

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	18
7.1.2 列挙型詳解	18
7.1.2.1 BindingPoints	18
7.1.3 関数詳解	18
7.1.3.1 ggError()	18
7.1.3.2 ggFBOError()	19
7.1.3.3 ggArraysObj()	19
7.1.3.4 ggConjugate()	20
7.1.3.5 ggCreateComputeShader()	21
7.1.3.6 ggCreateNormalMap()	21
7.1.3.7 ggCreateShader()	22
7.1.3.8 ggCross() [1/2]	23
7.1.3.9 ggCross() [2/2]	23
7.1.3.10 ggDistance3() [1/2]	24
7.1.3.11 ggDistance3() [2/2]	25
7.1.3.12 ggDistance4() [1/2]	25
7.1.3.13 ggDistance4() [2/2]	26

7.1.3.14 ggDot3() [1/2]	27
7.1.3.15 ggDot3() [2/2]	28
7.1.3.16 ggDot4() [1/2]	29
7.1.3.17 ggDot4() [2/2]	29
7.1.3.18 ggElementsMesh()	30
7.1.3.19 ggElementsObj()	31
7.1.3.20 ggElementsSphere()	32
7.1.3.21 ggEllipse()	33
7.1.3.22 ggEulerQuaternion() [1/2]	33
7.1.3.23 ggEulerQuaternion() [2/2]	34
7.1.3.24 ggFrustum()	35
7.1.3.25 ggIdentity()	35
7.1.3.26 ggIdentityQuaternion()	36
7.1.3.27 ggInit()	37
7.1.3.28 ggInvert() [1/2]	37
7.1.3.29 ggInvert() [2/2]	38
7.1.3.30 ggLength3() [1/2]	38
7.1.3.31 ggLength3() [2/2]	39
7.1.3.32 ggLength4() [1/2]	40
7.1.3.33 ggLength4() [2/2]	41
7.1.3.34 ggLoadComputeShader()	41
7.1.3.35 ggLoadHeight()	42
7.1.3.36 ggLoadImage()	43
7.1.3.37 ggLoadShader() [1/2]	43
7.1.3.38 ggLoadShader() [2/2]	44
7.1.3.39 ggLoadSimpleObj() [1/2]	45
7.1.3.40 ggLoadSimpleObj() [2/2]	46
7.1.3.41 ggLoadTexture()	47
7.1.3.42 ggLookat() [1/3]	47
7.1.3.43 ggLookat() [2/3]	48
7.1.3.44 ggLookat() [3/3]	49
7.1.3.45 ggMatrixQuaternion() [1/2]	50
7.1.3.46 ggMatrixQuaternion() [2/2]	51
7.1.3.47 ggNorm()	51
7.1.3.48 ggNormal()	52
7.1.3.49 ggNormalize()	53
7.1.3.50 ggNormalize3() [1/4]	53
7.1.3.51 ggNormalize3() [2/4]	54
7.1.3.52 ggNormalize3() [3/4]	55
7.1.3.53 ggNormalize3() [4/4]	56
7.1.3.54 ggNormalize4() [1/4]	56
7.1.3.55 ggNormalize4() [2/4]	57

7.1.3.56 ggNormalize4() [3/4]	57
7.1.3.57 ggNormalize4() [4/4]	58
7.1.3.58 ggOrthogonal()	58
7.1.3.59 ggPerspective()	59
7.1.3.60 ggPointsCube()	60
7.1.3.61 ggPointsSphere()	61
7.1.3.62 ggQuaternion() [1/2]	61
7.1.3.63 ggQuaternion() [2/2]	62
7.1.3.64 ggQuaternionMatrix()	63
7.1.3.65 ggQuaternionTransposeMatrix()	63
7.1.3.66 ggReadImage()	64
7.1.3.67 ggRectangle()	65
7.1.3.68 ggRotate() [1/5]	66
7.1.3.69 ggRotate() [2/5]	67
7.1.3.70 ggRotate() [3/5]	68
7.1.3.71 ggRotate() [4/5]	68
7.1.3.72 ggRotate() [5/5]	69
7.1.3.73 ggRotateQuaternion() [1/3]	70
7.1.3.74 ggRotateQuaternion() [2/3]	70
7.1.3.75 ggRotateQuaternion() [3/3]	71
7.1.3.76 ggRotateX()	72
7.1.3.77 ggRotateY()	73
7.1.3.78 ggRotateZ()	73
7.1.3.79 ggSaveColor()	74
7.1.3.80 ggSaveDepth()	75
7.1.3.81 ggSaveTga()	75
7.1.3.82 ggScale() [1/3]	76
7.1.3.83 ggScale() [2/3]	77
7.1.3.84 ggScale() [3/3]	77
7.1.3.85 ggSlerp() [1/4]	78
7.1.3.86 ggSlerp() [2/4]	79
7.1.3.87 ggSlerp() [3/4]	79
7.1.3.88 ggSlerp() [4/4]	80
7.1.3.89 ggTranslate() [1/3]	81
7.1.3.90 ggTranslate() [2/3]	81
7.1.3.91 ggTranslate() [3/3]	82
7.1.3.92 ggTranspose()	83
7.1.3.93 operator*()	83
7.1.3.94 operator+() [1/2]	84
7.1.3.95 operator+() [2/2]	84
7.1.3.96 operator-() [1/2]	85
7.1.3.97 operator-() [2/2]	85

7.1.3.98 operator/()	86
7.1.4 変数詳解	86
7.1.4.1 ggBufferAlignment	86
8 クラス詳解	87
8.1 Config クラス	87
8.1.1 詳解	87
8.1.2 構築子と解体子	87
8.1.2.1 Config() [1/2]	87
8.1.2.2 Config() [2/2]	88
8.1.2.3 ~Config()	88
8.1.3 関数詳解	88
8.1.3.1 load()	89
8.1.3.2 save()	89
8.1.4 フレンドと関連関数の詳解	90
8.1.4.1 Menu	90
8.2 Draw クラス	90
8.2.1 詳解	91
8.2.2 構築子と解体子	91
8.2.2.1 Draw()	91
8.2.2.2 ~Draw()	91
8.2.3 関数詳解	91
8.2.3.1 draw()	92
8.3 GgApp クラス	92
8.3.1 詳解	92
8.3.2 関数詳解	92
8.3.2.1 main()	93
8.4 gg::GgBuffer< T > クラステンプレート	93
8.4.1 詳解	94
8.4.2 構築子と解体子	94
8.4.2.1 GgBuffer() [1/2]	94
8.4.2.2 ~GgBuffer()	95
8.4.2.3 GgBuffer() [2/2]	95
8.4.3 関数詳解	95
8.4.3.1 bind()	95
8.4.3.2 copy()	95
8.4.3.3 getBuffer()	96
8.4.3.4 getCount()	96
8.4.3.5 getStride()	96
8.4.3.6 getTarget()	97
8.4.3.7 map() [1/2]	97
8.4.3.8 map() [2/2]	97

8.4.3.9 operator=()	98
8.4.3.10 read()	98
8.4.3.11 send()	98
8.4.3.12 unbind()	99
8.4.3.13 unmap()	99
8.5 gg::GgColorTexture クラス	99
8.5.1 詳解	99
8.5.2 構築子と解体子	100
8.5.2.1 GgColorTexture() [1/3]	100
8.5.2.2 GgColorTexture() [2/3]	100
8.5.2.3 GgColorTexture() [3/3]	101
8.5.2.4 ~GgColorTexture()	101
8.5.3 関数詳解	101
8.5.3.1 load() [1/2]	102
8.5.3.2 load() [2/2]	102
8.6 gg::GgElements クラス	103
8.6.1 詳解	104
8.6.2 構築子と解体子	104
8.6.2.1 GgElements() [1/2]	104
8.6.2.2 GgElements() [2/2]	105
8.6.2.3 ~GgElements()	105
8.6.3 関数詳解	105
8.6.3.1 draw()	105
8.6.3.2 getIndexBuffer()	106
8.6.3.3 getIndexCount()	106
8.6.3.4 load()	106
8.6.3.5 send()	107
8.7 gg::GgMatrix クラス	107
8.7.1 詳解	110
8.7.2 構築子と解体子	110
8.7.2.1 GgMatrix() [1/4]	110
8.7.2.2 GgMatrix() [2/4]	110
8.7.2.3 GgMatrix() [3/4]	111
8.7.2.4 GgMatrix() [4/4]	111
8.7.3 関数詳解	112
8.7.3.1 frustum()	112
8.7.3.2 get() [1/2]	113
8.7.3.3 get() [2/2]	114
8.7.3.4 invert()	115
8.7.3.5 loadFrustum()	115
8.7.3.6 loadIdentity()	116
8.7.3.7 loadInvert() [1/2]	117

8.7.3.8 loadInvert() [2/2]	117
8.7.3.9 loadLookat() [1/3]	118
8.7.3.10 loadLookat() [2/3]	119
8.7.3.11 loadLookat() [3/3]	119
8.7.3.12 loadNormal() [1/2]	120
8.7.3.13 loadNormal() [2/2]	121
8.7.3.14 loadOrthogonal()	122
8.7.3.15 loadPerspective()	122
8.7.3.16 loadRotate() [1/5]	123
8.7.3.17 loadRotate() [2/5]	124
8.7.3.18 loadRotate() [3/5]	124
8.7.3.19 loadRotate() [4/5]	125
8.7.3.20 loadRotate() [5/5]	126
8.7.3.21 loadRotateX()	127
8.7.3.22 loadRotateY()	127
8.7.3.23 loadRotateZ()	128
8.7.3.24 loadScale() [1/3]	129
8.7.3.25 loadScale() [2/3]	129
8.7.3.26 loadScale() [3/3]	130
8.7.3.27 loadTranslate() [1/3]	131
8.7.3.28 loadTranslate() [2/3]	131
8.7.3.29 loadTranslate() [3/3]	132
8.7.3.30 loadTranspose() [1/2]	133
8.7.3.31 loadTranspose() [2/2]	133
8.7.3.32 lookat() [1/3]	134
8.7.3.33 lookat() [2/3]	135
8.7.3.34 lookat() [3/3]	135
8.7.3.35 normal()	137
8.7.3.36 operator*() [1/3]	137
8.7.3.37 operator*() [2/3]	138
8.7.3.38 operator*() [3/3]	138
8.7.3.39 operator*=(()) [1/2]	139
8.7.3.40 operator*=(()) [2/2]	140
8.7.3.41 operator+() [1/2]	141
8.7.3.42 operator+() [2/2]	141
8.7.3.43 operator+=(()) [1/2]	142
8.7.3.44 operator+=(()) [2/2]	143
8.7.3.45 operator-() [1/2]	143
8.7.3.46 operator-() [2/2]	144
8.7.3.47 operator-=(()) [1/2]	144
8.7.3.48 operator-=(()) [2/2]	145
8.7.3.49 operator/() [1/2]	146

8.7.3.50 operator/() [2/2]	146
8.7.3.51 operator/=(()) [1/2]	147
8.7.3.52 operator/=(()) [2/2]	148
8.7.3.53 operator=()	148
8.7.3.54 orthogonal()	149
8.7.3.55 perspective()	150
8.7.3.56 projection() [1/4]	151
8.7.3.57 projection() [2/4]	151
8.7.3.58 projection() [3/4]	151
8.7.3.59 projection() [4/4]	152
8.7.3.60 rotate() [1/5]	152
8.7.3.61 rotate() [2/5]	153
8.7.3.62 rotate() [3/5]	153
8.7.3.63 rotate() [4/5]	154
8.7.3.64 rotate() [5/5]	155
8.7.3.65 rotateX()	156
8.7.3.66 rotateY()	156
8.7.3.67 rotateZ()	157
8.7.3.68 scale() [1/3]	158
8.7.3.69 scale() [2/3]	158
8.7.3.70 scale() [3/3]	159
8.7.3.71 translate() [1/3]	160
8.7.3.72 translate() [2/3]	160
8.7.3.73 translate() [3/3]	161
8.7.3.74 transpose()	162
8.8 gg::GgNormalTexture クラス	163
8.8.1 詳解	163
8.8.2 構築子と解体子	163
8.8.2.1 GgNormalTexture() [1/3]	164
8.8.2.2 GgNormalTexture() [2/3]	164
8.8.2.3 GgNormalTexture() [3/3]	164
8.8.2.4 ~GgNormalTexture()	165
8.8.3 関数詳解	165
8.8.3.1 load() [1/2]	165
8.8.3.2 load() [2/2]	166
8.9 gg::GgPoints クラス	167
8.9.1 詳解	168
8.9.2 構築子と解体子	168
8.9.2.1 GgPoints() [1/2]	168
8.9.2.2 GgPoints() [2/2]	168
8.9.2.3 ~GgPoints()	168
8.9.3 関数詳解	169

8.9.3.1 draw()	169
8.9.3.2 getBuffer()	169
8.9.3.3 getCount()	170
8.9.3.4 load()	170
8.9.3.5 send()	170
8.10 gg::GgPointShader クラス	171
8.10.1 詳解	172
8.10.2 構築子と解体子	172
8.10.2.1 GgPointShader() [1/3]	172
8.10.2.2 GgPointShader() [2/3]	172
8.10.2.3 GgPointShader() [3/3]	172
8.10.2.4 ~GgPointShader()	173
8.10.3 関数詳解	173
8.10.3.1 get()	173
8.10.3.2 load() [1/2]	173
8.10.3.3 load() [2/2]	174
8.10.3.4 loadMatrix() [1/2]	175
8.10.3.5 loadMatrix() [2/2]	175
8.10.3.6 loadModelviewMatrix() [1/2]	176
8.10.3.7 loadModelviewMatrix() [2/2]	176
8.10.3.8 loadProjectionMatrix() [1/2]	177
8.10.3.9 loadProjectionMatrix() [2/2]	177
8.10.3.10 unuse()	177
8.10.3.11 use() [1/5]	178
8.10.3.12 use() [2/5]	178
8.10.3.13 use() [3/5]	178
8.10.3.14 use() [4/5]	179
8.10.3.15 use() [5/5]	179
8.11 gg::GgQuaternion クラス	180
8.11.1 詳解	183
8.11.2 構築子と解体子	183
8.11.2.1 GgQuaternion() [1/5]	183
8.11.2.2 GgQuaternion() [2/5]	183
8.11.2.3 GgQuaternion() [3/5]	183
8.11.2.4 GgQuaternion() [4/5]	184
8.11.2.5 GgQuaternion() [5/5]	184
8.11.3 関数詳解	184
8.11.3.1 add() [1/4]	184
8.11.3.2 add() [2/4]	185
8.11.3.3 add() [3/4]	186
8.11.3.4 add() [4/4]	186
8.11.3.5 conjugate()	187

8.11.3.6 divide() [1/4]	188
8.11.3.7 divide() [2/4]	188
8.11.3.8 divide() [3/4]	189
8.11.3.9 divide() [4/4]	190
8.11.3.10 euler() [1/2]	190
8.11.3.11 euler() [2/2]	191
8.11.3.12 get()	192
8.11.3.13 getConjugateMatrix() [1/3]	192
8.11.3.14 getConjugateMatrix() [2/3]	193
8.11.3.15 getConjugateMatrix() [3/3]	193
8.11.3.16 getMatrix() [1/3]	194
8.11.3.17 getMatrix() [2/3]	194
8.11.3.18 getMatrix() [3/3]	195
8.11.3.19 invert()	196
8.11.3.20 load() [1/4]	196
8.11.3.21 load() [2/4]	197
8.11.3.22 load() [3/4]	197
8.11.3.23 load() [4/4]	198
8.11.3.24 loadAdd() [1/4]	199
8.11.3.25 loadAdd() [2/4]	200
8.11.3.26 loadAdd() [3/4]	200
8.11.3.27 loadAdd() [4/4]	201
8.11.3.28 loadConjugate() [1/2]	202
8.11.3.29 loadConjugate() [2/2]	202
8.11.3.30 loadDivide() [1/4]	203
8.11.3.31 loadDivide() [2/4]	203
8.11.3.32 loadDivide() [3/4]	204
8.11.3.33 loadDivide() [4/4]	205
8.11.3.34 loadEuler() [1/2]	206
8.11.3.35 loadEuler() [2/2]	206
8.11.3.36 loadIdentity()	207
8.11.3.37 loadInvert() [1/2]	208
8.11.3.38 loadInvert() [2/2]	209
8.11.3.39 loadMatrix() [1/2]	210
8.11.3.40 loadMatrix() [2/2]	211
8.11.3.41 loadMultiply() [1/4]	212
8.11.3.42 loadMultiply() [2/4]	213
8.11.3.43 loadMultiply() [3/4]	213
8.11.3.44 loadMultiply() [4/4]	214
8.11.3.45 loadNormalize() [1/2]	215
8.11.3.46 loadNormalize() [2/2]	215
8.11.3.47 loadRotate() [1/3]	216

8.11.3.48 loadRotate() [2/3]	216
8.11.3.49 loadRotate() [3/3]	217
8.11.3.50 loadRotateX()	218
8.11.3.51 loadRotateY()	219
8.11.3.52 loadRotateZ()	219
8.11.3.53 loadSlerp() [1/4]	220
8.11.3.54 loadSlerp() [2/4]	221
8.11.3.55 loadSlerp() [3/4]	221
8.11.3.56 loadSlerp() [4/4]	222
8.11.3.57 loadSubtract() [1/4]	223
8.11.3.58 loadSubtract() [2/4]	223
8.11.3.59 loadSubtract() [3/4]	224
8.11.3.60 loadSubtract() [4/4]	224
8.11.3.61 multiply() [1/4]	225
8.11.3.62 multiply() [2/4]	226
8.11.3.63 multiply() [3/4]	226
8.11.3.64 multiply() [4/4]	226
8.11.3.65 norm()	227
8.11.3.66 normalize()	228
8.11.3.67 operator*() [1/3]	228
8.11.3.68 operator*() [2/3]	229
8.11.3.69 operator*() [3/3]	229
8.11.3.70 operator*() [1/3]	229
8.11.3.71 operator*() [2/3]	230
8.11.3.72 operator*() [3/3]	230
8.11.3.73 operator+() [1/3]	231
8.11.3.74 operator+() [2/3]	231
8.11.3.75 operator+() [3/3]	231
8.11.3.76 operator+=() [1/3]	232
8.11.3.77 operator+=() [2/3]	232
8.11.3.78 operator+=() [3/3]	233
8.11.3.79 operator-() [1/3]	233
8.11.3.80 operator-() [2/3]	233
8.11.3.81 operator-() [3/3]	234
8.11.3.82 operator-=() [1/3]	234
8.11.3.83 operator-=() [2/3]	235
8.11.3.84 operator-=() [3/3]	235
8.11.3.85 operator/() [1/3]	236
8.11.3.86 operator/() [2/3]	236
8.11.3.87 operator/() [3/3]	236
8.11.3.88 operator/=() [1/3]	237
8.11.3.89 operator/=() [2/3]	237

8.11.3.90 operator/() [3/3]	238
8.11.3.91 operator=() [1/2]	238
8.11.3.92 operator=() [2/2]	239
8.11.3.93 rotate() [1/3]	239
8.11.3.94 rotate() [2/3]	240
8.11.3.95 rotate() [3/3]	240
8.11.3.96 rotateX()	241
8.11.3.97 rotateY()	242
8.11.3.98 rotateZ()	242
8.11.3.99 slerp() [1/2]	243
8.11.3.100 slerp() [2/2]	243
8.11.3.101 subtract() [1/4]	244
8.11.3.102 subtract() [2/4]	244
8.11.3.103 subtract() [3/4]	245
8.11.3.104 subtract() [4/4]	246
8.12 gg::GgShader クラス	247
8.12.1 詳解	247
8.12.2 構築子と解体子	247
8.12.2.1 GgShader() [1/3]	247
8.12.2.2 GgShader() [2/3]	248
8.12.2.3 GgShader() [3/3]	248
8.12.2.4 ~GgShader()	248
8.12.3 関数詳解	248
8.12.3.1 get()	249
8.12.3.2 operator=()	249
8.12.3.3 unuse()	249
8.12.3.4 use()	249
8.13 gg::GgShape クラス	250
8.13.1 詳解	250
8.13.2 構築子と解体子	250
8.13.2.1 GgShape() [1/2]	250
8.13.2.2 ~GgShape()	251
8.13.2.3 GgShape() [2/2]	251
8.13.3 関数詳解	251
8.13.3.1 draw()	251
8.13.3.2 get()	252
8.13.3.3 getMode()	252
8.13.3.4 operator=()	253
8.13.3.5 setMode()	253
8.14 gg::GgSimpleObj クラス	253
8.14.1 詳解	253
8.14.2 構築子と解体子	253

8.14.2.1 GgSimpleObj()	253
8.14.2.2 ~GgSimpleObj()	254
8.14.3 関数詳解	254
8.14.3.1 draw()	254
8.14.3.2 get()	255
8.15 gg::GgSimpleShader クラス	256
8.15.1 詳解	257
8.15.2 構築子と解体子	257
8.15.2.1 GgSimpleShader() [1/4]	257
8.15.2.2 GgSimpleShader() [2/4]	257
8.15.2.3 GgSimpleShader() [3/4]	258
8.15.2.4 GgSimpleShader() [4/4]	258
8.15.2.5 ~GgSimpleShader()	258
8.15.3 関数詳解	259
8.15.3.1 load() [1/2]	259
8.15.3.2 load() [2/2]	259
8.15.3.3 loadMatrix()	260
8.15.3.4 loadMatrix()	261
8.15.3.5 loadMatrix()	261
8.15.3.6 loadMatrix()	262
8.15.3.7 loadModelviewMatrix()	262
8.15.3.8 loadModelviewMatrix()	263
8.15.3.9 loadModelviewMatrix()	263
8.15.3.10 loadModelviewMatrix()	263
8.15.3.11 operator=()	264
8.15.3.12 use() [1/13]	264
8.15.3.13 use() [2/13]	264
8.15.3.14 use() [3/13]	265
8.15.3.15 use() [4/13]	266
8.15.3.16 use() [5/13]	266
8.15.3.17 use() [6/13]	267
8.15.3.18 use() [7/13]	267
8.15.3.19 use() [8/13]	268
8.15.3.20 use() [9/13]	268
8.15.3.21 use() [10/13]	269
8.15.3.22 use() [11/13]	269
8.15.3.23 use() [12/13]	269
8.15.3.24 use() [13/13]	270
8.16 gg::GgTexture クラス	270
8.16.1 詳解	271
8.16.2 構築子と解体子	271
8.16.2.1 GgTexture() [1/2]	271

8.16.2.2 ~GgTexture()	272
8.16.2.3 GgTexture() [2/2]	272
8.16.3 関数詳解	272
8.16.3.1 bind()	272
8.16.3.2 getHeight()	272
8.16.3.3 getSize() [1/2]	273
8.16.3.4 getSize() [2/2]	273
8.16.3.5 getTexture()	273
8.16.3.6 getWidth()	274
8.16.3.7 operator=()	274
8.16.3.8 unbind()	274
8.17 gg::GgTrackball クラス	275
8.17.1 詳解	276
8.17.2 構築子と解体子	276
8.17.2.1 GgTrackball()	276
8.17.2.2 ~GgTrackball()	277
8.17.3 関数詳解	277
8.17.3.1 begin()	277
8.17.3.2 end()	278
8.17.3.3 get()	278
8.17.3.4 getMatrix()	279
8.17.3.5 getQuaternion()	279
8.17.3.6 getScale() [1/3]	279
8.17.3.7 getScale() [2/3]	279
8.17.3.8 getScale() [3/3]	280
8.17.3.9 getStart() [1/3]	280
8.17.3.10 getStart() [2/3]	280
8.17.3.11 getStart() [3/3]	281
8.17.3.12 motion()	281
8.17.3.13 operator=()	282
8.17.3.14 region() [1/2]	282
8.17.3.15 region() [2/2]	283
8.17.3.16 reset()	283
8.17.3.17 rotate()	284
8.18 gg::GgTriangles クラス	284
8.18.1 詳解	285
8.18.2 構築子と解体子	286
8.18.2.1 GgTriangles() [1/2]	286
8.18.2.2 GgTriangles() [2/2]	286
8.18.2.3 ~GgTriangles()	286
8.18.3 関数詳解	287
8.18.3.1 draw()	287

8.18.3.2 getBuffer()	287
8.18.3.3 getCount()	288
8.18.3.4 load()	288
8.18.3.5 send()	288
8.19 gg::GgUniformBuffer< T > クラステンプレート	289
8.19.1 詳解	289
8.19.2 構築子と解体子	290
8.19.2.1 GgUniformBuffer() [1/3]	290
8.19.2.2 GgUniformBuffer() [2/3]	290
8.19.2.3 GgUniformBuffer() [3/3]	290
8.19.2.4 ~GgUniformBuffer()	291
8.19.3 関数詳解	291
8.19.3.1 bind()	291
8.19.3.2 copy()	291
8.19.3.3 fill()	292
8.19.3.4 getBuffer()	292
8.19.3.5 getCount()	292
8.19.3.6 getStride()	293
8.19.3.7 getTarget()	293
8.19.3.8 load() [1/2]	293
8.19.3.9 load() [2/2]	294
8.19.3.10 map() [1/2]	294
8.19.3.11 map() [2/2]	294
8.19.3.12 read()	295
8.19.3.13 send()	295
8.19.3.14 unbind()	296
8.19.3.15 unmap()	296
8.20 gg::GgVector クラス	296
8.20.1 詳解	297
8.20.2 構築子と解体子	298
8.20.2.1 GgVector() [1/4]	298
8.20.2.2 GgVector() [2/4]	298
8.20.2.3 GgVector() [3/4]	298
8.20.2.4 GgVector() [4/4]	299
8.20.3 関数詳解	299
8.20.3.1 distance3()	299
8.20.3.2 distance4()	300
8.20.3.3 dot3()	300
8.20.3.4 dot4()	301
8.20.3.5 length3()	301
8.20.3.6 length4()	302
8.20.3.7 normalize3()	302

8.20.3.8 normalize4()	303
8.20.3.9 operator*() [1/2]	303
8.20.3.10 operator*() [2/2]	304
8.20.3.11 operator*=(()) [1/2]	304
8.20.3.12 operator*=(()) [2/2]	305
8.20.3.13 operator+() [1/2]	305
8.20.3.14 operator+() [2/2]	305
8.20.3.15 operator+=() [1/2]	306
8.20.3.16 operator+=() [2/2]	306
8.20.3.17 operator-() [1/2]	307
8.20.3.18 operator-() [2/2]	307
8.20.3.19 operator-=() [1/2]	307
8.20.3.20 operator-=() [2/2]	308
8.20.3.21 operator/() [1/2]	308
8.20.3.22 operator/() [2/2]	309
8.20.3.23 operator/=(()) [1/2]	309
8.20.3.24 operator/=(()) [2/2]	309
8.21 gg::GgVertex 構造体	310
8.21.1 詳解	311
8.21.2 構築子と解体子	311
8.21.2.1 GgVertex() [1/4]	311
8.21.2.2 GgVertex() [2/4]	311
8.21.2.3 GgVertex() [3/4]	311
8.21.2.4 GgVertex() [4/4]	312
8.21.3 メンバ詳解	312
8.21.3.1 normal	312
8.21.3.2 position	313
8.22 gg::GgSimpleShader::Light 構造体	313
8.22.1 詳解	314
8.22.2 メンバ詳解	314
8.22.2.1 ambient	314
8.22.2.2 diffuse	314
8.22.2.3 position	314
8.22.2.4 specular	314
8.23 gg::GgSimpleShader::LightBuffer クラス	315
8.23.1 詳解	316
8.23.2 構築子と解体子	316
8.23.2.1 LightBuffer() [1/2]	316
8.23.2.2 LightBuffer() [2/2]	316
8.23.2.3 ~LightBuffer()	317
8.23.3 関数詳解	317
8.23.3.1 load() [1/2]	317

8.23.3.2 load() [2/2]	317
8.23.3.3 loadAmbient() [1/3]	318
8.23.3.4 loadAmbient() [2/3]	318
8.23.3.5 loadAmbient() [3/3]	319
8.23.3.6 loadColor()	319
8.23.3.7 loadDiffuse() [1/3]	320
8.23.3.8 loadDiffuse() [2/3]	320
8.23.3.9 loadDiffuse() [3/3]	320
8.23.3.10 loadPosition() [1/4]	321
8.23.3.11 loadPosition() [2/4]	321
8.23.3.12 loadPosition() [3/4]	322
8.23.3.13 loadPosition() [4/4]	322
8.23.3.14 loadSpecular() [1/3]	323
8.23.3.15 loadSpecular() [2/3]	323
8.23.3.16 loadSpecular() [3/3]	323
8.23.3.17 select()	324
8.24 gg::GgSimpleShader::Material 構造体	324
8.24.1 詳解	325
8.24.2 メンバ詳解	325
8.24.2.1 ambient	325
8.24.2.2 diffuse	326
8.24.2.3 shininess	326
8.24.2.4 specular	326
8.25 gg::GgSimpleShader::MaterialBuffer クラス	326
8.25.1 詳解	327
8.25.2 構築子と解体子	327
8.25.2.1 MaterialBuffer() [1/2]	327
8.25.2.2 MaterialBuffer() [2/2]	328
8.25.2.3 ~MaterialBuffer()	328
8.25.3 関数詳解	328
8.25.3.1 load() [1/2]	328
8.25.3.2 load() [2/2]	329
8.25.3.3 loadAmbient() [1/2]	329
8.25.3.4 loadAmbient() [2/2]	330
8.25.3.5 loadAmbientAndDiffuse() [1/2]	330
8.25.3.6 loadAmbientAndDiffuse() [2/2]	330
8.25.3.7 loadDiffuse() [1/2]	331
8.25.3.8 loadDiffuse() [2/2]	331
8.25.3.9 loadShininess() [1/2]	332
8.25.3.10 loadShininess() [2/2]	332
8.25.3.11 loadSpecular() [1/2]	333
8.25.3.12 loadSpecular() [2/2]	333

8.25.3.13 select()	334
8.26 Menu クラス	334
8.26.1 詳解	334
8.26.2 構築子と解体子	334
8.26.2.1 Menu() [1/2]	335
8.26.2.2 Menu() [2/2]	335
8.26.2.3 ~Menu()	335
8.26.3 関数詳解	335
8.26.3.1 draw()	335
8.26.3.2 operator=()	335
8.26.4 フレンドと関連関数の詳解	335
8.26.4.1 Draw	336
8.27 Window クラス	336
8.27.1 詳解	337
8.27.2 構築子と解体子	337
8.27.2.1 Window() [1/2]	337
8.27.2.2 Window() [2/2]	338
8.27.2.3 ~Window()	338
8.27.3 関数詳解	338
8.27.3.1 get()	339
8.27.3.2 getAltArrowX()	339
8.27.3.3 getAltArrowY()	339
8.27.3.4 getAltArrow()	339
8.27.3.5 getArrow() [1/2]	340
8.27.3.6 getArrow() [2/2]	340
8.27.3.7 getArrowX()	341
8.27.3.8 getArrowY()	341
8.27.3.9 getAspect()	341
8.27.3.10 getControlArrow()	342
8.27.3.11 getControlArrowX()	342
8.27.3.12 getControlArrowY()	342
8.27.3.13 getFboHeight()	343
8.27.3.14 getFboSize() [1/2]	343
8.27.3.15 getFboSize() [2/2]	343
8.27.3.16 getFboWidth()	343
8.27.3.17 getHeight()	344
8.27.3.18 getKey()	344
8.27.3.19 getLastKey()	344
8.27.3.20 getMouse() [1/3]	345
8.27.3.21 getMouse() [2/3]	345
8.27.3.22 getMouse() [3/3]	345
8.27.3.23 getMouseX()	346

8.27.3.24 getMouseY()	346
8.27.3.25 getRotation()	346
8.27.3.26 getRotationMatrix()	347
8.27.3.27 getScrollMatrix()	347
8.27.3.28 getShiftArrow()	347
8.27.3.29 getShiftArrowX()	348
8.27.3.30 getShiftArrowY()	348
8.27.3.31 getSize() [1/2]	348
8.27.3.32 getSize() [2/2]	348
8.27.3.33 getTranslation()	349
8.27.3.34 getTranslationMatrix()	349
8.27.3.35 getUserPointer()	350
8.27.3.36 getWheel() [1/3]	350
8.27.3.37 getWheel() [2/3]	350
8.27.3.38 getWheel() [3/3]	350
8.27.3.39 getWheelX()	351
8.27.3.40 getWheelY()	351
8.27.3.41 getWidth()	351
8.27.3.42 initialize()	351
8.27.3.43 operator bool()	352
8.27.3.44 operator=()	352
8.27.3.45 reset()	352
8.27.3.46 resetRotation()	353
8.27.3.47 resetTranslation()	353
8.27.3.48 restoreViewport()	353
8.27.3.49 selectInterface()	353
8.27.3.50 setClose()	353
8.27.3.51 setKeyboardFunc()	355
8.27.3.52 setMenubarHeight()	355
8.27.3.53 setMouseFunc()	355
8.27.3.54 setResizeFunc()	356
8.27.3.55 setUserPointer()	356
8.27.3.56 setVelocity()	356
8.27.3.57 setWheelFunc()	357
8.27.3.58 shouldClose()	357
8.27.3.59 swapBuffers()	357
9 ファイル詳解	359
9.1 Config.cpp ファイル	359
9.1.1 詳解	359
9.1.2 変数詳解	360
9.1.2.1 defaultLight	360

9.2 Config.h ファイル	360
9.2.1 詳解	361
9.2.2 マクロ定義詳解	362
9.2.2.1 TCharToUtf8	362
9.2.2.2 Utf8ToTChar	362
9.2.3 型定義詳解	362
9.2.3.1 pathChar	362
9.2.3.2 pathString	362
9.3 Draw.cpp ファイル	363
9.3.1 詳解	363
9.4 Draw.h ファイル	363
9.4.1 詳解	364
9.5 gg.cpp ファイル	365
9.5.1 詳解	365
9.6 gg.h ファイル	365
9.6.1 詳解	369
9.6.2 マクロ定義詳解	370
9.6.2.1 ggError	370
9.6.2.2 ggFBOError	370
9.7 GgApp.h ファイル	370
9.7.1 マクロ定義詳解	371
9.7.1.1 GG_USE_IMGUI	371
9.8 ggsample01.cpp ファイル	372
9.8.1 マクロ定義詳解	372
9.8.1.1 CONFIG_FILE	372
9.8.1.2 PROJECT_NAME	372
9.9 main.cpp ファイル	373
9.9.1 マクロ定義詳解	373
9.9.1.1 HEADER_STR	373
9.9.2 関数詳解	373
9.9.2.1 main()	374
9.10 Menu.cpp ファイル	374
9.10.1 詳解	375
9.10.2 変数詳解	375
9.10.2.1 menuFont	375
9.10.2.2 menuFontSize	375
9.11 Menu.h ファイル	375
9.11.1 詳解	376
9.12 parseconfig.h ファイル	376
9.12.1 詳解	377
9.12.2 関数詳解	378
9.12.2.1 getString() [1/3]	378

9.12.2.2 getString() [2/3]	378
9.12.2.3 getString() [3/3]	379
9.12.2.4 getValue() [1/2]	379
9.12.2.5 getValue() [2/2]	380
9.12.2.6 getVector()	380
9.12.2.7 setString() [1/3]	381
9.12.2.8 setString() [2/3]	381
9.12.2.9 setString() [3/3]	382
9.12.2.10 setValue() [1/2]	382
9.12.2.11 setValue() [2/2]	383
9.12.2.12 setVector()	383
9.13 README.md ファイル	384
9.14 Window.h ファイル	384
9.14.1 詳解	385
9.14.2 マクロ定義詳解	386
9.14.2.1 GG_BUTTON_COUNT	386
9.14.2.2 GG_INTERFACE_COUNT	386
Index	387

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) 2011-2022 Kohe Tokoi. All Rights Reserved.

