

ゲームグラフィックス特論

構築: Doxygen 1.9.1

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	19
7.1.2 型定義詳解	20
7.1.2.1 GgVector	20
7.1.3 列挙型詳解	20
7.1.3.1 BindingPoints	20
7.1.4 関数詳解	20
7.1.4.1 ggError()	20
7.1.4.2 ggFBOError()	21
7.1.4.3 ggArraysObj()	21
7.1.4.4 ggConjugate()	22
7.1.4.5 ggCreateComputeShader()	22
7.1.4.6 ggCreateNormalMap()	23
7.1.4.7 ggCreateShader()	24
7.1.4.8 ggCross()	25
7.1.4.9 ggDot3()	25
7.1.4.10 ggDot4() [1/2]	26
7.1.4.11 ggDot4() [2/2]	26

7.1.4.12 ggElementsMesh()	26
7.1.4.13 ggElementsObj()	27
7.1.4.14 ggElementsSphere()	28
7.1.4.15 ggEllipse()	29
7.1.4.16 ggEulerQuaternion() [1/2]	29
7.1.4.17 ggEulerQuaternion() [2/2]	30
7.1.4.18 ggFrustum()	31
7.1.4.19 ggIdentity()	31
7.1.4.20 ggIdentityQuaternion()	32
7.1.4.21 ggInit()	33
7.1.4.22 ggInvert() [1/2]	33
7.1.4.23 ggInvert() [2/2]	34
7.1.4.24 ggLength3()	34
7.1.4.25 ggLength4() [1/2]	35
7.1.4.26 ggLength4() [2/2]	36
7.1.4.27 ggLoadComputeShader()	36
7.1.4.28 ggLoadHeight()	37
7.1.4.29 ggLoadImage()	38
7.1.4.30 ggLoadShader()	38
7.1.4.31 ggLoadSimpleObj() [1/2]	39
7.1.4.32 ggLoadSimpleObj() [2/2]	40
7.1.4.33 ggLoadTexture()	41
7.1.4.34 ggLookat() [1/3]	41
7.1.4.35 ggLookat() [2/3]	42
7.1.4.36 ggLookat() [3/3]	43
7.1.4.37 ggMatrixQuaternion() [1/2]	44
7.1.4.38 ggMatrixQuaternion() [2/2]	44
7.1.4.39 ggNorm()	45
7.1.4.40 ggNormal()	46
7.1.4.41 ggNormalize()	46
7.1.4.42 ggNormalize3()	47
7.1.4.43 ggNormalize4() [1/2]	48
7.1.4.44 ggNormalize4() [2/2]	48
7.1.4.45 ggOrthogonal()	49
7.1.4.46 ggPerspective()	50
7.1.4.47 ggPointsCube()	51
7.1.4.48 ggPointsSphere()	51
7.1.4.49 ggQuaternion() [1/2]	53
7.1.4.50 ggQuaternion() [2/2]	53
7.1.4.51 ggQuaternionMatrix()	54
7.1.4.52 ggQuaternionTransposeMatrix()	55
7.1.4.53 ggReadImage()	56

7.1.4.54 ggRectangle()	57
7.1.4.55 ggRotate() [1/5]	57
7.1.4.56 ggRotate() [2/5]	58
7.1.4.57 ggRotate() [3/5]	58
7.1.4.58 ggRotate() [4/5]	59
7.1.4.59 ggRotate() [5/5]	60
7.1.4.60 ggRotateQuaternion() [1/3]	60
7.1.4.61 ggRotateQuaternion() [2/3]	61
7.1.4.62 ggRotateQuaternion() [3/3]	62
7.1.4.63 ggRotateX()	63
7.1.4.64 ggRotateY()	63
7.1.4.65 ggRotateZ()	64
7.1.4.66 ggSaveColor()	64
7.1.4.67 ggSaveDepth()	65
7.1.4.68 ggSaveTga()	66
7.1.4.69 ggScale() [1/3]	66
7.1.4.70 ggScale() [2/3]	67
7.1.4.71 ggScale() [3/3]	68
7.1.4.72 ggSlerp() [1/4]	68
7.1.4.73 ggSlerp() [2/4]	69
7.1.4.74 ggSlerp() [3/4]	70
7.1.4.75 ggSlerp() [4/4]	71
7.1.4.76 ggTranslate() [1/3]	71
7.1.4.77 ggTranslate() [2/3]	72
7.1.4.78 ggTranslate() [3/3]	73
7.1.4.79 ggTranspose()	73
7.1.5 変数詳解	74
7.1.5.1 ggBufferAlignment	74
8 クラス詳解	75
8.1 GgApp クラス	75
8.1.1 詳解	75
8.1.2 関数詳解	75
8.1.2.1 main()	75
8.2 gg::GgBuffer< T > クラステンプレート	76
8.2.1 詳解	76
8.2.2 構築子と解体子	77
8.2.2.1 GgBuffer() [1/2]	77
8.2.2.2 ~GgBuffer()	77
8.2.2.3 GgBuffer() [2/2]	77
8.2.3 関数詳解	78
8.2.3.1 bind()	78

8.2.3.2 copy()	78
8.2.3.3 getBuffer()	78
8.2.3.4 getCount()	79
8.2.3.5 getStride()	79
8.2.3.6 getTarget()	79
8.2.3.7 map() [1/2]	80
8.2.3.8 map() [2/2]	80
8.2.3.9 operator=()	80
8.2.3.10 read()	81
8.2.3.11 send()	81
8.2.3.12 unbind()	81
8.2.3.13 unmap()	82
8.3 gg::GgColorTexture クラス	82
8.3.1 詳解	82
8.3.2 構築子と解体子	82
8.3.2.1 GgColorTexture() [1/3]	83
8.3.2.2 GgColorTexture() [2/3]	83
8.3.2.3 GgColorTexture() [3/3]	84
8.3.2.4 ~GgColorTexture()	84
8.3.3 関数詳解	84
8.3.3.1 load() [1/2]	85
8.3.3.2 load() [2/2]	85
8.4 gg::GgElements クラス	86
8.4.1 詳解	87
8.4.2 構築子と解体子	87
8.4.2.1 GgElements() [1/2]	88
8.4.2.2 GgElements() [2/2]	88
8.4.2.3 ~GgElements()	88
8.4.3 関数詳解	89
8.4.3.1 draw()	89
8.4.3.2 getIndexBuffer()	89
8.4.3.3 getIndexCount()	90
8.4.3.4 load()	90
8.4.3.5 send()	90
8.5 gg::GgMatrix クラス	91
8.5.1 詳解	95
8.5.2 構築子と解体子	95
8.5.2.1 GgMatrix() [1/3]	95
8.5.2.2 GgMatrix() [2/3]	95
8.5.2.3 GgMatrix() [3/3]	96
8.5.2.4 ~GgMatrix()	96
8.5.3 関数詳解	97

8.5.3.1 add() [1/2]	97
8.5.3.2 add() [2/2]	97
8.5.3.3 divide() [1/2]	98
8.5.3.4 divide() [2/2]	99
8.5.3.5 frustum()	100
8.5.3.6 get() [1/3]	100
8.5.3.7 get() [2/3]	101
8.5.3.8 get() [3/3]	102
8.5.3.9 invert()	102
8.5.3.10 load() [1/2]	103
8.5.3.11 load() [2/2]	103
8.5.3.12 loadAdd() [1/2]	104
8.5.3.13 loadAdd() [2/2]	105
8.5.3.14 loadDivide() [1/2]	105
8.5.3.15 loadDivide() [2/2]	106
8.5.3.16 loadFrustum()	107
8.5.3.17 loadIdentity()	107
8.5.3.18 loadInvert() [1/2]	108
8.5.3.19 loadInvert() [2/2]	108
8.5.3.20 loadLookat() [1/3]	109
8.5.3.21 loadLookat() [2/3]	110
8.5.3.22 loadLookat() [3/3]	111
8.5.3.23 loadMultiply() [1/2]	112
8.5.3.24 loadMultiply() [2/2]	113
8.5.3.25 loadNormal() [1/2]	114
8.5.3.26 loadNormal() [2/2]	114
8.5.3.27 loadOrthogonal()	115
8.5.3.28 loadPerspective()	116
8.5.3.29 loadRotate() [1/5]	117
8.5.3.30 loadRotate() [2/5]	117
8.5.3.31 loadRotate() [3/5]	118
8.5.3.32 loadRotate() [4/5]	119
8.5.3.33 loadRotate() [5/5]	120
8.5.3.34 loadRotateX()	120
8.5.3.35 loadRotateY()	121
8.5.3.36 loadRotateZ()	122
8.5.3.37 loadScale() [1/3]	122
8.5.3.38 loadScale() [2/3]	123
8.5.3.39 loadScale() [3/3]	124
8.5.3.40 loadSubtract() [1/2]	124
8.5.3.41 loadSubtract() [2/2]	125
8.5.3.42 loadTranslate() [1/3]	126

8.5.3.43 loadTranslate() [2/3]	126
8.5.3.44 loadTranslate() [3/3]	127
8.5.3.45 loadTranspose() [1/2]	128
8.5.3.46 loadTranspose() [2/2]	128
8.5.3.47 lookat() [1/3]	129
8.5.3.48 lookat() [2/3]	130
8.5.3.49 lookat() [3/3]	130
8.5.3.50 multiply() [1/2]	131
8.5.3.51 multiply() [2/2]	132
8.5.3.52 normal()	132
8.5.3.53 operator*() [1/3]	133
8.5.3.54 operator*() [2/3]	134
8.5.3.55 operator*() [3/3]	134
8.5.3.56 operator*=() [1/2]	134
8.5.3.57 operator*=() [2/2]	135
8.5.3.58 operator+() [1/2]	135
8.5.3.59 operator+() [2/2]	136
8.5.3.60 operator+=() [1/2]	136
8.5.3.61 operator+=() [2/2]	137
8.5.3.62 operator-() [1/2]	137
8.5.3.63 operator-() [2/2]	138
8.5.3.64 operator-=() [1/2]	138
8.5.3.65 operator-=() [2/2]	139
8.5.3.66 operator/() [1/2]	139
8.5.3.67 operator/() [2/2]	140
8.5.3.68 operator/=() [1/2]	140
8.5.3.69 operator/=() [2/2]	141
8.5.3.70 operator=() [1/2]	141
8.5.3.71 operator=() [2/2]	142
8.5.3.72 operator[]() [1/2]	142
8.5.3.73 operator[]() [2/2]	143
8.5.3.74 orthogonal()	143
8.5.3.75 perspective()	144
8.5.3.76 projection() [1/4]	144
8.5.3.77 projection() [2/4]	145
8.5.3.78 projection() [3/4]	145
8.5.3.79 projection() [4/4]	145
8.5.3.80 rotate() [1/5]	146
8.5.3.81 rotate() [2/5]	146
8.5.3.82 rotate() [3/5]	147
8.5.3.83 rotate() [4/5]	148
8.5.3.84 rotate() [5/5]	148

8.5.3.85 rotateX()	149
8.5.3.86 rotateY()	150
8.5.3.87 rotateZ()	150
8.5.3.88 scale() [1/3]	151
8.5.3.89 scale() [2/3]	152
8.5.3.90 scale() [3/3]	152
8.5.3.91 subtract() [1/2]	153
8.5.3.92 subtract() [2/2]	154
8.5.3.93 translate() [1/3]	154
8.5.3.94 translate() [2/3]	155
8.5.3.95 translate() [3/3]	156
8.5.3.96 transpose()	157
8.5.4 フレンドと関連関数の詳解	157
8.5.4.1 GgQuaternion	157
8.6 gg::GgNormalTexture クラス	158
8.6.1 詳解	158
8.6.2 構築子と解体子	158
8.6.2.1 GgNormalTexture() [1/3]	158
8.6.2.2 GgNormalTexture() [2/3]	158
8.6.2.3 GgNormalTexture() [3/3]	159
8.6.2.4 ~GgNormalTexture()	160
8.6.3 関数詳解	160
8.6.3.1 load() [1/2]	160
8.6.3.2 load() [2/2]	161
8.7 gg::GgPoints クラス	162
8.7.1 詳解	163
8.7.2 構築子と解体子	163
8.7.2.1 GgPoints() [1/2]	163
8.7.2.2 GgPoints() [2/2]	163
8.7.2.3 ~GgPoints()	164
8.7.3 関数詳解	164
8.7.3.1 draw()	164
8.7.3.2 getBuffer()	165
8.7.3.3 getCount()	165
8.7.3.4 load()	165
8.7.3.5 send()	166
8.8 gg::GgPointShader クラス	166
8.8.1 詳解	167
8.8.2 構築子と解体子	167
8.8.2.1 GgPointShader() [1/2]	168
8.8.2.2 GgPointShader() [2/2]	168
8.8.2.3 ~GgPointShader()	168

8.8.3 関数詳解	168
8.8.3.1 get()	169
8.8.3.2 load()	169
8.8.3.3 loadMatrix() [1/2]	170
8.8.3.4 loadMatrix() [2/2]	170
8.8.3.5 loadModelviewMatrix() [1/2]	171
8.8.3.6 loadModelviewMatrix() [2/2]	171
8.8.3.7 loadProjectionMatrix() [1/2]	172
8.8.3.8 loadProjectionMatrix() [2/2]	172
8.8.3.9 unuse()	172
8.8.3.10 use() [1/5]	173
8.8.3.11 use() [2/5]	173
8.8.3.12 use() [3/5]	173
8.8.3.13 use() [4/5]	174
8.8.3.14 use() [5/5]	174
8.9 gg::GgQuaternion クラス	175
8.9.1 詳解	178
8.9.2 構築子と解体子	179
8.9.2.1 GgQuaternion() [1/5]	179
8.9.2.2 GgQuaternion() [2/5]	179
8.9.2.3 GgQuaternion() [3/5]	179
8.9.2.4 GgQuaternion() [4/5]	181
8.9.2.5 GgQuaternion() [5/5]	181
8.9.2.6 ~GgQuaternion()	182
8.9.3 関数詳解	182
8.9.3.1 add() [1/4]	182
8.9.3.2 add() [2/4]	183
8.9.3.3 add() [3/4]	183
8.9.3.4 add() [4/4]	184
8.9.3.5 conjugate()	185
8.9.3.6 divide() [1/4]	185
8.9.3.7 divide() [2/4]	186
8.9.3.8 divide() [3/4]	187
8.9.3.9 divide() [4/4]	187
8.9.3.10 euler() [1/2]	188
8.9.3.11 euler() [2/2]	189
8.9.3.12 get() [1/2]	190
8.9.3.13 get() [2/2]	190
8.9.3.14 getConjugateMatrix() [1/3]	190
8.9.3.15 getConjugateMatrix() [2/3]	191
8.9.3.16 getConjugateMatrix() [3/3]	191
8.9.3.17 getMatrix() [1/3]	192

8.9.3.18 getMatrix() [2/3]	192
8.9.3.19 getMatrix() [3/3]	193
8.9.3.20 invert()	194
8.9.3.21 load() [1/4]	194
8.9.3.22 load() [2/4]	195
8.9.3.23 load() [3/4]	195
8.9.3.24 load() [4/4]	196
8.9.3.25 loadAdd() [1/4]	197
8.9.3.26 loadAdd() [2/4]	198
8.9.3.27 loadAdd() [3/4]	198
8.9.3.28 loadAdd() [4/4]	199
8.9.3.29 loadConjugate() [1/2]	200
8.9.3.30 loadConjugate() [2/2]	200
8.9.3.31 loadDivide() [1/4]	201
8.9.3.32 loadDivide() [2/4]	202
8.9.3.33 loadDivide() [3/4]	202
8.9.3.34 loadDivide() [4/4]	203
8.9.3.35 loadEuler() [1/2]	204
8.9.3.36 loadEuler() [2/2]	204
8.9.3.37 loadIdentity()	205
8.9.3.38 loadInvert() [1/2]	206
8.9.3.39 loadInvert() [2/2]	207
8.9.3.40 loadMatrix() [1/2]	208
8.9.3.41 loadMatrix() [2/2]	209
8.9.3.42 loadMultiply() [1/4]	210
8.9.3.43 loadMultiply() [2/4]	211
8.9.3.44 loadMultiply() [3/4]	211
8.9.3.45 loadMultiply() [4/4]	212
8.9.3.46 loadNormalize() [1/2]	213
8.9.3.47 loadNormalize() [2/2]	213
8.9.3.48 loadRotate() [1/3]	214
8.9.3.49 loadRotate() [2/3]	214
8.9.3.50 loadRotate() [3/3]	215
8.9.3.51 loadRotateX()	216
8.9.3.52 loadRotateY()	217
8.9.3.53 loadRotateZ()	217
8.9.3.54 loadSlerp() [1/4]	218
8.9.3.55 loadSlerp() [2/4]	219
8.9.3.56 loadSlerp() [3/4]	219
8.9.3.57 loadSlerp() [4/4]	220
8.9.3.58 loadSubtract() [1/4]	221
8.9.3.59 loadSubtract() [2/4]	221

8.9.3.60 loadSubtract() [3/4]	222
8.9.3.61 loadSubtract() [4/4]	222
8.9.3.62 multiply() [1/4]	223
8.9.3.63 multiply() [2/4]	223
8.9.3.64 multiply() [3/4]	224
8.9.3.65 multiply() [4/4]	224
8.9.3.66 norm()	225
8.9.3.67 normalize()	226
8.9.3.68 operator*() [1/2]	226
8.9.3.69 operator*() [2/2]	227
8.9.3.70 operator*=() [1/2]	227
8.9.3.71 operator*=() [2/2]	227
8.9.3.72 operator+() [1/2]	228
8.9.3.73 operator+() [2/2]	228
8.9.3.74 operator+=() [1/2]	229
8.9.3.75 operator+=() [2/2]	229
8.9.3.76 operator-() [1/2]	230
8.9.3.77 operator-() [2/2]	230
8.9.3.78 operator-=() [1/2]	231
8.9.3.79 operator-=() [2/2]	231
8.9.3.80 operator/() [1/2]	232
8.9.3.81 operator/() [2/2]	232
8.9.3.82 operator/=() [1/2]	233
8.9.3.83 operator/=() [2/2]	233
8.9.3.84 operator=() [1/2]	234
8.9.3.85 operator=() [2/2]	234
8.9.3.86 rotate() [1/3]	235
8.9.3.87 rotate() [2/3]	235
8.9.3.88 rotate() [3/3]	236
8.9.3.89 rotateX()	237
8.9.3.90 rotateY()	238
8.9.3.91 rotateZ()	238
8.9.3.92 slerp() [1/2]	239
8.9.3.93 slerp() [2/2]	239
8.9.3.94 subtract() [1/4]	240
8.9.3.95 subtract() [2/4]	240
8.9.3.96 subtract() [3/4]	241
8.9.3.97 subtract() [4/4]	242
8.10 gg::GgShader クラス	242
8.10.1 詳解	243
8.10.2 構築子と解体子	243
8.10.2.1 GgShader() [1/2]	243

8.10.2.2 ~GgShader()	244
8.10.2.3 GgShader() [2/2]	244
8.10.3 関数詳解	244
8.10.3.1 get()	244
8.10.3.2 operator=()	244
8.10.3.3 unuse()	245
8.10.3.4 use()	245
8.11 gg::GgShape クラス	245
8.11.1 詳解	246
8.11.2 構築子と解体子	246
8.11.2.1 GgShape() [1/2]	246
8.11.2.2 ~GgShape()	247
8.11.2.3 GgShape() [2/2]	247
8.11.3 関数詳解	247
8.11.3.1 draw()	247
8.11.3.2 get()	248
8.11.3.3 getMode()	248
8.11.3.4 operator=()	249
8.11.3.5 setMode()	249
8.12 gg::GgSimpleObj クラス	249
8.12.1 詳解	249
8.12.2 構築子と解体子	250
8.12.2.1 GgSimpleObj()	250
8.12.2.2 ~GgSimpleObj()	250
8.12.3 関数詳解	250
8.12.3.1 draw()	250
8.12.3.2 get()	251
8.13 gg::GgSimpleShader クラス	251
8.13.1 詳解	254
8.13.2 構築子と解体子	254
8.13.2.1 GgSimpleShader() [1/3]	254
8.13.2.2 GgSimpleShader() [2/3]	254
8.13.2.3 GgSimpleShader() [3/3]	255
8.13.2.4 ~GgSimpleShader()	255
8.13.3 関数詳解	255
8.13.3.1 load()	255
8.13.3.2 loadMatrix() [1/4]	256
8.13.3.3 loadMatrix() [2/4]	257
8.13.3.4 loadMatrix() [3/4]	257
8.13.3.5 loadMatrix() [4/4]	258
8.13.3.6 loadModelviewMatrix() [1/4]	258
8.13.3.7 loadModelviewMatrix() [2/4]	259

8.13.3.8 loadModelviewMatrix() [3/4]	259
8.13.3.9 loadModelviewMatrix() [4/4]	259
8.13.3.10 operator=()	260
8.13.3.11 use() [1/13]	260
8.13.3.12 use() [2/13]	260
8.13.3.13 use() [3/13]	261
8.13.3.14 use() [4/13]	261
8.13.3.15 use() [5/13]	262
8.13.3.16 use() [6/13]	262
8.13.3.17 use() [7/13]	263
8.13.3.18 use() [8/13]	263
8.13.3.19 use() [9/13]	264
8.13.3.20 use() [10/13]	264
8.13.3.21 use() [11/13]	264
8.13.3.22 use() [12/13]	265
8.13.3.23 use() [13/13]	265
8.14 gg::GgTexture クラス	266
8.14.1 詳解	266
8.14.2 構築子と解体子	266
8.14.2.1 GgTexture() [1/2]	267
8.14.2.2 ~GgTexture()	267
8.14.2.3 GgTexture() [2/2]	267
8.14.3 関数詳解	267
8.14.3.1 bind()	268
8.14.3.2 getHeight()	268
8.14.3.3 getSize() [1/2]	268
8.14.3.4 getSize() [2/2]	268
8.14.3.5 getTexture()	269
8.14.3.6 getWidth()	269
8.14.3.7 operator=()	270
8.14.3.8 unbind()	270
8.15 gg::GgTrackball クラス	270
8.15.1 詳解	271
8.15.2 構築子と解体子	271
8.15.2.1 GgTrackball()	272
8.15.2.2 ~GgTrackball()	272
8.15.3 関数詳解	272
8.15.3.1 begin()	272
8.15.3.2 end()	273
8.15.3.3 get()	273
8.15.3.4 getMatrix()	274
8.15.3.5 getQuaternion()	274

8.15.3.6 getScale() [1/3]	274
8.15.3.7 getScale() [2/3]	274
8.15.3.8 getScale() [3/3]	275
8.15.3.9 getStart() [1/3]	275
8.15.3.10 getStart() [2/3]	275
8.15.3.11 getStart() [3/3]	276
8.15.3.12 motion()	276
8.15.3.13 region() [1/2]	277
8.15.3.14 region() [2/2]	277
8.15.3.15 reset()	278
8.15.3.16 rotate()	278
8.16 gg::GgTriangles クラス	279
8.16.1 詳解	280
8.16.2 構築子と解体子	280
8.16.2.1 GgTriangles() [1/2]	280
8.16.2.2 GgTriangles() [2/2]	281
8.16.2.3 ~GgTriangles()	281
8.16.3 関数詳解	281
8.16.3.1 draw()	281
8.16.3.2 getBuffer()	282
8.16.3.3 getCount()	282
8.16.3.4 load()	282
8.16.3.5 send()	283
8.17 gg::GgUniformBuffer< T > クラステンプレート	283
8.17.1 詳解	284
8.17.2 構築子と解体子	284
8.17.2.1 GgUniformBuffer() [1/3]	285
8.17.2.2 GgUniformBuffer() [2/3]	285
8.17.2.3 GgUniformBuffer() [3/3]	285
8.17.2.4 ~GgUniformBuffer()	286
8.17.3 関数詳解	286
8.17.3.1 bind()	286
8.17.3.2 copy()	286
8.17.3.3 fill()	286
8.17.3.4 getBuffer()	287
8.17.3.5 getCount()	287
8.17.3.6 getStride()	288
8.17.3.7 getTarget()	288
8.17.3.8 load() [1/2]	288
8.17.3.9 load() [2/2]	289
8.17.3.10 map() [1/2]	289
8.17.3.11 map() [2/2]	289

8.17.3.12 read()	290
8.17.3.13 send()	290
8.17.3.14 unbind()	291
8.17.3.15 unmap()	291
8.18 gg::GgVertex 構造体	291
8.18.1 詳解	292
8.18.2 構築子と解体子	292
8.18.2.1 GgVertex() [1/4]	292
8.18.2.2 GgVertex() [2/4]	292
8.18.2.3 GgVertex() [3/4]	292
8.18.2.4 GgVertex() [4/4]	293
8.18.3 メンバ詳解	293
8.18.3.1 normal	293
8.18.3.2 position	294
8.19 gg::GgSimpleShader::Light 構造体	294
8.19.1 詳解	294
8.19.2 メンバ詳解	294
8.19.2.1 ambient	294
8.19.2.2 diffuse	295
8.19.2.3 position	295
8.19.2.4 specular	295
8.20 gg::GgSimpleShader::LightBuffer クラス	295
8.20.1 詳解	297
8.20.2 構築子と解体子	297
8.20.2.1 LightBuffer() [1/2]	297
8.20.2.2 LightBuffer() [2/2]	297
8.20.2.3 ~LightBuffer()	298
8.20.3 関数詳解	298
8.20.3.1 load() [1/2]	298
8.20.3.2 load() [2/2]	298
8.20.3.3 loadAmbient() [1/3]	299
8.20.3.4 loadAmbient() [2/3]	299
8.20.3.5 loadAmbient() [3/3]	300
8.20.3.6 loadColor()	300
8.20.3.7 loadDiffuse() [1/3]	301
8.20.3.8 loadDiffuse() [2/3]	301
8.20.3.9 loadDiffuse() [3/3]	301
8.20.3.10 loadPosition() [1/4]	302
8.20.3.11 loadPosition() [2/4]	302
8.20.3.12 loadPosition() [3/4]	303
8.20.3.13 loadPosition() [4/4]	303
8.20.3.14 loadSpecular() [1/3]	304

8.20.3.15 loadSpecular() [2/3]	304
8.20.3.16 loadSpecular() [3/3]	304
8.20.3.17 select()	305
8.21 gg::GgSimpleShader::Material 構造体	305
8.21.1 詳解	306
8.21.2 メンバ詳解	306
8.21.2.1 ambient	306
8.21.2.2 diffuse	306
8.21.2.3 shininess	306
8.21.2.4 specular	306
8.22 gg::GgSimpleShader::MaterialBuffer クラス	307
8.22.1 詳解	308
8.22.2 構築子と解体子	308
8.22.2.1 MaterialBuffer() [1/2]	308
8.22.2.2 MaterialBuffer() [2/2]	309
8.22.2.3 ~MaterialBuffer()	309
8.22.3 関数詳解	309
8.22.3.1 load() [1/2]	309
8.22.3.2 load() [2/2]	310
8.22.3.3 loadAmbient() [1/2]	310
8.22.3.4 loadAmbient() [2/2]	310
8.22.3.5 loadAmbientAndDiffuse() [1/2]	311
8.22.3.6 loadAmbientAndDiffuse() [2/2]	311
8.22.3.7 loadDiffuse() [1/2]	312
8.22.3.8 loadDiffuse() [2/2]	312
8.22.3.9 loadShininess() [1/2]	313
8.22.3.10 loadShininess() [2/2]	313
8.22.3.11 loadSpecular() [1/2]	314
8.22.3.12 loadSpecular() [2/2]	314
8.22.3.13 select()	314
8.23 Window クラス	315
8.23.1 詳解	317
8.23.2 構築子と解体子	317
8.23.2.1 Window() [1/2]	318
8.23.2.2 Window() [2/2]	318
8.23.2.3 ~Window()	318
8.23.3 関数詳解	319
8.23.3.1 get()	319
8.23.3.2 getAltArrowX()	319
8.23.3.3 getAltArrowY()	319
8.23.3.4 getAltArrow()	319
8.23.3.5 getArrow() [1/2]	320

8.23.3.6 getArrow() [2/2]	320
8.23.3.7 getArrowX()	321
8.23.3.8 getArrowY()	321
8.23.3.9 getAspect()	321
8.23.3.10 getControlArrow()	322
8.23.3.11 getControlArrowX()	322
8.23.3.12 getControlArrowY()	322
8.23.3.13 getHeight()	323
8.23.3.14 getKey()	323
8.23.3.15 getMouse() [1/3]	323
8.23.3.16 getMouse() [2/3]	323
8.23.3.17 getMouse() [3/3]	324
8.23.3.18 getMouseX()	324
8.23.3.19 getMouseY()	324
8.23.3.20 getRotation()	325
8.23.3.21 getRotationMatrix()	325
8.23.3.22 getScrollMatrix()	325
8.23.3.23 getShiftArrow()	326
8.23.3.24 getShiftArrowX()	326
8.23.3.25 getShiftArrowY()	326
8.23.3.26 getSize() [1/2]	327
8.23.3.27 getSize() [2/2]	327
8.23.3.28 getTranslation()	327
8.23.3.29 getTranslationMatrix()	328
8.23.3.30 getUserPointer()	328
8.23.3.31 getWheel() [1/3]	328
8.23.3.32 getWheel() [2/3]	328
8.23.3.33 getWheel() [3/3]	329
8.23.3.34 getWheelX()	329
8.23.3.35 getWheelY()	329
8.23.3.36 getWidth()	330
8.23.3.37 init()	330
8.23.3.38 operator bool()	330
8.23.3.39 operator=()	331
8.23.3.40 reset()	331
8.23.3.41 resetRotation()	331
8.23.3.42 resetTranslation()	331
8.23.3.43 restoreViewport()	331
8.23.3.44 selectInterface()	331
8.23.3.45 setClose()	332
8.23.3.46 setKeyboardFunc()	332
8.23.3.47 setMouseFunc()	332

8.23.3.48 setResizeFunc() [1/2]	333
8.23.3.49 setResizeFunc() [2/2]	333
8.23.3.50 setUserPointer()	333
8.23.3.51 setVelocity()	334
8.23.3.52 shouldClose()	334
8.23.3.53 swapBuffers()	334
9 ファイル詳解	335
9.1 gg.cpp ファイル	335
9.1.1 詳解	335
9.2 gg.h ファイル	336
9.2.1 詳解	341
9.3 GgApp.h ファイル	341
9.3.1 マクロ定義詳解	342
9.3.1.1 USE_IMGUI	342
9.4 ggsample01.cpp ファイル	342
9.5 main.cpp ファイル	343
9.5.1 マクロ定義詳解	344
9.5.1.1 HEADER_STR	344
9.5.2 関数詳解	344
9.5.2.1 main()	344
9.6 README.md ファイル	344
9.7 Window.h ファイル	344
9.7.1 詳解	345
9.7.2 変数詳解	346
9.7.2.1 BUTTON_COUNT	346
9.7.2.2 INTERFACE_COUNT	346
Index	347

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

Copyright (c) 2011-2021 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 18.3 / Windows 10 (Visual Studio 2017) / macOS 10.15 (Xcode 11) 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。

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／実行できなかつたら知らせてください。
- fork 推奨ですか？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

Doxygen 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、macOS 用、および Linux 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、宿題のために別に用意する必要はありません。`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、マウスカーソルが Dear ImGui のウィンドウ上にあるとき(IsAnyWindowHovered() == 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 のいずれかのウィンドウが選択されているとき(IsAnyWindowFocused() == 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
        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_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 は使用している OpenGL のローダを自動判別するのですが、この授業オリジナルの [gg.h](#) / [gg.cpp](#) は見つけてくれません。そこで、この中の imconfig.h の**最後**に、以下の定義を追加しています。

```
// My custom OpenGL loader  
#define IMGUI_IMPL_OPENGL_LOADER_CUSTOM ".../gg.h"
```

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

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 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。

GgApp	75
gg::GgBuffer< T >	76
gg::GgColorTexture	82
gg::GgMatrix	91
gg::GgNormalTexture	158
gg::GgPointShader	166
gg::GgSimpleShader	251
gg::GgQuaternion	175
gg::GgShader	242
gg::GgShape	245
gg::GgPoints	162
gg::GgTriangles	279
gg::GgElements	86
gg::GgSimpleObj	249
gg::GgTexture	266
gg::GgTrackball	270
gg::GgUniformBuffer< T >	283
gg::GgUniformBuffer< Light >	283
gg::GgSimpleShader::LightBuffer	295
gg::GgUniformBuffer< Material >	283
gg::GgSimpleShader::MaterialBuffer	307
gg::GgVertex	291
gg::GgSimpleShader::Light	294
gg::GgSimpleShader::Material	305
Window	315

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。

GgApp	75
gg::GgBuffer< T >	
バッファオブジェクト	76
gg::GgColorTexture	
カラーマップ	82
gg::GgElements	
三角形で表した形状データ (Elements 形式)	86
gg::GgMatrix	
変換行列	91
gg::GgNormalTexture	
法線マップ	158
gg::GgPoints	
点	162
gg::GgPointShader	
点のシェーダ	166
gg::GgQuaternion	
四元数	175
gg::GgShader	
シェーダの基底クラス	242
gg::GgShape	
形状データの基底クラス	245
gg::GgSimpleObj	
Wavefront OBJ 形式のファイル (Arrays 形式)	249
gg::GgSimpleShader	
三角形に単純な陰影付けを行うシェーダ	251
gg::GgTexture	
テクスチャ	266
gg::GgTrackball	
簡易トラックボール処理	270
gg::GgTriangles	
三角形で表した形状データ (Arrays 形式)	279
gg::GgUniformBuffer< T >	
ユニフォームバッファオブジェクト	283
gg::GgVertex	
三角形の頂点データ	291

gg::GgSimpleShader::Light	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	294
gg::GgSimpleShader::LightBuffer	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	295
gg::GgSimpleShader::Material	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	305
gg::GgSimpleShader::MaterialBuffer	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	307
Window	ウィンドウ関連の処理	315

Chapter 6

ファイル索引

6.1 ファイル一覧

ファイル一覧です。

gg.cpp	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	335
gg.h	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	336
GgApp.h		341
ggsample01.cpp		342
main.cpp		343
Window.h	ゲームグラフィックス特論の宿題用補助プログラムのウィンドウ関連の処理	344

Chapter 7

名前空間詳解

7.1 gg 名前空間

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

クラス

- class **GgMatrix**
変換行列.
- class **GgQuaternion**
四元数.
- class **GgTrackball**
簡易トラックボール処理.
- class **GgTexture**
テクスチャ.
- class **GgColorTexture**
カラーマップ.
- class **GgNormalTexture**
法線マップ.
- class **GgBuffer**
バッファオブジェクト.
- class **GgUniformBuffer**
ユニフォームバッファオブジェクト.
- class **GgShape**
形状データの基底クラス.
 - class **GgPoints**
点.
 - struct **GgVertex**
三角形の頂点データ.
 - class **GgTriangles**
三角形で表した形状データ (*Arrays* 形式).
 - class **GgElements**
三角形で表した形状データ (*Elements* 形式).
 - class **GgShader**
シェーダの基底クラス.

