMFVPROC glGetUniformLocation;
00598 extern PFNGLGETUNIFORMI64VARBPROC glGetUniformLocationi64vARB;
00599 extern PFNGLGETUNIFORMI64NVNVPROC glGetUniformLocationi64vNV;
00600 extern PFNGLGETUNIFORMINDICESPROC glGetUniformLocationIndices;
00601 extern PFNGLGETUNIFORMIVPROC glGetUniformLocationiv;
00602 extern PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocationLocation;
00603 extern PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformLocationSubroutineuiv;
00604 extern PFNGLGETUNIFORMUI64VARBPROC glGetUniformLocationui64vARB;
00605 extern PFNGLGETUNIFORMUI64NVNVPROC glGetUniformLocationui64vNV;
00606 extern PFNGLGETUNIFORMUIVPROC glGetUniformLocationuiv;
00607 extern PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00608 extern PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexeddiv;
00609 extern PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00610 extern PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00611 extern PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00612 extern PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00613 extern PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00614 extern PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribArraybdv;
00615 extern PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribArraybfv;
00616 extern PFNGLGETVERTEXATTRIBIIIVPROC glGetVertexAttribArrayIiiv;
00617 extern PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribArrayIuiv;
00618 extern PFNGLGETVERTEXATTRIBLIVPROC glGetVertexAttribArrayLdv;
00619 extern PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribArrayLdsv;
00620 extern PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribArrayLi64vNV;
00621 extern PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribArrayAttribLui64vARB;
00622 extern PFNGLGETVERTEXATTRIBLUI64NVNVPROC glGetVertexAttribArrayAttribLui64vNV;
00623 extern PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribArrayPointerv;
00624 extern PFNGLGETVKPROCAADDRNVPROC glGetVkProcAddrNV;
00625 extern PFNGLHINTPROC glHint;
00626 extern PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00627 extern PFNGLINSETEVENTMARKEREXTPROC glInsertEventMarkerEXT;
00628 extern PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00629 extern PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
00630 extern PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferDataSubData;
00631 extern PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
00632 extern PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
00633 extern PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
00634 extern PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
00635 extern PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00636 extern PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00637 extern PFNGLISBUFFERPROC glIsBuffer;
00638 extern PFNGLISBUFERRESIDENTNVPROC glIsBufferResidentNV;
00639 extern PFNGLCOMMANDLISTNVPROC glIsCommandListNV;
00640 extern PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00641 extern PFNGLISENABLEDIPROC glIsEnabledi;
00642 extern PFNGLISENABLEDPROC glIsEnabled;
00643 extern PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00644 extern PFNGLISIMAGEHANDLEARTBPROC glIsImageHandleResidentARB;
00645 extern PFNGLISIMAGEHANDLEARTNVPROC glIsImageHandleResidentNV;
00646 extern PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00647 extern PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00648 extern PFNGLISPATHNVPROC glIsPathNV;
00649 extern PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00650 extern PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00651 extern PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00652 extern PFNGLISPROGRAMPROC glIsProgram;
00653 extern PFNGLISQUERYPROC glIsQuery;
00654 extern PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00655 extern PFNGLISSAMPLERPROC glIsSampler;
00656 extern PFNGLISSHADERPROC glIsShader;
00657 extern PFNGLISSTATENVPROC glIsStateNV;
00658 extern PFNGLISSYNCPROC glIsSync;
00659 extern PFNGLISTEXTUREHANDLEARTBPROC glIsTextureHandleResidentARB;
00660 extern PFNGLISTTEXTUREHANDLEARTNVPROC glIsTextureHandleResidentNV;
00661 extern PFNGLISTTEXTUREPROC glIsTexture;
00662 extern PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00663 extern PFNGLISVERTEXARRAYPROC glIsVertexArray;
00664 extern PFNGLLABELOBJECTEXTPROC glLabelObjectEXT;

```

```
00665 extern PFNGLLINEWIDTHPROC glLineWidth;
00666 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 glMatrixLoadddEXT;
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 PFNGLMATRIXROTATEFEXTPROC glMatrixRotateffEXT;
00708 extern PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
00709 extern PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
00710 extern PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
00711 extern PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslateffEXT;
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 PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00727 extern PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00728 extern PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00729 extern PFNGLMULTITEXCOORDPOINTEREEXTPROC glMultiTexCoordPointerEXT;
00730 extern PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00731 extern PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00732 extern PFNGLMULTITEXENVVIEXTPROC glMultiTexEnvivEXT;
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 glMultiTexParameteriuvEXT;
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 PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00768 extern PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00769 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSVARBPROC glNamedFramebufferSampleLocationsfvARB;
00770 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFNVPROC glNamedFramebufferSampleLocationsfvNV;
00771 extern PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00772 extern PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00773 extern PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00774 extern PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00775 extern PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00776 extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00777 extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00778 extern PFNGLNAMEDFRAMEBUFFERTEXTUREPROC 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 glNamedProgramLocalParameterI4uivEXT;
00787 extern PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00788 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IEXTPROC glNamedProgramLocalParametersI4iEXT;
00789 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
00790 extern PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00791 extern PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00792 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERGEEXTPROC
    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 glPathParameterfNV;
00812 extern PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00813 extern PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
00814 extern PFNGLPATHPARAMETERINVNVPROC glPathParameterivNV;
00815 extern PFNGLPATHSTENCILDEPTHHOFFSETNVPROC glPathStencilDepthOffsetNV;
00816 extern PFNGLPATHSTENCILDEPTHFUNCNVPROC glPathStencilFuncNV;
00817 extern PFNGLPATHSTRINGNVPROC glPathStringNV;
00818 extern PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00819 extern PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00820 extern PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00821 extern PFNGLPIXELSTOREFPROC 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 PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00831 extern PFNGLPOLYGONOFFSETPROC glPolygonOffset;
00832 extern PFNGLPODEBUGGROUPPROC glPopDebugGroup;
00833 extern PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00834 extern PFNGLPRIMITIVEBOUNDINGBOXARBPROC 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 PFNGLPROGRAMUNIFORM3DPROC 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 PFNGLPROGRAMUNIFORM4DPROC 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 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
00945 extern PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
00946 extern PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00947 extern PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC 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 PFNGLSAMPLEPARAMETERFFPROC glSamplerParameterf;
00997 extern PFNGLSAMPLEPARAMETERRFVPROC glSamplerParameterfv;
00998 extern PFNGLSAMPLEPARAMETERIIVPROC glSamplerParameterIiiv;
00999 extern PFNGLSAMPLEPARAMETERIIPROC glSamplerParameterIi;
01000 extern PFNGLSAMPLEPARAMETERIUIVPROC glSamplerParameterIuiv;
01001 extern PFNGLSAMPLEPARAMETERIIVPROC glSamplerParameteriv;
01002 extern PFNGLSCISSORARRAYPROC glScissorArrayv;
01003 extern PFNGLSCISSORINDEXEDPROC glScissorIndexed;
01004 extern PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
01005 extern PFNGLSCISSORPROC glScissor;
01006 extern PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
01007 extern PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
01008 extern PFNGLSHADERBINARYPROC glShaderBinary;
01009 extern PFNGLSHADERSOURCEPROC glShaderSource;
01010 extern PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
01011 extern PFNGLSIGNALKFENCENVPROC 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 PFNGLSTENCILOPPROC glStencilOp;
01022 extern PFNGLSTENCILOPSEPARATEPROC glStencilOpSeparate;
01023 extern PFNGLSTENCILOPSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
01024 extern PFNGLSTENCILOSTROKEPATHNVPROC glStencilStrokePathNV;
01025 extern PFNGLSTENCIILTENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01026 extern PFNGLSTENCIILTENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01027 extern PFNGLSTENCIILTENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01028 extern PFNGLSTENCIILTENCOVERSTROKEPATHNVPROC 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 PFNGLTEXPARAMETERIPROC glTexParameterIi;
01044 extern PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01045 extern PFNGLTEXPARAMETERIVPROC glTexParameteriv;
01046 extern PFNGLTEXSTORAGE1DPROC glTexStorage1D;
01047 extern PFNGLTEXSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
01048 extern PFNGLTEXSTORAGE2DPROC glTexStorage2D;
01049 extern PFNGLTEXSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
01050 extern PFNGLTEXSTORAGE3DPROC glTexStorage3D;
01051 extern PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01052 extern PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01053 extern PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01054 extern PFNGLTEXTUREBARRIERNVPROC glTextureBarrier;
01055 extern PFNGLTEXTUREBARRIERTEXTPROC glTextureBufferEXT;
01056 extern PFNGLTEXTUREBUFFEREXTPROC glTextureBuffer;
01057 extern PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01058 extern PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01059 extern PFNGLTEXTUREBUFFERRANGEXTPROC glTextureBufferRangeEXT;
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 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIiuvEXT;
01070 extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIiuv;
01071 extern PFNGLTEXTUREPARAMETERIIPROC glTextureParameteri;
01072 extern PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIiuvEXT;
01073 extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIiuv;
01074 extern PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterivEXT;
01075 extern PFNGLTEXTUREPARAMETERIVPROC glTextureParameteriv;
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 PFNGLTRANSFORMFEEDBACKRANGEPROC 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 PFNGLUNIFORM1IIPROC glUniform1i;
01107 extern PFNGLUNIFORM1IVPROC glUniform1iv;
01108 extern PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01109 extern PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01110 extern PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
01111 extern PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
01112 extern PFNGLUNIFORM1UIIPROC 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 PFNGLUNIFORM2IIPROC glUniform2i;
01123 extern PFNGLUNIFORM2IVPROC glUniform2iv;
01124 extern PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01125 extern PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01126 extern PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
01127 extern PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
01128 extern PFNGLUNIFORM2UIIPROC 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 PFNGLUNIFORM3IIPROC glUniform3i;
01139 extern PFNGLUNIFORM3IVPROC glUniform3iv;
01140 extern PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01141 extern PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01142 extern PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
01143 extern PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
01144 extern PFNGLUNIFORM3UIIPROC 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 PFNGLUNIFORM4IIPROC glUniform4i;
01155 extern PFNGLUNIFORM4IVPROC glUniform4iv;
01156 extern PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01157 extern PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01158 extern PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
01159 extern PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
01160 extern PFNGLUNIFORM4UIIPROC glUniform4ui;
01161 extern PFNGLUNIFORM4UIVPROC glUniform4uiv;
01162 extern PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01163 extern PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01164 extern PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01165 extern PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
01166 extern PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
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 glUniformui64vNV;
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 PFNGLVERTEXARRAYATTRIBIFORMATPROC glVertexArrayAttribIFormat;
01199 extern PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
01200 extern PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01201 extern PFNGLVERTEXARRAYBINDVERTEXBUFFEREEXTPROC glVertexArrayBindVertexBufferEXT;
01202 extern PFNGLVERTEXARRAYCOLOROFFSETSETEXTPROC 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 glVertexArrayVertexAttribBindingEXT;
01212 extern PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01213 extern PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01214 extern PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01215 extern PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01216 extern PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
01217 extern PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
01218 extern PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01219 extern PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01220 extern PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
01221 extern PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01222 extern PFNGLVERTEXARRAYVERTEXOFFSETSETEXTPROC glVertexArrayVertexOffsetEXT;
01223 extern PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
01224 extern PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
01225 extern PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
01226 extern PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
01227 extern PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
01228 extern PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
01229 extern PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
01230 extern PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
01231 extern PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
01232 extern PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
01233 extern PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2s;
01234 extern PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
01235 extern PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
01236 extern PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
01237 extern PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
01238 extern PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
01239 extern PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
01240 extern PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
01241 extern PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
01242 extern PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
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 glVertexAttrib4nui;
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 PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
01268 extern PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01269 extern PFNGLVERTEXATTRIBI2IUIVPROC glVertexAttribI2ui;
01270 extern PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2uiv;
01271 extern PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
01272 extern PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
```

```

01273 extern PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
01274 extern PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01275 extern PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
01276 extern PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01277 extern PFNGLVERTEXATTRIBI4IPROC 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 glVertexAttribL1ui64NV;
01293 extern PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
01294 extern PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribL1ui64vNV;
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 PFNGLVIEWPORTINDEXEDFPROC glViewportIndexeddf;
01329 extern PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;
01330 extern PFNGLVIEWPORTPOSITIONWSCALENVPROC 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
01531     inline GLfloat ggDistance4(const GLfloat* a, const GLfloat* b)
01532     {
01533         const GLfloat c[] { a[0] - b[0], a[1] - b[1], a[2] - b[2], a[3] - b[3] };
01534         return ggLength4(c);
01535     }
01536
01543     inline void ggNormalize4(const GLfloat* a, GLfloat* b)
01544     {
01545         const GLfloat l { ggLength4(a) };
01546         assert(l > 0.0f);
01547         b[0] = a[0] / l;
01548         b[1] = a[1] / l;
01549         b[2] = a[2] / l;
01550         b[3] = a[3] / l;
01551     }
01552
01558     inline void ggNormalize4(GLfloat* a)
01559     {
01560         const GLfloat l { ggLength4(a) };
01561         assert(l > 0.0f);
01562         a[0] /= l;
01563         a[1] /= l;
01564         a[2] /= l;
01565         a[3] /= l;
01566     }
01567
01571     class GgVector : public std::array<GLfloat, 4>
01572     {
```

```
01573 public:
01574     GgVector() = default;
01575
01576     constexpr GgVector(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3) :
01577         std::array<GLfloat, 4>{ v0, v1, v2, v3 }
01578     {
01579     }
01580
01581     constexpr GgVector(const GLfloat* a) :
01582         GgVector{ a[0], a[1], a[2], a[3] }
01583     {
01584     }
01585
01586     constexpr GgVector(GLfloat c) :
01587         GgVector{ c, c, c, c }
01588     {
01589     }
01590
01591     GgVector(const GgVector& v) = default;
01592
01593     GgVector(GgVector&& v) = default;
01594
01595     ~GgVector() = default;
01596
01597     GgVector& operator=(const GgVector& v) = default;
01598
01599     GgVector& operator=(GgVector&& v) = default;
01600
01601     inline GgVector operator+(const GgVector& v) const
01602     {
01603         return GgVector{ (*this)[0] + v[0], (*this)[1] + v[1], (*this)[2] + v[2], (*this)[3] + v[3] };
01604     }
01605
01606     inline GgVector operator+(GLfloat c) const
01607     {
01608         return *this + GgVector(c);
01609     }
01610
01611     inline GgVector& operator+=(const GgVector& v)
01612     {
01613         *this = *this + v;
01614         return *this;
01615     }
01616
01617     inline GgVector& operator+=(GLfloat c)
01618     {
01619         *this = *this + c;
01620         return *this;
01621     }
01622
01623     inline GgVector operator-(const GgVector& v) const
01624     {
01625         return GgVector{ (*this)[0] - v[0], (*this)[1] - v[1], (*this)[2] - v[2], (*this)[3] - v[3] };
01626     }
01627
01628     inline GgVector operator-(GLfloat c) const
01629     {
01630         return *this - GgVector(c);
01631     }
01632
01633     inline GgVector& operator-=(const GgVector& v)
01634     {
01635         *this = *this - v;
01636         return *this;
01637     }
01638
01639     inline GgVector& operator-=(GLfloat c)
01640     {
01641         *this = *this - c;
01642         return *this;
01643     }
01644
01645     inline GgVector operator*(const GgVector& v) const
01646     {
01647         return GgVector{ (*this)[0] * v[0], (*this)[1] * v[1], (*this)[2] * v[2], (*this)[3] * v[3] };
01648     }
01649
01650     inline GgVector operator*(GLfloat c) const
01651     {
01652         return *this * GgVector(c);
01653     }
01654
01655     inline GgVector& operator*=(const GgVector& v)
01656     {
01657         *this = *this * v;
01658         return *this;
01659     }
```

```
01765     }
01766
01767     inline GgVector& operator*(GLfloat c)
01768     {
01769         *this = *this * c;
01770         return *this;
01771     }
01772
01773     inline GgVector operator/(const GgVector& v) const
01774     {
01775         return GgVector{ (*this)[0] / v[0], (*this)[1] / v[1], (*this)[2] / v[2], (*this)[3] / v[3] };
01776     }
01777
01778     inline GgVector operator/(GLfloat c) const
01779     {
01780         return *this / GgVector(c);
01781     }
01782
01783     inline GgVector& operator/=(GgVector& v)
01784     {
01785         *this = *this / v;
01786         return *this;
01787     }
01788
01789     inline GgVector& operator/=(GLfloat c)
01790     {
01791         *this = *this / c;
01792         return *this;
01793     }
01794
01795     inline GLfloat dot3(const GgVector& v) const
01796     {
01797         return ggDot3(data(), v.data());
01798     }
01799
01800     inline GLfloat length3() const
01801     {
01802         return ggLength3(data());
01803     }
01804
01805     inline GLfloat distance3(const GgVector& v) const
01806     {
01807         return ggDistance3(data(), v.data());
01808     }
01809
01810     inline GgVector normalize3() const
01811     {
01812         GgVector b;
01813         ggNormalize3(data(), b.data());
01814         return b;
01815     }
01816
01817     inline GLfloat dot4(const GgVector& v) const
01818     {
01819         return ggDot4(data(), v.data());
01820     }
01821
01822     inline GLfloat length4() const
01823     {
01824         return ggLength4(data());
01825     }
01826
01827     inline GLfloat distance4(const GgVector& v) const
01828     {
01829         return ggDistance4(data(), v.data());
01830     }
01831
01832     inline GgVector normalize4() const
01833     {
01834         GgVector b;
01835         ggNormalize4(data(), b.data());
01836         return b;
01837     }
01838
01839     inline const GgVector& operator+(const GgVector& v)
01840     {
01841         return v;
01842     }
01843
01844     inline GgVector operator+(GLfloat a, const GgVector& b)
01845     {
01846         return GgVector{ a + b[0], a + b[1], a + b[2], a + b[3] };
01847     }
01848
01849     inline const GgVector operator-(const GgVector& v)
01850     {
```

```

01945     return GgVector{ -v[0], -v[1], -v[2], -v[3] };
01946 }
01947
01948 inline GgVector operator-(GLfloat a, const GgVector& b)
01949 {
01950     return GgVector{ a - b[0], a - b[1], a - b[2], a - b[3] };
01951 }
01952
01953
01954 inline GgVector operator*(GLfloat a, const GgVector& b)
01955 {
01956     return GgVector{ a * b[0], a * b[1], a * b[2], a * b[3] };
01957 }
01958
01959
01960 inline GgVector operator/(GLfloat a, const GgVector& b)
01961 {
01962     return GgVector{ a / b[0], a / b[1], a / b[2], a / b[3] };
01963 }
01964
01965
01966 inline GgVector ggCross(const GgVector& a, const GgVector& b)
01967 {
01968     GgVector c;
01969     ggCross(c.data(), a.data(), b.data());
01970     return c;
01971 }
01972
01973
01974 inline GLfloat ggDot3(const GgVector& a, const GgVector& b)
01975 {
01976     return ggDot3(a.data(), b.data());
01977 }
01978
01979
01980 inline GLfloat ggLength3(const GgVector& a)
01981 {
01982     return ggLength3(a.data());
01983 }
01984
01985
01986 inline GLfloat ggDistance3(const GgVector& a, const GgVector& b)
01987 {
01988     return ggDistance3(a.data(), b.data());
01989 }
01990
01991
01992 inline GgVector ggNormalize3(const GgVector& a)
01993 {
01994     GgVector b;
01995     ggNormalize3(a.data(), b.data());
01996     return b;
01997 }
01998
01999
02000
02001 inline void ggNormalize3(GgVector* a)
02002 {
02003     ggNormalize3(a->data());
02004 }
02005
02006
02007 inline GLfloat ggDot4(const GgVector& a, const GgVector& b)
02008 {
02009     return ggDot4(a.data(), b.data());
02010 }
02011
02012
02013 inline GLfloat ggLength4(const GgVector& a)
02014 {
02015     return ggLength4(a.data());
02016 }
02017
02018
02019 inline GLfloat ggDistance4(const GgVector& a, const GgVector& b)
02020 {
02021     return ggDistance4(a.data(), b.data());
02022 }
02023
02024
02025 inline GgVector ggNormalize4(const GgVector& a)
02026 {
02027     GgVector b;
02028     ggNormalize4(a.data(), b.data());
02029     return b;
02030 }
02031
02032
02033
02034 inline void ggNormalize4(GgVector* a)
02035 {
02036     ggNormalize4(a->data());
02037 }
02038
02039
02040 class GgMatrix : public std::array<GLfloat, 16>
02041 {
02042     // 行列 a とベクトル b の積をベクトル c に格納する
02043     void projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02044
02045     // 行列 a と行列 b の積を行列 c に格納する
02046     void multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133

```

```
02134     public:
02135
02136     GgMatrix() = default;
02137
02138     constexpr GgMatrix(
02139         GLfloat m00, GLfloat m01, GLfloat m02, GLfloat m03,
02140         GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13,
02141         GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23,
02142         GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33
02143     ) :
02144         std::array<GLfloat, 16>{ m00, m01, m02, m03, m10, m11, m12, m13, m20, m21, m22, m23, m30, m31,
02145         m32, m33 }
02146     {
02147     }
02148
02149     constexpr GgMatrix(const GLfloat* a) :
02150         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],
02151         a[13], a[14], a[15] }
02152     {
02153     }
02154
02155     constexpr GgMatrix(GLfloat c) :
02156         GgMatrix{ c, c }
02157     {
02158     }
02159
02160     GgMatrix(const GgMatrix& m) = default;
02161
02162     GgMatrix(GgMatrix&& m) = default;
02163
02164     ~GgMatrix() = default;
02165
02166     GgMatrix& operator=(const GgMatrix& m) = default;
02167
02168     GgMatrix& operator=(GgMatrix&& m) = default;
02169
02170     GgMatrix& operator=(const GLfloat* a)
02171     {
02172         *this = GgMatrix(a);
02173         return *this;
02174     }
02175
02176     GgMatrix operator+(const GLfloat* a) const
02177     {
02178         GgMatrix t;
02179         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] + a[i];
02180         return t;
02181     }
02182
02183     GgMatrix operator+(const GgMatrix& m) const
02184     {
02185         return operator+(m.data());
02186     }
02187
02188     GgMatrix& operator+=(const GLfloat* a)
02189     {
02190         for (std::size_t i = 0; i < size(); ++i) data()[i] += a[i];
02191         return *this;
02192     }
02193
02194     GgMatrix& operator+=(const GgMatrix& m)
02195     {
02196         return operator+=(m.data());
02197     }
02198
02199     GgMatrix operator-(const GLfloat* a) const
02200     {
02201         GgMatrix t;
02202         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] - a[i];
02203         return t;
02204     }
02205
02206     GgMatrix operator-(const GgMatrix& m) const
02207     {
02208         return operator-(m.data());
02209     }
02210
02211     GgMatrix& operator-=(const GLfloat* a)
02212     {
02213         for (std::size_t i = 0; i < size(); ++i) data()[i] -= a[i];
02214         return *this;
02215     }
02216
02217     GgMatrix& operator-=(const GgMatrix& m)
02218     {
02219         return operator-=(m.data());
02220     }
```

```
02325
02326     GgMatrix operator*(const GLfloat* a) const
02327     {
02328         GgMatrix t;
02329         multiply(t.data(), data(), a);
02330         return t;
02331     }
02332
02333     GgMatrix operator*(const GgMatrix& m) const
02334     {
02335         return operator*(m.data());
02336     }
02337
02338     GgMatrix& operator*=(const GLfloat* a)
02339     {
02340         *this = operator*(a);
02341         return *this;
02342     }
02343
02344     GgMatrix& operator*=(const GgMatrix& m)
02345     {
02346         return operator*=(m.data());
02347     }
02348
02349     GgMatrix operator/(const GLfloat* a) const
02350     {
02351         GgMatrix i, t;
02352         i.loadInvert(a);
02353         multiply(t.data(), data(), i.data());
02354         return t;
02355     }
02356
02357     GgMatrix operator/(const GgMatrix& m) const
02358     {
02359         return operator/(m.data());
02360     }
02361
02362     GgMatrix& operator/=(const GLfloat* a)
02363     {
02364         *this = operator/(a);
02365         return *this;
02366     }
02367
02368     GgMatrix& operator/=(const GgMatrix& m)
02369     {
02370         return operator/=(m.data());
02371     }
02372
02373     GgMatrix& loadIdentity();
02374
02375     GgMatrix& loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02376
02377     GgMatrix& loadTranslate(const GLfloat* t)
02378     {
02379         return loadTranslate(t[0], t[1], t[2]);
02380     }
02381
02382     GgMatrix& loadTranslate(const GgVector& t)
02383     {
02384         return loadTranslate(t[0], t[1], t[2], t[3]);
02385     }
02386
02387     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02388
02389     GgMatrix& loadScale(const GLfloat* s)
02390     {
02391         return loadScale(s[0], s[1], s[2]);
02392     }
02393
02394     GgMatrix& loadScale(const GgVector& s)
02395     {
02396         return loadScale(s[0], s[1], s[2], s[3]);
02397     }
02398
02399     GgMatrix& loadRotateX(GLfloat a);
02400
02401     GgMatrix& loadRotateY(GLfloat a);
02402
02403     GgMatrix& loadRotateZ(GLfloat a);
02404
02405     GgMatrix& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
02406
02407     GgMatrix& loadRotate(const GLfloat* r, GLfloat a)
02408     {
02409         return loadRotate(r[0], r[1], r[2], a);
02410     }
```