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

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

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

Chapter 2

ggsample01

2.1 ゲームグラフィックス特論A 第1回 宿題

OpenGL の開発環境を整備してください。

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

2.2 宿題プログラムの作成に必要な環境

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

2.3 補足

このプログラムを実行すると、次のような図形が表示されます。

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

2.4 宿題プログラム用補助プログラムについて

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

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

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

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

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

2.4.1 補助プログラムのドキュメント

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

2.4.2 補助プログラムの使い方

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

```
#include "Window.h"
int main()
{
    Window::init();
    Window window;
    while (window)
    {
        // 
        // ここで OpenGL による描画を行う
        //
    }
}
```

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

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

2.4.3 Oculus Rift を使う場合

#include "Window.h" の前に #define USE_OOCULUS_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_OOCULUS_RIFT
#include "Window.h"
```

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

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

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

2.4.4 Dear ImGui を使う場合

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

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

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

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

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

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

```
// ウィンドウ関連の処理
#define USE_IMGUI
#include "Window.h"
int main()
{
    // ウィンドウ関連の初期設定
    Window::init(4, 1);
    // ウィンドウを作成する
    Window window("Window Title", 1280, 720);
    // ImGui の初期設定
    ImGui::StyleColorsDark();
    // 背景色を指定する
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f);
    // ウィンドウが開いている間繰り返す
    while (window)
    {
        // ImGui のフレームを準備する
        ImGui::NewFrame();
        // ImGui のフレームに一つ目の ImGui のウィンドウを描く
        ImGui::Begin("Control panel");
        ImGui::Text("Frame rate: %6.2f fps", ImGui::GetIO().Framerate);
        if (ImGui::Button("Quit")) window.setClose();
        ImGui::End();
        // ImGui のフレームに描画する
        ImGui::Render();
        // ウィンドウを消去する
        glClear(GL_COLOR_BUFFER_BIT);
        //
        // ここで OpenGL による描画を行う
        //
        // カラーバッファを入れ替えてイベントを取り出す
        window.swapBuffers();
    }
}
```

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

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

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

2.4.4.1 imconfig.h の変更点

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

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

Chapter 3

名前空間索引

3.1 名前空間一覧

全名前空間の一覧です。

[gg](#) 15

Chapter 4

階層索引

4.1 クラス階層

クラス階層一覧です。大雑把に文字符号順で並べられています。

std::array	107
gg::GgMatrix	296
gg::GgVector	180
gg::GgQuaternion	275
gg::GgTrackball	
Config	87
Draw	90
GgApp	92
gg::GgBuffer< T >	93
gg::GgColorTexture	99
gg::GgNormalTexture	163
gg::GgPointShader	171
gg::GgSimpleShader	256
gg::GgShader	247
gg::GgShape	250
gg::GgPoints	167
gg::GgTriangles	284
gg::GgElements	103
gg::GgSimpleObj	253
gg::GgTexture	270
gg::GgUniformBuffer< T >	289
gg::GgUniformBuffer< Light >	289
gg::GgSimpleShader::LightBuffer	315
gg::GgUniformBuffer< Material >	289
gg::GgSimpleShader::MaterialBuffer	326
gg::GgVertex	310
gg::GgSimpleShader::Light	313
gg::GgSimpleShader::Material	324
Menu	334
Window	336

Chapter 5

クラス索引

5.1 クラス一覧

クラス・構造体・共用体・インターフェースの一覧です。

Config	87
Draw	90
GgApp	92
gg::GgBuffer< T >	93
gg::GgColorTexture	99
gg::GgElements	103
gg::GgMatrix	107
gg::GgNormalTexture	163
gg::GgPoints	167
gg::GgPointShader	171
gg::GgQuaternion	180
gg::GgShader	247
gg::GgShape	250
gg::GgSimpleObj	253
gg::GgSimpleShader	256
gg::GgTexture	270
gg::GgTrackball	275
gg::GgTriangles	284
gg::GgUniformBuffer< T >	289
gg::GgVector	296
gg::GgVertex	310
gg::GgSimpleShader::Light	313
gg::GgSimpleShader::LightBuffer	315
gg::GgSimpleShader::Material	324
gg::GgSimpleShader::MaterialBuffer	326
Menu	334
Window	336

Chapter 6

ファイル索引

6.1 ファイル一覧

ファイル一覧です。

Config.cpp	359
Config.h	360
Draw.cpp	363
Draw.h	363
gg.cpp	365
gg.h	365
GgApp.h	370
ggsample01.cpp	372
main.cpp	373
Menu.cpp	374
Menu.h	375
parseconfig.h	376
Window.h	384

Chapter 7

名前空間詳解

7.1 gg 名前空間

クラス

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

列挙型

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

関数

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

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

- std::shared_ptr< GgElements > ggElementsSphere (GLfloat radius=1.0f, int slices=16, int stacks=8)
- bool ggLoadSimpleObj (const std::string &name, std::vector< std::array< GLuint, 3 >> &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, bool normalize=false)
- bool ggLoadSimpleObj (const std::string &name, std::vector< std::array< GLuint, 3 >> &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)

変数

- GLint ggBufferAlignment
使用している GPU のバッファアライメント.

7.1.1 詳解

ゲームグラフィックス特論の宿題用補助プログラムの名前空間

7.1.2 列挙型詳解

7.1.2.1 BindingPoints

```
enum gg::BindingPoints
```

光源と材質の uniform buffer object の結合ポイント.

列挙値

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

gg.h の 1336 行目に定義があります。

7.1.3 関数詳解

7.1.3.1 _ggError()

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

OpenGL のエラーをチェックする.

引数

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

覚え書き

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

gg.cpp の 2562 行目に定義があります。

7.1.3.2 `ggFBOError()`

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

FBO のエラーをチェックする。

引数

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

覚え書き

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

gg.cpp の 2606 行目に定義があります。

7.1.3.3 `ggArraysObj()`

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

Wavefront OBJ ファイルを読み込む (Arrays 形式)

引数

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

戻り値

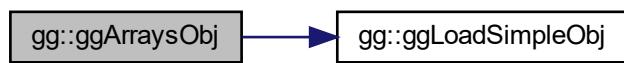
`GgTriangles` 型の `shared_ptr`.

覚え書き

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

`gg.cpp` の 5185 行目に定義があります。

呼び出し関係図:



7.1.3.4 `ggConjugate()`

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

共役四元数を返す。

引数

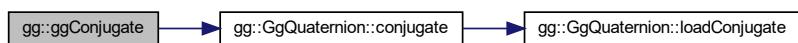
<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

四元数 `q` の共役四元数。

`gg.h` の 4731 行目に定義があります。

呼び出し関係図:



7.1.3.5 ggCreateComputeShader()

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

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

引数

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0)。

gg.cpp の 4958 行目に定義があります。

7.1.3.6 ggCreateNormalMap()

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

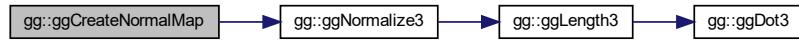
グレースケール画像(8bit)から法線マップのデータを作成する。

引数

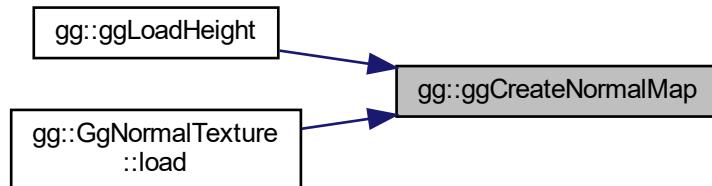
<i>hmap</i>	グレースケール画像のデータ.
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数.
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数.
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_RGBA).
<i>nz</i>	法線の z 成分の割合.
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット.
<i>nmap</i>	法線マップを格納する vector.

gg.cpp の 3788 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.7 ggCreateShader()

```

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

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

引数

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ0).

gg.cpp の 4780 行目に定義があります。

7.1.3.8 ggCross() [1/2]

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

GgVector 型の 3 要素の外積.

引数

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

戻り値

a と *b* の外積.

覚え書き

戻り値の w(第4) 要素は 0.

gg.h の 1975 行目に定義があります。

呼び出し関係図:



7.1.3.9 ggCross() [2/2]

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

3 要素の外積.

引数

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

gg.h の 1416 行目に定義があります。

呼び出し関係図:



7.1.3.10 ggDistance3() [1/2]

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

GgVector 型の 3 要素の距離.

引数

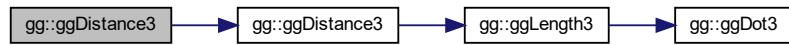
<i>a</i>	GgVector 型のベクトル.
<i>b</i>	GgVector 型のベクトル.

戻り値

a と *b* の距離.

gg.h の 2012 行目に定義があります。

呼び出し関係図:



7.1.3.11 ggDistance3() [2/2]

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

3 要素の距離.

引数

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

戻り値

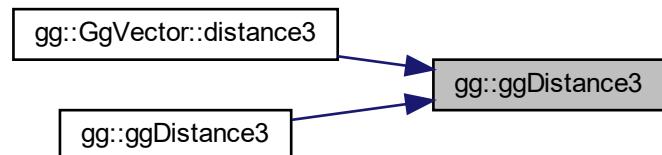
a と *b* の距離.

gg.h の 1453 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.12 ggDistance4() [1/2]

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

GgVector 型の 4 要素の距離.

引数

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

戻り値

2 つの GgVector *a, b* の距離.

gg.h の 2078 行目に定義があります。

呼び出し関係図:



7.1.3.13 ggDistance4() [2/2]

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

GLfloat 型の 4 要素の距離.

引数

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

戻り値

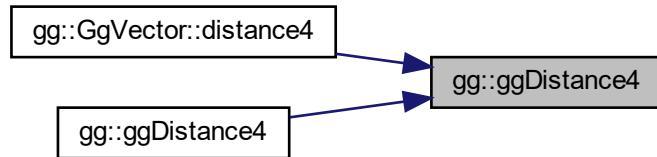
`a` と `b` のそれぞれの 4 要素の距離.

`gg.h` の 1516 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.14 `ggDot3()` [1/2]

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

`GgVector` 型の 3 要素の内積.

引数

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

戻り値

`a` と `b` のそれぞれの 3 要素の内積.

gg.h の 1989 行目に定義がります。

呼び出し関係図:



7.1.3.15 ggDot3() [2/2]

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

3 要素の内積.

引数

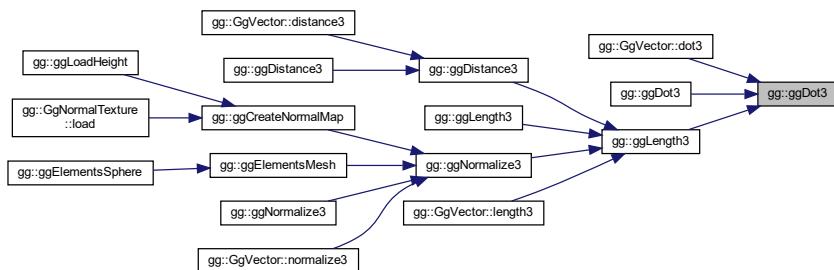
<i>a</i>	GLfloat 型の 3 要素の配列変数.
<i>b</i>	GLfloat 型の 3 要素の配列変数.

戻り値

a と *b* のそれぞれの 3 要素の内積.

gg.h の 1430 行目に定義がります。

被呼び出し関係図:



7.1.3.16 ggDot4() [1/2]

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

GgVector 型の 4 要素の内積.

引数

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

戻り値

a と *b* のそれぞれの 4 要素の内積.

gg.h の 2055 行目に定義があります。

呼び出し関係図:



7.1.3.17 ggDot4() [2/2]

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

GLfloat 型の 4 要素の内積

引数

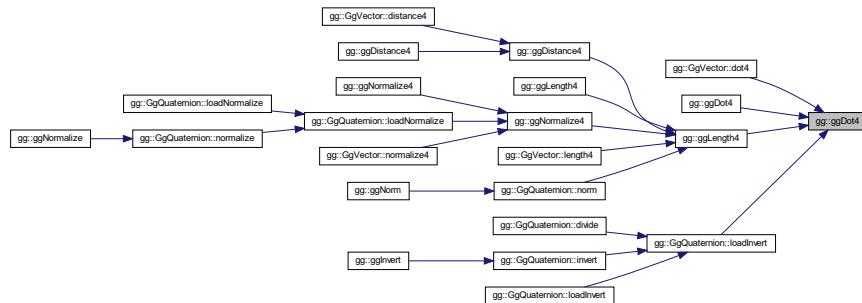
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	GLfloat 型の 4 要素の配列変数.

戻り値

a と *b* のそれぞれの 4 要素の内積.

gg.h の 1493 行目に定義があります。

被呼び出し関係図:



7.1.3.18 ggElementsMesh()

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

メッシュ形状を作成する (Elements 形式).

引数

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

戻り値

GgElements 型の shared_ptr.

覚え書き

メッシュ状に GgElements 形式の三角形データを生成する.

gg.cpp の 5219 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.19 ggElementsObj()

```

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

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

引数

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

戻り値

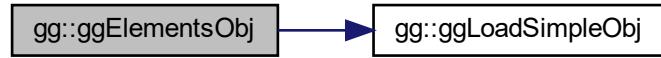
GgElements 型の shared_ptr.

覚え書き

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

gg.cpp の 5201 行目に定義があります。

呼び出し関係図:



7.1.3.20 ggElementsSphere()

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

球状に三角形データを生成する (Elements 形式).

引数

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

戻り値

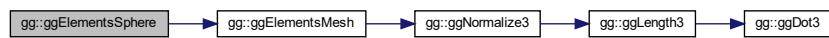
[GgElements](#) 型の shared_ptr.

覚え書き

球状に [GgElements](#) 形式の三角形データを生成する.

gg.cpp の 5314 行目に定義があります。

呼び出し関係図:



7.1.3.21 ggEllipse()

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

橢円状に三角形を生成する。

引数

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

戻り値

GgTriangles 型の shared_ptr.

gg.cpp の 5160 行目に定義があります。

7.1.3.22 ggEulerQuaternion() [1/2]

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

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

引数

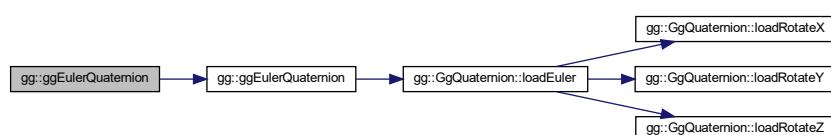
<i>e</i>	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転を表す四元数。

gg.h の 4645 行目に定義があります。

呼び出し関係図:



7.1.3.23 ggEulerQuaternion() [2/2]

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

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

引数

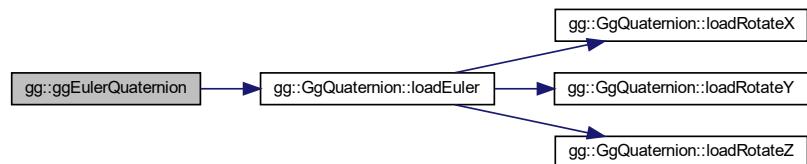
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転を表す四元数.

gg.h の 4633 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.24 ggFrustum()

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

透視透視投影変換行列を返す。

引数

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

戻り値

求めた透視投影変換行列。

gg.h の 3344 行目に定義があります。

呼び出し関係図:



7.1.3.25 ggIdentity()

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

単位行列を返す。

戻り値

単位行列.

gg.h の 3069 行目に定義がります。

呼び出し関係図:



7.1.3.26 ggIdentityQuaternion()

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

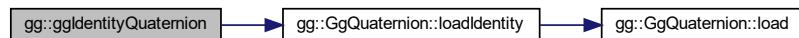
単位四元数を返す.

戻り値

単位四元数.

gg.h の 4531 行目に定義がります。

呼び出し関係図:



7.1.3.27 ggInit()

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

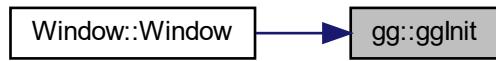
ゲームグラフィックス特論の都合にもとづく初期化を行う。

覚え書き

WindowsにおいてOpenGL 1.2以降のAPIを有効化する。

gg.cpp の 1307 行目に定義があります。

呼び出し関係図:



7.1.3.28 ggInvert() [1/2]

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

逆行列を返す。

引数

<i>m</i>	元の変換行列。
----------	---------

戻り値

m の逆行列。

gg.h の 3389 行目に定義があります。

呼び出し関係図:



7.1.3.29 ggInvert() [2/2]

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

四元数の逆元を求める。

引数

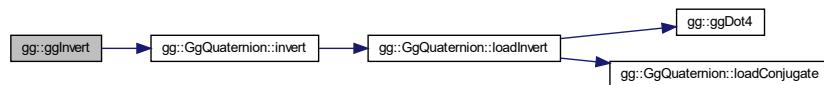
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

四元数 *q* の逆元。

gg.h の 4742 行目に定義があります。

呼び出し関係図:



7.1.3.30 ggLength3() [1/2]

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

GgVector 型の 3 要素の長さ。

引数

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

戻り値

a の 3 要素の長さ。

gg.h の 2000 行目に定義があります。

呼び出し関係図:



7.1.3.31 ggLength3() [2/2]

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

3要素の長さ。

引数

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

戻り値

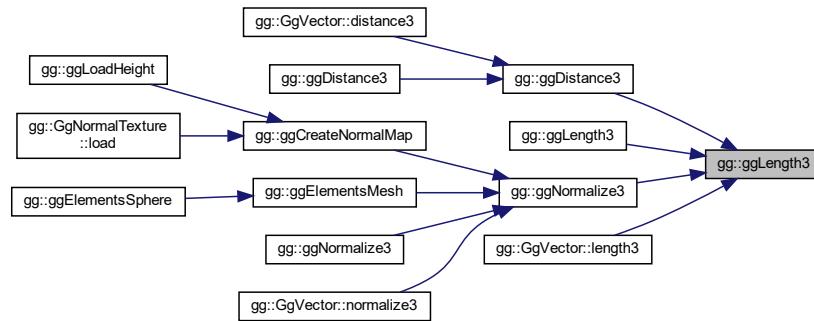
a の 3 要素の長さ。

gg.h の 1441 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.32 `ggLength4()` [1/2]

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

`GgVector` 型の 4 要素の長さ.

引数

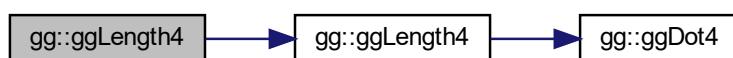
a	<code>GgVector</code> 型の変数.
---	-----------------------------

戻り値

a の 4 要素の長さ.

`gg.h` の 2066 行目に定義があります。

呼び出し関係図:



7.1.3.33 ggLength4() [2/2]

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

GLfloat 型の 4 要素の長さ.

引数

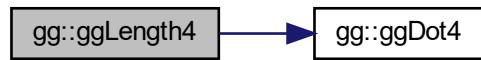
a	GLfloat 型の 4 要素の配列変数.
---	-----------------------

戻り値

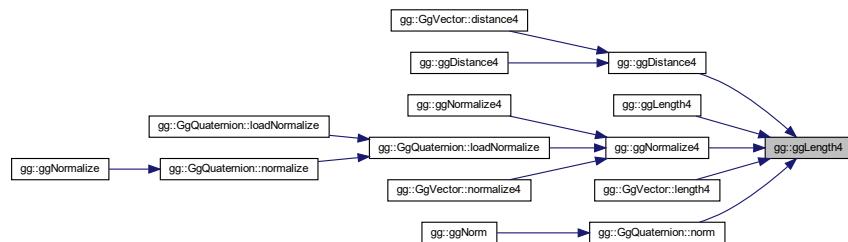
a の 4 要素の長さ.

gg.h の 1504 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.34 ggLoadComputeShader()

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

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

引数

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

戻り値

プログラムオブジェクトのプログラム名(作成できなければ 0).

gg.cpp の 4999 行目に定義があります。

7.1.3.35 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

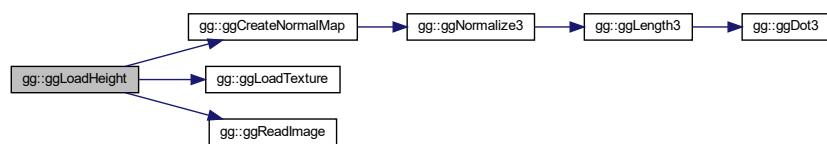
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

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 3873 行目に定義があります。

呼び出し関係図:



7.1.3.36 ggLoadImage()

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

テクスチャメモリを確保して TGA 画像ファイルを読み込む。

引数

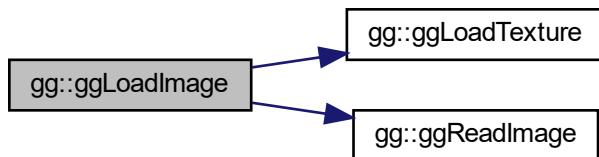
<i>name</i>	読み込むファイル名。
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)。
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる。
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

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 3740 行目に定義があります。

呼び出し関係図:



7.1.3.37 ggLoadShader() [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

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

戻り値

プログラムオブジェクトのプログラム名 (作成できなければ 0).

gg.h の 5131 行目に定義があります。

呼び出し関係図:



7.1.3.38 ggLoadShader() [2/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.

引数

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

戻り値

プログラムオブジェクトのプログラム名 (作成できなければ 0).

gg.cpp の 4925 行目に定義があります。

被呼び出し関係図:



7.1.3.39 ggLoadSimpleObj() [1/2]

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

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

引数

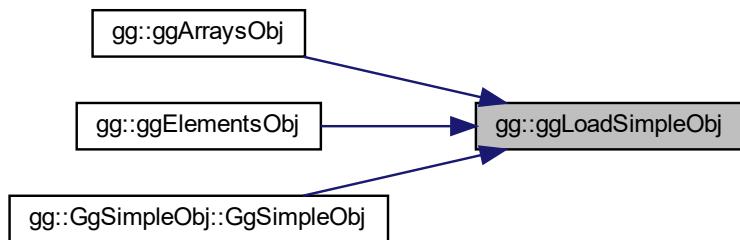
<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの GgSimpleShader::Material 型の材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>normalize</i>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

gg.cpp の 4532 行目に定義があります。

被呼び出し関係図:



7.1.3.40 ggLoadSimpleObj() [2/2]

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

三角形分割された OBJ ファイルを読み込む (Elements 形式).

引数

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

戻り値

ファイルの読み込みに成功したら true.

gg.cpp の 4621 行目に定義があります。

7.1.3.41 ggLoadTexture()

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

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

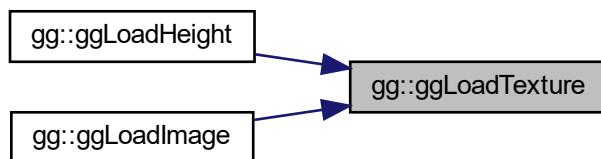
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャメモリの確保のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <code>image</code> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <code>image</code> の縦の画素数.
<i>format</i>	<code>image</code> のフォーマット.
<i>type</i>	<code>image</code> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</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 3696 行目に定義があります。

被呼び出し関係図:



7.1.3.42 ggLookat() [1/3]

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

ビュー変換行列を返す。

引数

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

戻り値

求めたビュー変換行列.

gg.h の 3304 行目に定義があります。

呼び出し関係図:



7.1.3.43 ggLookat() [2/3]

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

ビュー変換行列を返す.

引数

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

戻り値

求めたビュー変換行列.

gg.h の 3286 行目に定義があります。

呼び出し関係図:



7.1.3.44 ggLookat() [3/3]

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

ビュー変換行列を返す。

引数

<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

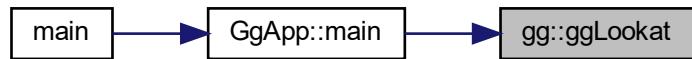
求めたビュー変換行列.

gg.h の 3268 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.45 ggMatrixQuaternion() [1/2]

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

回転の変換行列 m を表す四元数を返す。

引数

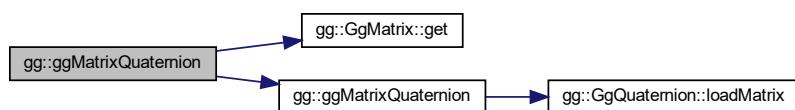
m	GgMatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4555 行目に定義があります。

呼び出し関係図:



7.1.3.46 ggMatrixQuaternion() [2/2]

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

回転の変換行列 m を表す四元数を返す。

引数

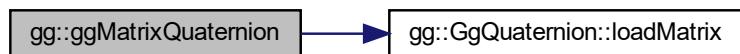
a	GLfloat 型の 16 要素の配列変数。
---	------------------------

戻り値

a による回転の変換に相当する四元数。

gg.h の 4543 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.47 ggNorm()

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

四元数のノルムを返す。

引数

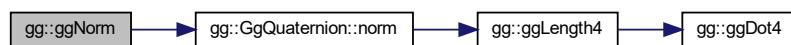
<code>q</code>	<code>GgQuaternion</code> 型の四元数.
----------------	----------------------------------

戻り値

四元数 `q` のノルム.

`gg.h` の 4709 行目に定義がります。

呼び出し関係図:



7.1.3.48 `ggNormal()`

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

法線変換行列を返す.

引数

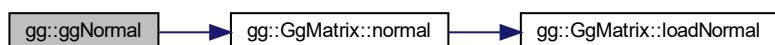
<code>m</code>	元の変換行列.
----------------	---------

戻り値

`m` の法線変換行列.

`gg.h` の 3400 行目に定義がります。

呼び出し関係図:



7.1.3.49 ggNormalize()

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

正規化した四元数を返す.

引数

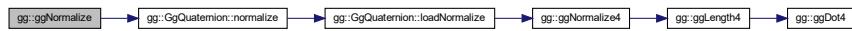
<i>q</i>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

四元数 *q* を正規化した四元数.

gg.h の 4720 行目に定義があります。

呼び出し関係図:



7.1.3.50 ggNormalize3() [1/4]

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

GgVector 型の 3 要素の正規化.

引数

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

戻り値

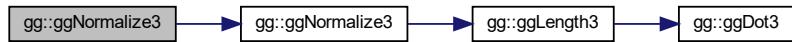
a の 3 要素を正規化したもの.

覚え書き

戻り値の w (第4) 要素は 0 になる.

gg.h の 2026 行目に定義があります。

呼び出し関係図:



7.1.3.51 ggNormalize3() [2/4]

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

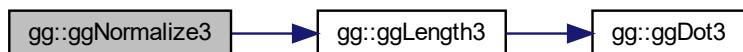
3要素の正規化。

引数

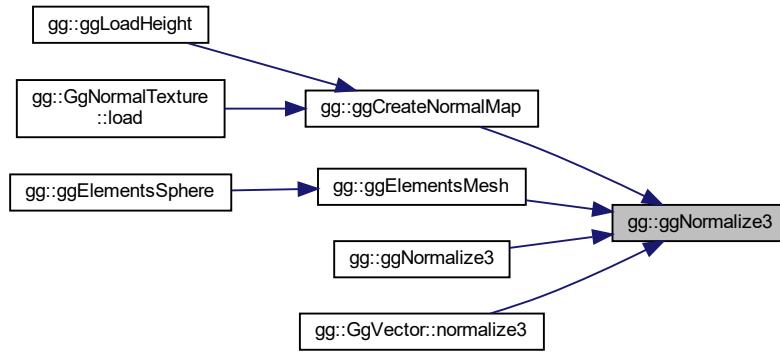
<i>a</i>	GLfloat 型の 3 要素の配列変数。
<i>b</i>	<i>a</i> の 3 要素を正規化した結果を格納する GLfloat 型の 3 要素の配列変数。

gg.h の 1465 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.52 `ggNormalize3()` [3/4]

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

`GgVector` 型の 3 要素の正規化。

引数

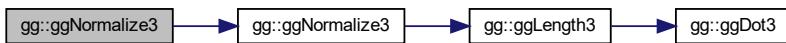
<code>a</code>	<code>GgVector</code> 型の変数のポインタ。
----------------	----------------------------------

覚え書き

`a` の `w` (第4) 要素は 0 になる。

`gg.h` の 2042 行目に定義があります。

呼び出し関係図:



7.1.3.53 ggNormalize3() [4/4]

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

3要素の正規化。

引数

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

gg.h の 1478 行目に定義があります。

呼び出し関係図:



7.1.3.54 ggNormalize4() [1/4]

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

GgVector型の4要素の正規化。

引数

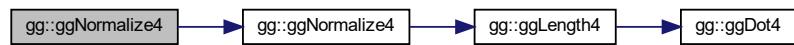
<i>a</i>	GgVector型の変数。
----------	---------------

戻り値

a の 4 要素を正規化した結果。

gg.h の 2089 行目に定義があります。

呼び出し関係図:



7.1.3.55 ggNormalize4() [2/4]

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

GLfloat 型の 4 要素の正規化。

引数

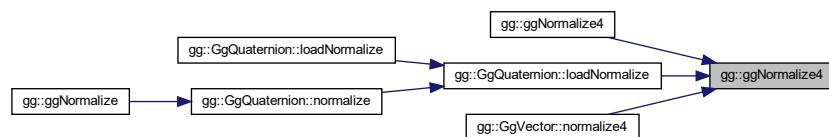
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	<i>a</i> を正規化した結果を格納する GLfloat 型の 4 要素の配列変数.

gg.h の 1528 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.56 ggNormalize4() [3/4]

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

GgVector 型の 4 要素の正規化。

引数

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

gg.h の 2101 行目に定義があります。

呼び出し関係図:



7.1.3.57 ggNormalize4() [4/4]

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

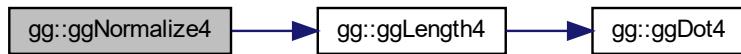
GLfloat 型の 4 要素の正規化。

引数

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

gg.h の 1542 行目に定義があります。

呼び出し関係図:



7.1.3.58 ggOrthogonal()

```
GgMatrix gg::ggOrthogonal (
    GLfloat left,
    GLfloat right,
```

```
GLfloat bottom,
GLfloat top,
GLfloat zNear,
GLfloat zFar ) [inline]
```

直交投影変換行列を返す。

引数

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

戻り値

求めた直交投影変換行列。

gg.h の 3325 行目に定義があります。

呼び出し関係図:



7.1.3.59 ggPerspective()

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

画角を指定して透視投影変換行列を返す。

引数

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

戻り値

求めた透視投影変換行列.

gg.h の 3363 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.60 ggPointsCube()

```

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

点群を立方体状に生成する.

引数

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

戻り値

`GgPoints` 型の `shared_ptr`.

`gg.cpp` の 5086 行目に定義がります。

7.1.3.61 ggPointsSphere()

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

点群を球状に生成する。

引数

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

戻り値

`GgPoints` 型の `shared_ptr`.

`gg.cpp` の 5112 行目に定義がります。

7.1.3.62 ggQuaternion() [1/2]

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

四元数を返す。

引数

<code>a</code>	<code>GLfloat</code> 型の <code>GLfloat</code> 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 4521 行目に定義がります。

呼び出し関係図:



7.1.3.63 ggQuaternion() [2/2]

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

四元数を返す.

引数

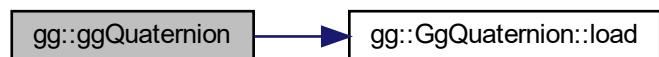
<i>x</i>	四元数の <i>x</i> 要素.
<i>y</i>	四元数の <i>y</i> 要素.
<i>z</i>	四元数の <i>z</i> 要素.
<i>w</i>	四元数の <i>w</i> 要素.

戻り値

四元数.

gg.h の 4509 行目に定義がります。

呼び出し関係図:



呼び出し関係図:



7.1.3.64 ggQuaternionMatrix()

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

四元数 q の回転の変換行列を返す。

引数

q	元の四元数。
-----	--------

戻り値

四元数 q が表す回転に相当する `GgMatrix` 型の変換行列。

gg.h の 4566 行目に定義があります。

呼び出し関係図:



7.1.3.65 ggQuaternionTransposeMatrix()

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

四元数 q の回転の転置した変換行列を返す。

引数

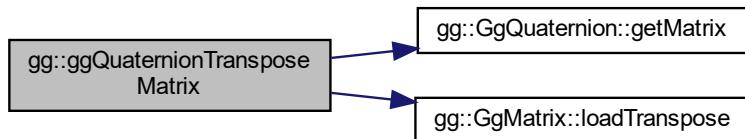
<i>q</i>	元の四元数.
----------	--------

戻り値

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

gg.h の 4579 行目に定義があります。

呼び出し関係図:



7.1.3.66 ggReadImage()

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

TGA ファイル (8/16/24/32bit) をメモリに読み込む.

引数

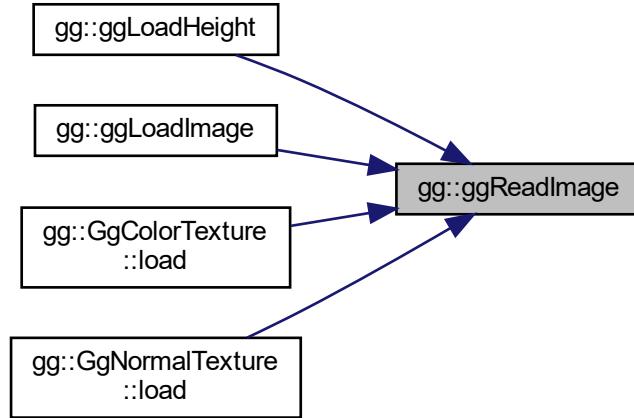
<i>name</i>	読み込むファイル名.
<i>image</i>	読み込んだデータを格納する vector.
<i>pWidth</i>	読み込んだ画像の横の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない.
<i>pHeight</i>	読み込んだ画像の縦の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない.
<i>pFormat</i>	読み込んだファイルの書式 (GL_RED, G_RG, GL_RGB, G_RGBA) の格納先のポインタ, <code>nullptr</code> なら格納しない.

戻り値

読み込みに成功すれば `true`, 失敗すれば `false`.

gg.cpp の 3576 行目に定義がります。

被呼び出し関係図:



7.1.3.67 `ggRectangle()`

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

矩形状に 2 枚の三角形を生成する。

引数

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

戻り値

`GgTriangles` 型の `shared_ptr`.

gg.cpp の 5139 行目に定義がります。

7.1.3.68 `ggRotate()` [1/5]

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

`r` 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<code>r</code>	回転軸のベクトルと回転角を表す <code>GgVector</code> 型の変数.
----------------	---

戻り値

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

gg.h の 3248 行目に定義があります。

呼び出し関係図:



7.1.3.69 ggRotate() [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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

戻り値

r を軸に a だけ回転する変換行列.

gg.h の 3224 行目に定義があります。

呼び出し関係図:



7.1.3.70 ggRotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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 3236 行目に定義があります。

呼び出し関係図:



7.1.3.71 ggRotate() [4/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<i>r</i>	回転軸のベクトルを表す 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 3211 行目に定義があります。

呼び出し関係図:



7.1.3.72 ggRotate() [5/5]

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

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

引数

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

戻り値

(*x*, *y*, *z*) を軸にさらに *a* 回転する変換行列.

gg.h の 3198 行目に定義があります。

呼び出し関係図:



7.1.3.73 ggRotateQuaternion() [1/3]

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

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

引数

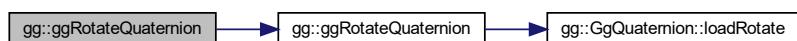
v	軸ベクトルを表す GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

回転を表す四元数.

gg.h の 4620 行目に定義があります。

呼び出し関係図:



7.1.3.74 ggRotateQuaternion() [2/3]

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

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

引数

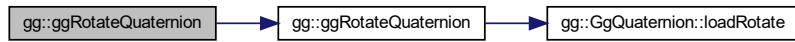
<i>v</i>	軸ベクトルを表す GLfloat 型の 3 要素の配列変数.
<i>a</i>	回転角.

戻り値

回転を表す四元数.

gg.h の 4609 行目に定義があります。

呼び出し関係図:



7.1.3.75 ggRotateQuaternion() [3/3]

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

(x, y, z) を軸として角度 a 回転する四元数を返す

引数

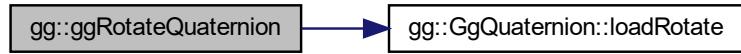
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

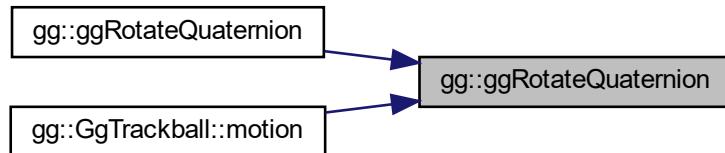
回転を表す四元数.

gg.h の 4596 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.76 ggRotateX()

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

x 軸中心の回転の変換行列を返す。

引数

a	回転角.
---	------

戻り値

x 軸中心に a だけ回転する変換行列。

gg.h の 3159 行目に定義があります。

呼び出し関係図:



7.1.3.77 ggRotateY()

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

y 軸中心の回転の変換行列を返す。

引数

a	回転角。
---	------

戻り値

y 軸中心に a だけ回転する変換行列。

gg.h の 3171 行目に定義があります。

呼び出し関係図:



7.1.3.78 ggRotateZ()

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

z 軸中心の回転の変換行列を返す。

引数

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 3183 行目に定義があります。

呼び出し関係図:



7.1.3.79 ggSaveColor()

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

カラー バッファの内容を TGA ファイルに保存する.

引数

name	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 3520 行目に定義があります。

呼び出し関係図:



7.1.3.80 ggSaveDepth()

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

デプスバッファの内容を TGA ファイルに保存する.

引数

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

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 3546 行目に定義があります。

呼び出し関係図:



7.1.3.81 ggSaveTga()

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

配列の内容を TGA ファイルに保存する.

引数

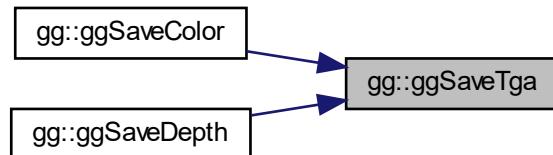
<i>name</i>	保存するファイル名.
<i>buffer</i>	画像データを格納した配列.
<i>width</i>	画像の横の画素数.
<i>height</i>	画像の縦の画素数.
<i>depth</i>	1 画素のバイト数.