- class `GgPointShader`
点のシェーダ.
- class `GgSimpleShader`
三角形に単純な陰影付けを行うシェーダ.
- class `GgSimpleObj`
Wavefront OBJ 形式のファイル (*Arrays* 形式).

型定義

- using `GgVector` = std::array< GLfloat, 4 >
4要素の单精度実数の配列.

列挙型

- enum `BindingPoints` { `LightBindingPoint` = 0 , `MaterialBindingPoint` }
光源と材質の *uniform buffer object* の結合ポイント.

関数

- void `ggInit` ()
ゲームグラフィックス特論の都合にもとづく初期化を行う.
- void `_ggError` (const std::string &name="", unsigned int line=0)
OpenGL のエラーをチェックする.
- void `_ggFBOError` (const std::string &name="", unsigned int line=0)
FBO のエラーをチェックする.
- bool `ggSaveTga` (const std::string &name, const void *buffer, unsigned int width, unsigned int height, unsigned int depth)
配列の内容を *TGA* ファイルに保存する.
- bool `ggSaveColor` (const std::string &name)
カラーバッファの内容を *TGA* ファイルに保存する.
- bool `ggSaveDepth` (const std::string &name)
デプスバッファの内容を *TGA* ファイルに保存する.
- bool `ggReadImage` (const std::string &name, std::vector< GLubyte > &image, GLsizei *pWidth, GLsizei *pHeight, GLenum *pFormat)
TGA ファイル (8/16/24/32bit) をメモリに読み込む.
- GLuint `ggLoadTexture` (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_BGR, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE)
テクスチャメモリを確保して画像データをテクスチャとして読み込む.
- GLuint `ggLoadImage` (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE)
テクスチャメモリを確保して *TGA* 画像ファイルを読み込む.
- void `ggCreateNormalMap` (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)
グレースケール画像 (8bit) から法線マップのデータを作成する.
- GLuint `ggLoadHeight` (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)
テクスチャメモリを確保して *TGA* 画像ファイルを読み込み法線マップを作成する.
- 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")

- 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.
- GLuint **ggLoadShader** (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)
シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.
 - GLuint **ggCreateComputeShader** (const std::string &csrc, const std::string &ctext="compute shader")
コンピュートシェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する.
 - GLuint **ggLoadComputeShader** (const std::string &comp)
コンピュートシェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.
 - GLfloat **ggLength3** (const GLfloat *a)
3要素の長さ.
 - void **ggNormalize3** (GLfloat *a)
3要素の正規化.
 - GLfloat **ggDot3** (const GLfloat *a, const GLfloat *b)
3要素の内積.
 - void **ggCross** (GLfloat *c, const GLfloat *a, const GLfloat *b)
3要素の外積.
 - GLfloat **ggLength4** (const GLfloat *a)
4要素の長さ.
 - GLfloat **ggLength4** (const GgVector &a)
*GgVector*型の長さ.
 - void **ggNormalize4** (GLfloat *a)
4要素の正規化.
 - void **ggNormalize4** (GgVector &a)
*GgVector*型の正規化.
 - GLfloat **ggDot4** (const GLfloat *a, const GLfloat *b)
4要素の内積
 - GLfloat **ggDot4** (const GgVector &a, const GgVector &b)
*GgVector*型の内積
 - **GgMatrix ggIdentity** ()
単位行列を返す.
 - **GgMatrix ggTranslate** (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
平行移動の変換行列を返す.
 - **GgMatrix ggTranslate** (const GLfloat *t)
平行移動の変換行列を返す.
 - **GgMatrix ggTranslate** (const GgVector &t)
平行移動の変換行列を返す.
 - **GgMatrix ggScale** (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
拡大縮小の変換行列を返す.
 - **GgMatrix ggScale** (const GLfloat *s)
拡大縮小の変換行列を返す.
 - **GgMatrix ggScale** (const GgVector &s)
拡大縮小の変換行列を返す.
 - **GgMatrix ggRotateX** (GLfloat a)
*x*軸中心の回転の変換行列を返す.
 - **GgMatrix ggRotateY** (GLfloat a)
*y*軸中心の回転の変換行列を返す.
 - **GgMatrix ggRotateZ** (GLfloat a)
*z*軸中心の回転の変換行列を返す.
 - **GgMatrix ggRotate** (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乗じた結果を返す.
 - **GgMatrix ggRotate** (const GLfloat *r, GLfloat a)

- **GgMatrix ggRotate** (const **GgVector** &r, GLfloat a)
 - 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.
- **GgMatrix ggRotate** (const GLfloat *r)
 - 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.
- **GgMatrix ggRotate** (const **GgVector** &r)
 - 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.
- **GgMatrix ggLookat** (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)
 - ビュー変換行列を返す.
- **GgMatrix ggLookat** (const GLfloat *e, const GLfloat *t, const GLfloat *u)
 - ビュー変換行列を返す.
- **GgMatrix ggLookat** (const **GgVector** &e, const **GgVector** &t, const **GgVector** &u)
 - ビュー変換行列を返す.
- **GgMatrix ggOrthogonal** (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
 - 直交投影変換行列を返す.
- **GgMatrix ggFrustum** (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
 - 透視透視投影変換行列を返す.
- **GgMatrix ggPerspective** (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)
 - 画角を指定して透視投影変換行列を返す.
- **GgMatrix ggTranspose** (const **GgMatrix** &m)
 - 転置行列を返す.
- **GgMatrix ggInvert** (const **GgMatrix** &m)
 - 逆行列を返す.
- **GgMatrix ggNormal** (const **GgMatrix** &m)
 - 法線変換行列を返す.
- **GgQuaternion ggQuaternion** (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
 - 四元数を返す
- **GgQuaternion ggQuaternion** (const GLfloat *a)
 - 四元数を返す
- **GgQuaternion ggIdentityQuaternion** ()
 - 単位四元数を返す
- **GgQuaternion ggMatrixQuaternion** (const GLfloat *a)
 - 回転の変換行列 m を表す四元数を返す.
- **GgQuaternion ggMatrixQuaternion** (const **GgMatrix** &m)
 - 回転の変換行列 m を表す四元数を返す.
- **GgMatrix ggQuaternionMatrix** (const **GgQuaternion** &q)
 - 四元数 q の回転の変換行列を返す.
- **GgMatrix ggQuaternionTransposeMatrix** (const **GgQuaternion** &q)
 - 四元数 q の回転の転置した変換行列を返す.
- **GgQuaternion ggRotateQuaternion** (GLfloat x, GLfloat y, GLfloat z, GLfloat a)
 - (x, y, z) を軸として角度 a 回転する四元数を返す.
- **GgQuaternion ggRotateQuaternion** (const GLfloat *v, GLfloat a)
 - (v[0], v[1], v[2]) を軸として角度 a 回転する四元数を返す.
- **GgQuaternion ggRotateQuaternion** (const GLfloat *v)
 - (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.
- **GgQuaternion ggEulerQuaternion** (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返す.
- **GgQuaternion ggEulerQuaternion** (const GLfloat *e)
 - 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.
- **GgQuaternion ggSlerp** (const GLfloat *a, const GLfloat *b, GLfloat t)
 -

- 二つの四元数の球面線形補間の結果を返す.
- `GgQuaternion ggSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
二つの四元数の球面線形補間の結果を返す.
- `GgQuaternion ggSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
二つの四元数の球面線形補間の結果を返す.
- `GgQuaternion ggSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
二つの四元数の球面線形補間の結果を返す.
- `GLfloat ggNorm (const GgQuaternion &q)`
四元数のノルムを返す.
- `GgQuaternion ggNormalize (const GgQuaternion &q)`
正規化した四元数を返す.
- `GgQuaternion ggConjugate (const GgQuaternion &q)`
共役四元数を返す.
- `GgQuaternion ggInvert (const GgQuaternion &q)`
四元数の逆元を求める.
- `GgPoints * ggPointsCube (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
点群を立方体状に生成する.
- `GgPoints * ggPointsSphere (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
点群を球状に生成する.
- `GgTriangles * ggRectangle (GLfloat width=1.0f, GLfloat height=1.0f)`
矩形状に 2 枚の三角形を生成する.
- `GgTriangles * ggEllipse (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)`
楕円状に三角形を生成する.
- `GgTriangles * ggArraysObj (const std::string &name, bool normalize=false)`
Wavefront OBJ ファイルを読み込む (*Arrays* 形式)
- `GgElements * ggElementsObj (const std::string &name, bool normalize=false)`
Wavefront OBJ ファイルを読み込む (*Elements* 形式).
- `GgElements * ggElementsMesh (GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3]=nullptr)`
メッシュ形状を作成する (*Elements* 形式).
- `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)`
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)
- `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)`
三角形分割された *OBJ* ファイルを読み込む (*Elements* 形式).

変数

- `GLint ggBufferAlignment`
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.

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 型定義詳解

7.1.2.1 GgVector

```
using gg::GgVector = typedef std::array<GLfloat, 4>
```

4要素の単精度実数の配列。

gg.h の 1340 行目に定義があります。

7.1.3 列挙型詳解

7.1.3.1 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 1326 行目に定義があります。

7.1.4 関数詳解

7.1.4.1 ggError()

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

OpenGL のエラーをチェックする。

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。

引数

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

gg.cpp の 2564 行目に定義がります。

7.1.4.2 `_ggFBOError()`

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

FBO のエラーをチェックする.

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.

引数

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

gg.cpp の 2608 行目に定義がります。

7.1.4.3 `ggArraysObj()`

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

Wavefront OBJ ファイルを読み込む (Arrays 形式)

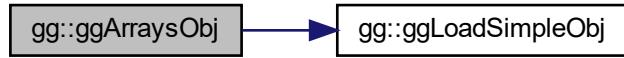
三角形分割された Wavefront OBJ ファイルを読み込んで GgArrays 形式の三角形データを生成する.

引数

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

gg.cpp の 5225 行目に定義がります。

呼び出し関係図:



7.1.4.4 ggConjugate()

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

共役四元数を返す。

引数

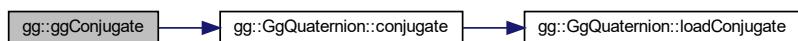
<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

四元数 `q` の共役四元数。

`gg.h` の 3802 行目に定義があります。

呼び出し関係図:



7.1.4.5 ggCreateComputeShader()

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

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

引数

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

gg.cpp の 4210 行目に定義があります。

7.1.4.6 ggCreateNormalMap()

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

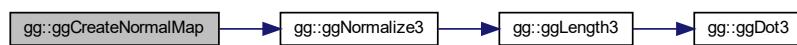
グレースケール画像(8bit)から法線マップのデータを作成する.

引数

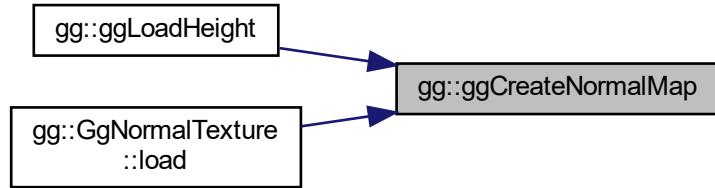
<i>hmap</i>	グレースケール画像のデータ.
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数.
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数.
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_BGR, GL_RGBA, GL_BGRA).
<i>nz</i>	法線の z 成分の割合.
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット.
<i>nmap</i>	法線マップを格納する vector.

gg.cpp の 3023 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:



7.1.4.7 ggCreateShader()

```

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

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

引数

<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 (<i>nullptr</i> 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 (<i>nullptr</i> 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 4039 行目に定義があります。

7.1.4.8 ggCross()

```
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 1632 行目に定義があります。

7.1.4.9 ggDot3()

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

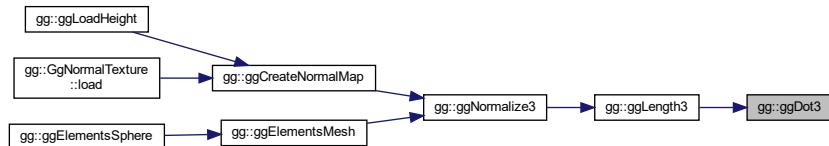
3要素の内積。

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。

gg.h の 1620 行目に定義があります。

被呼び出し関係図:



7.1.4.10 ggDot4() [1/2]

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

GgVector 型の内積

引数

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

gg.h の 1707 行目に定義があります。

7.1.4.11 ggDot4() [2/2]

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

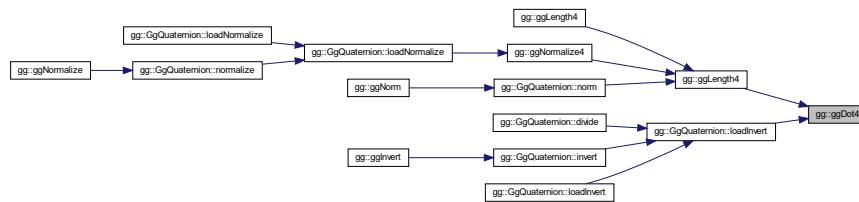
4 要素の内積

引数

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

gg.h の 1696 行目に定義があります。

被呼び出し関係図:



7.1.4.12 ggElementsMesh()

```
gg::GgElements * gg::ggElementsMesh (
    GLuint slices,
```

```

GLuint stacks,
const GLfloat(*) pos[3],
const GLfloat(*) norm[3] = nullptr )

```

メッシュ形状を作成する (Elements 形式).

メッシュ状に [GgElements](#) 形式の三角形データを生成する.

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.

gg.cpp の 5259 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:



7.1.4.13 ggElementsObj()

```

gg::GgElements * gg::ggElementsObj (
    const std::string & name,
    bool normalize = false )

```

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

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.

引数

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

gg.cpp の 5241 行目に定義があります。

呼び出し関係図:



7.1.4.14 ggElementsSphere()

```
gg::GgElements * gg::ggElementsSphere (
    GLfloat radius = 1.0f,
    int slices = 16,
    int stacks = 8 )
```

球状に三角形データを生成する (Elements 形式).

球状に GgElements 形式の三角形データを生成する.

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.

gg.cpp の 5355 行目に定義があります。

呼び出し関係図:



7.1.4.15 ggEllipse()

```
gg::GgTriangles * gg::ggEllipse (
    GLfloat width = 1.0f,
    GLfloat height = 1.0f,
    GLuint slices = 16 )
```

橢円状に三角形を生成する。

引数

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

gg.cpp の 5199 行目に定義があります。

7.1.4.16 ggEulerQuaternion() [1/2]

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

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

引数

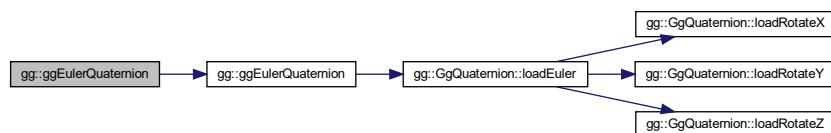
<i>e</i>	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転を表す四元数。

gg.h の 3737 行目に定義があります。

呼び出し関係図:



7.1.4.17 ggEulerQuaternion() [2/2]

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

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

引数

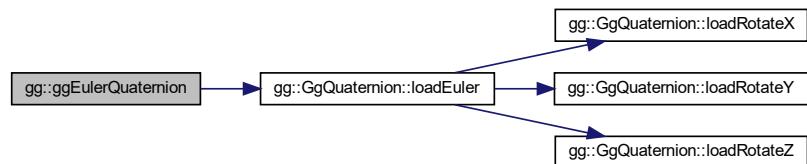
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転を表す四元数.

gg.h の 3728 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:



7.1.4.18 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 2741 行目に定義があります。

呼び出し関係図:



7.1.4.19 ggIdentity()

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

単位行列を返す。

戻り値

単位行列

gg.h の 2523 行目に定義がります。

呼び出し関係図:



7.1.4.20 gglIdentityQuaternion()

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

単位四元数を返す

戻り値

単位四元数.

gg.h の 3649 行目に定義がります。

呼び出し関係図:



7.1.4.21 ggInit()

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

ゲームグラフィックス特論の都合にもとづく初期化を行う。

WindowsにおいてOpenGL 1.2以降のAPIを有効化する。

gg.cpp の 1309 行目に定義があります。

呼び出し関係図:



7.1.4.22 ggInvert() [1/2]

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

逆行列を返す。

引数

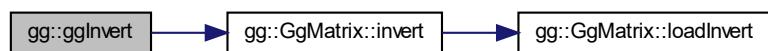
<code>m</code>	元の変換行列。
----------------	---------

戻り値

`m` の逆行列。

gg.h の 2777 行目に定義があります。

呼び出し関係図:



7.1.4.23 ggInvert() [2/2]

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

四元数の逆元を求める。

引数

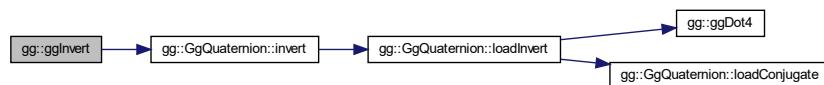
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

四元数 *q* の逆元。

gg.h の 3810 行目に定義があります。

呼び出し関係図:



7.1.4.24 ggLength3()

```
GLfloat gg::ggLength3 (
    const GLfloat * a )
```

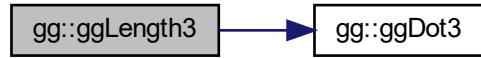
3 要素の長さ。

引数

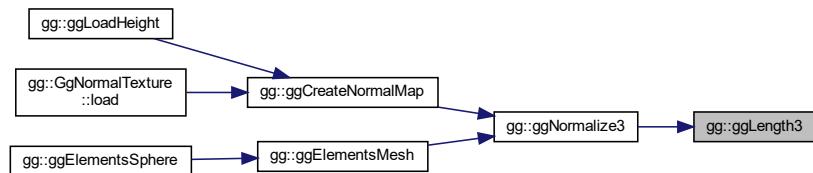
<i>a</i>	GLfloat 型の 3 要素の配列変数。
----------	-----------------------

gg.cpp の 4271 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:



7.1.4.25 ggLength4() [1/2]

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

GgVector 型の長さ.

引数

a	GgVector 型の変数.
---	----------------

gg.h の 1651 行目に定義があります。

呼び出し関係図:



7.1.4.26 ggLength4() [2/2]

```
GLfloat gg::ggLength4 (
    const GLfloat * a )
```

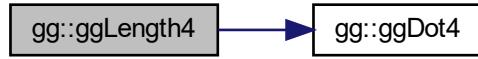
4要素の長さ。

引数

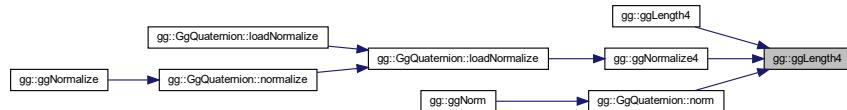
<i>a</i>	GLfloat型の4要素の配列変数。
----------	--------------------

gg.cpp の 4281 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:



7.1.4.27 ggLoadComputeShader()

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

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

引数

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

gg.cpp の 4251 行目に定義があります。

7.1.4.28 ggLoadHeight()

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

テクスチャメモリを確保して TGA 画像ファイルを読み込み法線マップを作成する.

引数

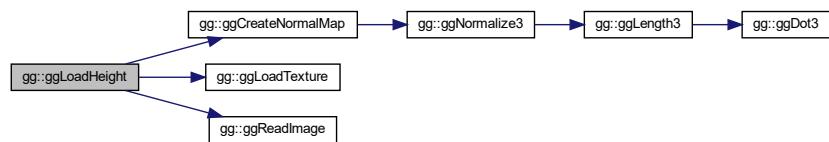
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない). ++
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

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 3110 行目に定義があります。

呼び出し関係図:



7.1.4.29 ggLoadImage()

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

テクスチャメモリを確保して TGA 画像ファイルを読み込む。

引数

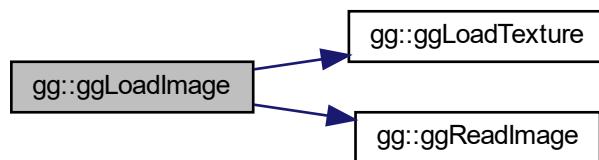
<i>name</i>	読み込むファイル名。
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない). ++
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

gg.cpp の 2961 行目に定義があります。

呼び出し関係図:



7.1.4.30 ggLoadShader()

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名 (<code>nullptr</code> 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 (<code>nullptr</code> 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 (<code>nullptr</code> 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 4177 行目に定義があります。

7.1.4.31 `ggLoadSimpleObj()` [1/2]

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

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

引数

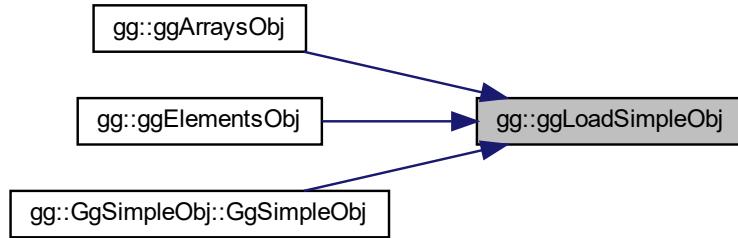
<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの <code>GgSimpleShader::Material</code> 型の材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>normalize</i>	<code>true</code> なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら `true`.

gg.cpp の 3791 行目に定義があります。

被呼び出し関係図:



7.1.4.32 `ggLoadSimpleObj()` [2/2]

```

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

三角形分割された OBJ ファイルを読み込む (Elements 形式).

引数

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

戻り値

ファイルの読み込みに成功したら true.

gg.cpp の 3880 行目に定義があります。

7.1.4.33 ggLoadTexture()

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

テクスチャメモリを確保して画像データをテクスチャとして読み込む。

引数

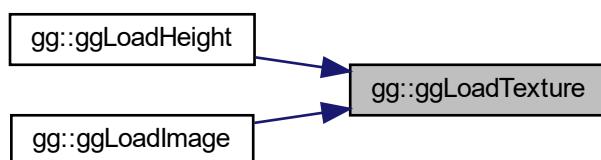
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャメモリの確保のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <code>image</code> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <code>image</code> の縦の画素数.
<i>format</i>	<code>image</code> のフォーマット.
<i>type</i>	<code>image</code> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは <code>GL_CLAMP_TO_EDGE</code> .

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 2921 行目に定義があります。

被呼び出し関係図:



7.1.4.34 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 2707 行目に定義があります。

呼び出し関係図:



7.1.4.35 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 2692 行目に定義があります。

呼び出し関係図:



7.1.4.36 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

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

戻り値

求めたビュー変換行列.

gg.h の 2677 行目に定義があります。

呼び出し関係図:



7.1.4.37 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

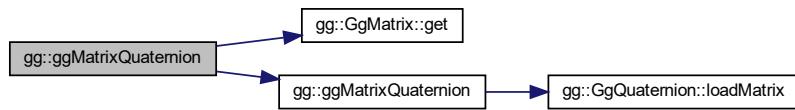
m	GgMatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 3667 行目に定義があります。

呼び出し関係図:



7.1.4.38 ggMatrixQuaternion() [2/2]

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

回転の変換行列 m を表す四元数を返す。

引数

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

戻り値

`a` による回転の変換に相当する四元数.

gg.h の 3658 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.4.39 ggNorm()

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

四元数のノルムを返す.

引数

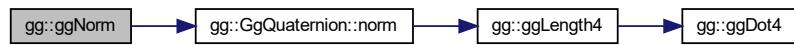
<code>q</code>	GgQuaternion 型の四元数.
----------------	---------------------

戻り値

四元数 \mathbf{q} のノルム.

gg.h の 3786 行目に定義があります。

呼び出し関係図:



7.1.4.40 ggNormal()

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

法線変換行列を返す.

引数

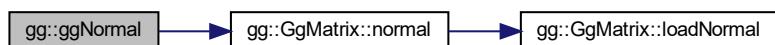
m	元の変換行列.
-----	---------

戻り値

\mathbf{m} の法線変換行列.

gg.h の 2785 行目に定義があります。

呼び出し関係図:



7.1.4.41 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 3794 行目に定義があります。

呼び出し関係図:



7.1.4.42 ggNormalize3()

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

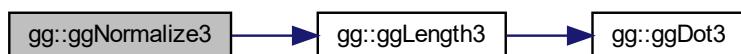
3 要素の正規化.

引数

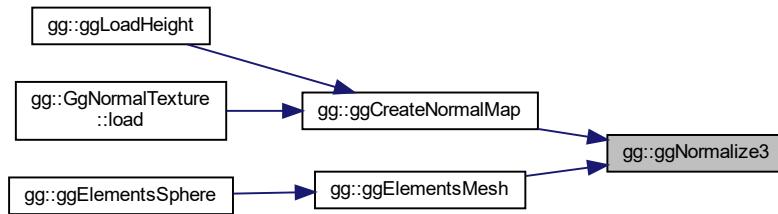
<code>a</code>	<code>GLfloat</code> 型の 3 要素の配列変数.
----------------	------------------------------------

`gg.h` の 1603 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:



7.1.4.43 ggNormalize4() [1/2]

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

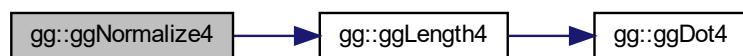
GgVector 型の正規化.

引数

a	GgVector 型の変数
---	---------------

gg.h の 1678 行目に定義があります。

呼び出し関係図:



7.1.4.44 ggNormalize4() [2/2]

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

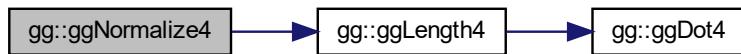
4 要素の正規化.

引数

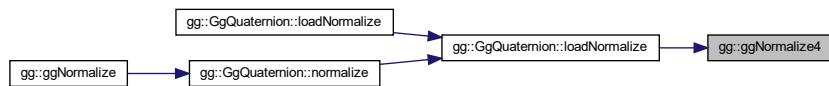
a	GLfloat 型の 4 要素の配列変数.
---	-----------------------

gg.h の 1661 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:



7.1.4.45 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 2725 行目に定義があります。

呼び出し関係図:



7.1.4.46 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 2757 行目に定義があります。

呼び出し関係図:



7.1.4.47 ggPointsCube()

```
gg::GgPoints * gg::ggPointsCube (
    GLsizei countv,
    GLfloat length = 1.0f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f )
```

点群を立方体状に生成する.

引数

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

gg.cpp の 5117 行目に定義があります。

7.1.4.48 ggPointsSphere()

```
gg::GgPoints * gg::ggPointsSphere (
    GLsizei countv,
    GLfloat radius = 0.5f,
    GLfloat cx = 0.0f,
```

```
GLfloat cy = 0.0f,  
GLfloat cz = 0.0f )
```

点群を球状に生成する。

引数

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

gg.cpp の 5147 行目に定義があります。

7.1.4.49 ggQuaternion() [1/2]

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

四元数を返す

引数

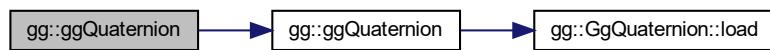
<i>a</i>	GLfloat 型の GLfloat 型の 4 要素の配列変数に格納した四元数.
----------	--

戻り値

四元数.

gg.h の 3642 行目に定義があります。

呼び出し関係図:



7.1.4.50 ggQuaternion() [2/2]

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

四元数を返す

引数

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

戻り値

四元数.

gg.h の 3633 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:



7.1.4.51 ggQuaternionMatrix()

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

四元数 *q* の回転の変換行列を返す.

引数

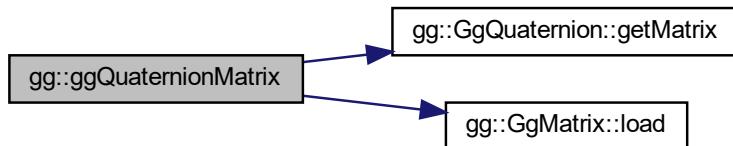
<i>q</i>	元の四元数.
----------	--------

戻り値

四元数 q が表す回転に相当する [GgMatrix](#) 型の変換行列.

gg.h の 3675 行目に定義があります。

呼び出し関係図:



7.1.4.52 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

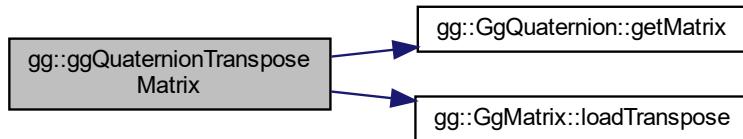
<code>q</code>	元の四元数.
----------------	--------

戻り値

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

gg.h の 3686 行目に定義があります。

呼び出し関係図:



7.1.4.53 ggReadImage()

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

TGA ファイル (8/16/24/32bit) をメモリに読み込む。

引数

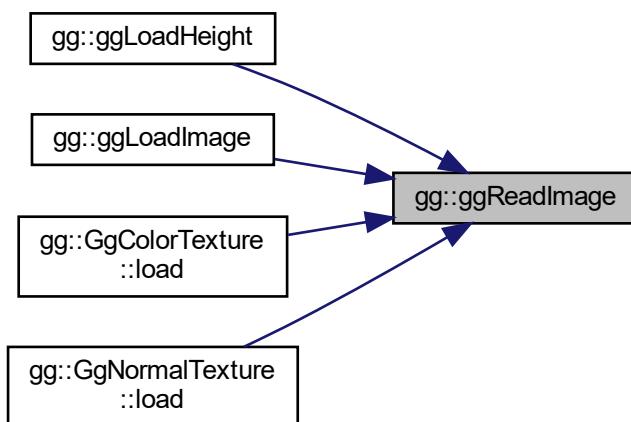
<i>name</i>	読み込むファイル名。
<i>image</i>	読み込んだデータを格納する vector。
<i>pWidth</i>	読み込んだ画像の横の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない。
<i>pHeight</i>	読み込んだ画像の縦の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない。
<i>pFormat</i>	読み込んだファイルの書式 (GL_RED, G_RG, GL_BGR, G_BGRA) の格納先のポインタ, <code>nullptr</code> なら格納しない。

戻り値

読み込みに成功すれば `true`, 失敗すれば `false`.

gg.cpp の 2801 行目に定義があります。

被呼び出し関係図:



7.1.4.54 ggRectangle()

```
gg::GgTriangles * gg::ggRectangle (
    GLfloat width = 1.0f,
    GLfloat height = 1.0f )
```

矩形状に 2 枚の三角形を生成する。

引数

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

gg.cpp の 5178 行目に定義があります。

7.1.4.55 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 GgVector 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 2660 行目に定義があります。

呼び出し関係図:



7.1.4.56 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

r	回転軸のベクトルを表す GgVector 型の変数.
a	回転角.

戻り値

r を軸に a だけ回転する変換行列.

gg.h の 2642 行目に定義があります。

呼び出し関係図:



7.1.4.57 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

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

戻り値

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

gg.h の 2651 行目に定義があります。

呼び出し関係図:



7.1.4.58 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 2632 行目に定義があります。

呼び出し関係図:



7.1.4.59 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

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 回転する変換行列.

gg.h の 2622 行目に定義があります。

呼び出し関係図:



7.1.4.60 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

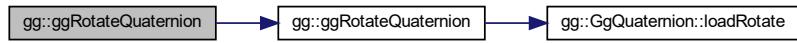
v	軸ベクトルを表す GLfloat 型の 4 要素の配列変数.
-----	--------------------------------

戻り値

回転を表す四元数.

gg.h の 3718 行目に定義があります。

呼び出し関係図:



7.1.4.61 ggRotateQuaternion() [2/3]

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

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

引数

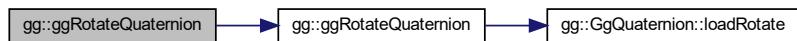
<i>v</i>	軸ベクトルを表す GLfloat 型の 3 要素の配列変数.
<i>a</i>	回転角.

戻り値

回転を表す四元数.

gg.h の 3710 行目に定義があります。

呼び出し関係図:



7.1.4.62 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

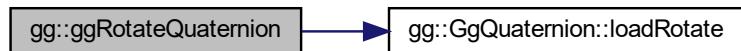
x	軸ベクトルの x 成分.
y	軸ベクトルの y 成分.
z	軸ベクトルの z 成分.
a	回転角.

戻り値

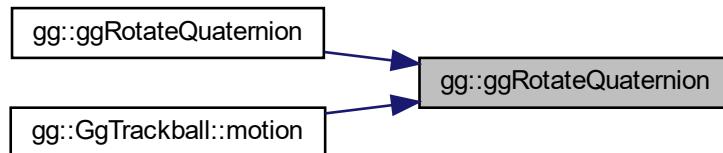
回転を表す四元数.

gg.h の 3700 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:



7.1.4.63 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 2592 行目に定義があります。

呼び出し関係図:



7.1.4.64 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 2601 行目に定義があります。

呼び出し関係図:



7.1.4.65 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 2610 行目に定義があります。

呼び出し関係図:



7.1.4.66 ggSaveColor()

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

カラーバッファの内容を TGA ファイルに保存する。

引数

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

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 2745 行目に定義がります。

呼び出し関係図:



7.1.4.67 ggSaveDepth()

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

デプスバッファの内容を TGA ファイルに保存する.

引数

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

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 2771 行目に定義がります。

呼び出し関係図:



7.1.4.68 ggSaveTga()

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

配列の内容を TGA ファイルに保存する。

引数

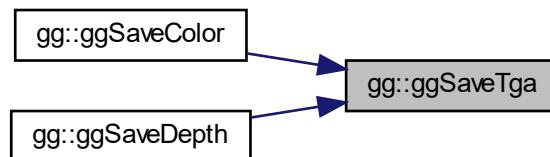
<i>name</i>	保存するファイル名.
<i>buffer</i>	画像データを格納した配列.
<i>width</i>	画像の横の画素数.
<i>height</i>	画像の縦の画素数.
<i>depth</i>	1画素のバイト数.

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 2655 行目に定義があります。

被呼び出し関係図:



7.1.4.69 ggScale() [1/3]

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

拡大縮小の変換行列を返す。

引数

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

戻り値

拡大縮小の変換行列.

gg.h の 2583 行目に定義がります。

呼び出し関係図:



7.1.4.70 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 2574 行目に定義がります。

呼び出し関係図:



7.1.4.71 ggScale() [3/3]

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

拡大縮小の変換行列を返す。

引数

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

戻り値

拡大縮小の変換行列。

gg.h の 2565 行目に定義があります。

呼び出し関係図:



7.1.4.72 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

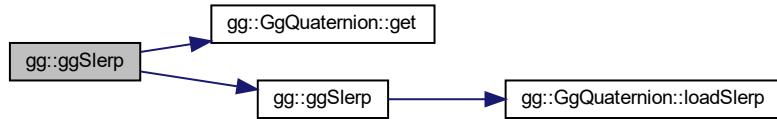
<i>q</i>	GgQuaternion 型の四元数.
<i>r</i>	GgQuaternion 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

q, r を *t* で内分した四元数.

gg.h の 3758 行目に定義があります。

呼び出し関係図:



7.1.4.73 ggSlerp() [2/4]

```

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

二つの四元数の球面線形補間の結果を返す.