```
02546     GgMatrix& loadRotate(const GgVector& r, GLfloat a)
02547     {
02548         return loadRotate(r[0], r[1], r[2], a);
02549     }
02550
02551     GgMatrix& loadRotate(const GLfloat* r)
02552     {
02553         return loadRotate(r[0], r[1], r[2], r[3]);
02554     }
02555
02556     GgMatrix& loadRotate(const GgVector& r)
02557     {
02558         return loadRotate(r[0], r[1], r[2], r[3]);
02559     }
02560
02561     GgMatrix& loadLookat(GLfloat ex, GLfloat ey, GLfloat ez,
02562                           GLfloat tx, GLfloat ty, GLfloat tz,
02563                           GLfloat ux, GLfloat uy, GLfloat uz);
02564
02565
02566     GgMatrix& loadLookat(const GLfloat* e, const GLfloat* t, const GLfloat* u)
02567     {
02568         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02569     }
02570
02571     GgMatrix& loadLookat(const GgVector& e, const GgVector& t, const GgVector& u)
02572     {
02573         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02574     }
02575
02576     GgMatrix& loadOrthogonal(GLfloat left, GLfloat right,
02577                               GLfloat bottom, GLfloat top,
02578                               GLfloat zNear, GLfloat zFar);
02579
02580     GgMatrix& loadFrustum(GLfloat left, GLfloat right,
02581                           GLfloat bottom, GLfloat top,
02582                           GLfloat zNear, GLfloat zFar);
02583
02584     GgMatrix& loadPerspective(GLfloat fovy, GLfloat aspect,
02585                               GLfloat zNear, GLfloat zFar);
02586
02587     GgMatrix& loadTranspose(const GLfloat* a);
02588
02589     GgMatrix& loadTranspose(const GgMatrix& m)
02590     {
02591         return loadTranspose(m.data());
02592     }
02593
02594     GgMatrix& loadInvert(const GLfloat* a);
02595
02596     GgMatrix& loadInvert(const GgMatrix& m)
02597     {
02598         return loadInvert(m.data());
02599     }
02600
02601     GgMatrix& loadNormal(const GLfloat* a);
02602
02603     GgMatrix& loadNormal(const GgMatrix& m)
02604     {
02605         return loadNormal(m.data());
02606     }
02607
02608     GgMatrix translate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02609     {
02610         GgMatrix m;
02611         return operator*(m.loadTranslate(x, y, z, w));
02612     }
02613
02614     GgMatrix translate(const GLfloat* t) const
02615     {
02616         return translate(t[0], t[1], t[2]);
02617     }
02618
02619     GgMatrix translate(const GgVector& t) const
02620     {
02621         return translate(t[0], t[1], t[2], t[3]);
02622     }
02623
02624     GgMatrix scale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02625     {
02626         GgMatrix m;
02627         return operator*(m.loadScale(x, y, z, w));
02628     }
02629
02630     GgMatrix scale(const GLfloat* s) const
02631     {
02632         return scale(s[0], s[1], s[2]);
02633     }
```

```

02778
02785     GgMatrix scale(const GgVector& s) const
02786     {
02787         return scale(s[0], s[1], s[2], s[3]);
02788     }
02789
02796     GgMatrix rotateX(GLfloat a) const
02797     {
02798         GgMatrix m;
02799         return operator*(m.loadRotateX(a));
02800     }
02801
02808     GgMatrix rotateY(GLfloat a) const
02809     {
02810         GgMatrix m;
02811         return operator*(m.loadRotateY(a));
02812     }
02813
02820     GgMatrix rotateZ(GLfloat a) const
02821     {
02822         GgMatrix m;
02823         return operator*(m.loadRotateZ(a));
02824     }
02825
02835     GgMatrix rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
02836     {
02837         GgMatrix m;
02838         return operator*(m.loadRotate(x, y, z, a));
02839     }
02840
02848     GgMatrix rotate(const GLfloat* r, GLfloat a) const
02849     {
02850         return rotate(r[0], r[1], r[2], a);
02851     }
02852
02860     GgMatrix rotate(const GgVector& r, GLfloat a) const
02861     {
02862         return rotate(r[0], r[1], r[2], a);
02863     }
02864
02871     GgMatrix rotate(const GLfloat* r) const
02872     {
02873         return rotate(r[0], r[1], r[2], r[3]);
02874     }
02875
02882     GgMatrix rotate(const GgVector& r) const
02883     {
02884         return rotate(r[0], r[1], r[2], r[3]);
02885     }
02886
02901     GgMatrix lookat(
02902         GLfloat ex, GLfloat ey, GLfloat ez,
02903         GLfloat tx, GLfloat ty, GLfloat tz,
02904         GLfloat ux, GLfloat uy, GLfloat uz
02905     ) const
02906     {
02907         GgMatrix m;
02908         return operator*(m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz));
02909     }
02910
02919     GgMatrix lookat(const GLfloat* e, const GLfloat* t, const GLfloat* u) const
02920     {
02921         return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02922     }
02923
02932     GgMatrix lookat(const GgVector& e, const GgVector& t, const GgVector& u) const
02933     {
02934         return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02935     }
02936
02948     GgMatrix orthogonal(
02949         GLfloat left, GLfloat right,
02950         GLfloat bottom, GLfloat top,
02951         GLfloat zNear, GLfloat zFar
02952     ) const
02953     {
02954         GgMatrix m;
02955         return operator*(m.loadOrthogonal(left, right, bottom, top, zNear, zFar));
02956     }
02957
02969     GgMatrix frustum(
02970         GLfloat left, GLfloat right,
02971         GLfloat bottom, GLfloat top,
02972         GLfloat zNear, GLfloat zFar
02973     ) const
02974     {
02975         GgMatrix m;

```

```
02976     return operator*(m.loadFrustum(left, right, bottom, top, zNear, zFar));
02977 }
02978
02988 GgMatrix perspective(
02989     GLfloat fovy, GLfloat aspect,
02990     GLfloat zNear, GLfloat zFar
02991 ) const
02992 {
02993     GgMatrix m;
02994     return operator*(m.loadPerspective(fovy, aspect, zNear, zFar));
02995 }
02996
03002 GgMatrix transpose() const
03003 {
03004     GgMatrix t;
03005     return t.loadTranspose(*this);
03006 }
03007
03013 GgMatrix invert() const
03014 {
03015     GgMatrix t;
03016     return t.loadInvert(*this);
03017 }
03018
03024 GgMatrix normal() const
03025 {
03026     GgMatrix t;
03027     return t.loadNormal(*this);
03028 }
03029
03036 void projection(GLfloat* c, const GLfloat* v) const
03037 {
03038     projection(c, data(), v);
03039 }
03040
03047 void projection(GLfloat* c, const GgVector& v) const
03048 {
03049     projection(c, v.data());
03050 }
03051
03058 void projection(GgVector& c, const GLfloat* v) const
03059 {
03060     projection(c.data(), v);
03061 }
03062
03069 void projection(GgVector& c, const GgVector& v) const
03070 {
03071     projection(c.data(), v.data());
03072 }
03073
03080 GgVector operator*(const GgVector& v) const
03081 {
03082     GgVector c;
03083     projection(c, v);
03084     return c;
03085 }
03086
03092 const GLfloat* get() const
03093 {
03094     return data();
03095 }
03096
03102 void get(GLfloat* a) const
03103 {
03104     std::copy(data(), data() + size(), a);
03105 }
03106 };
03107
03113 inline GgMatrix ggIdentity()
03114 {
03115     GgMatrix t;
03116     return t.loadIdentity();
03117 };
03118
03128 inline GgMatrix ggTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03129 {
03130     GgMatrix m;
03131     return m.loadTranslate(x, y, z, w);
03132 }
03133
03140 inline GgMatrix ggTranslate(const GLfloat* t)
03141 {
03142     GgMatrix m;
03143     return m.loadTranslate(t[0], t[1], t[2]);
03144 }
03145
03152 inline GgMatrix ggTranslate(const GgVector& t)
```

```

03153 {
03154     GgMatrix m;
03155     return m.loadTranslate(t[0], t[1], t[2], t[3]);
03156 }
03157
03158 inline GgMatrix ggScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03159 {
03160     GgMatrix m;
03161     return m.loadScale(x, y, z, w);
03162 }
03163
03164 inline GgMatrix ggScale(const GLfloat* s)
03165 {
03166     GgMatrix m;
03167     return m.loadScale(s[0], s[1], s[2]);
03168 }
03169
03170 inline GgMatrix ggScale(const GgVector& s)
03171 {
03172     GgMatrix m;
03173     return m.loadScale(s[0], s[1], s[2], s[3]);
03174 }
03175
03176 inline GgMatrix ggRotateX(GLfloat a)
03177 {
03178     GgMatrix m;
03179     return m.loadRotateX(a);
03180 }
03181
03182 inline GgMatrix ggRotateY(GLfloat a)
03183 {
03184     GgMatrix m;
03185     return m.loadRotateY(a);
03186 }
03187
03188 inline GgMatrix ggRotateZ(GLfloat a)
03189 {
03190     GgMatrix m;
03191     return m.loadRotateZ(a);
03192 }
03193
03194 inline GgMatrix ggRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03195 {
03196     GgMatrix m;
03197     return m.loadRotate(x, y, z, a);
03198 }
03199
03200 inline GgMatrix ggRotate(const GLfloat* r, GLfloat a)
03201 {
03202     GgMatrix m;
03203     return m.loadRotate(r[0], r[1], r[2], a);
03204 }
03205
03206 inline GgMatrix ggRotate(const GgVector& r, GLfloat a)
03207 {
03208     GgMatrix m;
03209     return m.loadRotate(r[0], r[1], r[2], a);
03210 }
03211
03212 inline GgMatrix ggRotate(const GLfloat* r)
03213 {
03214     GgMatrix m;
03215     return m.loadRotate(r[0], r[1], r[2], r[3]);
03216 }
03217
03218 inline GgMatrix ggRotate(const GgVector& r)
03219 {
03220     GgMatrix m;
03221     return m.loadRotate(r[0], r[1], r[2], r[3]);
03222 }
03223
03224 inline GgMatrix ggLookat(
03225     GLfloat ex, GLfloat ey, GLfloat ez,           // 視点の位置
03226     GLfloat tx, GLfloat ty, GLfloat tz,           // 目標点の位置
03227     GLfloat ux, GLfloat uy, GLfloat uz           // 上方向のベクトル
03228 )
03229 {
03230     GgMatrix m;
03231     return m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz);
03232 }
03233
03234 inline GgMatrix ggLookat(
03235     const GLfloat* e,                           // 視点の位置
03236     const GLfloat* t,                           // 目標点の位置
03237     const GLfloat* u                           // 上方向のベクトル
03238 )
03239 {

```

```
03336     GgMatrix m;
03337     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03338 }
03339
03340 inline GgMatrix ggLookat(
03341     const GgVector& e,                                // 視点の位置
03342     const GgVector& t,                                // 目標点の位置
03343     const GgVector& u                                // 上方向のベクトル
03344 )
03345 {
03346     GgMatrix m;
03347     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03348 }
03349
03350 inline GgMatrix ggOrthogonal(GLfloat left, GLfloat right,
03351     GLfloat bottom, GLfloat top,
03352     GLfloat zNear, GLfloat zFar)
03353 {
03354     GgMatrix m;
03355     return m.loadOrthogonal(left, right, bottom, top, zNear, zFar);
03356 }
03357
03358 inline GgMatrix ggFrustum(
03359     GLfloat left, GLfloat right,
03360     GLfloat bottom, GLfloat top,
03361     GLfloat zNear, GLfloat zFar
03362 )
03363 {
03364     GgMatrix m;
03365     return m.loadFrustum(left, right, bottom, top, zNear, zFar);
03366 }
03367
03368 inline GgMatrix ggPerspective(
03369     GLfloat fovy, GLfloat aspect,
03370     GLfloat zNear, GLfloat zFar
03371 )
03372 {
03373     GgMatrix m;
03374     return m.loadPerspective(fovy, aspect, zNear, zFar);
03375 }
03376
03377 inline GgMatrix ggTranspose(const GgMatrix& m)
03378 {
03379     return m.transpose();
03380 }
03381
03382 inline GgMatrix ggInvert(const GgMatrix& m)
03383 {
03384     return m.invert();
03385 }
03386
03387 inline GgMatrix ggNormal(const GgMatrix& m)
03388 {
03389     return m.normal();
03390 }
03391
03392 class GgQuaternion : public GgVector
03393 {
03394     // GgQuaternion 型の四元数 p と四元数 q の積を四元数 r に求める
03395     void multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const;
03396
03397     // GgQuaternion 型の四元数 q が表す回転の変換行列を m に求める
03398     void toMatrix(GLfloat* m, const GLfloat* q) const;
03399
03400     // 回転の変換行列 m が表す四元数を q に求める
03401     void toQuaternion(GLfloat* q, const GLfloat* m) const;
03402
03403     // 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
03404     void slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const;
03405
03406 public:
03407
03408     GgQuaternion() = default;
03409
03410     constexpr GgQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w) :
03411         GgVector{ x, y, z, w }
03412     {
03413     }
03414
03415     constexpr GgQuaternion(GLfloat c) :
03416         GgQuaternion{ c, c, c, c }
03417     {
03418     }
03419
03420     GgQuaternion(const GLfloat* a) :
03421         GgQuaternion{ a[0], a[1], a[2], a[3] }
03422     {
03423     }
```

```

03504     }
03505
03511     GgQuaternion(const GgVector& v) :
03512         GgQuaternion{ v[0], v[1], v[2], v[3] }
03513     {
03514     }
03515
03521     GgQuaternion(const GgQuaternion& q) = default;
03522
03528     GgQuaternion(GgQuaternion&& q) = default;
03529
03533     ~GgQuaternion() = default;
03534
03541     GgQuaternion& operator=(const GgQuaternion& q) = default;
03542
03549     GgQuaternion& operator=(GgQuaternion&& q) = default;
03550
03556     GLfloat norm() const
03557     {
03558         return ggLength4(*this);
03559     }
03560
03570     GgQuaternion& loadAdd(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03571     {
03572         (*this)[0] += x;
03573         (*this)[1] += y;
03574         (*this)[2] += z;
03575         (*this)[3] += w;
03576
03577         return *this;
03578     }
03579
03586     GgQuaternion& loadAdd(const GLfloat* a)
03587     {
03588         return loadAdd(a[0], a[1], a[2], a[3]);
03589     }
03590
03597     GgQuaternion& loadAdd(const GgVector& v)
03598     {
03599         return loadAdd(v[0], v[1], v[2], v[3]);
03600     }
03601
03608     GgQuaternion& loadAdd(const GgQuaternion& q)
03609     {
03610         return loadAdd(q[0], q[1], q[2], q[3]);
03611     }
03612
03622     GgQuaternion& loadSubtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03623     {
03624         (*this)[0] -= x;
03625         (*this)[1] -= y;
03626         (*this)[2] -= z;
03627         (*this)[3] -= w;
03628
03629         return *this;
03630     }
03631
03638     GgQuaternion& loadSubtract(const GLfloat* a)
03639     {
03640         return loadSubtract(a[0], a[1], a[2], a[3]);
03641     }
03642
03649     GgQuaternion& loadSubtract(const GgVector& v)
03650     {
03651         return loadSubtract(v[0], v[1], v[2], v[3]);
03652     }
03653
03660     GgQuaternion& loadSubtract(const GgQuaternion& q)
03661     {
03662         return loadSubtract(q[0], q[1], q[2], q[3]);
03663     }
03664
03674     GgQuaternion& loadMultiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03675     {
03676         const GLfloat a[] { x, y, z, w };
03677         return loadMultiply(a);
03678     }
03679
03686     GgQuaternion& loadMultiply(const GLfloat* a)
03687     {
03688         return *this = multiply(a);
03689     }
03690
03697     GgQuaternion& loadMultiply(const GgVector& v)
03698     {
03699         return loadMultiply(v.data());
03700     }

```

```
03701
03702     GgQuaternion& loadMultiply(const GgQuaternion& q)
03703     {
03704         return loadMultiply(q.data());
03705     }
03706
03707     GgQuaternion& loadDivide(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03708     {
03709         const GLfloat a[] { x, y, z, w };
03710         return loadDivide(a);
03711     }
03712
03713     GgQuaternion& loadDivide(const GLfloat* a)
03714     {
03715         return *this = divide(a);
03716     }
03717
03718     GgQuaternion& loadDivide(const GgVector& v)
03719     {
03720         return loadDivide(v.data());
03721     }
03722
03723     GgQuaternion& loadDivide(const GgQuaternion& q)
03724     {
03725         return loadDivide(q.data());
03726     }
03727
03728     GgQuaternion add(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03729     {
03730         GgQuaternion s{ *this };
03731         return s.loadAdd(x, y, z, w);
03732     }
03733
03734     GgQuaternion add(const GLfloat* a) const
03735     {
03736         return add(a[0], a[1], a[2], a[3]);
03737     }
03738
03739     GgQuaternion add(const GgVector& v) const
03740     {
03741         return add(v[0], v[1], v[2], v[3]);
03742     }
03743
03744     GgQuaternion add(const GgQuaternion& q) const
03745     {
03746         return add(q[0], q[1], q[2], q[3]);
03747     }
03748
03749     GgQuaternion subtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03750     {
03751         GgQuaternion s{ *this };
03752         return s.loadSubtract(x, y, z, w);
03753     }
03754
03755     GgQuaternion subtract(const GLfloat* a) const
03756     {
03757         return subtract(a[0], a[1], a[2], a[3]);
03758     }
03759
03760     GgQuaternion subtract(const GgVector& v) const
03761     {
03762         return subtract(v[0], v[1], v[2], v[3]);
03763     }
03764
03765     GgQuaternion subtract(const GgQuaternion& q) const
03766     {
03767         return subtract(q[0], q[1], q[2], q[3]);
03768     }
03769
03770     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03771     {
03772         const GLfloat a[] { x, y, z, w };
03773         return multiply(a);
03774     }
03775
03776     GgQuaternion multiply(const GLfloat* a) const
03777     {
03778         GgQuaternion s;
03779         multiply(s.data(), data(), a);
03780         return s;
03781     }
03782
03783     GgQuaternion multiply(const GgVector& v) const
03784     {
03785         return multiply(v.data());
03786     }
03787
03788     GgQuaternion multiply(const GgQuaternion& q) const
03789     {
03790         return multiply(q[0], q[1], q[2], q[3]);
03791     }
03792
03793     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03794     {
03795         const GLfloat a[] { x, y, z, w };
03796         return multiply(a);
03797     }
03798
03799     GgQuaternion multiply(const GgVector& v) const
03800     {
03801         return multiply(v.data());
03802     }
03803
03804     GgQuaternion multiply(const GgQuaternion& q) const
03805     {
03806         return multiply(q[0], q[1], q[2], q[3]);
03807     }
03808
03809     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03810     {
03811         const GLfloat a[] { x, y, z, w };
03812         return multiply(a);
03813     }
03814
03815     GgQuaternion multiply(const GLfloat* a) const
03816     {
03817         GgQuaternion s;
03818         multiply(s.data(), data(), a);
03819         return s;
03820     }
03821
03822     GgQuaternion multiply(const GgVector& v) const
03823     {
03824         return multiply(v.data());
03825     }
03826
03827     GgQuaternion multiply(const GgQuaternion& q) const
03828     {
03829         return multiply(q[0], q[1], q[2], q[3]);
03830     }
03831
03832     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03833     {
03834         const GLfloat a[] { x, y, z, w };
03835         return multiply(a);
03836     }
03837
03838     GgQuaternion multiply(const GLfloat* a) const
03839     {
03840         GgQuaternion s;
03841         multiply(s.data(), data(), a);
03842         return s;
03843     }
03844
03845     GgQuaternion multiply(const GgVector& v) const
03846     {
03847         return multiply(v.data());
03848     }
03849
03850     GgQuaternion multiply(const GgQuaternion& q) const
03851     {
03852         return multiply(q[0], q[1], q[2], q[3]);
03853     }
03854
03855     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03856     {
03857         const GLfloat a[] { x, y, z, w };
03858         return multiply(a);
03859     }
03860
03861     GgQuaternion multiply(const GLfloat* a) const
03862     {
03863         GgQuaternion s;
03864         multiply(s.data(), data(), a);
03865         return s;
03866     }
03867
03868     GgQuaternion multiply(const GgVector& v) const
03869     {
03870         return multiply(v.data());
03871     }
03872
03873     GgQuaternion multiply(const GgQuaternion& q) const
03874     {
03875         return multiply(q[0], q[1], q[2], q[3]);
03876     }
03877
03878     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03879     {
03880         const GLfloat a[] { x, y, z, w };
03881         return multiply(a);
03882     }
03883
03884     GgQuaternion multiply(const GLfloat* a) const
03885     {
03886         GgQuaternion s;
03887         multiply(s.data(), data(), a);
03888         return s;
03889     }
03890
03891     GgQuaternion multiply(const GgVector& v) const
03892     {
03893         return multiply(v.data());
03894     }
03895
```

```
03902     GgQuaternion multiply(const GgQuaternion& q) const
03903     {
03904         return multiply(q.data());
03905     }
03906
03916     GgQuaternion divide(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03917     {
03918         const GLfloat a[] { x, y, z, w };
03919         return divide(a);
03920     }
03921
03928     GgQuaternion divide(const GLfloat* a) const
03929     {
03930         GgQuaternion s, ia;
03931         ia.loadInvert(a);
03932         multiply(s.data(), data(), ia.data());
03933         return s;
03934     }
03935
03942     GgQuaternion divide(const GgVector& v) const
03943     {
03944         return divide(v.data());
03945     }
03946
03953     GgQuaternion divide(const GgQuaternion& q) const
03954     {
03955         return divide(q.data());
03956     }
03957
03958     // 演算子
03959     GgQuaternion& operator=(const GLfloat* a)
03960     {
03961         return *this = GgQuaternion(a);
03962     }
03963     GgQuaternion& operator=(const GgVector& v)
03964     {
03965         return *this = GgQuaternion(v);
03966     }
03967     GgQuaternion& operator+=(const GLfloat* a)
03968     {
03969         return loadAdd(a);
03970     }
03971     GgQuaternion& operator+=(const GgVector& v)
03972     {
03973         return loadAdd(v);
03974     }
03975     GgQuaternion& operator+=(const GgQuaternion& q)
03976     {
03977         return loadAdd(q);
03978     }
03979     GgQuaternion& operator==(const GLfloat* a)
03980     {
03981         return loadSubtract(a);
03982     }
03983     GgQuaternion& operator==(const GgVector& v)
03984     {
03985         return loadSubtract(v);
03986     }
03987     GgQuaternion& operator==(const GgQuaternion& q)
03988     {
03989         return operator==(q.data());
03990     }
03991     GgQuaternion& operator*=(const GLfloat* a)
03992     {
03993         return loadMultiply(a);
03994     }
03995     GgQuaternion& operator*=(const GgVector& v)
03996     {
03997         return loadMultiply(v);
03998     }
03999     GgQuaternion& operator*=(const GgQuaternion& q)
04000     {
04001         return operator*=(q.data());
04002     }
04003     GgQuaternion& operator/=(const GLfloat* a)
04004     {
04005         return loadDivide(a);
04006     }
04007     GgQuaternion& operator/=(const GgVector& v)
04008     {
04009         return loadDivide(v);
04010     }
04011     GgQuaternion& operator/=(const GgQuaternion& q)
04012     {
04013         return operator/=(q.data());
04014     }
04015     GgQuaternion operator+(const GLfloat* a) const
```