戻り値

保存に成功すれば `true`, 失敗すれば `false`.

`gg.cpp` の 3430 行目に定義があります。

呼び出し関係図:



7.1.3.82 `ggScale()` [1/3]

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

拡大縮小の変換行列を返す.

引数

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

戻り値

拡大縮小の変換行列.

`gg.h` の 3147 行目に定義があります。

呼び出し関係図:



7.1.3.83 **ggScale()** [2/3]

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

拡大縮小の変換行列を返す.

引数

<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

拡大縮小の変換行列.

gg.h の 3135 行目に定義があります。

呼び出し関係図:

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

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

拡大縮小の変換行列を返す.

引数

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

戻り値

拡大縮小の変換行列.

gg.h の 3123 行目に定義があります。

呼び出し関係図:



7.1.3.85 ggSlerp() [1/4]

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

二つの四元数の球面線形補間の結果を返す.

引数

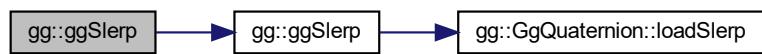
<i>q</i>	GgQuaternion 型の四元数.
<i>r</i>	GgQuaternion 型の四元数.
<i>t</i>	補間パラメータ.

戻り値

q, r を *t* で内分した四元数.

gg.h の 4672 行目に定義があります。

呼び出し関係図:



7.1.3.86 ggSlerp() [2/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

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

戻り値

q, a を *t* で内分した四元数。

`gg.h` の 4685 行目に定義があります。

呼び出し関係図:



7.1.3.87 ggSlerp() [3/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

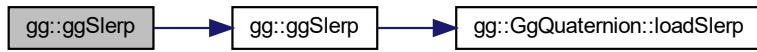
<i>a</i>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
<i>q</i>	<code>GgQuaternion</code> 型の四元数。
<i>t</i>	補間パラメータ。

戻り値

a, q を t で内分した四元数.

gg.h の 4698 行目に定義があります。

呼び出し関係図:



7.1.3.88 ggSlerp() [4/4]

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

二つの四元数の球面線形補間の結果を返す.

引数

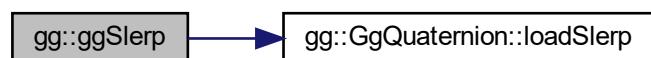
a	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
b	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数.
t	補間パラメータ.

戻り値

a, b を t で内分した四元数.

gg.h の 4658 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.89 ggTranslate() [1/3]

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

平行移動の変換行列を返す。

引数

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

戻り値

平行移動の変換行列。

gg.h の 3108 行目に定義があります。

呼び出し関係図:



7.1.3.90 ggTranslate() [2/3]

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

平行移動の変換行列を返す。

引数

<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 3096 行目に定義がります。

呼び出し関係図:



7.1.3.91 ggTranslate() [3/3]

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

平行移動の変換行列を返す.

引数

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

戻り値

平行移動の変換行列.

gg.h の 3084 行目に定義がります。

呼び出し関係図:



7.1.3.92 ggTranspose()

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

転置行列を返す。

引数

m	元の変換行列。
---	---------

戻り値

m の転置行列。

gg.h の 3378 行目に定義があります。

呼び出し関係図:



7.1.3.93 operator*()

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

スカラーに GgVector 型の各要素を乗じた積を返す。

引数

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

戻り値

a に *b* の各要素を乗じた積.

gg.h の 1948 行目に定義があります。

7.1.3.94 operator+() [1/2]

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

何もしない.

引数

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

戻り値

v の値.

gg.h の 1901 行目に定義があります。

7.1.3.95 operator+() [2/2]

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

スカラーに GgVector 型の各要素を足した和を返す.

引数

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

戻り値

a に b の各要素を足した和.

gg.h の 1913 行目に定義があります。

7.1.3.96 operator-() [1/2]

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

符号の反転.

引数

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

戻り値

v の値の符号を反転した結果.

gg.h の 1924 行目に定義があります。

7.1.3.97 operator-() [2/2]

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

スカラーから GgVector 型の各要素を引いた差を返す.

引数

a	GLfloat 型の値.
b	GgVector 型のベクトル.

戻り値

a から b の各要素を引いた差.

gg.h の 1936 行目に定義があります。

7.1.3.98 operator/()

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

スカラーを `GgVector` 型の各要素で割った商を返す.

引数

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

戻り値

`a` を `b` の各要素で割った商.

`gg.h` の 1960 行目に定義があります。

7.1.4 変数詳解

7.1.4.1 ggBufferAlignment

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

使用している GPU のバッファアライメント.

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

Chapter 8

クラス詳解

8.1 Config クラス

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

公開メンバ関数

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

フレンド

- class `Menu`

8.1.1 詳解

構成データ

`Config.h` の 40 行目に定義があります。

8.1.2 構築子と解体子

8.1.2.1 `Config()` [1/2]

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

コンストラクタ

`Config.cpp` の 25 行目に定義があります。

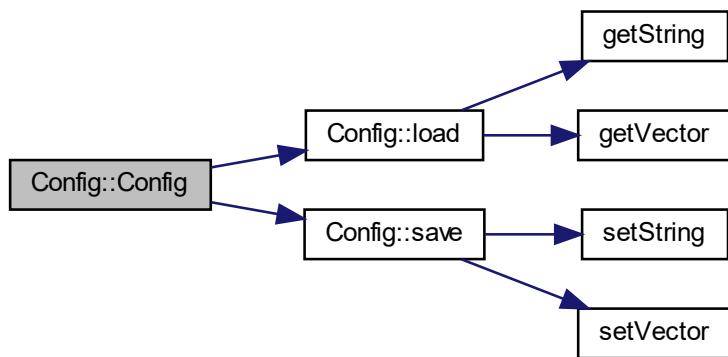
8.1.2.2 Config() [2/2]

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

ファイルから構成データを読み込むコンストラクタ

Config.cpp の 39 行目に定義がります。

呼び出し関係図:



8.1.2.3 ~Config()

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

デストラクタ

Config.cpp の 49 行目に定義がります。

8.1.3 関数詳解

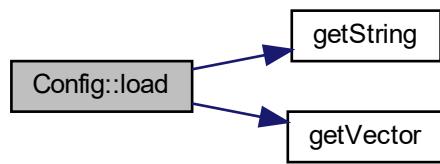
8.1.3.1 load()

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

設定ファイルを読み込む

Config.cpp の 56 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



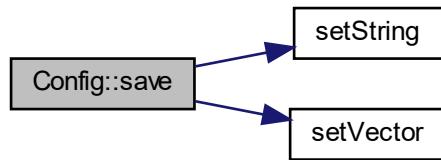
8.1.3.2 save()

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

設定ファイルを書き出す

Config.cpp の 97 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.1.4 フレンドと関連関数の詳解

8.1.4.1 Menu

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

`Config.h` の 43 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

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

8.2 Draw クラス

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

公開メンバ関数

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

8.2.1 詳解

図形の描画クラス

Draw.h の 17 行目に定義がります。

8.2.2 構築子と解体子

8.2.2.1 Draw()

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

コンストラクタ

Draw.cpp の 13 行目に定義がります。

8.2.2.2 ~Draw()

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

デストラクタ

Draw.cpp の 29 行目に定義がります。

8.2.3 関数詳解

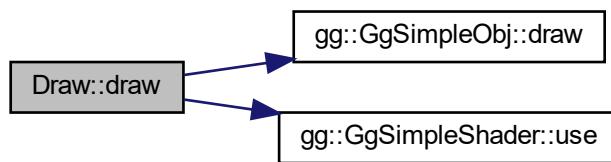
8.2.3.1 draw()

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

描画する

Draw.cpp の 36 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.3 GgApp クラス

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

公開メンバ関数

- int [main](#) (int argc, const char *const *argv)

8.3.1 詳解

GgApp.h の 15 行目に定義があります。

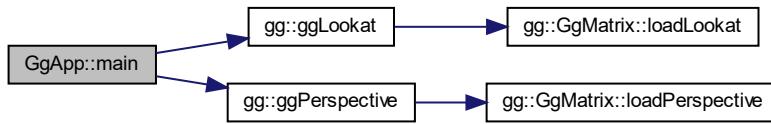
8.3.2 関数詳解

8.3.2.1 main()

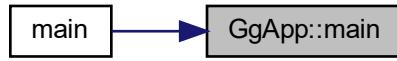
```
int GgApp::main (
    int argc,
    const char *const * argv )
```

ggsample01.cpp の 28 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



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](#)

8.4 gg::GgBuffer< T > クラステンプレート

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

公開 メンバ関数

- [GgBuffer \(GLenum target, const T *data, GLsizei stride, GLsizei count, GLenum usage\)](#)
- virtual [~GgBuffer \(\)](#)
- [GgBuffer \(const GgBuffer< T > &o\)=delete](#)
- [GgBuffer< T > & operator= \(const GgBuffer< T > &o\)=delete](#)
- const GLuint & [getTarget \(\) const](#)
- GLsizeiptr [getStride \(\) const](#)
- const GLsizei & [getCount \(\) const](#)

- const GLuint & `getBuffer () const`
- void `bind () const`
- void `unbind () const`
- void * `map () const`
- void * `map (GLint first, GLsizei count) const`
- void `unmap () const`
- void `send (const T *data, GLint first, GLsizei count) const`
- void `read (T *data, GLint first, GLsizei count) const`
- void `copy (GLuint src_buffer, GLint src_first=0, GLint dst_first=0, GLsizei count=0) const`

8.4.1 詳解

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

バッファオブジェクト.

覚え書き

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

gg.h の 5504 行目に定義があります。

8.4.2 構築子と解体子

8.4.2.1 GgBuffer() [1/2]

```
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

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

gg.h の 5516 行目に定義がります。

8.4.2.2 ~GgBuffer()

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

デストラクタ。

gg.h の 5516 行目に定義がります。

8.4.2.3 GgBuffer() [2/2]

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

コピーコンストラクタは使用禁止。

8.4.3 関数詳解

8.4.3.1 bind()

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

バッファオブジェクトを結合する。

gg.h の 5609 行目に定義がります。

8.4.3.2 copy()

```
template<typename T >
void gg::GgBuffer< T >::copy (
    GLuint src_buffer,
    GLint src_first = 0,
    GLint dst_first = 0,
    GLsizei count = 0 ) const [inline]
```

別のバッファオブジェクトからデータを複写する。

引数

<i>src_buffer</i>	複写元のバッファオブジェクト名.
<i>src_first</i>	複写元 (<i>src_buffer</i>) の先頭のデータの位置.
<i>dst_first</i>	複写先 (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 5714 行目に定義があります。

8.4.3.3 [getBuffer\(\)](#)

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

戻り値

このバッファオブジェクト名.

gg.h の 5601 行目に定義があります。

8.4.3.4 [getCount\(\)](#)

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

戻り値

このバッファオブジェクトが保持するデータの数.

gg.h の 5591 行目に定義があります。

8.4.3.5 [getStride\(\)](#)

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

戻り値

このバッファオブジェクトのデータの間隔.

gg.h の 5581 行目に定義があります。

8.4.3.6 getTarget()

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

バッファオブジェクトのターゲットを取り出す。

戻り値

このバッファオブジェクトのターゲット。

gg.h の 5571 行目に定義があります。

8.4.3.7 map() [1/2]

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

バッファオブジェクトをマップする。

戻り値

マップしたメモリの先頭のポインタ。

gg.h の 5627 行目に定義があります。

8.4.3.8 map() [2/2]

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

バッファオブジェクトの指定した範囲をマップする。

引数

<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 5644 行目に定義があります。

8.4.3.9 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.

8.4.3.10 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 5687 行目に定義があります。

8.4.3.11 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 5669 行目に定義があります。

8.4.3.12 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 5617 行目に定義があります。

8.4.3.13 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 5657 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.5 gg::GgColorTexture クラス

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

公開メンバ関数

- [GgColorTexture \(\)](#)
- [GgColorTexture \(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
- [GgColorTexture \(const std::string &name, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
- virtual [~GgColorTexture \(\)](#)
- void [load \(const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)
- void [load \(const std::string &name, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE\)](#)

8.5.1 詳解

カラーマップ。

覚え書き

カラー画像を読み込んでテクスチャを作成する。

gg.h の 5296 行目に定義があります。

8.5.2 構築子と解体子

8.5.2.1 GgColorTexture() [1/3]

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

コンストラクタ.

gg.h の 5306 行目に定義があります。

8.5.2.2 GgColorTexture() [2/3]

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

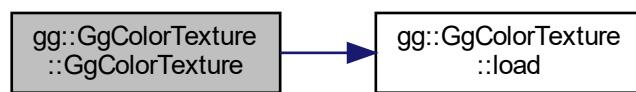
メモリ上のデータからテクスチャを作成するコンストラクタ.

引数

<i>image</i>	テクスチャとして用いる画像データ, nullptr ならデータを読み込まない.
<i>width</i>	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 5321 行目に定義があります。

呼び出し関係図:



8.5.2.3 GgColorTexture() [3/3]

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

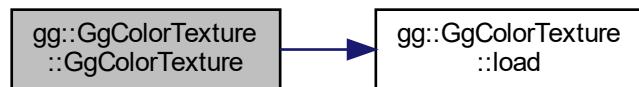
ファイルからデータを読み込んでテクスチャを作成するコンストラクタ。

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 5341 行目に定義があります。

呼び出し関係図:



8.5.2.4 ~GgColorTexture()

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

デストラクタ。

gg.h の 5349 行目に定義があります。

8.5.3 関数詳解

8.5.3.1 `load()` [1/2]

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

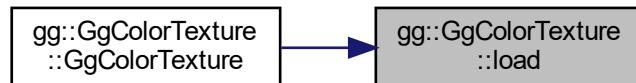
テクスチャを作成してメモリ上のデータを読み込む。

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 5364 行目に定義があります。

被呼び出し関係図:



8.5.3.2 `load()` [2/2]

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

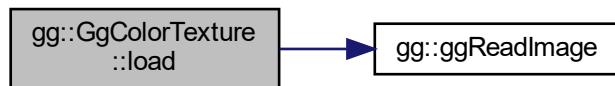
テクスチャを作成してファイルからデータを読み込む。

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 3917 行目に定義があります。

呼び出し関係図:



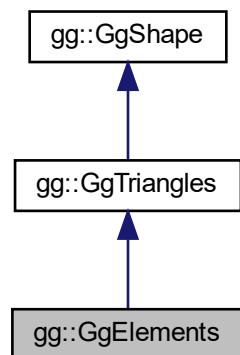
このクラス詳解は次のファイルから抽出されました:

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

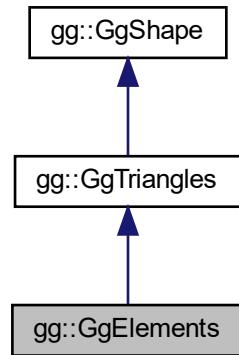
8.6 gg::GgElements クラス

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

gg::GgElements の継承関係図



gg::GgElements 連携図



公開 メンバ関数

- `GgElements (GLenum mode=GL_TRIANGLES)`
- `GgElements (const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum mode=GL_TRIANGLES, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~GgElements ()`
- `const GLsizei & getIndexCount () const`
- `const GLuint & getIndexBuffer () const`
- `void send (const GgVertex *vert, GLuint firstv, GLsizei countv, const GLuint *face=nullptr, GLuint firstf=0, GLsizei countf=0) const`
- `void load (const GgVertex *vert, GLsizei countv, const GLuint *face, GLsizei countf, GLenum usage=GL_STATIC_DRAW)`
- `virtual void draw (GLint first=0, GLsizei count=0) const`

8.6.1 詳解

三角形で表した形状データ (Elements 形式).

gg.h の 6412 行目に定義があります。

8.6.2 構築子と解体子**8.6.2.1 GgElements() [1/2]**

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

コンストラクタ.

引数

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

gg.h の 6425 行目に定義がります。

8.6.2.2 GgElements() [2/2]

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

コンストラクタ.

引数

<i>vert</i>	この図形の頂点属性の配列 (nullptr ならデータを転送しない).
<i>countv</i>	頂点数.
<i>face</i>	三角形の頂点インデックス.
<i>countf</i>	三角形の頂点数.
<i>mode</i>	描画する基本図形の種類.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6440 行目に定義がります。

8.6.2.3 ~GgElements()

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

gg.h の 6456 行目に定義がります。

8.6.3 関数詳解

8.6.3.1 draw()

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

インデックスを使った三角形の描画.

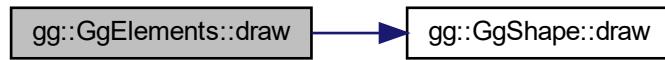
引数

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

[gg::GgTriangles](#)を再実装しています。

gg.cpp の 5073 行目に定義があります。

呼び出し関係図:



8.6.3.2 getIndexBuffer()

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

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

戻り値

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

gg.h の 6474 行目に定義があります。

8.6.3.3 getIndexCount()

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

データの数を取り出す.

戻り値

この図形の三角形数.

gg.h の 6464 行目に定義があります。

8.6.3.4 load()

```

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

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

引数

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

gg.h の 6511 行目に定義があります。

8.6.3.5 send()

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

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

引数

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

gg.h の 6489 行目に定義があります。

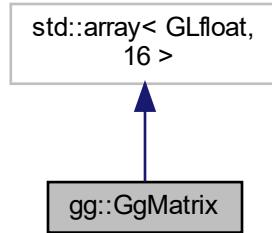
このクラス詳解は次のファイルから抽出されました:

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

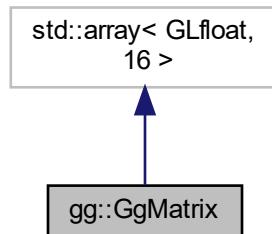
8.7 gg::GgMatrix クラス

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

gg::GgMatrix の継承関係図



gg::GgMatrix 連携図



公開メンバ関数

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

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

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

8.7.1 詳解

変換行列。

`gg.h` の 2109 行目に定義があります。

8.7.2 構築子と解体子

8.7.2.1 `GgMatrix()` [1/4]

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

コンストラクタ。

`gg.h` の 2122 行目に定義があります。

8.7.2.2 `GgMatrix()` [2/4]

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

コンストラクタ。

引数

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

gg.h の 2146 行目に定義がります。

8.7.2.3 GgMatrix() [3/4]

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

コンストラクタ.

引数

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

gg.h の 2161 行目に定義がります。

8.7.2.4 GgMatrix() [4/4]

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

コンストラクタ.

引数

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

gg.h の 2171 行目に定義があります。

呼び出し関係図:



8.7.3 関数詳解

8.7.3.1 frustum()

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

透視投影変換を乗じた結果を返す。

引数

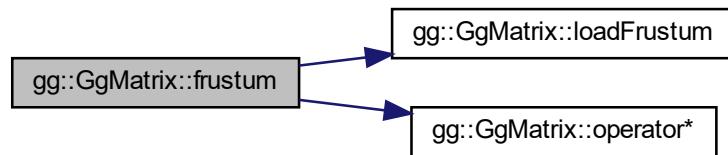
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2925 行目に定義があります。

呼び出し関係図:



8.7.3.2 `get()` [1/2]

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

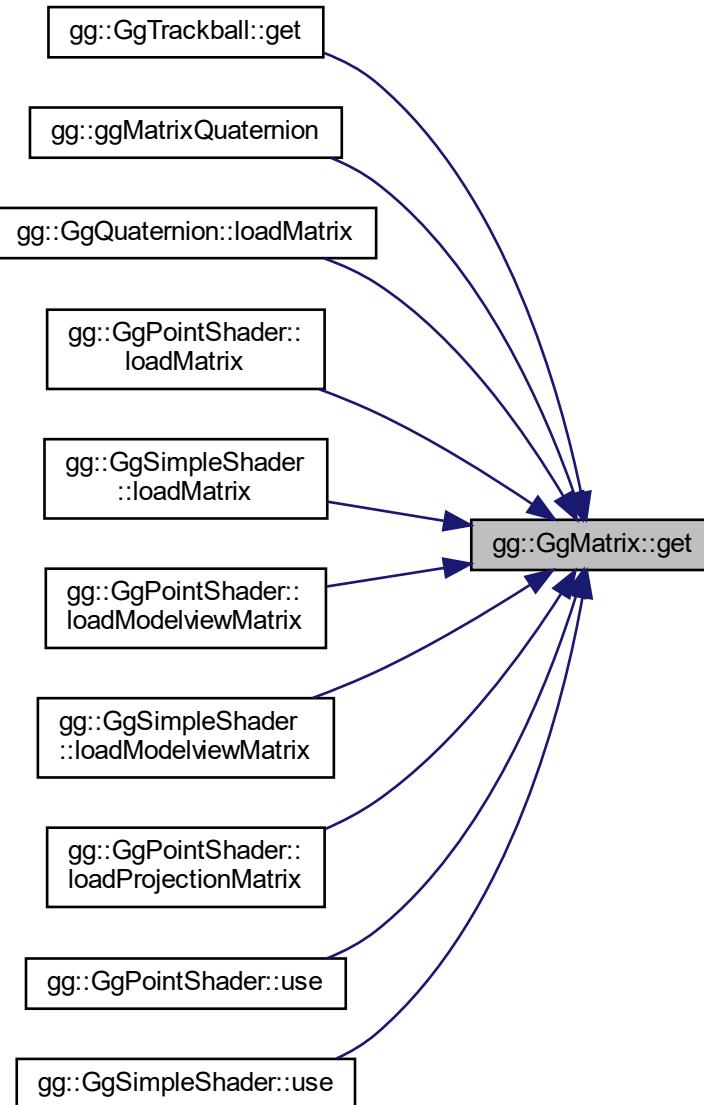
変換行列を取り出す。

戻り値

変換行列を格納した `GLfloat` 型の 16 要素の配列変数。

`gg.h` の 3048 行目に定義があります。

被呼び出し関係図:



8.7.3.3 `get()` [2/2]

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

変換行列を取り出す。

引数

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

gg.h の 3058 行目に定義があります。

8.7.3.4 invert()

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

逆行列を返す。

戻り値

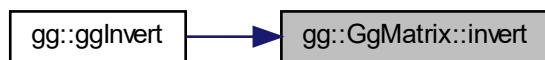
逆行列。

gg.h の 2969 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.5 loadFrustum()

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

透視透視投影変換行列を格納する。

引数

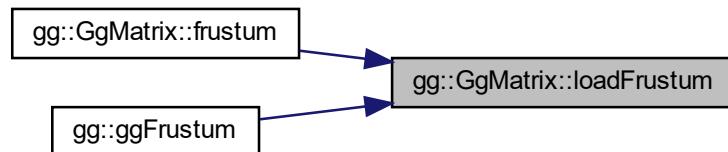
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 3034 行目に定義があります。

被呼び出し関係図:



8.7.3.6 loadIdentity()

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

単位行列を格納する.

gg.cpp の 2687 行目に定義があります。

被呼び出し関係図:



8.7.3.7 loadInvert() [1/2]

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

逆行列を格納する。

引数

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

戻り値

設定した `m` の逆行列。

`gg.h` の 2648 行目に定義があります。

呼び出し関係図:



8.7.3.8 loadInvert() [2/2]

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

逆行列を格納する。

引数

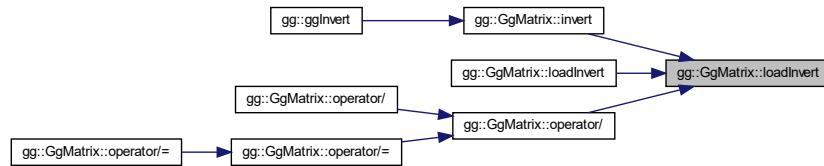
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列。
----------------	-------------------------------------

戻り値

設定した `a` の逆行列。

`gg.cpp` の 2844 行目に定義があります。

被呼び出し関係図:



8.7.3.9 loadLookat() [1/3]

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

ビュー変換行列を格納する。

引数

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

戻り値

設定したビュー変換行列.

gg.h の 2568 行目に定義があります。

呼び出し関係図:



8.7.3.10 loadLookat() [2/3]

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

ビュー変換行列を格納する。

引数

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

戻り値

設定したビュー変換行列。

gg.h の 2555 行目に定義があります。

呼び出し関係図:



8.7.3.11 loadLookat() [3/3]

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

ビュー変換行列を格納する。

引数

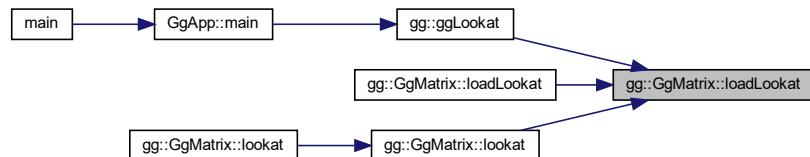
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列.

`gg.cpp` の 2946 行目に定義があります。

被呼び出し関係図:



8.7.3.12 `loadNormal()` [1/2]

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

法線変換行列を格納する.

引数

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

戻り値

設定した `m` の法線変換行列.

gg.h の 2667 行目に定義があります。

呼び出し関係図:



8.7.3.13 loadNormal() [2/2]

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

法線変換行列を格納する。

引数

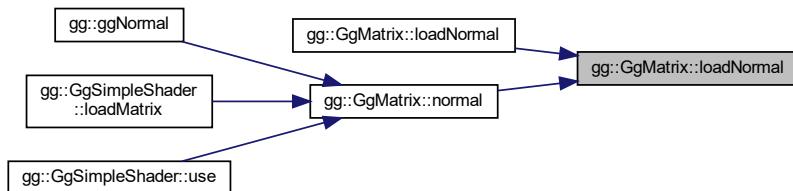
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

設定した m の法線変換行列。

gg.cpp の 2926 行目に定義があります。

被呼び出し関係図:



8.7.3.14 loadOrthogonal()

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

直交投影変換行列を格納する。

引数

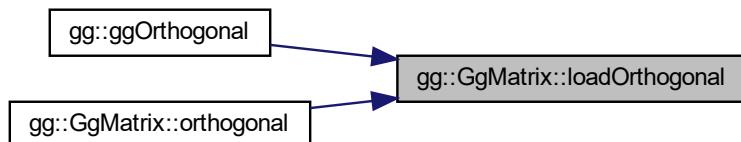
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した直交投影変換行列。

gg.cpp の 3004 行目に定義があります。

被呼び出し関係図:



8.7.3.15 loadPerspective()

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

画角を指定して透視投影変換行列を格納する。

引数

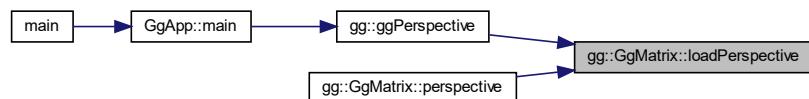
<i>fovy</i>	<i>y</i> 方向の画角.
<i>aspect</i>	縦横比.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

gg.cpp の 3064 行目に定義があります。

被呼び出し関係図:



8.7.3.16 loadRotate() [1/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2524 行目に定義があります。

呼び出し関係図:



8.7.3.17 loadRotate() [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

`gg.h` の 2502 行目に定義があります。

呼び出し関係図:



8.7.3.18 loadRotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2513 行目に定義があります。

呼び出し関係図:



8.7.3.19 loadRotate() [4/5]

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

r 方向のベクトルを軸とする回転の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2490 行目に定義があります。

呼び出し関係図:



8.7.3.20 loadRotate() [5/5]

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

(x, y, z) 方向のベクトルを軸とする回転の変換行列を格納する.

引数

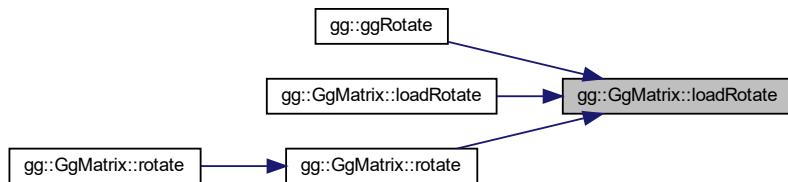
<i>x</i>	回転軸の x 成分.
<i>y</i>	回転軸の y 成分.
<i>z</i>	回転軸の z 成分.
<i>a</i>	回転角.

戻り値

設定した変換行列.

gg.cpp の 2780 行目に定義があります。

被呼び出し関係図:



8.7.3.21 loadRotateX()

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

x 軸中心の回転の変換行列を格納する。

引数

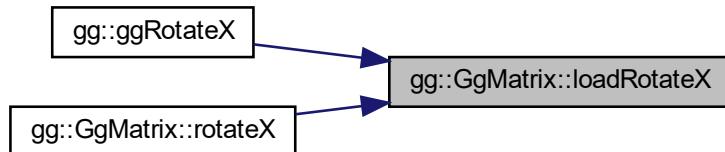
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2732 行目に定義があります。

被呼び出し関係図:



8.7.3.22 loadRotateY()

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

y 軸中心の回転の変換行列を格納する。

引数

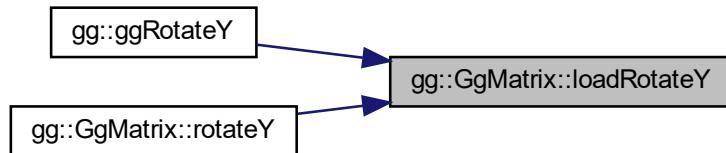
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2748 行目に定義があります。

被呼び出し関係図:



8.7.3.23 loadRotateZ()

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

z 軸中心の回転の変換行列を格納する。

引数

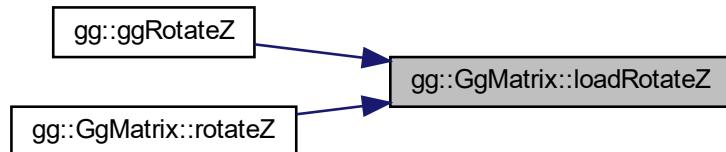
a	回転角。
---	------

戻り値

設定した変換行列。

gg.cpp の 2764 行目に定義があります。

被呼び出し関係図:



8.7.3.24 loadScale() [1/3]

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

拡大縮小の変換行列を格納する。

引数

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

戻り値

設定した変換行列。

gg.h の 2443 行目に定義があります。

呼び出し関係図:



8.7.3.25 loadScale() [2/3]

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

拡大縮小の変換行列を格納する。

引数

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 2432 行目に定義があります。

呼び出し関係図:



8.7.3.26 loadScale() [3/3]

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

拡大縮小の変換行列を格納する。

引数

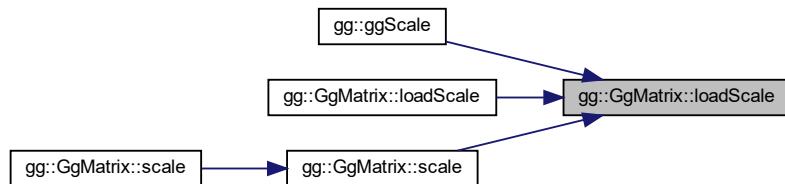
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 拡大率のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2716 行目に定義があります。

被呼び出し関係図:



8.7.3.27 loadTranslate() [1/3]

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

平行移動の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.h の 2410 行目に定義があります。

呼び出し関係図:



8.7.3.28 loadTranslate() [2/3]

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

平行移動の変換行列を格納する.

引数

<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 2399 行目に定義があります。

呼び出し関係図:



8.7.3.29 loadTranslate() [3/3]

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

平行移動の変換行列を格納する。

引数

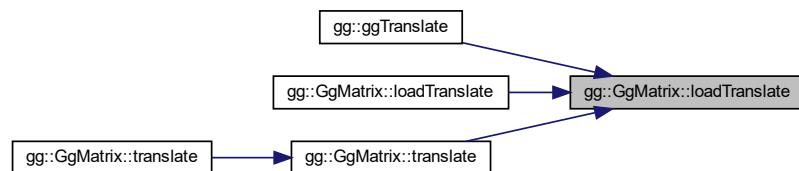
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

設定した変換行列.

gg.cpp の 2700 行目に定義があります。

被呼び出し関係図:



8.7.3.30 loadTranspose() [1/2]

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

転置行列を格納する。

引数

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

戻り値

設定した `m` の転置行列。

`gg.h` の 2629 行目に定義があります。

呼び出し関係図:



8.7.3.31 loadTranspose() [2/2]

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

転置行列を格納する。

引数

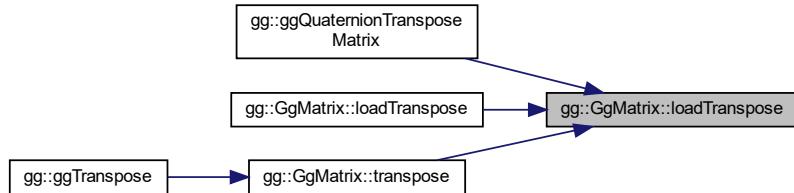
<code>a</code>	<code>GLfloat</code> 型の 16 要素の変換行列。
----------------	-------------------------------------

戻り値

設定した `a` の転置行列。

`gg.cpp` の 2819 行目に定義があります。

被呼び出し関係図:



8.7.3.32 lookat() [1/3]

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

ビュー変換を乗じた結果を返す.

引数

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

戻り値

ビュー変換行列を乗じた変換行列.

gg.h の 2888 行目に定義があります。

呼び出し関係図:



8.7.3.33 lookat() [2/3]

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

ビュー変換を乗じた結果を返す。

引数

<i>e</i>	視点の位置を格納した <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 2875 行目に定義があります。

呼び出し関係図:



8.7.3.34 lookat() [3/3]

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

ビュー変換を乗じた結果を返す。

引数

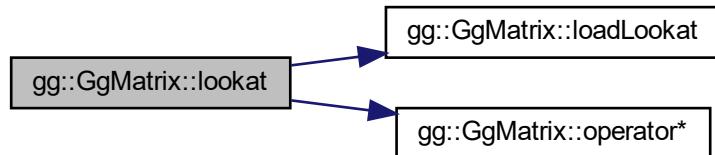
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

ビュー変換行列を乗じた変換行列.

`gg.h` の 2857 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.35 normal()

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

法線変換行列を返す。

戻り値

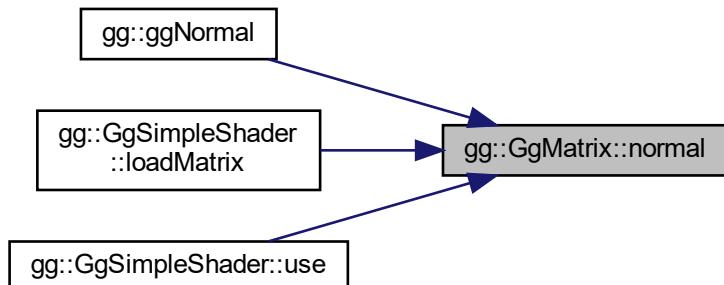
法線変換行列。

gg.h の 2980 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.36 operator*() [1/3]

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

変換行列に別の変換行列を乗算した値を返す。

引数

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

戻り値

変換行列に a を乗じた `GgMatrix` 型の変換行列.

gg.h の 2301 行目に定義があります。

呼び出し関係図:



8.7.3.37 operator*() [2/3]

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

ベクトルに対して投影変換を行う.

引数

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

戻り値

v 変換結果の `GgVector` 型のベクトル.

gg.h の 3036 行目に定義があります。

8.7.3.38 operator*() [3/3]

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

変換行列に配列に格納した変換行列を乗算した値を返す.

引数

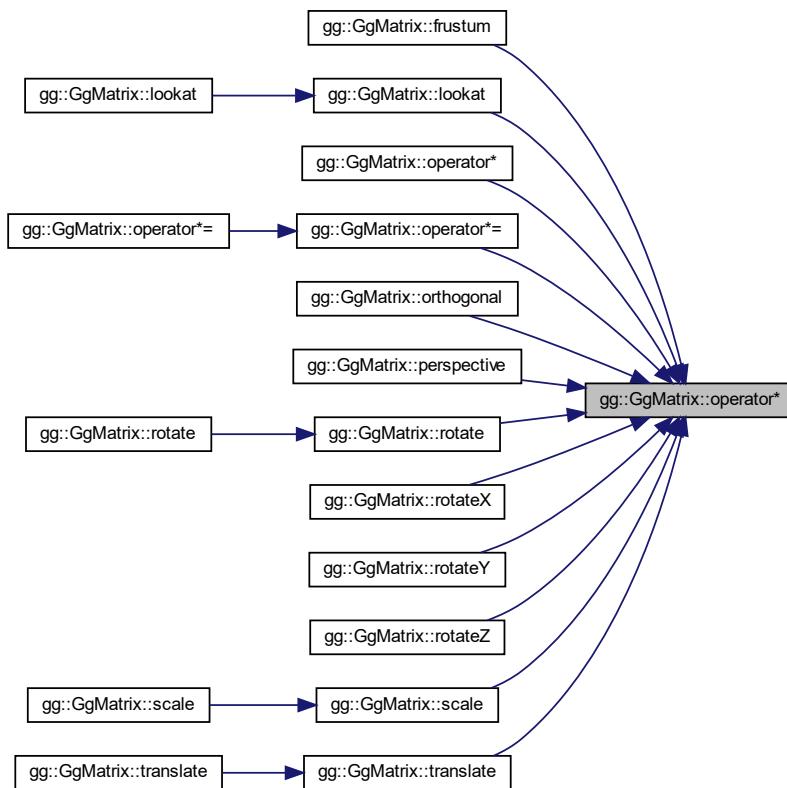
a	変換行列を格納した GLfloat 型の 16 要素の配列変数.
---	----------------------------------

戻り値

変換行列に a を乗じた GgMatrix 型の変換行列.

gg.h の 2288 行目に定義があります。

被呼び出し関係図:



8.7.3.39 operator*=() [1/2]

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

変換行列に別の変換行列を乗算した結果を格納する.