引数

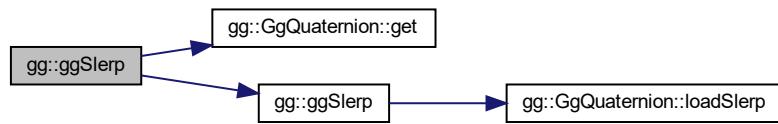
<i>q</i>	GgQuaternion 型の四元数.
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

q, a を *t* で内分した四元数.

gg.h の 3768 行目に定義があります。

呼び出し関係図:



7.1.4.74 ggSlerp() [3/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

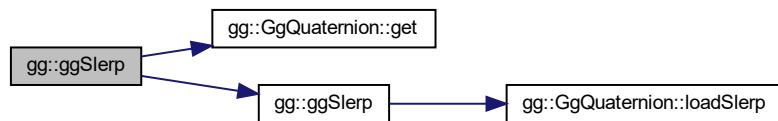
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>q</i>	GgQuaternion 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

a, *q* を *t* で内分した四元数.

gg.h の 3778 行目に定義があります。

呼び出し関係図:



7.1.4.75 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

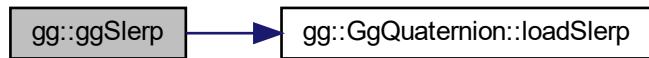
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

a, b を *t* で内分した四元数.

gg.h の 3747 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:



7.1.4.76 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 2553 行目に定義がります。

呼び出し関係図:



7.1.4.77 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 2544 行目に定義がります。

呼び出し関係図:



7.1.4.78 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 2535 行目に定義があります。

呼び出し関係図:



7.1.4.79 ggTranspose()

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

転置行列を返す。

引数

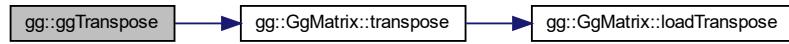
<i>m</i>	元の変換行列.
----------	---------

戻り値

m の転置行列.

gg.h の 2769 行目に定義があります。

呼び出し関係図:



7.1.5 変数詳解

7.1.5.1 ggBufferAlignment

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

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.

使用している GPU のバッファアライメント

Chapter 8

クラス詳解

8.1 GgApp クラス

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

公開メンバ関数

- void **main** (int argc, const char *const *argv)

8.1.1 詳解

GgApp.h の 15 行目に定義があります。

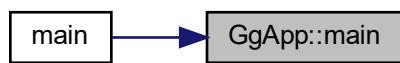
8.1.2 関数詳解

8.1.2.1 main()

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

ggsample01.cpp の 21 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:

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

8.2 gg::GgBuffer< T > クラステンプレート

バッファオブジェクト.

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

公開メンバ関数

- `GgBuffer (GLenum target, const T *data, GLsizei stride, GLsizei count, GLenum usage)`
コンストラクタ.
- `virtual ~GgBuffer ()`
デストラクタ.
- `GgBuffer (const GgBuffer< T > &o)=delete`
コピー構造子は使用禁止.
- `GgBuffer< T > & operator= (const GgBuffer< T > &o)=delete`
代入演算子は使用禁止.
- `const GLuint & getTarget () const`
バッファオブジェクトのターゲットを取り出す.
- `GLsizeiptr getStride () const`
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.
- `const GLsizei & getCount () const`
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.
- `const GLuint & getBuffer () const`
バッファオブジェクト名を取り出す.
- `void bind () const`
バッファオブジェクトを結合する.
- `void unbind () const`
バッファオブジェクトを解放する.
- `void * map () const`
バッファオブジェクトをマップする.
- `void * map (GLint first, GLsizei count) const`
バッファオブジェクトの指定した範囲をマップする.
- `void unmap () const`
バッファオブジェクトをアンマップする.
- `void send (const T *data, GLint first, GLsizei count) const`
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.
- `void read (T *data, GLint first, GLsizei count) const`
バッファオブジェクトのデータから抽出する.
- `void copy (GLuint src_buffer, GLint src_first=0, GLint dst_first=0, GLsizei count=0) const`
別のバッファオブジェクトからデータを複写する.

8.2.1 詳解

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

バッファオブジェクト.

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 4225 行目に定義があります。

8.2.2 構築子と解体子

8.2.2.1 GgBuffer() [1/2]

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

コンストラクタ.

引数

<i>target</i>	バッファオブジェクトのターゲット.
<i>data</i>	データが格納されている領域の先頭のポインタ (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 4237 行目に定義があります。

8.2.2.2 ~GgBuffer()

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

デストラクタ.

gg.h の 4237 行目に定義があります。

8.2.2.3 GgBuffer() [2/2]

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

コピー構築子は使用禁止.

8.2.3 関数詳解

8.2.3.1 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 4307 行目に定義があります。

8.2.3.2 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 (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 4381 行目に定義があります。

8.2.3.3 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 4301 行目に定義があります。

8.2.3.4 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 4294 行目に定義があります。

8.2.3.5 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 4287 行目に定義があります。

8.2.3.6 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 4280 行目に定義があります。

8.2.3.7 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 4320 行目に定義があります。

8.2.3.8 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

<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 4330 行目に定義があります。

8.2.3.9 operator=()

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

代入演算子は使用禁止。

8.2.3.10 `read()`

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

バッファオブジェクトのデータから抽出する。

引数

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

gg.h の 4365 行目に定義があります。

8.2.3.11 `send()`

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

すでに確保したバッファオブジェクトにデータを転送する。

引数

<i>data</i>	転送元のデータが格納されて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 4350 行目に定義があります。

8.2.3.12 `unbind()`

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

バッファオブジェクトを解放する。

gg.h の 4313 行目に定義があります。

8.2.3.13 unmap()

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

バッファオブジェクトをアンマップする。

gg.h の 4341 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](#)

8.3 gg::GgColorTexture クラス

カラーマップ.

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

公開メンバ関数

- [GgColorTexture\(\)](#)
コンストラクタ.
- [GgColorTexture\(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_BGR, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
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.
- [GgColorTexture\(const std::string &name, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
ファイルからデータを読み込んでテクスチャを作成するコンストラクタ.
- [virtual ~GgColorTexture\(\)](#)
デストラクタ.
- [void load\(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_BGR, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
テクスチャを作成してメモリ上のデータを読み込む.
- [void load\(const std::string &name, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
テクスチャを作成してファイルからデータを読み込む.

8.3.1 詳解

カラーマップ.

カラー画像を読み込んでテクスチャを作成する.

gg.h の 4050 行目に定義があります。

8.3.2 構築子と解体子

8.3.2.1 GgColorTexture() [1/3]

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

コンストラクタ.

gg.h の 4058 行目に定義があります。

8.3.2.2 GgColorTexture() [2/3]

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

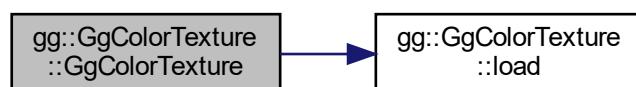
メモリ上のデータからテクスチャを作成するコンストラクタ.

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>	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.

gg.h の 4070 行目に定義があります。

呼び出し関係図:



8.3.2.3 GgColorTexture() [3/3]

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

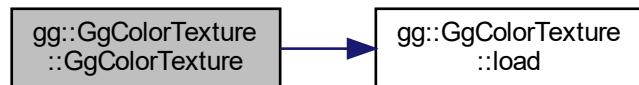
ファイルからデータを読み込んでテクスチャを作成するコンストラクタ.

引数

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

gg.h の 4087 行目に定義があります。

呼び出し関係図:



8.3.2.4 ~GgColorTexture()

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

デストラクタ.

gg.h の 4093 行目に定義があります。

8.3.3 関数詳解

8.3.3.1 load() [1/2]

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

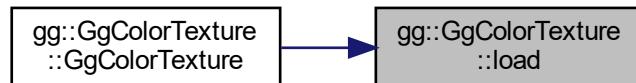
テクスチャを作成してメモリ上のデータを読み込む。

引数

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

gg.h の 4105 行目に定義があります。

被呼び出し関係図:



8.3.3.2 load() [2/2]

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

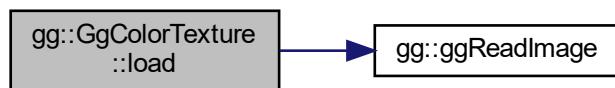
テクスチャを作成してファイルからデータを読み込む。

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 3154 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:

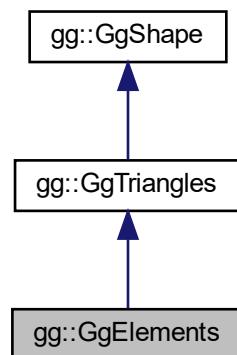
- [gg.h](#)
- [gg.cpp](#)

8.4 gg::GgElements クラス

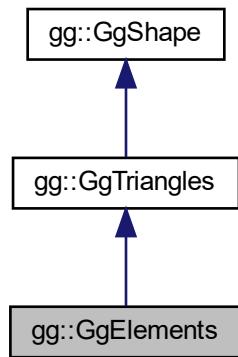
三角形で表した形状データ (Elements 形式).

```
#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)`
コンストラクタ.
- `GgElements (const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum mode=GL_TRIANGLES, GLenum usage=GL_STATIC_DRAW)`
コンストラクタ.
- `virtual ~GgElements ()`
デストラクタ.
- `const GLsizei & getIndexCount () const`
データの数を取り出す.
- `const GLuint & getIndexBuffer () const`
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.
- `void send (const GgVertex *vert, GLuint firstv, GLsizei countv, const GLuint *face=nullptr, GLuint firstf=0, GLsizei countf=0) const`
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.
- `void load (const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum usage=GL_STATIC_DRAW)`
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.
- `virtual void draw (GLint first=0, GLsizei count=0) const`
インデックスを使った三角形の描画.

8.4.1 詳解

三角形で表した形状データ (Elements 形式).

gg.h の 4935 行目に定義があります。

8.4.2 構築子と解体子

8.4.2.1 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 4945 行目に定義がります。

8.4.2.2 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 4957 行目に定義がります。

8.4.2.3 ~GgElements()

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

デストラクタ.

gg.h の 4971 行目に定義がります。

8.4.3 関数詳解

8.4.3.1 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.

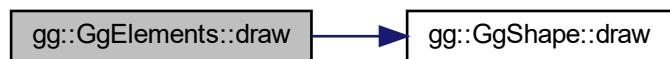
引数

<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 5104 行目に定義があります。

呼び出し関係図:



8.4.3.2 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 4984 行目に定義があります。

8.4.3.3 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 4977 行目に定義があります。

8.4.3.4 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 5015 行目に定義があります。

8.4.3.5 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 4996 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](#)

8.5 gg::GgMatrix クラス

変換行列.

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

公開 メンバ関数

- [GgMatrix \(\)](#)
コンストラクタ.
- [GgMatrix \(const GLfloat *a\)](#)
コンストラクタ.
- [GgMatrix \(const GgMatrix &m\)](#)
コピーコンストラクタ.
- [~GgMatrix \(\)](#)
デストラクタ.
- [GgMatrix & load \(const GLfloat *a\)](#)
配列変数の値を格納する.
- [GgMatrix & load \(const GgMatrix &m\)](#)
別の変換行列の値を格納する.
- [GgMatrix & loadAdd \(const GLfloat *a\)](#)
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.
- [GgMatrix & loadAdd \(const GgMatrix &m\)](#)
変換行列に別の変換行列を加算した結果を格納する.
- [GgMatrix & loadSubtract \(const GLfloat *a\)](#)
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.
- [GgMatrix & loadSubtract \(const GgMatrix &m\)](#)
変換行列から別の変換行列を減算した結果を格納する.
- [GgMatrix & loadMultiply \(const GLfloat *a\)](#)
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.
- [GgMatrix & loadMultiply \(const GgMatrix &m\)](#)

- 変換行列に別の変換行列を乗算した結果を格納する.
- `GgMatrix & loadDivide (const GLfloat *a)`
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.
- `GgMatrix & loadDivide (const GgMatrix &m)`
変換行列を別の変換行列で除算した結果を格納する.
- `GgMatrix add (const GLfloat *a) const`
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.
- `GgMatrix add (const GgMatrix &m) const`
変換行列に別の変換行列を加算した値を返す.
- `GgMatrix subtract (const GLfloat *a) const`
変換行列から配列に格納した変換行列を減算した値を返す.
- `GgMatrix subtract (const GgMatrix &m) const`
変換行列から別の変換行列を減算した値を返す.
- `GgMatrix multiply (const GLfloat *a) const`
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.
- `GgMatrix multiply (const GgMatrix &m) const`
変換行列に別の変換行列を乗算した値を返す.
- `GgMatrix divide (const GLfloat *a) const`
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.
- `GgMatrix divide (const GgMatrix &m) const`
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.
- `GgMatrix & operator= (const GLfloat *a)`
- `GgMatrix & operator= (const GgMatrix &m)`
- `GgMatrix & operator+= (const GLfloat *a)`
- `GgMatrix & operator+= (const GgMatrix &m)`
- `GgMatrix & operator-= (const GLfloat *a)`
- `GgMatrix & operator-= (const GgMatrix &m)`
- `GgMatrix & operator*=(const GLfloat *a)`
- `GgMatrix & operator*=(const GgMatrix &m)`
- `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) const`
- `GgMatrix operator- (const GgMatrix &m) const`
- `GgMatrix operator* (const GLfloat *a) const`
- `GgMatrix operator* (const GgMatrix &m) const`
- `GgMatrix operator/ (const GLfloat *a) const`
- `GgMatrix operator/ (const GgMatrix &m) const`
- `GgMatrix & loadIdentity ()`
単位行列を格納する.
- `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.
- `GgMatrix & loadTranslate (const GLfloat *t)`
平行移動の変換行列を格納する.
- `GgMatrix & loadTranslate (const GgVector &t)`
平行移動の変換行列を格納する.
- `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.
- `GgMatrix & loadScale (const GLfloat *s)`
拡大縮小の変換行列を格納する.
- `GgMatrix & loadScale (const GgVector &s)`

- 拡大縮小の変換行列を格納する.
- `GgMatrix & loadRotateX (GLfloat a)`
 x 軸中心の回転の変換行列を格納する.
- `GgMatrix & loadRotateY (GLfloat a)`
 y 軸中心の回転の変換行列を格納する.
- `GgMatrix & loadRotateZ (GLfloat a)`
 z 軸中心の回転の変換行列を格納する.
- `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.
- `GgMatrix & loadRotate (const GLfloat *r, GLfloat a)`
 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.
- `GgMatrix & loadRotate (const GgVector &r, GLfloat a)`
 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.
- `GgMatrix & loadRotate (const GLfloat *r)`
 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.
- `GgMatrix & loadRotate (const GgVector &r)`
 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.
- `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.
- `GgMatrix & loadLookat (const GLfloat *e, const GLfloat *t, const GLfloat *u)`
ビュー変換行列を格納する.
- `GgMatrix & loadLookat (const GgVector &e, const GgVector &t, const GgVector &u)`
ビュー変換行列を格納する.
- `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.
- `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.
- `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.
- `GgMatrix & loadTranspose (const GLfloat *a)`
転置行列を格納する.
- `GgMatrix & loadTranspose (const GgMatrix &m)`
転置行列を格納する.
- `GgMatrix & loadInvert (const GLfloat *a)`
逆行列を格納する.
- `GgMatrix & loadInvert (const GgMatrix &m)`
逆行列を格納する.
- `GgMatrix & loadNormal (const GLfloat *a)`
法線変換行列を格納する.
- `GgMatrix & loadNormal (const GgMatrix &m)`
法線変換行列を格納する.
- `GgMatrix translate (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f) const`
平行移動変換を乗じた結果を返す.
- `GgMatrix translate (const GLfloat *t) const`
平行移動変換を乗じた結果を返す.
- `GgMatrix translate (const GgVector &t) const`
平行移動変換を乗じた結果を返す.
- `GgMatrix scale (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f) const`
拡大縮小変換を乗じた結果を返す.

- **GgMatrix scale** (const GLfloat *s) const
拡大縮小変換を乗じた結果を返す.
- **GgMatrix scale** (const GgVector &s) const
拡大縮小変換を乗じた結果を返す.
- **GgMatrix rotateX** (GLfloat a) const
x 軸中心の回転変換を乗じた結果を返す.
- **GgMatrix rotateY** (GLfloat a) const
y 軸中心の回転変換を乗じた結果を返す.
- **GgMatrix rotateZ** (GLfloat a) const
z 軸中心の回転変換を乗じた結果を返す.
- **GgMatrix rotate** (GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
(*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.
- **GgMatrix rotate** (const GLfloat *r, GLfloat a) const
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.
- **GgMatrix rotate** (const GgVector &r, GLfloat a) const
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.
- **GgMatrix rotate** (const GLfloat *r) const
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.
- **GgMatrix rotate** (const GgVector &r) const
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.
- **GgMatrix lookat** (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz) const
ビュー変換を乗じた結果を返す.
- **GgMatrix lookat** (const GLfloat *e, const GLfloat *t, const GLfloat *u) const
ビュー変換を乗じた結果を返す.
- **GgMatrix lookat** (const GgVector &e, const GgVector &t, const GgVector &u) const
ビュー変換を乗じた結果を返す.
- **GgMatrix orthogonal** (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar) const
直交投影変換を乗じた結果を返す.
- **GgMatrix frustum** (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar) const
透視投影変換を乗じた結果を返す.
- **GgMatrix perspective** (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar) const
画角を指定して透視投影変換を乗じた結果を返す.
- **GgMatrix transpose** () const
転置行列を返す.
- **GgMatrix invert** () const
逆行行列を返す.
- **GgMatrix normal** () const
法線変換行列を返す.
- **void projection** (GLfloat *c, const GLfloat *v) const
ベクトルに対して投影変換を行う.
- **void projection** (GLfloat *c, const GgVector &v) const
ベクトルに対して投影変換を行う.
- **void projection** (GgVector &c, const GLfloat *v) const
ベクトルに対して投影変換を行う.
- **void projection** (GgVector &c, const GgVector &v) const
ベクトルに対して投影変換を行う.
- **GgVector operator*** (const GgVector &v) const
ベクトルに対して投影変換を行う.
- **const GLfloat * get** () const

- 変換行列を取り出す.
• void [get](#) (GLfloat *a) const
 変換行列を取り出す.
- const GLfloat & [get](#) (int i) const
 変換行列の要素を取り出す.
- const GLfloat & [operator\[\]](#) (std::size_t i) const
 変換行列の要素にアクセスする.
- GLfloat & [operator\[\]](#) (std::size_t i)
 変換行列の要素にアクセスする.

フレンド

- class [GgQuaternion](#)

8.5.1 詳解

変換行列.

gg.h の 1715 行目に定義があります。

8.5.2 構築子と解体子

8.5.2.1 GgMatrix() [1/3]

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

コンストラクタ.

gg.h の 1732 行目に定義があります。

8.5.2.2 GgMatrix() [2/3]

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

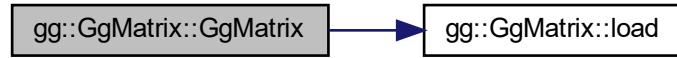
コンストラクタ.

引数

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

gg.h の 1738 行目に定義があります。

呼び出し関係図:



8.5.2.3 GgMatrix() [3/3]

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

コピー構造関数。

引数

<i>m</i>	GgMatrix 型の変数。
----------	----------------

gg.h の 1745 行目に定義があります。

呼び出し関係図:



8.5.2.4 ~GgMatrix()

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

デストラクタ。

gg.h の 1751 行目に定義があります。

8.5.3 関数詳解

8.5.3.1 add() [1/2]

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

変換行列に別の変換行列を加算した値を返す.

引数

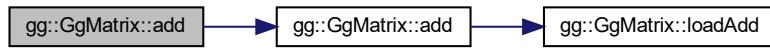
<code>m</code>	<code>GgMatrix</code> 型の変数.
----------------	-----------------------------

戻り値

変換行列に `m` を加えた `GgMatrix` 型の値.

gg.h の 1850 行目に定義があります。

呼び出し関係図:



8.5.3.2 add() [2/2]

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

変換行列に配列に格納した変換行列を加算した値を返す.

引数

<code>a</code>	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` 型の値.

gg.h の 1841 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.3 divide() [1/2]

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

変換行列を配列に格納した変換行列で除算した値を返す。

引数

<i>m</i>	GgMatrix 型の変数。
----------	----------------

戻り値

変換行列を *m* で割った GgMatrix 型の値。

gg.h の 1904 行目に定義があります。

呼び出し関係図:



8.5.3.4 divide() [2/2]

```
GgMatrix gg::GgMatrix::divide (
    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

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

戻り値

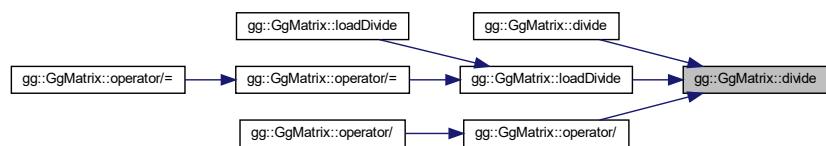
変換行列を a で割った GgMatrix 型の値.

gg.h の 1893 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.5 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

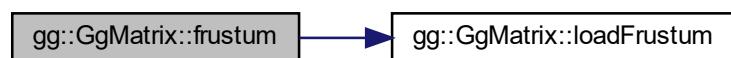
<i>left</i>	ウィンドウの左端の位置。
<i>right</i>	ウィンドウの右端の位置。
<i>bottom</i>	ウィンドウの下端の位置。
<i>top</i>	ウィンドウの上端の位置。
<i>zNear</i>	視点から前方面までの位置。
<i>zFar</i>	視点から後方面までの位置。

戻り値

透視投影変換行列を乗じた変換行列。

gg.h の 2394 行目に定義があります。

呼び出し関係図:



8.5.3.6 get() [1/3]

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

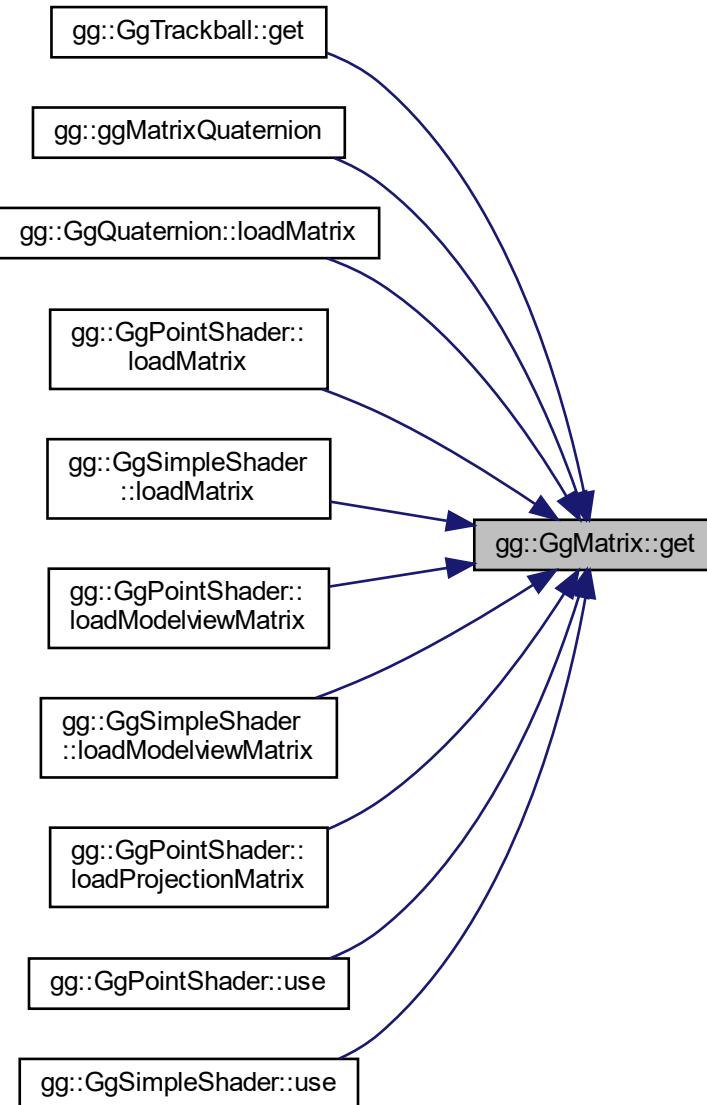
変換行列を取り出す。

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 2487 行目に定義があります。

被呼び出し関係図:



8.5.3.7 get() [2/3]

```
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 2494 行目に定義があります。

8.5.3.8 get() [3/3]

```
const GLfloat& gg::GgMatrix::get (
    int i ) const [inline]
```

変換行列の要素を取り出す.

戻り値

変換行列を格納した GLfloat 型の 16 要素の配列変数 の i 番目の要素.

gg.h の 2501 行目に定義があります。

8.5.3.9 invert()

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

逆行列を返す.

戻り値

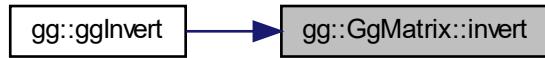
逆行列.

gg.h の 2429 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.10 load() [1/2]

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

別の変換行列の値を格納する.

引数

<code>m</code>	<code>GgMatrix</code> 型の変数.
----------------	-----------------------------

戻り値

`m` を代入した `GgMatrix` 型の値.

gg.h の 1767 行目に定義があります。

呼び出し関係図:



8.5.3.11 load() [2/2]

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

配列変数の値を格納する.

引数

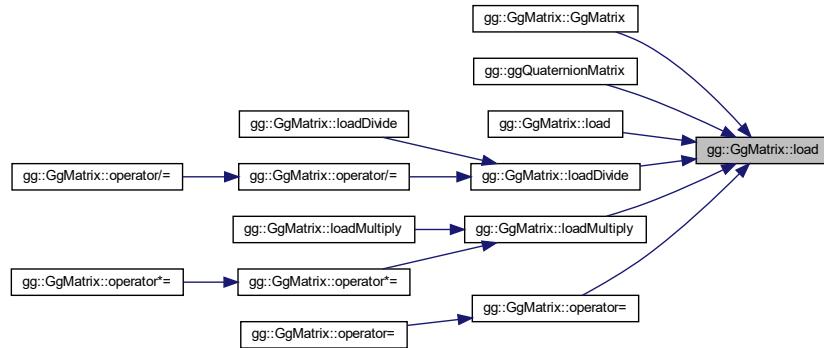
<code>a</code>	GLfloat 型の 16 要素の配列変数.
----------------	------------------------

戻り値

`a` を代入した `GgMatrix` 型の値.

gg.h の 1758 行目に定義があります。

被呼び出し関係図:



8.5.3.12 loadAdd() [1/2]

```
GgMatrix& gg::GgMatrix::loadAdd (
    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>	<i>GgMatrix</i> 型の変数.
----------	-----------------------

戻り値

変換行列に *m* を加えた *GgMatrix* 型の値.

gg.h の 1784 行目に定義があります。

呼び出し関係図:



8.5.3.13 loadAdd() [2/2]

```
GgMatrix& gg::GgMatrix::loadAdd (
    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

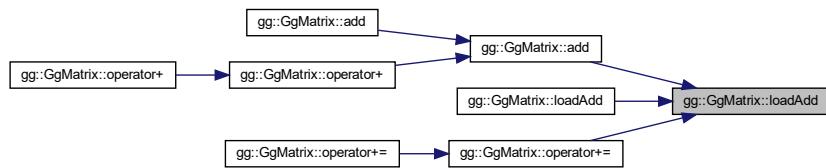
<i>a</i>	GLfloat 型の 16 要素の配列変数。
----------	------------------------

戻り値

変換行列に *a* を加えた GgMatrix 型の値。

gg.h の 1775 行目に定義があります。

被呼び出し関係図:



8.5.3.14 loadDivide() [1/2]

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

変換行列を別の変換行列で除算した結果を格納する。

引数

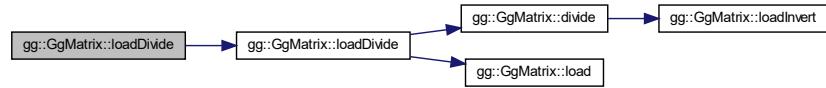
<i>m</i>	GgMatrix 型の変数。
----------	----------------

戻り値

変換行列に *m* を乗じた GgMatrix 型の値。

gg.h の 1833 行目に定義があります。

呼び出し関係図:



8.5.3.15 loadDivide() [2/2]

```
GgMatrix& gg::GgMatrix::loadDivide (
    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

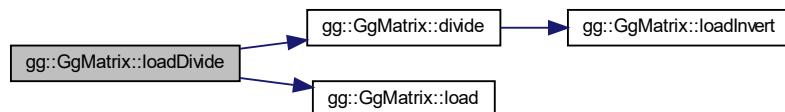
a	GLfloat 型の 16 要素の配列変数。
---	------------------------

戻り値

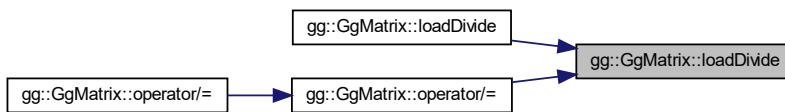
変換行列に a を乗じた GgMatrix 型の値。

gg.h の 1825 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.16 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

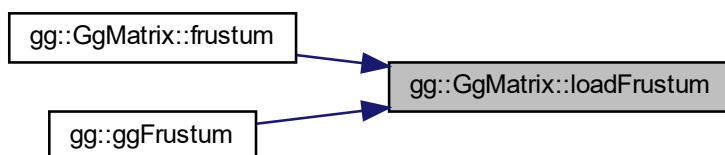
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視透視投影変換行列.

gg.cpp の 4660 行目に定義があります。

被呼び出し関係図:



8.5.3.17 loadIdentity()

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

単位行列を格納する。

gg.cpp の 4313 行目に定義がります。

呼び出し関係図:



8.5.3.18 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 2180 行目に定義がります。

呼び出し関係図:



8.5.3.19 loadInvert() [2/2]

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

逆行列を格納する。

引数

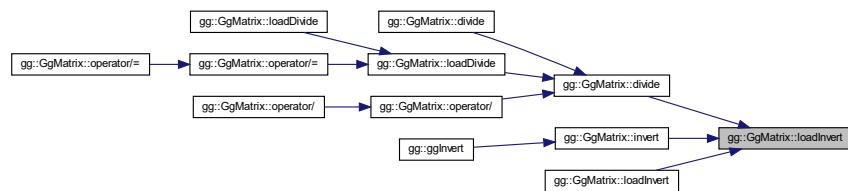
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

設定した a の逆行列.

gg.cpp の 4470 行目に定義があります。

被呼び出し関係図:



8.5.3.20 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	視点の位置の 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 2121 行目に定義があります。

呼び出し関係図:



8.5.3.21 loadLookat() [2/3]

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

ビュー変換行列を格納する。

引数

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

戻り値

設定したビュー変換行列.

gg.h の 2111 行目に定義があります。

呼び出し関係図:



8.5.3.22 loadLookat() [3/3]

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

ビュー変換行列を格納する。

引数

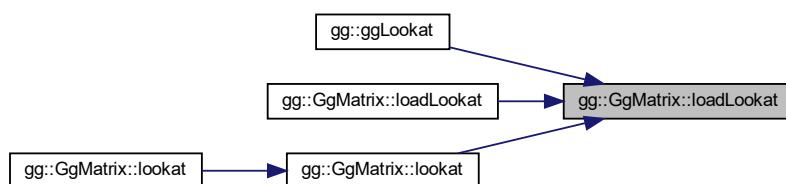
<i>ex</i>	視点の位置の x 座標値.
<i>ey</i>	視点の位置の y 座標値.
<i>ez</i>	視点の位置の z 座標値.
<i>tx</i>	目標点の位置の x 座標値.
<i>ty</i>	目標点の位置の y 座標値.
<i>tz</i>	目標点の位置の z 座標値.
<i>ux</i>	上方向のベクトルの x 成分.
<i>uy</i>	上方向のベクトルの y 成分.
<i>uz</i>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列.

gg.cpp の 4572 行目に定義があります。

被呼び出し関係図:



8.5.3.23 `loadMultiply()` [1/2]

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

変換行列に別の変換行列を乗算した結果を格納する。

引数

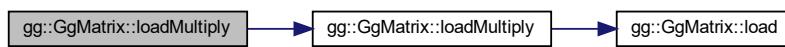
<code>m</code>	GgMatrix 型の変数.
----------------	----------------

戻り値

変換行列に `m` を掛けた GgMatrix 型の値.

gg.h の 1817 行目に定義があります。

呼び出し関係図:



8.5.3.24 loadMultiply() [2/2]

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

変換行列に配列に格納した変換行列を乗算した結果を格納する.