```
04016  {
04017      return add(a);
04018  }
04019  GgQuaternion operator+(const GgVector& v) const
04020  {
04021      return add(v);
04022  }
04023  GgQuaternion operator+(const GgQuaternion& q) const
04024  {
04025      return operator+(q.data());
04026  }
04027  GgQuaternion operator-(const GLfloat* a) const
04028  {
04029      return add(a);
04030  }
04031  GgQuaternion operator-(const GgVector& v) const
04032  {
04033      return add(v);
04034  }
04035  GgQuaternion operator-(const GgQuaternion& q) const
04036  {
04037      return operator-(q.data());
04038  }
04039  GgQuaternion operator*(const GLfloat* a) const
04040  {
04041      return multiply(a);
04042  }
04043  GgQuaternion operator*(const GgVector& v) const
04044  {
04045      return multiply(v);
04046  }
04047  GgQuaternion operator*(const GgQuaternion& q) const
04048  {
04049      return operator*(q.data());
04050  }
04051  GgQuaternion operator/(const GLfloat* a) const
04052  {
04053      return divide(a);
04054  }
04055  GgQuaternion operator/(const GgVector& v) const
04056  {
04057      return divide(v);
04058  }
04059  GgQuaternion operator/(const GgQuaternion& q) const
04060  {
04061      return operator/(q.data());
04062  }
04063
04070  GgQuaternion& loadMatrix(const GLfloat* a)
04071  {
04072      toQuaternion(data(), a);
04073      return *this;
04074  }
04075
04082  GgQuaternion& loadMatrix(const GgMatrix& m)
04083  {
04084      return loadMatrix(m.get());
04085  }
04086
04092  GgQuaternion& loadIdentity()
04093  {
04094      return *this = GgQuaternion(0.0f, 0.0f, 0.0f, 1.0f);
04095  }
04096
04106  GgQuaternion& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
04107
04115  GgQuaternion& loadRotate(const GLfloat* v, GLfloat a)
04116  {
04117      return loadRotate(v[0], v[1], v[2], a);
04118  }
04119
04126  GgQuaternion& loadRotate(const GLfloat* v)
04127  {
04128      return loadRotate(v[0], v[1], v[2], v[3]);
04129  }
04130
04137  GgQuaternion& loadRotateX(GLfloat a);
04138
04145  GgQuaternion& loadRotateY(GLfloat a);
04146
04153  GgQuaternion& loadRotateZ(GLfloat a);
04154
04164  GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
04165  {
04166      GgQuaternion q;
04167      return multiply(q.loadRotate(x, y, z, a));
04168 }
```

```
04169
04177     GgQuaternion rotate(const GLfloat* v, GLfloat a) const
04178     {
04179         return rotate(v[0], v[1], v[2], a);
04180     }
04181
04188     GgQuaternion rotate(const GLfloat* v) const
04189     {
04190         return rotate(v[0], v[1], v[2], v[3]);
04191     }
04192
04199     GgQuaternion rotateX(GLfloat a) const
04200     {
04201         return rotate(1.0f, 0.0f, 0.0f, a);
04202     }
04203
04210     GgQuaternion rotateY(GLfloat a) const
04211     {
04212         return rotate(0.0f, 1.0f, 0.0f, a);
04213     }
04214
04221     GgQuaternion rotateZ(GLfloat a) const
04222     {
04223         return rotate(0.0f, 0.0f, 1.0f, a);
04224     }
04225
04234     GgQuaternion& loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll);
04235
04242     GgQuaternion& loadEuler(const GLfloat* e)
04243     {
04244         return loadEuler(e[0], e[1], e[2]);
04245     }
04246
04255     GgQuaternion euler(GLfloat heading, GLfloat pitch, GLfloat roll) const
04256     {
04257         GgQuaternion r;
04258         return multiply(r.loadEuler(heading, pitch, roll));
04259     }
04260
04267     GgQuaternion euler(const GLfloat* e) const
04268     {
04269         return euler(e[0], e[1], e[2]);
04270     }
04271
04281     GgQuaternion euler(const GgVector3& e) const
04282     {
04283         return euler(e[0], e[1], e[2]);
04284     }
04285
04294     GgQuaternion& loadSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04295     {
04296         slerp(data(), a, b, t);
04297         return *this;
04298     }
04299
04308     GgQuaternion& loadSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04309     {
04310         return loadSlerp(q.data(), r.data(), t);
04311     }
04312
04321     GgQuaternion& loadSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04322     {
04323         return loadSlerp(q.data(), a, t);
04324     }
04325
04334     GgQuaternion& loadSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04335     {
04336         return loadSlerp(a, q.data(), t);
04337     }
04338
04345     GgQuaternion& loadNormalize(const GLfloat* a);
04346
04353     GgQuaternion& loadNormalize(const GgQuaternion& q)
04354     {
04355         return loadNormalize(q.data());
04356     }
04357
04364     GgQuaternion& loadConjugate(const GLfloat* a);
04365
04372     GgQuaternion& loadConjugate(const GgQuaternion& q)
04373     {
04374         return loadConjugate(q.data());
04375     }
04376
04383     GgQuaternion& loadInvert(const GLfloat* a);
04384
04391     GgQuaternion& loadInvert(const GgQuaternion& q)
```

```
04392     {
04393         return loadInvert(q.data());
04394     }
04395
04403     GgQuaternion slerp(GLfloat* a, GLfloat t) const
04404     {
04405         GgQuaternion p;
04406         slerp(p.data(), data(), a, t);
04407         return p;
04408     }
04409
04417     GgQuaternion slerp(const GgQuaternion& q, GLfloat t) const
04418     {
04419         GgQuaternion p;
04420         slerp(p.data(), data(), q.data(), t);
04421         return p;
04422     }
04423
04429     GgQuaternion normalize() const
04430     {
04431         GgQuaternion q;
04432         q.loadNormalize(data());
04433         return q;
04434     }
04435
04441     GgQuaternion conjugate() const
04442     {
04443         GgQuaternion q;
04444         q.loadConjugate(data());
04445         return q;
04446     }
04447
04453     GgQuaternion invert() const
04454     {
04455         GgQuaternion q;
04456         q.loadInvert(data());
04457         return q;
04458     }
04459
04465     void get(GLfloat* a) const
04466     {
04467         a[0] = data()[0];
04468         a[1] = data()[1];
04469         a[2] = data()[2];
04470         a[3] = data()[3];
04471     }
04472
04478     void getMatrix(GLfloat* a) const
04479     {
04480         toMatrix(a, data());
04481     }
04482
04488     void getMatrix(GgMatrix& m) const
04489     {
04490         getMatrix(m.data());
04491     }
04492
04498     GgMatrix getMatrix() const
04499     {
04500         GgMatrix m;
04501         getMatrix(m);
04502         return m;
04503     }
04504
04510     void getConjugateMatrix(GLfloat* a) const
04511     {
04512         GgQuaternion c;
04513         c.loadConjugate(data());
04514         toMatrix(a, c.data());
04515     }
04516
04522     void getConjugateMatrix(GgMatrix& m) const
04523     {
04524         getConjugateMatrix(m.data());
04525     }
04526
04532     GgMatrix getConjugateMatrix() const
04533     {
04534         GgMatrix m;
04535         getConjugateMatrix(m);
04536         return m;
04537     }
04538
04539
04545     inline GgQuaternion ggIdentityQuaternion()
04546     {
04547         GgQuaternion q;
```

```
04548     return q.loadIdentity();
04549 }
04550
04551 inline GgQuaternion ggMatrixQuaternion(const GLfloat* a)
04552 {
04553     GgQuaternion q;
04554     return q.loadMatrix(a);
04555 }
04556
04557 inline GgQuaternion ggMatrixQuaternion(const GgMatrix& m)
04558 {
04559     return ggMatrixQuaternion(m.get());
04560 }
04561
04562
04563 inline GgMatrix ggQuaternionMatrix(const GgQuaternion& q)
04564 {
04565     GLfloat m[16];
04566     q.getMatrix(m);
04567     return GgMatrix{m};
04568 }
04569
04570 inline GgMatrix ggQuaternionTransposeMatrix(const GgQuaternion& q)
04571 {
04572     GLfloat m[16];
04573     q.getMatrix(m);
04574     GgMatrix t;
04575     return t.loadTranspose(m);
04576 }
04577
04578 inline GgQuaternion ggRotateQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
04579 {
04580     GgQuaternion q;
04581     return q.loadRotate(x, y, z, a);
04582 }
04583
04584
04585 inline GgQuaternion ggRotateQuaternion(const GLfloat* v, GLfloat a)
04586 {
04587     return ggRotateQuaternion(v[0], v[1], v[2], a);
04588 }
04589
04590
04591 inline GgQuaternion ggRotateQuaternion(const GLfloat* v)
04592 {
04593     return ggRotateQuaternion(v[0], v[1], v[2], v[3]);
04594 }
04595
04596
04597 inline GgQuaternion ggEulerQuaternion(GLfloat heading, GLfloat pitch, GLfloat roll)
04598 {
04599     GgQuaternion q;
04600     return q.loadEuler(heading, pitch, roll);
04601 }
04602
04603
04604 inline GgQuaternion ggEulerQuaternion(const GLfloat* e)
04605 {
04606     return ggEulerQuaternion(e[0], e[1], e[2]);
04607 }
04608
04609
04610 inline GgQuaternion ggEulerQuaternion(const GgVector& e)
04611 {
04612     return ggEulerQuaternion(e[0], e[1], e[2]);
04613 }
04614
04615
04616 inline GgQuaternion ggSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04617 {
04618     GgQuaternion r;
04619     return r.loadSlerp(a, b, t);
04620 }
04621
04622
04623 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04624 {
04625     return ggSlerp(q.data(), r.data(), t);
04626 }
04627
04628
04629 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04630 {
04631     return ggSlerp(q.data(), a, t);
04632 }
04633
04634
04635 inline GgQuaternion ggSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04636 {
04637     return ggSlerp(a, q.data(), t);
04638 }
04639
04640
04641 inline GLfloat ggNorm(const GgQuaternion& q)
04642 {
04643     return q.norm();
04644 }
04645
```

```
04745 inline GgQuaternion ggNormalize(const GgQuaternion& q)
04746 {
04747     return q.normalize();
04748 }
04749
04750 inline GgQuaternion ggConjugate(const GgQuaternion& q)
04751 {
04752     return q.conjugate();
04753 }
04754
04755 inline GgQuaternion ggInvert(const GgQuaternion& q)
04756 {
04757     return q.invert();
04758 }
04759
04760
04761 class GgTrackball : public GgQuaternion
04762 {
04763     bool drag;           // ドラッグ中か否か
04764     GLfloat start[2];   // ドラッグ開始位置
04765     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
04766     GgQuaternion cq;    // 回軸の初期値 (四元数)
04767     GgMatrix rt;        // 回軸の変換行列
04768
04769 public:
04770
04771     GgTrackball(const GgQuaternion& q = ggIdentityQuaternion())
04772     {
04773         reset(q);
04774     }
04775
04776     ~GgTrackball() = default;
04777
04778     GgTrackball& operator=(const GgQuaternion& q)
04779     {
04780         GgQuaternion::operator=(q);
04781         return *this;
04782     }
04783
04784     void region(GLfloat w, GLfloat h);
04785
04786     void region(int w, int h)
04787     {
04788         region(static_cast<GLfloat>(w), static_cast<GLfloat>(h));
04789     }
04790
04791     void begin(GLfloat x, GLfloat y);
04792
04793     void motion(GLfloat x, GLfloat y);
04794
04795     void rotate(const GgQuaternion& q);
04796
04797     void end(GLfloat x, GLfloat y);
04798
04799     void reset(const GgQuaternion& q = ggIdentityQuaternion());
04800
04801     const GLfloat* getStart() const
04802     {
04803         return start;
04804     }
04805
04806     const GLfloat& getStart(int direction) const
04807     {
04808         return start[direction];
04809     }
04810
04811     void getStart(GLfloat* position) const
04812     {
04813         position[0] = start[0];
04814         position[1] = start[1];
04815     }
04816
04817     const GLfloat* getScale() const
04818     {
04819         return scale;
04820     }
04821
04822     const GLfloat getScale(int direction) const
04823     {
04824         return scale[direction];
04825     }
04826
04827     void getScale(GLfloat* factor) const
04828     {
04829         factor[0] = scale[0];
04830         factor[1] = scale[1];
04831     }
04832
04833
04834
04835
04836
04837
04838
04839
04840
04841
04842
```

```
04948     const GgQuaternion& getQuaternion() const
04949     {
04950         return *this;
04951     }
04952
04953     const GgMatrix& getMatrix() const
04954     {
04955         return rt;
04956     }
04957
04958     const GLfloat* get() const
04959     {
04960         return rt.get();
04961     }
04962
04963 };
04964
04965 extern bool ggSaveTga(
04966     const std::string& name,
04967     const void* buffer,
04968     unsigned int width,
04969     unsigned int height,
04970     unsigned int depth
04971 );
04972
04973
04974 extern bool ggSaveColor(const std::string& name);
04975
04976 extern bool ggSaveDepth(const std::string& name);
04977
04978 extern bool ggReadImage(
04979     const std::string& name,
04980     std::vector<GLubyte>& image,
04981     GLsizei* pWidth,
04982     GLsizei* pHeight,
04983     GLenum* pFormat
04984 );
04985
04986
04987 extern GLuint ggLoadTexture(
04988     const GLvoid* image,
04989     GLsizei width,
04990     GLsizei height,
04991     GLenum format = GL_RGB,
04992     GLenum type = GL_UNSIGNED_BYTE,
04993     GLenum internal = GL_RGB,
04994     GLenum wrap = GL_CLAMP_TO_EDGE,
04995     bool swizzle = false
04996 );
04997
04998
04999 extern GLuint ggLoadImage(
05000     const std::string& name,
05001     GLsizei* pWidth = nullptr,
05002     GLsizei* pHeight = nullptr,
05003     GLenum internal = GL_RGBA,
05004     GLenum wrap = GL_CLAMP_TO_EDGE
05005 );
05006
05007
05008 extern void ggCreateNormalMap(
05009     const GLubyte* hmap,
05010     GLsizei width,
05011     GLsizei height,
05012     GLenum format,
05013     GLfloat nz,
05014     GLenum internal,
05015     std::vector<GgVector>& nmap
05016 );
05017
05018
05019 extern GLuint ggLoadHeight(
05020     const std::string& name,
05021     GLfloat nz,
05022     GLsizei* pWidth = nullptr,
05023     GLsizei* pHeight = nullptr,
05024     GLenum internal = GL_RGBA
05025 );
05026
05027
05028 extern GLuint ggCreateShader(
05029     const std::string& vsrc,
05030     const std::string& fsrc = "",
05031     const std::string& gsrc = "",
05032     GLint nvarying = 0,
05033     const char* const* varyings = nullptr,
05034     const std::string& vtext = "vertex shader",
05035     const std::string& ftext = "fragment shader",
05036     const std::string& gtext = "geometry shader");
05037
05038
05039 extern GLuint ggLoadShader(
05040     const std::string& vert,
05041     const std::string& frag = "",
05042     const std::string& geom = "",
```

```
05144     GLint nvarying = 0,
05145     const char* const* varyings = nullptr
05146 );
05147
05148 inline GLuint ggLoadShader(
05149     const std::array<std::string, 3>& files,
05150     GLint nvarying = 0,
05151     const char* const* varyings = nullptr
05152 )
05153 {
05154     return ggLoadShader(files[0], files[1], files[2], nvarying, varyings);
05155 }
05156 #if !defined(__APPLE__)
05157 extern GLuint ggCreateComputeShader(
05158     const std::string& csrc,
05159     const std::string& ctext = "compute shader"
05160 );
05161
05162 extern GLuint ggLoadComputeShader(const std::string& comp);
05163 #endif
05164
05165 class GgTexture
05166 {
05167     // テクスチャ名
05168     GLuint texture;
05169
05170     // テクスチャの縦横の画素数
05171     GLsizei size[2];
05172
05173 public:
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 = false
05183     ) :
05184         texture{ ggLoadTexture(image, width, height, format, type, internal, wrap, swizzle) },
05185         size{ width, height }
05186     {
05187     }
05188
05189     GgTexture(const GgTexture& texture) = delete;
05190
05191     GgTexture(GgTexture&& texture) = default;
05192
05193     virtual ~GgTexture()
05194     {
05195         glBindTexture(GL_TEXTURE_2D, 0);
05196         glDeleteTextures(1, &texture);
05197     }
05198
05199     GgTexture& operator=(const GgTexture& texture) = delete;
05200
05201     GgTexture& operator=(GgTexture&& texture) = default;
05202
05203     void bind() const
05204     {
05205         glBindTexture(GL_TEXTURE_2D, texture);
05206     }
05207
05208     void unbind() const
05209     {
05210         glBindTexture(GL_TEXTURE_2D, 0);
05211     }
05212
05213     void swapRandB(bool swizzle) const;
05214
05215     const GLsizei& getWidth() const
05216     {
05217         return size[0];
05218     }
05219
05220     const GLsizei& getHeight() const
05221     {
05222         return size[1];
05223     }
05224
05225     void getSize(GLsizei* size) const
05226     {
05227         size[0] = getWidth();
05228         size[1] = getHeight();
05229     }
05230
05231
05232
05233
05234
05235
05236
05237
05238
05239
05240
05241
05242
05243
05244
05245
05246
05247
05248
05249
05250
05251
05252
05253
05254
05255
05256
05257
05258
05259
05260
05261
05262
05263
05264
05265
05266
05267
05268
05269
05270
05271
05272
05273
05274
05275
05276
05277
05278
05279
05280
05281
05282
05283
05284
05285
05286
05287
05288
05289
05290
05291
05292
05293
05294
05295
05296
05297
05298
05299
05300
05301
05302
05303
05304
05305
05306
05307
05308
05309
05310
05311
05312
05313
05314
05315
05316
05317
05318
05319
05320
```

```
05321     }
05322
05323     const GLsizei* getSize() const
05324     {
05325         return size;
05326     }
05327
05328     const GLuint& getTexture() const
05329     {
05330         return texture;
05331     }
05332
05333 };
05334
05335 class GgColorTexture
05336 {
05337     // テクスチャ
05338     std::shared_ptr<GgTexture> texture;
05339
05340 public:
05341     GgColorTexture()
05342     {
05343     }
05344
05345     GgColorTexture(
05346         const GLvoid* image,
05347         GLsizei width,
05348         GLsizei height,
05349         GLenum format = GL_RGB,
05350         GLenum type = GL_UNSIGNED_BYTE,
05351         GLenum internal = GL_RGB,
05352         GLenum wrap = GL_CLAMP_TO_EDGE,
05353         bool swizzle = true
05354     )
05355     {
05356         load(image, width, height, format, type, internal, wrap, swizzle);
05357     }
05358
05359     GgColorTexture(
05360         const std::string& name,
05361         GLenum internal = 0,
05362         GLenum wrap = GL_CLAMP_TO_EDGE
05363     )
05364     {
05365         load(name, internal, wrap);
05366     }
05367
05368     virtual ~GgColorTexture()
05369     {
05370     }
05371
05372     void load(
05373         const GLvoid* image,
05374         GLsizei width,
05375         GLsizei height,
05376         GLenum format = GL_RGB,
05377         GLenum type = GL_UNSIGNED_BYTE,
05378         GLenum internal = GL_RGB,
05379         GLenum wrap = GL_CLAMP_TO_EDGE,
05380         bool swizzle = true
05381     )
05382     {
05383         // テクスチャを作成する
05384         texture = std::make_shared<GgTexture>(image, width, height, format, type, internal, wrap,
05385         swizzle);
05386     }
05387
05388     void load(
05389         const std::string& name,
05390         GLenum internal = 0,
05391         GLenum wrap = GL_CLAMP_TO_EDGE
05392     );
05393
05394     class GgNormalTexture
05395     {
05396         // テクスチャ
05397         std::shared_ptr<GgTexture> texture;
05398
05399 public:
05400     GgNormalTexture()
05401     {
05402     }
05403
05404     GgNormalTexture(
05405         const GLubyte* image,
```

```
05486     GLsizei width,
05487     GLsizei height,
05488     GLenum format = GL_RED,
05489     GLfloat nz = 1.0f,
05490     GLenum internal = GL_RGBA
05491 }
05492 {
05493 // 法線マップのテクスチャを作成する
05494 load(image, width, height, format, nz, internal);
05495 }
05496
05504 GgNormalTexture(
05505     const std::string& name,
05506     GLfloat nz = 1.0f,
05507     GLenum internal = GL_RGBA
05508 )
05509 {
05510 // 法線マップのテクスチャを作成する
05511 load(name, nz, internal);
05512 }
05513
05517 virtual ~GgNormalTexture()
05518 {
05519 }
05520
05531 void load(
05532     const GLubyte* hmap,
05533     GLsizei width,
05534     GLsizei height,
05535     GLenum format = GL_RED,
05536     GLfloat nz = 1.0f,
05537     GLenum internal = GL_RGBA
05538 )
05539 {
05540 // 法線マップ
05541     std::vector<GgVector> nmap;
05542
05543 // 法線マップを作成する
05544     ggCreateNormalMap(hmap, width, height, format, nz, internal, nmap);
05545
05546 // テクスチャを作成する
05547     texture = std::make_shared<GgTexture>(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal,
05548     GL_REPEAT);
05549 }
05550
05557 void load(
05558     const std::string& name,
05559     GLfloat nz = 1.0f,
05560     GLenum internal = GL_RGBA
05561 );
05562 };
05563
05570 template <typename T>
05571 class GgBuffer
05572 {
05573 // ターゲット
05574     const GLenum target;
05575
05576 // 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
05577     const GLsizei stride;
05578
05579 // データの数
05580     const GLsizei count;
05581
05582 // バッファオブジェクト
05583     const GLuint buffer;
05584
05585 public:
05586
05596     GgBuffer<T>(
05597         GLenum target,
05598         const T* data,
05599         GLsizei stride,
05600         GLsizei count,
05601         GLenum usage
05602     ) :
05603         target{ target },
05604         stride{ stride },
05605         count{ count },
05606         buffer{ [] { GLuint buffer; glGenBuffers(1, &buffer); return buffer; } () }
05607     {
05608 // 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
05609         glBindBuffer(target, buffer);
05610         glBufferData(target, getStride() * count, data, usage);
05611     }
05612
05618     GgBuffer<T>(const GgBuffer<T>& buffer) = delete;
```

```

05619 GgBuffer<T>(GgBuffer<T>&& buffer) = default;
05625
05626
05630 virtual ~GgBuffer<T>()
05631 {
05632     // バッファオブジェクトを削除する
05633     glBindBuffer(target, 0);
05634     glDeleteBuffers(1, &buffer);
05635 }
05636
05643 GgBuffer<T>& operator=(const GgBuffer<T>& buffer) = delete;
05644
05651 GgBuffer<T>& operator=(GgBuffer<T>&& buffer) = default;
05652
05658 const GLuint& getTarget() const
05659 {
05660     return target;
05661 }
05662
05668 GLsizeiptr getStride() const
05669 {
05670     return static_cast<GLsizeiptr>(stride);
05671 }
05672
05678 const GLsizei& getCount() const
05679 {
05680     return count;
05681 }
05682
05688 const GLuint& getBuffer() const
05689 {
05690     return buffer;
05691 }
05692
05696 void bind() const
05697 {
05698     glBindBuffer(target, buffer);
05699 }
05700
05704 void unbind() const
05705 {
05706     glBindBuffer(target, 0);
05707 }
05708
05714 void* map() const
05715 {
05716     glBindBuffer(target, buffer);
05717 #if defined(GL_GLES_PROTOTYPES)
05718     return glMapBufferRange(target, 0, getStride() * count, GL_MAP_WRITE_BIT);
05719 #else
05720     return glMapBuffer(target, GL_WRITE_ONLY);
05721 #endif
05722 }
05723
05731 void* map(GLint first, GLsizei count) const
05732 {
05733     // count が 0 なら全データをマップする
05734     if (count == 0) count = getCount();
05735     if (first + count > getCount()) count = getCount() - first;
05736
05737     glBindBuffer(target, buffer);
05738     return glMapBufferRange(target, getStride() * first, getStride() * count, GL_MAP_WRITE_BIT);
05739 }
05740
05744 void unmap() const
05745 {
05746     glUnmapBuffer(target);
05747 }
05748
05756 void send(const T* data, GLint first, GLsizei count) const
05757 {
05758     // count が 0 なら全データを転送する
05759     if (count == 0) count = getCount();
05760     if (first + count > getCount()) count = getCount() - first;
05761
05762     // データを既存のバッファオブジェクトに転送する
05763     glBindBuffer(target, buffer);
05764     glBufferSubData(target, getStride() * first, getStride() * count, data);
05765 }
05766
05774 void read(T* data, GLint first, GLsizei count) const
05775 {
05776     // count が 0 なら全データを抽出する
05777     if (count == 0) count = getCount();
05778     if (first + count > getCount()) count = getCount() - first;
05779
05780     // データをバッファオブジェクトから抽出する

```