引数

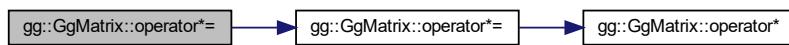
<code>m</code>	GgMatrix 型の変換行列.
----------------	------------------

戻り値

`m` を掛けた変換行列の参照.

gg.h の 2324 行目に定義があります。

呼び出し関係図:



8.7.3.40 operator*=() [2/2]

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

変換行列に配列に格納した変換行列を乗算した結果を格納する.

引数

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

戻り値

`a` を掛けた変換行列の参照.

gg.h の 2312 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.41 operator+() [1/2]

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

変換行列に別の変換行列を加算した値を返す.

引数

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

戻り値

変換行列に a を加えた GgMatrix 型の変換行列.

gg.h の 2207 行目に定義があります。

呼び出し関係図:



8.7.3.42 operator+() [2/2]

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

変換行列に配列に格納した変換行列を加算した値を返す.

引数

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

戻り値

変換行列に a を加えた GgMatrix 型の変換行列.

gg.h の 2194 行目に定義があります。

呼び出し関係図:



8.7.3.43 operator+=() [1/2]

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

変換行列に別の変換行列を加算した結果を格納する.

引数

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

戻り値

m を加えた変換行列の参照.

gg.h の 2230 行目に定義があります。

呼び出し関係図:



8.7.3.44 operator+=() [2/2]

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

変換行列に配列に格納した変換行列を加算した結果を格納する。

引数

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

戻り値

a を加えた変換行列の参照。

gg.h の 2218 行目に定義があります。

被呼び出し関係図:



8.7.3.45 operator-() [1/2]

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

変換行列から別の変換行列を減算した値を返す。

引数

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

戻り値

変換行列から m を引いた GgMatrix 型の変換行列。

gg.h の 2254 行目に定義があります。

呼び出し関係図:



8.7.3.46 operator-() [2/2]

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

変換行列から配列に格納した変換行列を減算した結果を格納する.

引数

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

戻り値

変換行列から *a* を引いた GgMatrix 型の変換行列.

gg.h の 2241 行目に定義があります。

被呼び出し関係図:



8.7.3.47 operator-() [1/2]

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

変換行列に別の変換行列を減算した結果を格納する.

引数

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

戻り値

`m` を引いた変換行列の参照.

gg.h の 2277 行目に定義があります。

呼び出し関係図:



8.7.3.48 operator-() [2/2]

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

変換行列に配列に格納した変換行列を減算した結果を格納する.

引数

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

戻り値

`a` を引いた変換行列の参照.

gg.h の 2265 行目に定義があります。

被呼び出し関係図:



8.7.3.49 operator/() [1/2]

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

変換行列を変換行列で除算した値を返す.

引数

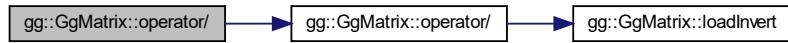
<i>a</i>	GgMatrix 型の変換行列.
----------	------------------

戻り値

変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2349 行目に定義があります。

呼び出し関係図:



8.7.3.50 operator/() [2/2]

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

変換行列を配列に格納した変換行列で除算した値を返す.

引数

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

戻り値

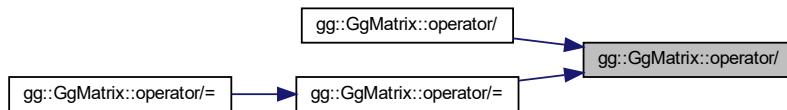
変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2335 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.51 operator/() [1/2]

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

変換行列を別の変換行列で除算した結果を格納する。

引数

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

戻り値

m で割った変換行列の参照。

gg.h の 2372 行目に定義があります。

呼び出し関係図:



8.7.3.52 operator/() [2/2]

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

変換行列を配列に格納した変換行列で除算した結果を格納する。

引数

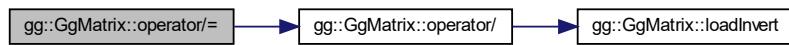
a	変換行列を格納した GLfloat 型の 16 要素の配列変数。
----------	----------------------------------

戻り値

a で割った変換行列の参照。

gg.h の 2360 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.53 operator=()

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

配列変数の値を格納する。

引数

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

戻り値

a を代入したこのオブジェクトの参照.

gg.h の 2182 行目に定義があります。

被呼び出し関係図:



8.7.3.54 orthogonal()

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

直交投影変換を乗じた結果を返す.

引数

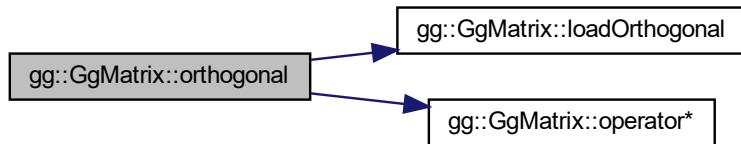
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

直交投影変換行列を乗じた変換行列.

gg.h の 2904 行目に定義があります。

呼び出し関係図:



8.7.3.55 `perspective()`

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

画角を指定して透視投影変換を乗じた結果を返す。

引数

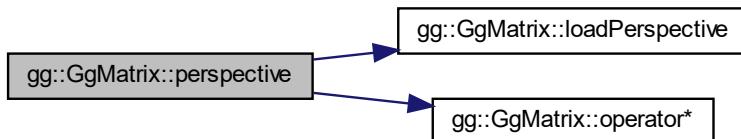
<code>fovy</code>	y 方向の画角.
<code>aspect</code>	縦横比.
<code>zNear</code>	視点から前方面までの位置.
<code>zFar</code>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

`gg.h` の 2944 行目に定義があります。

呼び出し関係図:



8.7.3.56 projection() [1/4]

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

ベクトルに対して投影変換を行う。

引数

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

gg.h の 3025 行目に定義があります。

8.7.3.57 projection() [2/4]

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

ベクトルに対して投影変換を行う。

引数

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

gg.h の 3014 行目に定義があります。

8.7.3.58 projection() [3/4]

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

ベクトルに対して投影変換を行う。

引数

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

gg.h の 3003 行目に定義があります。

8.7.3.59 projection() [4/4]

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

ベクトルに対して投影変換を行う。

引数

<i>c</i>	変換結果を格納する <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 2992 行目に定義があります。

8.7.3.60 rotate() [1/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す。

引数

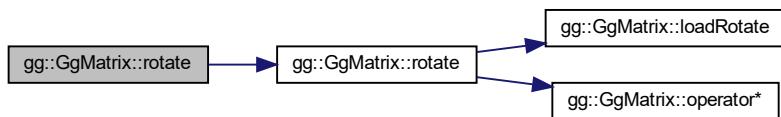
<i>r</i>	回転軸の方向ベクトルと回転角を格納した <code>GgVector</code> 型の変数)。
----------	--

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 2838 行目に定義があります。

呼び出し関係図:



8.7.3.61 rotate() [2/5]

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

r 方向のベクトルを軸とする回転変換を乗じた結果を返す.

引数

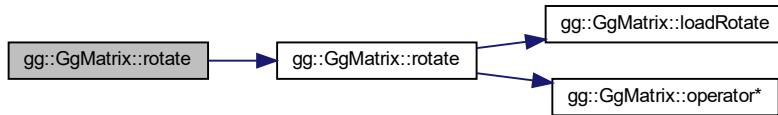
r	回転軸の方向ベクトルを格納した <code>GgVector</code> 型の変数.
a	回転角.

戻り値

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

gg.h の 2816 行目に定義があります。

呼び出し関係図:



8.7.3.62 rotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

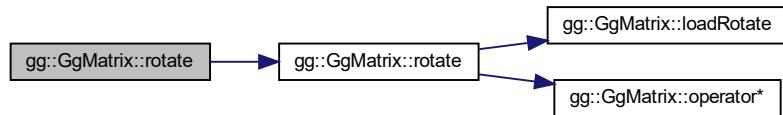
r	回転軸の方向ベクトルと回転角を格納した <code>GLfloat</code> 型の 4 要素の配列変数 (x, y, z, a).
-----	---

戻り値

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

gg.h の 2827 行目に定義があります。

呼び出し関係図:



8.7.3.63 `rotate()` [4/5]

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

`r` 方向のベクトルを軸とする回転変換を乗じた結果を返す.

引数

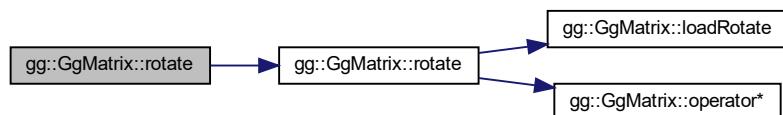
<code>r</code>	回転軸の方向ベクトルを格納した <code>GLfloat</code> 型の 3 要素の配列変数 (<code>x, y, z</code>).
<code>a</code>	回転角.

戻り値

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

`gg.h` の 2804 行目に定義があります。

呼び出し関係図:



8.7.3.64 rotate() [5/5]

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

(x, y, z) 方向のベクトルを軸とする回転変換を乗じた結果を返す。

引数

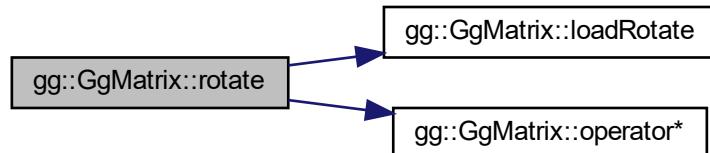
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

(x, y, z) を軸にさらに a 回転した変換行列.

gg.h の 2791 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.65 rotateX()

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

x 軸中心の回転変換を乗じた結果を返す。

引数

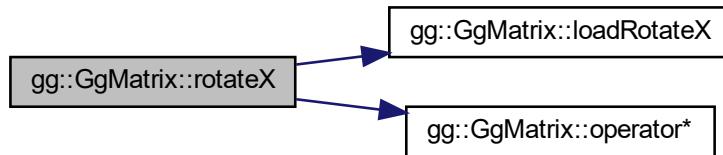
a	回転角.
---	------

戻り値

x 軸中心にさらに a 回転した変換行列。

gg.h の 2752 行目に定義があります。

呼び出し関係図:



8.7.3.66 rotateY()

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

y 軸中心の回転変換を乗じた結果を返す。

引数

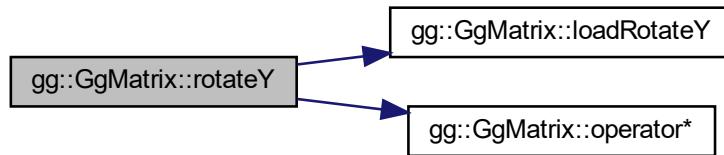
a	回転角.
---	------

戻り値

y 軸中心にさらに a 回転した変換行列。

gg.h の 2764 行目に定義があります。

呼び出し関係図:



8.7.3.67 rotateZ()

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

`z` 軸中心の回転変換を乗じた結果を返す.

引数

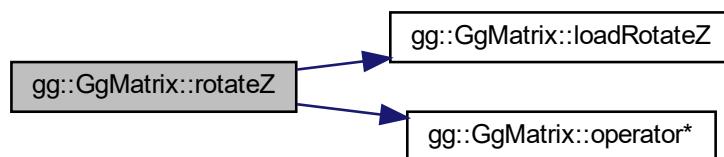
a	回転角.
---	------

戻り値

`z` 軸中心にさらに `a` 回転した変換行列.

`gg.h` の 2776 行目に定義があります。

呼び出し関係図:



8.7.3.68 scale() [1/3]

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

拡大縮小変換を乗じた結果を返す.

引数

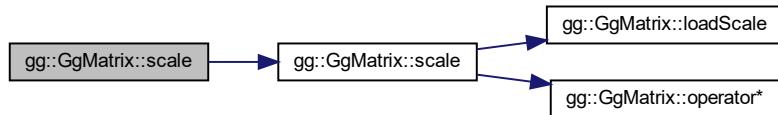
s	拡大率の GgVector 型の変数.
----------	---------------------

戻り値

拡大縮小した結果の変換行列.

gg.h の 2741 行目に定義があります。

呼び出し関係図:



8.7.3.69 scale() [2/3]

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

拡大縮小変換を乗じた結果を返す.

引数

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

戻り値

拡大縮小した結果の変換行列.

gg.h の 2730 行目に定義があります。

呼び出し関係図:



8.7.3.70 scale() [3/3]

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

拡大縮小変換を乗じた結果を返す。

引数

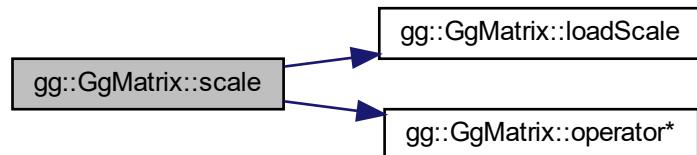
<code>x</code>	<code>x</code> 方向の拡大率。
<code>y</code>	<code>y</code> 方向の拡大率。
<code>z</code>	<code>z</code> 方向の拡大率。
<code>w</code>	<code>w</code> 移動量のスケールファクタ (<code>= 1.0f</code>)。

戻り値

拡大縮小した結果の変換行列。

`gg.h` の 2718 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



8.7.3.71 translate() [1/3]

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

平行移動変換を乗じた結果を返す.

引数

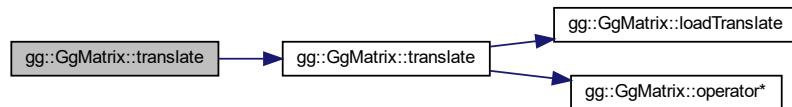
<i>t</i>	移動量の GgVector 型の変数.
----------	---------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2704 行目に定義があります。

呼び出し関係図:



8.7.3.72 translate() [2/3]

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

平行移動変換を乗じた結果を返す.

引数

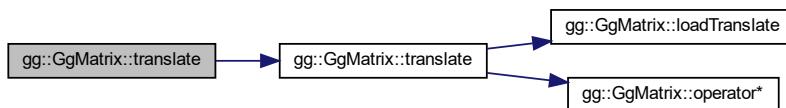
<i>t</i>	移動量の GLfloat 型の 3 要素の配列変数 (<i>x</i> , <i>y</i> , <i>z</i>).
----------	---

戻り値

平行移動した結果の変換行列.

gg.h の 2693 行目に定義がります。

呼び出し関係図:



8.7.3.73 translate() [3/3]

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

平行移動変換を乗じた結果を返す.

引数

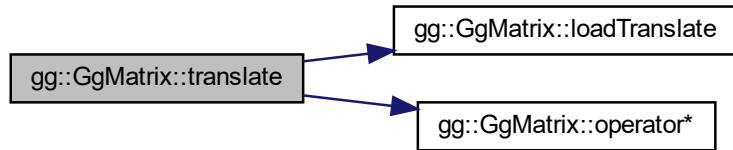
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

平行移動した結果の変換行列.

gg.h の 2681 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.7.3.74 transpose()

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

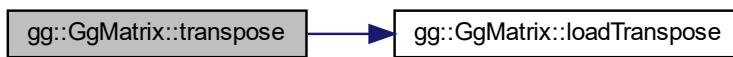
転置行列を返す。

戻り値

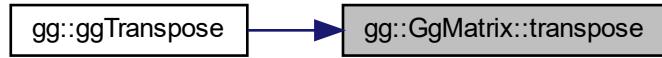
転置行列。

`gg.h` の 2958 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.8 gg::GgNormalTexture クラス

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

公開メンバ関数

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

8.8.1 詳解

法線マップ.

覚え書き

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

gg.h の 5394 行目に定義があります。

8.8.2 構築子と解体子

8.8.2.1 GgNormalTexture() [1/3]

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

コンストラクタ.

gg.h の 5404 行目に定義があります。

8.8.2.2 GgNormalTexture() [2/3]

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

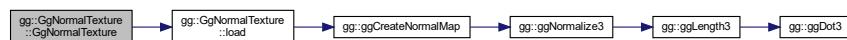
メモリ上のデータから法線マップのテクスチャを作成するコンストラクタ.

引数

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

gg.h の 5418 行目に定義があります。

呼び出し関係図:



8.8.2.3 GgNormalTexture() [3/3]

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

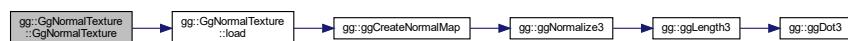
ファイルからデータを読み込んで法線マップのテクスチャを作成するコンストラクタ.

引数

<i>name</i>	画像ファイル名.
<i>nz</i>	法線マップの z 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5438 行目に定義があります。

呼び出し関係図:



8.8.2.4 ~GgNormalTexture()

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

デストラクタ.

gg.h の 5451 行目に定義があります。

8.8.3 関数詳解

8.8.3.1 load() [1/2]

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

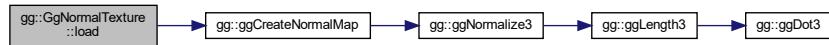
メモリ上のデータから法線マップのテクスチャを作成する.

引数

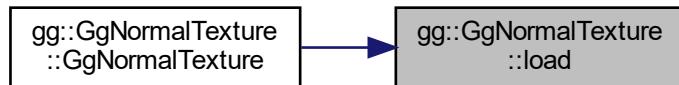
<i>hmap</i>	テクスチャとして用いる画像データ, nullptr ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット (GL_RED, GL_RG, GL_RGB, GL_RGBA).
<i>nz</i>	法線マップの z 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5465 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.8.3.2 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 3952 行目に定義がります。

呼び出し関係図:



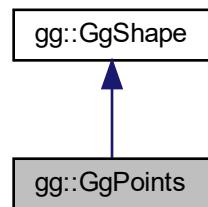
このクラス詳解は次のファイルから抽出されました:

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

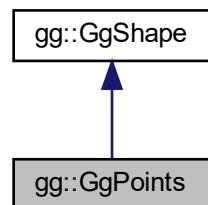
8.9 gg::GgPoints クラス

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

gg::GgPoints の継承関係図



gg::GgPoints 連携図



公開 メンバ関数

- [GgPoints \(GLenum mode=GL_POINTS\)](#)
- [GgPoints \(const GgVector *pos, GLsizei countv, GLenum mode=GL_POINTS, GLenum usage=GL_STATIC_DRAW\)](#)
- virtual [~GgPoints \(\)](#)
- const GLsizei & [getCount \(\) const](#)
- const GLuint & [getBuffer \(\) const](#)
- void [send \(const GgVector *pos, GLint first=0, GLsizei count=0\) const](#)
- void [load \(const GgVector *pos, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- virtual void [draw \(GLint first=0, GLsizei count=0\) const](#)

8.9.1 詳解

点.

gg.h の 6159 行目に定義がります。

8.9.2 構築子と解体子

8.9.2.1 GgPoints() [1/2]

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

コンストラクタ.

gg.h の 6170 行目に定義がります。

8.9.2.2 GgPoints() [2/2]

```
gg::GgPoints::GgPoints (
    const GgVector * pos,
    GLsizei countv,
    GLenum mode = GL_POINTS,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	この図形の頂点の位置のデータの配列 (nullptr ならデータを転送しない).
<i>countv</i>	頂点数.
<i>mode</i>	描画する基本図形の種類.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 6183 行目に定義がります。

8.9.2.3 ~GgPoints()

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

デストラクタ.

gg.h の 6197 行目に定義がります。

8.9.3 関数詳解

8.9.3.1 draw()

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

点の描画.

引数

<i>first</i>	描画を開始する最初の点の番号.
<i>count</i>	描画する点の数, 0 なら全部の点を描く.

[gg::GgShape](#) を再実装しています。

gg.cpp の 5030 行目に定義があります。

呼び出し関係図:



8.9.3.2 getBuffer()

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

頂点の位置データを格納した頂点バッファオブジェクト名を取り出す.

戻り値

この図形の頂点の位置データを格納した頂点バッファオブジェクト名.

gg.h の 6216 行目に定義があります。

8.9.3.3 getCount()

```
const GLsizei& gg::GgPoints::getCount ( ) const [inline]
```

データの数を取り出す.

戻り値

この図形の頂点の位置データの数(頂点数).

gg.h の 6206 行目に定義があります。

8.9.3.4 load()

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

バッファオブジェクトを確保して頂点の位置データを格納する.

引数

<i>pos</i>	頂点の位置データが格納されてい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 5017 行目に定義があります。

8.9.3.5 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 6228 行目に定義がります。

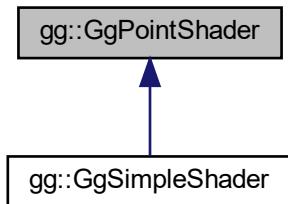
このクラス詳解は次のファイルから抽出されました:

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

8.10 gg::GgPointShader クラス

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

gg::GgPointShader の継承関係図



公開 メンバ関数

- [GgPointShader \(\)](#)
- [GgPointShader \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr\)](#)
- [GgPointShader \(const std::array< std::string, 3 > &files, int nvarying=0, const char *const *varyings=nullptr\)](#)
- [virtual ~GgPointShader \(\)](#)
- [bool load \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr\)](#)
- [bool load \(const std::array< std::string, 3 > &files, GLint nvarying=0, const char *const *varyings=nullptr\)](#)
- [virtual void loadProjectionMatrix \(const GLfloat *mp\) const](#)
- [virtual void loadProjectionMatrix \(const GgMatrix &mp\) const](#)
- [virtual void loadModelviewMatrix \(const GLfloat *mv\) const](#)
- [virtual void loadModelviewMatrix \(const GgMatrix &mv\) const](#)
- [virtual void loadMatrix \(const GLfloat *mp, const GLfloat *mv\) const](#)
- [virtual void loadMatrix \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- [virtual void use \(\) const](#)
- [void use \(const GLfloat *mp\) const](#)
- [void use \(const GgMatrix &mp\) const](#)
- [void use \(const GLfloat *mp, const GLfloat *mv\) const](#)
- [void use \(const GgMatrix &mp, const GgMatrix &mv\) const](#)
- [void unuse \(\) const](#)
- [GLuint get \(\) const](#)

8.10.1 詳解

点のシェーダ.

gg.h の 6763 行目に定義がります。

8.10.2 構築子と解体子

8.10.2.1 GgPointShader() [1/3]

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

コンストラクタ.

gg.h の 6779 行目に定義がります。

8.10.2.2 GgPointShader() [2/3]

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

コンストラクタ.

引数

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

gg.h の 6794 行目に定義がります。

8.10.2.3 GgPointShader() [3/3]

```
gg::GgPointShader::GgPointShader (
    const std::array< std::string, 3 > & files,
```

```
int nvarying = 0,
const char *const * varyings = nullptr ) [inline]
```

コンストラクタ.

引数

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

gg.h の 6813 行目に定義があります。

8.10.2.4 ~GgPointShader()

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

デストラクタ.

gg.h の 6825 行目に定義があります。

8.10.3 関数詳解

8.10.3.1 get()

```
GLuint gg::GgPointShader::get() const [inline]
```

シェーダのプログラム名を得る.

戻り値

シェーダのプログラム名.

gg.h の 7009 行目に定義があります。

8.10.3.2 load() [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する.

引数

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

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 6872 行目に定義があります。

8.10.3.3 **load()** [2/2]

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

シェーダのソースファイルを読み込む.

引数

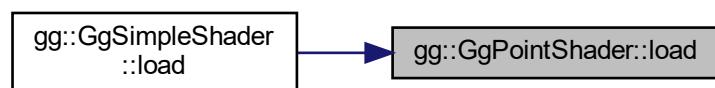
<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト.

戻り値

プログラムオブジェクトが作成できれば true.

gg.h の 6839 行目に定義があります。

被呼び出し関係図:



8.10.3.4 loadMatrix() [1/2]

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

投影変換行列とモデルビュー変換行列を設定する。

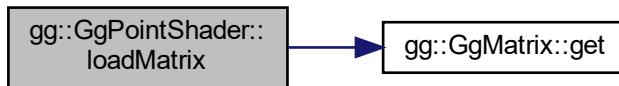
引数

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

[gg::GgSimpleShader](#)で再実装されています。

gg.h の 6939 行目に定義があります。

呼び出し関係図:



8.10.3.5 loadMatrix() [2/2]

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

投影変換行列とモデルビュー変換行列を設定する。

引数

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

[gg::GgSimpleShader](#)で再実装されています。

gg.h の 6927 行目に定義がります。

8.10.3.6 loadModelviewMatrix() [1/2]

```
virtual void gg::GgPointShader::loadModelviewMatrix (
    const GgMatrix & mv ) const [inline], [virtual]
```

モデルビュー変換行列を設定する。

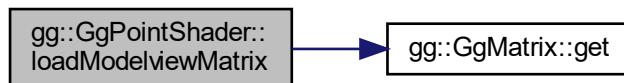
引数

<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
-----------	------------------------

gg::GgSimpleShaderで再実装されています。

gg.h の 6916 行目に定義がります。

呼び出し関係図:



8.10.3.7 loadModelviewMatrix() [2/2]

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

モデルビュー変換行列を設定する。

引数

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

gg::GgSimpleShaderで再実装されています。

gg.h の 6906 行目に定義がります。

8.10.3.8 loadProjectionMatrix() [1/2]

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

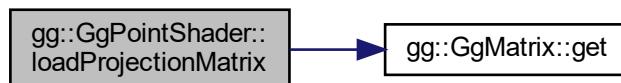
投影変換行列を設定する。

引数

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

gg.h の 6896 行目に定義があります。

呼び出し関係図:



8.10.3.9 loadProjectionMatrix() [2/2]

```
virtual void gg::GgPointShader::loadProjectionMatrix (
    const GLfloat * mp ) const [inline], [virtual]
```

投影変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
-----------	------------------------------------

gg.h の 6886 行目に定義があります。

8.10.3.10 unuse()

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

シェーダプログラムの使用を終了する。

gg.h の 6999 行目に定義があります。

8.10.3.11 use() [1/5]

```
virtual void gg::GgPointShader::use ( ) const [inline], [virtual]
```

シェーダプログラムの使用を開始する。

`gg::GgSimpleShader`で再実装されています。

`gg.h` の 6947 行目に定義があります。

8.10.3.12 use() [2/5]

```
void gg::GgPointShader::use (
    const GgMatrix & mp ) const [inline]
```

投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<code>mp</code>	<code>GgMatrix</code> 型の投影変換行列。
-----------------	---------------------------------

`gg.h` の 6968 行目に定義があります。

呼び出し関係図:



8.10.3.13 use() [3/5]

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

投影変換行列とモデルビューを設定してシェーダプログラムの使用を開始する。

引数

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

gg.h の 6991 行目に定義があります。

呼び出し関係図:



8.10.3.14 use() [4/5]

```
void gg::GgPointShader::use (
    const GLfloat * mp ) const [inline]
```

投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列。
-----------	------------------------------------

gg.h の 6957 行目に定義があります。

8.10.3.15 use() [5/5]