引数

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

戻り値

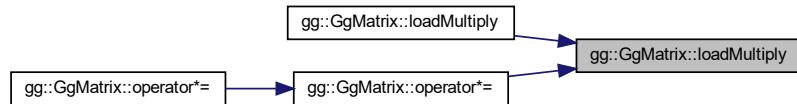
変換行列に `a` を掛けた GgMatrix 型の値.

gg.h の 1809 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.25 loadNormal() [1/2]

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

法線変換行列を格納する。

引数

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

戻り値

設定した *m* の法線変換行列。

gg.h の 2193 行目に定義があります。

呼び出し関係図:



8.5.3.26 loadNormal() [2/2]

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

法線変換行列を格納する。

引数

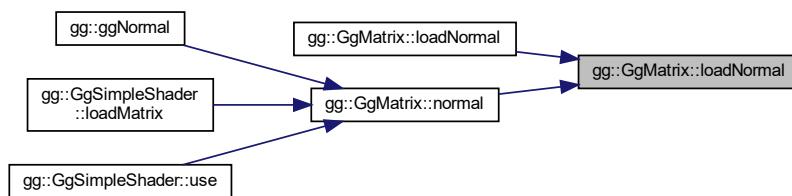
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

設定した m の法線変換行列.

gg.cpp の 4552 行目に定義があります。

被呼び出し関係図:



8.5.3.27 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

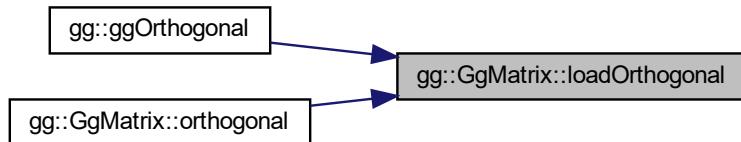
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した直交投影変換行列.

gg.cpp の 4630 行目に定義があります。

被呼び出し関係図:



8.5.3.28 loadPerspective()

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

画角を指定して透視投影変換行列を格納する.

引数

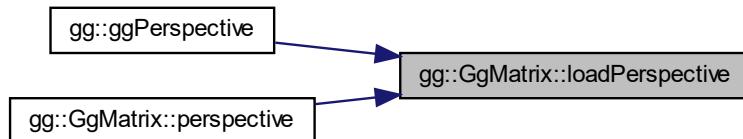
<i>fovy</i>	<i>y</i> 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 4690 行目に定義があります。

被呼び出し関係図:



8.5.3.29 loadRotate() [1/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する。

引数

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

戻り値

設定した変換行列.

gg.h の 2086 行目に定義があります。

呼び出し関係図:



8.5.3.30 loadRotate() [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する。

引数

<i>r</i>	回転軸の方向ベクトルを格納した GgVector 型の変数.
<i>a</i>	回転角.

戻り値

設定した変換行列.

gg.h の 2070 行目に定義があります。

呼び出し関係図:



8.5.3.31 loadRotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2078 行目に定義があります。

呼び出し関係図:



8.5.3.32 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

r	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 (x, y, z).
a	回転角.

戻り値

設定した変換行列.

gg.h の 2061 行目に定義があります。

呼び出し関係図:



8.5.3.33 loadRotate() [5/5]

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

(x, y, z) 方向のベクトルを軸とする回転の変換行列を格納する。

引数

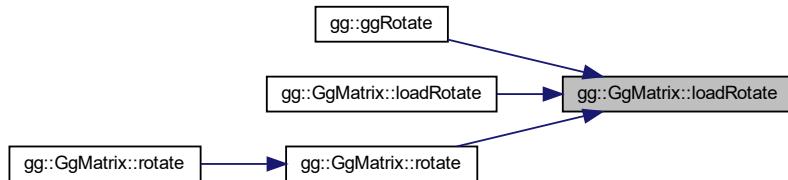
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

設定した変換行列.

gg.cpp の 4406 行目に定義があります。

被呼び出し関係図:



8.5.3.34 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

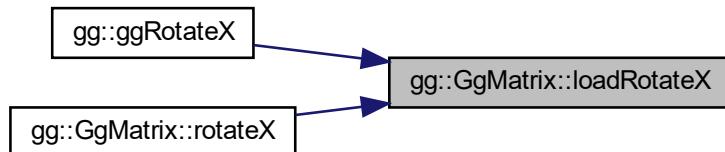
a	回転角.
-----	------

戻り値

設定した変換行列.

gg.cpp の 4358 行目に定義があります。

被呼び出し関係図:



8.5.3.35 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

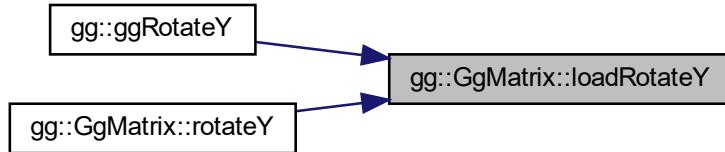
a	回転角.
---	------

戻り値

設定した変換行列.

gg.cpp の 4374 行目に定義があります。

被呼び出し関係図:



8.5.3.36 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

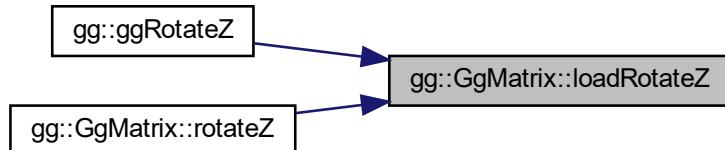
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 4390 行目に定義があります。

被呼び出し関係図:



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

戻り値

設定した変換行列.

gg.h の 2029 行目に定義があります。

呼び出し関係図:



8.5.3.38 loadScale() [2/3]

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

拡大縮小の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2021 行目に定義があります。

呼び出し関係図:



8.5.3.39 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

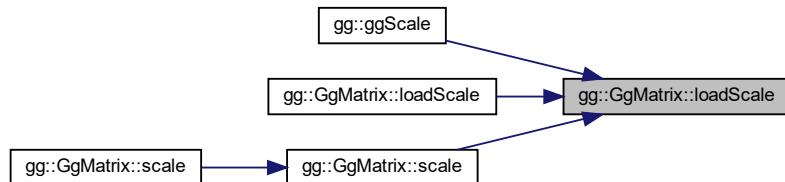
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 拡大率のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 4342 行目に定義があります。

被呼び出し関係図:



8.5.3.40 loadSubtract() [1/2]

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

変換行列から別の変換行列を減算した結果を格納する。

引数

<i>m</i>	<code>GgMatrix</code> 型の変数.
----------	-----------------------------

戻り値

変換行列に m を引いた GgMatrix 型の値.

gg.h の 1801 行目に定義があります。

呼び出し関係図:



8.5.3.41 loadSubtract() [2/2]

```
GgMatrix& gg::GgMatrix::loadSubtract (
    const GLfloat * a ) [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

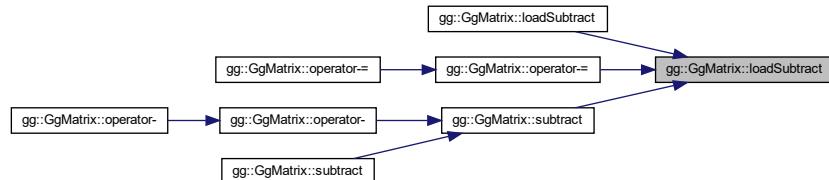
a	GLfloat 型の 16 要素の配列変数.
---	------------------------

戻り値

変換行列に a を引いた GgMatrix 型の値.

gg.h の 1792 行目に定義があります。

被呼び出し関係図:



8.5.3.42 `loadTranslate()` [1/3]

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

平行移動の変換行列を格納する。

引数

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

戻り値

設定した変換行列。

gg.h の 2005 行目に定義があります。

呼び出し関係図:



8.5.3.43 `loadTranslate()` [2/3]

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

平行移動の変換行列を格納する。

引数

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

戻り値

設定した変換行列。

gg.h の 1997 行目に定義があります。

呼び出し関係図:



8.5.3.44 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

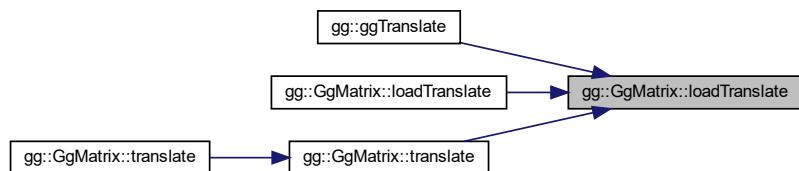
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 4326 行目に定義があります。

被呼び出し関係図:



8.5.3.45 `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 2167 行目に定義があります。

呼び出し関係図:



8.5.3.46 `loadTranspose()` [2/2]

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

転置行列を格納する。

引数

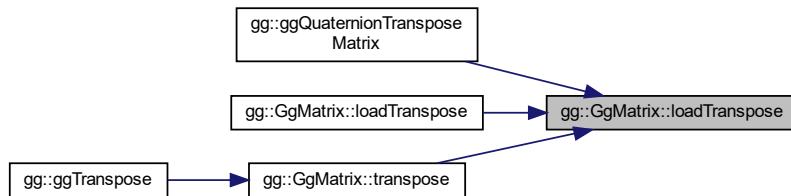
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列。
----------------	-------------------------------------

戻り値

設定した `a` の転置行列。

`gg.cpp` の 4445 行目に定義があります。

被呼び出し関係図:



8.5.3.47 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

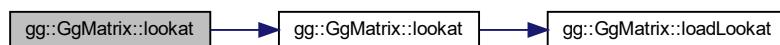
<i>e</i>	視点の位置を格納した GgVector 型の変数.
<i>t</i>	目標点の位置を格納した GgVector 型の変数.
<i>u</i>	上方向のベクトルを格納した GgVector 型の変数.

戻り値

ビュー変換行列を乗じた変換行列.

gg.h の 2363 行目に定義があります。

呼び出し関係図:



8.5.3.48 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 2353 行目に定義があります。

呼び出し関係図:



8.5.3.49 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

<i>ex</i>	視点の位置の x 座標値.
-----------	---------------

引数

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

戻り値

ビュー変換行列を乗じた変換行列.

gg.h の 2338 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.50 multiply() [1/2]

```
GgMatrix gg::GgMatrix::multiply (
    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> 型の変数.
----------------	-----------------------------

戻り値

変換行列に `m` を掛けた `GgMatrix` 型の値.

`gg.h` の 1885 行目に定義がります。

8.5.3.51 `multiply()` [2/2]

```
GgMatrix gg::GgMatrix::multiply (
    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

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

戻り値

変換行列に `a` を掛けた `GgMatrix` 型の値.

`gg.h` の 1875 行目に定義がります。

8.5.3.52 `normal()`

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

法線変換行列を返す.

戻り値

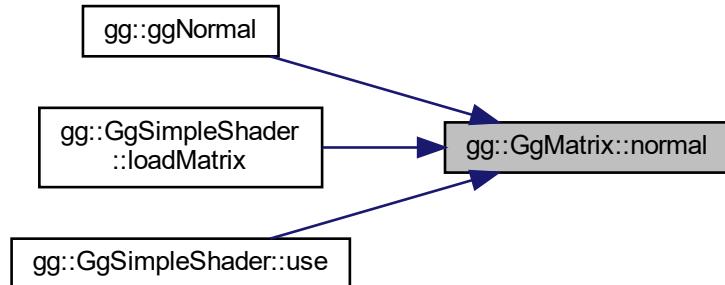
法線変換行列.

`gg.h` の 2437 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.53 operator*() [1/3]

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

gg.h の 1970 行目に定義があります。

呼び出し関係図:



8.5.3.54 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

c 変換結果の GgVector 型の値。

gg.h の 2478 行目に定義があります。

8.5.3.55 operator*() [3/3]

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

gg.h の 1966 行目に定義があります。

呼び出し関係図:

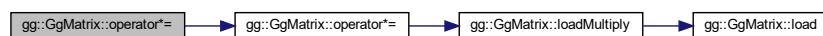


8.5.3.56 operator*=(()) [1/2]

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

gg.h の 1938 行目に定義があります。

呼び出し関係図:

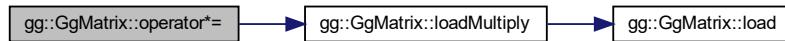


8.5.3.57 operator*=() [2/2]

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

gg.h の 1934 行目に定義があります。

呼び出し関係図:



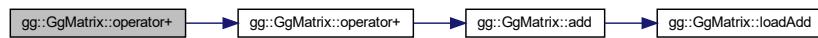
被呼び出し関係図:

**8.5.3.58 operator+() [1/2]**

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

gg.h の 1954 行目に定義があります。

呼び出し関係図:



8.5.3.59 operator+() [2/2]

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

gg.h の 1950 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

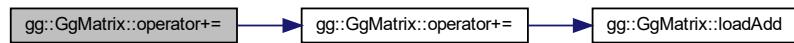


8.5.3.60 operator+=() [1/2]

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

gg.h の 1922 行目に定義があります。

呼び出し関係図:



8.5.3.61 operator+=() [2/2]

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

gg.h の 1918 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.62 operator-() [1/2]

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

gg.h の 1962 行目に定義があります。

呼び出し関係図:

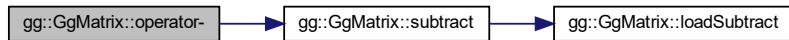


8.5.3.63 operator-() [2/2]

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

gg.h の 1958 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.64 operator-=(()) [1/2]

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

gg.h の 1930 行目に定義があります。

呼び出し関係図:



8.5.3.65 operator-=() [2/2]

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

gg.h の 1926 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.66 operator/() [1/2]

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

gg.h の 1978 行目に定義があります。

呼び出し関係図:

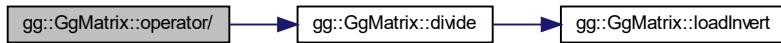


8.5.3.67 operator/() [2/2]

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

gg.h の 1974 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

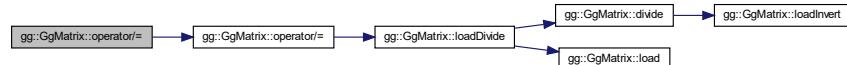


8.5.3.68 operator/=(()) [1/2]

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

gg.h の 1946 行目に定義があります。

呼び出し関係図:

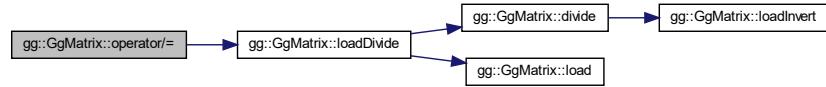


8.5.3.69 operator/() [2/2]

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

gg.h の 1942 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

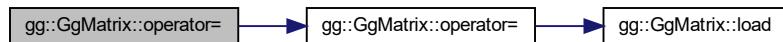


8.5.3.70 operator=(()) [1/2]

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

gg.h の 1914 行目に定義があります。

呼び出し関係図:



8.5.3.71 operator=() [2/2]

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

gg.h の 1910 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.72 operator[]() [1/2]

```
GLfloat& gg::GgMatrix::operator[] (
    std::size_t i )  [inline]
```

変換行列の要素にアクセスする。

戻り値

変換行列を格納した `GLfloat` 型の 16 要素の配列変数 の `i` 番目の要素の参照。

gg.h の 2515 行目に定義があります。

8.5.3.73 operator[]() [2/2]

```
const GLfloat& gg::GgMatrix::operator[] (
    std::size_t i ) const [inline]
```

変換行列の要素にアクセスする。

戻り値

変換行列を格納した `GLfloat` 型の 16 要素の配列変数 の *i* 番目の要素の参照。

`gg.h` の 2508 行目に定義があります。

8.5.3.74 orthogonal()

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

直交投影変換を乗じた結果を返す。

引数

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

戻り値

直交投影変換行列を乗じた変換行列。

`gg.h` の 2376 行目に定義があります。

呼び出し関係図:



8.5.3.75 perspective()

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

画角を指定して透視投影変換を乗じた結果を返す。

引数

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

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2410 行目に定義があります。

呼び出し関係図:



8.5.3.76 projection() [1/4]

```
void gg::GgMatrix::projection (
    GgVector & c,
    const GgVector & v ) const [inline]
```

ベクトルに対して投影変換を行う.

引数

<i>c</i>	変換結果を格納する GgVector 型の変数.
<i>v</i>	元のベクトルの GgVector 型の変数.

gg.h の 2470 行目に定義がります。

8.5.3.77 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

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

gg.h の 2462 行目に定義がります。

8.5.3.78 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

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 GgVector 型の変数。

gg.h の 2454 行目に定義がります。

8.5.3.79 projection() [4/4]

```
void gg::GgMatrix::projection (
    GLfloat * c,
    const GLfloat * v ) const [inline]
```

ベクトルに対して投影変換を行う。

引数

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

gg.h の 2446 行目に定義があります。

8.5.3.80 `rotate()` [1/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

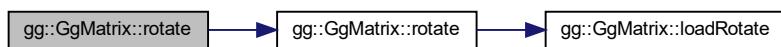
<i>r</i>	回転軸の方向ベクトルと回転角を格納した <code>GgVector</code> 型の変数).
----------	--

戻り値

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

gg.h の 2322 行目に定義があります。

呼び出し関係図:



8.5.3.81 `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

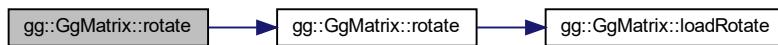
<i>r</i>	回転軸の方向ベクトルを格納した GgVector 型の変数.
<i>a</i>	回転角.

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 2306 行目に定義があります。

呼び出し関係図:



8.5.3.82 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

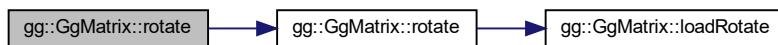
<i>r</i>	回転軸の方向ベクトルと回転角を格納した GLfloat 型の 4 要素の配列変数 (x, y, z, a).
----------	--

戻り値

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

gg.h の 2314 行目に定義があります。

呼び出し関係図:



8.5.3.83 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 <code>GLfloat</code> 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 2297 行目に定義があります。

呼び出し関係図:



8.5.3.84 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

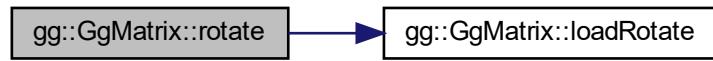
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

(x, y, z) を軸にさらに a 回転した変換行列.

gg.h の 2287 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.85 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

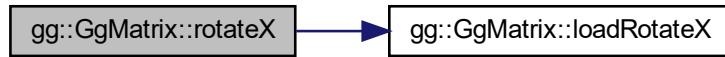
a	回転角.
---	------

戻り値

x 軸中心にさらに a 回転した変換行列.

gg.h の 2257 行目に定義があります。

呼び出し関係図:



8.5.3.86 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 2266 行目に定義があります。

呼び出し関係図:



8.5.3.87 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 2275 行目に定義があります。

呼び出し関係図:



8.5.3.88 scale() [1/3]

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

拡大縮小変換を乗じた結果を返す.

引数

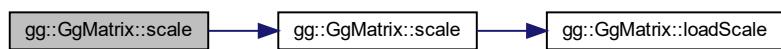
s	拡大率の GgVector 型の変数.
---	---------------------

戻り値

拡大縮小した結果の変換行列.

gg.h の 2249 行目に定義があります。

呼び出し関係図:



8.5.3.89 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

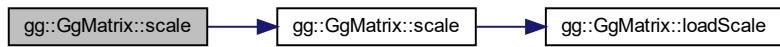
<i>s</i>	拡大率の GLfloat 型の 3 要素の配列変数 (<i>x</i> , <i>y</i> , <i>z</i>).
----------	---

戻り値

拡大縮小した結果の変換行列.

gg.h の 2241 行目に定義があります。

呼び出し関係図:



8.5.3.90 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 2232 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.91 subtract() [1/2]

```
GgMatrix gg::GgMatrix::subtract (
    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

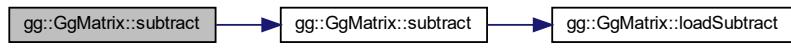
<i>m</i>	GgMatrix 型の変数。
----------	----------------

戻り値

変換行列に *m* を引いた GgMatrix 型の値。

gg.h の 1867 行目に定義があります。

呼び出し関係図:



8.5.3.92 subtract() [2/2]

```
GgMatrix gg::GgMatrix::subtract (
    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値を返す。

引数

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

戻り値

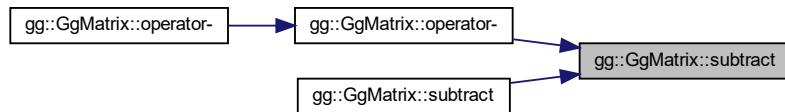
変換行列に *a* を引いた GgMatrix 型の値.

gg.h の 1858 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.93 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

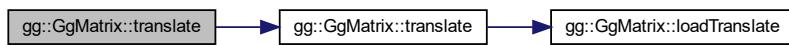
<i>t</i>	移動量の GgVector 型の変数.
----------	---------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2221 行目に定義がります。

呼び出し関係図:



8.5.3.94 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 2213 行目に定義がります。

呼び出し関係図:



8.5.3.95 `translate()` [3/3]

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

平行移動変換を乗じた結果を返す。

引数

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

戻り値

平行移動した結果の変換行列.

gg.h の 2204 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.3.96 transpose()

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

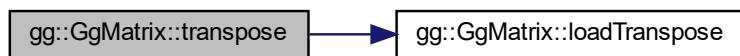
転置行列を返す。

戻り値

転置行列。

gg.h の 2421 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.5.4 フレンドと関連関数の詳解

8.5.4.1 GgQuaternion

```
friend class GgQuaternion [friend]
```

gg.h の 1727 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::GgNormalTexture クラス

法線マップ.

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

公開メンバ関数

- [GgNormalTexture \(\)](#)
コンストラクタ.
- [GgNormalTexture \(const GLubyte *image, GLsizei width, GLsizei height, GLenum format=GL_RED, GLfloat nz=1.0f, GLenum internal=GL_RGBA\)](#)
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.
- [GgNormalTexture \(const std::string &name, GLfloat nz=1.0f, GLenum internal=GL_RGBA\)](#)
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.
- [virtual ~GgNormalTexture \(\)](#)
デストラクタ.
- [void load \(const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format=GL_RED, GLfloat nz=1.0f, GLenum internal=GL_RGBA\)](#)
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.
- [void load \(const std::string &name, GLfloat nz=1.0f, GLenum internal=GL_RGBA\)](#)
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.

8.6.1 詳解

法線マップ.

高さマップ（グレイスケール画像）を読み込んで法線マップのテクスチャを作成する.

gg.h の 4131 行目に定義があります。

8.6.2 構築子と解体子

8.6.2.1 GgNormalTexture() [1/3]

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

コンストラクタ.

gg.h の 4139 行目に定義があります。

8.6.2.2 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]
```

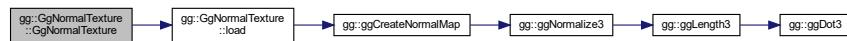
メモリ上のデータから法線マップのテクスチャを作成するコンストラクタ.

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 <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 4150 行目に定義があります。

呼び出し関係図:



8.6.2.3 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 <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 4167 行目に定義があります。

呼び出し関係図:



8.6.2.4 ~GgNormalTexture()

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

デストラクタ。

gg.h の 4178 行目に定義があります。

8.6.3 関数詳解

8.6.3.1 load() [1/2]

```
void gg::GgNormalTexture::load(
    const GLubyte * hmap,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_RED,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA ) [inline]
```

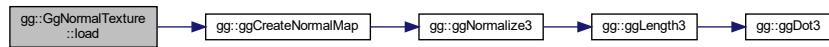
メモリ上のデータから法線マップのテクスチャを作成する。

引数

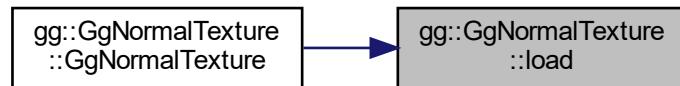
<i>hmap</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない。
<i>width</i>	テクスチャとして用いる画像データの横幅。
<i>height</i>	テクスチャとして用いる画像データの高さ。
<i>format</i>	テクスチャとして用いる画像データのフォーマット (<code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code> , <code>GL_RGBA</code>).
<i>nz</i>	法線マップの z 成分の値。
<i>internal</i>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

gg.h の 4189 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:



8.6.3.2 load() [2/2]

```

void gg::GgNormalTexture::load (
    const std::string & name,
    GLfloat nz = 1.0f,
    GLenum internal = GL_RGBA )
  
```

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。

引数

<i>name</i>	画像ファイル名(1 チャネルの TGA 画像)。
<i>nz</i>	法線マップの z 成分の値。
<i>internal</i>	テクスチャの内部フォーマット。

gg.cpp の 3203 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:

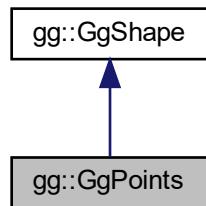
- [gg.h](#)
- [gg.cpp](#)

8.7 gg::GgPoints クラス

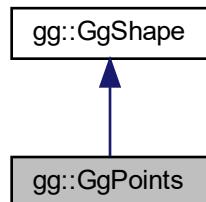
点.

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

gg::GgPoints の継承関係図



gg::GgPoints 連携図



公開 メンバ関数

- [GgPoints \(GLenum mode=GL_POINTS\)](#)
コンストラクタ.
- [GgPoints \(const GgVector *pos, GLsizei countv, GLenum mode=GL_POINTS, GLenum usage=GL_STATIC_DRAW\)](#)
コンストラクタ.
- virtual [~GgPoints \(\)](#)
デストラクタ.
- const GLsizei & [getCount \(\) const](#)
データの数を取り出す.
- const GLuint & [getBuffer \(\) const](#)
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.

- void **send** (const **GgVector** *pos, GLint first=0, GLsizei count=0) const
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.
- void **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.
- virtual void **draw** (GLint first=0, GLsizei count=0) const
点の描画.

8.7.1 詳解

点.

gg.h の 4741 行目に定義があります。

8.7.2 構築子と解体子

8.7.2.1 GgPoints() [1/2]

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

コンストラクタ.

gg.h の 4750 行目に定義があります。

8.7.2.2 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 4760 行目に定義があります。

8.7.2.3 ~GgPoints()

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

デストラクタ。

gg.h の 4772 行目に定義があります。

8.7.3 関数詳解

8.7.3.1 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 5061 行目に定義があります。

呼び出し関係図:



8.7.3.2 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 4785 行目に定義があります。

8.7.3.3 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 4778 行目に定義があります。

8.7.3.4 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 5048 行目に定義があります。

8.7.3.5 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 4794 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:

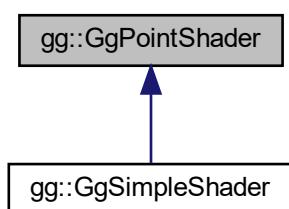
- [gg.h](#)
- [gg.cpp](#)

8.8 gg::GgPointShader クラス

点のシェーダ.

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

gg::GgPointShader の継承関係図



公開メンバ関数

- **GgPointShader ()**
コンストラクタ.
- **GgPointShader (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvary-ing=0, const char *const *varyings=nullptr)**
コンストラクタ
- **virtual ~GgPointShader ()**
デストラクタ.
- **bool load (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvary-ing=0, const char *const *varyings=nullptr)**
シェーダのソースファイルを読み込む.
- **virtual void loadProjectionMatrix (const GLfloat *mp) const**
投影変換行列を設定する.
- **virtual void loadProjectionMatrix (const GgMatrix &mp) const**
投影変換行列を設定する.
- **virtual void loadModelviewMatrix (const GLfloat *mv) const**
モデルビュー変換行列を設定する.
- **virtual void loadModelviewMatrix (const GgMatrix &mv) const**
モデルビュー変換行列を設定する.
- **virtual void loadMatrix (const GLfloat *mp, const GLfloat *mv) const**
投影変換行列とモデルビュー変換行列を設定する.
- **virtual void loadMatrix (const GgMatrix &mp, const GgMatrix &mv) const**
投影変換行列とモデルビュー変換行列を設定する.
- **virtual void use () const**
シェーダプログラムの使用を開始する.
- **void use (const GLfloat *mp) const**
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.
- **void use (const GgMatrix &mp) const**
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.
- **void use (const GLfloat *mp, const GLfloat *mv) const**
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.
- **void use (const GgMatrix &mp, const GgMatrix &mv) const**
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.
- **void unuse () const**
シェーダプログラムの使用を終了する.
- **GLuint get () const**
シェーダのプログラム名を得る.

8.8.1 詳解

点のシェーダ.

gg.h の 5220 行目に定義があります。

8.8.2 構築子と解体子

8.8.2.1 GgPointShader() [1/2]

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

コンストラクタ。

gg.h の 5234 行目に定義があります。

8.8.2.2 GgPointShader() [2/2]

```
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(0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(0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 5246 行目に定義があります。

8.8.2.3 ~GgPointShader()

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

デストラクタ。

gg.h の 5259 行目に定義があります。

8.8.3 関数詳解

8.8.3.1 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 5386 行目に定義があります。

8.8.3.2 load()

```
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

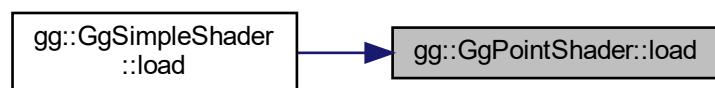
<i>vert</i>	バーテックスシェーダのソースファイル名。
<i>frag</i>	フラグメントシェーダのソースファイル名(0なら不使用)。
<i>geom</i>	ジオメトリシェーダのソースファイル名(0なら不使用)。
<i>nvarying</i>	フィードバックする varying 変数の数(0なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト。

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 5270 行目に定義があります。

被呼び出し関係図:



8.8.3.3 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。

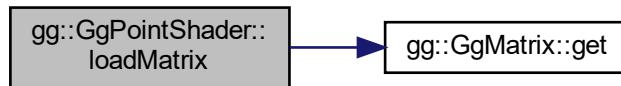
引数

<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 5335 行目に定義があります。

呼び出し関係図:



8.8.3.4 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 5326 行目に定義があります。

8.8.3.5 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。

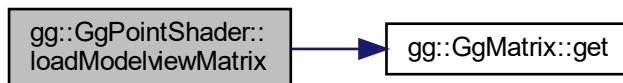
引数

<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 5318 行目に定義があります。

呼び出し関係図:



8.8.3.6 loadModelviewMatrix() [2/2]

```
virtual void gg::GgPointShader::loadModelviewMatrix (
    const GLfloat * mv ) const [inline], [virtual]
```

モデルビュー変換行列を設定する。

引数

<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
-----------	--

gg::GgSimpleShaderで再実装されています。

gg.h の 5311 行目に定義があります。

8.8.3.7 loadProjectionMatrix() [1/2]

```
virtual void gg::GgPointShader::loadProjectionMatrix (
    const GgMatrix & mp ) const [inline], [virtual]
```

投影変換行列を設定する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
-----------	--------------------

gg.h の 5304 行目に定義があります。

呼び出し関係図:



8.8.3.8 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 5297 行目に定義があります。

8.8.3.9 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 5379 行目に定義があります。

8.8.3.10 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 5341 行目に定義があります。

8.8.3.11 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 5356 行目に定義があります。

呼び出し関係図:



8.8.3.12 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 5373 行目に定義があります。

呼び出し関係図:



8.8.3.13 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 5348 行目に定義があります。

8.8.3.14 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 5364 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](#)

8.9 gg::GgQuaternion クラス

四元数.