```
05781     glBindBuffer(target, buffer);
05782 #if defined(GL_GLES_PROTOTYPES)
05783     const GLsizeiptr begin{ getStride() * first };
05784     const GLsizeiptr range{ getStride() * count };
05785     T* const source{ glMapBufferRange(target, begin, range, GL_MAP_READ_BIT) };
05786     std::copy(source, source + count, data);
05787     glUnmapBuffer(target);
05788 #else
05789     glGetBufferSubData(target, getStride() * first, getStride() * count, data);
05790 #endif
05791 }
05792
05801 void copy(GLuint src_buffer, GLint src_first = 0, GLint dst_first = 0, GLsizei count = 0) const
05802 {
05803     // count が 0 なら全データを複写する
05804     if (count == 0) count = getCount();
05805     if (src_first + count > getCount()) count = getCount() - src_first;
05806     if (dst_first + count > getCount()) count = getCount() - dst.first;
05807
05808     // データの間隔
05809     const GLsizeiptr stride{ getStride() };
05810
05811     glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
05812     glBindBuffer(GL_COPY_WRITE_BUFFER, buffer);
05813     glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
05814         stride * src_first, stride * dst_first, stride * count);
05815     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
05816     glBindBuffer(GL_COPY_READ_BUFFER, 0);
05817 }
05818 };
05819
05826 template <typename T>
05827 class GgUniformBuffer
05828 {
05829     // ユニフォームバッファオブジェクト
05830     std::shared_ptr<GgBuffer<T>> uniform;
05831
05832 public:
05833
05837     GgUniformBuffer<T>()
05838     {
05839     }
05840
05848     GgUniformBuffer<T>(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05849     {
05850         load(data, count, usage);
05851     }
05852
05860     GgUniformBuffer<T>(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05861     {
05862         load(data, count, usage);
05863     }
05864
05868     virtual ~GgUniformBuffer<T>()
05869     {
05870     }
05871
05877     const GLuint& getTarget() const
05878     {
05879         return uniform->getTarget();
05880     }
05881
05887     GLsizeiptr getStride() const
05888     {
05889         return uniform->getStride();
05890     }
05891
05897     const GLsizei& getCount() const
05898     {
05899         return uniform->getCount();
05900     }
05901
05907     const GLuint& getBuffer() const
05908     {
05909         return uniform->getBuffer();
05910     }
05911
05915     void bind() const
05916     {
05917         uniform->bind();
05918     }
05919
05923     void unbind() const
05924     {
05925         uniform->unbind();
05926     }
05927
```

```

05933     void* map() const
05934     {
05935         return uniform->map();
05936     }
05937
05945     void* map(GLint first, GLsizei count) const
05946     {
05947         return uniform->map(first, count);
05948     }
05949
05953     void unmap() const
05954     {
05955         uniform->unmap();
05956     }
05957
05965     void load(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05966     {
05967         // バッファオブジェクト上のデータの間隔
05968         const GLsizei stride{ ((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1 } *
05969             ggBufferAlignment };
05970
05971         // ユニフォームバッファオブジェクトを確保する
05972         uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05973
05974         // 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
05975         if (data) send(data, 0, sizeof(T), 0, count);
05976
05978     void load(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05979     {
05980         // バッファオブジェクト上のデータの間隔
05981         const GLsizei stride{ ((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1 } *
05982             ggBufferAlignment };
05983
05984         // ユニフォームバッファオブジェクトを確保する
05985         uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05986
05987         // 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
05988         fill(&data, 0, sizeof(T), 0, count);
05989
05990         // ターゲット
05991         const GLuint target{ getTarget() };
05992
05993         // データの間隔
05994         const GLsizeiptr stride{ getStride() };
05995
06005     void send(
06006         const GLvoid* data,
06007         GLint offset = 0,
06008         GLsizei size = sizeof(T),
06009         GLint first = 0,
06010         GLsizei count = 0
06011     ) const
06012     {
06013         // count が 0 なら全データを転送する
06014         if (count == 0) count = getCount();
06015         if (first + count > getCount()) count = getCount() - first;
06016
06017         // 転送元のデータの先頭
06018         const char* source{ reinterpret_cast<const char*>(data) };
06019
06020         // ターゲット
06021         const GLuint target{ getTarget() };
06022
06023         // データの間隔
06024         const GLsizeiptr stride{ getStride() };
06025
06026         // 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
06027         bind();
06028         for (GLsizei i = 0; i < count; ++i)
06029         {
06030             glBufferSubData(target, stride * (first + i) + offset, size, source + size * i);
06031         }
06032
06033
06043     void fill(
06044         const GLvoid* data,
06045         GLint offset = 0,
06046         GLsizei size = sizeof(T),
06047         GLint first = 0,
06048         GLsizei count = 0
06049     ) const
06050     {
06051         // count が 0 なら全データを転送する
06052         if (count == 0) count = getCount();
06053         if (first + count > getCount()) count = getCount() - first;
06054
06055         // ターゲット
06056         const GLuint target{ getTarget() };
06057
06058         // データの間隔
06059         const GLsizeiptr stride{ getStride() };

```

```
06060 // 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
06061 bind();
06062 for (GLsizei i = 0; i < count; ++i)
06063 {
06064     glBufferSubData(target, stride * (first + i) + offset, size, data);
06065 }
06066 }
06067 }

06078 void read(
06079     GLvoid* data,
06080     GLint offset = 0,
06081     GLsizei size = sizeof(T),
06082     GLint first = 0,
06083     GLsizei count = 0
06084 ) const
06085 {
06086     // count が 0 なら全データを転送する
06087     if (count == 0) count = getCount();
06088     if (first + count > getCount()) count = getCount() - first;
06089
06090     // 抽出先のデータの先頭
06091     char* const destination{ reinterpret_cast<char*>(data) };
06092
06093     // ターゲット
06094     const GLuint target{ getTarget() };
06095
06096     // データの間隔
06097     const GLsizeiptr stride{ getStride() };
06098
06099     // 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
06100     bind();
06101 #if defined(GL_GLES_PROTOTYPES)
06102     const char* const source{ glMapBufferRange(target, stride * first, stride * count,
06103     GL_MAP_READ_BIT) };
06104     for (GLsizei i = 0; i < count; ++i)
06105     {
06106         const char* const begin{ source + stride * i + offset };
06107         const char* const end{ begin + size };
06108         std::copy(begin, end, destination + sizeof(T) * i);
06109     }
06110     glUnmapBuffer(target);
06111 #else
06112     for (GLsizei i = 0; i < count; ++i)
06113     {
06114         glGetBufferSubData(target, stride * (first + i) + offset, size, destination + sizeof(T) * i);
06115     }
06116 #endif
06117 }

06126 void copy(
06127     GLuint src_buffer,
06128     GLint src_first = 0,
06129     GLint dst_first = 0,
06130     GLsizei count = 0
06131 ) const
06132 {
06133     // count が 0 なら全データを複写する
06134     if (count == 0) count = getCount();
06135     if (src.first + count > getCount()) count = getCount() - src.first;
06136     if (dst.first + count > getCount()) count = getCount() - dst.first;
06137
06138     // データの間隔
06139     const GLsizeiptr stride{ getStride() };
06140
06141     // 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
06142     glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
06143     glBindBuffer(GL_COPY_WRITE_BUFFER, getBuffer());
06144     for (GLsizei i = 0; i < count; ++i)
06145     {
06146         glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
06147             stride * (src.first + i), stride * (dst.first + i), sizeof(T));
06148     }
06149     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
06150     glBindBuffer(GL_COPY_READ_BUFFER, 0);
06151 }
06152 };

06157 class GgVertexArray
06158 {
06159     // 頂点配列オブジェクト
06160     const GLuint vao;
06161
06162 public:
06163     GgVertexArray(GLenum mode = 0) :
06164         vao{ [] { GLuint vao; glGenVertexArrays(1, &vao); return vao; } () }
06165 }
```

```

06171     {
06172         glBindVertexArray(vao);
06173     }
06174
06175     GgVertexArray(const GgVertexArray& array) = delete;
06176
06177     GgVertexArray(GgVertexArray&& array) = default;
06178
06179     virtual ~GgVertexArray()
06180     {
06181         glBindVertexArray(0);
06182         glDeleteVertexArrays(1, &vao);
06183     }
06184
06185     GgVertexArray& operator=(const GgVertexArray& array) = delete;
06186
06187     GgVertexArray& operator=(GgVertexArray&& array) = default;
06188
06189     const GLuint& get() const
06190     {
06191         return vao;
06192     }
06193
06194     void bind() const
06195     {
06196         glBindVertexArray(vao);
06197     }
06198
06199     class GgShape
06200     {
06201         // 頂点配列オブジェクト
06202         std::shared_ptr<GgVertexArray> object;
06203
06204         // 基本図形の種類
06205         GLenum mode;
06206
06207         public:
06208
06209         GgShape(GLenum mode = 0) :
06210             object{ std::make_shared<GgVertexArray>() },
06211             mode{ mode }
06212         {
06213         }
06214
06215         virtual ~GgShape()
06216         {
06217         }
06218
06219         virtual explicit operator bool() const noexcept
06220         {
06221             return object.get() != nullptr;
06222         }
06223
06224         virtual bool operator!() const noexcept
06225         {
06226             return !static_cast<bool>(*this);
06227         }
06228
06229         const GLuint& get() const
06230         {
06231             return object->get();
06232         }
06233
06234         void setMode(GLenum mode)
06235         {
06236             this->mode = mode;
06237         }
06238
06239         const GLenum& getMode() const
06240         {
06241             return this->mode;
06242         }
06243
06244         virtual void draw(GLint first = 0, GLsizei count = 0) const
06245         {
06246             object->bind();
06247         }
06248     };
06249
06250     class GgPoints
06251         : public GgShape
06252     {
06253         // 頂点バッファオブジェクト
06254         std::shared_ptr<GgBuffer<GgVector>> position;
06255
06256         public:

```

```
06340
06344     GgPoints(GLenum mode = GL_POINTS) :
06345         GgShape(mode)
06346     {
06347     }
06348
06357     GgPoints(
06358         const GgVector* pos,
06359         GLsizei countv,
06360         GLenum mode = GL_POINTS,
06361         GLenum usage = GL_STATIC_DRAW
06362     ) :
06363         GgPoints(mode)
06364     {
06365         load(pos, countv, usage);
06366     }
06367
06371     virtual ~GgPoints()
06372     {
06373     }
06374
06380     explicit operator bool() const noexcept
06381     {
06382         return position.get() != nullptr;
06383     }
06384
06390     bool operator!() const noexcept
06391     {
06392         return !static_cast<bool>(*this);
06393     }
06394
06400     const GLsizei& getCount() const
06401     {
06402         return position->getCount();
06403     }
06404
06410     const GLuint& getBuffer() const
06411     {
06412         return position->getBuffer();
06413     }
06414
06422     void send(const GgVector* pos, GLint first = 0, GLsizei count = 0) const
06423     {
06424         position->send(pos, first, count);
06425     }
06426
06434     void load(const GgVector* pos, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06435
06442     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06443 };
06444
06448 struct GgVertex
06449 {
06451     GgVector position;
06452
06454     GgVector normal;
06455
06459     GgVertex()
06460     {
06461     }
06462
06469     GgVertex(const GgVector& pos, const GgVector& norm) :
06470         position(pos),
06471         normal(norm)
06472     {
06473     }
06474
06485     GgVertex(
06486         GLfloat px, GLfloat py, GLfloat pz,
06487         GLfloat nx, GLfloat ny, GLfloat nz
06488     ) :
06489         position{ px, py, pz, 1.0f },
06490         normal{ nx, ny, nz, 0.0f }
06491     {
06492     }
06493
06500     GgVertex(const GLfloat* pos, const GLfloat* norm) :
06501         GgVertex(pos[0], pos[1], pos[2], norm[0], norm[1], norm[2])
06502     {
06503     }
06504 };
06505
06509 class GgTriangles
06510     : public GgShape
06511 {
06512     // 頂点属性
06513     std::shared_ptr<GgBuffer<GgVertex>> vertex;
```

```
06514
06515     public:
06516
06522     GgTriangles(GLenum mode = GL_TRIANGLES) :
06523         GgShape(mode)
06524     {
06525     }
06526
06535     GgTriangles(
06536         const GgVertex* vert,
06537         GLsizei count,
06538         GLenum mode = GL_TRIANGLES,
06539         GLenum usage = GL_STATIC_DRAW
06540     ) :
06541         GgTriangles(mode)
06542     {
06543         load(vert, count, usage);
06544     }
06545
06549     virtual ~GgTriangles()
06550     {
06551     }
06552
06558     const GLsizei& getCount() const
06559     {
06560         return vertex->getCount();
06561     }
06562
06568     const GLuint& getBuffer() const
06569     {
06570         return vertex->getBuffer();
06571     }
06572
06580     void send(const GgVertex* vert, GLint first = 0, GLsizei count = 0) const
06581     {
06582         vertex->send(vert, first, count);
06583     }
06584
06592     void load(const GgVertex* vert, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06593
06600     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06601 };
06602
06606 class GgElements
06607     : public GgTriangles
06608 {
06609     // インデックスを格納する頂点バッファオブジェクト
06610     std::shared_ptr<GgBuffer<GLuint>> index;
06611
06612     public:
06613
06619     GgElements(GLenum mode = GL_TRIANGLES) :
06620         GgTriangles(mode)
06621     {
06622     }
06623
06634     GgElements(
06635         const GgVertex* vert,
06636         GLsizei countv,
06637         const GLuint* face,
06638         GLsizei countf,
06639         GLenum mode = GL_TRIANGLES,
06640         GLenum usage = GL_STATIC_DRAW
06641     ) :
06642         GgElements(mode)
06643     {
06644         load(vert, countv, face, countf, usage);
06645     }
06646
06650     virtual ~GgElements()
06651     {
06652     }
06653
06658     const GLsizei& getIndexCount() const
06659     {
06660         return index->getCount();
06661     }
06662
06668     const GLuint& getIndexBuffer() const
06669     {
06670         return index->getBuffer();
06671     }
06672
06683     void send(
06684         const GgVertex* vert,
06685         GLuint firstv,
06686         GLsizei countv,
```

```
06687     const GLuint* face = nullptr,
06688     GLuint firstf = 0,
06689     GLsizei countf = 0
06690 } const
06691 {
06692     GgTriangles::send(vert, firstv, countv);
06693     if (face != nullptr && countf > 0) index->send(face, firstf, countf);
06694 }
06695
06705 void load(
06706     const GgVertex* vert,
06707     GLsizei countv,
06708     const GLuint* face,
06709     GLsizei countf,
06710     GLenum usage = GL_STATIC_DRAW
06711 )
06712 {
06713     // 頂点バッファオブジェクトを作成する
06714     GgTriangles::load(vert, countv, usage);
06715
06716     // インデックスの頂点バッファオブジェクトを作成する
06717     index = std::make_shared<GgBuffer<GLuint>>(GL_ELEMENT_ARRAY_BUFFER, face,
06718         static_cast<GLsizei>(sizeof(GLuint)), countf, usage);
06719     }
06720
06721     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06722 };
06723
06724 extern std::shared_ptr<GgPoints> ggPointsCube(
06725     GLsizei countv,
06726     GLfloat length = 1.0f,
06727     GLfloat cx = 0.0f,
06728     GLfloat cy = 0.0f,
06729     GLfloat cz = 0.0f
06730 );
06731
06732 extern std::shared_ptr<GgPoints> ggPointsSphere(
06733     GLsizei countv,
06734     GLfloat radius = 0.5f,
06735     GLfloat cx = 0.0f,
06736     GLfloat cy = 0.0f,
06737     GLfloat cz = 0.0f
06738 );
06739
06740 extern std::shared_ptr<GgTriangles> ggRectangle(
06741     GLfloat width = 1.0f,
06742     GLfloat height = 1.0f
06743 );
06744
06745 extern std::shared_ptr<GgTriangles> ggEllipse(
06746     GLfloat width = 1.0f,
06747     GLfloat height = 1.0f,
06748     GLuint slices = 16
06749 );
06750
06751 extern std::shared_ptr<GgTriangles> ggArraysObj(
06752     const std::string& name,
06753     bool normalize = false
06754 );
06755
06756 extern std::shared_ptr<GgElements> ggElementsObj(
06757     const std::string& name,
06758     bool normalize = false
06759 );
06760
06761 extern std::shared_ptr<GgElements> ggElementsMesh(
06762     GLuint slices,
06763     GLuint stacks,
06764     const GLfloat(*pos)[3],
06765     const GLfloat(*norm)[3] = nullptr
06766 );
06767
06768 extern std::shared_ptr<GgElements> ggElementsSphere(
06769     GLfloat radius = 1.0f,
06770     int slices = 16,
06771     int stacks = 8
06772 );
06773
06774 class GgShader
06775 {
06776     // プログラム名
06777     const GLuint program;
06778
06779 public:
06780     GgShader(
06781         const std::string& vert,
```

```
06882     const std::string& frag = "",
06883     const std::string& geom = "",
06884     int nvarying = 0,
06885     const char* const* varyings = nullptr
06886 ) :
06887     program(ggLoadShader(vert, frag, geom, nvarying, varyings))
06888 {
06889 }
06890
06891 GgShader(
06892     const std::array<std::string, 3>& files,
06893     int nvarying = 0,
06894     const char* const* varyings = nullptr
06895 ) :
06896     GgShader(files[0], files[1], files[2], nvarying, varyings)
06897 {
06898 }
06899
06900     GgShader(const GgShader& shader) = delete;
06901
06902     GgShader(GgShader&& shader) = default;
06903
06904     virtual ~GgShader()
06905 {
06906     // 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
06907     glUseProgram(0);
06908     glDeleteProgram(program);
06909 }
06910
06911     GgShader& operator=(const GgShader& shader) = delete;
06912
06913     GgShader& operator=(GgShader&& shader) = default;
06914
06915     void use() const
06916     {
06917         glUseProgram(program);
06918     }
06919
06920     void unuse() const
06921     {
06922         glUseProgram(0);
06923     }
06924
06925     GLuint get() const
06926     {
06927         return program;
06928     }
06929 };
06930
06931 class GgPointShader
06932 {
06933     // シェーダー
06934     std::shared_ptr<GgShader> shader;
06935
06936     // 投影変換行列の uniform 変数の場所
06937     GLint mpLoc;
06938
06939     // モデルビュー変換行列の uniform 変数の場所
06940     GLint mvLoc;
06941
06942 public:
06943     GgPointShader() :
06944         mpLoc{ -1 },
06945         mvLoc{ -1 }
06946     {
06947     }
06948
06949     GgPointShader(
06950         const std::string& vert,
06951         const std::string& frag = "",
06952         const std::string& geom = "",
06953         GLint nvarying = 0,
06954         const char* const* varyings = nullptr
06955 ) :
06956     GgPointShader()
06957     {
06958         load(vert, frag, geom, nvarying, varyings);
06959     }
06960
06961     GgPointShader(
06962         const std::array<std::string, 3>& files,
06963         int nvarying = 0,
06964         const char* const* varyings = nullptr
06965 ) :
06966     GgPointShader(files[0], files[1], files[2], nvarying, varyings)
06967     {
```

```
07034     }
07035
07039     virtual ~GgPointShader()
07040     {
07041     }
07042
07053     bool load(
07054         const std::string& vert,
07055         const std::string& frag = "",
07056         const std::string& geom = "",
07057         GLint nvarying = 0,
07058         const char* const* varyings = nullptr
07059     )
07060     {
07061         // シェーダを作成する
07062         shader = std::make_shared<GgShader>(vert, frag, geom, nvarying, varyings);
07063
07064         // プログラム名を取り出す
07065         const GLuint program(shader->get());
07066
07067         // プログラムオブジェクトが作成できていなければ戻る
07068         if (program == 0) return false;
07069
07070         // 変換行列の uniform 変数の場所
07071         mpLoc = glGetUniformLocation(program, "mp");
07072         mvLoc = glGetUniformLocation(program, "mv");
07073
07074         // プログラムオブジェクトの作成に成功した
07075         return true;
07076     }
07077
07086     bool load(
07087         const std::array<std::string, 3>& files,
07088         GLint nvarying = 0,
07089         const char* const* varyings = nullptr
07090     )
07091     {
07092         return load(files[0], files[1], files[2], nvarying, varyings);
07093     }
07100     virtual void loadProjectionMatrix(const GLfloat* mp) const
07101     {
07102         glUniformMatrix4fv(mpLoc, 1, GL_FALSE, mp);
07103     }
07104
07110     virtual void loadProjectionMatrix(const GgMatrix& mp) const
07111     {
07112         loadProjectionMatrix(mp.get());
07113     }
07114
07120     virtual void loadModelviewMatrix(const GLfloat* mv) const
07121     {
07122         glUniformMatrix4fv(mvLoc, 1, GL_FALSE, mv);
07123     }
07124
07130     virtual void loadModelviewMatrix(const GgMatrix& mv) const
07131     {
07132         loadModelviewMatrix(mv.get());
07133     }
07134
07141     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07142     {
07143         loadProjectionMatrix(mp);
07144         loadModelviewMatrix(mv);
07145     }
07146
07153     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
07154     {
07155         loadMatrix(mp.get(), mv.get());
07156     }
07157
07161     virtual void use() const
07162     {
07163         shader->use();
07164     }
07165
07171     void use(const GLfloat* mp) const
07172     {
07173         use();
07174         loadProjectionMatrix(mp);
07175     }
07176
07182     void use(const GgMatrix& mp) const
07183     {
07184         use(mp.get());
07185     }
07186
```

```

07193 void use(const GLfloat* mp, const GLfloat* mv) const
07194 {
07195     use(mp);
07196     loadModelviewMatrix(mv);
07197 }
07198
07199 void use(const GgMatrix& mp, const GgMatrix& mv) const
07200 {
07201     use(mp.get(), mv.get());
07202 }
07203
07204 void unuse() const
07205 {
07206     shader->unuse();
07207 }
07208
07209 GLuint get() const
07210 {
07211     return shader->get();
07212 }
07213
07214 class GgSimpleShader
07215     : public GgPointShader
07216 {
07217     // モデルビュー変換の法線変換行列の uniform 変数の場所
07218     GLint mnLoc;
07219
07220 public:
07221
07222     GgSimpleShader() :
07223         GgPointShader(),
07224         mnLoc{ -1 }
07225     {
07226     }
07227
07228     GgSimpleShader(
07229         const std::string& vert,
07230         const std::string& frag = "",
07231         const std::string& geom = "",
07232         GLint nvarying = 0,
07233         const char* const* varyings = nullptr
07234     )
07235     {
07236         load(vert, frag, geom, nvarying, varyings);
07237     }
07238
07239     GgSimpleShader(
07240         const std::array<std::string, 3>& files,
07241         GLint nvarying = 0,
07242         const char* const* varyings = nullptr
07243     ) :
07244         GgSimpleShader(files[0], files[1], files[2], nvarying, varyings)
07245     {
07246     }
07247
07248     GgSimpleShader(const GgSimpleShader& o) :
07249         GgPointShader(o),
07250         mnLoc{ o.mnLoc }
07251     {
07252     }
07253
07254     virtual ~GgSimpleShader()
07255     {
07256     }
07257
07258     GgSimpleShader& operator=(const GgSimpleShader& o)
07259     {
07260         if (&o != this)
07261         {
07262             GgPointShader::operator=(o);
07263             mnLoc = o.mnLoc;
07264         }
07265
07266         return *this;
07267     }
07268
07269     bool load(
07270         const std::string& vert,
07271         const std::string& frag = "",
07272         const std::string& geom = "",
07273         GLint nvarying = 0,
07274         const char* const* varyings = nullptr
07275     );
07276
07277     bool load(
07278         const std::array<std::string, 3>& files,
07279

```