```
void gg::GgPointShader::use (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline]
```

投影変換行列とモデルビューを設定してシェーダプログラムの使用を開始する。

引数

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

gg.h の 6979 行目に定義があります。

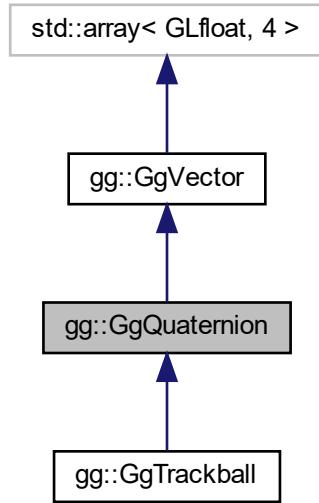
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

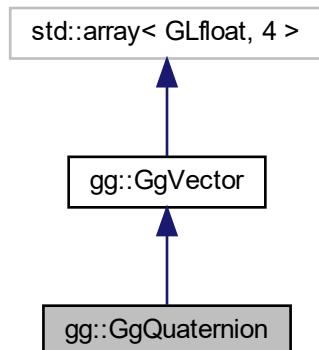
8.11 gg::GgQuaternion クラス

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

gg::GgQuaternion の継承関係図



gg::GgQuaternion 連携図



公開メンバ関数

- [GgQuaternion \(\)](#)

- constexpr `GgQuaternion` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- constexpr `GgQuaternion` (GLfloat c)
- `GgQuaternion` (const GLfloat *a)
- `GgQuaternion` (const `GgVector` &v)
- `GLfloat norm () const`
- `GgQuaternion & load` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- `GgQuaternion & load` (const GLfloat *a)
- `GgQuaternion & load` (const `GgVector` &v)
- `GgQuaternion & load` (const `GgQuaternion` &q)
- `GgQuaternion & loadAdd` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- `GgQuaternion & loadAdd` (const GLfloat *a)
- `GgQuaternion & loadAdd` (const `GgVector` &v)
- `GgQuaternion & loadAdd` (const `GgQuaternion` &q)
- `GgQuaternion & loadSubtract` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- `GgQuaternion & loadSubtract` (const GLfloat *a)
- `GgQuaternion & loadSubtract` (const `GgVector` &v)
- `GgQuaternion & loadSubtract` (const `GgQuaternion` &q)
- `GgQuaternion & loadMultiply` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- `GgQuaternion & loadMultiply` (const GLfloat *a)
- `GgQuaternion & loadMultiply` (const `GgVector` &v)
- `GgQuaternion & loadMultiply` (const `GgQuaternion` &q)
- `GgQuaternion & loadDivide` (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- `GgQuaternion & loadDivide` (const GLfloat *a)
- `GgQuaternion & loadDivide` (const `GgVector` &v)
- `GgQuaternion & loadDivide` (const `GgQuaternion` &q)
- `GgQuaternion add` (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
- `GgQuaternion add` (const GLfloat *a) const
- `GgQuaternion add` (const `GgVector` &v) const
- `GgQuaternion add` (const `GgQuaternion` &q) const
- `GgQuaternion subtract` (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
- `GgQuaternion subtract` (const GLfloat *a) const
- `GgQuaternion subtract` (const `GgVector` &v) const
- `GgQuaternion subtract` (const `GgQuaternion` &q) const
- `GgQuaternion multiply` (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
- `GgQuaternion multiply` (const GLfloat *a) const
- `GgQuaternion multiply` (const `GgVector` &v) const
- `GgQuaternion multiply` (const `GgQuaternion` &q) const
- `GgQuaternion divide` (GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
- `GgQuaternion divide` (const GLfloat *a) const
- `GgQuaternion divide` (const `GgVector` &v) const
- `GgQuaternion divide` (const `GgQuaternion` &q) const
- `GgQuaternion & operator=` (const GLfloat *a)
- `GgQuaternion & operator=` (const `GgVector` &v)
- `GgQuaternion & operator+=` (const GLfloat *a)
- `GgQuaternion & operator+=` (const `GgVector` &v)
- `GgQuaternion & operator+=` (const `GgQuaternion` &q)
- `GgQuaternion & operator-=` (const GLfloat *a)
- `GgQuaternion & operator-=` (const `GgVector` &v)
- `GgQuaternion & operator-=` (const `GgQuaternion` &q)
- `GgQuaternion & operator*= (const GLfloat *a)`
- `GgQuaternion & operator*= (const GgVector &v)`
- `GgQuaternion & operator*= (const GgQuaternion &q)`
- `GgQuaternion & operator/= (const GLfloat *a)`
- `GgQuaternion & operator/= (const GgVector &v)`
- `GgQuaternion & operator/= (const GgQuaternion &q)`

- `GgQuaternion operator+ (const GLfloat *a) const`
- `GgQuaternion operator+ (const GgVector &v) const`
- `GgQuaternion operator+ (const GgQuaternion &q) const`
- `GgQuaternion operator- (const GLfloat *a) const`
- `GgQuaternion operator- (const GgVector &v) const`
- `GgQuaternion operator- (const GgQuaternion &q) const`
- `GgQuaternion operator* (const GLfloat *a) const`
- `GgQuaternion operator* (const GgVector &v) const`
- `GgQuaternion operator* (const GgQuaternion &q) const`
- `GgQuaternion operator/ (const GLfloat *a) const`
- `GgQuaternion operator/ (const GgVector &v) const`
- `GgQuaternion operator/ (const GgQuaternion &q) const`
- `GgQuaternion & loadMatrix (const GLfloat *a)`
- `GgQuaternion & loadMatrix (const GgMatrix &m)`
- `GgQuaternion & loadIdentity ()`
- `GgQuaternion & loadRotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgQuaternion & loadRotate (const GLfloat *v, GLfloat a)`
- `GgQuaternion & loadRotate (const GLfloat *v)`
- `GgQuaternion & loadRotateX (GLfloat a)`
- `GgQuaternion & loadRotateY (GLfloat a)`
- `GgQuaternion & loadRotateZ (GLfloat a)`
- `GgQuaternion rotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a) const`
- `GgQuaternion rotate (const GLfloat *v, GLfloat a) const`
- `GgQuaternion rotate (const GLfloat *v) const`
- `GgQuaternion rotateX (GLfloat a) const`
- `GgQuaternion rotateY (GLfloat a) const`
- `GgQuaternion rotateZ (GLfloat a) const`
- `GgQuaternion & loadEuler (GLfloat heading, GLfloat pitch, GLfloat roll)`
- `GgQuaternion & loadEuler (const GLfloat *e)`
- `GgQuaternion euler (GLfloat heading, GLfloat pitch, GLfloat roll) const`
- `GgQuaternion euler (const GLfloat *e) const`
- `GgQuaternion & loadSlerp (const GLfloat *a, const GLfloat *b, GLfloat t)`
- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
- `GgQuaternion & loadSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
- `GgQuaternion & loadSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
- `GgQuaternion & loadNormalize (const GLfloat *a)`
- `GgQuaternion & loadNormalize (const GgQuaternion &q)`
- `GgQuaternion & loadConjugate (const GLfloat *a)`
- `GgQuaternion & loadConjugate (const GgQuaternion &q)`
- `GgQuaternion & loadInvert (const GLfloat *a)`
- `GgQuaternion & loadInvert (const GgQuaternion &q)`
- `GgQuaternion slerp (GLfloat *a, GLfloat t) const`
- `GgQuaternion slerp (const GgQuaternion &q, GLfloat t) const`
- `GgQuaternion normalize () const`
- `GgQuaternion conjugate () const`
- `GgQuaternion invert () const`
- `void get (GLfloat *a) const`
- `void getMatrix (GLfloat *a) const`
- `void getMatrix (GgMatrix &m) const`
- `GgMatrix getMatrix () const`
- `void getConjugateMatrix (GLfloat *a) const`
- `void getConjugateMatrix (GgMatrix &m) const`
- `GgMatrix getConjugateMatrix () const`

8.11.1 詳解

四元数.

gg.h の 3408 行目に定義がります。

8.11.2 構築子と解体子

8.11.2.1 GgQuaternion() [1/5]

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

コンストラクタ.

gg.h の 3427 行目に定義がります。

8.11.2.2 GgQuaternion() [2/5]

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

コンストラクタ.

引数

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

gg.h の 3439 行目に定義がります。

8.11.2.3 GgQuaternion() [3/5]

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

コンストラクタ.

引数

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

gg.h の 3449 行目に定義があります。

8.11.2.4 GgQuaternion() [4/5]

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

コンストラクタ.

引数

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

gg.h の 3459 行目に定義があります。

8.11.2.5 GgQuaternion() [5/5]

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

コンストラクタ.

引数

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

gg.h の 3469 行目に定義があります。

8.11.3 関数詳解

8.11.3.1 add() [1/4]

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

四元数に別の四元数を加算した結果を返す.

引数

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

戻り値

`q` を加えた四元数.

gg.h の 3778 行目に定義がります。

呼び出し関係図:



8.11.3.2 add() [2/4]

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

四元数に別の四元数を加算した結果を返す.

引数

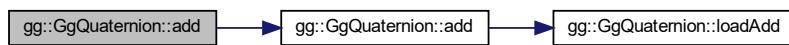
<code>v</code>	四元数を格納した <code>GgVector</code> 型の変数.
----------------	--------------------------------------

戻り値

`v` を加えた四元数.

gg.h の 3767 行目に定義がります。

呼び出し関係図:



8.11.3.3 add() [3/4]

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

四元数に別の四元数を加算した結果を返す.

引数

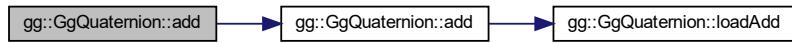
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
----------	--------------------------------

戻り値

a を加えた四元数.

gg.h の 3756 行目に定義があります。

呼び出し関係図:



8.11.3.4 add() [4/4]

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

四元数に別の四元数を加算した結果を返す.

引数

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

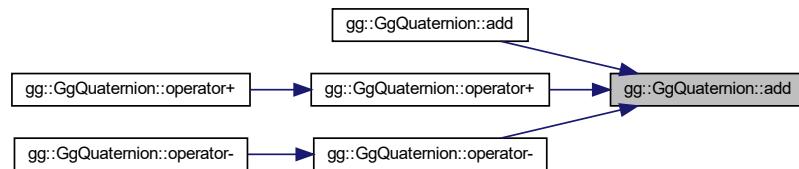
(x, y, z, w) を加えた四元数.

gg.h の 3744 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.5 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 4401 行目に定義がります。

呼び出し関係図:



呼び出し関係図:



8.11.3.6 divide() [1/4]

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

四元数を別の四元数で除算した結果を返す.

引数

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

戻り値

q で割った四元数.

gg.h の 3927 行目に定義があります。

呼び出し関係図:



8.11.3.7 divide() [2/4]

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

四元数を別の四元数で除算した結果を返す.

引数

<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 3916 行目に定義があります。

呼び出し関係図:



8.11.3.8 divide() [3/4]

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

四元数を別の四元数で除算した結果を返す.

引数

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

戻り値

`a` で割った四元数.

gg.h の 3902 行目に定義があります。

呼び出し関係図:



8.11.3.9 divide() [4/4]

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

四元数を別の四元数で除算した結果を返す.

引数

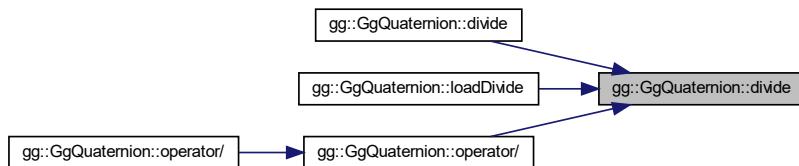
<i>x</i>	割る四元数の <i>x</i> 要素.
<i>y</i>	割る四元数の <i>y</i> 要素.
<i>z</i>	割る四元数の <i>z</i> 要素.
<i>w</i>	割る四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を割った四元数.

gg.h の 3890 行目に定義があります。

被呼び出し関係図:



8.11.3.10 euler() [1/2]

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

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

引数

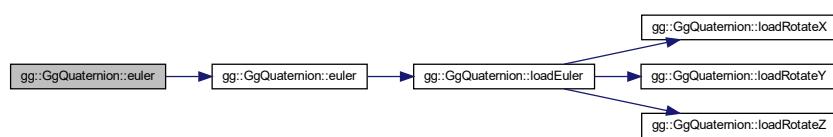
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転した四元数.

gg.h の 4241 行目に定義がります。

呼び出し関係図:



8.11.3.11 euler() [2/2]

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

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

引数

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

戻り値

回転した四元数.

gg.h の 4229 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.12 get()

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

四元数を取り出す。

引数

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

gg.h の 4425 行目に定義があります。

8.11.3.13 getConjugateMatrix() [1/3]

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

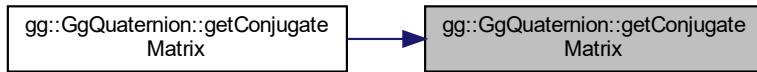
四元数の共役が表す回転の変換行列を取り出す。

戻り値

回転の変換を表す [GgMatrix](#) 型の変換行列.

gg.h の 4492 行目に定義があります。

被呼び出し関係図:



8.11.3.14 getConjugateMatrix() [2/3]

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

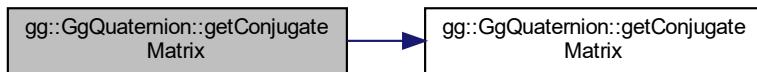
四元数の共役が表す回転の変換行列を *m* に求める.

引数

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

gg.h の 4482 行目に定義があります。

呼び出し関係図:



8.11.3.15 getConjugateMatrix() [3/3]

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

四元数の共役が表す回転の変換行列を *a* に求める.

引数

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

gg.h の 4470 行目に定義があります。

呼び出し関係図:



8.11.3.16 getMatrix() [1/3]

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

四元数が表す回転の変換行列を取り出す。

戻り値

回転の変換を表す `GgMatrix` 型の変換行列。

gg.h の 4458 行目に定義があります。

被呼び出し関係図:



8.11.3.17 getMatrix() [2/3]

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

四元数が表す回転の変換行列を m に求める。

引数

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

gg.h の 4448 行目に定義があります。

呼び出し関係図:



8.11.3.18 getMatrix() [3/3]

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

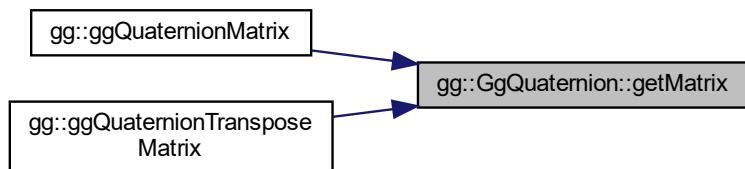
四元数が表す回転の変換行列を *a* に求める。

引数

<i>a</i>	回転の変換行列を格納する GLfloat 型の 16 要素の配列変数.
----------	-------------------------------------

gg.h の 4438 行目に定義があります。

被呼び出し関係図:



8.11.3.19 invert()

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

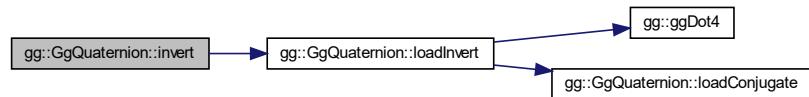
逆元に変換する。

戻り値

四元数の逆元。

gg.h の 4413 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.20 load() [1/4]

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

四元数を格納する。

引数

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

戻り値

設定した四元数.

gg.h の 3531 行目に定義がります。

8.11.3.21 `load()` [2/4]

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

四元数を格納する.

引数

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

戻り値

設定した四元数.

gg.h の 3519 行目に定義がります。

8.11.3.22 `load()` [3/4]

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

四元数を格納する.

引数

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

戻り値

設定した四元数.

gg.h の 3508 行目に定義がります。

呼び出し関係図:



8.11.3.23 load() [4/4]

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

四元数を格納する。

引数

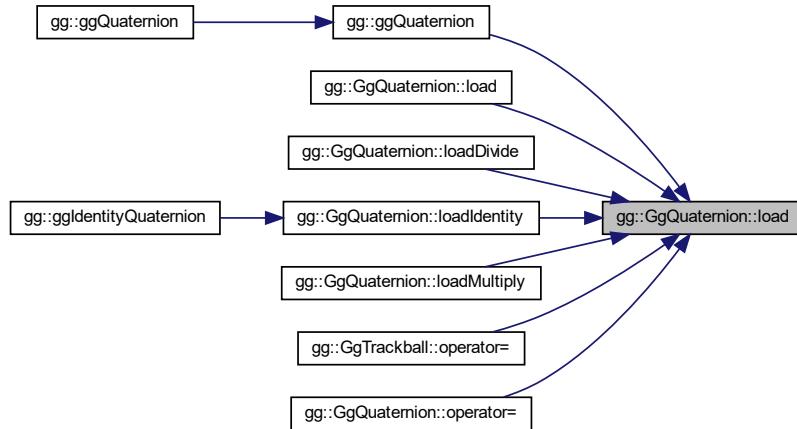
<i>x</i>	四元数の x 要素.
<i>y</i>	四元数の y 要素.
<i>z</i>	四元数の z 要素.
<i>w</i>	四元数の w 要素.

戻り値

設定した四元数.

gg.h の 3493 行目に定義があります。

被呼び出し関係図:



8.11.3.24 loadAdd() [1/4]

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

四元数に別の四元数を加算した結果を格納する。

引数

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

戻り値

`q` を加えた四元数。

gg.h の 3583 行目に定義があります。

呼び出し関係図:



8.11.3.25 `loadAdd()` [2/4]

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

四元数に別の四元数を加算した結果を格納する。

引数

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

戻り値

`v` を加えた四元数。

`gg.h` の 3572 行目に定義があります。

呼び出し関係図:



8.11.3.26 `loadAdd()` [3/4]

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

四元数に別の四元数を加算した結果を格納する。

引数

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

戻り値

`a` を加えた四元数。

`gg.h` の 3561 行目に定義があります。

呼び出し関係図:



8.11.3.27 loadAdd() [4/4]

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

四元数に別の四元数を加算した結果を格納する。

引数

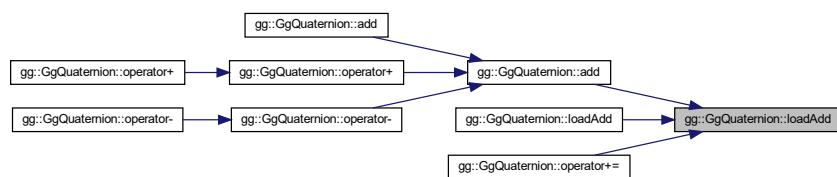
<i>x</i>	加える四元数の <i>x</i> 要素.
<i>y</i>	加える四元数の <i>y</i> 要素.
<i>z</i>	加える四元数の <i>z</i> 要素.
<i>w</i>	加える四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を加えた四元数。

gg.h の 3546 行目に定義があります。

被呼び出し関係図:



8.11.3.28 loadConjugate() [1/2]

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

引数に指定した四元数の共役四元数を格納する。

引数

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

戻り値

共役四元数。

gg.h の 4332 行目に定義があります。

呼び出し関係図:



8.11.3.29 loadConjugate() [2/2]

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

引数に指定した四元数の共役四元数を格納する。

引数

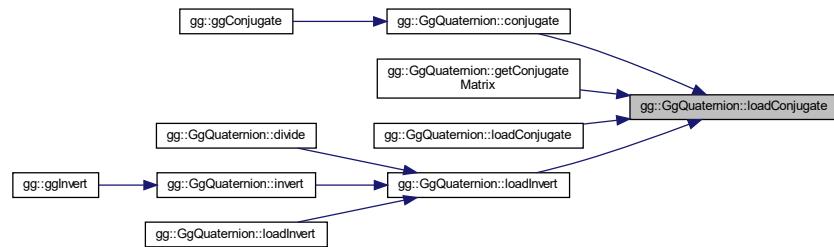
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
----------	--------------------------------

戻り値

共役四元数。

gg.cpp の 3274 行目に定義があります。

被呼び出し関係図:



8.11.3.30 loadDivide() [1/4]

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

四元を別の四元数で除算した結果を格納する。

引数

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

戻り値

q で割った四元数。

gg.h の 3730 行目に定義があります。

呼び出し関係図:



8.11.3.31 loadDivide() [2/4]

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

四元を別の四元数で除算した結果を格納する。

引数

v	四元数を格納した GgVector 型の変数.
---	---

戻り値

v で割った四元数.

gg.h の 3719 行目に定義があります。

呼び出し関係図:



8.11.3.32 loadDivide() [3/4]

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

四元を別の四元数で除算した結果を格納する.

引数

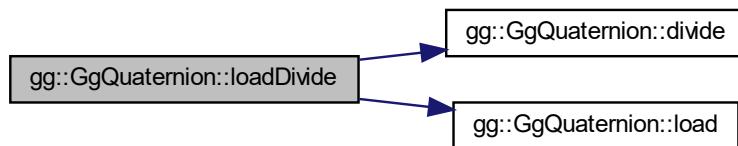
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a で割った四元数.

gg.h の 3708 行目に定義があります。

呼び出し関係図:



8.11.3.33 loadDivide() [4/4]

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

四元を別の四元数で除算した結果を格納する.

引数

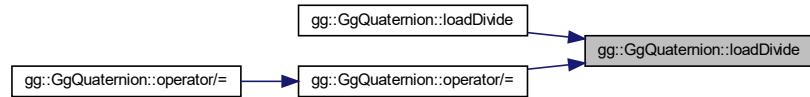
x	割る四元数の x 要素.
y	割る四元数の y 要素.
z	割る四元数の z 要素.
w	割る四元数の w 要素.

戻り値

(x, y, z, w) を割った四元数.

gg.h の 3696 行目に定義があります。

被呼び出し関係図:



8.11.3.34 loadEuler() [1/2]

```
gg::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 4216 行目に定義があります。

呼び出し関係図:



8.11.3.35 loadEuler() [2/2]

```
gg::GgQuaternion & gg::GgQuaternion::loadEuler (
    GLfloat heading,
    GLfloat pitch,
    GLfloat roll )
```

オイラー角 (heading, pitch, roll) で与えられた回転を表す四元数を格納する。

引数

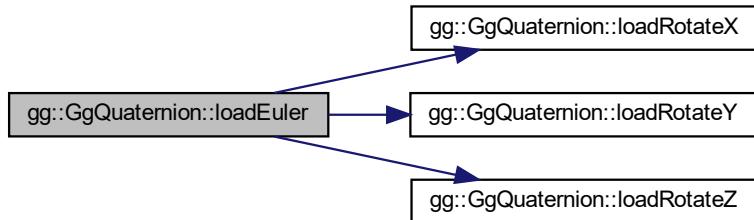
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

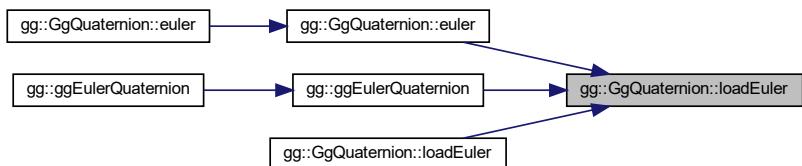
格納した回転を表す四元数.

gg.cpp の 3243 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.36 loadIdentity()

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

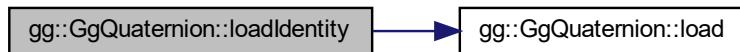
単位元を格納する.

戻り値

格納された単位元.

gg.h の 4066 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.37 loadInvert() [1/2]

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

引数に指定した四元数の逆元を格納する.

引数

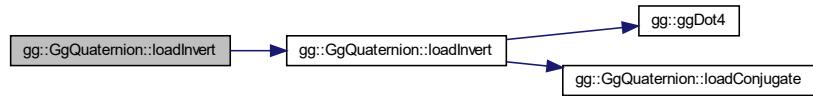
<i>q</i>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

四元数の逆元.

gg.h の 4351 行目に定義があります。

呼び出し関係図:



8.11.3.38 `loadInvert()` [2/2]

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

引数に指定した四元数の逆元を格納する。

引数

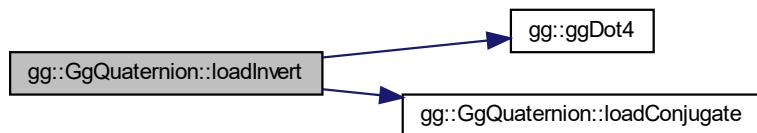
<code>a</code>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
----------------	---

戻り値

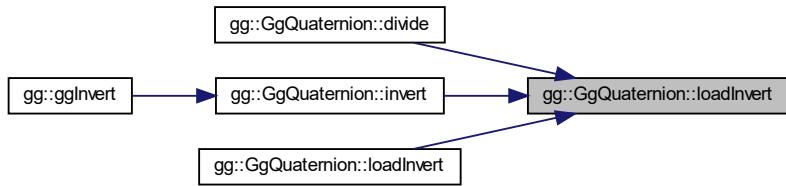
四元数の逆元。

gg.cpp の 3288 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



8.11.3.39 loadMatrix() [1/2]

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

回転の変換行列 m を表す四元数を格納する。

引数

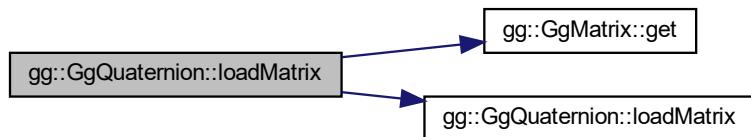
m	Ggmatrix 型の変換行列。
-----	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4056 行目に定義があります。

呼び出し関係図:



8.11.3.40 loadMatrix() [2/2]

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

回転の変換行列を表す四元数を格納する。

引数

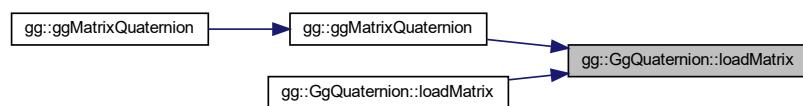
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 4044 行目に定義があります。

呼び出し関係図:



8.11.3.41 loadMultiply() [1/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

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

戻り値

q を乗じた四元数.

gg.h の 3682 行目に定義があります。

呼び出し関係図:



8.11.3.42 loadMultiply() [2/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

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

戻り値

v を乗じた四元数.

gg.h の 3671 行目に定義があります。

呼び出し関係図:



8.11.3.43 loadMultiply() [3/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

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

戻り値

a を乗じた四元数.

gg.h の 3660 行目に定義があります。

呼び出し関係図:



8.11.3.44 loadMultiply() [4/4]

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

四元数に別の四元数を乗算した結果を格納する。

引数

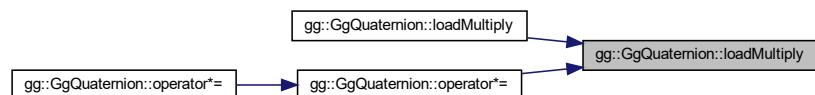
<i>x</i>	掛ける四元数の <i>x</i> 要素.
<i>y</i>	掛ける四元数の <i>y</i> 要素.
<i>z</i>	掛ける四元数の <i>z</i> 要素.
<i>w</i>	掛ける四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を掛けた四元数。

gg.h の 3648 行目に定義があります。

被呼び出し関係図:



8.11.3.45 **loadNormalize()** [1/2]

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

引数に指定した四元数を正規化して格納する。

引数

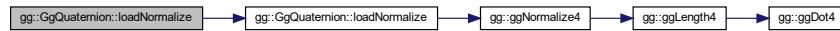
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

正規化された四元数。

gg.h の 4313 行目に定義があります。

呼び出し関係図:

8.11.3.46 **loadNormalize()** [2/2]

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

引数に指定した四元数を正規化して格納する。

引数

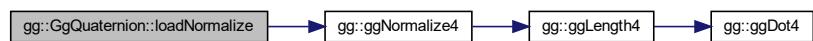
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
----------	--------------------------------

戻り値

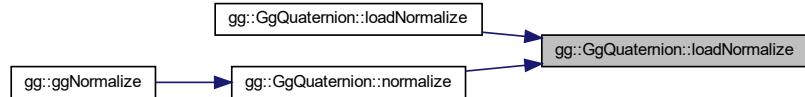
正規化された四元数。

gg.cpp の 3259 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.47 loadRotate() [1/3]

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

(v[0], v[1], v[2]) を軸として角度 v[3] 回転する四元数を格納する。

引数

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 4100 行目に定義があります。

呼び出し関係図:



8.11.3.48 loadRotate() [2/3]

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

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

引数

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

戻り値

格納された回転を表す四元数.

gg.h の 4089 行目に定義があります。

呼び出し関係図:



8.11.3.49 loadRotate() [3/3]

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

(x, y, z) を軸として角度 a 回転する四元数を格納する.

引数

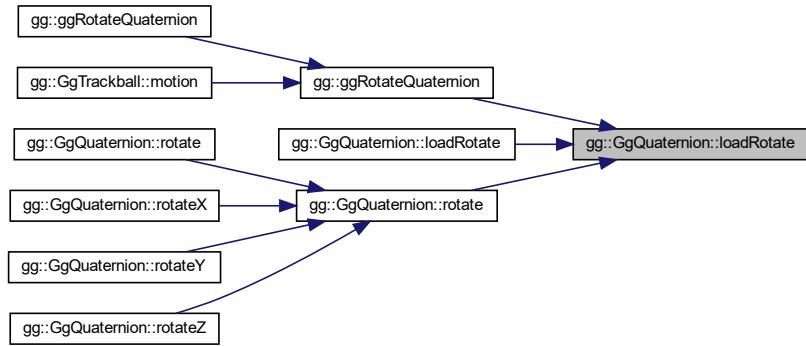
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

格納された回転を表す四元数.

gg.cpp の 3177 行目に定義があります。

被呼び出し関係図:



8.11.3.50 loadRotateX()

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

x 軸中心に角度 a 回転する四元数を格納する。

引数

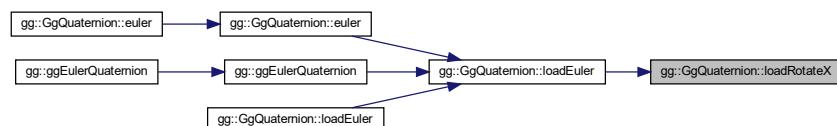
a	回転角.
-----	------

戻り値

格納された回転を表す四元数。

`gg.cpp` の 3201 行目に定義があります。

被呼び出し関係図:



8.11.3.51 loadRotateY()

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

y 軸中心に角度 a 回転する四元数を格納する。

引数

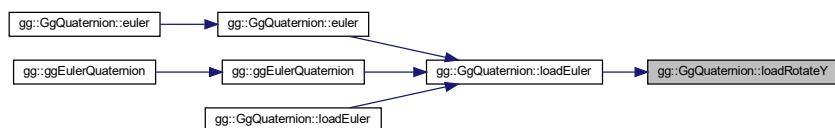
a	回転角.
---	------

戻り値

格納された回転を表す四元数。

gg.cpp の 3215 行目に定義があります。

被呼び出し関係図:



8.11.3.52 loadRotateZ()

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

z 軸中心に角度 a 回転する四元数を格納する。

引数

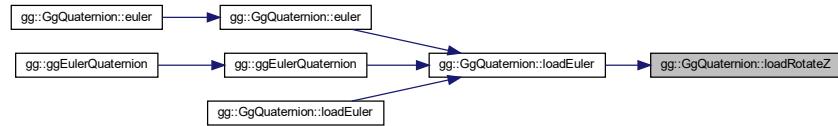
a	回転角.
---	------

戻り値

格納された回転を表す四元数。

gg.cpp の 3229 行目に定義があります。

被呼び出し関係図:



8.11.3.53 loadSlerp() [1/4]

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

球面線形補間の結果を格納する。

引数

<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 4268 行目に定義があります。

呼び出し関係図:



8.11.3.54 loadSlerp() [2/4]

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

球面線形補間の結果を格納する。

引数

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

戻り値

格納した *q*, *a* を *t* で内分した四元数。

gg.h の 4281 行目に定義があります。

呼び出し関係図:



8.11.3.55 loadSlerp() [3/4]

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

球面線形補間の結果を格納する。

引数

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

戻り値

格納した a, q を t で内分した四元数.

gg.h の 4294 行目に定義があります。

呼び出し関係図:



8.11.3.56 loadSlerp() [4/4]

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

球面線形補間の結果を格納する.

引数

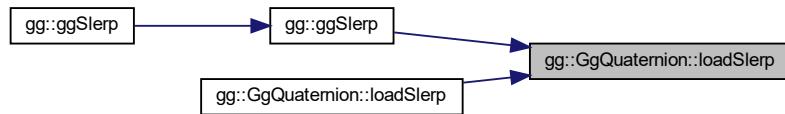
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した a, b を t で内分した四元数.

gg.h の 4254 行目に定義があります。

被呼び出し関係図:



8.11.3.57 loadSubtract() [1/4]

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

四元数から別の四元数を減算した結果を格納する.

引数

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

戻り値

q を引いた四元数.

gg.h の 3634 行目に定義がります。

呼び出し関係図:

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

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

四元数から別の四元数を減算した結果を格納する.

引数

<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 3623 行目に定義がります。

呼び出し関係図:



8.11.3.59 loadSubtract() [3/4]

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

四元数から別の四元数を減算した結果を格納する。

引数

<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 3612 行目に定義があります。

呼び出し関係図:



8.11.3.60 loadSubtract() [4/4]

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

四元数から別の四元数を減算した結果を格納する。

引数

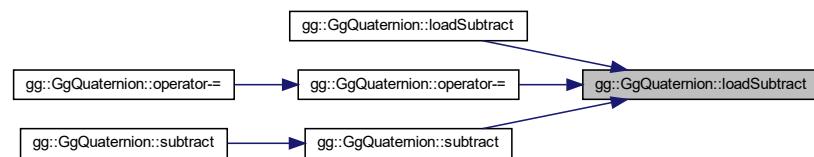
<i>x</i>	引く四元数の <i>x</i> 要素.
<i>y</i>	引く四元数の <i>y</i> 要素.
<i>z</i>	引く四元数の <i>z</i> 要素.
<i>w</i>	引く四元数の <i>w</i> 要素.

戻り値

(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 3597 行目に定義があります。

被呼び出し関係図:



8.11.3.61 multiply() [1/4]

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

四元数に別の四元数を乗算した結果を返す.

引数

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

戻り値

q を掛けた四元数.

gg.h の 3876 行目に定義があります。

8.11.3.62 multiply() [2/4]

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

四元数に別の四元数を乗算した結果を返す。

引数

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

戻り値

v を掛けた四元数。

[gg.h](#) の 3865 行目に定義がります。

8.11.3.63 multiply() [3/4]

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

四元数に別の四元数を乗算した結果を返す。

引数

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

戻り値

a を掛けた四元数。

[gg.h](#) の 3852 行目に定義がります。

8.11.3.64 multiply() [4/4]

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

四元数に別の四元数を乗算した結果を返す。

引数

<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 3840 行目に定義があります。

8.11.3.65 norm()

```
GLfloat gg::GgQuaternion::norm () const [inline]
```

四元数のノルムを求める.

戻り値

四元数のノルム.

gg.h の 3479 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.66 normalize()

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

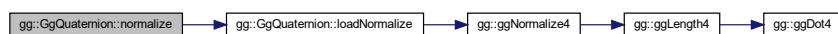
正規化する。

戻り値

正規化された四元数。

gg.h の 4389 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.67 operator*() [1/3]

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

gg.h の 4021 行目に定義があります。

呼び出し関係図:



8.11.3.68 operator*() [2/3]

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

gg.h の 4017 行目に定義があります。

8.11.3.69 operator*() [3/3]

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

gg.h の 4013 行目に定義があります。

被呼び出し関係図:

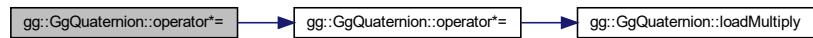


8.11.3.70 operator*=(()) [1/3]

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

gg.h の 3973 行目に定義があります。

呼び出し関係図:



8.11.3.71 operator*=() [2/3]

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

gg.h の 3969 行目に定義があります。

呼び出し関係図:



8.11.3.72 operator*=() [3/3]

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

gg.h の 3965 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

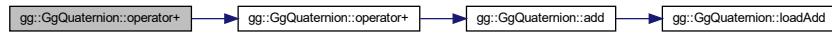


8.11.3.73 operator+() [1/3]

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

gg.h の 3997 行目に定義がります。

呼び出し関係図:

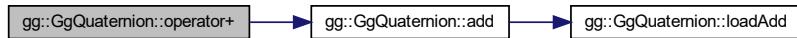


8.11.3.74 operator+() [2/3]

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

gg.h の 3993 行目に定義がります。

呼び出し関係図:

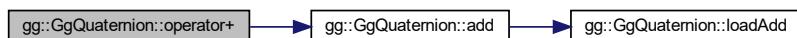


8.11.3.75 operator+() [3/3]

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

gg.h の 3989 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.76 operator+=() [1/3]

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

gg.h の 3949 行目に定義があります。

呼び出し関係図:



8.11.3.77 operator+=() [2/3]

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

gg.h の 3945 行目に定義があります。

呼び出し関係図:

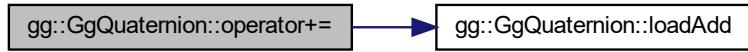


8.11.3.78 operator+=() [3/3]

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

gg.h の 3941 行目に定義があります。

呼び出し関係図:



8.11.3.79 operator-() [1/3]

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

gg.h の 4009 行目に定義があります。

呼び出し関係図:

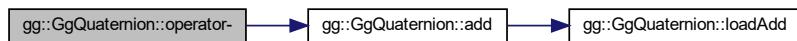


8.11.3.80 operator-() [2/3]

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

gg.h の 4005 行目に定義があります。

呼び出し関係図:

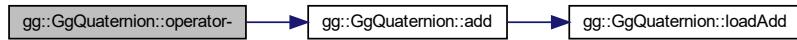


8.11.3.81 operator-() [3/3]

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

gg.h の 4001 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

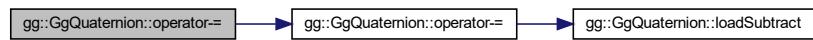


8.11.3.82 operator-=() [1/3]

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

gg.h の 3961 行目に定義があります。

呼び出し関係図:



8.11.3.83 operator=() [2/3]

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

gg.h の 3957 行目に定義があります。

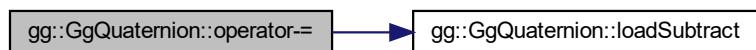
呼び出し関係図:

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

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

gg.h の 3953 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

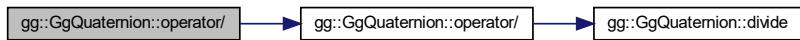


8.11.3.85 operator/() [1/3]

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

gg.h の 4033 行目に定義がります。

呼び出し関係図:

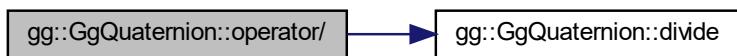


8.11.3.86 operator/() [2/3]

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

gg.h の 4029 行目に定義がります。

呼び出し関係図:



8.11.3.87 operator/() [3/3]

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

gg.h の 4025 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.88 operator/=() [1/3]

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

gg.h の 3985 行目に定義があります。

呼び出し関係図:



8.11.3.89 operator/=() [2/3]

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

gg.h の 3981 行目に定義があります。

呼び出し関係図:



8.11.3.90 operator/() [3/3]

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

gg.h の 3977 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.91 operator=() [1/2]

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

gg.h の 3937 行目に定義がります。

呼び出し関係図:



8.11.3.92 operator=() [2/2]

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

gg.h の 3933 行目に定義があります。

呼び出し関係図:

**8.11.3.93 rotate() [1/3]**

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

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

引数

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 4162 行目に定義があります。

呼び出し関係図:



8.11.3.94 `rotate()` [2/3]

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

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

引数

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

戻り値

回転した四元数.

gg.h の 4151 行目に定義があります。

呼び出し関係図:



8.11.3.95 `rotate()` [3/3]

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

四元数を (x, y, z) を軸として角度 a 回転した四元数を返す.

引数

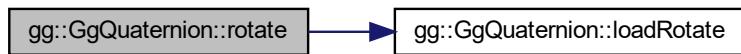
x	軸ベクトルの x 成分.
y	軸ベクトルの y 成分.
z	軸ベクトルの z 成分.
a	回転角.

戻り値

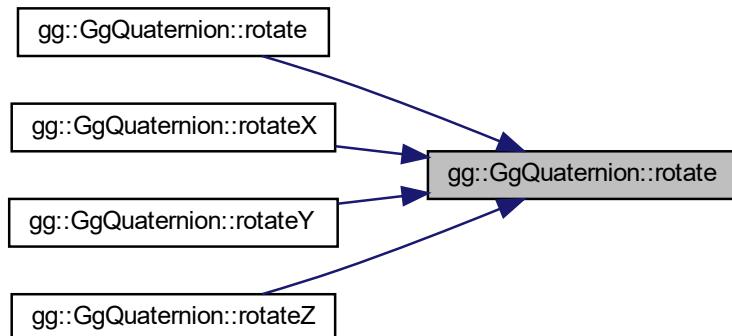
回転した四元数.

gg.h の 4138 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.96 rotateX()

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

四元数を x 軸中心に角度 a 回転した四元数を返す.

引数

<i>a</i>	回転角.
----------	------

戻り値

回転した四元数.

gg.h の 4173 行目に定義があります。

呼び出し関係図:



8.11.3.97 rotateY()

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

四元数を y 軸中心に角度 a 回転した四元数を返す.

引数

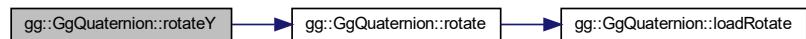
a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4184 行目に定義があります。

呼び出し関係図:



8.11.3.98 rotateZ()

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

四元数を z 軸中心に角度 a 回転した四元数を返す.

引数

a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4195 行目に定義があります。

呼び出し関係図:



8.11.3.99 slerp() [1/2]

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

球面線形補間の結果を返す.

引数

q	GgQuaternion 型の四元数.
t	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 4377 行目に定義があります。

8.11.3.100 slerp() [2/2]

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

球面線形補間の結果を返す.

引数

<i>a</i>	四元数を格納した 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 4363 行目に定義があります。

8.11.3.101 subtract() [1/4]

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

四元数から別の四元数を減算した結果を返す.

引数

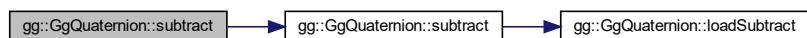
<i>q</i>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

q を引いた四元数.

gg.h の 3826 行目に定義があります。

呼び出し関係図:



8.11.3.102 subtract() [2/4]

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

四元数から別の四元数を減算した結果を返す.

引数

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

戻り値

v を引いた四元数.

gg.h の 3815 行目に定義がります。

呼び出し関係図:



8.11.3.103 subtract() [3/4]

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

四元数から別の四元数を減算した結果を返す.

引数

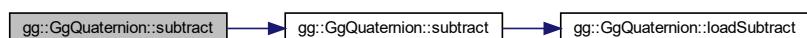
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a を引いた四元数.

gg.h の 3804 行目に定義がります。

呼び出し関係図:



8.11.3.104 subtract() [4/4]

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

四元数から別の四元数を減算した結果を返す。

引数

<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 3792 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.12 gg::GgShader クラス

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

公開メンバ関数

- `GgShader` (const std::string &vert, const std::string &frag="", const std::string &geom="", int nvarying=0, const char *const *varyings=nullptr)
- `GgShader` (const std::array< std::string, 3 > &files, int nvarying=0, const char *const *varyings=nullptr)
- `GgShader` (const GgShader &o)=delete
- virtual `~GgShader` ()
- `GgShader & operator=` (const GgShader &o)=delete
- void `use` () const
- void `unuse` () const
- GLuint `get` () const

8.12.1 詳解

シェーダの基底クラス。

覚え書き

シェーダのクラスはこのクラスを派生して作る。

gg.h の 6670 行目に定義があります。

8.12.2 構築子と解体子

8.12.2.1 GgShader() [1/3]

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

コンストラクタ。

引数

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

構造: Doxygen

gg.h の 6686 行目に定義がります。

8.12.2.2 GgShader() [2/3]

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

コンストラクタ。

引数

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

gg.h の 6704 行目に定義がります。

8.12.2.3 GgShader() [3/3]

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

コピー構造子は使用禁止。

8.12.2.4 ~GgShader()

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

デストラクタ。

gg.h の 6721 行目に定義がります。

8.12.3 関数詳解

8.12.3.1 get()

```
GLuint gg::GgShader::get ( ) const [inline]
```

シェーダのプログラム名を得る.

戻り値

シェーダのプログラム名.

gg.h の 6754 行目に定義があります。

8.12.3.2 operator=()

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

代入演算子は使用禁止.

8.12.3.3 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 6744 行目に定義があります。

8.12.3.4 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 6736 行目に定義があります。

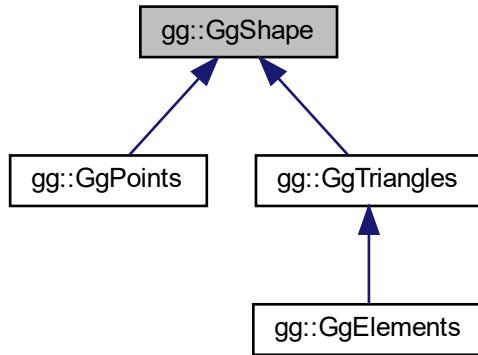
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.13 gg::GgShape クラス

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

gg::GgShape の継承関係図



公開 メンバ 関数

- `GgShape (GLenum mode=0)`
- `virtual ~GgShape ()`
- `GgShape (const GgShape &o)=delete`
- `GgShape & operator= (const GgShape &o)=delete`
- `const GLuint & get () const`
- `void setMode (GLenum mode)`
- `const GLenum & getMode () const`
- `virtual void draw (GLint first=0, GLsizei count=0) const`

8.13.1 詳解

形状データの基底クラス.

覚え書き

形状データのクラスはこのクラスを派生して作る. 基本図形の種類と頂点配列オブジェクトを保持する.

gg.h の 6074 行目に定義があります。

8.13.2 構築子と解体子

8.13.2.1 GgShape() [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 6089 行目に定義があります。

8.13.2.2 ~GgShape()

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

デストラクタ.

gg.h の 6099 行目に定義があります。

8.13.2.3 GgShape() [2/2]

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

コピー構造関数は使用禁止.

8.13.3 関数詳解

8.13.3.1 draw()

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

図形の描画, 派生クラスでこの手続きをオーバーライドする.