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

公開 メンバ関数

- **GgQuaternion ()**
コンストラクタ.
- **GgQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)**
コンストラクタ.
- **GgQuaternion (const GgVector &v)**
コンストラクタ.
- **GgQuaternion (const GLfloat *a)**
コンストラクタ.
- **GgQuaternion (const GgQuaternion &q)**
コピーコンストラクタ.
- **~GgQuaternion ()**
デストラクタ.
- **GLfloat norm () const**
四元数のノルムを求める.
- **GgQuaternion & load (GLfloat x, GLfloat y, GLfloat z, GLfloat w)**
四元数を格納する.
- **GgQuaternion & load (const GgVector &v)**
四元数を格納する.
- **GgQuaternion & load (const GLfloat *a)**
四元数を格納する.
- **GgQuaternion & load (const GgQuaternion &q)**
四元数を格納する.
- **GgQuaternion & loadAdd (GLfloat x, GLfloat y, GLfloat z, GLfloat w)**
四元数に別の四元数を加算した結果を格納する.
- **GgQuaternion & loadAdd (const GgVector &v)**
四元数に別の四元数を加算した結果を格納する.
- **GgQuaternion & loadAdd (const GLfloat *a)**
四元数に別の四元数を加算した結果を格納する.
- **GgQuaternion & loadAdd (const GgQuaternion &q)**
四元数に別の四元数を加算した結果を格納する.
- **GgQuaternion & loadSubtract (GLfloat x, GLfloat y, GLfloat z, GLfloat w)**
四元数から別の四元数を減算した結果を格納する.
- **GgQuaternion & loadSubtract (const GgVector &v)**
四元数から別の四元数を減算した結果を格納する.
- **GgQuaternion & loadSubtract (const GLfloat *a)**
四元数から別の四元数を減算した結果を格納する.
- **GgQuaternion & loadSubtract (const GgQuaternion &q)**
四元数から別の四元数を減算した結果を格納する.
- **GgQuaternion & loadMultiply (GLfloat x, GLfloat y, GLfloat z, GLfloat w)**
四元数に別の四元数を乗算した結果を格納する.
- **GgQuaternion & loadMultiply (const GgVector &v)**
四元数に別の四元数を乗算した結果を格納する.

- [GgQuaternion & loadMultiply \(const GLfloat *a\)](#)
四元数に別の四元数を乗算した結果を格納する.
- [GgQuaternion & loadMultiply \(const GgQuaternion &q\)](#)
四元数に別の四元数を乗算した結果を格納する.
- [GgQuaternion & loadDivide \(GLfloat x, GLfloat y, GLfloat z, GLfloat w\)](#)
四元を別の四元数で除算した結果を格納する.
- [GgQuaternion & loadDivide \(const GgVector &v\)](#)
四元を別の四元数で除算した結果を格納する.
- [GgQuaternion & loadDivide \(const GLfloat *a\)](#)
四元を別の四元数で除算した結果を格納する.
- [GgQuaternion & loadDivide \(const GgQuaternion &q\)](#)
四元を別の四元数で除算した結果を格納する.
- [GgQuaternion add \(GLfloat x, GLfloat y, GLfloat z, GLfloat w\) const](#)
四元数に別の四元数を加算した結果を返す.
- [GgQuaternion add \(const GgVector &v\) const](#)
四元数に別の四元数を加算した結果を返す.
- [GgQuaternion add \(const GLfloat *a\) const](#)
四元数に別の四元数を加算した結果を返す.
- [GgQuaternion add \(const GgQuaternion &q\) const](#)
四元数に別の四元数を加算した結果を返す.
- [GgQuaternion subtract \(GLfloat x, GLfloat y, GLfloat z, GLfloat w\) const](#)
四元数から別の四元数を減算した結果を返す.
- [GgQuaternion subtract \(const GgVector &v\) const](#)
四元数から別の四元数を減算した結果を返す.
- [GgQuaternion subtract \(const GLfloat *a\) const](#)
四元数から別の四元数を減算した結果を返す.
- [GgQuaternion subtract \(const GgQuaternion &q\) const](#)
四元数から別の四元数を減算した結果を返す.
- [GgQuaternion multiply \(GLfloat x, GLfloat y, GLfloat z, GLfloat w\) const](#)
四元数に別の四元数を乗算した結果を返す.
- [GgQuaternion multiply \(const GgVector &v\) const](#)
四元数に別の四元数を乗算した結果を返す.
- [GgQuaternion multiply \(const GLfloat *a\) const](#)
四元数に別の四元数を乗算した結果を返す.
- [GgQuaternion multiply \(const GgQuaternion &q\) const](#)
四元数に別の四元数を乗算した結果を返す.
- [GgQuaternion divide \(GLfloat x, GLfloat y, GLfloat z, GLfloat w\) const](#)
四元数を別の四元数で除算した結果を返す.
- [GgQuaternion divide \(const GgVector &v\) const](#)
四元数を別の四元数で除算した結果を返す.
- [GgQuaternion divide \(const GLfloat *a\) const](#)
四元数を別の四元数で除算した結果を返す.
- [GgQuaternion divide \(const GgQuaternion &q\) const](#)
四元数を別の四元数で除算した結果を返す.
- [GgQuaternion & operator= \(const GLfloat *a\)](#)
- [GgQuaternion & operator= \(const GgQuaternion &q\)](#)
- [GgQuaternion & operator+= \(const GLfloat *a\)](#)
- [GgQuaternion & operator+= \(const GgQuaternion &q\)](#)
- [GgQuaternion & operator-= \(const GLfloat *a\)](#)
- [GgQuaternion & operator-= \(const GgQuaternion &q\)](#)
- [GgQuaternion & operator*=\(const GLfloat *a\)](#)

- `GgQuaternion & operator*=(const GgQuaternion &q)`
- `GgQuaternion & operator/=(const GLfloat *a)`
- `GgQuaternion & operator/=(const GgQuaternion &q)`
- `GgQuaternion operator+(const GLfloat *a) const`
- `GgQuaternion operator+(const GgQuaternion &q) const`
- `GgQuaternion operator-(const GLfloat *a) const`
- `GgQuaternion operator-(const GgQuaternion &q) const`
- `GgQuaternion operator*(const GLfloat *a) const`
- `GgQuaternion operator*(const GgQuaternion &q) const`
- `GgQuaternion operator/(const GLfloat *a) const`
- `GgQuaternion operator/(const GgQuaternion &q) const`
- `GgQuaternion & loadMatrix(const GLfloat *a)`
回転の変換行列を表す四元数を格納する.
- `GgQuaternion & loadMatrix(const GgMatrix &m)`
回転の変換行列 m を表す四元数を格納する.
- `GgQuaternion & loadIdentity()`
単位元を格納する.
- `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.
- `GgQuaternion & loadRotate(const GLfloat *v, GLfloat a)`
 $(v[0], v[1], v[2])$ を軸として角度 a 回転する四元数を格納する.
- `GgQuaternion & loadRotate(const GLfloat *v)`
 $(v[0], v[1], v[2])$ を軸として角度 $v[3]$ 回転する四元数を格納する.
- `GgQuaternion & loadRotateX(GLfloat a)`
 x 軸中心に角度 a 回転する四元数を格納する.
- `GgQuaternion & loadRotateY(GLfloat a)`
 y 軸中心に角度 a 回転する四元数を格納する.
- `GgQuaternion & loadRotateZ(GLfloat a)`
 z 軸中心に角度 a 回転する四元数を格納する.
- `GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const`
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.
- `GgQuaternion rotate(const GLfloat *v, GLfloat a) const`
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.
- `GgQuaternion rotate(const GLfloat *v) const`
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.
- `GgQuaternion rotateX(GLfloat a) const`
四元数を x 軸中心に角度 a 回転した四元数を返す.
- `GgQuaternion rotateY(GLfloat a) const`
四元数を y 軸中心に角度 a 回転した四元数を返す.
- `GgQuaternion rotateZ(GLfloat a) const`
四元数を z 軸中心に角度 a 回転した四元数を返す.
- `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.
- `GgQuaternion & loadEuler(const GLfloat *e)`
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.
- `GgQuaternion euler(GLfloat heading, GLfloat pitch, GLfloat roll) const`
四元数をオイラー角 ($heading, pitch, roll$) で回転した四元数を返す.
- `GgQuaternion euler(const GLfloat *e) const`
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.
- `GgQuaternion & loadSlerp(const GLfloat *a, const GLfloat *b, GLfloat t)`
球面線形補間の結果を格納する.

- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
球面線形補間の結果を格納する.
- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
球面線形補間の結果を格納する.
- `GgQuaternion & loadSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
球面線形補間の結果を格納する.
- `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.
- `GgQuaternion & loadNormalize (const GgQuaternion &q)`
引数に指定した四元数を正規化して格納する.
- `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.
- `GgQuaternion & loadConjugate (const GgQuaternion &q)`
引数に指定した四元数の共役四元数を格納する.
- `GgQuaternion & loadInvert (const GLfloat *a)`
引数に指定した四元数の逆元を格納する.
- `GgQuaternion & loadInvert (const GgQuaternion &q)`
引数に指定した四元数の逆元を格納する.
- `GgQuaternion slerp (GLfloat *a, GLfloat t) const`
球面線形補間の結果を返す.
- `GgQuaternion slerp (const GgQuaternion &q, GLfloat t) const`
球面線形補間の結果を返す.
- `GgQuaternion normalize () const`
正規化する.
- `GgQuaternion conjugate () const`
共役四元数に変換する.
- `GgQuaternion invert () const`
逆元に変換する.
- `const GLfloat * get () const`
四元数を取り出す.
- `void get (GLfloat *a) const`
四元数を取り出す.
- `void getMatrix (GLfloat *a) const`
四元数が表す回転の変換行列を *a* に求める.
- `void getMatrix (GgMatrix &m) const`
四元数が表す回転の変換行列を *m* に求める.
- `GgMatrix getMatrix () const`
四元数が表す回転の変換行列を取り出す.
- `void getConjugateMatrix (GLfloat *a) const`
四元数の共役が表す回転の変換行列を *a* に求める.
- `void getConjugateMatrix (GgMatrix &m) const`
四元数の共役が表す回転の変換行列を *m* に求める.
- `GgMatrix getConjugateMatrix () const`
四元数の共役が表す回転の変換行列を取り出す.

8.9.1 詳解

四元数.

gg.h の 2793 行目に定義があります。

8.9.2 構築子と解体子

8.9.2.1 GgQuaternion() [1/5]

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

コンストラクタ.

gg.h の 2813 行目に定義があります。

8.9.2.2 GgQuaternion() [2/5]

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

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 2822 行目に定義があります。

呼び出し関係図:



8.9.2.3 GgQuaternion() [3/5]

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

コンストラクタ.

引数

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

gg.h の 2829 行目に定義がります。

呼び出し関係図:



8.9.2.4 GgQuaternion() [4/5]

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

コンストラクタ.

引数

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

gg.h の 2836 行目に定義がります。

呼び出し関係図:



8.9.2.5 GgQuaternion() [5/5]

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

コピーコンストラクタ.

引数

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

gg.h の 2843 行目に定義がります。

呼び出し関係図:



8.9.2.6 ~GgQuaternion()

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

デストラクタ.

gg.h の 2849 行目に定義がります。

8.9.3 関数詳解

8.9.3.1 add() [1/4]

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

四元数に別の四元数を加算した結果を返す.

引数

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

戻り値

`q` を加えた四元数.

gg.h の 3085 行目に定義がります。

呼び出し関係図:



8.9.3.2 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

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 3069 行目に定義があります。

呼び出し関係図:



8.9.3.3 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

a を加えた四元数.

gg.h の 3077 行目に定義があります。

呼び出し関係図:



8.9.3.4 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

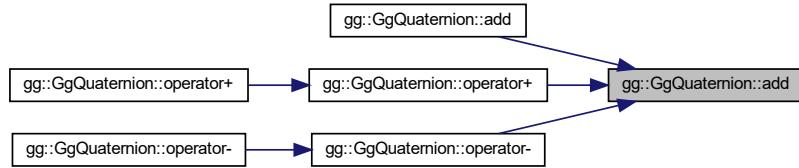
x	加える四元数の x 要素.
y	加える四元数の y 要素.
z	加える四元数の z 要素.
w	加える四元数の w 要素.

戻り値

(x, y, z, w) を加えた四元数.

gg.h の 3056 行目に定義があります。

被呼び出し関係図:



8.9.3.5 conjugate()

`GgQuaternion gg::GgQuaternion::conjugate () const [inline]`

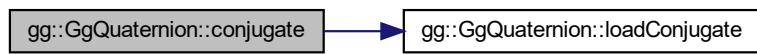
共役四元数に変換する。

戻り値

共役四元数。

gg.h の 3545 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



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

戻り値

`q` で割った四元数.

gg.h の 3202 行目に定義がります。

呼び出し関係図:



8.9.3.7 divide() [2/4]

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

四元数を別の四元数で除算した結果を返す.

引数

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

戻り値

`v` で割った四元数.

gg.h の 3183 行目に定義がります。

呼び出し関係図:



8.9.3.8 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 3191 行目に定義があります。

呼び出し関係図:



8.9.3.9 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

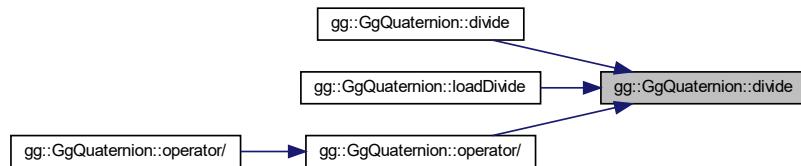
x	割る四元数の x 要素.
y	割る四元数の y 要素.
z	割る四元数の z 要素.
w	割る四元数の w 要素.

戻り値

(x, y, z, w) を割った四元数.

gg.h の 3174 行目に定義があります。

被呼び出し関係図:



8.9.3.10 euler() [1/2]

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

四元数をオイラー角 ($e[0], e[1], e[2]$) で回転した四元数を返す.

引数

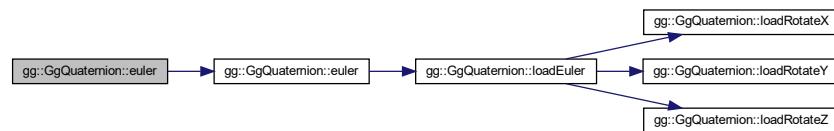
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
---	---

戻り値

回転した四元数.

gg.h の 3427 行目に定義があります。

呼び出し関係図:



8.9.3.11 euler() [2/2]

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

四元数をオイラー角 (heading, pitch, roll) で回転した四元数を返す.

引数

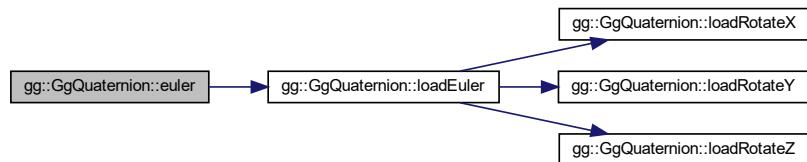
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転した四元数.

gg.h の 3418 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.12 get() [1/2]

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

四元数を取り出す。

戻り値

四元数を表す GLfloat 型の 4 要素の配列変数。

gg.h の 3563 行目に定義があります。

被呼び出し関係図:



8.9.3.13 get() [2/2]

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

四元数を取り出す。

引数

a	四元数を格納する GLfloat 型の 4 要素の配列変数。
---	--------------------------------

gg.h の 3570 行目に定義があります。

8.9.3.14 getConjugateMatrix() [1/3]

```
GgMatrix gg::GgQuaternion::getConjugateMatrix( ) const [inline]
```

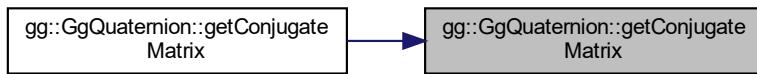
四元数の共役が表す回転の変換行列を取り出す。

戻り値

回転の変換を表す [GgMatrix](#) 型の変換行列.

gg.h の 3619 行目に定義があります。

被呼び出し関係図:



8.9.3.15 getConjugateMatrix() [2/3]

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

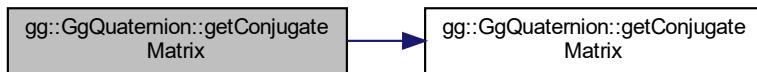
四元数の共役が表す回転の変換行列を *m* に求める.

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 3612 行目に定義があります。

呼び出し関係図:



8.9.3.16 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

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 3603 行目に定義があります。

呼び出し関係図:



8.9.3.17 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 3594 行目に定義があります。

被呼び出し関係図:



8.9.3.18 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

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

gg.h の 3587 行目に定義があります。

呼び出し関係図:



8.9.3.19 getMatrix() [3/3]

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

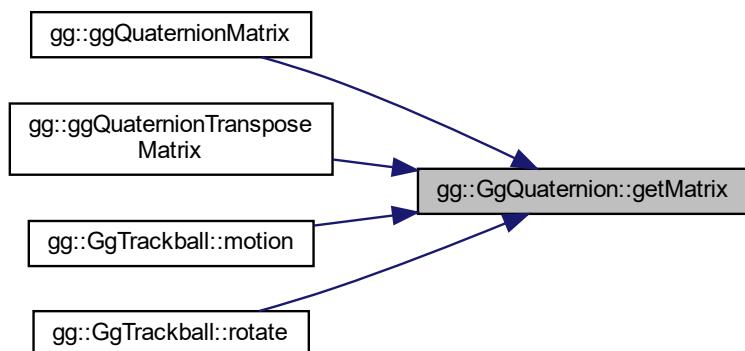
四元数が表す回転の変換行列を *a* に求める。

引数

<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 3580 行目に定義があります。

被呼び出し関係図:



8.9.3.20 invert()

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

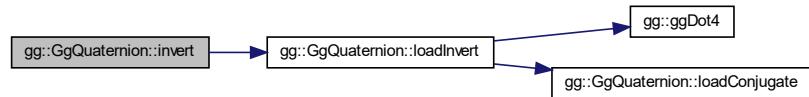
逆元に変換する。

戻り値

四元数の逆元。

gg.h の 3554 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.21 load() [1/4]

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

四元数を格納する。

引数

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

戻り値

設定した四元数.

gg.h の 2895 行目に定義があります。

呼び出し関係図:



8.9.3.22 load() [2/4]

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

四元数を格納する.

引数

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

戻り値

設定した四元数.

gg.h の 2878 行目に定義があります。

8.9.3.23 load() [3/4]

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

四元数を格納する.

引数

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

戻り値

設定した四元数.

gg.h の 2887 行目に定義があります。

呼び出し関係図:



8.9.3.24 load() [4/4]

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

四元数を格納する.

引数

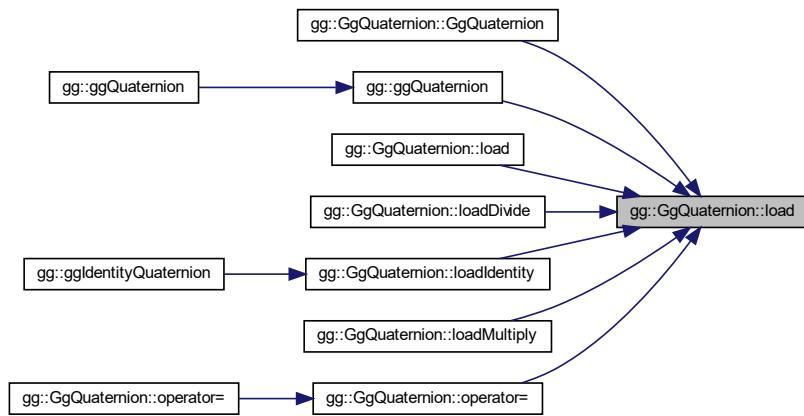
<i>x</i>	四元数の x 要素.
<i>y</i>	四元数の y 要素.
<i>z</i>	四元数の z 要素.
<i>w</i>	四元数の w 要素.

戻り値

設定した四元数.

gg.h の 2866 行目に定義があります。

被呼び出し関係図:



8.9.3.25 loadAdd() [1/4]

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

四元数に別の四元数を加算した結果を格納する.

引数

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

戻り値

q を加えた四元数.

gg.h の 2934 行目に定義があります。

呼び出し関係図:



8.9.3.26 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 2918 行目に定義があります。

呼び出し関係図:



8.9.3.27 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 2926 行目に定義があります。

呼び出し関係図:



8.9.3.28 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

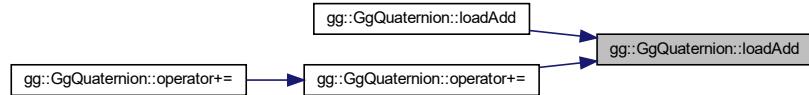
x	加える四元数の x 要素.
y	加える四元数の y 要素.
z	加える四元数の z 要素.
w	加える四元数の w 要素.

戻り値

(x, y, z, w) を加えた四元数.

gg.h の 2906 行目に定義があります。

被呼び出し関係図:



8.9.3.29 loadConjugate() [1/2]

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

引数に指定した四元数の共役四元数を格納する。

引数

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

戻り値

共役四元数。

gg.h の 3494 行目に定義があります。

呼び出し関係図:



8.9.3.30 loadConjugate() [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadConjugate (
    const GLfloat * a )
```

引数に指定した四元数の共役四元数を格納する。

引数

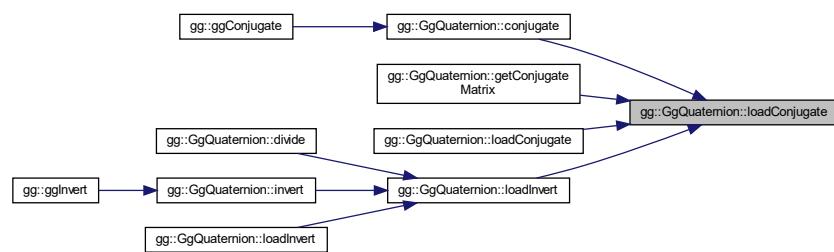
<code>a</code>	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------------	--------------------------------

戻り値

共役四元数.

gg.cpp の 4900 行目に定義があります。

被呼び出し関係図:



8.9.3.31 loadDivide() [1/4]

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

四元を別の四元数で除算した結果を格納する.

引数

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

戻り値

`q` で割った四元数.

gg.h の 3045 行目に定義があります。

呼び出し関係図:



8.9.3.32 loadDivide() [2/4]

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

四元を別の四元数で除算した結果を格納する。

引数

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

戻り値

v で割った四元数。

gg.h の 3029 行目に定義があります。

呼び出し関係図:



8.9.3.33 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

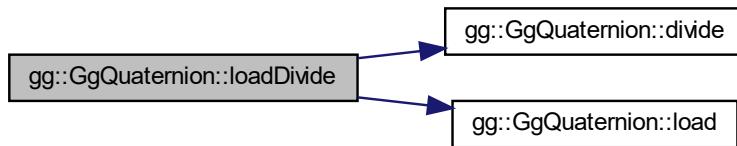
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a で割った四元数.

gg.h の 3037 行目に定義があります。

呼び出し関係図:



8.9.3.34 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

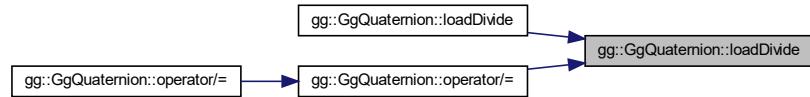
x	割る四元数の x 要素.
y	割る四元数の y 要素.
z	割る四元数の z 要素.
w	割る四元数の w 要素.

戻り値

(x, y, z, w) を割った四元数.

gg.h の 3020 行目に定義があります。

被呼び出し関係図:



8.9.3.35 loadEuler() [1/2]

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

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

引数

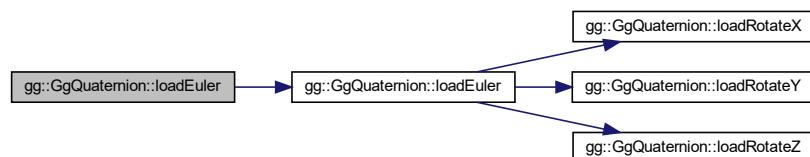
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

格納した回転を表す四元数。

gg.h の 3408 行目に定義があります。

呼び出し関係図:



8.9.3.36 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

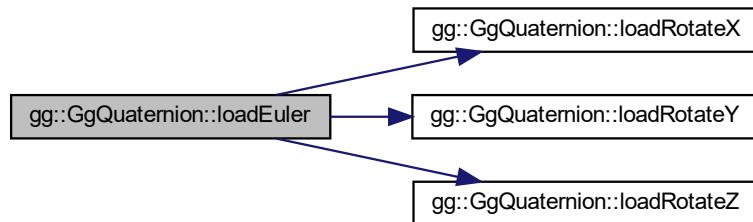
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

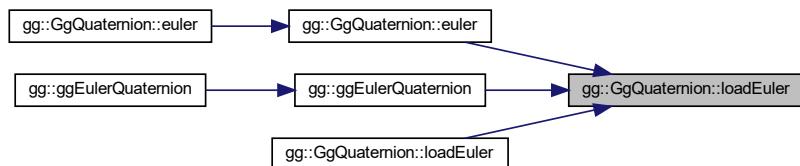
格納した回転を表す四元数.

gg.cpp の 4869 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.37 loadIdentity()

`GgQuaternion& gg::GgQuaternion::loadIdentity () [inline]`

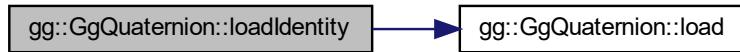
単位元を格納する.

戻り値

格納された単位元.

gg.h の 3300 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.38 loadInvert() [1/2]

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

引数に指定した四元数の逆元を格納する.

引数

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

戻り値

四元数の逆元.

gg.h の 3507 行目に定義があります。

呼び出し関係図:



8.9.3.39 `loadInvert()` [2/2]

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

引数に指定した四元数の逆元を格納する。

引数

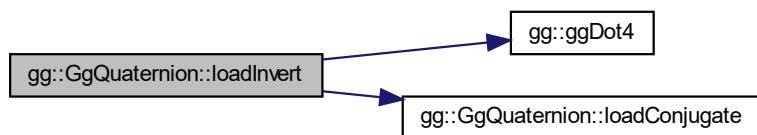
a	四元数を格納した GLfloat 型の 4 要素の配列変数。
---	--------------------------------

戻り値

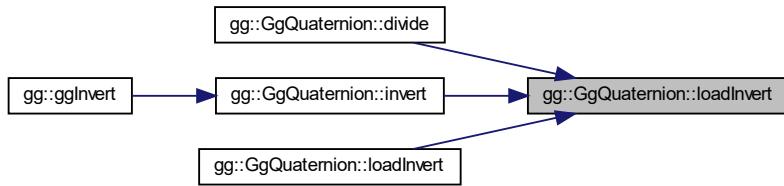
四元数の逆元。

gg.cpp の 4914 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



8.9.3.40 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

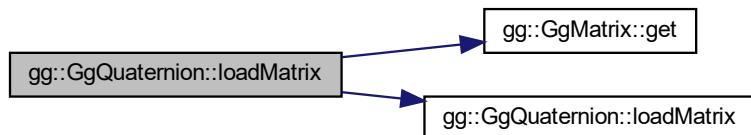
m	Ggmatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 3293 行目に定義があります。

呼び出し関係図:



8.9.3.41 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

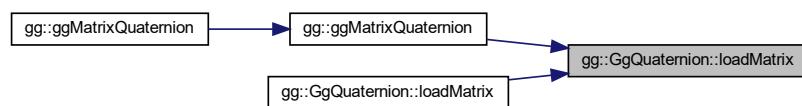
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 3284 行目に定義があります。

呼び出し関係図:



8.9.3.42 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 3009 行目に定義があります。

呼び出し関係図:



8.9.3.43 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 2993 行目に定義があります。

呼び出し関係図:



8.9.3.44 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 3001 行目に定義があります。

呼び出し関係図:



8.9.3.45 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

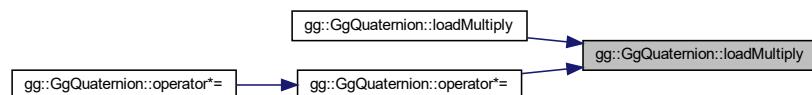
<i>x</i>	掛ける四元数の <i>x</i> 要素.
<i>y</i>	掛ける四元数の <i>y</i> 要素.
<i>z</i>	掛ける四元数の <i>z</i> 要素.
<i>w</i>	掛ける四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を掛けた四元数。

gg.h の 2984 行目に定義があります。

被呼び出し関係図:



8.9.3.46 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

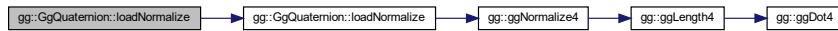
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

正規化された四元数。

gg.h の 3481 行目に定義があります。

呼び出し関係図:



8.9.3.47 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

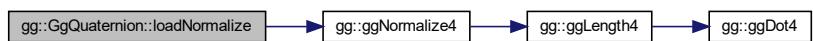
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
----------	--------------------------------

戻り値

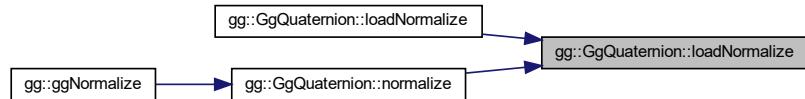
正規化された四元数。

gg.cpp の 4885 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.48 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 3325 行目に定義があります。

呼び出し関係図:



8.9.3.49 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 3317 行目に定義があります。

呼び出し関係図:



8.9.3.50 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

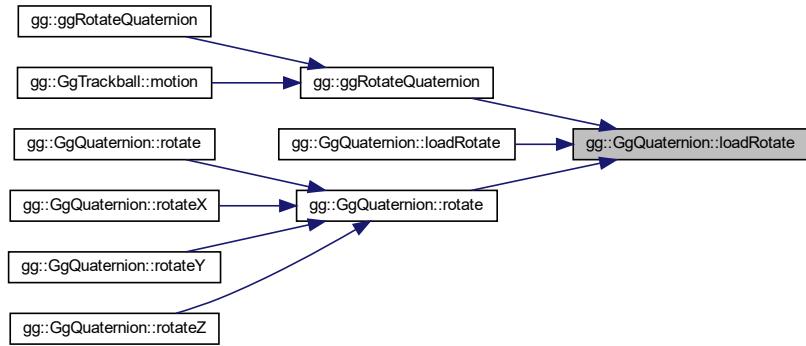
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

格納された回転を表す四元数.

gg.cpp の 4803 行目に定義があります。

被呼び出し関係図:



8.9.3.51 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

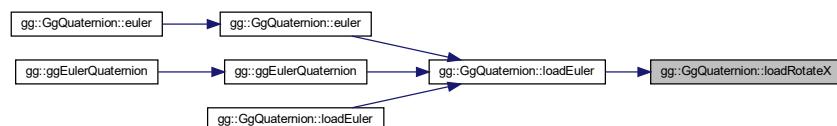
a	回転角.
-----	------

戻り値

格納された回転を表す四元数。

`gg.cpp` の 4827 行目に定義があります。

被呼び出し関係図:



8.9.3.52 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

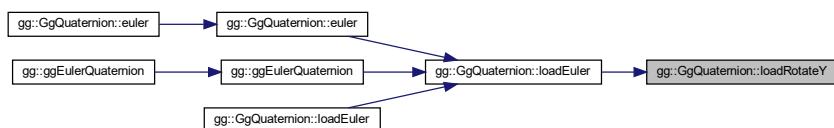
a	回転角.
----------	------

戻り値

格納された回転を表す四元数。

gg.cpp の 4841 行目に定義があります。

被呼び出し関係図:



8.9.3.53 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

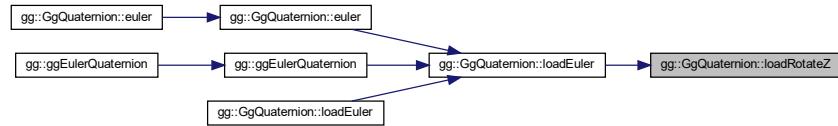
a	回転角.
----------	------

戻り値

格納された回転を表す四元数。

gg.cpp の 4855 行目に定義があります。

被呼び出し関係図:



8.9.3.54 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 3448 行目に定義があります。

呼び出し関係図:



8.9.3.55 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 3458 行目に定義があります。

呼び出し関係図:



8.9.3.56 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 3468 行目に定義があります。

呼び出し関係図:



8.9.3.57 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

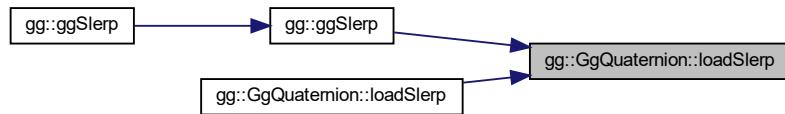
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した a, b を t で内分した四元数.

gg.h の 3437 行目に定義があります。

被呼び出し関係図:



8.9.3.58 loadSubtract() [1/4]

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

四元数から別の四元数を減算した結果を格納する.

引数

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

戻り値

q を引いた四元数.

gg.h の 2973 行目に定義がります。

呼び出し関係図:

**8.9.3.59 loadSubtract() [2/4]**

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

四元数から別の四元数を減算した結果を格納する.

引数

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

戻り値

v を引いた四元数.

gg.h の 2957 行目に定義がります。

呼び出し関係図:



8.9.3.60 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

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

戻り値

a を引いた四元数。

gg.h の 2965 行目に定義があります。

呼び出し関係図:



8.9.3.61 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

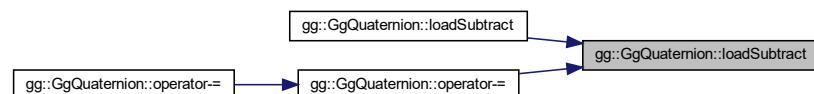
<i>x</i>	引く四元数の <i>x</i> 要素.
<i>y</i>	引く四元数の <i>y</i> 要素.
<i>z</i>	引く四元数の <i>z</i> 要素.
<i>w</i>	引く四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 2945 行目に定義があります。

被呼び出し関係図:



8.9.3.62 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 3163 行目に定義があります。

8.9.3.63 multiply() [2/4]

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

四元数に別の四元数を乗算した結果を返す.

引数

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

戻り値

v を掛けた四元数.

gg.h の 3145 行目に定義がります。

8.9.3.64 multiply() [3/4]

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

四元数に別の四元数を乗算した結果を返す.

引数

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

戻り値

a を掛けた四元数.

gg.h の 3153 行目に定義がります。

8.9.3.65 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

<i>x</i>	掛ける四元数の x 要素.
<i>y</i>	掛ける四元数の y 要素.
<i>z</i>	掛ける四元数の z 要素.
<i>w</i>	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 3136 行目に定義があります。

8.9.3.66 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 2855 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.67 normalize()