```
07346     GLint nvarying = 0,
07347     const char* const* varyings = nullptr
07348 }
07349 {
07350     return load(files[0], files[1], files[2], nvarying, varyings);
07351 }
07352
07353 virtual void loadModelviewMatrix(const GLfloat* mv, const GLfloat* mn) const
07354 {
07355     GgPointShader::loadModelviewMatrix(mv);
07356     glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07357 }
07358
07359 virtual void loadModelviewMatrix(const GgMatrix& mv, const GgMatrix& mn) const
07360 {
07361     loadModelviewMatrix(mv.get(), mn.get());
07362 }
07363
07364 virtual void loadModelviewMatrix(const GLfloat* mv) const
07365 {
07366     loadModelviewMatrix(mv, GgMatrix(mv).normal().get());
07367 }
07368
07369 virtual void loadModelviewMatrix(const GgMatrix& mv) const
07370 {
07371     loadModelviewMatrix(mv.get());
07372 }
07373
07374 virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07375 {
07376     GgPointShader::loadMatrix(mp, mv);
07377     glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07378 }
07379
07380 virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07381 {
07382     loadMatrix(mp.get(), mv.get(), mn.get());
07383 }
07384
07385 virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07386 {
07387     loadMatrix(mp, mv, GgMatrix(mv).normal());
07388 }
07389
07390 virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
07391 {
07392     loadMatrix(mp, mv, mv.normal());
07393 }
07394
07395 struct Light
07396 {
07397     GgVector ambient;
07398     GgVector diffuse;
07399     GgVector specular;
07400     GgVector position;
07401 };
07402
07403 class LightBuffer
07404     : public GgUniformBuffer<Light>
07405 {
07406     public:
07407
07408     LightBuffer(
07409         const Light* light = nullptr,
07410         GLsizei count = 1,
07411         GLenum usage = GL_STATIC_DRAW
07412     ) :
07413         GgUniformBuffer<Light>(light, count, usage)
07414     {
07415     }
07416
07417     LightBuffer(
07418         const Light& light,
07419         GLsizei count = 1,
07420         GLenum usage = GL_STATIC_DRAW
07421     ) :
07422         GgUniformBuffer<Light>(light, count, usage)
07423     {
07424     }
07425
07426     LightBuffer(
07427         GgVector ambient,
07428         GgVector diffuse,
07429         GgVector specular,
07430         GgVector position,
07431         GLsizei count = 1,
07432         GLenum usage = GL_STATIC_DRAW
07433     )
07434 }
```

```

07511     ) :
07512         GgUniformBuffer<Light>(Light{ ambient, diffuse, specular, position }, count, usage)
07513     {
07514     }
07515
07519     virtual ~LightBuffer()
07520     {
07521     }
07522
07523     void loadAmbient(
07524         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07525         GLint first = 0, GLsizei count = 1
07526     ) const;
07527
07528     void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07529
07530     void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07531     {
07532         // first 番目のブロックから count 個の ambient 要素に値を設定する
07533         send(ambient, offsetof(Light, ambient), sizeof(Light::ambient), first, count);
07534     }
07535
07536     void loadDiffuse(
07537         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07538         GLint first = 0, GLsizei count = 1
07539     ) const;
07540
07541     void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07542
07543     void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07544     {
07545         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07546         send(diffuse, offsetof(Light, diffuse), sizeof(Light::diffuse), first, count);
07547     }
07548
07549     void loadSpecular(
07550         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07551         GLint first = 0, GLsizei count = 1
07552     ) const;
07553
07554     void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07555
07556     void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07557     {
07558         // first 番目のブロックから count 個の specular 要素に値を設定する
07559         send(specular, offsetof(Light, specular), sizeof(Light::specular), first, count);
07560     }
07561
07562     void loadColor(const Light& color, GLint first = 0, GLsizei count = 1) const;
07563
07564     void loadPosition(
07565         GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f,
07566         GLint first = 0, GLsizei count = 1
07567     ) const;
07568
07569     void loadPosition(const GgVector& position, GLint first = 0, GLsizei count = 1) const;
07570
07571     void loadPosition(const GLfloat* position, GLint first = 0, GLsizei count = 1) const
07572     {
07573         // first 番目のブロックから count 個の position 要素に値を設定する
07574         send(position, offsetof(Light, position), sizeof(Light::position), first, count);
07575     }
07576
07577     void loadPosition(const GgVector* position, GLint first = 0, GLsizei count = 1) const
07578     {
07579         loadPosition(position->data(), first, count);
07580     }
07581
07582     void load(const Light* light, GLint first = 0, GLsizei count = 1) const
07583     {
07584         send(light, 0, sizeof(Light), first, count);
07585     }
07586
07587     void load(const Light& light, GLint first = 0, GLsizei count = 1) const
07588     {
07589         load(&light, first, count);
07590     }
07591
07592     void select(GLint i = 0) const
07593     {
07594         // バッファオブジェクトの i 番目のブロックの位置
07595         const GLintptr offset(static_cast<GLintptr>(getStride()) * i);
07596         glBindBufferRange(getTarget(), LightBindingPoint, getBuffer(), offset, sizeof(Light));
07597     }
07598 };
07599
07600 struct Material

```

```
07733     {
07734         GgVector ambient;
07735         GgVector diffuse;
07736         GgVector specular;
07737         GLfloat shininess;
07738     };
07739
07740     class MaterialBuffer
07741         : public GgUniformBuffer<Material>
07742     {
07743     public:
07744
07745         MaterialBuffer(
07746             const Material* material = nullptr,
07747             GLsizei count = 1,
07748             GLenum usage = GL_STATIC_DRAW
07749         ) :
07750             GgUniformBuffer<Material>(material, count, usage)
07751         {
07752         }
07753
07754         MaterialBuffer(
07755             const Material& material,
07756             GLsizei count = 1,
07757             GLenum usage = GL_STATIC_DRAW
07758         ) :
07759             GgUniformBuffer<Material>(material, count, usage)
07760         {
07761         }
07762
07763
07764         MaterialBuffer(
07765             GgVector ambient,
07766             GgVector diffuse,
07767             GgVector specular,
07768             GLfloat shininess,
07769             GLsizei count = 1,
07770             GLenum usage = GL_STATIC_DRAW
07771         ) :
07772             GgUniformBuffer<Material>(Material{ ambient, diffuse, specular, shininess }, count, usage)
07773         {
07774         }
07775
07776         virtual ~MaterialBuffer()
07777     {
07778     }
07779
07780     void loadAmbient(
07781         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07782         GLint first = 0, GLsizei count = 1
07783     ) const;
07784
07785     void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07786
07787     void loadAmbient(GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07788     {
07789         // 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
07790         send(ambient, offsetof(Material, ambient), sizeof(Material::ambient), first, count);
07791     }
07792
07793     void loadDiffuse(
07794         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07795         GLint first = 0, GLsizei count = 1
07796     ) const;
07797
07798     void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07799
07800     void loadDiffuse(GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07801     {
07802         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07803         send(diffuse, offsetof(Material, diffuse), sizeof(Material::diffuse), first, count);
07804     }
07805
07806     void loadAmbientAndDiffuse(
07807         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07808         GLint first = 0, GLsizei count = 1
07809     ) const;
07810
07811     void loadAmbientAndDiffuse(const GgVector& color, GLint first = 0, GLsizei count = 1) const;
07812
07813     void loadAmbientAndDiffuse(GLfloat* color, GLint first = 0, GLsizei count = 1) const;
07814
07815     void loadSpecular(
07816         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07817         GLint first = 0, GLsizei count = 1
07818     ) const;
07819
07820     void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
```

```

07939
07940     void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07941     {
07942         // first 番目のブロックから count 個の specular 要素に値を設定する
07943         send(specular, offsetof(Material, specular), sizeof(Material::specular), first, count);
07944     }
07945
07946     void loadShininess(GLfloat shininess, GLint first = 0, GLsizei count = 1) const;
07947
07948     void loadShininess(const GLfloat* shininess, GLint first = 0, GLsizei count = 1) const;
07949
07950     void load(const Material* material, GLint first = 0, GLsizei count = 1) const
07951     {
07952         send(material, 0, sizeof(Material), first, count);
07953     }
07954
07955     void load(const Material& material, GLint first = 0, GLsizei count = 1) const
07956     {
07957         load(&material, first, count);
07958     }
07959
08000     void select(GLint i = 0) const
08001     {
08002         // バッファオブジェクトの i 番目のブロックの位置
08003         const GLintptr offset{ static_cast<GLintptr>(getStride()) * i };
08004         glBindBufferRange(getTarget(), MaterialBindingPoint, getBuffer(), offset, sizeof(Material));
08005     }
08006
08007
08011     void use() const
08012     {
08013         // プログラムオブジェクトは基底クラスで指定する
08014         GgPointShader::use();
08015     }
08016
08024     void use(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
08025     {
08026         // プログラムオブジェクトを指定する
08027         use();
08028
08029         // 変換行列を設定する
08030         loadMatrix(mp, mv, mn);
08031     }
08032
08040     void use(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
08041     {
08042         use(mp.get(), mv.get(), mn.get());
08043     }
08044
08051     void use(const GLfloat* mp, const GLfloat* mv) const
08052     {
08053         use(mp, mv, GgMatrix(mv).normal().get());
08054     }
08055
08062     void use(const GgMatrix& mp, const GgMatrix& mv) const
08063     {
08064         use(mp, mv, mv.normal());
08065     }
08066
08073     void use(const LightBuffer* light, GLint i = 0) const
08074     {
08075         // プログラムオブジェクトを指定する
08076         use();
08077
08078         // 光源を設定する
08079         light->select(i);
08080     }
08081
08088     void use(const LightBuffer& light, GLint i = 0) const
08089     {
08090         use(&light, i);
08091     }
08092
08102     void use(
08103         const GLfloat* mp,
08104         const GLfloat* mv,
08105         const GLfloat* mn,
08106         const LightBuffer* light,
08107         GLint i = 0
08108     ) const
08109     {
08110         // 光源を指定してプログラムオブジェクトを指定する
08111         use(light, i);
08112
08113         // 変換行列を設定する
08114         loadMatrix(mp, mv, mn);
08115     }

```

```

08116     void use(
08117         const GgMatrix& mp,
08118         const GgMatrix& mv,
08119         const GgMatrix& mn,
08120         const LightBuffer& light,
08121         GLint i = 0
08122     ) const
08123     {
08124         use(mp.get(), mv.get(), mn.get(), &light, i);
08125     }
08126
08127     void use(
08128         const GLfloat* mp,
08129         const GLfloat* mv,
08130         const LightBuffer* light,
08131         GLint i = 0
08132     ) const
08133     {
08134         use(mp.get(), mv.get(), mn.get(), &light, i);
08135     }
08136
08137     void use(
08138         const GgMatrix& mp,
08139         const GgMatrix& mv,
08140         const LightBuffer& light,
08141         GLint i = 0
08142     ) const
08143     {
08144         use(mp, mv, GgMatrix(mv).normal().get(), light, i);
08145     }
08146
08147     void use(
08148         const GLfloat* mp,
08149         const GLfloat* mv,
08150         const LightBuffer* light,
08151         GLint i = 0
08152     ) const
08153     {
08154         use(mp, mv, mv.normal(), light, i);
08155     }
08156
08157     void use(const GLfloat* mp, const LightBuffer* light, GLint i = 0) const
08158     {
08159         // 光源を指定してプログラムオブジェクトを指定する
08160         use(light, i);
08161
08162         // 投影変換行列を設定する
08163         loadProjectionMatrix(mp);
08164     }
08165
08166     void use(const GgMatrix& mp, const LightBuffer& light, GLint i = 0) const
08167     {
08168         // 光源を指定してプログラムオブジェクトを指定する
08169         use(mp.get(), &light, i);
08170     }
08171 };
08172
08173 extern bool ggLoadSimpleObj(
08174     const std::string& name,
08175     std::vector<std::array<GLuint, 3>>& group,
08176     std::vector<GgSimpleShader::Material>& material,
08177     std::vector<GgVertex>& vert,
08178     bool normalize = false
08179 );
08180
08181
08182 extern bool ggLoadSimpleObj(
08183     const std::string& name,
08184     std::vector<std::array<GLuint, 3>>& group,
08185     std::vector<GgSimpleShader::Material>& material,
08186     std::vector<GgVertex>& vert,
08187     std::vector<GLuint>& face,
08188     bool normalize = false
08189 );
08190
08191
08192 class GgSimpleObj
08193 {
08194     // 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
08195     std::shared_ptr<std::vector<std::array<GLuint, 3>>> group;
08196
08197     // ポリゴングループごとの材質のユニフォームバッファ
08198     std::shared_ptr<GgSimpleShader::MaterialBuffer> material;
08199
08200     // この图形の形状データ
08201     std::shared_ptr<GgElements> data;
08202
08203 public:
08204
08205     GgSimpleObj(const std::string& name, bool normalize = false);
08206
08207     virtual ~GgSimpleObj()
08208     {
08209     }
08210
08211     explicit operator bool() const noexcept
08212     {
08213         return data.get() != nullptr;
08214     }

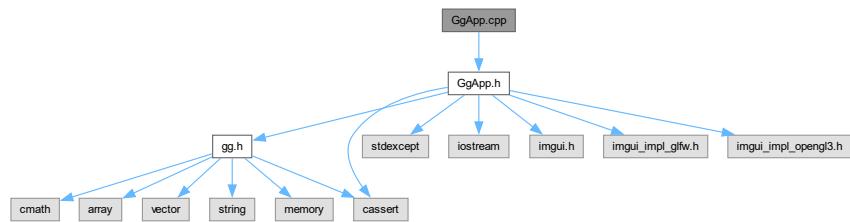
```

```

08279     }
08280
08286     bool operator!() const noexcept
08287     {
08288         return !static_cast<bool>(*this);
08289     }
08290
08296     const GgTriangles* get() const
08297     {
08298         return data.get();
08299     }
08300
08307     virtual void draw(GLint first = 0, GLsizei count = 0) const;
08308 };
08309 }
```

9.9 GgApp.cpp ファイル

#include "GgApp.h"
GgApp.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。

9.10 GgApp.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
00025 #include "GgApp.h"
00026
00027 /**
00028 // GLFW のエラー表示
00029 */
00030 static void glfwErrorCallback(int error, const char* description)
00031 {
00032     #if defined(__aarch64__)
00033         if (error == 65544) return;
00034     #endif
00035     throw std::runtime_error(description);
00036 }
00037
00038 /**
00039 // GgApp クラスのコンストラクタ
00040 /**
00041 GgApp::GgApp(int major, int minor)
00042 {
00043     // GLFW のエラー処理関数を登録する
00044     glfwSetErrorCallback(glfwErrorCallback);
00045
00046     // OpenGL の major 番号が指定されていれば
00047     // OpenGL のバージョンを指定する
00048     // OpenGL ES 3 のコンテキストを指定する
00049     // Core Profile を選択する (macOS の都合)
00050     // OpenGL Version 3.2 以降なら
00051     // Oculus Rift では SRGB でレンダリングする
00052     // ImGui のバージョンをチェックする
00053     // ImGui のコンテキストを作成する
00054     // ImGui::CreateContext();
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 // GgApp::~GgApp()
00094 {
00095 #if defined(IMGUI_VERSION)
00096     // Shutdown Platform/Renderer bindings
00097     ImGui_ImplOpenGL3_Shutdown();
00098     ImGui_ImplGlfw_Shutdown();
00099     ImGui::DestroyContext();
00100 #endif
00101
00102 // プログラム終了時に GLFW を終了する
00103     glfwTerminate();
00104 }
00105
00106 //
00107 // マウスや矢印キーによる平行移動量を初期化する
00108 //
00109 // マウスホイールの回転量を初期化する
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 // Window クラスのコンストラクタ
00346 //
00347 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 #if defined(IMGUI_VERSION)
00450 // ImGui の新規フレームを作成する
00451 ImGui_ImplOpenGL3_NewFrame();
00452 ImGui_ImplGlfw_NewFrame();
00453 ImGui::NewFrame();
00454
00455 // ImGui の状態を取り出す
00456 const ImGuiIO& io{ ImGui::GetIO() };
00457
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 #else
00465 // マウスの現在位置を調べる
00466 double x, y;
00467 glfwGetCursorPos(window, &x, &y);
00468
00469 // マウスの位置を更新する
00470 current_if.mouse = std::array<GLfloat, 2>{ static_cast<GLfloat>(x), static_cast<GLfloat>(y) };
00471
00472 #endif
00473
00474 // マウスドラッグ
00475 for (int button = GLFW_MOUSE_BUTTON_1; button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT; ++button)
00476 {
00477     // マウスボタンを押していたら
00478     if (status[button])
00479     {
00480         // 現在位置と平行移動量を更新する
00481         current_if.calcTranslation(button, velocity);
00482     }
00483 }
00484
00485 return true;
00486 }
00487
00488
00489 //
00490 // カラーバッファを入れ替える
00491 //
00492 void GgApp::Window::swapBuffers() const
00493 {
00494 #if defined(IMGUI_VERSION)
00495 // ImGui の描画データがあればフレームをレンダリングする
00496 ImGui::Render();
00497 ImDrawData* data{ ImGui::GetDrawData() };
00498 if (data) ImGui_ImplOpenGL3_RenderDrawData(data);
00499
00500
00501 // エラーチェック
00502 ggError();
00503
00504 // カラーバッファを入れ替える
00505 glfwSwapBuffers(window);
00506 }
00507
00508 //
00509 // ビューポートのサイズを更新する
00510 //
00511 void GgApp::Window::updateViewport()
00512 {
00513 // フレームバッファの大きさを求める
00514 glfwGetFramebufferSize(window, &fboSize[0], &fboSize[1]);
00515
00516 #if defined(IMGUI_VERSION)
00517 // 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
00518 fboSize[1] -= menubarHeight;
00519
00520 // ウィンドウの縦横比を保存する
00521 aspect = static_cast<GLfloat>(fboSize[0]) / static_cast<GLfloat>(fboSize[1]);

```

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

```

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

```

```

00697     const ovrTextureSwapChainDesc colorDesc
00698     {
00699         ovrTexture_2D,           // Type
00700         OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00701         1,                      // ArraySize
00702         textureSize.w,          // Width
00703         textureSize.h,          // Height
00704         1,                      // MipLevels
00705         1,                      // SampleCount
00706         ovrFalse,               // StaticImage
00707         0, 0
00708     };
00709
00710     // 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
00711     oculus.layerData.ColorTexture[eye] = nullptr;
00712     if (OVR_SUCCESS(ovr_CreateTextureSwapChainGL(oculus.session, &colorDesc,
00713     &oculus.layerData.ColorTexture[eye])))
00714     {
00715         // 作成したテクスチャチェインの長さを取得する
00716         int length(0);
00717         if (OVR_SUCCESS(ovr_GetTextureSwapChainLength(oculus.session, oculus.layerData.ColorTexture[eye],
00718         &length)))
00719         {
00720             // テクスチャチェインの個々の要素について
00721             for (int i = 0; i < length; ++i)
00722             {
00723                 // テクスチャのパラメータを設定する
00724                 GLuint texId{ 0 };
00725                 ovr_GetTextureSwapChainBufferGL(oculus.session, oculus.layerData.ColorTexture[eye], i,
00726                 &texId);
00727                 glBindTexture(GL_TEXTURE_2D, texId);
00728                 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00729                 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00730                 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00731                 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
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
00746     // 描画データに視野を設定する
00747     oculus.layerData.EyeFov.Fov[eye] = fov;
00748
00749     // 描画データにビューポートを設定する
00750     oculus.layerData.EyeFov.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00751     oculus.layerData.EyeFov.Viewport[eye].Size = textureSize;
00752
00753     // 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
00754     ovrSwapTextureSet* colorTexture{ nullptr };
00755     ovr_CreateSwapTextureSetGL(oculus.session, GL_RGB8_ALPHA8, textureSize.w, textureSize.h,
00756     &colorTexture);
00757     oculus.layerData.EyeFov.ColorTexture[eye] = colorTexture;
00758
00759     // 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
00760     ovrSwapTextureSet* depthTexture{ nullptr };
00761     ovr_CreateSwapTextureSetGL(oculus.session, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h,
00762     &depthTexture);
00763     oculus.layerData.EyeFovDepth.DepthTexture[eye] = depthTexture;
00764
00765 # endif
00766 }
00767 # if OVR_PRODUCT_VERSION > 0
00768
00769     // 姿勢のトラッキングにおける床の高さを 0 に設定する
00770     ovr_SetTrackingOriginType(oculus.session, ovrTrackingOrigin_FloorLevel);
00771
00772     // ミラー表示用の FBO を作成する
00773     const GLsizei* size{ oculus.window->GetSize() };
00774     const ovrMirrorTextureDesc mirrorDesc
00775     {
00776         OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00777         oculus.mirrorWidth = size[0],   // Width

```

```

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

```

```
00863     if (layerData.ColorTexture[eye])
00864     {
00865         ovr_DestroyTextureSwapChain(session, layerData.ColorTexture[eye]);
00866         layerData.ColorTexture[eye] = nullptr;
00867     }
00868
00869     // デブスマッファとして使ったテクスチャを削除する
00870     glDeleteTextures(1, oculusDepth + eye);
00871     oculusDepth[eye] = 0;
00872
00873 # else
00874
00875     // 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
00876     auto* const colorTexture(layerData.EyeFov.ColorTexture[eye]);
00877     for (int i = 0; i < colorTexture->TextureCount; ++i)
00878     {
00879         const auto* const ctex(reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[i]));
00880         glDeleteTextures(1, &ctex->OGL.TexId);
00881     }
00882     ovr_DestroySwapTextureSet(session, colorTexture);
00883
00884     // デブスマッファとして使ったテクスチャを削除する
00885     auto* const depthTexture(layerData.EyeFovDepth.DepthTexture[eye]);
00886     for (int i = 0; i < depthTexture->TextureCount; ++i)
00887     {
00888         const auto* const dtex(reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[i]));
00889         glDeleteTextures(1, &dtex->OGL.TexId);
0090     }
00901     ovr_DestroySwapTextureSet(session, depthTexture);
00902
00903 # endif
00904 }
00905
00906 // Oculus Rift のセッションを破棄する
00907 ovrDestroy(session);
00908 session = nullptr;
00909
00910 // カラースペースを元に戻す
00911 glDisable(GL_FRAMEBUFFER_SRGB);
00912
00913 // バックバッファに描く
00914 glDrawBuffer(GL_BACK);
00915
00916 // 垂直同期タイミングに合わせる
00917 glfwSwapInterval(1);
00918
00919
00920 // //
00921 // Oculus Rift による描画開始
00922 //
00923 bool GgApp::Oculus::begin()
00924 {
00925     # if OVR_PRODUCT_VERSION > 0
00926
00927     // セッションの状態を取得する
00928     ovrSessionStatus sessionStatus;
00929     ovr_GetSessionStatus(session, &sessionStatus);
00930
00931     // 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
00932     if (sessionStatus.ShouldQuit) window->setClose(GLFW_TRUE);
00933
00934     // Oculus Rift に表示されていないときは戻る
00935     if (!sessionStatus.IsVisible) return false;
00936
00937     // 現在の状態をトラッキングの原点にする
00938     if (sessionStatus.ShouldRecenter) ovr_RecenterTrackingOrigin(session);
00939
00940     // 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
00941     const ovrEyeRenderDesc eyeRenderDesc[]
00942     {
00943         ovr_GetRenderDesc(session, ovrEyeType(0), hmdDesc.DefaultEyeFov[0]),
00944         ovr_GetRenderDesc(session, ovrEyeType(1), hmdDesc.DefaultEyeFov[1])
00945     };
00946
00947     // 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
00948     const ovrPosef hmdToEyePose[]
00949     {
00950         eyeRenderDesc[0].HmdToEyePose,
00951         eyeRenderDesc[1].HmdToEyePose
00952     };
00953
00954     // 視点の姿勢情報を取得する
00955     ovr_GetEyePoses(session, frameIndex, ovrTrue, hmdToEyePose, layerData.RenderPose,
00956     &layerData.SensorSampleTime);
00957
00958 # else
00959
```

```

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