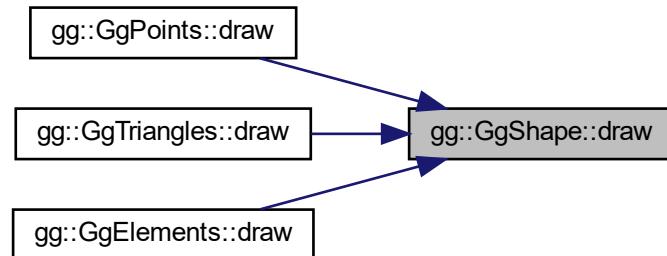
引数

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

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

gg.h の 6150 行目に定義があります。

被呼び出し関係図:



8.13.3.2 get()

`const GLuint& gg::GgShape::get () const [inline]`

頂点配列オブジェクト名を取り出す。

戻り値

頂点配列オブジェクト名。

`gg.h` の 6119 行目に定義があります。

被呼び出し関係図:



8.13.3.3 getMode()

`const GLenum& gg::GgShape::getMode () const [inline]`

基本図形の検査。

戻り値

この頂点配列オブジェクトの基本図形の種類。

`gg.h` の 6139 行目に定義があります。

8.13.3.4 operator=()

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

代入演算子は使用禁止。

8.13.3.5 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 6129 行目に定義がります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.14 gg::GgSimpleObj クラス

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

公開メンバ関数

- [GgSimpleObj](#) (const std::string &name, bool normalize=false)
- virtual [~GgSimpleObj](#) ()
デストラクタ。
- const [GgTriangles * get](#) () const
- virtual void [draw](#) (GLint first=0, GLsizei count=0) const

8.14.1 詳解

Wavefront OBJ 形式のファイル (Arrays 形式)。

gg.h の 7959 行目に定義がります。

8.14.2 構築子と解体子

8.14.2.1 GgSimpleObj()

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

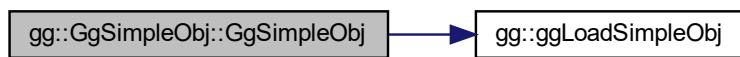
コンストラクタ。

引数

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

gg.cpp の 5802 行目に定義があります。

呼び出し関係図:



8.14.2.2 ~GgSimpleObj()

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

デストラクタ.

gg.h の 7982 行目に定義があります。

8.14.3 関数詳解

8.14.3.1 draw()

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

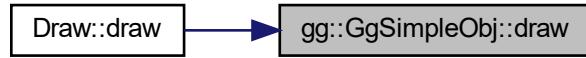
Wavefront OBJ 形式のデータを描画する手続き.

引数

<i>first</i>	描画する最初のバーツ番号.
<i>count</i>	描画するバーツの数, 0 なら全部のバーツを描く.

gg.cpp の 5830 行目に定義があります。

被呼び出し関係図:



8.14.3.2 get()

```
const GgTriangles* gg::GgSimpleObj::get() const [inline]
```

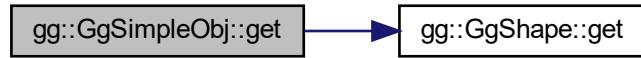
形状データの取り出し.

戻り値

GgTriangles 型の形状データのポインタ.

gg.h の 7991 行目に定義があります。

呼び出し関係図:



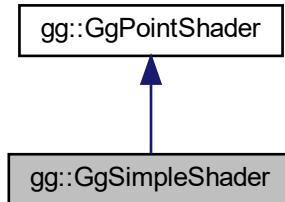
このクラス詳解は次のファイルから抽出されました:

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

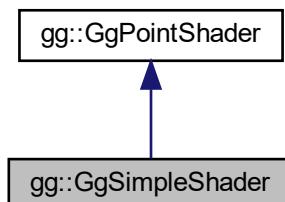
8.15 gg::GgSimpleShader クラス

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

gg::GgSimpleShader の継承関係図



gg::GgSimpleShader 連携図



クラス

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

公開メンバ関数

- [GgSimpleShader \(\)](#)
- [GgSimpleShader \(const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr\)](#)
- [GgSimpleShader \(const std::array< std::string, 3 > &files, GLint nvarying=0, const char *const *varyings=nullptr\)](#)
- [GgSimpleShader \(const GgSimpleShader &o\)](#)

- virtual ~GgSimpleShader ()
- GgSimpleShader & operator= (const GgSimpleShader &o)
- bool load (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)
- bool load (const std::array< std::string, 3 >&files, GLint nvarying=0, const char *const *const *varyings=nullptr)
- virtual void loadModelviewMatrix (const GLfloat *mv, const GLfloat *mn) const
- virtual void loadModelviewMatrix (const GgMatrix &mv, const GgMatrix &mn) const
- virtual void loadModelviewMatrix (const GLfloat *mv) const
- virtual void loadModelviewMatrix (const GgMatrix &mv) const
- virtual void loadMatrix (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn) const
- virtual void loadMatrix (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn) const
- virtual void loadMatrix (const GLfloat *mp, const GLfloat *mv) const
- virtual void loadMatrix (const GgMatrix &mp, const GgMatrix &mv) const
- void use () const
- void use (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn) const
- void use (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn) const
- void use (const GLfloat *mp, const GLfloat *mv) const
- void use (const GgMatrix &mp, const GgMatrix &mv) const
- void use (const LightBuffer *light, GLint i=0) const
- void use (const LightBuffer &light, GLint i=0) const
- void use (const GLfloat *mp, const GLfloat *mv, const GLfloat *mn, const LightBuffer *light, GLint i=0) const
- void use (const GgMatrix &mp, const GgMatrix &mv, const GgMatrix &mn, const LightBuffer &light, GLint i=0) const
- void use (const GLfloat *mp, const GLfloat *mv, const LightBuffer *light, GLint i=0) const
- void use (const GgMatrix &mp, const GgMatrix &mv, const LightBuffer &light, GLint i=0) const
- void use (const GLfloat *mp, const LightBuffer *light, GLint i=0) const
- void use (const GgMatrix &mp, const LightBuffer &light, GLint i=0) const

8.15.1 詳解

三角形に単純な陰影付けを行うシェーダ.

gg.h の 7018 行目に定義があります。

8.15.2 構築子と解体子

8.15.2.1 GgSimpleShader() [1/4]

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

コンストラクタ.

gg.h の 7035 行目に定義があります。

8.15.2.2 GgSimpleShader() [2/4]

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

コンストラクタ.

引数

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

gg.h の 7052 行目に定義がります。

8.15.2.3 GgSimpleShader() [3/4]

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

コンストラクタ.

引数

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

gg.h の 7070 行目に定義がります。

8.15.2.4 GgSimpleShader() [4/4]

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

コピーコンストラクタ.

gg.h の 7082 行目に定義がります。

8.15.2.5 ~GgSimpleShader()

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

デストラクタ.

gg.h の 7093 行目に定義がります。

8.15.3 関数詳解

8.15.3.1 load() [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

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

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 7139 行目に定義があります。

8.15.3.2 load() [2/2]

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

シェーダのソースプログラムの文字列からプログラムオブジェクトを作成する。

引数

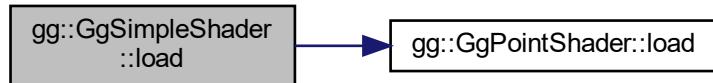
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列.
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用) .
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用).

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.cpp の 5785 行目に定義があります。

呼び出し関係図:



8.15.3.3 loadMatrix() [1/4]

```

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

```

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

引数

<i>mp</i>	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 7233 行目に定義があります。

呼び出し関係図:



8.15.3.4 loadMatrix() [2/4]

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

投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

引数

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

gg.h の 7211 行目に定義があります。

呼び出し関係図:



8.15.3.5 loadMatrix() [3/4]

```
virtual void gg::GgSimpleShader::loadMatrix (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline], [virtual]
```

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

引数

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

gg::GgPointShader を再実装しています。

gg.h の 7222 行目に定義があります。

8.15.3.6 loadMatrix() [4/4]

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

投影変換行列とモデルビュー変換行列と法線変換行列を設定する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.

gg.h の 7198 行目に定義があります。

8.15.3.7 loadModelviewMatrix() [1/4]

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

モデルビュー変換行列とそれから求めた法線変換行列を設定する。

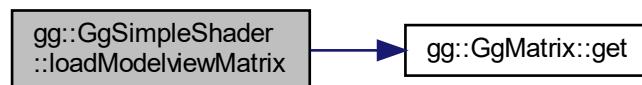
引数

<i>mv</i>	GgMatrix 型のモデルビュー変換行列.
-----------	------------------------

gg::GgPointShader を再実装しています。

gg.h の 7186 行目に定義があります。

呼び出し関係図:



8.15.3.8 loadModelviewMatrix() [2/4]

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

モデルビュー変換行列と法線変換行列を設定する。

引数

<i>mv</i>	GgMatrix 型のモデルビュー変換行列。
<i>mn</i>	GgMatrix 型のモデルビュー変換行列の法線変換行列。

gg.h の 7166 行目に定義があります。

呼び出し関係図:



8.15.3.9 loadModelviewMatrix() [3/4]

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

モデルビュー変換行列とそれから求めた法線変換行列を設定する。

引数

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

gg::GgPointShader を再実装しています。

gg.h の 7176 行目に定義があります。

8.15.3.10 loadModelviewMatrix() [4/4]

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

モデルビュー変換行列と法線変換行列を設定する。

引数

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

gg.h の 7154 行目に定義があります。

8.15.3.11 operator=()

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

代入演算子.

gg.h の 7100 行目に定義があります。

8.15.3.12 use() [1/13]

```
void gg::GgSimpleShader::use ( ) const [inline], [virtual]
```

シェーダプログラムの使用を開始する.

gg::GgPointShaderを再実装しています。

gg.h の 7726 行目に定義があります。

被呼び出し関係図:



8.15.3.13 use() [2/13]

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

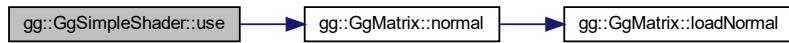
投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する.

引数

<i>mp</i>	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 7777 行目に定義があります。

呼び出し関係図:



8.15.3.14 use() [3/13]

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

投影変換行列とモデルビュー変換行列と法線変換行列を指定してシェーダプログラムの使用を開始する。

引数

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

gg.h の 7755 行目に定義があります。

呼び出し関係図:



8.15.3.15 `use()` [4/13]

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

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

引数

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

gg.h の 7841 行目に定義があります。

呼び出し関係図:



8.15.3.16 `use()` [5/13]

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

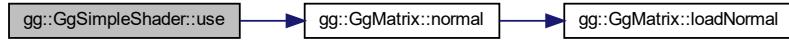
光源を指定し投影変換行列とモデルビュー変換行列を設定しモデルビュー変換行列から求めた法線変換行列を設定してシェーダプログラムの使用を開始する。

引数

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

gg.h の 7878 行目に定義があります。

呼び出し関係図:



8.15.3.17 use() [6/13]

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

光源を指定し投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GgMatrix 型の投影変換行列。
<i>light</i>	光源の特性の gg::LightBuffer 構造体。
<i>i</i>	光源データの uniform block のインデックス。

gg.h の 7911 行目に定義があります。

呼び出し関係図:



8.15.3.18 use() [7/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv ) const [inline]
```

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

引数

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

gg.h の 7766 行目に定義があります。

8.15.3.19 `use()` [8/13]

```
void gg::GgSimpleShader::use (
    const GLfloat * mp,
    const GLfloat * mv,
    const GLfloat * mn ) const [inline]
```

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

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.

gg.h の 7739 行目に定義があります。

8.15.3.20 `use()` [9/13]

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

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

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>mv</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列.
<i>mn</i>	GLfloat 型の 16 要素の配列変数に格納されたモデルビュー変換行列の法線変換行列.
<i>light</i>	光源の特性の <code>gg::LightBuffer</code> 構造体のポインタ.
<i>i</i>	光源データの uniform block のインデックス.

gg.h の 7817 行目に定義があります。

8.15.3.21 use() [10/13]

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

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

引数

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

gg.h の 7860 行目に定義があります。

8.15.3.22 use() [11/13]

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

光源を指定し投影変換行列を設定してシェーダプログラムの使用を開始する。

引数

<i>mp</i>	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
<i>light</i>	光源の特性の 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 7895 行目に定義があります。

8.15.3.23 use() [12/13]

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

光源を指定してシェーダプログラムの使用を開始する。

引数

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

`gg.h` の 7803 行目に定義があります。

8.15.3.24 `use()` [13/13]

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

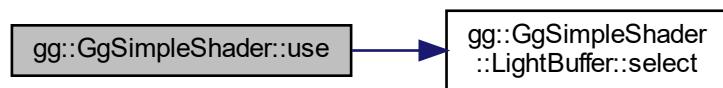
光源を指定してシェーダプログラムの使用を開始する。

引数

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

`gg.h` の 7788 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.16 `gg::GgTexture` クラス

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

公開メンバ関数

- `GgTexture` (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE)
- virtual ~`GgTexture` ()
- `GgTexture` (const `GgTexture` &o)=delete
- `GgTexture` & `operator=` (const `GgTexture` &o)=delete
- void `bind` () const
- void `unbind` () const
- const GLsizei & `getWidth` () const
- const GLsizei & `getHeight` () const
- void `getSize` (GLsizei *size) const
- const GLsizei * `getSize` () const
- const GLuint & `getTexture` () const

8.16.1 詳解

テクスチャ.

覚え書き

画像データを読み込んでテクスチャマップを作成する.

gg.h の 5168 行目に定義があります。

8.16.2 構築子と解体子

8.16.2.1 `GgTexture()` [1/2]

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

<code>image</code>	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.
<code>width</code>	テクスチャの横の画素数.
<code>height</code>	テクスチャの縦の画素数.
<code>format</code>	読み込む画像のフォーマット.
<code>type</code>	画像のデータ型.
<small>構築子 Doxygen</small>	テクスチャの内部フォーマット.
<code>wrap</code>	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 5189 行目に定義があります。

8.16.2.2 ~GgTexture()

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

デストラクタ.

gg.h の 5206 行目に定義があります。

8.16.2.3 GgTexture() [2/2]

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

コピー構造関数は使用禁止.

8.16.3 関数詳解

8.16.3.1 bind()

```
void gg::GgTexture::bind() const [inline]
```

テクスチャの使用開始(このテクスチャを使用する際に呼び出す).

gg.h の 5225 行目に定義があります。

8.16.3.2 getHeight()

```
const GLsizei& gg::GgTexture::getHeight() const [inline]
```

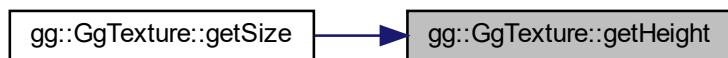
使用しているテクスチャの縦の画素数を取り出す.

戻り値

テクスチャの縦の画素数.

gg.h の 5253 行目に定義があります。

被呼び出し関係図:



8.16.3.3 getSize() [1/2]

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

使用しているテクスチャのサイズを取り出す。

戻り値

テクスチャのサイズを格納した配列へのポインタ。

gg.h の 5274 行目に定義があります。

8.16.3.4 getSize() [2/2]

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

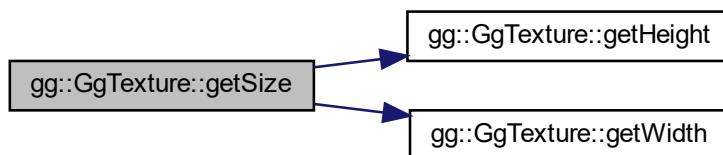
使用しているテクスチャのサイズを取り出す。

引数

size	テクスチャのサイズを格納する GLsizei 型の 2 要素の配列変数。
-------------	--------------------------------------

gg.h の 5263 行目に定義があります。

呼び出し関係図:



8.16.3.5 getTexture()

```
const GLuint& gg::GgTexture::getTexture ( ) const [inline]
```

使用しているテクスチャのテクスチャ名を得る。

戻り値

テクスチャ名.

gg.h の 5284 行目に定義があります。

8.16.3.6 getWidth()

```
const GLsizei& gg::GgTexture::getWidth () const [inline]
```

使用しているテクスチャの横の画素数を取り出す.

戻り値

テクスチャの横の画素数.

gg.h の 5243 行目に定義があります。

被呼び出し関係図:



8.16.3.7 operator=()

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

代入演算子は使用禁止.

8.16.3.8 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 5233 行目に定義があります。

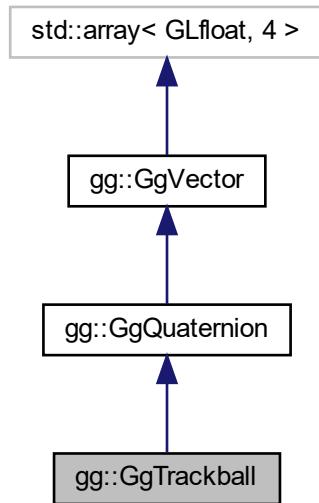
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

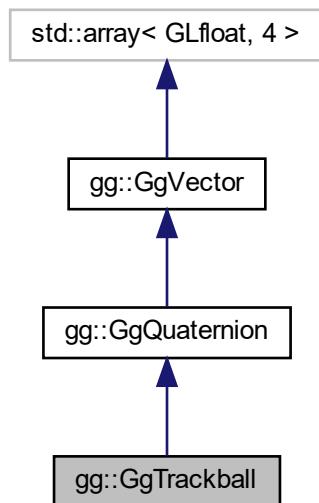
8.17 gg::GgTrackball クラス

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

gg::GgTrackball の継承関係図



gg::GgTrackball 連携図



公開メンバ関数

- `GgTrackball (const GgQuaternion &q=ggIdentityQuaternion())`
- `virtual ~GgTrackball ()`
- `GgTrackball & operator= (const GgQuaternion &q)`
- `void region (GLfloat w, GLfloat h)`
- `void region (int w, int h)`
- `void begin (GLfloat x, GLfloat y)`
- `void motion (GLfloat x, GLfloat y)`
- `void rotate (const GgQuaternion &q)`
- `void end (GLfloat x, GLfloat y)`
- `void reset (const GgQuaternion &q=ggIdentityQuaternion())`
- `const GLfloat * getStart () const`
- `const GLfloat & getStart (int direction) const`
- `void getStart (GLfloat *position) const`
- `const GLfloat * getScale () const`
- `const GLfloat getScale (int direction) const`
- `void getScale (GLfloat *factor) const`
- `const GgQuaternion & getQuaternion () const`
- `const GgMatrix & getMatrix () const`
- `const GLfloat * get () const`

8.17.1 詳解

簡易トラックボール処理。

gg.h の 4750 行目に定義がります。

8.17.2 構築子と解体子

8.17.2.1 GgTrackball()

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

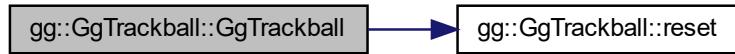
コンストラクタ。

引数

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

gg.h の 4765 行目に定義がります。

呼び出し関係図:



8.17.2.2 ~GgTrackball()

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

デストラクタ.

gg.h の 4773 行目に定義があります。

8.17.3 関数詳解

8.17.3.1 begin()

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

トラックボール処理を開始する.

引数

x	現在のマウスの x 座標.
y	現在のマウスの y 座標.

覚え書き

マウスのドラッグ開始時(マウスボタンを押したとき)に呼び出す.

gg.cpp の 3343 行目に定義があります。

8.17.3.2 end()

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

トラックボール処理を停止する。

引数

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

覚え書き

マウスのドラッグ終了時(マウスボタンを離したとき)に呼び出す。

gg.cpp の 3408 行目に定義があります。

8.17.3.3 get()

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

現在の回転の変換行列を取り出す。

戻り値

回転の変換を表す GLfloat 型の 16 要素の配列。

gg.h の 4945 行目に定義があります。

呼び出し関係図:



8.17.3.4 getMatrix()

```
const GgMatrix& gg::GgTrackball::getMatrix () const [inline]
```

現在の回転の変換行列を取り出す。

戻り値

回転の変換を表す `GgMatrix` 型の変換行列。

gg.h の 4935 行目に定義があります。

8.17.3.5 getQuaternion()

```
const GgQuaternion& gg::GgTrackball::getQuaternion () const [inline]
```

現在の回転の四元数を取り出す。

戻り値

回転の変換を表す `Quaternion` 型の四元数。

gg.h の 4925 行目に定義があります。

8.17.3.6 getScale() [1/3]

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

トランクボール処理の換算係数を取り出す。

戻り値

トランクボールの換算係数のポインタ。

gg.h の 4894 行目に定義があります。

8.17.3.7 getScale() [2/3]

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

トラン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 4914 行目に定義がります。

8.17.3.8 `getScale()` [3/3]

```
const GLfloat gg::GgTrackball::getScale (
    int direction ) const [inline]
```

トラックボール処理の換算係数を取り出す.

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向.
------------------	-----------------------

gg.h の 4904 行目に定義がります。

8.17.3.9 `getStart()` [1/3]

```
const GLfloat* gg::GgTrackball::getStart ( ) const [inline]
```

トラックボール処理の開始位置を取り出す.

戻り値

トラックボールの開始位置のポインタ.

gg.h の 4865 行目に定義がります。

8.17.3.10 `getStart()` [2/3]

```
void gg::GgTrackball::getStart (
    GLfloat * position ) const [inline]
```

トラックボール処理の開始位置を取り出す.

引数

<i>position</i>	トラックボールの開始位置を格納する 2 要素の配列.
-----------------	----------------------------

gg.h の 4883 行目に定義がります。

8.17.3.11 getStart() [3/3]

```
const GLfloat& gg::GgTrackball::getStart (
    int direction ) const [inline]
```

トラックボール処理の開始位置を取り出す。

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向.
------------------	-----------------------

gg.h の 4873 行目に定義がります。

8.17.3.12 motion()

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

回転の変換行列を計算する。

引数

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

覚え書き

マウスのドラッグ中に呼び出す。

gg.cpp の 3359 行目に定義がります。

呼び出し関係図:



8.17.3.13 operator=()

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

代入.

引数

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

gg.h の 4782 行目に定義があります。

呼び出し関係図:



8.17.3.14 region() [1/2]

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

トラックボール処理するマウスの移動範囲を指定する.

引数

<i>w</i>	領域の横幅.
<i>h</i>	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す.

gg.cpp の 3330 行目に定義があります。

被呼び出し関係図:



8.17.3.15 region() [2/2]

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

トラックボール処理するマウスの移動範囲を指定する。

引数

w	領域の横幅.
h	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す。

gg.h の 4808 行目に定義があります。

呼び出し関係図:



8.17.3.16 reset()

```
void gg::GgTrackball::reset (
    const GgQuaternion & q = ggIdentityQuaternion() )
```

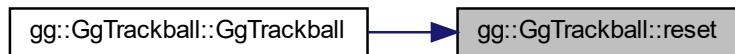
トラックボールをリセットする。

引数

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

gg.cpp の 3312 行目に定義がります。

被呼び出し関係図:



8.17.3.17 `rotate()`

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

トラックボールの回転角を修正する。

引数

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

gg.cpp の 3387 行目に定義がります。

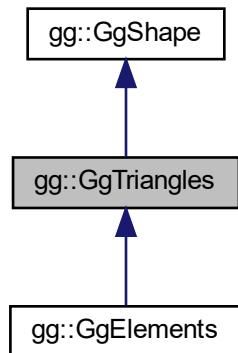
このクラス詳解は次のファイルから抽出されました:

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

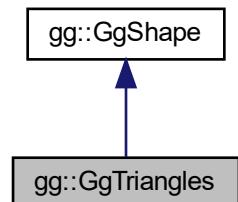
8.18 `gg::GgTriangles` クラス

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

gg::GgTriangles の継承関係図



gg::GgTriangles 連携図



公開 メンバ関数

- [GgTriangles \(GLenum mode=GL_TRIANGLES\)](#)
- [GgTriangles \(const GgVertex *vert, GLsizei count, GLenum mode=GL_TRIANGLES, GLenum usage=GL_STATIC_DRAW\)](#)
- virtual [~GgTriangles \(\)](#)
- const GLsizei & [getCount \(\) const](#)
- const GLuint & [getBuffer \(\) const](#)
- void [send \(const GgVertex *vert, GLint first=0, GLsizei count=0\) const](#)
- void [load \(const GgVertex *vert, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- virtual void [draw \(GLint first=0, GLsizei count=0\) const](#)

8.18.1 詳解

三角形で表した形状データ (Arrays 形式).

gg.h の 6315 行目に定義があります。

8.18.2 構築子と解体子

8.18.2.1 GgTriangles() [1/2]

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

コンストラクタ.

引数

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

gg.h の 6328 行目に定義がります。

8.18.2.2 GgTriangles() [2/2]

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

コンストラクタ.

引数

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

gg.h の 6341 行目に定義がります。

8.18.2.3 ~GgTriangles()

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

デストラクタ.

gg.h の 6355 行目に定義がります。

8.18.3 関数詳解

8.18.3.1 draw()

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

三角形の描画.

引数

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

[gg::GgShape](#)を再実装しています。

[gg::GgElements](#)で再実装されています。

gg.cpp の 5061 行目に定義があります。

呼び出し関係図:



8.18.3.2 getBuffer()

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

頂点属性を格納した頂点バッファオブジェクト名を取り出す.

戻り値

この図形の頂点属性を格納した頂点バッファオブジェクト名.

gg.h の 6374 行目に定義があります。

8.18.3.3 getCount()

```
const GLsizei& gg::GgTriangles::getCount ( ) const [inline]
```

データの数を取り出す.

戻り値

この図形の頂点属性の数(頂点数).

gg.h の 6364 行目に定義があります。

8.18.3.4 load()

```
void gg::GgTriangles::load (
    const GgVertex * vert,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW )
```

バッファオブジェクトを確保して頂点属性を格納する.

引数

<i>vert</i>	頂点属性が格納されてい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 5042 行目に定義があります。

8.18.3.5 send()

```
void gg::GgTriangles::send (
    const GgVertex * vert,
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

既存のバッファオブジェクトに頂点属性を転送する.

引数

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

gg.h の 6386 行目に定義がります。

このクラス詳解は次のファイルから抽出されました:

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

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

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

公開 メンバ関数

- [GgUniformBuffer \(\)](#)
- [GgUniformBuffer \(const T *data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [GgUniformBuffer \(const T &data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [virtual ~GgUniformBuffer \(\)](#)
- [const GLuint & getTarget \(\) const](#)
- [GLsizeiptr getStride \(\) const](#)
- [const GLsizei & getCount \(\) const](#)
- [const GLuint & getBuffer \(\) const](#)
- [void bind \(\) const](#)
- [void unbind \(\) const](#)
- [void * map \(\) const](#)
- [void * map \(GLint first, GLsizei count\) const](#)
- [void unmap \(\) const](#)
- [void load \(const T *data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [void load \(const T &data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [void send \(const GLvoid *data, GLint offset=0, GLsizei size=sizeof\(T\), GLint first=0, GLsizei count=0\) const](#)
- [void fill \(const GLvoid *data, GLint offset=0, GLsizei size=sizeof\(T\), GLint first=0, GLsizei count=0\) const](#)
- [void read \(GLvoid *data, GLint offset=0, GLsizei size=sizeof\(T\), GLint first=0, GLsizei count=0\) const](#)
- [void copy \(GLuint src_buffer, GLint src_first=0, GLint dst_first=0, GLsizei count=0\) const](#)

8.19.1 詳解

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

ユニフォームバッファオブジェクト.

覚え書き

ユニフォーム変数を格納するバッファオブジェクトの基底クラス.

gg.h の 5740 行目に定義がります。

8.19.2 構築子と解体子

8.19.2.1 GgUniformBuffer() [1/3]

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

コンストラクタ.

gg.h の 5743 行目に定義があります。

8.19.2.2 GgUniformBuffer() [2/3]

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

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

引数

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

gg.h の 5743 行目に定義があります。

8.19.2.3 GgUniformBuffer() [3/3]

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

ユニフォームバッファオブジェクトの全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 5743 行目に定義があります。

8.19.2.4 ~GgUniformBuffer()

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

デストラクタ.

gg.h の 5743 行目に定義があります。

8.19.3 関数詳解

8.19.3.1 bind()

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

ユニフォームバッファオブジェクトを結合する.

gg.h の 5828 行目に定義があります。

8.19.3.2 copy()

```
template<typename T >
void gg::GgUniformBuffer< T >::copy (
    GLuint src_buffer,
    GLint src_first = 0,
    GLint dst_first = 0,
    GLsizei count = 0 ) const [inline]
```

別のバッファオブジェクトからデータを複写する.

引数

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

gg.h の 6039 行目に定義があります。

8.19.3.3 fill()

```
template<typename T >
void gg::GgUniformBuffer< T >::fill (
    const GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトの全ブロックのメンバーを同じデータを格納する。

引数

<i>data</i>	格納するデータ.
<i>offset</i>	格納先のメンバのブロックの先頭からのバイトオフセット.
<i>size</i>	格納するデータの一箇あたりのバイト数.
<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 5956 行目に定義があります。

8.19.3.4 getBuffer()

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

ユニフォームバッファオブジェクト名を取り出す。

戻り値

このユニフォームバッファオブジェクト名。

gg.h の 5820 行目に定義があります。

8.19.3.5 getCount()

```
template<typename T >
const GLsizei& gg::GgUniformBuffer< T >::getCount () const [inline]
```

データの数を取り出す。

戻り値

このユニフォームバッファオブジェクトのデータの数。

gg.h の 5810 行目に定義があります。

8.19.3.6 getStride()

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

ユニフォームバッファオブジェクトのアライメントを考慮したデータの間隔を取り出す.

戻り値

このユニフォームバッファオブジェクトのデータの間隔.

gg.h の 5800 行目に定義があります。

8.19.3.7 getTarget()

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

ユニフォームバッファオブジェクトのターゲットを取り出す.

戻り値

このユニフォームバッファオブジェクトのターゲット.

gg.h の 5790 行目に定義があります。

8.19.3.8 load() [1/2]

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

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

引数

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

gg.h の 5897 行目に定義があります。

8.19.3.9 `load()` [2/2]

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

ユニフォームバッファオブジェクトを確保してブロックごとにデータを転送する。

引数

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

`gg.h` の 5878 行目に定義があります。

8.19.3.10 `map()` [1/2]

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

ユニフォームバッファオブジェクトをマップする。

戻り値

マップしたメモリの先頭のポインタ.

`gg.h` の 5846 行目に定義があります。

8.19.3.11 `map()` [2/2]

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

ユニフォームバッファオブジェクトの指定した範囲をマップする。

引数

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

戻り値

マップしたメモリの先頭のポインタ.

gg.h の 5858 行目に定義があります。

8.19.3.12 read()

```
template<typename T >
void gg::GgUniformBuffer< T >::read (
    GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトからデータを抽出する.

引数

<i>data</i>	抽出先の領域の先頭のポインタ.
<i>offset</i>	抽出元のユニフォームバッファオブジェクトのメンバのブロックの先頭からのバイトオフセット.
<i>size</i>	抽出するデータの一個あたりのバイト数.
<i>first</i>	抽出元のユニフォームバッファオブジェクトのブロックの先頭の番号.
<i>count</i>	抽出するデータの数(0ならユニフォームバッファオブジェクト全体).

gg.h の 5991 行目に定義があります。

8.19.3.13 send()

```
template<typename T >
void gg::GgUniformBuffer< T >::send (
    const GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトを確保してユニフォームバッファオブジェクトのブロックごとのメンバを同じデータで埋める.

引数

<i>data</i>	データが格納されている領域の先頭のポインタ.
<i>offset</i>	格納先のメンバのブロックの先頭からのバイトオフセット.
<i>size</i>	格納するデータの一個あたりのバイト数.
<i>first</i>	格納先のバッファオブジェクトのブロックの先頭の番号.
<i>count</i>	格納するデータの数.

gg.h の 5918 行目に定義があります。

8.19.3.14 unbind()

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

ユニフォームバッファオブジェクトを解放する。

gg.h の 5836 行目に定義があります。

8.19.3.15 unmap()

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

バッファオブジェクトをアンマップする。

gg.h の 5866 行目に定義があります。

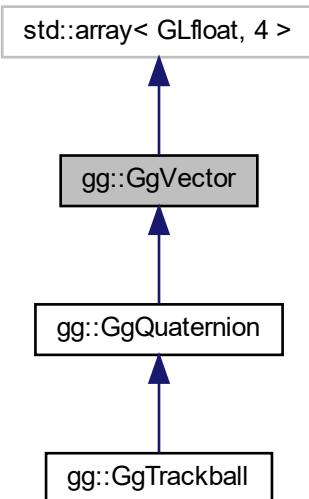
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

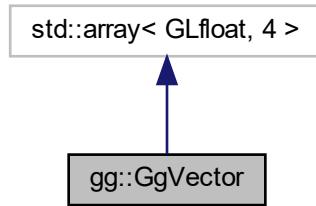
8.20 gg::GgVector クラス

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

gg::GgVector の継承関係図



gg::GgVector 連携図



公開 メンバ関数

- `GgVector ()`
- `constexpr GgVector (GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3)`
- `constexpr GgVector (GLfloat c)`
- `constexpr GgVector (const GLfloat *a)`
- `GgVector operator+ (const GgVector &v) const`
- `GgVector operator+ (GLfloat c) const`
- `GgVector & operator+= (const GgVector &v)`
- `GgVector & operator+= (GLfloat c)`
- `GgVector operator- (const GgVector &v) const`
- `GgVector operator- (GLfloat c) const`
- `GgVector & operator-= (const GgVector &v)`
- `GgVector & operator-= (GLfloat c)`
- `GgVector operator* (const GgVector &v) const`
- `GgVector operator* (GLfloat c) const`
- `GgVector & operator*= (const GgVector &v)`
- `GgVector & operator*= (GLfloat c)`
- `GgVector operator/ (const GgVector &v) const`
- `GgVector & operator/= (GgVector &v)`
- `GgVector operator/ (GLfloat c) const`
- `GgVector & operator/= (GLfloat c)`
- `GLfloat dot3 (const GgVector &v) const`
- `GLfloat length3 () const`
- `GLfloat distance3 (const GgVector &v) const`
- `GgVector normalize3 () const`
- `GLfloat dot4 (const GgVector &v) const`
- `GLfloat length4 () const`
- `GLfloat distance4 (const GgVector &v) const`
- `GgVector normalize4 () const`

8.20.1 詳解

4 要素の単精度実数の配列。

gg.h の 1554 行目に定義があります。

8.20.2 構築子と解体子

8.20.2.1 GgVector() [1/4]

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

コンストラクタ。

gg.h の 1561 行目に定義があります。

8.20.2.2 GgVector() [2/4]

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

コンストラクタ。

引数

<i>v0</i>	GLfloat 型の値。
<i>v1</i>	GLfloat 型の値。
<i>v2</i>	GLfloat 型の値。
<i>v3</i>	GLfloat 型の値。

gg.h の 1573 行目に定義があります。

8.20.2.3 GgVector() [3/4]

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

コンストラクタ。

引数

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

gg.h の 1583 行目に定義があります。

8.20.2.4 GgVector() [4/4]

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

コンストラクタ.

引数

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

gg.h の 1593 行目に定義があります。

8.20.3 関数詳解

8.20.3.1 distance3()

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

GgVector 型の 3 要素の距離.

引数

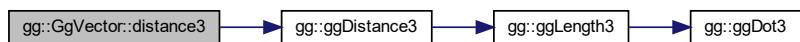
<i>v</i>	GgVector 型のベクトル.
----------	------------------

戻り値

オブジェクトと *v* の 3 要素の距離.

gg.h の 1833 行目に定義があります。

呼び出し関係図:



8.20.3.2 distance4()

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

GgVector 型の 4 要素の距離.

引数

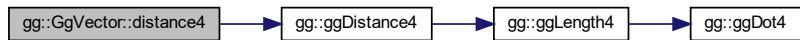
v	GgVector 型のベクトル.
---	------------------

戻り値

オブジェクトと v の 4 要素の距離.

gg.h の 1877 行目に定義があります。

呼び出し関係図:



8.20.3.3 dot3()

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

GgVector 型の 3 要素の内積.

引数

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

戻り値

オブジェクトと v のそれぞれの 3 要素の内積.

gg.h の 1812 行目に定義があります。

呼び出し関係図:



8.20.3.4 dot4()

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

GgVector 型の 4 要素の内積.

引数

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

戻り値

オブジェクトと v のそれぞれの 4 要素の内積.

gg.h の 1856 行目に定義があります。

呼び出し関係図:



8.20.3.5 length3()