```
GgQuaternion gg::GgQuaternion::normalize () const [inline]
```

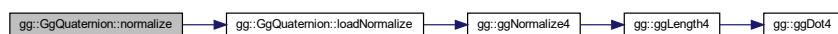
正規化する。

戻り値

正規化された四元数。

gg.h の 3536 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.68 operator*() [1/2]

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

gg.h の 3268 行目に定義があります。

呼び出し関係図:



8.9.3.69 operator*() [2/2]

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

gg.h の 3264 行目に定義がります。

呼び出し関係図:

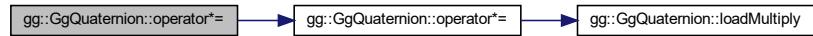


8.9.3.70 operator*=(() [1/2]

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

gg.h の 3236 行目に定義がります。

呼び出し関係図:



8.9.3.71 operator*=(() [2/2]

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

gg.h の 3232 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:

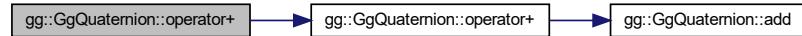


8.9.3.72 operator+() [1/2]

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

gg.h の 3252 行目に定義があります。

呼び出し関係図:



8.9.3.73 operator+() [2/2]

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

gg.h の 3248 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.74 operator+=() [1/2]

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

gg.h の 3220 行目に定義があります。

呼び出し関係図:



8.9.3.75 operator+=() [2/2]

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

gg.h の 3216 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

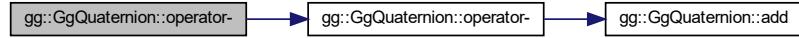


8.9.3.76 operator-() [1/2]

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

gg.h の 3260 行目に定義があります。

呼び出し関係図:



8.9.3.77 operator-() [2/2]

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

gg.h の 3256 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.78 operator-() [1/2]

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

gg.h の 3228 行目に定義があります。

呼び出し関係図:



8.9.3.79 operator-() [2/2]

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

gg.h の 3224 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

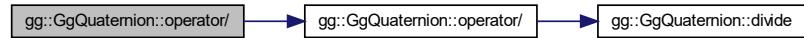


8.9.3.80 operator/() [1/2]

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

gg.h の 3276 行目に定義があります。

呼び出し関係図:



8.9.3.81 operator/() [2/2]

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

gg.h の 3272 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.82 operator/() [1/2]

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

gg.h の 3244 行目に定義があります。

呼び出し関係図:



8.9.3.83 operator/() [2/2]

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

gg.h の 3240 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

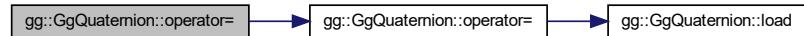


8.9.3.84 operator=() [1/2]

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

gg.h の 3212 行目に定義があります。

呼び出し関係図:



8.9.3.85 operator=() [2/2]

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

gg.h の 3208 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



8.9.3.86 rotate() [1/3]

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

四元数を ($v[0], v[1], v[2]$) を軸として角度 $v[3]$ 回転した四元数を返す.

引数

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

戻り値

回転した四元数.

gg.h の 3369 行目に定義があります。

呼び出し関係図:



8.9.3.87 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 3361 行目に定義があります。

呼び出し関係図:



8.9.3.88 `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

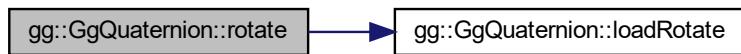
<i>x</i>	軸ベクトルの <i>x</i> 成分.
<i>y</i>	軸ベクトルの <i>y</i> 成分.
<i>z</i>	軸ベクトルの <i>z</i> 成分.
<i>a</i>	回転角.

戻り値

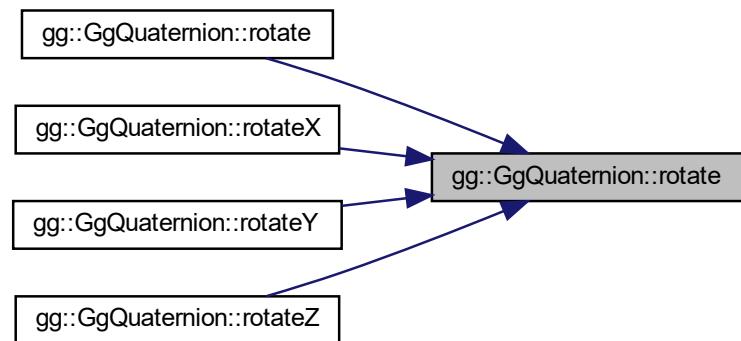
回転した四元数.

gg.h の 3351 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.9.3.89 rotateX()

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

四元数を x 軸中心に角度 a 回転した四元数を返す.

引数

a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 3377 行目に定義があります。

呼び出し関係図:



8.9.3.90 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

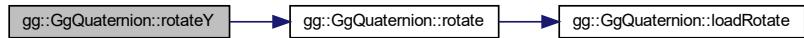
a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 3385 行目に定義があります。

呼び出し関係図:



8.9.3.91 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 3393 行目に定義があります。

呼び出し関係図:



8.9.3.92 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 3527 行目に定義があります。

8.9.3.93 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 3516 行目に定義があります。

8.9.3.94 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

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

戻り値

q を引いた四元数.

gg.h の 3125 行目に定義があります。

呼び出し関係図:



8.9.3.95 subtract() [2/4]

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

四元数から別の四元数を減算した結果を返す.

引数

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

戻り値

\mathbf{v} を引いた四元数.

gg.h の 3109 行目に定義がります。

呼び出し関係図:



8.9.3.96 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

\mathbf{a} を引いた四元数.

gg.h の 3117 行目に定義がります。

呼び出し関係図:



8.9.3.97 `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

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

戻り値

(x, y, z, w) を引いた四元数.

`gg.h` の 3096 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](#)

8.10 `gg::GgShader` クラス

シェーダの基底クラス.

```
#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)

コンストラクタ.
- **virtual ~GgShader ()**

デストラクタ.
- **GgShader (const GgShader &o)=delete**

コピー構造子は使用禁止.
- **GgShader & operator= (const GgShader &o)=delete**

代入演算子は使用禁止.
- **void use () const**

シェーダプログラムの使用を開始する.
- **void unuse () const**

シェーダプログラムの使用を終了する.
- **GLuint get () const**

シェーダのプログラム名を得る.

8.10.1 詳解

シェーダの基底クラス.

シェーダのクラスはこのクラスを派生して作る.

gg.h の 5159 行目に定義があります。

8.10.2 構築子と解体子

8.10.2.1 GgShader() [1/2]

```
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(0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(0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 5172 行目に定義があります。

8.10.2.2 ~GgShader()

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

デストラクタ。

gg.h の 5184 行目に定義があります。

8.10.2.3 GgShader() [2/2]

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

コピー構造関数は使用禁止。

8.10.3 関数詳解

8.10.3.1 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 5211 行目に定義があります。

8.10.3.2 operator=()

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

代入演算子は使用禁止。

8.10.3.3 unuse()

```
void gg::GgShader::unuse ( ) const [inline]
```

シェーダプログラムの使用を終了する。

gg.h の 5204 行目に定義があります。

8.10.3.4 use()

```
void gg::GgShader::use ( ) const [inline]
```

シェーダプログラムの使用を開始する。

gg.h の 5198 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

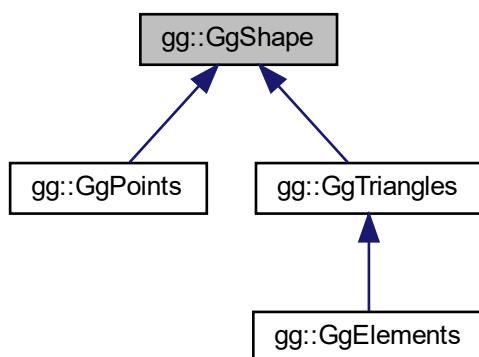
- [gg.h](#)

8.11 gg::GgShape クラス

形状データの基底クラス。

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

gg::GgShape の継承関係図



公開メンバ関数

- **GgShape (GLenum mode=0)**
コンストラクタ.
- **virtual ~GgShape ()**
デストラクタ.
- **GgShape (const GgShape &o)=delete**
コピーコンストラクタは使用禁止.
- **GgShape & operator= (const GgShape &o)=delete**
代入演算子は使用禁止.
- **const GLuint & get () const**
頂点配列オブジェクト名を取り出す.
- **void setMode (GLenum mode)**
基本図形の設定.
- **const GLenum & getMode () const**
基本図形の検査.
- **virtual void draw (GLint first=0, GLsizei count=0) const**
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.

8.11.1 詳解

形状データの基底クラス.

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 4676 行目に定義があります。

8.11.2 構築子と解体子

8.11.2.1 GgShape() [1/2]

```
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 4688 行目に定義があります。

8.11.2.2 ~GgShape()

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

デストラクタ。

gg.h の 4696 行目に定義があります。

8.11.2.3 GgShape() [2/2]

```
gg::GgShape::GgShape ( const GgShape & o ) [delete]
```

コピー構造関数は使用禁止。

8.11.3 関数詳解

8.11.3.1 draw()

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

図形の描画、派生クラスでこの手続きをオーバーライドする。

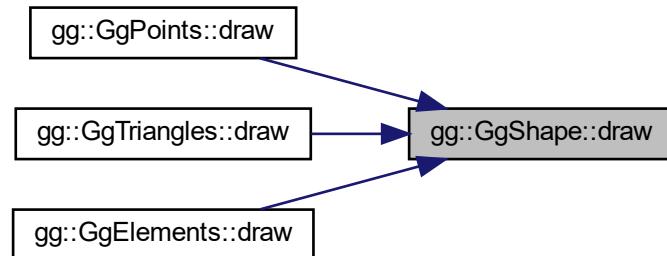
引数

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

[gg::GgElements](#), [gg::GgTriangles](#), [gg::GgPoints](#)で再実装されています。

gg.h の 4732 行目に定義があります。

被呼び出し関係図:



8.11.3.2 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 4710 行目に定義があります。

被呼び出し関係図:



8.11.3.3 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 4724 行目に定義があります。

8.11.3.4 operator=()

```
GgShape& gg::GgShape::operator= (
    const GgShape & o ) [delete]
```

代入演算子は使用禁止。

8.11.3.5 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 4717 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::GgSimpleObj クラス

Wavefront OBJ 形式のファイル (Arrays 形式)。

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

公開メンバ関数

- [GgSimpleObj](#) (const std::string &name, bool normalize=false)
コンストラクタ。
- virtual [~GgSimpleObj](#) ()
デストラクタ。
- const [GgTriangles * get](#) () const
形状データの取り出し。
- virtual void [draw](#) (GLint first=0, GLsizei count=0) const
Wavefront OBJ 形式のデータを描画する手続き。

8.12.1 詳解

Wavefront OBJ 形式のファイル (Arrays 形式)。

gg.h の 6121 行目に定義があります。

8.12.2 構築子と解体子

8.12.2.1 GgSimpleObj()

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

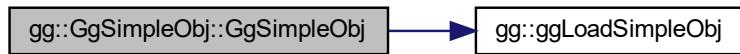
コンストラクタ。

引数

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

gg.cpp の 5843 行目に定義があります。

呼び出し関係図:



8.12.2.2 ~GgSimpleObj()

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

デストラクタ。

gg.h の 6140 行目に定義があります。

8.12.3 関数詳解

8.12.3.1 draw()

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

Wavefront OBJ 形式のデータを描画する手続き。

引数

<i>first</i>	描画する最初のパーティクル番号.
<i>count</i>	描画するパーティクルの数, 0 なら全部のパーティクルを描く.

gg.cpp の 5871 行目に定義があります。

8.12.3.2 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 6146 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

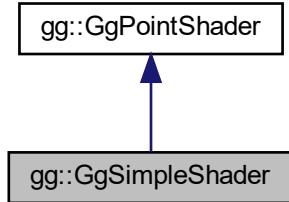
- [gg.h](#)
- [gg.cpp](#)

8.13 gg::GgSimpleShader クラス

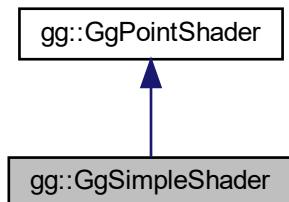
三角形に単純な陰影付けを行うシェーダ.

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

gg::GgSimpleShader の継承関係図



gg::GgSimpleShader 連携図



クラス

- struct [Light](#)
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.
- class [LightBuffer](#)
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.
- struct [Material](#)
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.
- class [MaterialBuffer](#)
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.

公開 メンバ関数

- [GgSimpleShader \(\)](#)
コンストラクタ.
- [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.
- [GgSimpleShader \(const GgSimpleShader &o\)](#)

- コピーコンストラクタ.
- `virtual ~GgSimpleShader ()`
デストラクタ.
- `GgSimpleShader & operator= (const GgSimpleShader &o)`
代入演算子
- `bool 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.
- `virtual void loadModelviewMatrix (const GLfloat *mv, const GLfloat *mn) const`
モデルビュー変換行列と法線変換行列を設定する.
- `virtual void loadModelviewMatrix (const GgMatrix &mv, const GgMatrix &mn) const`
モデルビュー変換行列と法線変換行列を設定する.
- `virtual void loadModelviewMatrix (const GLfloat *mv) const`
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.
- `virtual void loadModelviewMatrix (const GgMatrix &mv) const`
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.
- `virtual void loadMatrix (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn) const`
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.
- `virtual void loadMatrix (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn) const`
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.
- `virtual void loadMatrix (const GLfloat *mp, const GLfloat *mv) const`
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.
- `virtual void loadMatrix (const GgMatrix &mp, const GgMatrix &mv) const`
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.
- `void use () const`
シェーダプログラムの使用を開始する.
- `void use (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn) const`
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.
- `void use (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn) const`
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.
- `void use (const GLfloat *mp, const GLfloat *mv) const`
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.
- `void use (const GgMatrix &mp, const GgMatrix &mv) const`
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.
- `void use (const LightBuffer *light, GLint i=0) const`
光源を指定してシェーダプログラムの使用を開始する.
- `void use (const LightBuffer &light, GLint i=0) const`
光源を指定してシェーダプログラムの使用を開始する.
- `void use (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn, const LightBuffer *light, GLint i=0) const`
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.
- `void use (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn, const LightBuffer &light, GLint i=0) const`
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.
- `void use (const GLfloat *mp, const GLfloat *mv, const LightBuffer *light, GLint i=0) const`

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.

- void **use** (const **GgMatrix** &mp, const **GgMatrix** &mv, const **LightBuffer** &light, GLint i=0) const
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.
- void **use** (const GLfloat *mp, const **LightBuffer** *light, GLint i=0) const
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.
- void **use** (const **GgMatrix** &mp, const **LightBuffer** &light, GLint i=0) const
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.

8.13.1 詳解

三角形に単純な陰影付けを行うシェーダ.

gg.h の 5395 行目に定義があります。

8.13.2 構築子と解体子

8.13.2.1 **GgSimpleShader()** [1/3]

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

コンストラクタ.

gg.h の 5410 行目に定義があります。

8.13.2.2 **GgSimpleShader()** [2/3]

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

コンストラクタ.

引数

<i>vert</i>	パーティクルシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(0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(0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 5424 行目に定義があります。

8.13.2.3 GgSimpleShader() [3/3]

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

コピー コンストラクタ.

gg.h の 5436 行目に定義があります。

8.13.2.4 ~GgSimpleShader()

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

デストラクタ.

gg.h の 5445 行目に定義があります。

8.13.3 関数詳解

8.13.3.1 load()

```
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

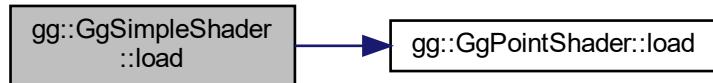
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列.
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列.
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列.
<i>nvarying</i>	Transform Feedback に使う <i>varying</i> 変数の数.
<i>varyings</i>	Transform Feedback に使う <i>varying</i> 変数の変数名の文字列の配列.

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 5826 行目に定義があります。

呼び出し関係図:



8.13.3.2 loadMatrix() [1/4]

```

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

```

投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定する。

引数

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

gg::GgPointShader を再実装しています。

gg.h の 5539 行目に定義があります。

呼び出し関係図:



8.13.3.3 loadMatrix() [2/4]

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

投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

引数

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

gg.h の 5523 行目に定義があります。

呼び出し関係図:



8.13.3.4 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 5531 行目に定義があります。

8.13.3.5 `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 5513 行目に定義があります。

8.13.3.6 `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。

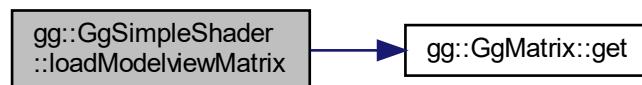
引数

<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 5504 行目に定義があります。

呼び出し関係図:



8.13.3.7 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 5490 行目に定義があります。

呼び出し関係図:



8.13.3.8 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 5497 行目に定義があります。

8.13.3.9 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 5481 行目に定義があります。

8.13.3.10 operator=()

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

代入演算子

gg.h の 5450 行目に定義があります。

8.13.3.11 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 5924 行目に定義があります。

8.13.3.12 use() [2/13]

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

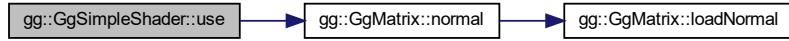
投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	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 5963 行目に定義がります。

呼び出し関係図:



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

gg.h の 5947 行目に定義がります。

呼び出し関係図:



8.13.3.14 use() [4/13]

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

```
const GgMatrix & mp,
const GgMatrix & mv,
const LightBuffer & light,
GLint i = 0 ) const [inline]
```

光源を指定し投影変換行列とモデルビュー変換行列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 6015 行目に定義があります。

8.13.3.15 use() [5/13]

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

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

引数

<i>mp</i>	GgMatrix 型の投影変換行列.
<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
<i>light</i>	光源の特性の gg::LightBuffer 構造体.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 6046 行目に定義があります。

8.13.3.16 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 6073 行目に定義があります。

8.13.3.17 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 5955 行目に定義があります。

8.13.3.18 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 5934 行目に定義があります。

8.13.3.19 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 5994 行目に定義があります。

8.13.3.20 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 6031 行目に定義があります。

8.13.3.21 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 6060 行目に定義があります。

8.13.3.22 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 5983 行目に定義があります。

8.13.3.23 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 5971 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::GgTexture クラス

テクスチャ.

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

公開メンバ関数

- [GgTexture \(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_BGR, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
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.
- [virtual ~GgTexture \(\)](#)
デストラクタ.
- [GgTexture \(const GgTexture &o\)=delete](#)
コピー構造子は使用禁止.
- [GgTexture & operator= \(const GgTexture &o\)=delete](#)
代入演算子は使用禁止.
- [void bind \(\) const](#)
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).
- [void unbind \(\) const](#)
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).
- [const GLsizei & getWidth \(\) const](#)
使用しているテクスチャの横の画素数を取り出す.
- [const GLsizei & getHeight \(\) const](#)
使用しているテクスチャの縦の画素数を取り出す.
- [void getSize \(GLsizei *size\) const](#)
使用しているテクスチャのサイズを取り出す.
- [const GLsizei * getSize \(\) const](#)
使用しているテクスチャのサイズを取り出す.
- [const GLuint & getTexture \(\) const](#)
使用しているテクスチャのテクスチャ名を得る.

8.14.1 詳解

テクスチャ.

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 3951 行目に定義があります。

8.14.2 構築子と解体子

8.14.2.1 GgTexture() [1/2]

```
gg::GgTexture::GgTexture (
    const GLvoid * image,
    GLsizei width,
    GLsizei height,
    GLenum format = GL_BGR,
    GLenum type = GL_UNSIGNED_BYTE,
    GLenum internal = GL_RGBA,
    GLenum wrap = GL_CLAMP_TO_EDGE ) [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> .

gg.h の 3969 行目に定義があります。

8.14.2.2 ~GgTexture()

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

デストラクタ。

gg.h の 3984 行目に定義があります。

8.14.2.3 GgTexture() [2/2]

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

コピーコンストラクタは使用禁止。

8.14.3 関数詳解

8.14.3.1 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 3997 行目に定義があります。

8.14.3.2 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 4017 行目に定義があります。

被呼び出し関係図:



8.14.3.3 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 4032 行目に定義があります。

8.14.3.4 getSize() [2/2]

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

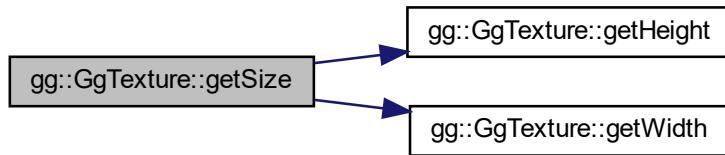
使用しているテクスチャのサイズを取り出す.

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 4024 行目に定義があります。

呼び出し関係図:



8.14.3.5 `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 4039 行目に定義があります。

8.14.3.6 `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 4010 行目に定義があります。

被呼び出し関係図:



8.14.3.7 operator=()

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

代入演算子は使用禁止.

8.14.3.8 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 4003 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](#)

8.15 gg::GgTrackball クラス

簡易トラックボール処理.

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

公開メンバ関数

- **GgTrackball ()**
 コンストラクタ.
- **virtual ~GgTrackball ()**
 デストラクタ.
- **void region (GLfloat w, GLfloat h)**
 トラックボール処理するマウスの移動範囲を指定する.
- **void region (int w, int h)**
 トラックボール処理するマウスの移動範囲を指定する.
- **void begin (GLfloat x, GLfloat y)**
 トラックボール処理を開始する.
- **void motion (GLfloat x, GLfloat y)**
 回転の変換行列を計算する.
- **void rotate (const GgQuaternion &q)**
 トラックボールの回転角を修正する.
- **void end (GLfloat x, GLfloat y)**
 トラックボール処理を停止する.
- **void reset ()**
 トラックボールをリセットする
- **const GLfloat * getStart () const**
 トラックボール処理の開始位置を取り出す.
- **const GLfloat & getStart (int direction) const**
 トラックボール処理の開始位置を取り出す.
- **void getStart (GLfloat *position) const**
 トラックボール処理の開始位置を取り出す.
- **const GLfloat * getScale () const**
 トラックボール処理の換算係数を取り出す.
- **const GLfloat getScale (int direction) const**
 トラックボール処理の換算係数を取り出す.
- **void getScale (GLfloat *factor) const**
 トラックボール処理の換算係数を取り出す.
- **const GgQuaternion & getQuaternion () const**
 現在の回転の四元数を取り出す.
- **const GgMatrix & getMatrix () const**
 現在の回転の変換行列を取り出す.
- **const GLfloat * get () const**
 現在の回転の変換行列を取り出す.

8.15.1 詳解

簡易トラックボール処理.

gg.h の 3818 行目に定義があります。

8.15.2 構築子と解体子

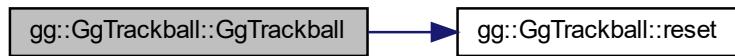
8.15.2.1 GgTrackball()

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

コンストラクタ.

gg.h の 3830 行目に定義がります。

呼び出し関係図:



8.15.2.2 ~GgTrackball()

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

デストラクタ.

gg.h の 3836 行目に定義がります。

8.15.3 関数詳解

8.15.3.1 begin()

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

トラックボール処理を開始する.

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.

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.

gg.cpp の 4969 行目に定義がります。

8.15.3.2 end()

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

トラックボール処理を停止する。

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。

引数

x	現在のマウスの x 座標.
y	現在のマウスの y 座標.

gg.cpp の 5033 行目に定義がります。

8.15.3.3 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 3940 行目に定義がります。

呼び出し関係図:



8.15.3.4 `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 3933 行目に定義があります。

8.15.3.5 `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 3926 行目に定義があります。

8.15.3.6 `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 3904 行目に定義があります。

8.15.3.7 `getScale() [2/3]`

```
void gg::GgTrackball::getScale (
    GLfloat * factor ) const [inline]
```

トラックボール処理の換算係数を取り出す。

引数

<i>factor</i>	トラックボールの換算係数を格納する 2 要素の配列.
---------------	----------------------------

gg.h の 3918 行目に定義がります。

8.15.3.8 `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 3911 行目に定義がります。

8.15.3.9 `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 3882 行目に定義がります。

8.15.3.10 `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 3896 行目に定義があります。

8.15.3.11 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 3889 行目に定義があります。

8.15.3.12 motion()

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

回転の変換行列を計算する。

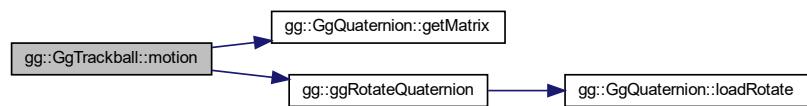
マウスのドラッグ中に呼び出す。

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.

gg.cpp の 4985 行目に定義があります。

呼び出し関係図:



8.15.3.13 region() [1/2]

```
void gg::GgTrackball::region (
    GLfloat w,
    GLfloat h )
```

トラックボール処理するマウスの移動範囲を指定する。

ウィンドウのリサイズ時に呼び出す。

引数

<i>w</i>	領域の横幅.
<i>h</i>	領域の高さ.

gg.cpp の 4956 行目に定義がります。

被呼び出し関係図:

**8.15.3.14 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。

ウィンドウのリサイズ時に呼び出す。

引数

<i>w</i>	領域の横幅.
<i>h</i>	領域の高さ.

gg.h の 3850 行目に定義がります。

呼び出し関係図:



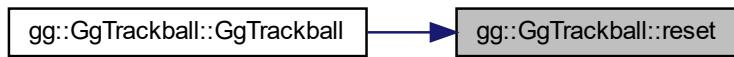
8.15.3.15 reset()

`void gg::GgTrackball::reset ()`

トラックボールをリセットする

gg.cpp の 4938 行目に定義があります。

被呼び出し関係図:



8.15.3.16 rotate()

`void gg::GgTrackball::rotate (`
`const GgQuaternion & q)`

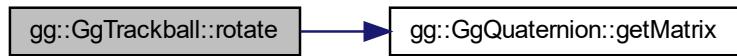
トラックボールの回転角を修正する。

引数

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

gg.cpp の 5012 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

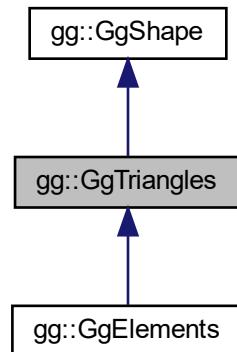
- [gg.h](#)
- [gg.cpp](#)

8.16 gg::GgTriangles クラス

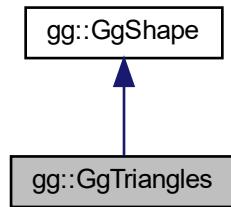
三角形で表した形状データ (Arrays 形式).

```
#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)`
コンストラクタ.
- `GgTriangles (const GgVertex *vert, GLsizei count, GLenum mode=GL_TRIANGLES, GLenum usage=GL_STATIC_DRAW)`
コンストラクタ.
- `virtual ~GgTriangles ()`
デストラクタ.
- `const GLsizei & getCount () const`
データの数を取り出す.
- `const GLuint & getBuffer () const`
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.
- `void send (const GgVertex *vert, GLint first=0, GLsizei count=0) const`
既存のバッファオブジェクトに頂点属性を転送する.
- `void 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.
- `virtual void draw (GLint first=0, GLsizei count=0) const`
三角形の描画.

8.16.1 詳解

三角形で表した形状データ (Arrays 形式).

gg.h の 4861 行目に定義があります。

8.16.2 構築子と解体子

8.16.2.1 GgTriangles() [1/2]

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

コンストラクタ.

引数

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

gg.h の 4871 行目に定義があります。

8.16.2.2 GgTriangles() [2/2]

```
gg::GgTriangles::GgTriangles (
    const GgVertex * vert,
    GLsizei count,
    GLenum mode = GL_TRIANGLES,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

コンストラクタ.

引数

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

gg.h の 4881 行目に定義があります。

8.16.2.3 ~GgTriangles()

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

デストラクタ.

gg.h の 4893 行目に定義があります。

8.16.3 関数詳解

8.16.3.1 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 5092 行目に定義があります。

呼び出し関係図:



8.16.3.2 `getBuffer()`

`const GLuint& gg::GgTriangles::getBuffer () const [inline]`

頂点属性を格納した頂点バッファオブジェクト名を取り出す。

戻り値

この図形の頂点属性を格納した頂点バッファオブジェクト名。

`gg.h` の 4906 行目に定義があります。

8.16.3.3 `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 4899 行目に定義があります。

8.16.3.4 `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 5073 行目に定義があります。

8.16.3.5 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 4915 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

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

8.17 gg::GgUniformBuffer< T > クラステンプレート

ユニフォームバッファオブジェクト.

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

公開メンバ関数

- [GgUniformBuffer \(\)](#)
コンストラクタ.
- [GgUniformBuffer \(const T *data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
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.
- [GgUniformBuffer \(const T &data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
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.

- virtual ~GgUniformBuffer ()
デストラクタ.
- const GLuint & **getTarget** () const
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.
- GLsizeiptr **getStride** () const
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.
- const GLsizei & **getCount** () const
データの数を取り出す.
- const GLuint & **getBuffer** () const
ユニフォームバッファオブジェクト名を取り出す.
- void **bind** () const
ユニフォームバッファオブジェクトを結合する.
- void **unbind** () const
ユニフォームバッファオブジェクトを解放する.
- void * **map** () const
ユニフォームバッファオブジェクトをマップする.
- void * **map** (GLint first, GLsizei count) const
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.
- void **unmap** () const
バッファオブジェクトをアンマップする.
- void **load** (const T *data, GLsizei count, GLenum usage=GL_STATIC_DRAW)
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.
- void **load** (const T &data, GLsizei count, GLenum usage=GL_STATIC_DRAW)
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.
- void **send** (const GLvoid *data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
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.
- void **fill** (const GLvoid *data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
ユニフォームバッファオブジェクトの全ブロックのメンバーを同じデータを格納する.
- void **read** (GLvoid *data, GLint offset=0, GLsizei size=sizeof(T), GLint first=0, GLsizei count=0) const
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.
- void **copy** (GLuint src_buffer, GLint src_first=0, GLint dst_first=0, GLsizei count=0) const
別のバッファオブジェクトからデータを複写する.

8.17.1 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.

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 4406 行目に定義があります。

8.17.2 構築子と解体子

8.17.2.1 GgUniformBuffer() [1/3]

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

コンストラクタ.

gg.h の 4409 行目に定義があります。

8.17.2.2 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 4409 行目に定義があります。

8.17.2.3 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 4409 行目に定義があります。

8.17.2.4 ~GgUniformBuffer()

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

デストラクタ.

gg.h の 4409 行目に定義があります。

8.17.3 関数詳解

8.17.3.1 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 4470 行目に定義があります。

8.17.3.2 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 4642 行目に定義があります。

8.17.3.3 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 4576 行目に定義があります。

8.17.3.4 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 4464 行目に定義があります。

8.17.3.5 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 4457 行目に定義があります。

8.17.3.6 `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 4450 行目に定義があります。

8.17.3.7 `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 4443 行目に定義があります。

8.17.3.8 `load()` [1/2]

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

ユニフォームバッファオブジェクトを確保して全てのブロックに同じデータを格納する.