```

```

01034     screen[3] = this->screen[eye][3];
01035
01036     // Oculus Rift の位置を返す
01037     position[0] = p.x;
01038     position[1] = p.y;
01039     position[2] = p.z;
01040
01041     // Oculus Rift の方向を返す
01042     orientation[0] = o.x;
01043     orientation[1] = o.y;
01044     orientation[2] = o.z;
01045     orientation[3] = o.w;
01046 }
01047
01048 //
01049 // Time Warp 处理に使う投影変換行列の成分の設定 (DK1, DK2)
01050 //
01051 void GgApp::Oculus::timewarp(const GgMatrix& projection)
01052 {
01053 # if OVR_PRODUCT_VERSION < 1
01054 // TimeWarp に使う変換行列の成分を設定する
01055 auto& posTimewarpProjectionDesc{ layerData.EyeFovDepth.ProjectionDesc };
01056 posTimewarpProjectionDesc.Projection22 = (projection.get()[4 * 2 + 2] + projection.get()[4 * 3 + 2])
* 0.5f;
01057 posTimewarpProjectionDesc.Projection23 = projection.get()[4 * 2 + 3] * 0.5f;
01058 posTimewarpProjectionDesc.Projection32 = projection.get()[4 * 3 + 2];
01059 # endif
01060 }
01061
01062 //
01063 // 図形の描画を完了する (CV1 以降)
01064 //
01065 void GgApp::Oculus::commit(int eye)
01066 {
01067 # if OVR_PRODUCT_VERSION > 0
01068 // GL_COLOR_ATTACHMENT0 に割り当てられたテクスチャが wglDXUnlockObjectsNV() によって
01069 // 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
01070 // FBO に結合されるのを避ける
01071 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
01072 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
01073 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, 0, 0);
01074
01075 // 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
01076 ovrCommitTextureSwapChain(session, layerData.ColorTexture[eye]);
01077 # endif
01078 }
01079
01080 //
01081 // フレームを転送する
01082 //
01083 bool GgApp::Oculus::submit(bool mirror)
01084 {
01085 // エラーチェック
01086 ggError();
01087
01088 # if OVR_PRODUCT_VERSION > 0
01089 // 描画データを Oculus Rift に転送する
01090 const auto* const layers{ &layerData.Header };
01091 if (OVR_FAILURE(ovrSubmitFrame(session, frameIndex++, nullptr, &layers, 1))) return false;
01092 # else
01093 // Oculus Rift 上の描画位置と拡大率を求める
01094 ovrViewScaleDesc viewScaleDesc;
01095 viewScaleDesc.HmdSpaceToWorldScaleInMeters = 1.0f;
01096 viewScaleDesc.HmdToEyeViewOffset[0] = eyeRenderDesc[0].HmdToEyeViewOffset;
01097 viewScaleDesc.HmdToEyeViewOffset[1] = eyeRenderDesc[1].HmdToEyeViewOffset;
01098
01099 // 描画データを更新する
01100 layerData.EyeFov.RenderPose[0] = eyePose[0];
01101 layerData.EyeFov.RenderPose[1] = eyePose[1];
01102
01103 // 描画データを oculus Rift に転送する
01104 const auto* const layers{ &layerData.Header };
01105 if (OVR_FAILURE(ovrSubmitFrame(session, 0, &viewScaleDesc, &layers, 1))) return false;
01106 # endif
01107
01108 // ミラー表示
01109 if (mirror)
01110 {
01111 # if OVR_PRODUCT_VERSION > 0
01112     const auto& sx1{ mirrorWidth };
01113     const auto& sy1{ mirrorHeight };
01114 # else
01115     const auto& sx1{ mirrorTexture->OGL.Header.TextureSize.w };
01116     const auto& sy1{ mirrorTexture->OGL.Header.TextureSize.h };
01117 # endif
01118 // ミラー表示のウィンドウのサイズ

```

```

01120     GLsizei size[2];
01121     window->getSize(size);
01122
01123     // ミラー表示の表示領域
01124     GLint dx0{ 0 }, dx1{ size[0] }, dy0{ 0 }, dy1{ size[1] };
01125
01126     // ミラー表示がウィンドウからはみ出ないようにする
01127     if ((size[0] *= syl) < (size[1] *= sx1))
01128     {
01129         const GLint tyl{ size[0] / sx1 };
01130         dy0 = (dy1 - tyl) / 2;
01131         dy1 = dy0 + tyl;
01132     }
01133     else
01134     {
01135         const GLint tx1{ size[1] / syl };
01136         dx0 = (dx1 - tx1) / 2;
01137         dx1 = dx0 + tx1;
01138     }
01139
01140     // 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
01141     glBindFramebuffer(GL_READ_FRAMEBUFFER, mirrorFbo);
01142     glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
01143     glBlitFramebuffer(0, syl, sx1, 0, dx0, dy0, dx1, dy1, GL_COLOR_BUFFER_BIT, GL_NEAREST);
01144     glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
01145
01146     // 残っている OpenGL コマンドを実行する
01147     glFlush();
01148 }
01149
01150     return true;
01151 }
01152 #endif

```

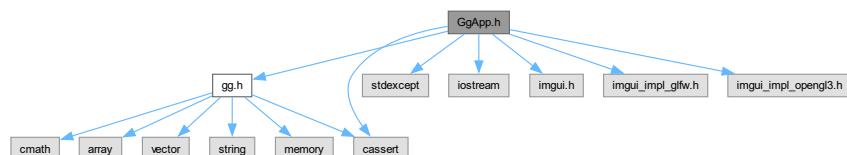
9.11 GgApp.h ファイル

```

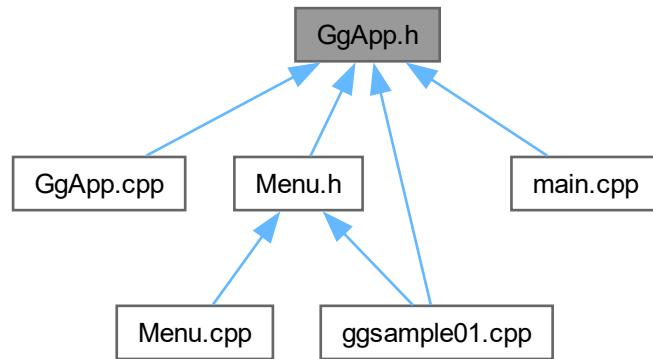
#include "gg.h"
#include <cassert>
#include <stdexcept>
#include <iostream>
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"

```

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`

9.11.1 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。

9.11.2 マクロ定義詳解

9.11.2.1 GG_BUTTON_COUNT

```
#define GG_BUTTON_COUNT 3
```

[GgApp.h](#) の 43 行目に定義があります。

9.11.2.2 GG_INTERFACE_COUNT

```
#define GG_INTERFACE_COUNT 5
```

GgApp.h の 48 行目に定義があります。

9.11.2.3 GG_USE_IMGUI

```
#define GG_USE_IMGUI
```

GgApp.h の 36 行目に定義があります。

9.12 GgApp.h

[詳解]

```
00001 #pragma once
00002 /*
00003 */
00004 ゲームグラフィックス特論用補助プログラム GLFW3 版
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
00025 /*
00026
00027
00028 // Dear ImGui を使うなら
00029 #define GG_USE_IMGUI
00030
00031
00032 // Oculus Rift を使うなら
00033 // #define GG_USE_OOCULUS_RIFT
00034
00035
00036 // 使用するマウスのボタン数
00037 #if !defined(GG_BUTTON_COUNT)
00038 # define GG_BUTTON_COUNT 3
00039 #endif
00040
00041 // 使用するユーザインターフェースの数
00042 #if !defined(GG_INTERFACE_COUNT)
00043 # define GG_INTERFACE_COUNT 5
00044 #endif
00045
00046 // 補助プログラム
00047 #if !defined(GG_INTERFACE_COUNT)
00048 # define GG_INTERFACE_COUNT 5
00049 #endif
00050
00051 // 標準ライブラリ
00052 #include "gg.h"
00053 using namespace gg;
00054
00055 // ImGui の組み込み
00056 #if defined(GG_USE_IMGUI)
00057 # include "imgui.h"
00058 # include "imgui_impl_glfw.h"
00059 # include "imgui_impl_opengl3.h"
00060 #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{},
```

```

00200     mouse{},
00201     wheel{},
00202     translation{}
00203 {
00204     resetTranslation();
00205 }
00206
00207 /**
00208 // マウスや矢印キーによる平行移動量を初期化する
00209 /**
00210 void resetTranslation();
00211
00212 /**
00213 // 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)
00214 /**
00215 void calcTranslation(int button, const std::array<GLfloat, 3>& velocity);
00216 };
00217
00218 // ヒューマンインターフェースデバイスのデータ
00219 std::array<HumanInterface, GG_INTERFACE_COUNT> interfaceData;
00220
00221 // ヒューマンインターフェースデバイスの番号
00222 int interfaceNo;
00223
00224 /**
00225 // ユーザー定義のコールバック関数へのポインタ
00226 /**
00227 void* userPointer;
00228 void (*resizeFunc)(const Window* window, int width, int height);
00229 void (*keyboardFunc)(const Window* window, int key, int scanCode, int action, int mods);
00230 void (*mouseFunc)(const Window* window, int button, int action, int mods);
00231 void (*wheelFunc)(const Window* window, double x, double y);
00232
00233 /**
00234 // ウィンドウのサイズ変更時の処理
00235 /**
00236 static void resize(GLFWwindow* window, int width, int height);
00237
00238 /**
00239 // キーボードをタイプした時の処理
00240 /**
00241 static void keyboard(GLFWwindow* window, int key, int scanCode, int action, int mods);
00242
00243 /**
00244 // マウスボタンを操作したときの処理
00245 /**
00246 static void mouse(GLFWwindow* window, int button, int action, int mods);
00247
00248 /**
00249 // マウスホイールを操作した時の処理
00250 /**
00251 static void wheel(GLFWwindow* window, double x, double y);
00252
00253 public:
00254
00255     Window(const std::string& title = "GLFW Window", int width = 640, int height = 480,
00256             int fullscreen = 0, GLFWwindow* share = nullptr);
00257
00258     Window(const Window& w) = delete;
00259
00260     Window(Window&& w) = default;
00261
00262     virtual ~Window()
00263     {
00264         // ウィンドウが作成されていなければ戻る
00265         if (!window) return;
00266
00267         // ウィンドウを破棄する
00268         glfwDestroyWindow(window);
00269     }
00270
00271     Window& operator=(const Window& w) = delete;
00272
00273     Window& operator=(Window&& w) = default;
00274
00275     auto* get() const
00276     {
00277         return window;
00278     }
00279
00280     void setClose(int flag = GLFW_TRUE) const
00281     {
00282         glfwSetWindowShouldClose(window, flag);
00283     }
00284
00285     bool shouldClose() const
00286     {

```

```
00336     // ウィンドウを閉じるべきなら true を返す
00337     return glfwWindowShouldClose(window) != GLFW_FALSE;
00338 }
00339
00340 explicit operator bool();
00341
00342 void swapBuffers() const;
00343
00344 void restoreViewport() const
00345 {
00346     if (!glfwGetWindowAttrib(window, GLFW_ICONIFIED)) glViewport(0, 0, fboSize[0], fboSize[1]);
00347 }
00348
00349 void updateViewport();
00350
00351 #if defined(IMGUI_VERSION)
00352     void setMenubarHeight(GLsizei height)
00353     {
00354         // メニューバーの高さを保存する
00355         menubarHeight = height;
00356
00357         // ビューポートを復帰する
00358         updateViewport();
00359     }
00360 #endif
00361
00362 auto getWidth() const
00363 {
00364     return size[0];
00365 }
00366
00367 auto getHeight() const
00368 {
00369     return size[1];
00370 }
00371
00372 auto getFboWidth() const
00373 {
00374     return fboSize[0];
00375 }
00376
00377 auto getFboHeight() const
00378 {
00379     return fboSize[1];
00380 }
00381
00382 const auto& getSize() const
00383 {
00384     return size;
00385 }
00386
00387 void getSize(GLsizei* size) const
00388 {
00389     size[0] = getWidth();
00390     size[1] = getHeight();
00391 }
00392
00393 const auto& getFboSize() const
00394 {
00395     return fboSize;
00396 }
00397
00398 void getFboSize(GLsizei* fboSize) const
00399 {
00400     fboSize[0] = getFboWidth();
00401     fboSize[1] = getFboHeight();
00402 }
00403
00404 auto getAspect() const
00405 {
00406     return aspect;
00407 }
00408
00409 bool getKey(int key) const
00410 {
00411 #if defined(IMGUI_VERSION)
00412     // 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
00413     if (ImGui::GetIO().WantCaptureKeyboard) return false;
00414 #endif
00415
00416     return glfwGetKey(window, key) != GLFW_RELEASE;
00417 }
00418
00419 void selectInterface(int no)
00420 {
00421     assert(static_cast<size_t>(no) < interfaceData.size());
00422     interfaceNo = no;
```

```
00497     }
00498
00506 void setVelocity(GLfloat vx, GLfloat vy, GLfloat vz = 0.1f)
00507 {
00508     velocity = std::array<GLfloat, 3>{ vx, vy, vz };
00509 }
00510
00516 int getLastKey()
00517 {
00518     auto& current_if{ interfaceData[interfaceNo] };
00519     const int key{ current_if.lastKey };
00520     current_if.lastKey = 0;
00521     return key;
00522 }
00523
00531 auto getArrow(int direction = 0, int mods = 0) const
00532 {
00533     const auto& current_if{ interfaceData[interfaceNo] };
00534     return static_cast<GLfloat>(current_if.arrow[mods & 3][direction & 1]);
00535 }
00536
00543 auto getArrowX(int mods = 0) const
00544 {
00545     return getArrow(0, mods);
00546 }
00547
00554 auto getArrowY(int mods = 0) const
00555 {
00556     return getArrow(1, mods);
00557 }
00558
00565 void getArrow(GLfloat* arrow, int mods = 0) const
00566 {
00567     arrow[0] = getArrowX(mods);
00568     arrow[1] = getArrowY(mods);
00569 }
00570
00576 auto getShiftArrowX() const
00577 {
00578     return getArrow(0, 1);
00579 }
00580
00586 auto getShiftArrowY() const
00587 {
00588     return getArrow(1, 1);
00589 }
00590
00596 void getShiftArrow(GLfloat* shift_arrow) const
00597 {
00598     shift_arrow[0] = getShiftArrowX();
00599     shift_arrow[1] = getShiftArrowY();
00600 }
00601
00607 auto getControlArrowX() const
00608 {
00609     return getArrow(0, 2);
00610 }
00611
00617 auto getControlArrowY() const
00618 {
00619     return getArrow(1, 2);
00620 }
00621
00627 void getControlArrow(GLfloat* control_arrow) const
00628 {
00629     control_arrow[0] = getControlArrowX();
00630     control_arrow[1] = getControlArrowY();
00631 }
00632
00638 auto getAltArrowX() const
00639 {
00640     return getArrow(0, 3);
00641 }
00642
00648 auto getAltArrowY() const
00649 {
00650     return getArrow(1, 3);
00651 }
00652
00658 void getAltArrow(GLfloat* alt_arrow) const
00659 {
00660     alt_arrow[0] = getAltArrowX();
00661     alt_arrow[1] = getAltArrowY();
00662 }
00663
00669 const auto* getMouse() const
00670 {
```

```
00671     const auto& current_if{ interfaceData[interfaceNo] };
00672     return current_if.mouse.data();
00673 }
00674
00680 void getMouse(GLfloat* position) const
00681 {
00682     const auto& current_if{ interfaceData[interfaceNo] };
00683     position[0] = current_if.mouse[0];
00684     position[1] = current_if.mouse[1];
00685 }
00686
00688 auto getMouse(int direction) const
00689 {
00690     const auto& current_if{ interfaceData[interfaceNo] };
00691     return current_if.mouse[direction & 1];
00692 }
00693
00704 auto getMouseX() const
00705 {
00706     const auto& current_if{ interfaceData[interfaceNo] };
00707     return current_if.mouse[0];
00708 }
00709
00715 auto getMouseY() const
00716 {
00717     const auto& current_if{ interfaceData[interfaceNo] };
00718     return current_if.mouse[1];
00719 }
00720
00726 const auto* getWheel() const
00727 {
00728     const auto& current_if{ interfaceData[interfaceNo] };
00729     return current_if.wheel.data();
00730 }
00731
00737 void getWheel(GLfloat* rotation) const
00738 {
00739     const auto& current_if{ interfaceData[interfaceNo] };
00740     rotation[0] = current_if.wheel[0];
00741     rotation[1] = current_if.wheel[1];
00742 }
00743
00750 auto getWheel(int direction) const
00751 {
00752     const auto& current_if{ interfaceData[interfaceNo] };
00753     return current_if.wheel[direction & 1];
00754 }
00755
00761 auto getWheelX() const
00762 {
00763     const auto& current_if{ interfaceData[interfaceNo] };
00764     return current_if.wheel[0];
00765 }
00766
00772 auto getWheelY() const
00773 {
00774     const auto& current_if{ interfaceData[interfaceNo] };
00775     return current_if.wheel[1];
00776 }
00777
00784 const auto& getTranslation(int button = GLFW_MOUSE_BUTTON_1) const
00785 {
00786     const auto& current_if{ interfaceData[interfaceNo] };
00787     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00788     return current_if.translation[button][1];
00789 }
00790
00797 auto getTranslationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00798 {
00799     const auto& current_if{ interfaceData[interfaceNo] };
00800     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00801     const auto& t{ current_if.translation[button][1] };
00802
00803     GgMatrix m
00804     {
00805         1.0f, 0.0f, 0.0f, 0.0f,
00806         0.0f, 1.0f, 0.0f, 0.0f,
00807         0.0f, 0.0f, 1.0f, 0.0f,
00808         t[0], t[1], t[2], 1.0f
00809     };
00810
00811     return m;
00812 }
00813
00820 auto getScrollMatrix(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     const auto& t{ current_if.translation[button][1] };
00825
00826     GgMatrix m
00827     {
00828         t[2] + 1.0f, 0.0f, 0.0f, 0.0f,
00829         0.0f, t[2] + 1.0f, 0.0f, 0.0f,
00830         0.0f, 0.0f, 1.0f, 0.0f,
00831         t[0], t[1], 0.0f, 1.0f
00832     };
00833
00834     return m;
00835 }
00836
00843     auto getRotation(int button = GLFW_MOUSE_BUTTON_1) const
00844     {
00845         const auto& current_if{ interfaceData[interfaceNo] };
00846         assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00847         return current_if.rotation[button].getQuaternion();
00848     }
00849
00856     auto getRotationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00857     {
00858         const auto& current_if{ interfaceData[interfaceNo] };
00859         assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00860         return current_if.rotation[button].getMatrix();
00861     }
00862
00866     void resetRotation()
00867     {
00868         // トラックボールを初期化する
00869         for (auto& tb : interfaceData[interfaceNo].rotation) tb.reset();
00870     }
00871
00875     void resetTranslation()
00876     {
00877         // 現在のインターフェースの平行移動量を初期化する
00878         interfaceData[interfaceNo].resetTranslation();
00879     }
00880
00884     void reset()
00885     {
00886         // トラックボール処理を初期化する
00887         resetRotation();
00888
00889         // 平行移動量を初期化する
00890         resetTranslation();
00891     }
00892
00898     void* getUserPointer() const
00899     {
00900         return userPointer;
00901     }
00902
00908     void setUserPointer(void* pointer)
00909     {
00910         userPointer = pointer;
00911     }
00912
00918     void setResizeFunc(void (*func)(const Window* window, int width, int height))
00919     {
00920         resizeFunc = func;
00921     }
00922
00928     void setKeyboardFunc(void (*func)(const Window* window, int key, int scancode, int action, int
00929     mods))
00930     {
00931         keyboardFunc = func;
00932     }
00933
00938     void setMouseFunc(void (*func)(const Window* window, int button, int action, int mods))
00939     {
00940         mouseFunc = func;
00941     }
00942
00948     void setWheelFunc(void (*func)(const Window* window, double x, double y))
00949     {
00950         wheelFunc = func;
00951     }
00952 };
00953
00954 #if defined(GG_USE_OOCULUS_RIFT)
00955     class Oculus
00956     {
00957         // Oculus Rift のセッション
00958         ovrSession session;
00959     };
00960
00961     class Oculus
00962     {
00963         // Oculus Rift のセッション
00964         ovrSession session;
00965     };

```

```
00966 // Oculus Rift の状態
00967 ovrHmdDesc hmdDesc;
00968
00969 // Oculus Rift へのレンダリングに使う FBO
00970 GLuint oculusFbo[ovrEye_Count];
00971
00972 // Oculus Rift のスクリーンのサイズ
00973 GLfloat screen[ovrEye_Count][4];
00974
00975 // ミラー表示用の FBO
00976 GLuint mirrorFbo;
00977
00978 // Oculus Rift のミラー表示を行うウィンドウ
00979 const Window* window;
00980
00981 # if OVR_PRODUCT_VERSION > 0
00982
00983 // Oculus Rift に送る描画データ
00984 ovrLayerEyeFov layerData;
00985
00986 // Oculus Rift にレンダリングするフレームの番号
00987 long long frameIndex;
00988
00989 // Oculus Rift へのレンダリングに使う FBO のデブステクスチャ
00990 GLuint oculusDepth[ovrEye_Count];
00991
00992 // ミラー表示用の FBO のサイズ
00993 int mirrorWidth, mirrorHeight;
00994
00995 // ミラー表示用の FBO のカラー テクスチャ
00996 ovrMirrorTexture mirrorTexture;
00997
00998 //
00999 // グラフィックスカードのデフォルトの LUID を得る
01000 //
01001 static ovrGraphicsLuid GetDefaultAdapterLuid();
01002
01003 //
01004 // グラフィックスカードの LUID の比較
01005 //
01006 static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);
01007
01008 # else
01009
01010 // Oculus Rift に送る描画データ
01011 ovrLayerUnion layerData;
01012
01013 // Oculus Rift のレンダリング情報
01014 ovrEyeRenderDesc eyeRenderDesc[ovrEye_Count];
01015
01016 // Oculus Rift の視点情報
01017 ovrPosef eyePose[ovrEye_Count];
01018
01019 // ミラー表示用の FBO のカラー テクスチャ
01020 ovrGLTexture* mirrorTexture;
01021
01022 # endif
01023
01024 //
01025 // コンストラクタ
01026 //
01027 Oculus();
01028
01029 //
01030 // デストラクタ
01031 //
01032 virtual ~Oculus() = default;
01033
01034 public:
01035
01036 // シングルトンなのでコピー・ムーブ禁止
01037 Oculus(const Oculus&) = delete;
01038 Oculus& operator=(const Oculus&) = delete;
01039 Oculus(Oculus&&) = delete;
01040 Oculus& operator=(Oculus&&) = delete;
01041
01042 static Oculus& initialize(const Window& window);
01043
01044 void terminate();
01045
01046 bool begin();
01047
01048 void select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation);
01049
01050 void timewarp(const GgMatrix& projection);
01051
01052 void commit(int eye);
```

```

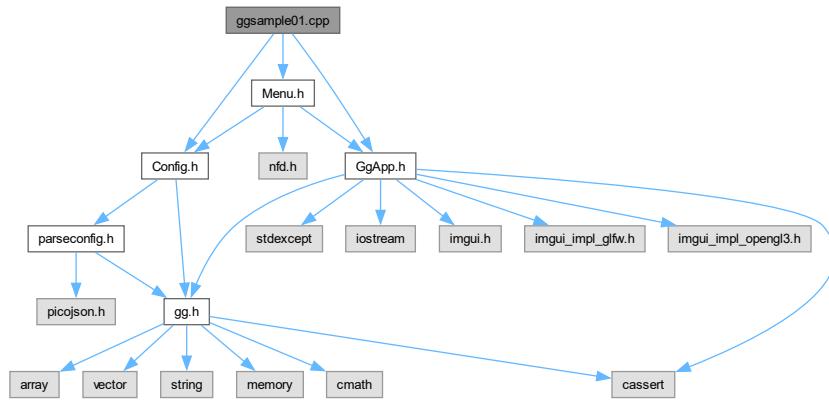
01084     bool submit(bool mirror = true);
01091 };
01092 };
01093 #endif
01094 };