```
GLfloat gg::GgVector::length3 ( ) const [inline]
```

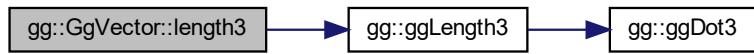
GgVector 型の 3 要素の長さ.

戻り値

オブジェクトの 4 要素の長さ.

gg.h の 1822 行目に定義があります。

呼び出し関係図:



8.20.3.6 length4()

`GLfloat gg::GgVector::length4 () const [inline]`

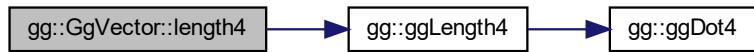
`GgVector` 型の 4 要素の長さ.

戻り値

オブジェクトの 4 要素の長さ.

gg.h の 1866 行目に定義があります。

呼び出し関係図:



8.20.3.7 normalize3()

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

`GgVector` 型の 4 要素の正規化.

引数

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

gg.h の 1843 行目に定義があります。

呼び出し関係図:



8.20.3.8 normalize4()

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

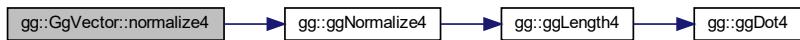
`GgVector` 型の 4 要素の正規化。

引数

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

gg.h の 1887 行目に定義があります。

呼び出し関係図:



8.20.3.9 operator*() [1/2]

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

`GgVector` 型の積を返す。

引数

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

戻り値

オブジェクトの各要素と `v` の各要素の要素ごとの積.

`gg.h` の 1708 行目に定義があります。

8.20.3.10 `operator*()` [2/2]

```
GgVector gg::GgVector::operator* (
    GLfloat c ) const [inline]
```

`GgVector` 型の各要素にスカラーを乗じた積を返す.

引数

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

戻り値

`a` の各要素に `c` を乗じた積.

`gg.h` の 1720 行目に定義があります。

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

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

`GgVector` 型を乗算する.

引数

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

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ乗算したオブジェクトの参照.

gg.h の 1730 行目に定義があります。

8.20.3.12 operator*=() [2/2]

```
GgVector& gg::GgVector::operator*=(  
    GLfloat c) [inline]
```

GgVector 型の各要素にスカラーを乗じる。

引数

c	GgVector 型のベクトル。
---	-------------------------

戻り値

オブジェクトの各要素に c を乗じたオブジェクトの参照。

gg.h の 1745 行目に定義があります。

8.20.3.13 operator+() [1/2]

```
GgVector gg::GgVector::operator+ (  
    const GgVector & v) const [inline]
```

GgVector 型の和を返す。

引数

v	GgVector 型のベクトル。
---	-------------------------

戻り値

オブジェクトの各要素と v の各要素の要素ごとの和。

gg.h の 1604 行目に定義があります。

8.20.3.14 operator+() [2/2]

```
GgVector gg::GgVector::operator+ (  
    GLfloat c) const [inline]
```

GgVector 型の各要素にスカラーを足した和を返す。

引数

<code>c</code>	GLfloat 型の値.
----------------	--------------

戻り値

`a` の各要素に `b` を足した和.

gg.h の 1615 行目に定義があります。

8.20.3.15 operator+=() [1/2]

```
GgVector& gg::GgVector::operator+= (
    const GgVector & v ) [inline]
```

`GgVector` 型を加算する.

引数

<code>v</code>	GgVector 型の変数.
----------------	----------------

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ加算したオブジェクトの参照.

gg.h の 1626 行目に定義があります。

8.20.3.16 operator+=() [2/2]

```
GgVector& gg::GgVector::operator+= (
    GLfloat c ) [inline]
```

`GgVector` 型の各要素にスカラーを加算する.

引数

<code>c</code>	GLfloat 型の値.
----------------	--------------

戻り値

オブジェクトの各要素に `c` を足したオブジェクトの参照.

gg.h の 1641 行目に定義があります。

8.20.3.17 operator-() [1/2]

```
GgVector gg::GgVector::operator- (
    const GgVector & v ) const [inline]
```

GgVector 型の差を返す.

引数

v	GgVector 型の変数.
---	----------------

戻り値

オブジェクトの各要素と v の各要素の要素ごとの差.

gg.h の 1656 行目に定義があります。

8.20.3.18 operator-() [2/2]

```
GgVector gg::GgVector::operator- (
    GLfloat c ) const [inline]
```

GgVector 型の各要素からスカラーを引いた差を返す.

引数

c	GLfloat 型の変数.
---	---------------

戻り値

オブジェクトの各要素から c を引いた差.

gg.h の 1667 行目に定義があります。

8.20.3.19 operator-=() [1/2]

```
GgVector& gg::GgVector::operator-= (
    const GgVector & v ) [inline]
```

GgVector 型を減算する.

引数

<code>v</code>	<code>GgVector</code> 型の変数.
----------------	-----------------------------

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ減算したオブジェクトの参照.

`gg.h` の 1678 行目に定義があります。

8.20.3.20 operator-() [2/2]

```
GgVector& gg::GgVector::operator-=
(GLfloat c) [inline]
```

`GgVector` 型の各要素からスカラーを引く.

引数

<code>c</code>	<code>GLfloat</code> 型の変数.
----------------	----------------------------

戻り値

オブジェクトの各要素から `c` を引いたオブジェクトの参照.

`gg.h` の 1693 行目に定義があります。

8.20.3.21 operator/() [1/2]

```
GgVector gg::GgVector::operator/
const GgVector & v) const [inline]
```

`GgVector` 型の商を返す.

引数

<code>v</code>	<code>GgVector</code> 型の変数.
----------------	-----------------------------

戻り値

オブジェクトの各要素を `v` の各要素で要素ごとに割った結果.

`gg.h` の 1760 行目に定義があります。

8.20.3.22 operator/() [2/2]

```
GgVector gg::GgVector::operator/ (
    GLfloat c ) const [inline]
```

GgVector 型の各要素をスカラーで割った商を返す。

引数

c	GLfloat 型の値.
---	--------------

戻り値

オブジェクトの各要素を c で割った商。

gg.h の 1786 行目に定義があります。

8.20.3.23 operator/=(()) [1/2]

```
GgVector& gg::GgVector::operator/= (
    GgVector & v ) [inline]
```

GgVector 型を除算する。

引数

v	GgVector 型の変数.
---	----------------

戻り値

オブジェクトの各要素に v の各要素をそれぞれ乗算したオブジェクトの参照。

gg.h の 1771 行目に定義があります。

8.20.3.24 operator/=(()) [2/2]

```
GgVector& gg::GgVector::operator/= (
    GLfloat c ) [inline]
```

GgVector 型の各要素をスカラーで割る。

引数

<code>c</code>	GLfloat 型の変数。
----------------	---------------

戻り値

オブジェクトの各要素を `c` で割ったオブジェクトの参照。

`gg.h` の 1797 行目に定義があります。

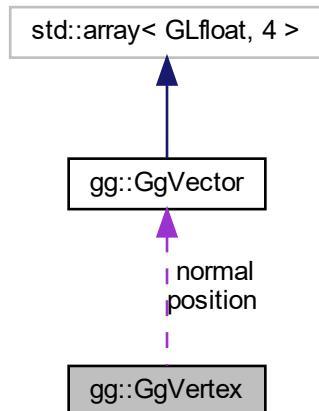
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.21 gg::GgVertex 構造体

```
#include <gg.h>
```

`gg::GgVertex` 連携図



公開メンバ関数

- [GgVertex \(\)](#)
- [GgVertex \(const GgVector &pos, const GgVector &norm\)](#)
- [GgVertex \(GLfloat px, GLfloat py, GLfloat pz, GLfloat nx, GLfloat ny, GLfloat nz\)](#)
- [GgVertex \(const GLfloat *pos, const GLfloat *norm\)](#)

公開変数類

- [GgVector position](#)
位置.
- [GgVector normal](#)
法線.

8.21.1 詳解

三角形の頂点データ.

gg.h の 6254 行目に定義がります。

8.21.2 構築子と解体子

8.21.2.1 GgVertex() [1/4]

```
gg::GgVertex::GgVertex ( ) [inline]
```

コンストラクタ.

gg.h の 6265 行目に定義がります。

8.21.2.2 GgVertex() [2/4]

```
gg::GgVertex::GgVertex ( const GgVector & pos, const GgVector & norm ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	GgVector 型の位置データ.
<i>norm</i>	GgVector 型の法線データ.

gg.h の 6275 行目に定義がります。

8.21.2.3 GgVertex() [3/4]

```
gg::GgVertex::GgVertex ( GLfloat px,
```

```
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 6291 行目に定義がります。

8.21.2.4 GgVertex() [4/4]

```
gg::GgVertex::GgVertex (
    const GLfloat * pos,
    const GLfloat * norm ) [inline]
```

コンストラクタ.

引数

<i>pos</i>	3 要素の 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 6306 行目に定義がります。

8.21.3 メンバ詳解

8.21.3.1 normal

[GgVector](#) gg::GgVertex::normal

法線.

gg.h の 6260 行目に定義がります。

8.21.3.2 position

`GgVector gg::GgVertex::position`

位置.

gg.h の 6257 行目に定義があります。

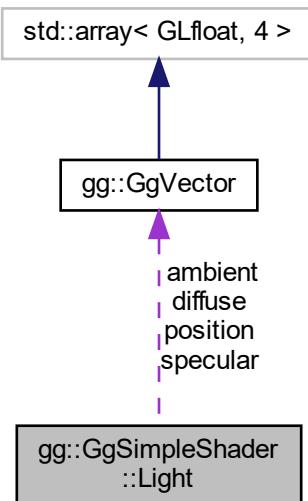
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

8.22 gg::GgSimpleShader::Light 構造体

#include <gg.h>

gg::GgSimpleShader::Light 連携図



公開変数類

- `GgVector ambient`
光源強度の環境光成分.
- `GgVector diffuse`
光源強度の拡散反射光成分.
- `GgVector specular`
光源強度の鏡面反射光成分.
- `GgVector position`
光源の位置.

8.22.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する光源データ.

gg.h の 7241 行目に定義があります。

8.22.2 メンバ詳解

8.22.2.1 ambient

`GgVector gg::GgSimpleShader::Light::ambient`

光源強度の環境光成分.

gg.h の 7243 行目に定義があります。

8.22.2.2 diffuse

`GgVector gg::GgSimpleShader::Light::diffuse`

光源強度の拡散反射光成分.

gg.h の 7244 行目に定義があります。

8.22.2.3 position

`GgVector gg::GgSimpleShader::Light::position`

光源の位置.

gg.h の 7246 行目に定義があります。

8.22.2.4 specular

`GgVector gg::GgSimpleShader::Light::specular`

光源強度の鏡面反射光成分.

gg.h の 7245 行目に定義があります。

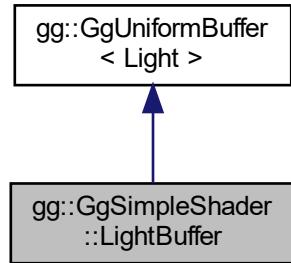
この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

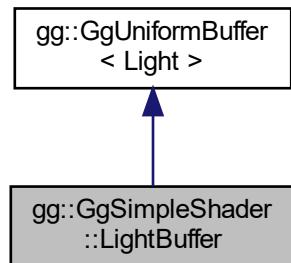
8.23 gg::GgSimpleShader::LightBuffer クラス

```
#include <gg.h>
```

gg::GgSimpleShader::LightBuffer の継承関係図



gg::GgSimpleShader::LightBuffer 連携図



公開メンバ関数

- `LightBuffer (const Light *light=nullptr, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `LightBuffer (const Light &light, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~LightBuffer ()`
- `void loadAmbient (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GLfloat *ambient, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GgVector &ambient, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GgVector &diffuse, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GLfloat *diffuse, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (const GgVector &specular, GLint first=0, GLsizei count=1) const`

- void **loadSpecular** (const GLfloat *specular, GLint first=0, GLsizei count=1) const
- void **loadColor** (const **Light** &color, GLint first=0, GLsizei count=1) const
- void **loadPosition** (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const **GgVector** &position, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const GLfloat *position, GLint first=0, GLsizei count=1) const
- void **loadPosition** (const **GgVector** *position, GLint first=0, GLsizei count=1) const
- void **load** (const **Light** *light, GLint first=0, GLsizei count=1) const
- void **load** (const **Light** &light, GLint first=0, GLsizei count=1) const
- void **select** (GLint i=0) const

8.23.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する光源データのユニフォームバッファオブジェクト。

gg.h の 7252 行目に定義があります。

8.23.2 構築子と解体子

8.23.2.1 LightBuffer() [1/2]

```
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 7264 行目に定義があります。

8.23.2.2 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 7280 行目に定義があります。

8.23.2.3 ~LightBuffer()

```
virtual gg::GgSimpleShader::LightBuffer::~LightBuffer () [inline], [virtual]
```

デストラクタ.

gg.h の 7292 行目に定義があります。

8.23.3 関数詳解

8.23.3.1 load() [1/2]

```
void gg::GgSimpleShader::LightBuffer::load (
    const Light & light,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の色と位置を設定する.

引数

<i>light</i>	光源の特性の GgSimpleShader::Light 構造体.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7484 行目に定義があります。

8.23.3.2 load() [2/2]

```
void gg::GgSimpleShader::LightBuffer::load (
    const Light * light,
```

```
GLint first = 0,
GLsizei count = 1 ) const [inline]
```

光源の色と位置を設定する。

引数

<i>light</i>	光源の特性の GgSimpleShader::Light 構造体のポインタ.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7472 行目に定義があります。

8.23.3.3 loadAmbient() [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    const GgVector & ambient,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の環境光成分を設定する。

引数

<i>ambient</i>	光源の強度の環境光成分を格納した GgVector 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5387 行目に定義があります。

8.23.3.4 loadAmbient() [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    const GLfloat * ambient,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の環境光成分を設定する。

引数

<i>ambient</i>	光源の強度の環境光成分を格納した GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7318 行目に定義がります。

8.23.3.5 loadAmbient() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の環境光成分を設定する。

引数

<i>r</i>	光源の強度の環境光成分の赤成分.
<i>g</i>	光源の強度の環境光成分の緑成分.
<i>b</i>	光源の強度の環境光成分の青成分.
<i>a</i>	光源の強度の拡散反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5361 行目に定義がります。

8.23.3.6 loadColor()

```
void gg::GgSimpleShader::LightBuffer::loadColor (
    const Light & color,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の色を設定するが位置は変更しない。

引数

<i>color</i>	光源の特性の 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 5514 行目に定義がります。

8.23.3.7 loadDiffuse() [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    const GgVector & diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の拡散反射光成分を設定する。

引数

<i>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 5439 行目に定義があります。

8.23.3.8 loadDiffuse() [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    const GLfloat * diffuse,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の拡散反射光成分を設定する。

引数

<i>diffuse</i>	光源の強度の拡散反射光成分を格納した GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7364 行目に定義があります。

8.23.3.9 loadDiffuse() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadDiffuse (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の拡散反射光成分を設定する。

引数

<i>r</i>	光源の強度の拡散反射光成分の赤成分.
<i>g</i>	光源の強度の拡散反射光成分の緑成分.
<i>b</i>	光源の強度の拡散反射光成分の青成分.
<i>a</i>	光源の強度の拡散反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5413 行目に定義があります。

8.23.3.10 loadPosition() [1/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GgVector & position,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の位置を設定する.

引数

<i>position</i>	光源の位置の同次座標を格納した GgVector 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5568 行目に定義があります。

8.23.3.11 loadPosition() [2/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GgVector * position,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の位置を設定する.

引数

<i>position</i>	光源の位置の同次座標を格納した GgVector 型の配列.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.h の 7460 行目に定義がります。

8.23.3.12 loadPosition() [3/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    const GLfloat * position,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の位置を設定する。

引数

<i>position</i>	光源の位置の同次座標を格納した <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 7447 行目に定義がります。

8.23.3.13 loadPosition() [4/4]

```
void gg::GgSimpleShader::LightBuffer::loadPosition (
    GLfloat x,
    GLfloat y,
    GLfloat z,
    GLfloat w = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の位置を設定する。

引数

<i>x</i>	光源の位置の x 座標。
<i>y</i>	光源の位置の y 座標。
<i>z</i>	光源の位置の z 座標。
<i>w</i>	光源の位置の w 座標, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5542 行目に定義がります。

8.23.3.14 loadSpecular() [1/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    const GgVector & specular,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>specular</i>	光源の強度の鏡面反射光成分を格納した GgVector 型の変数.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5491 行目に定義があります。

8.23.3.15 loadSpecular() [2/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    const GLfloat * specular,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>specular</i>	光源の強度の鏡面反射光成分を格納した 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 7401 行目に定義があります。

8.23.3.16 loadSpecular() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadSpecular (
    GLfloat r,
    GLfloat g,
    GLfloat b,
    GLfloat a = 1.0f,
    GLint first = 0,
    GLsizei count = 1 ) const
```

光源の強度の鏡面反射光成分を設定する。

引数

<i>r</i>	光源の強度の鏡面反射光成分の赤成分.
<i>g</i>	光源の強度の鏡面反射光成分の緑成分.
<i>b</i>	光源の強度の鏡面反射光成分の青成分.
<i>a</i>	光源の強度の鏡面反射光成分の不透明度, デフォルトは 1.
<i>first</i>	値を設定する光源データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5465 行目に定義があります。

8.23.3.17 select()

```
void gg::GgSimpleShader::LightBuffer::select (
    GLint i = 0 ) const [inline]
```

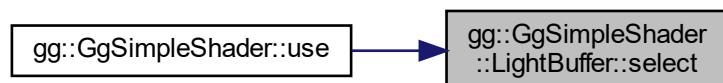
光源を選択する.

引数

<i>i</i>	光源データの uniform block のインデックス.
----------	-------------------------------

gg.h の 7494 行目に定義があります。

被呼び出し関係図:



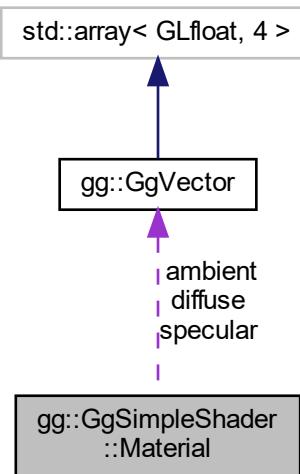
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

8.24 gg::GgSimpleShader::Material 構造体

```
#include <gg.h>
```

gg::GgSimpleShader::Material 連携図



公開変数類

- [GgVector ambient](#)
環境光に対する反射係数.
- [GgVector diffuse](#)
拡散反射係数.
- [GgVector specular](#)
鏡面反射係数.
- [GLfloat shininess](#)
輝き係数.

8.24.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する材質データ.

gg.h の 7505 行目に定義があります。

8.24.2 メンバ詳解

8.24.2.1 ambient

[GgVector gg::GgSimpleShader::Material::ambient](#)

環境光に対する反射係数.

gg.h の 7507 行目に定義があります。

8.24.2.2 diffuse

```
GgVector gg::GgSimpleShader::Material::diffuse
```

拡散反射係数.

gg.h の 7508 行目に定義があります。

8.24.2.3 shininess

```
GLfloat gg::GgSimpleShader::Material::shininess
```

輝き係数.

gg.h の 7510 行目に定義があります。

8.24.2.4 specular

```
GgVector gg::GgSimpleShader::Material::specular
```

鏡面反射係数.

gg.h の 7509 行目に定義があります。

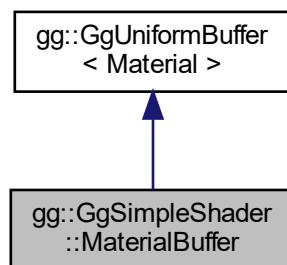
この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

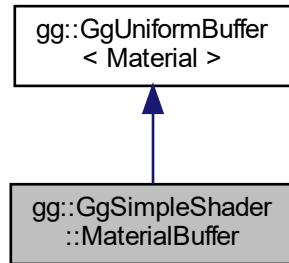
8.25 gg::GgSimpleShader::MaterialBuffer クラス

```
#include <gg.h>
```

gg::GgSimpleShader::MaterialBuffer の継承関係図



gg::GgSimpleShader::MaterialBuffer 連携図



公開メンバ関数

- `MaterialBuffer (const Material *material=nullptr, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `MaterialBuffer (const Material &material, GLsizei count=1, GLenum usage=GL_STATIC_DRAW)`
- `virtual ~MaterialBuffer ()`
- `void loadAmbient (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbient (const GLfloat *ambient, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadDiffuse (const GLfloat *diffuse, GLint first=0, GLsizei count=1) const`
- `void loadAmbientAndDiffuse (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadAmbientAndDiffuse (const GLfloat *color, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (GLfloat r, GLfloat g, GLfloat b, GLfloat a=1.0f, GLint first=0, GLsizei count=1) const`
- `void loadSpecular (const GLfloat *specular, GLint first=0, GLsizei count=1) const`
- `void loadShininess (GLfloat shininess, GLint first=0, GLsizei count=1) const`
- `void loadShininess (const GLfloat *shininess, GLint first=0, GLsizei count=1) const`
- `void load (const Material *material, GLint first=0, GLsizei count=1) const`
- `void load (const Material &material, GLint first=0, GLsizei count=1) const`
- `void select (GLint i=0) const`

8.25.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する材質データのユニフォームバッファオブジェクト。gg.h の 7516 行目に定義があります。

8.25.2 構築子と解体子

8.25.2.1 MaterialBuffer() [1/2]

```

gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material * material = nullptr,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
  
```

デフォルトコンストラクタ。

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データのポインタ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

gg.h の 7528 行目に定義があります。

8.25.2.2 MaterialBuffer() [2/2]

```
gg::GgSimpleShader::MaterialBuffer::MaterialBuffer (
    const Material & material,
    GLsizei count = 1,
    GLenum usage = GL_STATIC_DRAW ) [inline]
```

同じデータで埋めるコンストラクタ.

引数

<i>material</i>	<code>GgSimpleShader::Material</code> 型の材質データ.
<i>count</i>	バッファ中の <code>GgSimpleShader::Material</code> 型の材質データの数.
<i>usage</i>	バッファの使い方のパターン, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される.

gg.h の 7544 行目に定義があります。

8.25.2.3 ~MaterialBuffer()

```
virtual gg::GgSimpleShader::MaterialBuffer::~MaterialBuffer () [inline], [virtual]
```

デストラクタ.

gg.h の 7556 行目に定義があります。

8.25.3 関数詳解

8.25.3.1 load() [1/2]

```
void gg::GgSimpleShader::MaterialBuffer::load (
    const Material & material,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

材質を設定する.

引数

<i>material</i>	光源の特性の GgSimpleShader::Material 構造体.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7705 行目に定義があります。

8.25.3.2 load() [2/2]

```
void gg::GgSimpleShader::MaterialBuffer::load (
    const Material * material,
    GLint first = 0,
    GLsizei count = 1 ) const [inline]
```

材質を設定する.

引数

<i>material</i>	光源の特性の GgSimpleShader::Material 構造体のポインタ.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7693 行目に定義があります。

8.25.3.3 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 7582 行目に定義があります。

8.25.3.4 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 5594 行目に定義がります。

8.25.3.5 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 <code>GLfloat</code> 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5678 行目に定義がります。

8.25.3.6 loadAmbientAndDiffuse() [2/2]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse (
    GLfloat r,
```

```
GLfloat r,
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 5652 行目に定義があります。

8.25.3.7 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 GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7610 行目に定義があります。

8.25.3.8 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.
<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 5623 行目に定義があります。

8.25.3.9 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 5768 行目に定義があります。

8.25.3.10 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 5747 行目に定義がります。

8.25.3.11 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 GLfloat 型の 4 要素の配列変数.
<i>first</i>	値を設定する材質データの最初の番号, デフォルトは 0.
<i>count</i>	値を設定する材質データの数, デフォルトは 1.

gg.h の 7662 行目に定義がります。

8.25.3.12 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 5721 行目に定義がります。

8.25.3.13 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 7715 行目に定義がります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)
- [gg.cpp](#)

8.26 Menu クラス

```
#include <Menu.h>
```

公開メンバ関数

- [Menu \(const Config &config\)](#)
- [Menu \(const Menu &menu\)=delete](#)
- [virtual ~Menu \(\)](#)
- [Menu & operator= \(const Menu &menu\)=delete](#)
- [void draw \(\)](#)

フレンド

- [class Draw](#)

8.26.1 詳解

メニューの描画

Menu.h の 20 行目に定義がります。

8.26.2 構築子と解体子

8.26.2.1 Menu() [1/2]

```
Menu::Menu ( const Config & config )
```

コンストラクタ

Menu.cpp の 19 行目に定義がります。

8.26.2.2 Menu() [2/2]

```
Menu::Menu ( const Menu & menu ) [delete]
```

8.26.2.3 ~Menu()

```
Menu::~Menu ( ) [virtual]
```

デストラクタ

Menu.cpp の 55 行目に定義がります。

8.26.3 関数詳解

8.26.3.1 draw()

```
void Menu::draw ( )
```

描画する

Menu.cpp の 80 行目に定義がります。

8.26.3.2 operator=()

```
Menu& Menu::operator= ( const Menu & menu ) [delete]
```

8.26.4 フレンドと関連関数の詳解

8.26.4.1 Draw

```
friend class Draw [friend]
```

Menu.h の 23 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [Menu.h](#)
- [Menu.cpp](#)

8.27 Window クラス

```
#include <Window.h>
```

公開メンバ関数

- [Window \(const std::string &title="GLFW Window", int width=640, int height=480, int fullscreen=0, GLFWwindow *share=nullptr\)](#)
- [Window \(const Window &w\)=delete](#)
- [Window & operator= \(const Window &w\)=delete](#)
- [virtual ~Window \(\)](#)
- [auto * get \(\) const](#)
- [void setClose \(int flag=GLFW_TRUE\) const](#)
- [bool shouldClose \(\) const](#)
- [operator bool \(\)](#)
- [void swapBuffers \(\) const](#)
- [auto getWidth \(\) const](#)
- [auto getHeight \(\) const](#)
- [auto getFboWidth \(\) const](#)
- [auto getFboHeight \(\) const](#)
- [const auto & getSize \(\) const](#)
- [void getSize \(GLsizei *size\) const](#)
- [const auto getFboSize \(\) const](#)
- [void getFboSize \(GLsizei *fboSize\) const](#)
- [auto getAspect \(\) const](#)
- [void restoreViewport \(\) const](#)
- [bool getKey \(int key\) const](#)
- [void selectInterface \(int no\)](#)
- [void setVelocity \(GLfloat vx, GLfloat vy, GLfloat vz=0.1f\)](#)
- [int getLastKey \(\)](#)
- [auto getArrow \(int direction=0, int mods=0\) const](#)
- [auto getArrowX \(int mods=0\) const](#)
- [auto getArrowY \(int mods=0\) const](#)
- [void getArrow \(GLfloat *arrow, int mods=0\) const](#)
- [auto getShiftArrowX \(\) const](#)
- [auto getShiftArrowY \(\) const](#)
- [void getShiftArrow \(GLfloat *shift_arrow\) const](#)
- [auto getControlArrowX \(\) const](#)
- [auto getControlArrowY \(\) const](#)
- [void getControlArrow \(GLfloat *control_arrow\) const](#)

- auto `getAltArrowX () const`
- auto `getAltArrowY () const`
- void `getAltArrow (GLfloat *alt_arrow) const`
- const auto * `getMouse () const`
- void `getMouse (GLfloat *position) const`
- auto `getMouse (int direction) const`
- auto `getMouseX () const`
- auto `getMouseY () const`
- const auto * `getWheel () const`
- void `getWheel (GLfloat *rotation) const`
- auto `getWheel (int direction) const`
- auto `getWheelX () const`
- auto `getWheelY () const`
- const auto & `getTranslation (int button=GLFW_MOUSE_BUTTON_1) const`
- auto `getTranslationMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
- auto `getScrollMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
- auto `getRotation (int button=GLFW_MOUSE_BUTTON_1) const`
- auto `getRotationMatrix (int button=GLFW_MOUSE_BUTTON_1) const`
- void `resetRotation ()`
- void `resetTranslation ()`
- void `reset ()`
- void * `getUserPointer () const`
- void `setUserPointer (void *pointer)`
- void `setResizeFunc (void(*func)(const Window *window, int width, int height))`
- void `setKeyboardFunc (void(*func)(const Window *window, int key, int scancode, int action, int mods))`
- void `setMouseFunc (void(*func)(const Window *window, int button, int action, int mods))`
- void `setWheelFunc (void(*func)(const Window *window, double x, double y))`
- void `setMenubarHeight (float menubarheight)`

静的公開メンバ関数

- static void `initialize (int major=0, int minor=1)`

8.27.1 詳解

ウィンドウ関連の処理.

覚え書き

GLFW を使って OpenGL のウィンドウを操作するラッパークラス.

Window.h の 94 行目に定義があります。

8.27.2 構築子と解体子

8.27.2.1 Window() [1/2]

```
Window::Window (
    const std::string & title = "GLFW Window",
    int width = 640,
    int height = 480,
    int fullscreen = 0,
    GLFWwindow * share = nullptr ) [inline]
```

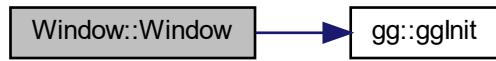
コンストラクタ.

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, フルスクリーン時は無視され実際のディスプレイの幅が使われる.
<i>height</i>	開くウィンドウの高さ, フルスクリーン時は無視され実際のディスプレイの高さが使われる.
<i>fullscreen</i>	フルスクリーン表示を行うディスプレイ番号, 0ならフルスクリーン表示を行わない.
<i>share</i>	共有するコンテキスト, <code>nullptr</code> ならコンテキストを共有しない.

Window.h の 478 行目に定義があります。

呼び出し関係図:



8.27.2.2 Window() [2/2]

```
Window::Window (
    const Window & w ) [delete]
```

8.27.2.3 ~Window()

```
virtual Window::~Window ( ) [inline], [virtual]
```

デストラクタ.

Window.h の 587 行目に定義があります。

8.27.3 関数詳解

8.27.3.1 get()

```
auto* 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 601 行目に定義があります。

8.27.3.2 getAltArrowX()

```
auto 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 965 行目に定義があります。

8.27.3.3 getAltArrowY()

```
auto 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 975 行目に定義があります。

8.27.3.4 getAltArrow()

```
void Window::getAltArrow (
    GLfloat * alt_arrow ) const [inline]
```

ALT キーを押しながら矢印キーを押したときの現在の値を得る。

引数

<i>alt_arrow</i>	ALT キーを押しながら矢印キーを押したときの値を格納する GLfloat 型の 2 要素の配列。
------------------	---

Window.h の 985 行目に定義があります。

8.27.3.5 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 892 行目に定義があります。

8.27.3.6 getArrow() [2/2]

```
auto 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 858 行目に定義があります。

8.27.3.7 getArrowX()

```
auto 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 870 行目に定義があります。

8.27.3.8 getArrowY()

```
auto 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 881 行目に定義があります。

8.27.3.9 getAspect()

```
auto 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 788 行目に定義があります。

8.27.3.10 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 954 行目に定義があります。

8.27.3.11 getControlArrowX()

```
auto 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 934 行目に定義があります。

8.27.3.12 getControlArrowY()

```
auto 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 944 行目に定義があります。

8.27.3.13 `getFboHeight()`

```
auto Window::getFboHeight ( ) const [inline]
```

FBO の高さを得る。

戻り値

FBO の高さ。

Window.h の 736 行目に定義があります。

8.27.3.14 `getFboSize() [1/2]`

```
const auto Window::getFboSize ( ) const [inline]
```

FBO のサイズを得る。

戻り値

FBO の幅と高さを格納した `GLsizei` 型の 2 要素の配列。

Window.h の 767 行目に定義があります。

8.27.3.15 `getFboSize() [2/2]`

```
void Window::getFboSize ( GLsizei * fboSize ) const [inline]
```

FBO のサイズを得る。

引数

<code>size</code>	FBO の幅と高さを格納した <code>GLsizei</code> 型の 2 要素の配列。
-------------------	---

Window.h の 777 行目に定義があります。

8.27.3.16 `getFboWidth()`

```
auto Window::getFboWidth ( ) const [inline]
```

FBO の横幅を得る。

戻り値

FBO の横幅.

Window.h の 726 行目に定義があります。

8.27.3.17 getHeight()

```
auto 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 716 行目に定義があります。

8.27.3.18 getKey()

```
bool Window::getKey ( int key ) const [inline]
```

キーが押されているかどうかを判定する.

戻り値

キーが押されていれば true.

Window.h の 806 行目に定義があります。

8.27.3.19 getLastKey()

```
int Window::getLastKey ( ) [inline]
```

最後にタイプしたキーを得る.

戻り値

最後にタイプしたキーの文字.

Window.h の 843 行目に定義があります。

8.27.3.20 getMouse() [1/3]

```
const auto* 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 996 行目に定義があります。

8.27.3.21 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 1007 行目に定義があります。

8.27.3.22 getMouse() [3/3]

```
auto 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 1020 行目に定義があります。

8.27.3.23 getMouseX()

```
auto 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 1031 行目に定義があります。

8.27.3.24 getMouseY()

```
auto 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 1042 行目に定義があります。

8.27.3.25 getRotation()

```
auto 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 1164 行目に定義があります。

8.27.3.26 getRotationMatrix()

```
auto Window::getRotationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボールの回転変換行列を得る。

引数

<i>button</i>	回転変換行列を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	--

戻り値

回転を行う GgMatrix 型の変換行列。

Window.h の 1177 行目に定義があります。

8.27.3.27 getScrollMatrix()

```
auto Window::getScrollMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによってオブジェクトの平行移動の変換行列を得る。

引数

<i>button</i>	平行移動量を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	---

戻り値

クリッピング座標系で平行移動を行う GgMatrix 型の変換行列。

Window.h の 1144 行目に定義があります。

8.27.3.28 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 923 行目に定義があります。

8.27.3.29 `getShiftArrowX()`

```
auto 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 903 行目に定義があります。

8.27.3.30 `getShiftArrowY()`

```
auto 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 913 行目に定義があります。

8.27.3.31 `getSize() [1/2]`

```
const auto& Window::getSize ( ) const [inline]
```

ウィンドウのサイズを得る。

戻り値

ウィンドウの幅と高さを格納した GLsizei 型の 2 要素の配列の参照。

Window.h の 746 行目に定義があります。

8.27.3.32 `getSize() [2/2]`

```
void Window::getSize (
    GLsizei * size ) const [inline]
```

ウィンドウのサイズを得る。

引数

<code>size</code>	ウィンドウの幅と高さを格納した <code>GLsizei</code> 型の 2 要素の配列.
-------------------	--

`Window.h` の 756 行目に定義があります。

8.27.3.33 `getTranslation()`

```
const auto& Window::getTranslation (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボール処理を考慮したマウスによるスクロールの変換行列を得る。

引数

<code>button</code>	平行移動量を取得するマウスボタン (<code>GLFW_MOUSE_BUTTON_[1,2]</code>).
---------------------	--

戻り値

平行移動量を格納した `GLfloat[3]` の配列のポインタ。

`Window.h` の 1111 行目に定義があります。

8.27.3.34 `getTranslationMatrix()`

```
auto Window::getTranslationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

マウスによって視点の平行移動の変換行列を得る。

引数

<code>button</code>	平行移動量を取得するマウスボタン (<code>GLFW_MOUSE_BUTTON_[1,2]</code>).
---------------------	--

戻り値

視点座標系で平行移動を行う `GgMatrix` 型の変換行列。

`Window.h` の 1124 行目に定義があります。

8.27.3.35 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 1219 行目に定義があります。

8.27.3.36 getWheel() [1/3]

```
const auto* 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 1053 行目に定義があります。

8.27.3.37 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 1064 行目に定義があります。

8.27.3.38 getWheel() [3/3]

```
auto 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 1077 行目に定義があります。

8.27.3.39 getWheelX()

```
auto Window::getWheelX () const [inline]
```

マウスホイールの X 方向の回転量を得る.

戻り値

マウスホイールの X 方向の回転量.

Window.h の 1088 行目に定義があります。

8.27.3.40 getWheelY()

```
auto Window::getWheelY () const [inline]
```

マウスホイールの Y 方向の回転量を得る.

戻り値

マウスホイールの Y 方向の回転量.