引数

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

`gg.h` の 4523 行目に定義があります。

8.17.3.9 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 4507 行目に定義があります。

8.17.3.10 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 4483 行目に定義があります。

8.17.3.11 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 4492 行目に定義があります。

8.17.3.12 `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

<code>data</code>	抽出先の領域の先頭のポインタ.
<code>offset</code>	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.
<code>size</code>	抽出するデータの一個あたりのバイト数.
<code>first</code>	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.
<code>count</code>	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 4608 行目に定義があります。

8.17.3.13 `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

<code>data</code>	データが格納されている領域の先頭のポインタ.
<code>offset</code>	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.
<code>size</code>	格納するデータの一個あたりのバイト数.
<code>first</code>	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.
<code>count</code>	格納するデータの数.

gg.h の 4541 行目に定義があります。

8.17.3.14 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 4476 行目に定義があります。

8.17.3.15 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 4498 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.18 gg::GgVertex 構造体

三角形の頂点データ。

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

公開メンバ関数

- [GgVertex \(\)](#)
法線。
- [GgVertex \(const GgVector &pos, const GgVector &norm\)](#)
コンストラクタ。
- [GgVertex \(GLfloat px, GLfloat py, GLfloat pz, GLfloat nx, GLfloat ny, GLfloat nz\)](#)
コンストラクタ。
- [GgVertex \(const GLfloat *pos, const GLfloat *norm\)](#)
コンストラクタ。

公開変数類

- `GgVector position`
- `GgVector normal`
位置.

8.18.1 詳解

三角形の頂点データ.

`gg.h` の 4814 行目に定義がります。

8.18.2 構築子と解体子

8.18.2.1 `GgVertex()` [1/4]

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

法線.

コンストラクタ.

`gg.h` の 4820 行目に定義がります。

8.18.2.2 `GgVertex()` [2/4]

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

コンストラクタ.

引数

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

`gg.h` の 4827 行目に定義がります。

8.18.2.3 `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 4840 行目に定義があります。

8.18.2.4 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 4852 行目に定義があります。

8.18.3 メンバ詳解

8.18.3.1 normal

`GgVector gg::GgVertex::normal`

位置.

gg.h の 4817 行目に定義があります。

8.18.3.2 position

`GgVector gg::GgVertex::position`

gg.h の 4816 行目に定義がります。

この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

8.19 gg::GgSimpleShader::Light 構造体

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.

`#include <gg.h>`

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.

8.19.1 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 5547 行目に定義がります。

8.19.2 メンバ詳解

8.19.2.1 ambient

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

光源強度の環境光成分.

gg.h の 5549 行目に定義がります。

8.19.2.2 diffuse

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

光源強度の拡散反射光成分.

gg.h の 5550 行目に定義があります。

8.19.2.3 position

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

光源の位置.

gg.h の 5552 行目に定義があります。

8.19.2.4 specular

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

光源強度の鏡面反射光成分.

gg.h の 5551 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:

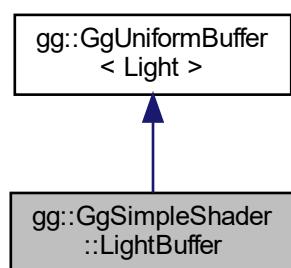
- `gg.h`

8.20 gg::GgSimpleShader::LightBuffer クラス

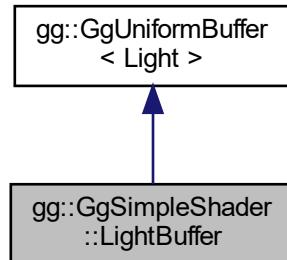
三角形に単純な陰影付けを行うシェーダが参照する光源データのユニフォームバッファオブジェクト.

`#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)
デフォルトコンストラクタ.
- **LightBuffer** (const [Light](#) &light, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)
同じデータで埋めるコンストラクタ.
- **virtual ~LightBuffer ()**
デストラクタ.
- **void 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.
- **void loadAmbient** (const GLfloat *ambient, GLint first=0, GLsizei count=1) const
光源の強度の環境光成分を設定する.
- **void 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.
- **void 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.
- **void 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.
- **void loadDiffuse** (const GLfloat *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.
- **void 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.
- **void 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.
- **void loadSpecular** (const GLfloat *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.
- **void 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.
- **void 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.
- **void loadPosition** (const [GgVector](#) &position, GLint first=0, GLsizei count=1) const
光源の位置を設定する.
- **void loadPosition** (const GLfloat *position, GLint first=0, GLsizei count=1) const

光源の位置を設定する.

- void **loadPosition** (const **GgVector** *position, GLint first=0, GLsizei count=1) const
光源の位置を設定する.
- void **load** (const **Light** *light, GLint first=0, GLsizei count=1) const
光源の色と位置を設定する.
- void **load** (const **Light** &light, GLint first=0, GLsizei count=1) const
光源の色と位置を設定する.
- void **select** (GLint i=0) const
光源を選択する.

8.20.1 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 5558 行目に定義があります。

8.20.2 構築子と解体子

8.20.2.1 LightBuffer() [1/2]

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

デフォルトコンストラクタ.

引数

<i>light</i>	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 5567 行目に定義があります。

8.20.2.2 LightBuffer() [2/2]

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

同じデータで埋めるコンストラクタ.

引数

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

gg.h の 5580 行目に定義があります。

8.20.2.3 ~LightBuffer()

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

デストラクタ.

gg.h の 5590 行目に定義があります。

8.20.3 関数詳解

8.20.3.1 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 5734 行目に定義があります。

8.20.3.2 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 5725 行目に定義があります。

8.20.3.3 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 5428 行目に定義があります。

8.20.3.4 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 5610 行目に定義がります。

8.20.3.5 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 5402 行目に定義がります。

8.20.3.6 loadColor()

```
void gg::GgSimpleShader::LightBuffer::loadColor (
    const Light & color,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の色を設定するが位置は変更しない。

引数

<i>color</i>	光源の特性の GgSimpleShader::Light 構造体.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5555 行目に定義がります。

8.20.3.7 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>specular</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 5480 行目に定義があります。

8.20.3.8 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 5644 行目に定義があります。

8.20.3.9 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 5454 行目に定義があります。

8.20.3.10 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 5609 行目に定義があります。

8.20.3.11 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 5716 行目に定義がります。

8.20.3.12 loadPosition() [3/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GLfloat * position,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の位置を設定する。

引数

<i>position</i>	光源の位置の同次座標を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 5706 行目に定義がります。

8.20.3.13 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 5583 行目に定義がります。

8.20.3.14 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 <code>GgVector</code> 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

`gg.cpp` の 5532 行目に定義があります。

8.20.3.15 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 5672 行目に定義があります。

8.20.3.16 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 5506 行目に定義があります。

8.20.3.17 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 5741 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](#)

8.21 gg::GgSimpleShader::Material 構造体

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.

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

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.

8.21.1 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 5752 行目に定義があります。

8.21.2 メンバ詳解

8.21.2.1 ambient

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

環境光に対する反射係数。

gg.h の 5754 行目に定義があります。

8.21.2.2 diffuse

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

拡散反射係数。

gg.h の 5755 行目に定義があります。

8.21.2.3 shininess

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

輝き係数。

gg.h の 5757 行目に定義があります。

8.21.2.4 specular

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

鏡面反射係数。

gg.h の 5756 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:

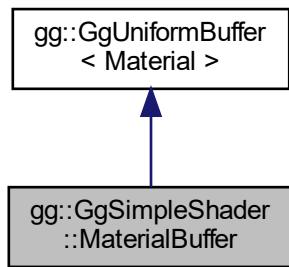
- [gg.h](#)

8.22 gg::GgSimpleShader::MaterialBuffer クラス

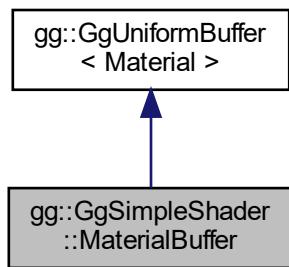
三角形に単純な陰影付けを行うシェーダが参照する材質データのユニフォームバッファオブジェクト。

```
#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)
デフォルトコンストラクタ.
- **MaterialBuffer** (const [Material](#) &material, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)
同じデータで埋めるコンストラクタ.
- **virtual ~MaterialBuffer ()**
デストラクタ
- **void 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.

- void **loadAmbient** (const GLfloat *ambient, GLint first=0, GLsizei count=1) const
環境光に対する反射係数を設定する.
- void **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.
- void **loadDiffuse** (const GLfloat *diffuse, GLint first=0, GLsizei count=1) const
拡散反射係数を設定する.
- void **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.
- void **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.
- void **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.
- void **loadSpecular** (const GLfloat *specular, GLint first=0, GLsizei count=1) const
鏡面反射係数を設定する.
- void **loadShininess** (GLfloat shininess, GLint first=0, GLsizei count=1) const
輝き係数を設定する.
- void **loadShininess** (const GLfloat *shininess, GLint first=0, GLsizei count=1) const
輝き係数を設定する.
- void **load** (const Material *material, GLint first=0, GLsizei count=1) const
材質を設定する.
- void **load** (const Material &material, GLint first=0, GLsizei count=1) const
材質を設定する.
- void **select** (GLint i=0) const
材質を選択する.

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 5763 行目に定義があります。

8.22.2 構築子と解体子

8.22.2.1 MaterialBuffer() [1/2]

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

デフォルトコンストラクタ.

引数

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

gg.h の 5772 行目に定義があります。

8.22.2.2 MaterialBuffer() [2/2]

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

同じデータで埋めるコンストラクタ。

引数

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

gg.h の 5785 行目に定義があります。

8.22.2.3 ~MaterialBuffer()

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

デストラクタ

gg.h の 5795 行目に定義があります。

8.22.3 関数詳解

8.22.3.1 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 5908 行目に定義がります。

8.22.3.2 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 5899 行目に定義がります。

8.22.3.3 loadAmbient() [1/2]

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

環境光に対する反射係数を設定する。

引数

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

gg.h の 5815 行目に定義がります。

8.22.3.4 loadAmbient() [2/2]

```
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 5635 行目に定義があります。

8.22.3.5 loadAmbientAndDiffuse() [1/2]

```
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 5719 行目に定義があります。

8.22.3.6 loadAmbientAndDiffuse() [2/2]

```
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 5693 行目に定義があります。

8.22.3.7 loadDiffuse() [1/2]

```
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 <code>GLfloat</code> 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 5837 行目に定義があります。

8.22.3.8 loadDiffuse() [2/2]

```
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.

引数

<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 5664 行目に定義がります。

8.22.3.9 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 5809 行目に定義がります。

8.22.3.10 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 5788 行目に定義がります。

8.22.3.11 loadSpecular() [1/2]

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

鏡面反射係数を設定する。

引数

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

`gg.h` の 5877 行目に定義があります。

8.22.3.12 loadSpecular() [2/2]

```
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 5762 行目に定義があります。

8.22.3.13 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 5915 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](#)

8.23 Window クラス

ウィンドウ関連の処理.

```
#include <Window.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)
コンストラクタ.
- **Window** (const **Window** &w)=delete
コピー構造子は使用禁止.
- **Window** & **operator=** (const **Window** &w)=delete
代入演算子は使用禁止.
- **virtual ~Window** ()
デストラクタ.
- **GLFWwindow *** **get** () const
ウィンドウの識別子のポインタを取得する.
- **void setClose** (int flag=GLFW_TRUE) const
ウィンドウのクローズフラグを設定する.
- **bool shouldClose** () const
ウィンドウを閉じるべきかどうか調べる.
- **operator bool** ()
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.
- **void swapBuffers** ()
カラーバッファを入れ替える.
- **GLsizei getWidth** () const
ウィンドウの横幅を得る.
- **GLsizei getHeight** () const
ウィンドウの高さを得る.
- **const GLsizei *** **getSize** () const
ウィンドウのサイズを得る.
- **void getSize** (GLsizei *size) const
ウィンドウのサイズを得る.
- **GLfloat getAspect** () const
ウィンドウのアスペクト比を得る.

- void `restoreViewport ()`
ビューポートをウィンドウ全体に設定する.
- bool `getKey (int key)`
キーが押されているかどうかを判定する.
- void `selectInterface (int no)`
インターフェースを選択する
- void `setVelocity (GLfloat vx, GLfloat vy, GLfloat vz=0.1f)`
マウスの移動速度を設定する
- GLfloat `getArrow (int direction=0, int mods=0) const`
矢印キーの現在の値を得る.
- GLfloat `getArrowX (int mods=0) const`
矢印キーの現在の X 値を得る.
- GLfloat `getArrowY (int mods=0) const`
矢印キーの現在の Y 値を得る.
- void `getArrow (GLfloat *arrow, int mods=0) const`
矢印キーの現在の値を得る.
- GLfloat `getShiftArrowX () const`
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.
- GLfloat `getShiftArrowY () const`
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.
- void `getShiftArrow (GLfloat *shift_arrow) const`
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.
- GLfloat `getControlArrowX () const`
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.
- GLfloat `getControlArrowY () const`
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.
- void `getControlArrow (GLfloat *control_arrow) const`
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.
- GLfloat `getAltArrowX () const`
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.
- GLfloat `getAltArrowY () const`
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.
- void `getAltArrow (GLfloat *alt_arrow) const`
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.
- const GLfloat * `getMouse () const`
マウスカーソルの現在位置を得る.
- void `getMouse (GLfloat *position) const`
マウスカーソルの現在位置を得る.
- const GLfloat `getMouse (int direction) const`
マウスカーソルの現在位置を得る.
- GLfloat `getMouseX () const`
マウスカーソルの現在位置の X 座標を得る.
- GLfloat `getMouseY () const`
マウスカーソルの現在位置の Y 座標を得る.
- const GLfloat * `getWheel () const`
マウスホイールの回転量を得る.
- void `getWheel (GLfloat *rotation) const`
マウスホイールの回転量を得る.
- GLfloat `getWheel (int direction) const`
マウスホイールの回転量を得る.
- const GLfloat `getWheelX () const`

- マウスホイールの X 方向の回転量を得る.
- const GLfloat `getWheelX () const`
マウスホイールの Y 方向の回転量を得る.
- const GLfloat * `getTranslation (int button=GLFW_MOUSE_BUTTON_1) const`
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.
- GgMatrix `getTranslationMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
マウスによって視点の平行移動の変換行列を得る.
- GgMatrix `getScrollMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
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.
- GgQuaternion `getRotation (int button=GLFW_MOUSE_BUTTON_1) const`
トラックボールの回転変換行列を得る.
- GgMatrix `getRotationMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
トラックボールの回転変換行列を得る.
- void `resetRotation ()`
トラックボール処理をリセットする
- void `resetTranslation ()`
現在位置と平行移動量をリセットする
- void `reset ()`
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
- void * `getUserPointer () const`
ユーザー pointer を取り出す.
- void `setUserPointer (void *pointer)`
任意のユーザ pointer を保存する.
- void `setResizeFunc (void(*func)(const Window *window, int width, int height))`
ユーザ定義の `resize` 関数を設定する.
- void `setKeyboardFunc (void(*func)(const Window *window, int key, int scancode, int action, int mods))`
ユーザ定義の `keyboard` 関数を設定する.
- void `setMouseFunc (void(*func)(const Window *window, int button, int action, int mods))`
ユーザ定義の `mouse` 関数を設定する.
- void `setResizeFunc (void(*func)(const Window *window, double x, double y))`
ユーザ定義の `wheel` 関数を設定する.

静的公開メンバ関数

- static void `init (int major=0, int minor=1)`
ゲームグラフィックス特論の宿題用補助プログラムの初期化. 最初に一度だけ実行する.

8.23.1 詳解

ウィンドウ関連の処理.

GLFW を使って OpenGL のウィンドウを操作するラッパークラス.

Window.h の 88 行目に定義があります。

8.23.2 構築子と解体子

8.23.2.1 Window() [1/2]

```
Window::Window (
    const std::string & title = "GLFW Window",
    int width = 640,
    int height = 480,
    int fullscreen = 0,
    GLFWwindow * share = nullptr ) [inline]
```

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.
<i>height</i>	開くウィンドウの高さ.
<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.

Window.h の 445 行目に定義があります。

呼び出し関係図:



8.23.2.2 Window() [2/2]

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

コピー コンストラクタは使用禁止.

8.23.2.3 ~Window()

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

デストラクタ.

Window.h の 528 行目に定義があります。

8.23.3 関数詳解

8.23.3.1 get()

```
GLFWwindow* 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。

Window.h の 545 行目に定義があります。

8.23.3.2 getAltArrowX()

```
GLfloat 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。

Window.h の 784 行目に定義があります。

8.23.3.3 getAltArrowY()

```
GLfloat 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。

Window.h の 791 行目に定義があります。

8.23.3.4 getAltIArrow()

```
void Window::getAltIArrow (
    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。
------------------	---

Window.h の 798 行目に定義があります。

8.23.3.5 getArrow() [1/2]

```
void 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)。

Window.h の 732 行目に定義があります。

8.23.3.6 getArrow() [2/2]

```
GLfloat 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。

Window.h の 707 行目に定義があります。

8.23.3.7 getArrowX()

```
GLfloat 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。

Window.h の 716 行目に定義があります。

8.23.3.8 getArrowY()

```
GLfloat 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。

Window.h の 724 行目に定義があります。

8.23.3.9 getAspect()

```
GLfloat Window::getAspect ( ) const [inline]
```

ウィンドウのアスペクト比を得る。

戻り値

ウィンドウの縦横比。

Window.h の 664 行目に定義があります。

8.23.3.10 getControlArrow()

```
void 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

<i>control_arrow</i>	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。
----------------------	--

Window.h の 776 行目に定義があります。

8.23.3.11 getControlArrowX()

```
GLfloat 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。

Window.h の 762 行目に定義があります。

8.23.3.12 getControlArrowY()

```
GLfloat 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。

Window.h の 769 行目に定義があります。

8.23.3.13 getHeight()

```
GLsizei 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.

Window.h の 642 行目に定義があります。

8.23.3.14 getKey()

```
bool Window::getKey (
    int key ) [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.

Window.h の 677 行目に定義があります。

8.23.3.15 getMouse() [1/3]

```
const GLfloat* 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.

Window.h の 806 行目に定義があります。

8.23.3.16 getMouse() [2/3]

```
void 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.
-----------------	---------------------------------------

Window.h の 814 行目に定義があります。

8.23.3.17 getMouse() [3/3]

```
const GLfloat 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.

Window.h の 824 行目に定義があります。

8.23.3.18 getMouseX()

```
GLfloat 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.

Window.h の 832 行目に定義があります。

8.23.3.19 getMouseY()

```
GLfloat 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.

Window.h の 840 行目に定義があります。

8.23.3.20 getRotation()

```
GgQuaternion 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。

Window.h の 933 行目に定義があります。

8.23.3.21 getRotationMatrix()

```
GgMatrix 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

<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 GgMatrix 型の変換行列。

Window.h の 943 行目に定義があります。

8.23.3.22 getScrollMatrix()

```
GgMatrix Window::getScrollMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによってオブジェクトの平行移動の変換行列を得る。

引数

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

戻り値

クリッピング座標系で平行移動を行う GgMatrix 型の変換行列.

Window.h の 916 行目に定義があります。

8.23.3.23 getShiftArrow()

```
void 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.
--------------------	---

Window.h の 754 行目に定義があります。

8.23.3.24 getShiftArrowX()

```
GLfloat 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.

Window.h の 740 行目に定義があります。

8.23.3.25 getShiftArrowY()

```
GLfloat 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.

Window.h の 747 行目に定義があります。

8.23.3.26 getSize() [1/2]

```
const GLsizei* 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。

Window.h の 649 行目に定義があります。

8.23.3.27 getSize() [2/2]

```
void 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 GLsizei 型の 2 要素の配列。
-------------	-------------------------------------

Window.h の 656 行目に定義があります。

8.23.3.28 getTranslation()

```
const GLfloat* 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 (GLFW_MOUSE_BUTTON_[1,2])。
---------------	---

戻り値

平行移動量を格納した GLfloat[3] の配列のポインタ。

Window.h の 889 行目に定義があります。

8.23.3.29 getTranslationMatrix()

```
GgMatrix Window::getTranslationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによって視点の平行移動の変換行列を得る。

引数

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

戻り値

視点座標系で平行移動を行う GgMatrix 型の変換行列。

Window.h の 899 行目に定義があります。

8.23.3.30 getUserPointer()

```
void* 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。

Window.h の 991 行目に定義があります。

8.23.3.31 getWheel() [1/3]

```
const GLfloat* 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。

Window.h の 848 行目に定義があります。

8.23.3.32 getWheel() [2/3]

```
void 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.
-----------------	--------------------------------------

Window.h の 856 行目に定義があります。

8.23.3.33 getWheel() [3/3]

```
GLfloat 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。

Window.h の 866 行目に定義があります。

8.23.3.34 getWheelX()

```
const GLfloat Window::getWheelX ( ) const [inline]
```

マウスホイールの X 方向の回転量を得る。

Window.h の 873 行目に定義があります。

8.23.3.35 getWheelY()

```
const GLfloat Window::getWheelY ( ) const [inline]
```

マウスホイールの Y 方向の回転量を得る。

Window.h の 880 行目に定義があります。

8.23.3.36 getWidth()

```
GLsizei 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.

Window.h の 635 行目に定義があります。

8.23.3.37 init()

```
static void Window::init (
    int major = 0,
    int minor = 1 ) [inline], [static]
```

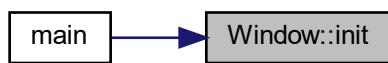
ゲームグラフィックス特論の宿題用補助プログラムの初期化, 最初に一度だけ実行する.

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.

Window.h の 387 行目に定義があります。

被呼び出し関係図:



8.23.3.38 operator bool()

```
Window::operator bool ( ) [inline]
```

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.

Window.h の 567 行目に定義があります。

8.23.3.39 operator=()

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

代入演算子は使用禁止.

8.23.3.40 reset()

```
void 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

Window.h の 980 行目に定義があります。

8.23.3.41 resetRotation()

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

トラックボール処理をリセットする

Window.h の 951 行目に定義があります。

8.23.3.42 resetTranslation()

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

現在位置と平行移動量をリセットする

Window.h の 961 行目に定義があります。

8.23.3.43 restoreViewport()

```
void Window::restoreViewport () [inline]
```

ビューポートをウィンドウ全体に設定する.

Window.h の 670 行目に定義があります。

8.23.3.44 selectInterface()

```
void 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
-----------	------------

Window.h の 689 行目に定義があります。

8.23.3.45 setClose()

```
void 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。
-------------	--

Window.h の 552 行目に定義があります。

8.23.3.46 setKeyboardFunc()

```
void 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。
-------------	---------------------------------------

Window.h の 1012 行目に定義があります。

8.23.3.47 setMouseFunc()

```
void 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 mouse 関数, マウスボタンの操作時に呼び出される.
-------------	-------------------------------------

Window.h の 1019 行目に定義があります。

8.23.3.48 setResizeFunc() [1/2]

```
void Window::setResizeFunc (
    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.
-------------	--------------------------------------

Window.h の 1026 行目に定義があります。

8.23.3.49 setResizeFunc() [2/2]

```
void 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 resize 関数, ウィンドウのサイズ変更時に呼び出される.
-------------	--

Window.h の 1005 行目に定義があります。

8.23.3.50 setUserPointer()

```
void 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.
----------------	--------------

Window.h の 998 行目に定義があります。

8.23.3.51 setVelocity()

```
void 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.

Window.h の 698 行目に定義があります。

8.23.3.52 shouldClose()

```
bool 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.

Window.h の 559 行目に定義があります。

8.23.3.53 swapBuffers()

```
void Window::swapBuffers () [inline]
```

カラーバッファを入れ替える。

Window.h の 619 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:

- [Window.h](#)

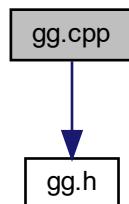
Chapter 9

ファイル詳解

9.1 gg.cpp ファイル

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。

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



9.1.1 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 GLFW3 版の定義。

著者

Kohe Tokoi

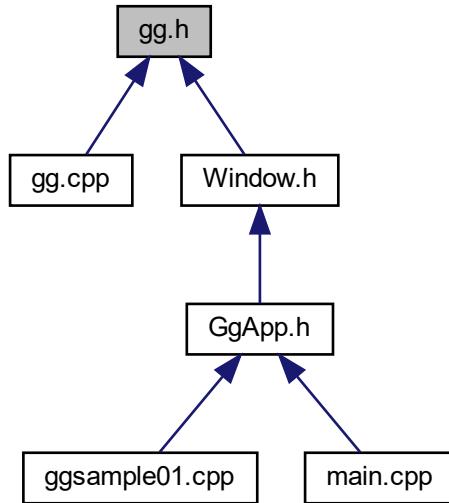
日付

March 31, 2021

9.2 gg.h ファイル

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.

被依存関係図:



クラス

- class `gg::GgMatrix`
変換行列.
- class `gg::GgQuaternion`
四元数.
- class `gg::GgTrackball`
簡易トラックボール処理.
- class `gg::GgTexture`
テクスチャ.
- class `gg::GgColorTexture`
カラーマップ.
- class `gg::GgNormalTexture`
法線マップ.
- class `gg::GgBuffer< T >`
バッファオブジェクト.
- class `gg::GgUniformBuffer< T >`
ユニフォームバッファオブジェクト.
- class `gg::GgShape`
形状データの基底クラス.
- class `gg::GgPoints`
点.
- struct `gg::GgVertex`

- 三角形の頂点データ.
- class `gg::GgTriangles`
三角形で表した形状データ (*Arrays* 形式).
- class `gg::GgElements`
三角形で表した形状データ (*Elements* 形式).
- class `gg::GgShader`
シェーダの基底クラス.
- class `gg::GgPointShader`
点のシェーダ.
- class `gg::GgSimpleShader`
三角形に単純な陰影付けを行うシェーダ.
- struct `gg::GgSimpleShader::Light`
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.
- class `gg::GgSimpleShader::LightBuffer`
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.
- struct `gg::GgSimpleShader::Material`
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.
- class `gg::GgSimpleShader::MaterialBuffer`
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.
- class `gg::GgSimpleObj`
Wavefront OBJ 形式のファイル (*Arrays* 形式).

名前空間

- `gg`
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

型定義

- using `gg::GgVector` = std::array< GLfloat, 4 >
4 要素の单精度実数の配列.

列挙型

- enum `gg::BindingPoints` { `gg::LightBindingPoint` = 0 , `gg::MaterialBindingPoint` }
光源と材質の *uniform buffer object* の結合ポイント.

関数

- void `gg::ggInit()`
ゲームグラフィックス特論の都合にもとづく初期化を行う.
- void `gg::ggError` (const std::string &name="", unsigned int line=0)
OpenGL のエラーをチェックする.
- void `gg::ggFBOError` (const std::string &name="", unsigned int line=0)
FBO のエラーをチェックする.
- bool `gg::ggSaveTga` (const std::string &name, const void *buffer, unsigned int width, unsigned int height, unsigned int depth)
配列の内容を *TGA* ファイルに保存する.

- **bool gg::ggSaveColor (const std::string &name)**
カラーバッファの内容を *TGA* ファイルに保存する.
- **bool gg::ggSaveDepth (const std::string &name)**
デプスバッファの内容を *TGA* ファイルに保存する.
- **bool gg::ggReadImage (const std::string &name, std::vector< GLubyte > &image, GLsizei *pWidth, GLsizei *pHeight, GLenum *pFormat)**
TGA ファイル (*8/16/24/32bit*) をメモリに読み込む.
- **GLuint gg::ggLoadTexture (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_BGR, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE)**
テクスチャメモリを確保して画像データをテクスチャとして読み込む.
- **GLuint gg::ggLoadImage (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE)**
テクスチャメモリを確保して *TGA* 画像ファイルを読み込む.
- **void gg::ggCreateNormalMap (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)**
グレースケール画像 (*8bit*) から法線マップのデータを作成する.
- **GLuint gg::ggLoadHeight (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)**
テクスチャメモリを確保して *TGA* 画像ファイルを読み込み法線マップを作成する.
- **GLuint gg::ggCreateShader (const std::string &vsrc, const std::string &fsrc="", const std::string &gsrc="", GLint nvarying=0, const char *const *varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string >ext="geometry shader")**
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.
- **GLuint gg::ggLoadShader (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)**
シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.
- **GLuint gg::ggCreateComputeShader (const std::string &csrc, const std::string &ctext="compute shader")**
コンピュートシェーダのソースプログラムの文字列を読み込んでプログラムオブジェクトを作成する.
- **GLuint gg::ggLoadComputeShader (const std::string &comp)**
コンピュートシェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.
- **GLfloat gg::ggLength3 (const GLfloat *a)**
3要素の長さ.
- **void gg::ggNormalize3 (GLfloat *a)**
3要素の正規化.
- **GLfloat gg::ggDot3 (const GLfloat *a, const GLfloat *b)**
3要素の内積.
- **void gg::ggCross (GLfloat *c, const GLfloat *a, const GLfloat *b)**
3要素の外積.
- **GLfloat gg::ggLength4 (const GLfloat *a)**
4要素の長さ.
- **GLfloat gg::ggLength4 (const GgVector &a)**
GgVector 型の長さ.
- **void gg::ggNormalize4 (GLfloat *a)**
4要素の正規化.
- **void gg::ggNormalize4 (GgVector &a)**
GgVector 型の正規化.
- **GLfloat gg::ggDot4 (const GLfloat *a, const GLfloat *b)**
4要素の内積
- **GLfloat gg::ggDot4 (const GgVector &a, const GgVector &b)**
GgVector 型の内積
- **GgMatrix gg::ggIdentity ()**
単位行列を返す.

- GgMatrix [gg::ggTranslate](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
平行移動の変換行列を返す.
- GgMatrix [gg::ggTranslate](#) (const GLfloat *t)
平行移動の変換行列を返す.
- GgMatrix [gg::ggTranslate](#) (const GgVector &t)
平行移動の変換行列を返す.
- GgMatrix [gg::ggScale](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)
拡大縮小の変換行列を返す.
- GgMatrix [gg::ggScale](#) (const GLfloat *s)
拡大縮小の変換行列を返す.
- GgMatrix [gg::ggScale](#) (const GgVector &s)
拡大縮小の変換行列を返す.
- GgMatrix [gg::ggRotateX](#) (GLfloat a)
 x 軸中心の回転の変換行列を返す.
- GgMatrix [gg::ggRotateY](#) (GLfloat a)
 y 軸中心の回転の変換行列を返す.
- GgMatrix [gg::ggRotateZ](#) (GLfloat a)
 z 軸中心の回転の変換行列を返す.
- GgMatrix [gg::ggRotate](#) (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乗じた結果を返す.
- GgMatrix [gg::ggRotate](#) (const GLfloat *r, GLfloat a)
 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.
- GgMatrix [gg::ggRotate](#) (const GgVector &r, GLfloat a)
 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.
- GgMatrix [gg::ggRotate](#) (const GLfloat *r)
 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.
- GgMatrix [gg::ggRotate](#) (const GgVector &r)
 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.
- GgMatrix [gg::ggLookat](#) (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)
ビュー変換行列を返す.
- GgMatrix [gg::ggLookat](#) (const GLfloat *e, const GLfloat *t, const GLfloat *u)
ビュー変換行列を返す.
- GgMatrix [gg::ggLookat](#) (const GgVector &e, const GgVector &t, const GgVector &u)
ビュー変換行列を返す.
- GgMatrix [gg::ggOrthogonal](#) (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
直交投影変換行列を返す.
- GgMatrix [gg::ggFrustum](#) (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)
透視透視投影変換行列を返す.
- GgMatrix [gg::ggPerspective](#) (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)
画角を指定して透視投影変換行列を返す.
- GgMatrix [gg::ggTranspose](#) (const GgMatrix &m)
転置行列を返す.
- GgMatrix [gg::ggInvert](#) (const GgMatrix &m)
逆行列を返す.
- GgMatrix [gg::ggNormal](#) (const GgMatrix &m)
法線変換行列を返す.
- GgQuaternion [gg::ggQuaternion](#) (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
四元数を返す.
- GgQuaternion [gg::ggQuaternion](#) (const GLfloat *a)

- 四元数を返す
- GgQuaternion `gg::ggIdentityQuaternion ()`
 単位四元数を返す
- GgQuaternion `gg::ggMatrixQuaternion (const GLfloat *a)`
 回転の変換行列 m を表す四元数を返す.
- GgQuaternion `gg::ggMatrixQuaternion (const GgMatrix &m)`
 回転の変換行列 m を表す四元数を返す.
- GgMatrix `gg::ggQuaternionMatrix (const GgQuaternion &q)`
 四元数 q の回転の変換行列を返す.
- GgMatrix `gg::ggQuaternionTransposeMatrix (const GgQuaternion &q)`
 四元数 q の回転の転置した変換行列を返す.
- GgQuaternion `gg::ggRotateQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
 (x, y, z) を軸として角度 a 回転する四元数を返す.
- GgQuaternion `gg::ggRotateQuaternion (const GLfloat *v, GLfloat a)`
 $(v[0], v[1], v[2])$ を軸として角度 a 回転する四元数を返す.
- GgQuaternion `gg::ggRotateQuaternion (const GLfloat *v)`
 $(v[0], v[1], v[2])$ を軸として角度 $v[3]$ 回転する四元数を返す.
- GgQuaternion `gg::ggEulerQuaternion (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返す.
- GgQuaternion `gg::ggEulerQuaternion (const GLfloat *e)`
 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.
- GgQuaternion `gg::ggSlerp (const GLfloat *a, const GLfloat *b, GLfloat t)`
 二つの四元数の球面線形補間の結果を返す.
- GgQuaternion `gg::ggSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
 二つの四元数の球面線形補間の結果を返す.
- GgQuaternion `gg::ggSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
 二つの四元数の球面線形補間の結果を返す.
- GgQuaternion `gg::ggSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
 二つの四元数の球面線形補間の結果を返す.
- GLfloat `gg::ggNorm (const GgQuaternion &q)`
 四元数のノルムを返す.
- GgQuaternion `gg::ggNormalize (const GgQuaternion &q)`
 正規化した四元数を返す.
- GgQuaternion `gg::ggConjugate (const GgQuaternion &q)`
 共役四元数を返す.
- GgQuaternion `gg::ggInvert (const GgQuaternion &q)`
 四元数の逆元を求める.
- GgPoints * `gg::ggPointsCube (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
 点群を立方体状に生成する.
- GgPoints * `gg::ggPointsSphere (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)`
 点群を球状に生成する.
- GgTriangles * `gg::ggRectangle (GLfloat width=1.0f, GLfloat height=1.0f)`
 矩形状に 2 枚の三角形を生成する.
- GgTriangles * `gg::ggEllipse (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)`
 楕円状に三角形を生成する.
- GgTriangles * `gg::ggArraysObj (const std::string &name, bool normalize=false)`
 Wavefront OBJ ファイルを読み込む (Arrays 形式)
- GgElements * `gg::ggElementsObj (const std::string &name, bool normalize=false)`
 Wavefront OBJ ファイルを読み込む (Elements 形式).