```

9.13 ggsample01.cpp ファイル

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

9.13.1 マクロ定義詳解

9.13.1.1 CONFIG_FILE

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

ggsample01.cpp の 15 行目に定義があります。

9.13.1.2 PROJECT_NAME

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

ggsample01.cpp の 10 行目に定義があります。

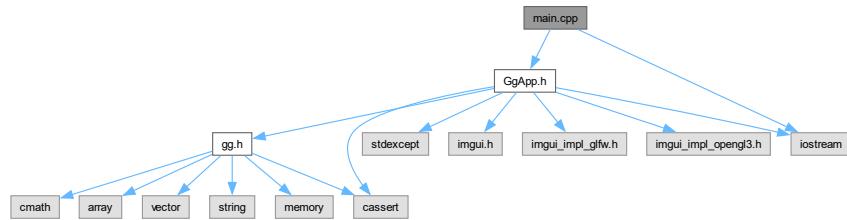
9.14 ggsample01.cpp

[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  
00005 // 補助プログラムのラッパー  
00006 #include "GgApp.h"  
00007  
00008 // プロジェクト名  
00009 #if !defined(PROJECT_NAME)  
00010 # define PROJECT_NAME "ggsample01"  
00011 #endif  
00012  
00013 // 構成ファイル名  
00014 #if !defined(CONFIG_FILE)  
00015 # define CONFIG_FILE PROJECT_NAME ".config.json"  
00016 #endif  
00017  
00018 // 構成データ  
00019 #include "Config.h"  
00020  
00021 // メニューの描画  
00022 #include "Menu.h"  
00023  
00024 //  
00025 // アプリケーション本体  
00026 //  
00027 int GgApp::main(int argc, const char* const* argv)  
00028 {  
00029     // 設定を読み込む  
00030     const Config config{ CONFIG_FILE };  
00031  
00032     // ウィンドウを作成する  
00033     Window window{ argc > 1 ? argv[1] : PROJECT_NAME, config.getWidth(), config.getHeight() };  
00034  
00035     // メニューを初期化する  
00036     Menu menu{ config };  
00037  
00038     // 背景色を設定する  
00039     glClearColor(0.1f, 0.2f, 0.3f, 0.0f);  
00040  
00041     // 隠面消去処理を設定する  
00042     glEnable(GL_DEPTH_TEST);  
00043     glEnable(GL_CULL_FACE);  
00044  
00045     // ビュー変換行列を設定する  
00046     const GgMatrix mv{ ggLookat(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f) };  
00047  
00048     // ウィンドウが開いている間繰り返す  
00049     while (window)  
00050     {  
00051         // ウィンドウを消去する  
00052         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
00053  
00054         // メニューを表示する  
00055         menu.draw();  
00056  
00057         // オブジェクトの回転の変換行列を設定する  
00058         const auto& mr{ window.getRotationMatrix(0) };  
00059  
00060         // 視点の平行移動の変換行列を設定する  
00061         const auto& mt{ window.getTranslationMatrix(1) };  
00062  
00063         // 投影変換行列を設定する  
00064         const GgMatrix&& mp{ ggPerspective(0.5f, window.getAspect(), 1.0f, 15.0f) };  
00065  
00066         // シェーダを指定する  
00067         menu.getShader().use(mp, mt * mv * mr, menu.getLight());  
00068  
00069         // シーンを描画する  
00070         menu.getModel().draw();  
00071  
00072         // カラーバッファを入れ替えてイベントを取り出す  
00073         window.swapBuffers();  
00074     }  
00075  
00076     return 0;  
00077 }
```

9.15 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)`

9.15.1 詳解

ゲームグラフィックス特論宿題アプリケーション

著者

Kohe Tokoi

日付

February 20, 2024

`main.cpp` に定義があります。

9.15.2 マクロ定義詳解

9.15.2.1 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。

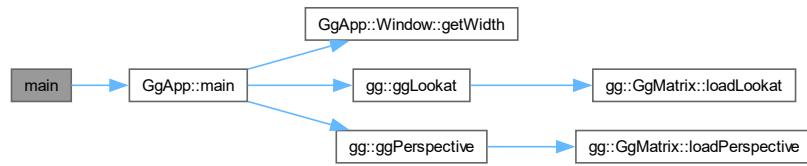
9.15.3 関数詳解

9.15.3.1 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:



9.16 main.cpp

[詳解]

```
00001
00008 #include "GgApp.h"
00009
00010 // MessageBox の準備
00011 #if defined(_MSC_VER)
00012 # include <atlstr.h>
00013 #elif defined(_APPLE_)
00014 # include <CoreFoundation/CoreFoundation.h>
00015 #else
00016 # include <iostream>
00017 #endif
00018 #define HEADER_STR "ゲームグラフィックス特論"
00019
00020 //
00021 // メインプログラム
00022 //
00023 int main(int argc, const char* const* argv) try
00024 {
00025     // アプリケーションのオブジェクトを生成する
00026 #if defined(GL_GLES_PROTOTYPES)
00027     GgApp app(3, 1);
00028 #else
00029     GgApp app(4, 1);
00030 #endif
00031
00032     // アプリケーションを実行する
00033     return app.main(argc, argv);
00034 }
00035 catch (const std::runtime_error &e)
00036 {
00037     // エラーメッセージを表示する
00038 #if defined(_MSC_VER)
00039     MessageBox(NULL, CString(e.what()), TEXT(HEADER_STR), MB_ICONERROR);
00040 #elif defined(_APPLE_)
00041     // the following code is copied from
00042     // http://blog.jorgearimany.com/2010/05/messagebox-from-windows-to-mac.html
00043     CFStringRef msg.ref = CFStringCreateWithCString(NULL, e.what(), kCFStringEncodingUTF8);
00044
00045     // result code from the message box
00046     CFOptionFlags result;
00047
00048     // launch the message box
  
```

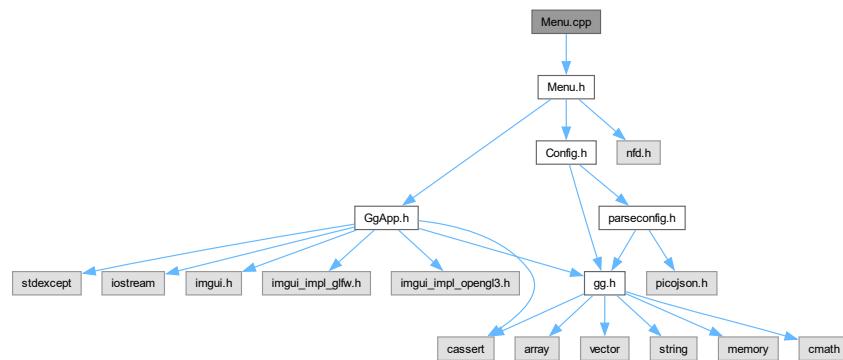
```

00049 CFUserNotificationDisplayAlert(
00050     0,                                     // no timeout
00051     kCFUserNotificationAlertLevel,          // change it depending message.type flags ( MB_ICONASTERISK.....
etc.)
00052     NULL,                                // icon url, use default, you can change it depending
00053     message_type flags
00054     NULL,                                // not used
00055     NULL,                                // localization of strings
00056     CFSTR(HEADER_STR),                   // header text
00057     msg.ref,                            // message text
00058     NULL,                                // default "ok" text in button
00059     NULL,                                // alternate button title
00060     NULL,                                // other button title, null--> no other button
00061     &result                                // response flags
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 }

```

9.17 Menu.cpp ファイル

#include "Menu.h"
 Menu.cpp の依存先関係図:



9.17.1 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。

9.18 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     light{ std::make_unique<GgSimpleShader>::LightBuffer>(config.light) },
00017     shader{ std::make.unique<GgSimpleShader>(config.shader) },
00018     model{ std::make.unique<GgSimpleObj>(config.model, true) }
00019 {
00020 #if defined(IMGUI_VERSION)
00021     //
00022     // ImGui の初期設定
00023     //
00024     // ファイルダイアログ (Native File Dialog Extended) を初期化する
00025     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 //
00072 // 描画する
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     // メニューの表示位置を決定する
00082     ImGui::SetNextWindowPos(ImVec2(4, 4), ImGuiCond_Once);
00083     ImGui::SetNextWindowSize(ImVec2(320, 98), ImGuiCond_Once);
00084
00085     // メニュー表示開始
00086     ImGui::Begin(u8"コントロールパネル");
00087
00088     // 光源

```

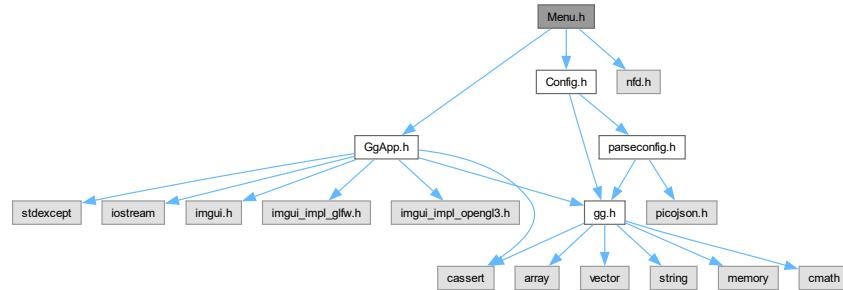
```

00089 if (ImGui::SliderFloat3(u8"光源位置", settings.light.position.data(), -10.0f, 10.0f, "%.2f"))
00090     light->loadPosition(settings.light.position);
00091 if (ImGui::ColorEdit3(u8"光源色", settings.light.diffuse.data(), ImGuiColorEditFlags_Float))
00092     light->loadDiffuse(settings.light.diffuse);
00093
00094 // メニュー表示終了
00095 ImGui::End();
00096 #endif
00097 }

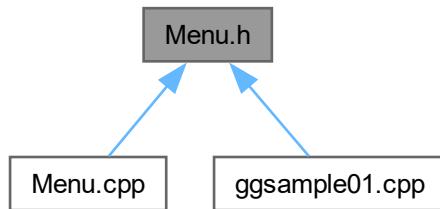
```

9.19 Menu.h ファイル

```
#include "GgApp.h"
#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`

9.19.1 詳解

メニューの描画クラスの定義

著者

Kohe Tokoi

日付

August 15, 2025

Menu.h に定義があります。

9.20 Menu.h

[詳解]

```
00001 #pragma once
00002
00010
00011 // 宿題用補助プログラムのラッパー
00012 #include "GgApp.h"
00013
00014 // 構成データ
00015 #include "Config.h"
00016
00017 // ファイルダイアログ
00018 #include "nfd.h"
00019
00023 class Menu
00024 {
00025     // 図形の描画クラスから参照する
00026     friend class Scene;
00027
00028     // オリジナルの構成データ
00029     const Config& defaults;
00030
00031     // 構成データのコピー
00032     Config settings;
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     // CAD データ
00041     std::unique_ptr<const GgSimpleObj> model;
00042
00043     // ファイルパスを取得する
00044     bool getFilePath(std::string& path, const nfdfilteritem_t* filter);
00045
00046 public:
00047
00051     Menu(const Config& config);
00052
00056     Menu(const Menu& menu) = delete;
00057
00061     Menu(Menu&& menu) = default;
00062
00066     virtual ~Menu();
00067
00071     Menu& operator=(const Menu& menu) = delete;
00072
00076     Menu& operator=(Menu&& menu) = default;
00077
00081     const auto& getLight() const
00082     {
00083         return *light;
00084     }
00085
```

```

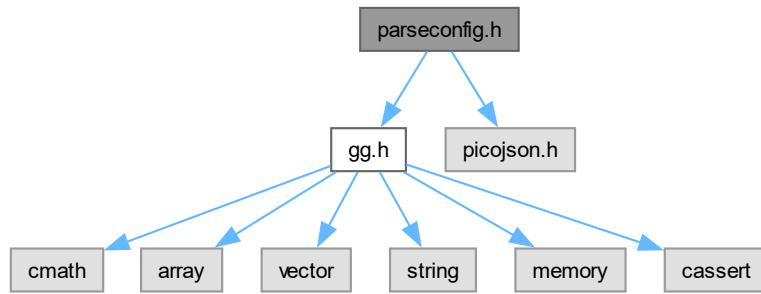
00089 const auto& getShader() const
00090 {
00091     return *shader;
00092 }
00093
00097 const auto& getModel() const
00098 {
00099     return *model;
00100 }
00101
00105 void draw();
00106 };

```

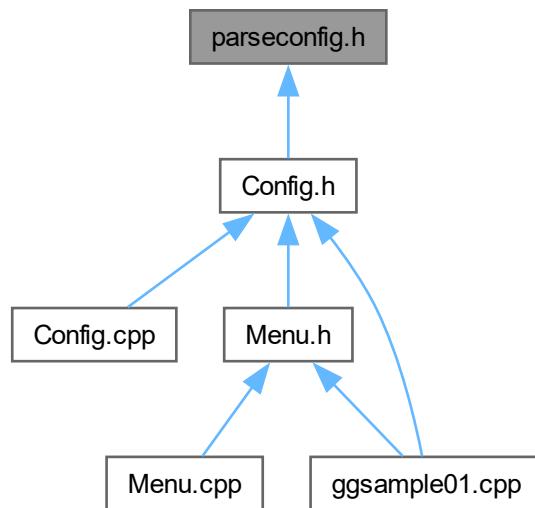
9.21 parseconfig.h ファイル

```
#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)

9.21.1 詳解

構成ファイルの読み取り補助

著者

Kohe Tokoi

日付

November 15, 2022

`parseconfig.h` に定義があります。

9.21.2 関数詳解

9.21.2.1 `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 行目に定義があります。

9.21.2.2 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:



9.21.2.3 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。

9.21.2.4 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 行目に定義があります。

9.21.2.5 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 行目に定義があります。

被呼び出し関係図:



9.21.2.6 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 行目に定義があります。

被呼び出し関係図:



9.21.2.7 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 行目に定義があります。

9.21.2.8 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:



9.21.2.9 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。

9.21.2.10 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。

9.21.2.11 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:



9.21.2.12 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:



9.22 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
00084 inline
00085 bool getVector(const picojson::object& object,
00086     const std::string& key, GgVector& vector)
00087 {
00088     // key に一致するオブジェクトを探す
00089     const auto&& value{ object.find(key) };
00090
00091     // オブジェクトが無いか配列でなかったら戻る
00092     if (value == object.end() || !value->second.is<picojson::array>()) return false;
  
```

```

00093
00094 // 配列を取り出す
00095 const auto& array{ value->second.get<picojson::array>() };
00096
00097 // 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
00098 const auto n{ std::min(sizeof(GgVector) / sizeof(GLfloat), array.size()) };
00099
00100 // 配列の要素について
00101 for (std::size_t i = 0; i < n; ++i)
00102 {
00103     // 要素が数値なら格納する
00104     if (array[i].is<double>()) vector[i] = static_cast<GLfloat>(array[i].get<double>());
00105 }
00106
00107 return true;
00108 }
00109
00110 inline
00111 bool getString(const picojson::object& object,
00112     const std::string& key, std::string& string)
00113 {
00114     // key に一致するオブジェクトを探す
00115     const auto&& value{ object.find(key) };
00116
00117     // オブジェクトが無いか文字列でなかったら戻る
00118     if (value == object.end() || !value->second.is<std::string>()) return false;
00119
00120     // 文字列として格納する
00121     string = value->second.get<std::string>();
00122
00123     return true;
00124 }
00125
00126 template <std::size_t U>
00127 bool getString(const picojson::object& object,
00128     const std::string& key, std::array<std::string, U>& strings)
00129 {
00130     // key に一致するオブジェクトを探す
00131     const auto&& value{ object.find(key) };
00132
00133     // オブジェクトが無いか配列でなかったら戻る
00134     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00135
00136     // 配列を取り出す
00137     const auto& array{ value->second.get<picojson::array>() };
00138
00139     // 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
00140     const auto n{ std::min(U, array.size()) };
00141
00142     // 配列の要素について
00143     for (std::size_t i = 0; i < n; ++i)
00144     {
00145         // 要素が文字列なら文字列として格納する
00146         strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
00147     }
00148
00149     return true;
00150 }
00151
00152 inline
00153 bool getString(const picojson::object& object,
00154     const std::string& key, std::vector<std::string>& strings)
00155 {
00156     // key に一致するオブジェクトを探す
00157     const auto&& value{ object.find(key) };
00158
00159     // オブジェクトが無いか配列でなかったら戻る
00160     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00161
00162     // 配列を取り出す
00163     const auto& array{ value->second.get<picojson::array>() };
00164
00165     // 配列のすべての要素について
00166     for (auto& element : array)
00167     {
00168         // 要素が文字列なら文字列として格納する
00169         strings.emplace_back(element.is<std::string>() ? element.get<std::string>() : "");
00170     }
00171
00172     return true;
00173 }
00174
00175 template <typename T>
00176 void setValue(picojson::object& object,
00177     const std::string& key, const T& scalar)
00178 {
00179     object.emplace(key, picojson::value(static_cast<double>(scalar)));

```

```

00210 }
00211
00212 template <typename T, std::size_t U>
00213     void setValue(picjson::object& object,
00214         const std::string& key, const std::array<T, U>& vector)
00215     {
00216         // picjson の配列
00217         picjson::array array;
00218
00219         // 配列のすべての要素について
00220         for (const auto& element : vector)
00221         {
00222             // 要素を picjson::array に追加する
00223             array.emplace_back(picjson::value(static_cast<double>(element)));
00224         }
00225
00226         // オブジェクトに追加する
00227         object.emplace(key, array);
00228     }
00229
00230     inline
00231     void setVector(picjson::object& object,
00232         const std::string& key, const GgVector& vector)
00233     {
00234         // picjson の配列
00235         picjson::array array;
00236
00237         // ベクトルのすべての要素について
00238         for (const auto& element : vector)
00239         {
00240             // 要素を picjson::array に追加する
00241             array.emplace_back(picjson::value(static_cast<double>(element)));
00242         }
00243
00244         // オブジェクトに追加する
00245         object.emplace(key, array);
00246     }
00247
00248     inline
00249     void setString(picjson::object& object,
00250         const std::string& key, const std::string& string)
00251     {
00252         object.emplace(key, picjson::value(string));
00253     }
00254
00255     template <std::size_t U>
00256     void setString(picjson::object& object,
00257         const std::string& key, const std::array<std::string, U>& strings)
00258     {
00259         // picjson の配列
00260         picjson::array array;
00261
00262         // 配列のすべての要素について
00263         for (const auto& string : strings)
00264         {
00265             // 要素を picjson::array に追加する
00266             array.emplace_back(picjson::value(string));
00267         }
00268
00269         // オブジェクトに追加する
00270         object.emplace(key, array);
00271     }
00272
00273     inline
00274     void setString(picjson::object& object,
00275         const std::string& key, const std::vector<std::string>& strings)
00276     {
00277         // picjson の配列
00278         picjson::array array;
00279
00280         // 配列のすべての要素について
00281         for (auto& string : strings)
00282         {
00283             // 要素を picjson::array に追加する
00284             array.emplace_back(picjson::value(string));
00285         }
00286
00287         // オブジェクトに追加する
00288         object.emplace(key, array);
00289     }
00290
00291     inline
00292     void setString(picjson::object& object,
00293         const std::string& key, const std::vector<std::string>& strings)
00294     {
00295         // picjson の配列
00296         picjson::array array;
00297
00298         // 配列のすべての要素について
00299         for (auto& string : strings)
00300         {
00301             // 要素を picjson::array に追加する
00302             array.emplace_back(picjson::value(string));
00303         }
00304
00305         // オブジェクトに追加する
00306         object.emplace(key, array);
00307     }
00308
00309     inline
00310     void setString(picjson::object& object,
00311         const std::string& key, const std::vector<std::string>& strings)
00312     {
00313         // picjson の配列
00314         picjson::array array;
00315
00316         // 配列のすべての要素について
00317         for (auto& string : strings)
00318         {
00319             // 要素を picjson::array に追加する
00320             array.emplace_back(picjson::value(string));
00321         }
00322
00323         // オブジェクトに追加する
00324         object.emplace(key, array);
00325     }
00326
00327 }
```

9.23 README.md ファイル

Index

-ggError
 gg, 18
-ggFBOError
 gg, 19
~Config
 Config, 86
~GgApp
 GgApp, 91
~GgBuffer
 gg::GgBuffer< T >, 95
~GgColorTexture
 gg::GgColorTexture, 103
~GgElements
 gg::GgElements, 107
~GgMatrix
 gg::GgMatrix, 114
~GgNormalTexture
 gg::GgNormalTexture, 167
~GgPointShader
 gg::GgPointShader, 175
~GgPoints
 gg::GgPoints, 171
~GgQuaternion
 gg::GgQuaternion, 188
~GgShader
 gg::GgShader, 253
~GgShape
 gg::GgShape, 256
~GgSimpleObj
 gg::GgSimpleObj, 259
~GgSimpleShader
 gg::GgSimpleShader, 264
~GgTexture
 gg::GgTexture, 278
~GgTrackball
 gg::GgTrackball, 286
~GgTriangles
 gg::GgTriangles, 296
~GgUniformBuffer
 gg::GgUniformBuffer< T >, 300
~GgVector
 gg::GgVector, 309
~GgVertexArray
 gg::GgVertexArray, 326
~LightBuffer
 gg::GgSimpleShader::LightBuffer, 333
~MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 345
~Menu
 Menu, 352
~Window
 GgApp::Window, 357
add
 gg::GgQuaternion, 189, 190
ambient
 gg::GgSimpleShader::Light, 329
 gg::GgSimpleShader::Material, 341
begin
 gg::GgTrackball, 287
bind
 gg::GgBuffer< T >, 95
 gg::GgTexture, 278
 gg::GgUniformBuffer< T >, 301
 gg::GgVertexArray, 327
BindingPoints
 gg, 18
Config, 85
 ~Config, 86
 Config, 85
 getHeight, 86
 getWidth, 86
 load, 87
 Menu, 89
 save, 88
Config.cpp, 381
 defaultLight, 382
Config.h, 384
CONFIG_FILE
 ggsample01.cpp, 548
conjugate
 gg::GgQuaternion, 191
copy
 gg::GgBuffer< T >, 95
 gg::GgUniformBuffer< T >, 301
defaultLight
 Config.cpp, 382
diffuse
 gg::GgSimpleShader::Light, 329
 gg::GgSimpleShader::Material, 341
distance3
 gg::GgVector, 310
distance4
 gg::GgVector, 310
divide
 gg::GgQuaternion, 192–194

dot3
 gg::GgVector, 311

dot4
 gg::GgVector, 311

draw
 gg::GgElements, 107
 gg::GgPoints, 171
 gg::GgShape, 256
 gg::GgSimpleObj, 260
 gg::GgTriangles, 297
 Menu, 353

end
 gg::GgTrackball, 288

euler
 gg::GgQuaternion, 195, 196

fill
 gg::GgUniformBuffer< T >, 301

frustum
 gg::GgMatrix, 115

get
 gg::GgMatrix, 115, 116
 gg::GgPointShader, 176
 gg::GgQuaternion, 197
 gg::GgShader, 253
 gg::GgShape, 257
 gg::GgSimpleObj, 260
 gg::GgTrackball, 288
 gg::GgVertexArray, 327
 GgApp::Window, 357

getAltArrowX
 GgApp::Window, 357

getAltArrowY
 GgApp::Window, 358

getAltArrow
 GgApp::Window, 359

getArrow
 GgApp::Window, 359, 360

getArrowX
 GgApp::Window, 360

getArrowY
 GgApp::Window, 361

getAspect
 GgApp::Window, 362

getBuffer
 gg::GgBuffer< T >, 97
 gg::GgPoints, 171
 gg::GgTriangles, 297
 gg::GgUniformBuffer< T >, 302

getConjugateMatrix
 gg::GgQuaternion, 198, 199

getControlArrow
 GgApp::Window, 362

getControlArrowX
 GgApp::Window, 363

getControlArrowY
 GgApp::Window, 363

getCount
 gg::GgBuffer< T >, 97
 gg::GgPoints, 172
 gg::GgTriangles, 297
 gg::GgUniformBuffer< T >, 302

getFboHeight
 GgApp::Window, 364

getFboSize
 GgApp::Window, 364, 365

getFboWidth
 GgApp::Window, 365

getHeight
 Config, 86
 gg::GgTexture, 278
 GgApp::Window, 366

getIndexBuffer
 gg::GgElements, 108

getIndexCount
 gg::GgElements, 108

getKey
 GgApp::Window, 366

getLastKey
 GgApp::Window, 366

getLight
 Menu, 353

getMatrix
 gg::GgQuaternion, 199–201
 gg::GgTrackball, 288

getMode
 gg::GgShape, 257

getModel
 Menu, 353

getMouse
 GgApp::Window, 367

getMouseX
 GgApp::Window, 368

getMouseY
 GgApp::Window, 368

getQuaternion
 gg::GgTrackball, 289

getRotation
 GgApp::Window, 368

getRotationMatrix
 GgApp::Window, 368

getScale
 gg::GgTrackball, 289, 290

getScrollMatrix
 GgApp::Window, 369

getShader
 Menu, 353

getShiftArrow
 GgApp::Window, 369

getShiftArrowX
 GgApp::Window, 370

getShiftArrowY
 GgApp::Window, 370

getSize
 gg::GgTexture, 278, 279

GgApp::Window, 371
getStart
 gg::GgTrackball, 290
getStride
 gg::GgBuffer< T >, 97
 gg::GgUniformBuffer< T >, 302
getString
 parseconfig.h, 557, 558
getTarget
 gg::GgBuffer< T >, 97
 gg::GgUniformBuffer< T >, 302
getTexture
 gg::GgTexture, 279
getTranslation
 GgApp::Window, 372
getTranslationMatrix
 GgApp::Window, 372
getUserPointer
 GgApp::Window, 373
getValue
 parseconfig.h, 559
getVector
 parseconfig.h, 560
getWheel
 GgApp::Window, 373
getWheelX
 GgApp::Window, 374
getWheelY
 GgApp::Window, 374
getWidth
 Config, 86
 gg::GgTexture, 279
 GgApp::Window, 374
gg, 15
 _ggError, 18
 _ggFBOError, 19
 BindingPoints, 18
 ggArraysObj, 19
 ggBufferAlignment, 84
 ggConjugate, 20
 ggCreateComputeShader, 20
 ggCreateNormalMap, 21
 ggCreateShader, 22
 ggCross, 23
 ggDistance3, 24, 25
 ggDistance4, 26
 ggDot3, 27, 28
 ggDot4, 28, 29
 ggElementsMesh, 30
 ggElementsObj, 31
 ggElementsSphere, 31
 ggEllipse, 32
 ggEulerQuaternion, 33, 34
 ggFrustum, 35
 ggIdentity, 36
 ggIdentityQuaternion, 36
 ggInit, 36
 ggInvert, 37
 ggLength3, 38, 39
 ggLength4, 40
 ggLoadComputeShader, 41
 ggLoadHeight, 42
 ggLoadImage, 42
 ggLoadShader, 43, 44
 ggLoadSimpleObj, 45, 46
 ggLoadTexture, 46
 ggLookat, 47–49
 ggMatrixQuaternion, 50
 ggNorm, 51
 ggNormal, 52
 ggNormalize, 52
 ggNormalize3, 53–55
 ggNormalize4, 55–57
 ggOrthogonal, 58
 ggPerspective, 58
 ggPointsCube, 59
 ggPointsSphere, 60
 ggQuaternionMatrix, 60
 ggQuaternionTransposeMatrix, 61
 ggReadImage, 62
 ggRectangle, 63
 ggRotate, 63–65, 67
 ggRotateQuaternion, 68, 69
 ggRotateX, 70
 ggRotateY, 70
 ggRotateZ, 71
 ggSaveColor, 71
 ggSaveDepth, 72
 ggSaveTga, 73
 ggScale, 74, 75
 ggSlerp, 76–78
 ggTranslate, 79, 80
 ggTranspose, 81
 LightBindingPoint, 18
 MaterialBindingPoint, 18
 operator+, 82
 operator-, 82, 83
 operator/, 83
 operator*, 81
 gg.cpp, 386
 gg.h, 463
 ggError, 468
 ggFBOError, 468
 pathChar, 468
 pathString, 468
 TCharToUtf8, 468
 Utf8ToTChar, 468
 gg::GgBuffer< T >, 93
 ~GgBuffer, 95
 bind, 95
 copy, 95
 getBuffer, 97
 getCount, 97
 getStride, 97
 getTarget, 97
 GgBuffer, 94, 95

map, 98
 operator=, 99
 read, 99
 send, 100
 unbind, 100
 unmap, 100
gg::GgColorTexture, 101
 ~GgColorTexture, 103
 GgColorTexture, 101, 102
 load, 103, 104
gg::GgElements, 105
 ~GgElements, 107
 draw, 107
 getIndexBuffer, 108
 getIndexCount, 108
 GgElements, 106
 load, 108
 send, 109
gg::GgMatrix, 109
 ~GgMatrix, 114
 frustum, 115
 get, 115, 116
 GgMatrix, 112–114
 invert, 117
 loadFrustum, 117
 loadIdentity, 118
 loadInvert, 119
 loadLookat, 120, 121
 loadNormal, 122, 123
 loadOrthogonal, 124
 loadPerspective, 125
 loadRotate, 125–128
 loadRotateX, 129
 loadRotateY, 130
 loadRotateZ, 130
 loadScale, 131, 132
 loadTranslate, 133, 134
 loadTranspose, 135, 136
 lookat, 136–138
 normal, 139
 operator+, 143, 144
 operator+=, 145
 operator-, 146
 operator-=, 147, 148
 operator/, 148, 149
 operator/=, 150
 operator=, 151, 152
 operator*, 140, 141
 operator*=, 142, 143
 orthogonal, 152
 perspective, 153
 projection, 153, 154
 rotate, 155–157
 rotateX, 158
 rotateY, 159
 rotateZ, 159
 scale, 160, 161
 translate, 162, 163
 transpose, 164
gg::GgNormalTexture, 165
 ~GgNormalTexture, 167
 GgNormalTexture, 166, 167
 load, 167, 168
gg::GgPoints, 169
 ~GgPoints, 171
 draw, 171
 getBuffer, 171
 getCount, 172
 GgPoints, 170
 load, 172
 operator bool, 172
 operator!, 173
 send, 173
gg::GgPointShader, 173
 ~GgPointShader, 175
 get, 176
 GgPointShader, 174, 175
 load, 176
 loadMatrix, 177, 178
 loadModelviewMatrix, 178, 179
 loadProjectionMatrix, 179, 180
 unuse, 180
 use, 180–182
gg::GgQuaternion, 182
 ~GgQuaternion, 188
 add, 189, 190
 conjugate, 191
 divide, 192–194
 euler, 195, 196
 get, 197
 getConjugateMatrix, 198, 199
 getMatrix, 199–201
 GgQuaternion, 186–188
 invert, 201
 loadAdd, 202–204
 loadConjugate, 205
 loadDivide, 207, 208, 210
 loadEuler, 211, 212
 loadIdentity, 213
 loadInvert, 213, 214
 loadMatrix, 215
 loadMultiply, 216, 217
 loadNormalize, 218, 219
 loadRotate, 220, 221
 loadRotateX, 222
 loadRotateY, 222
 loadRotateZ, 223
 loadSlerp, 224–226
 loadSubtract, 226–228
 multiply, 229, 230
 norm, 231
 normalize, 231
 operator+, 234, 235
 operator+=, 235, 236
 operator-, 236, 237
 operator-=, 237, 238

operator/, 239
operator/=, 240, 241
operator=, 241, 242
operator*, 232
operator*=, 233
rotate, 243, 244
rotateX, 245
rotateY, 246
rotateZ, 246
slerp, 247
subtract, 248, 249
gg::GgShader, 250
 ~GgShader, 253
 get, 253
 GgShader, 251, 253
 operator=, 254
 unuse, 254
 use, 254
gg::GgShape, 255
 ~GgShape, 256
 draw, 256
 get, 257
 getMode, 257
 GgShape, 256
 operator bool, 257
 operator!, 258
 setMode, 258
gg::GgSimpleObj, 258
 ~GgSimpleObj, 259
 draw, 260
 get, 260
 GgSimpleObj, 259
 operator bool, 260
 operator!, 260
gg::GgSimpleShader, 261
 ~GgSimpleShader, 264
 GgSimpleShader, 263, 264
 load, 264, 265
 loadMatrix, 265–267
 loadModelviewMatrix, 267–269
 operator=, 269
 use, 269–275
gg::GgSimpleShader::Light, 328
 ambient, 329
 diffuse, 329
 position, 329
 specular, 329
gg::GgSimpleShader::LightBuffer, 330
 ~LightBuffer, 333
 LightBuffer, 332
 load, 333
 loadAmbient, 334
 loadColor, 335
 loadDiffuse, 335, 336
 loadPosition, 336, 337
 loadSpecular, 338, 339
 select, 339
gg::GgSimpleShader::Material, 340
 ambient, 341
 diffuse, 341
 shininess, 341
 specular, 341
gg::GgSimpleShader::MaterialBuffer, 342
 ~MaterialBuffer, 345
 load, 345
 loadAmbient, 346
 loadAmbientAndDiffuse, 347, 348
 loadDiffuse, 348, 349
 loadShininess, 349, 350
 loadSpecular, 350, 351
 MaterialBuffer, 344
 select, 351
gg::GgTexture, 276
 ~GgTexture, 278
 bind, 278
 getHeight, 278
 getSize, 278, 279
 getTexture, 279
 getWidth, 279
 GgTexture, 277
 operator=, 280
 swapRandB, 281
 unbind, 281
gg::GgTrackball, 281
 ~GgTrackball, 286
 begin, 287
 end, 288
 get, 288
 getMatrix, 288
 getQuaternion, 289
 getScale, 289, 290
 getStart, 290
 GgTrackball, 286
 motion, 291
 operator=, 291
 region, 292
 reset, 293
 rotate, 294
gg::GgTriangles, 294
 ~GgTriangles, 296
 draw, 297
 getBuffer, 297
 getCount, 297
 GgTriangles, 296
 load, 298
 send, 298
gg::GgUniformBuffer< T >, 299
 ~GgUniformBuffer, 300
 bind, 301
 copy, 301
 fill, 301
 getBuffer, 302
 getCount, 302
 getStride, 302
 getTarget, 302
 GgUniformBuffer, 299, 300

load, 302, 303
 map, 303
 read, 304
 send, 304
 unbind, 305
 unmap, 305
gg::GgVector, 306
 ~GgVector, 309
 distance3, 310
 distance4, 310
 dot3, 311
 dot4, 311
GgVector, 307–309
 length3, 312
 length4, 312
 normalize3, 313
 normalize4, 313
 operator+, 316
 operator+=, 317
 operator-, 318
 operator-=, 319
 operator/, 319, 320
 operator/=, 320, 322
 operator=, 322
 operator*, 313, 314
 operator**=, 314, 316
gg::GgVertex, 323
 GgVertex, 324
 normal, 325
 position, 325
gg::GgVertexArray, 325
 ~GgVertexArray, 326
 bind, 327
 get, 327
 GgVertexArray, 326
 operator=, 327
GG_BUTTON_COUNT
 GgApp.h, 539
GG_INTERFACE_COUNT
 GgApp.h, 539
GG_USE_IMGUI
 GgApp.h, 540
GgApp, 89
 ~GgApp, 91
 GgApp, 90, 91
 main, 91
 operator=, 92, 93
GgApp.cpp, 524
GgApp.h, 538
 GG_BUTTON_COUNT, 539
 GG_INTERFACE_COUNT, 539
 GG_USE_IMGUI, 540
GgApp::Window, 354
 ~Window, 357
 get, 357
 getAltArrowX, 357
 getAltArrowY, 358
 getAltIArrow, 359
 getArrow, 359, 360
 getArrowX, 360
 getArrowY, 361
 getAspect, 362
 getControlArrow, 362
 getControlArrowX, 363
 getControlArrowY, 363
 getFboHeight, 364
 getFboSize, 364, 365
 getFboWidth, 365
 getHeight, 366
 getKey, 366
 getLastKey, 366
 getMouse, 367
 getX, 368
 getY, 368
 getRotation, 368
 getRotationMatrix, 368
 getScrollMatrix, 369
 getShiftArrow, 369
 getShiftArrowX, 370
 getShiftArrowY, 370
 getSize, 371
 getTranslation, 372
 getTranslationMatrix, 372
 getUserPointer, 373
 getWheel, 373
 getWheelX, 374
 getWheelY, 374
 getWidth, 374
 operator bool, 374
 operator=, 375
 reset, 375
 resetRotation, 376
 resetTranslation, 376
 restoreViewport, 376
 selectInterface, 377
 setClose, 377
 setKeyboardFunc, 377
 setMouseFunc, 377
 setResizeFunc, 378
 setUserPointer, 378
 setVelocity, 378
 setWheelFunc, 379
 shouldClose, 379
 swapBuffers, 379
 updateViewport, 379
 Window, 356
ggArraysObj
 gg, 19
GgBuffer
 gg::GgBuffer< T >, 94, 95
ggBufferAlignment
 gg, 84
GgColorTexture
 gg::GgColorTexture, 101, 102
ggConjugate
 gg, 20

ggCreateComputeShader
 gg, 20
ggCreateNormalMap
 gg, 21
ggCreateShader
 gg, 22
ggCross
 gg, 23
ggDistance3
 gg, 24, 25
ggDistance4
 gg, 26
ggDot3
 gg, 27, 28
ggDot4
 gg, 28, 29
GgElements
 gg::GgElements, 106
ggElementsMesh
 gg, 30
ggElementsObj
 gg, 31
ggElementsSphere
 gg, 31
ggEllipse
 gg, 32
ggError
 gg.h, 468
ggEulerQuaternion
 gg, 33, 34
ggFBOError
 gg.h, 468
ggFrustum
 gg, 35
ggIdentity
 gg, 36
ggIdentityQuaternion
 gg, 36
ggInit
 gg, 36
ggInvert
 gg, 37
ggLength3
 gg, 38, 39
ggLength4
 gg, 40
ggLoadComputeShader
 gg, 41
ggLoadHeight
 gg, 42
ggLoadImage
 gg, 42
ggLoadShader
 gg, 43, 44
ggLoadSimpleObj
 gg, 45, 46
ggLoadTexture
 gg, 46
ggLookat
 gg, 47–49
GgMatrix
 gg::GgMatrix, 112–114
ggMatrixQuaternion
 gg, 50
ggNorm
 gg, 51
ggNormal
 gg, 52
ggNormalize
 gg, 52
ggNormalize3
 gg, 53–55
ggNormalize4
 gg, 55–57
GgNormalTexture
 gg::GgNormalTexture, 166, 167
ggOrthogonal
 gg, 58
ggPerspective
 gg, 58
GgPoints
 gg::GgPoints, 170
ggPointsCube
 gg, 59
GgPointShader
 gg::GgPointShader, 174, 175
ggPointsSphere
 gg, 60
GgQuaternion
 gg::GgQuaternion, 186–188
ggQuaternionMatrix
 gg, 60
ggQuaternionTransposeMatrix
 gg, 61
ggReadImage
 gg, 62
ggRectangle
 gg, 63
ggRotate
 gg, 63–65, 67
ggRotateQuaternion
 gg, 68, 69
ggRotateX
 gg, 70
ggRotateY
 gg, 70
ggRotateZ
 gg, 71
ggsample01, 3
ggsample01.cpp, 548
 CONFIG_FILE, 548
 PROJECT_NAME, 548
ggSaveColor
 gg, 71
ggSaveDepth
 gg, 72

ggSaveTga
 gg, 73
 ggScale
 gg, 74, 75
 GgShader
 gg::GgShader, 251, 253
 GgShape
 gg::GgShape, 256
 GgSimpleObj
 gg::GgSimpleObj, 259
 GgSimpleShader
 gg::GgSimpleShader, 263, 264
 ggSlerp
 gg, 76–78
 GgTexture
 gg::GgTexture, 277
 GgTrackball
 gg::GgTrackball, 286
 ggTranslate
 gg, 79, 80
 ggTranspose
 gg, 81
 GgTriangles
 gg::GgTriangles, 296
 GgUniformBuffer
 gg::GgUniformBuffer< T >, 299, 300
 GgVector
 gg::GgVector, 307–309
 GgVertex
 gg::GgVertex, 324
 GgVertexArray
 gg::GgVertexArray, 326

 HEADER_STR
 main.cpp, 550

 invert
 gg::GgMatrix, 117
 gg::GgQuaternion, 201

 length3
 gg::GgVector, 312
 length4
 gg::GgVector, 312
 LightBindingPoint
 gg, 18
 LightBuffer
 gg::GgSimpleShader::LightBuffer, 332
 load
 Config, 87
 gg::GgColorTexture, 103, 104
 gg::GgElements, 108
 gg::GgNormalTexture, 167, 168
 gg::GgPoints, 172
 gg::GgPointShader, 176
 gg::GgSimpleShader, 264, 265
 gg::GgSimpleShader::LightBuffer, 333
 gg::GgSimpleShader::MaterialBuffer, 345
 gg::GgTriangles, 298
 gg::GgUniformBuffer< T >, 302, 303
 loadAdd
 gg::GgQuaternion, 202–204
 loadAmbient
 gg::GgSimpleShader::LightBuffer, 334
 gg::GgSimpleShader::MaterialBuffer, 346
 loadAmbientAndDiffuse
 gg::GgSimpleShader::MaterialBuffer, 347, 348
 loadColor
 gg::GgSimpleShader::LightBuffer, 335
 loadConjugate
 gg::GgQuaternion, 205
 loadDiffuse
 gg::GgSimpleShader::LightBuffer, 335, 336
 gg::GgSimpleShader::MaterialBuffer, 348, 349
 loadDivide
 gg::GgQuaternion, 207, 208, 210
 loadEuler
 gg::GgQuaternion, 211, 212
 loadFrustum
 gg::GgMatrix, 117
 loadIdentity
 gg::GgMatrix, 118
 gg::GgQuaternion, 213
 loadInvert
 gg::GgMatrix, 119
 gg::GgQuaternion, 213, 214
 loadLookat
 gg::GgMatrix, 120, 121
 loadMatrix
 gg::GgPointShader, 177, 178
 gg::GgQuaternion, 215
 gg::GgSimpleShader, 265–267
 loadModelviewMatrix
 gg::GgPointShader, 178, 179
 gg::GgSimpleShader, 267–269
 loadMultiply
 gg::GgQuaternion, 216, 217
 loadNormal
 gg::GgMatrix, 122, 123
 loadNormalize
 gg::GgQuaternion, 218, 219
 loadOrthogonal
 gg::GgMatrix, 124
 loadPerspective
 gg::GgMatrix, 125
 loadPosition
 gg::GgSimpleShader::LightBuffer, 336, 337
 loadProjectionMatrix
 gg::GgPointShader, 179, 180
 loadRotate
 gg::GgMatrix, 125–128
 gg::GgQuaternion, 220, 221
 loadRotateX
 gg::GgMatrix, 129
 gg::GgQuaternion, 222
 loadRotateY
 gg::GgMatrix, 130

gg::GgQuaternion, 222
loadRotateZ
 gg::GgMatrix, 130
 gg::GgQuaternion, 223
loadScale
 gg::GgMatrix, 131, 132
loadShininess
 gg::GgSimpleShader::MaterialBuffer, 349, 350
loadSlerp
 gg::GgQuaternion, 224–226
loadSpecular
 gg::GgSimpleShader::LightBuffer, 338, 339
 gg::GgSimpleShader::MaterialBuffer, 350, 351
loadSubtract
 gg::GgQuaternion, 226–228
loadTranslate
 gg::GgMatrix, 133, 134
loadTranspose
 gg::GgMatrix, 135, 136
lookat
 gg::GgMatrix, 136–138

main
 GgApp, 91
 main.cpp, 551
main.cpp, 550
 HEADER_STR, 550
 main, 551
map
 gg::GgBuffer< T >, 98
 gg::GgUniformBuffer< T >, 303
MaterialBindingPoint
 gg, 18
MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 344
Menu, 351
 ~Menu, 352
 Config, 89
 draw, 353
 getLight, 353
 getModel, 353
 getShader, 353
 Menu, 352
 operator=, 353
 Scene, 354
Menu.cpp, 552
Menu.h, 554
motion
 gg::GgTrackball, 291
multiply
 gg::GgQuaternion, 229, 230

norm
 gg::GgQuaternion, 231
normal
 gg::GgMatrix, 139
 gg::GgVertex, 325
normalize
 gg::GgQuaternion, 231
normalize3
 gg::GgVector, 313
normalize4
 gg::GgVector, 313

operator bool
 gg::GgPoints, 172
 gg::GgShape, 257
 gg::GgSimpleObj, 260
 GgApp::Window, 374
operator!
 gg::GgPoints, 173
 gg::GgShape, 258
 gg::GgSimpleObj, 260
operator+
 gg, 82
 gg::GgMatrix, 143, 144
 gg::GgQuaternion, 234, 235
 gg::GgVector, 316
operator+=
 gg::GgMatrix, 145
 gg::GgQuaternion, 235, 236
 gg::GgVector, 317
operator-
 gg, 82, 83
 gg::GgMatrix, 146
 gg::GgQuaternion, 236, 237
 gg::GgVector, 318
operator-=
 gg::GgMatrix, 147, 148
 gg::GgQuaternion, 237, 238
 gg::GgVector, 319
operator/
 gg, 83
 gg::GgMatrix, 148, 149
 gg::GgQuaternion, 239
 gg::GgVector, 319, 320
operator/=
 gg::GgMatrix, 150
 gg::GgQuaternion, 240, 241
 gg::GgVector, 320, 322
operator=
 gg::GgBuffer< T >, 99
 gg::GgMatrix, 151, 152
 gg::GgQuaternion, 241, 242
 gg::GgShader, 254
 gg::GgSimpleShader, 269
 gg::GgTexture, 280
 gg::GgTrackball, 291
 gg::GgVector, 322
 gg::GgVertexArray, 327
 GgApp, 92, 93
 GgApp::Window, 375
 Menu, 353
operator*
 gg, 81
 gg::GgMatrix, 140, 141
 gg::GgQuaternion, 232
 gg::GgVector, 313, 314

operator*=
 gg::GgMatrix, 142, 143
 gg::GgQuaternion, 233
 gg::GgVector, 314, 316

orthogonal
 gg::GgMatrix, 152

parseconfig.h, 556
 getString, 557, 558
 getValue, 559
 getVector, 560
 setString, 560–562
 setValue, 562
 setVector, 563

pathChar
 gg.h, 468

pathString
 gg.h, 468

perspective
 gg::GgMatrix, 153

position
 gg::GgSimpleShader::Light, 329
 gg::GgVertex, 325

PROJECT_NAME
 ggsample01.cpp, 548

projection
 gg::GgMatrix, 153, 154

read
 gg::GgBuffer< T >, 99
 gg::GgUniformBuffer< T >, 304

README.md, 566

region
 gg::GgTrackball, 292

reset
 gg::GgTrackball, 293
 GgApp::Window, 375

resetRotation
 GgApp::Window, 376

resetTranslation
 GgApp::Window, 376

restoreViewport
 GgApp::Window, 376

rotate
 gg::GgMatrix, 155–157
 gg::GgQuaternion, 243, 244
 gg::GgTrackball, 294

rotateX
 gg::GgMatrix, 158
 gg::GgQuaternion, 245

rotateY
 gg::GgMatrix, 159
 gg::GgQuaternion, 246

rotateZ
 gg::GgMatrix, 159
 gg::GgQuaternion, 246

save
 Config, 88

scale
 gg::GgMatrix, 160, 161

Scene
 Menu, 354

select
 gg::GgSimpleShader::LightBuffer, 339
 gg::GgSimpleShader::MaterialBuffer, 351

selectInterface
 GgApp::Window, 377

send
 gg::GgBuffer< T >, 100
 gg::GgElements, 109
 gg::GgPoints, 173
 gg::GgTriangles, 298
 gg::GgUniformBuffer< T >, 304

setClose
 GgApp::Window, 377

setKeyboardFunc
 GgApp::Window, 377

setMode
 gg::GgShape, 258

setMouseFunc
 GgApp::Window, 377

setResizeFunc
 GgApp::Window, 378

setString
 parseconfig.h, 560–562

setUserPointer
 GgApp::Window, 378

setValue
 parseconfig.h, 562

setVector
 parseconfig.h, 563

setVelocity
 GgApp::Window, 378

setWheelFunc
 GgApp::Window, 379

shininess
 gg::GgSimpleShader::Material, 341

shouldClose
 GgApp::Window, 379

slerp
 gg::GgQuaternion, 247

specular
 gg::GgSimpleShader::Light, 329
 gg::GgSimpleShader::Material, 341

subtract
 gg::GgQuaternion, 248, 249

swapBuffers
 GgApp::Window, 379

swapRandB
 gg::GgTexture, 281

TCharToUtf8
 gg.h, 468

translate
 gg::GgMatrix, 162, 163

transpose
 gg::GgMatrix, 164

unbind
 gg::GgBuffer< T >, [100](#)
 gg::GgTexture, [281](#)
 gg::GgUniformBuffer< T >, [305](#)

unmap
 gg::GgBuffer< T >, [100](#)
 gg::GgUniformBuffer< T >, [305](#)

unuse
 gg::GgPointShader, [180](#)
 gg::GgShader, [254](#)

updateViewport
 GgApp::Window, [379](#)

use
 gg::GgPointShader, [180–182](#)
 gg::GgShader, [254](#)
 gg::GgSimpleShader, [269–275](#)

Utf8ToTChar
 gg.h, [468](#)

Window
 GgApp::Window, [356](#)

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](#)