- `GgElements * gg::ggElementsMesh (GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3]=nullptr)`
メッシュ形状を作成する (*Elements* 形式).
- `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)`
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)
- `bool gg::ggLoadSimpleObj (const std::string &name, std::vector< std::array< GLuint, 3 >> &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)`
三角形分割された *OBJ* ファイルを読み込む (*Elements* 形式).

変数

- GLint `gg::ggBufferAlignment`

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.

9.2.1 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

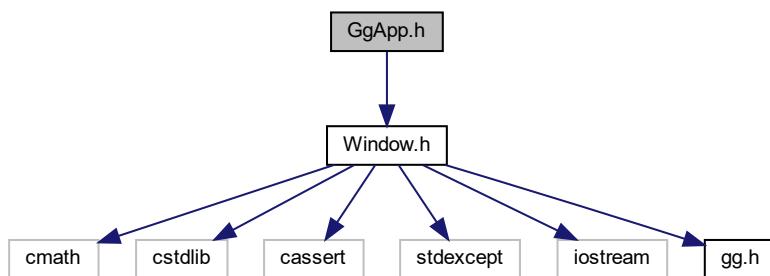
Kohe Tokoi

日付

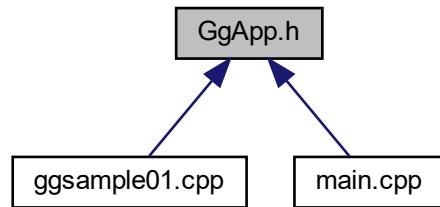
March 31, 2021

9.3 GgApp.h ファイル

```
#include "Window.h"
GgApp.h の依存先関係図:
```



被依存関係図:



クラス

- class [GgApp](#)

マクロ定義

- [#define USE_IMGUI](#)

9.3.1 マクロ定義詳解

9.3.1.1 USE_IMGUI

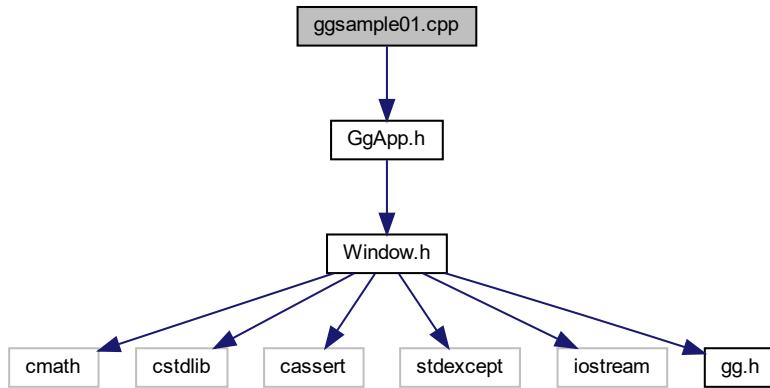
```
#define USE_IMGUI
```

GgApp.h の 10 行目に定義があります。

9.4 ggsample01.cpp ファイル

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

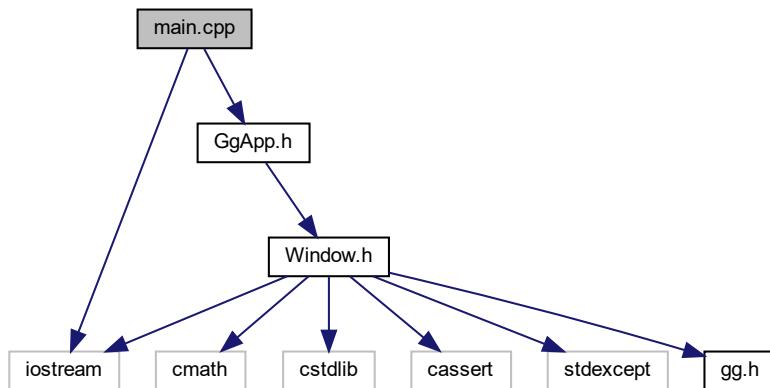
ggsample01.cpp の依存先関係図:



9.5 main.cpp ファイル

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

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)`

9.5.1 マクロ定義詳解

9.5.1.1 HEADER_STR

```
#define HEADER_STR "ゲーム、グラフィックス特論"
```

main.cpp の 15 行目に定義があります。

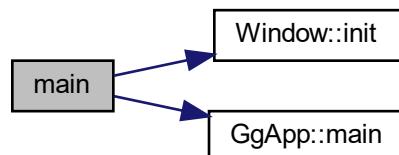
9.5.2 関数詳解

9.5.2.1 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:



9.6 README.md ファイル

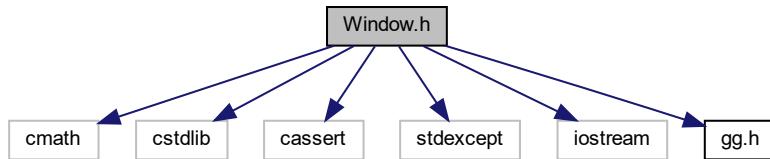
9.7 Window.h ファイル

ゲームグラフィックス特論の宿題用補助プログラムのウィンドウ関連の処理.

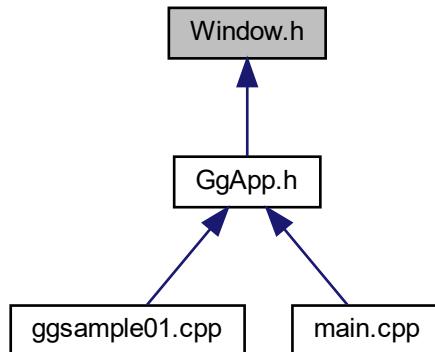
```
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <stdexcept>
#include <iostream>
```

```
#include "gg.h"
```

Window.h の依存先関係図:



被依存関係図:



クラス

- class **Window**
 ウィンドウ関連の処理.

変数

- constexpr int **BUTTON_COUNT** { 3 }
- constexpr int **INTERFACE_COUNT** { 5 }

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のウィンドウ関連の処理.

著者

Kohe Tokoi

日付

March 31, 2021

9.7.2 変数詳解

9.7.2.1 BUTTON_COUNT

```
constexpr int BUTTON_COUNT { 3 } [constexpr]
```

Window.h の 43 行目に定義があります。

9.7.2.2 INTERFACE_COUNT

```
constexpr int INTERFACE_COUNT { 5 } [constexpr]
```

Window.h の 46 行目に定義があります。

Index

_ggError
 gg, 20
_ggFBOError
 gg, 21
~GgBuffer
 gg::GgBuffer< T >, 77
~GgColorTexture
 gg::GgColorTexture, 84
~GgElements
 gg::GgElements, 88
~GgMatrix
 gg::GgMatrix, 96
~GgNormalTexture
 gg::GgNormalTexture, 159
~GgPointShader
 gg::GgPointShader, 168
~GgPoints
 gg::GgPoints, 164
~GgQuaternion
 gg::GgQuaternion, 182
~GgShader
 gg::GgShader, 244
~GgShape
 gg::GgShape, 246
~GgSimpleObj
 gg::GgSimpleObj, 250
~GgSimpleShader
 gg::GgSimpleShader, 255
~GgTexture
 gg::GgTexture, 267
~GgTrackball
 gg::GgTrackball, 272
~GgTriangles
 gg::GgTriangles, 281
~GgUniformBuffer
 gg::GgUniformBuffer< T >, 285
~LightBuffer
 gg::GgSimpleShader::LightBuffer, 298
~MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 309
~Window
 Window, 318

add
 gg::GgMatrix, 97
 gg::GgQuaternion, 182–184
ambient
 gg::GgSimpleShader::Light, 294
 gg::GgSimpleShader::Material, 306

begin
 gg::GgTrackball, 272
bind
 gg::GgBuffer< T >, 78
 gg::GgTexture, 267
 gg::GgUniformBuffer< T >, 286
BindingPoints
 gg, 20
BUTTON_COUNT
 Window.h, 346

conjugate
 gg::GgQuaternion, 185
copy
 gg::GgBuffer< T >, 78
 gg::GgUniformBuffer< T >, 286

diffuse
 gg::GgSimpleShader::Light, 294
 gg::GgSimpleShader::Material, 306

divide
 gg::GgMatrix, 98, 99
 gg::GgQuaternion, 185–187

draw
 gg::GgElements, 89
 gg::GgPoints, 164
 gg::GgShape, 247
 gg::GgSimpleObj, 250
 gg::GgTriangles, 281

end
 gg::GgTrackball, 273

euler
 gg::GgQuaternion, 188

fill
 gg::GgUniformBuffer< T >, 286

frustum
 gg::GgMatrix, 99

get
 gg::GgMatrix, 100–102
 gg::GgPointShader, 168
 gg::GgQuaternion, 189, 190
 gg::GgShader, 244
 gg::GgShape, 248
 gg::GgSimpleObj, 251
 gg::GgTrackball, 273
 Window, 319

getAltArrowX
 Window, 319

getAltArrowY
 Window, 319
getAltArrow
 Window, 319
getArrow
 Window, 320
getArrowX
 Window, 320
getArrowY
 Window, 321
getAspect
 Window, 321
getBuffer
 gg::GgBuffer< T >, 78
 gg::GgPoints, 164
 gg::GgTriangles, 282
 gg::GgUniformBuffer< T >, 287
getConjugateMatrix
 gg::GgQuaternion, 190, 191
getControlArrow
 Window, 321
getControlArrowX
 Window, 322
getControlArrowY
 Window, 322
getCount
 gg::GgBuffer< T >, 78
 gg::GgPoints, 165
 gg::GgTriangles, 282
 gg::GgUniformBuffer< T >, 287
getHeight
 gg::GgTexture, 268
 Window, 322
getIndexBuffer
 gg::GgElements, 89
getIndexCount
 gg::GgElements, 89
getKey
 Window, 323
getMatrix
 gg::GgQuaternion, 192, 193
 gg::GgTrackball, 273
getMode
 gg::GgShape, 248
getMouse
 Window, 323, 324
getMouseX
 Window, 324
getMouseY
 Window, 324
getQuaternion
 gg::GgTrackball, 274
getRotation
 Window, 324
getRotationMatrix
 Window, 325
getScale
 gg::GgTrackball, 274, 275
getScrollMatrix
 Window, 325
getShiftArrow
 Window, 326
getShiftArrowX
 Window, 326
getShiftArrowY
 Window, 326
getSize
 gg::GgTexture, 268
 Window, 326, 327
getStart
 gg::GgTrackball, 275, 276
getStride
 gg::GgBuffer< T >, 79
 gg::GgUniformBuffer< T >, 287
getTarget
 gg::GgBuffer< T >, 79
 gg::GgUniformBuffer< T >, 288
getTexture
 gg::GgTexture, 269
getTranslation
 Window, 327
getTranslationMatrix
 Window, 327
getUserPointer
 Window, 328
getWheel
 Window, 328, 329
getWheelX
 Window, 329
getWheelY
 Window, 329
getWidth
 gg::GgTexture, 269
 Window, 329
gg, 15
 _ggError, 20
 _ggFBOError, 21
 BindingPoints, 20
 ggArraysObj, 21
 ggBufferAlignment, 74
 ggConjugate, 22
 ggCreateComputeShader, 22
 ggCreateNormalMap, 23
 ggCreateShader, 24
 ggCross, 24
 ggDot3, 25
 ggDot4, 25, 26
 ggElementsMesh, 26
 ggElementsObj, 27
 ggElementsSphere, 28
 ggEllipse, 28
 ggEulerQuaternion, 29
 ggFrustum, 30
 ggIdentity, 31
 ggIdentityQuaternion, 32
 ggInit, 32

ggInvert, 33
ggLength3, 34
ggLength4, 35
ggLoadComputeShader, 36
ggLoadHeight, 37
ggLoadImage, 37
ggLoadShader, 38
ggLoadSimpleObj, 39, 40
ggLoadTexture, 40
ggLookat, 41–43
ggMatrixQuaternion, 44
ggNorm, 45
ggNormal, 46
ggNormalize, 46
ggNormalize3, 47
ggNormalize4, 48
ggOrthogonal, 49
ggPerspective, 50
ggPointsCube, 51
ggPointsSphere, 51
ggQuaternion, 53
ggQuaternionMatrix, 54
ggQuaternionTransposeMatrix, 55
ggReadImage, 55
ggRectangle, 56
ggRotate, 57–59
ggRotateQuaternion, 60, 61
ggRotateX, 62
ggRotateY, 63
ggRotateZ, 64
ggSaveColor, 64
ggSaveDepth, 65
ggSaveTga, 66
ggScale, 66–68
ggSlerp, 68–70
ggTranslate, 71–73
ggTranspose, 73
GgVector, 20
LightBindingPoint, 20
MaterialBindingPoint, 20
gg.cpp, 335
gg.h, 336
gg::GgBuffer< T >, 76
 ~GgBuffer, 77
 bind, 78
 copy, 78
 getBuffer, 78
 getCount, 78
 getStride, 79
 getTarget, 79
 GgBuffer, 77
 map, 79, 80
 operator=, 80
 read, 80
 send, 81
 unbind, 81
 unmap, 81
gg::GgColorTexture, 82
 ~GgColorTexture, 84
 GgColorTexture, 82, 83
 load, 84, 85
gg::GgElements, 86
 ~GgElements, 88
 draw, 89
 getIndexBuffer, 89
 getIndexCount, 89
 GgElements, 87, 88
 load, 90
 send, 90
gg::GgMatrix, 91
 ~GgMatrix, 96
 add, 97
 divide, 98, 99
 frustum, 99
 get, 100–102
 GgMatrix, 95, 96
 GgQuaternion, 157
 invert, 102
 load, 102, 103
 loadAdd, 104
 loadDivide, 105, 106
 loadFrustum, 106
 loadIdentity, 107
 loadInvert, 108
 loadLookat, 109, 110
 loadMultiply, 111, 113
 loadNormal, 114
 loadOrthogonal, 115
 loadPerspective, 116
 loadRotate, 117–119
 loadRotateX, 120
 loadRotateY, 121
 loadRotateZ, 122
 loadScale, 122, 123
 loadSubtract, 124, 125
 loadTranslate, 125–127
 loadTranspose, 127, 128
 lookat, 129, 130
 multiply, 131, 132
 normal, 132
 operator*, 133, 134
 operator*=, 134
 operator+, 135
 operator+=, 136
 operator-, 137
 operator-=, 138
 operator/, 139
 operator/=, 140
 operator=, 141
 operator[], 142
 orthogonal, 143
 perspective, 144
 projection, 144, 145
 rotate, 146–148
 rotateX, 149
 rotateY, 150

rotateZ, 150
 scale, 151, 152
 subtract, 153, 154
 translate, 154, 155
 transpose, 156
 gg::GgNormalTexture, 158
 ~GgNormalTexture, 159
 GgNormalTexture, 158, 159
 load, 160, 161
 gg::GgPoints, 162
 ~GgPoints, 164
 draw, 164
 getBuffer, 164
 getCount, 165
 GgPoints, 163
 load, 165
 send, 165
 gg::GgPointShader, 166
 ~GgPointShader, 168
 get, 168
 GgPointShader, 167, 168
 load, 169
 loadMatrix, 170
 loadModelviewMatrix, 171
 loadProjectionMatrix, 171, 172
 unuse, 172
 use, 172–174
 gg::GgQuaternion, 175
 ~GgQuaternion, 182
 add, 182–184
 conjugate, 185
 divide, 185–187
 euler, 188
 get, 189, 190
 getConjugateMatrix, 190, 191
 getMatrix, 192, 193
 GgQuaternion, 179, 181
 invert, 194
 load, 194–196
 loadAdd, 197–199
 loadConjugate, 200
 loadDivide, 201–203
 loadEuler, 204
 loadIdentity, 205
 loadInvert, 206, 207
 loadMatrix, 208
 loadMultiply, 210–212
 loadNormalize, 212, 213
 loadRotate, 214, 215
 loadRotateX, 216
 loadRotateY, 216
 loadRotateZ, 217
 loadSlerp, 218–220
 loadSubtract, 220–222
 multiply, 223, 224
 norm, 225
 normalize, 225
 operator*, 226
 operator*=, 227
 operator+, 228
 operator+=, 229
 operator-, 230
 operator-=, 231
 operator/, 232
 operator/=, 233
 operator=, 234
 rotate, 235, 236
 rotateX, 237
 rotateY, 238
 rotateZ, 238
 slerp, 239
 subtract, 240, 241
 gg::GgShader, 242
 ~GgShader, 244
 get, 244
 GgShader, 243, 244
 operator=, 244
 unuse, 244
 use, 245
 gg::GgShape, 245
 ~GgShape, 246
 draw, 247
 get, 248
 getMode, 248
 GgShape, 246, 247
 operator=, 248
 setMode, 249
 gg::GgSimpleObj, 249
 ~GgSimpleObj, 250
 draw, 250
 get, 251
 GgSimpleObj, 250
 gg::GgSimpleShader, 251
 ~GgSimpleShader, 255
 GgSimpleShader, 254, 255
 load, 255
 loadMatrix, 256, 257
 loadModelviewMatrix, 258, 259
 operator=, 260
 use, 260–265
 gg::GgSimpleShader::Light, 294
 ambient, 294
 diffuse, 294
 position, 295
 specular, 295
 gg::GgSimpleShader::LightBuffer, 295
 ~LightBuffer, 298
 LightBuffer, 297
 load, 298
 loadAmbient, 299, 300
 loadColor, 300
 loadDiffuse, 300, 301
 loadPosition, 302, 303
 loadSpecular, 303, 304
 select, 305
 gg::GgSimpleShader::Material, 305

ambient, 306
diffuse, 306
shininess, 306
specular, 306
gg::GgSimpleShader::MaterialBuffer, 307
 ~MaterialBuffer, 309
 load, 309, 310
 loadAmbient, 310
 loadAmbientAndDiffuse, 311
 loadDiffuse, 312
 loadShininess, 313
 loadSpecular, 313, 314
 MaterialBuffer, 308, 309
 select, 314
gg::GgTexture, 266
 ~GgTexture, 267
 bind, 267
 getHeight, 268
 getSize, 268
 getTexture, 269
 getWidth, 269
 GgTexture, 266, 267
 operator=, 270
 unbind, 270
gg::GgTrackball, 270
 ~GgTrackball, 272
 begin, 272
 end, 273
 get, 273
 getMatrix, 273
 getQuaternion, 274
 getScale, 274, 275
 getStart, 275, 276
 GgTrackball, 271
 motion, 276
 region, 276, 277
 reset, 278
 rotate, 278
gg::GgTriangles, 279
 ~GgTriangles, 281
 draw, 281
 getBuffer, 282
 getCount, 282
 GgTriangles, 280, 281
 load, 282
 send, 283
gg::GgUniformBuffer< T >, 283
 ~GgUniformBuffer, 285
 bind, 286
 copy, 286
 fill, 286
 getBuffer, 287
 getCount, 287
 getStride, 287
 getTarget, 288
 GgUniformBuffer, 284, 285
 load, 288
 map, 289
 read, 290
 send, 290
 unbind, 291
 unmap, 291
gg::GgVertex, 291
 GgVertex, 292, 293
 normal, 293
 position, 293
GgApp, 75
 main, 75
GgApp.h, 341
 USE_IMGUI, 342
ggArraysObj
 gg, 21
GgBuffer
 gg::GgBuffer< T >, 77
ggBufferAlignment
 gg, 74
GgColorTexture
 gg::GgColorTexture, 82, 83
ggConjugate
 gg, 22
ggCreateComputeShader
 gg, 22
ggCreateNormalMap
 gg, 23
ggCreateShader
 gg, 24
ggCross
 gg, 24
ggDot3
 gg, 25
ggDot4
 gg, 25, 26
GgElements
 gg::GgElements, 87, 88
ggElementsMesh
 gg, 26
ggElementsObj
 gg, 27
ggElementsSphere
 gg, 28
ggEllipse
 gg, 28
ggEulerQuaternion
 gg, 29
ggFrustum
 gg, 30
ggIdentity
 gg, 31
ggIdentityQuaternion
 gg, 32
ggInit
 gg, 32
ggInvert
 gg, 33
ggLength3
 gg, 34

ggLength4
 gg, 35
 ggLoadComputeShader
 gg, 36
 ggLoadHeight
 gg, 37
 ggLoadImage
 gg, 37
 ggLoadShader
 gg, 38
 ggLoadSimpleObj
 gg, 39, 40
 ggLoadTexture
 gg, 40
 ggLookat
 gg, 41–43
 GgMatrix
 gg::GgMatrix, 95, 96
 ggMatrixQuaternion
 gg, 44
 ggNorm
 gg, 45
 ggNormal
 gg, 46
 ggNormalize
 gg, 46
 ggNormalize3
 gg, 47
 ggNormalize4
 gg, 48
 GgNormalTexture
 gg::GgNormalTexture, 158, 159
 ggOrthogonal
 gg, 49
 ggPerspective
 gg, 50
 GgPoints
 gg::GgPoints, 163
 ggPointsCube
 gg, 51
 GgPointShader
 gg::GgPointShader, 167, 168
 ggPointsSphere
 gg, 51
 GgQuaternion
 gg::GgMatrix, 157
 gg::GgQuaternion, 179, 181
 ggQuaternion
 gg, 53
 ggQuaternionMatrix
 gg, 54
 ggQuaternionTransposeMatrix
 gg, 55
 ggReadImage
 gg, 55
 ggRectangle
 gg, 56
 ggRotate
 gg, 57–59
 ggRotateQuaternion
 gg, 60, 61
 ggRotateX
 gg, 62
 ggRotateY
 gg, 63
 ggRotateZ
 gg, 64
 ggsample01.cpp, 342
 ggSaveColor
 gg, 64
 ggSaveDepth
 gg, 65
 ggSaveTga
 gg, 66
 ggScale
 gg, 66–68
 GgShader
 gg::GgShader, 243, 244
 GgShape
 gg::GgShape, 246, 247
 GgSimpleObj
 gg::GgSimpleObj, 250
 GgSimpleShader
 gg::GgSimpleShader, 254, 255
 ggSlerp
 gg, 68–70
 GgTexture
 gg::GgTexture, 266, 267
 GgTrackball
 gg::GgTrackball, 271
 ggTranslate
 gg, 71–73
 ggTranspose
 gg, 73
 GgTriangles
 gg::GgTriangles, 280, 281
 GgUniformBuffer
 gg::GgUniformBuffer< T >, 284, 285
 GgVector
 gg, 20
 GgVertex
 gg::GgVertex, 292, 293

 HEADER_STR
 main.cpp, 344

 init
 Window, 330
 INTERFACE_COUNT
 Window.h, 346
 invert
 gg::GgMatrix, 102
 gg::GgQuaternion, 194

 LightBindingPoint
 gg, 20
 LightBuffer

gg::GgSimpleShader::LightBuffer, 297
load
 gg::GgColorTexture, 84, 85
 gg::GgElements, 90
 gg::GgMatrix, 102, 103
 gg::GgNormalTexture, 160, 161
 gg::GgPoints, 165
 gg::GgPointShader, 169
 gg::GgQuaternion, 194–196
 gg::GgSimpleShader, 255
 gg::GgSimpleShader::LightBuffer, 298
 gg::GgSimpleShader::MaterialBuffer, 309, 310
 gg::GgTriangles, 282
 gg::GgUniformBuffer< T >, 288
loadAdd
 gg::GgMatrix, 104
 gg::GgQuaternion, 197–199
loadAmbient
 gg::GgSimpleShader::LightBuffer, 299, 300
 gg::GgSimpleShader::MaterialBuffer, 310
loadAmbientAndDiffuse
 gg::GgSimpleShader::MaterialBuffer, 311
loadColor
 gg::GgSimpleShader::LightBuffer, 300
loadConjugate
 gg::GgQuaternion, 200
loadDiffuse
 gg::GgSimpleShader::LightBuffer, 300, 301
 gg::GgSimpleShader::MaterialBuffer, 312
loadDivide
 gg::GgMatrix, 105, 106
 gg::GgQuaternion, 201–203
loadEuler
 gg::GgQuaternion, 204
loadFrustum
 gg::GgMatrix, 106
loadIdentity
 gg::GgMatrix, 107
 gg::GgQuaternion, 205
loadInvert
 gg::GgMatrix, 108
 gg::GgQuaternion, 206, 207
loadLookat
 gg::GgMatrix, 109, 110
loadMatrix
 gg::GgPointShader, 170
 gg::GgQuaternion, 208
 gg::GgSimpleShader, 256, 257
loadModelviewMatrix
 gg::GgPointShader, 171
 gg::GgSimpleShader, 258, 259
loadMultiply
 gg::GgMatrix, 111, 113
 gg::GgQuaternion, 210–212
loadNormal
 gg::GgMatrix, 114
loadNormalize
 gg::GgQuaternion, 212, 213
loadOrthogonal
 gg::GgMatrix, 115
loadPerspective
 gg::GgMatrix, 116
loadPosition
 gg::GgSimpleShader::LightBuffer, 302, 303
loadProjectionMatrix
 gg::GgPointShader, 171, 172
loadRotate
 gg::GgMatrix, 117–119
 gg::GgQuaternion, 214, 215
loadRotateX
 gg::GgMatrix, 120
 gg::GgQuaternion, 216
loadRotateY
 gg::GgMatrix, 121
 gg::GgQuaternion, 216
loadRotateZ
 gg::GgMatrix, 122
 gg::GgQuaternion, 217
loadScale
 gg::GgMatrix, 122, 123
loadShininess
 gg::GgSimpleShader::MaterialBuffer, 313
loadSlerp
 gg::GgQuaternion, 218–220
loadSpecular
 gg::GgSimpleShader::LightBuffer, 303, 304
 gg::GgSimpleShader::MaterialBuffer, 313, 314
loadSubtract
 gg::GgMatrix, 124, 125
 gg::GgQuaternion, 220–222
loadTranslate
 gg::GgMatrix, 125–127
loadTranspose
 gg::GgMatrix, 127, 128
lookat
 gg::GgMatrix, 129, 130
main
 GgApp, 75
 main.cpp, 344
main.cpp, 343
 HEADER_STR, 344
 main, 344
map
 gg::GgBuffer< T >, 79, 80
 gg::GgUniformBuffer< T >, 289
MaterialBindingPoint
 gg, 20
MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 308, 309
motion
 gg::GgTrackball, 276
multiply
 gg::GgMatrix, 131, 132
 gg::GgQuaternion, 223, 224
norm

gg::GgQuaternion, 225
normal
 gg::GgMatrix, 132
 gg::GgVertex, 293
normalize
 gg::GgQuaternion, 225
operator bool
 Window, 330
operator*
 gg::GgMatrix, 133, 134
 gg::GgQuaternion, 226
operator*=
 gg::GgMatrix, 134
 gg::GgQuaternion, 227
operator+
 gg::GgMatrix, 135
 gg::GgQuaternion, 228
operator+=
 gg::GgMatrix, 136
 gg::GgQuaternion, 229
operator-
 gg::GgMatrix, 137
 gg::GgQuaternion, 230
operator-=
 gg::GgMatrix, 138
 gg::GgQuaternion, 231
operator/
 gg::GgMatrix, 139
 gg::GgQuaternion, 232
operator/=
 gg::GgMatrix, 140
 gg::GgQuaternion, 233
operator=
 gg::GgBuffer< T >, 80
 gg::GgMatrix, 141
 gg::GgQuaternion, 234
 gg::GgShader, 244
 gg::GgShape, 248
 gg::GgSimpleShader, 260
 gg::GgTexture, 270
 Window, 331
operator[]
 gg::GgMatrix, 142
orthogonal
 gg::GgMatrix, 143
perspective
 gg::GgMatrix, 144
position
 gg::GgSimpleShader::Light, 295
 gg::GgVertex, 293
projection
 gg::GgMatrix, 144, 145
read
 gg::GgBuffer< T >, 80
 gg::GgUniformBuffer< T >, 290
README.md, 344
region
 gg::GgTrackball, 276, 277
reset
 gg::GgTrackball, 278
 Window, 331
resetRotation
 Window, 331
resetTranslation
 Window, 331
restoreViewport
 Window, 331
rotate
 gg::GgMatrix, 146–148
 gg::GgQuaternion, 235, 236
 gg::GgTrackball, 278
rotateX
 gg::GgMatrix, 149
 gg::GgQuaternion, 237
rotateY
 gg::GgMatrix, 150
 gg::GgQuaternion, 238
rotateZ
 gg::GgMatrix, 150
 gg::GgQuaternion, 238
scale
 gg::GgMatrix, 151, 152
select
 gg::GgSimpleShader::LightBuffer, 305
 gg::GgSimpleShader::MaterialBuffer, 314
selectInterface
 Window, 331
send
 gg::GgBuffer< T >, 81
 gg::GgElements, 90
 gg::GgPoints, 165
 gg::GgTriangles, 283
 gg::GgUniformBuffer< T >, 290
setClose
 Window, 332
setKeyboardFunc
 Window, 332
setMode
 gg::GgShape, 249
setMouseFunc
 Window, 332
setResizeFunc
 Window, 333
setUserPointer
 Window, 333
setVelocity
 Window, 334
shininess
 gg::GgSimpleShader::Material, 306
shouldClose
 Window, 334
slerp
 gg::GgQuaternion, 239
specular

gg::GgSimpleShader::Light, 295
gg::GgSimpleShader::Material, 306
subtract
 gg::GgMatrix, 153, 154
 gg::GgQuaternion, 240, 241
swapBuffers
 Window, 334

translate
 gg::GgMatrix, 154, 155
transpose
 gg::GgMatrix, 156

unbind
 gg::GgBuffer< T >, 81
 gg::GgTexture, 270
 gg::GgUniformBuffer< T >, 291
unmap
 gg::GgBuffer< T >, 81
 gg::GgUniformBuffer< T >, 291
unuse
 gg::GgPointShader, 172
 gg::GgShader, 244
use
 gg::GgPointShader, 172–174
 gg::GgShader, 245
 gg::GgSimpleShader, 260–265
USE_IMGUI
 GgApp.h, 342

Window, 315
 ~Window, 318
 get, 319
 getAltArrowX, 319
 getAltArrowY, 319
 getAltArrow, 319
 getArrow, 320
 getArrowX, 320
 getArrowY, 321
 getAspect, 321
 getControlArrow, 321
 getControlArrowX, 322
 getControlArrowY, 322
 getHeight, 322
 getKey, 323
 getMouse, 323, 324
 getMouseX, 324
 getMouseY, 324
 getRotation, 324
 getRotationMatrix, 325
 getScrollMatrix, 325
 getShiftArrow, 326
 getShiftArrowX, 326
 getShiftArrowY, 326
 getSize, 326, 327
 getTranslation, 327
 getTranslationMatrix, 327
 getUserPointer, 328
 getWheel, 328, 329
 getWheelX, 329
 getWheelY, 329
 getWidth, 329
 init, 330
 operator bool, 330
 operator=, 331
 reset, 331
 resetRotation, 331
 resetTranslation, 331
 restoreViewport, 331
 selectInterface, 331
 setClose, 332
 setKeyboardFunc, 332
 setMouseFunc, 332
 setResizeFunc, 333
 setUserPointer, 333
 setVelocity, 334
 shouldClose, 334
 swapBuffers, 334
 Window, 317, 318
Window.h, 344
 BUTTON_COUNT, 346
 INTERFACE_COUNT, 346