Window.h の 1099 行目に定義があります。

8.27.3.41 getWidth()

```
auto 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 706 行目に定義があります。

8.27.3.42 initialize()

```
static void Window::initialize (
    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 419 行目に定義があります。

被呼び出し関係図:



8.27.3.43 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 632 行目に定義があります。

8.27.3.44 operator=()

```
Window& Window::operator= (
    const Window & w ) [delete]
```

8.27.3.45 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 1205 行目に定義があります。

8.27.3.46 resetRotation()

```
void Window::resetRotation ( ) [inline]
```

トラックボール処理を初期化する。

Window.h の 1187 行目に定義があります。

8.27.3.47 resetTranslation()

```
void Window::resetTranslation ( ) [inline]
```

平行移動量を初期化する。

Window.h の 1196 行目に定義があります。

8.27.3.48 restoreViewport()

```
void Window::restoreViewport ( ) const [inline]
```

ビューポートをウィンドウ全体に設定する。

Window.h の 796 行目に定義があります。

8.27.3.49 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 821 行目に定義があります。

8.27.3.50 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

<code>flag</code>	クローズフラグ, 0 (<code>GLFW_FALSE</code>) 以外ならウィンドウを閉じる.
-------------------	---

Window.h の 611 行目に定義があります。

8.27.3.51 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

<code>func</code>	ユーザ定義の <code>keyboard</code> 関数, キーボードの操作時に呼び出される.
-------------------	--

Window.h の 1249 行目に定義があります。

8.27.3.52 setMenubarHeight()

```
void Window::setMenubarHeight (
    float menubarheight ) [inline]
```

表示領域をメニューバーの高さだけ減らす。

引数

メニューバーの高さ.	
------------	--

Window.h の 1279 行目に定義があります。

8.27.3.53 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 1259 行目に定義があります。

8.27.3.54 setResizeFunc()

```
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 1239 行目に定義があります。

8.27.3.55 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 1229 行目に定義があります。

8.27.3.56 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 833 行目に定義があります。

8.27.3.57 setWheelFunc()

```
void Window::setWheelFunc (
    void(*)(const Window *window, double x, double y) func ) [inline]
```

ユーザ定義の wheel 関数を設定する。

引数

<i>func</i>	ユーザ定義の wheel 関数, マウスホイールの操作時に呼び出される.
-------------	--------------------------------------

Window.h の 1269 行目に定義があります。

8.27.3.58 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 621 行目に定義があります。

8.27.3.59 swapBuffers()

```
void Window::swapBuffers ( ) const [inline]
```

カラーバッファを入れ替える。

Window.h の 686 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:

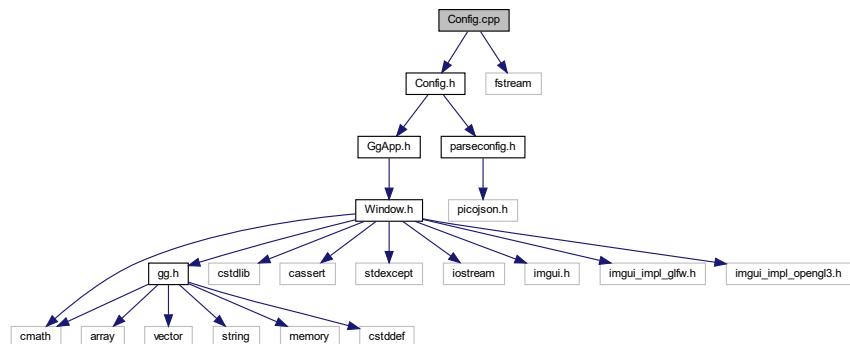
- [Window.h](#)

Chapter 9

ファイル詳解

9.1 Config.cpp ファイル

```
#include "Config.h"
#include <fstream>
Config.cpp の依存先関係図:
```



変数

- constexpr GgSimpleShader::Light defaultLight

9.1.1 詳解

設定構造体の実装

著者

Kohe Tokoi

日付

November 15, 2022

9.1.2 変数詳解

9.1.2.1 defaultLight

```
constexpr GgSimpleShader::Light defaultLight [constexpr]
```

初期値:

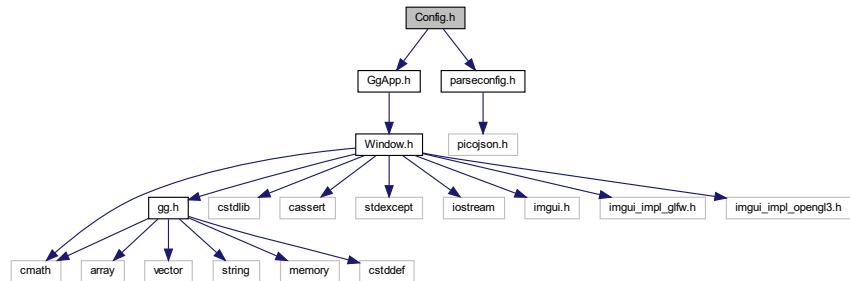
```
{
    { 0.2f, 0.2f, 0.2f, 1.0f },
    { 1.0f, 1.0f, 1.0f, 0.0f },
    { 1.0f, 1.0f, 1.0f, 0.0f },
    { 0.5f, 0.5f, 1.0f, 1.0f }
}
```

Config.cpp の 14 行目に定義があります。

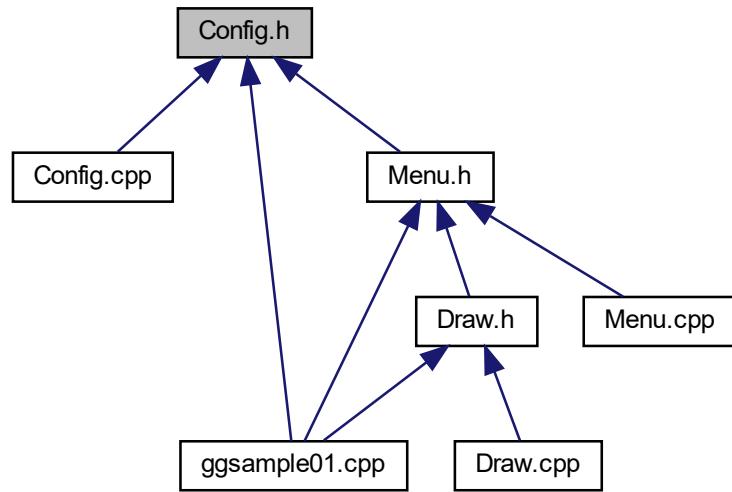
9.2 Config.h ファイル

```
#include "GgApp.h"
#include "parseconfig.h"
```

Config.h の依存先関係図:



被依存関係図:



クラス

- class [Config](#)

マクロ定義

- `#define Utf8ToTChar(string) (string)`
- `#define TCharToUtf8(cstring) (cstring)`

型定義

- `using pathString = std::string`
- `using pathChar = char`

9.2.1 詳解

構成データクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

9.2.2 マクロ定義詳解

9.2.2.1 TCHARToUtf8

```
#define TCHARToUtf8(  
    cstring ) (cstring)
```

Config.h の 34 行目に定義がります。

9.2.2.2 Utf8ToTChar

```
#define Utf8ToTChar(  
    string ) (string)
```

Config.h の 33 行目に定義がります。

9.2.3 型定義詳解

9.2.3.1 pathChar

```
using pathChar = char
```

Config.h の 32 行目に定義がります。

9.2.3.2 pathString

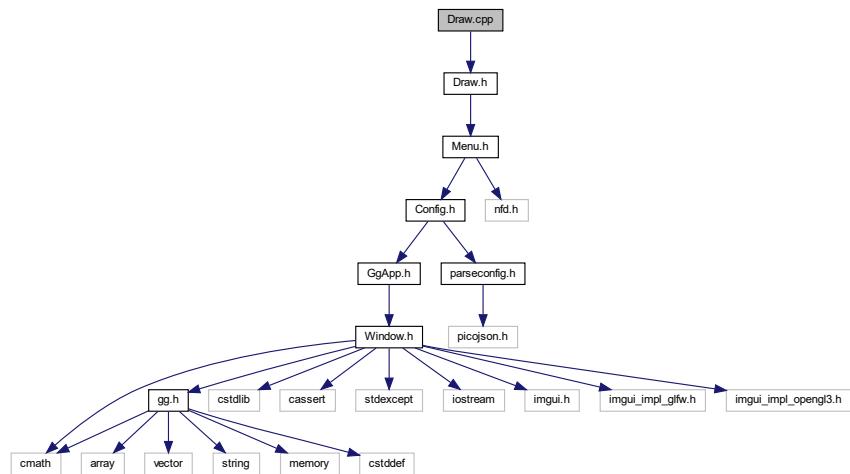
```
using pathString = std::string
```

Config.h の 31 行目に定義がります。

9.3 Draw.cpp ファイル

```
#include "Draw.h"
```

Draw.cpp の依存先関係図:



9.3.1 詳解

図形の描画クラスの実装

著者

Kohe Tokoi

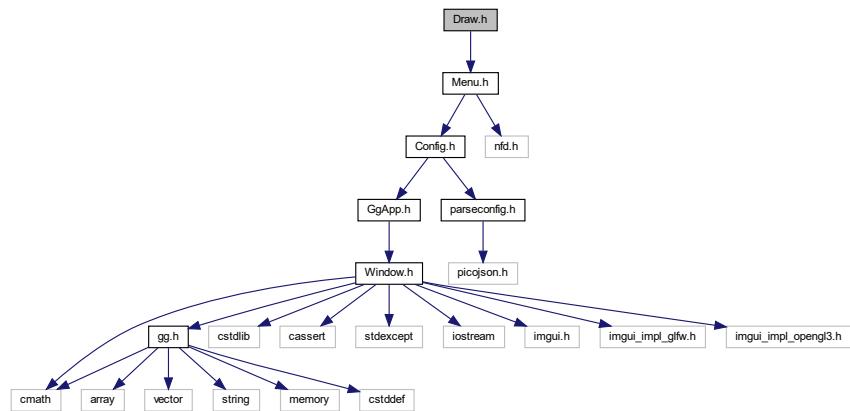
日付

November 15, 2022

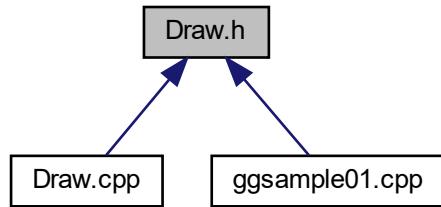
9.4 Draw.h ファイル

```
#include "Menu.h"
```

Draw.h の依存先関係図:



被依存関係図:



クラス

- class **Draw**

9.4.1 詳解

図形の描画クラスの定義

著者

Kohe Tokoi

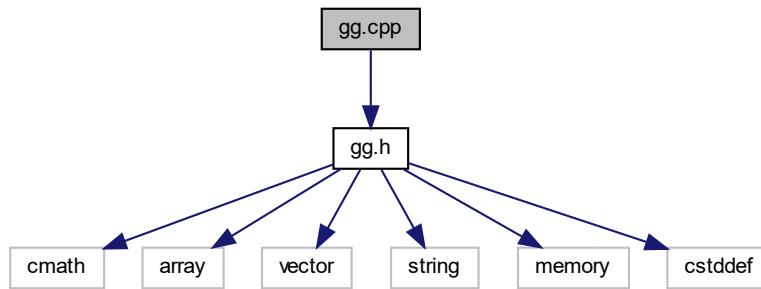
日付

November 15, 2022

9.5 gg.cpp ファイル

```
#include "gg.h"
```

gg.cpp の依存先関係図:



9.5.1 詳解

ゲームグラフィックス特論の宿題用補助プログラム GLFW3 版の定義。

著者

Kohe Tokoi

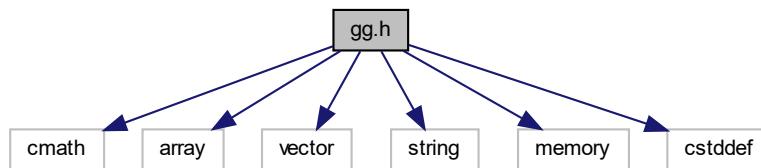
日付

March 31, 2021

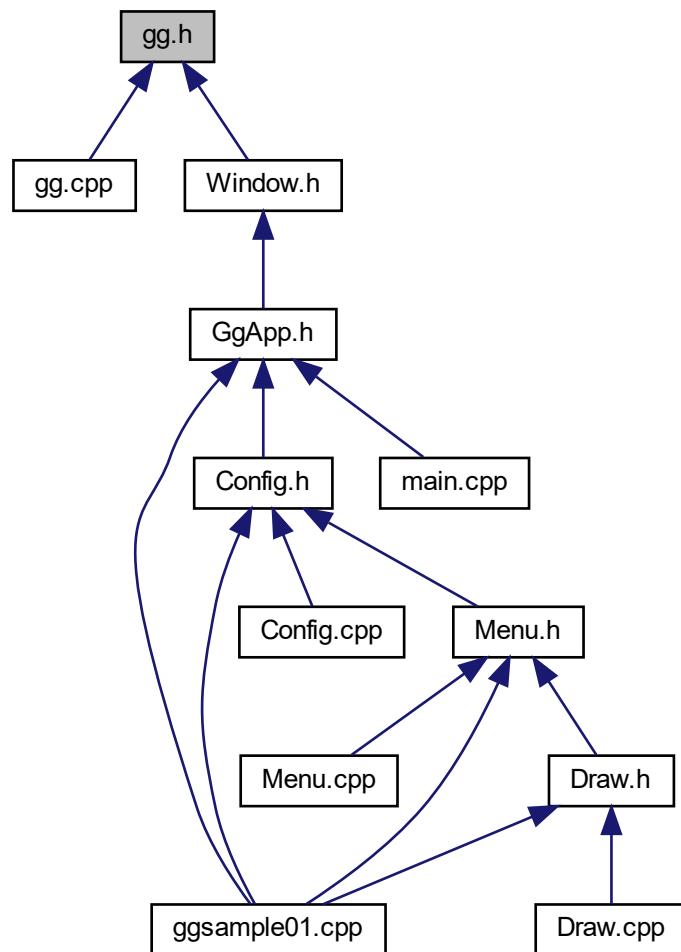
9.6 gg.h ファイル

```
#include <cmath>
#include <array>
#include <vector>
#include <string>
#include <memory>
#include <cstddef>
```

gg.h の依存先関係図:



被依存関係図:



クラス

- class `gg::GgVector`
- class `gg::GgMatrix`
- class `gg::GgQuaternion`
- class `gg::GgTrackball`
- class `gg::GgTexture`
- class `gg::GgColorTexture`
- class `gg::GgNormalTexture`
- class `gg::GgBuffer< T >`
- class `gg::GgUniformBuffer< T >`
- class `gg::GgShape`
- class `gg::GgPoints`
- struct `gg::GgVertex`
- class `gg::GgTriangles`
- class `gg::GgElements`

- class `gg::GgShader`
- class `gg::GgPointShader`
- class `gg::GgSimpleShader`
- struct `gg::GgSimpleShader::Light`
- class `gg::GgSimpleShader::LightBuffer`
- struct `gg::GgSimpleShader::Material`
- class `gg::GgSimpleShader::MaterialBuffer`
- class `gg::GgSimpleObj`

名前空間

- `gg`

マクロ定義

- `#define ggError()`
- `#define ggFBOError()`

列挙型

- enum `gg::BindingPoints { gg::LightBindingPoint = 0 , gg::MaterialBindingPoint }`

関数

- void `gg::ggInit ()`
- void `gg::ggError (const std::string &name="", unsigned int line=0)`
- void `gg::ggFBOError (const std::string &name="", unsigned int line=0)`
- void `gg::ggCross (GLfloat *c, const GLfloat *a, const GLfloat *b)`
- GLfloat `gg::ggDot3 (const GLfloat *a, const GLfloat *b)`
- GLfloat `gg::ggLength3 (const GLfloat *a)`
- GLfloat `gg::ggDistance3 (const GLfloat *a, const GLfloat *b)`
- void `gg::ggNormalize3 (const GLfloat *a, GLfloat *b)`
- void `gg::ggNormalize3 (GLfloat *a)`
- GLfloat `gg::ggDot4 (const GLfloat *a, const GLfloat *b)`
- GLfloat `gg::ggLength4 (const GLfloat *a)`
- GLfloat `gg::ggDistance4 (const GLfloat *a, const GLfloat *b)`
- void `gg::ggNormalize4 (const GLfloat *a, GLfloat *b)`
- void `gg::ggNormalize4 (GLfloat *a)`
- const GgVector & `gg::operator+ (const GgVector &v)`
- GgVector `gg::operator+ (GLfloat a, const GgVector &b)`
- const GgVector `gg::operator- (const GgVector &v)`
- GgVector `gg::operator- (GLfloat a, const GgVector &b)`
- GgVector `gg::operator* (GLfloat a, const GgVector &b)`
- GgVector `gg::operator/ (GLfloat a, const GgVector &b)`
- GgVector `gg::ggCross (const GgVector &a, const GgVector &b)`
- GLfloat `gg::ggDot3 (const GgVector &a, const GgVector &b)`
- GLfloat `gg::ggLength3 (const GgVector &a)`
- GLfloat `gg::ggDistance3 (const GgVector &a, const GgVector &b)`
- GgVector `gg::ggNormalize3 (const GgVector &a)`
- void `gg::ggNormalize3 (GgVector *a)`
- GLfloat `gg::ggDot4 (const GgVector &a, const GgVector &b)`

- `GLfloat gg::ggLength4 (const GgVector &a)`
- `GLfloat gg::ggDistance4 (const GgVector &a, const GgVector &b)`
- `GgVector gg::ggNormalize4 (const GgVector &a)`
- `void gg::ggNormalize4 (GgVector *a)`
- `GgMatrix gg::ggIdentity ()`
- `GgMatrix gg::ggTranslate (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix gg::ggTranslate (const GLfloat *t)`
- `GgMatrix gg::ggTranslate (const GgVector &t)`
- `GgMatrix gg::ggScale (GLfloat x, GLfloat y, GLfloat z, GLfloat w=1.0f)`
- `GgMatrix gg::ggScale (const GLfloat *s)`
- `GgMatrix gg::ggScale (const GgVector &s)`
- `GgMatrix gg::ggRotateX (GLfloat a)`
- `GgMatrix gg::ggRotateY (GLfloat a)`
- `GgMatrix gg::ggRotateZ (GLfloat a)`
- `GgMatrix gg::ggRotate (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgMatrix gg::ggRotate (const GLfloat *r, GLfloat a)`
- `GgMatrix gg::ggRotate (const GgVector &r, GLfloat a)`
- `GgMatrix gg::ggRotate (const GLfloat *r)`
- `GgMatrix gg::ggRotate (const GgVector &r)`
- `GgMatrix gg::ggLookat (GLfloat ex, GLfloat ey, GLfloat ez, GLfloat tx, GLfloat ty, GLfloat tz, GLfloat ux, GLfloat uy, GLfloat uz)`
- `GgMatrix gg::ggLookat (const GLfloat *e, const GLfloat *t, const GLfloat *u)`
- `GgMatrix gg::ggLookat (const GgVector &e, const GgVector &t, const GgVector &u)`
- `GgMatrix gg::ggOrthogonal (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix gg::ggFrustum (GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat zNear, GLfloat zFar)`
- `GgMatrix gg::ggPerspective (GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar)`
- `GgMatrix gg::ggTranspose (const GgMatrix &m)`
- `GgMatrix gg::ggInvert (const GgMatrix &m)`
- `GgMatrix gg::ggNormal (const GgMatrix &m)`
- `GgQuaternion gg::ggQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `GgQuaternion gg::ggQuaternion (const GLfloat *a)`
- `GgQuaternion gg::ggIdentityQuaternion ()`
- `GgQuaternion gg::ggMatrixQuaternion (const GLfloat *a)`
- `GgQuaternion gg::ggMatrixQuaternion (const GgMatrix &m)`
- `GgMatrix gg::ggQuaternionMatrix (const GgQuaternion &q)`
- `GgMatrix gg::ggQuaternionTransposeMatrix (const GgQuaternion &q)`
- `GgQuaternion gg::ggRotateQuaternion (GLfloat x, GLfloat y, GLfloat z, GLfloat a)`
- `GgQuaternion gg::ggRotateQuaternion (const GLfloat *v, GLfloat a)`
- `GgQuaternion gg::ggRotateQuaternion (const GLfloat *v)`
- `GgQuaternion gg::ggEulerQuaternion (GLfloat heading, GLfloat pitch, GLfloat roll)`
- `GgQuaternion gg::ggEulerQuaternion (const GLfloat *e)`
- `GgQuaternion gg::ggSlerp (const GLfloat *a, const GLfloat *b, GLfloat t)`
- `GgQuaternion gg::ggSlerp (const GgQuaternion &q, const GgQuaternion &r, GLfloat t)`
- `GgQuaternion gg::ggSlerp (const GgQuaternion &q, const GLfloat *a, GLfloat t)`
- `GgQuaternion gg::ggSlerp (const GLfloat *a, const GgQuaternion &q, GLfloat t)`
- `GLfloat gg::ggNorm (const GgQuaternion &q)`
- `GgQuaternion gg::ggNormalize (const GgQuaternion &q)`
- `GgQuaternion gg::ggConjugate (const GgQuaternion &q)`
- `GgQuaternion gg::ggInvert (const GgQuaternion &q)`
- `bool gg::ggSaveTga (const std::string &name, const void *buffer, unsigned int width, unsigned int height, unsigned int depth)`
- `bool gg::ggSaveColor (const std::string &name)`
- `bool gg::ggSaveDepth (const std::string &name)`

- bool `gg::ggReadImage` (const std::string &name, std::vector< GLubyte > &image, GLsizei *pWidth, GLsizei *pHeight, GLenum *pFormat)
- GLuint `gg::ggLoadTexture` (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGB, GLenum wrap=GL_CLAMP_TO_EDGE)
- GLuint `gg::ggLoadImage` (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE)
- void `gg::ggCreateNormalMap` (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)
- GLuint `gg::ggLoadHeight` (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)
- GLuint `gg::ggCreateShader` (const std::string &vsrc, const std::string &fsrc="", const std::string &gsrc="", GLint nvarying=0, const char *const *varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string >ext="geometry shader")
- GLuint `gg::ggLoadShader` (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)
- GLuint `gg::ggLoadShader` (const std::array< std::string, 3 > &files, GLint nvarying=0, const char *const *varyings=nullptr)
- GLuint `gg::ggCreateComputeShader` (const std::string &csrc, const std::string &ctext="compute shader")
- GLuint `gg::ggLoadComputeShader` (const std::string &comp)
- std::shared_ptr< GgPoints > `gg::ggPointsCube` (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared_ptr< GgPoints > `gg::ggPointsSphere` (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::shared_ptr< GgTriangles > `gg::ggRectangle` (GLfloat width=1.0f, GLfloat height=1.0f)
- std::shared_ptr< GgTriangles > `gg::ggEllipse` (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)
- std::shared_ptr< GgTriangles > `gg::ggArraysObj` (const std::string &name, bool normalize=false)
- std::shared_ptr< GgElements > `gg::ggElementsObj` (const std::string &name, bool normalize=false)
- std::shared_ptr< GgElements > `gg::ggElementsMesh` (GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3]=nullptr)
- std::shared_ptr< GgElements > `gg::ggElementsSphere` (GLfloat radius=1.0f, int slices=16, int stacks=8)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, bool normalize=false)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)

変数

- GLint `gg::ggBufferAlignment`

使用している GPU のバッファアライメント。

9.6.1 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

9.6.2 マクロ定義詳解

9.6.2.1 ggError

```
#define ggError( )
```

OpenGL のエラーの発生を検知したときにソースファイル名と行番号を表示する。

覚え書き

このマクロを置いた場所（より前）で OpenGL のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

gg.h の 1379 行目に定義があります。

9.6.2.2 ggFBOError

```
#define ggFBOError( )
```

FBO のエラーの発生を検知したときにソースファイル名と行番号を表示する。

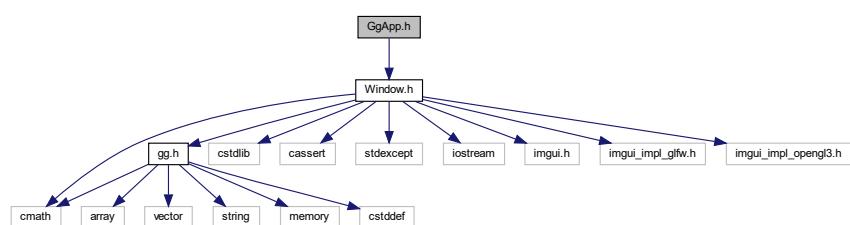
覚え書き

このマクロを置いた場所（より前）で FBO のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

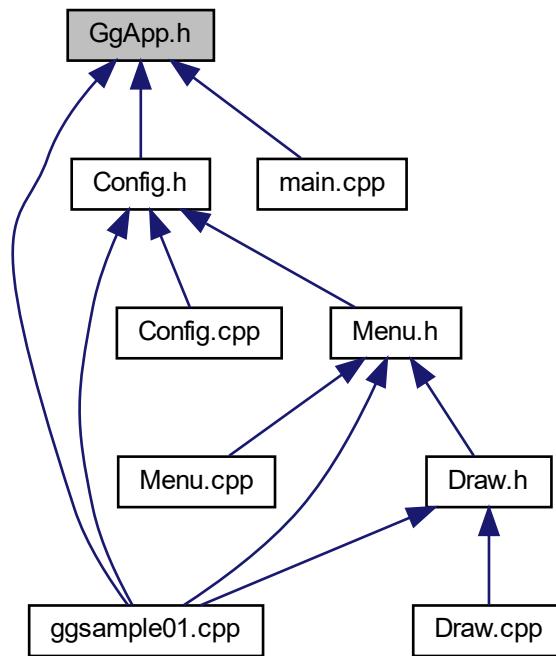
gg.h の 1406 行目に定義があります。

9.7 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 GG_USE_IMGUI`

9.7.1 マクロ定義詳解

9.7.1.1 GG_USE_IMGUI

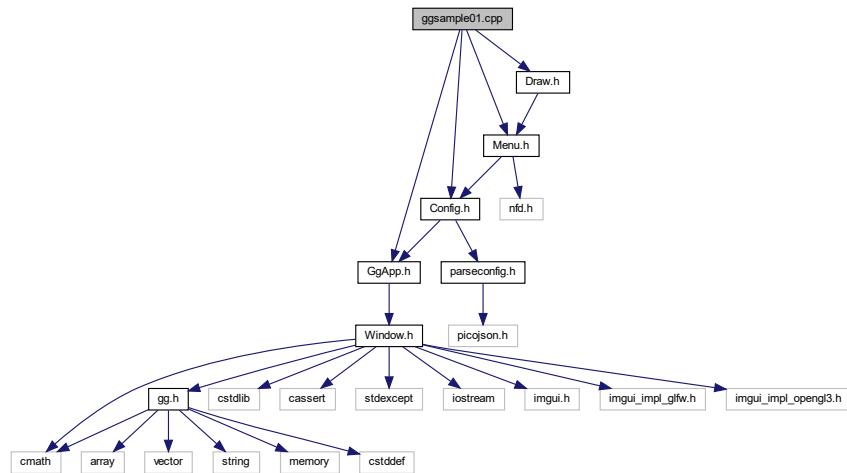
```
#define GG_USE_IMGUI
```

GgApp.h の 4 行目に定義があります。

9.8 ggsample01.cpp ファイル

```
#include "GgApp.h"
#include "Config.h"
#include "Menu.h"
#include "Draw.h"
```

ggsample01.cpp の依存先関係図:



マクロ定義

- `#define PROJECT_NAME "matching"`
- `#define CONFIG_FILE PROJECT_NAME "_config.json"`

9.8.1 マクロ定義詳解

9.8.1.1 CONFIG_FILE

```
#define CONFIG_FILE PROJECT_NAME "_config.json"
```

ggsample01.cpp の 13 行目に定義があります。

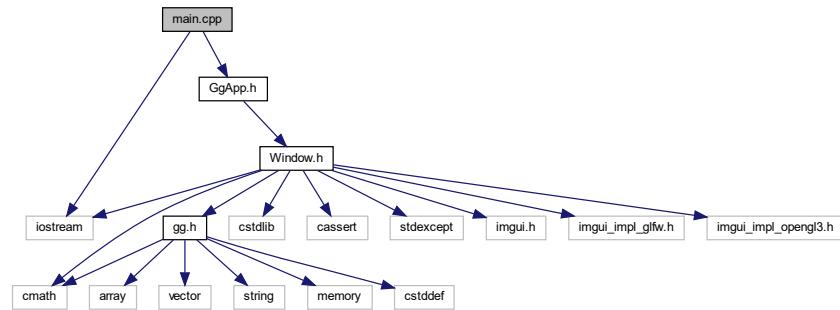
9.8.1.2 PROJECT_NAME

```
#define PROJECT_NAME "matching"
```

ggsample01.cpp の 8 行目に定義があります。

9.9 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)`

9.9.1 マクロ定義詳解

9.9.1.1 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。

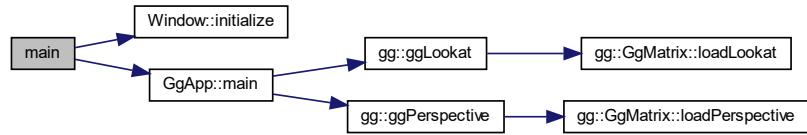
9.9.2 関数詳解

9.9.2.1 main()

```
int main (
    int argc,
    const char *const * argv )
```

main.cpp の 23 行目に定義があります。

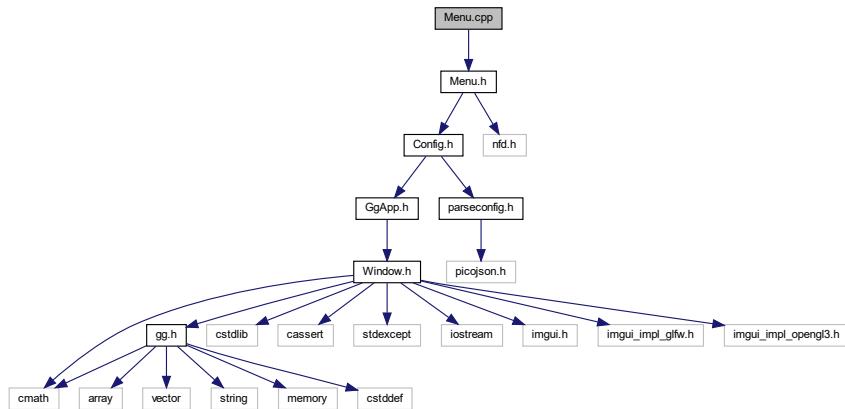
呼び出し関係図:



9.10 Menu.cpp ファイル

```
#include "Menu.h"
```

Menu.cpp の依存先関係図:



変数

- constexpr char menuFont[] { "Mplus1-Regular.ttf" }
- constexpr float menuFontSize { 20.0f }

9.10.1 詳解

メニューの描画クラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

9.10.2 変数詳解

9.10.2.1 menuFont

```
constexpr char menuFont[] { "Mplus1-Regular.ttf" } [constexpr]
```

Menu.cpp の 11 行目に定義があります。

9.10.2.2 menuFontSize

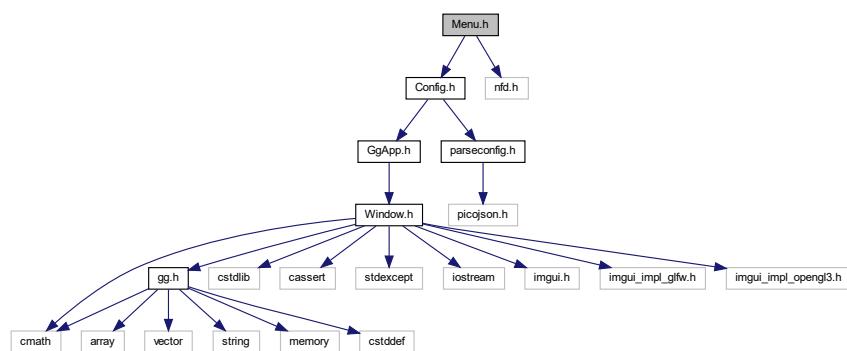
```
constexpr float menuFontSize { 20.0f } [constexpr]
```

Menu.cpp の 14 行目に定義があります。

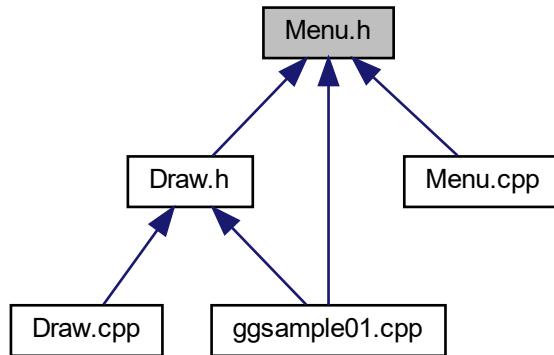
9.11 Menu.h ファイル

```
#include "Config.h"
#include "nfd.h"
```

Menu.h の依存先関係図:



被依存関係図:



クラス

- class [Menu](#)

9.11.1 詳解

メニューの描画クラスの定義

著者

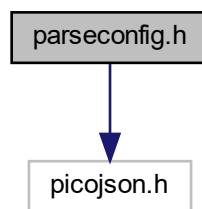
Kohe Tokoi

日付

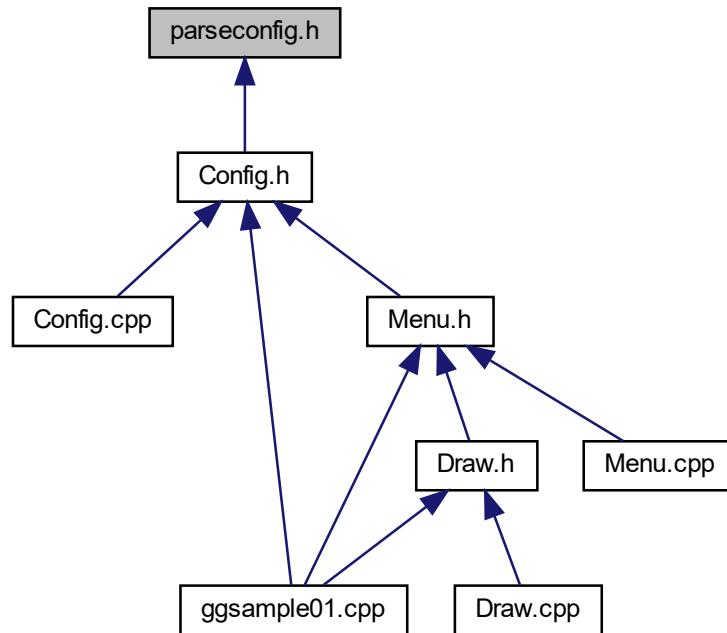
November 15, 2022

9.12 parseconfig.h ファイル

#include "picojson.h"
parseconfig.h の依存先関係図:



被依存関係図:



関数

- template<typename T >
 bool **getValue** (const picojson::object &object, const std::string &key, T &scalar)
- template<typename T , std::size_t U>
 bool **getValue** (const picojson::object &object, const std::string &key, std::array< T, U > &vector)
- bool **getVector** (const picojson::object &object, const std::string &key, GgVector &vector)
- bool **getString** (const picojson::object &object, const std::string &key, std::string &string)
- template<std::size_t U>
 bool **getString** (const picojson::object &object, const std::string &key, std::array< std::string, U > &strings)
- bool **getString** (const picojson::object &object, const std::string &key, std::vector< std::string > &strings)
- template<typename T >
 void **setValue** (picojson::object &object, const std::string &key, const T &scalar)
- template<typename T , std::size_t U>
 void **setValue** (picojson::object &object, const std::string &key, const std::array< T, U > &vector)
- void **setVector** (picojson::object &object, const std::string &key, const GgVector &vector)
- void **setString** (picojson::object &object, const std::string &key, const std::string &string)
- template<std::size_t U>
 void **setString** (picojson::object &object, const std::string &key, const std::array< std::string, U > &strings)
- void **setString** (picojson::object &object, const std::string &key, const std::vector< std::string > &strings)

9.12.1 詳解

構成ファイルの読み取り補助

著者

Kohe Tokoi

日付

November 15, 2022

9.12.2 関数詳解

9.12.2.1 getString() [1/3]

```
template<std::size_t U>
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトから文字列の配列を取得する

テンプレート引数

<i>U</i>	構成ファイルから取得する配列の要素の文字列の数
----------	-------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

parseconfig.h の 138 行目に定義があります。

9.12.2.2 getString() [2/3]

```
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::string & string ) [inline]
```

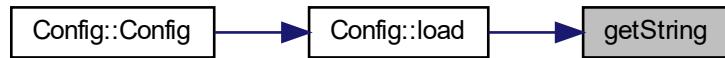
構成ファイルの JSON オブジェクトから文字列を取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>string</i>	取得した文字列を格納する変数

parseconfig.h の 114 行目に定義があります。

被呼び出し関係図:



9.12.2.3 getString() [3/3]

```
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトから文字列のベクトルを取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

parseconfig.h の 171 行目に定義があります。

9.12.2.4 getValue() [1/2]

```
template<typename T , std::size_t U>
bool getValue (
    const picojson::object & object,
    const std::string & key,
    std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトから数値の配列を取得する

テンプレート引数

<i>T</i>	構成ファイルから取得する配列の要素の数値のデータ型
<i>U</i>	構成ファイルから取得する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>vector</i>	取得した数値の配列を格納する変数

parseconfig.h の 48 行目に定義があります。

9.12.2.5 getValue() [2/2]

```
template<typename T >
bool getValue (
    const picojson::object & object,
    const std::string & key,
    T & scalar )
```

構成ファイルの JSON オブジェクトから数値を取得する

テンプレート引数

<i>T</i>	構成ファイルから取得する数値のデータ型
----------	---------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>scalar</i>	取得した JSON オブジェクトの数値を格納する変数

parseconfig.h の 23 行目に定義があります。

9.12.2.6 getVector()

```
bool getVector (
    const picojson::object & object,
    const std::string & key,
    GgVector & vector ) [inline]
```

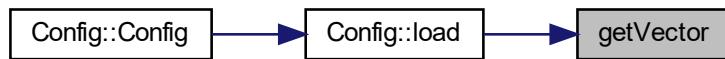
構成ファイルの JSON オブジェクトから 4 要素の数値のベクトルを取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>vector</i>	取得した数値の配列を格納する変数

parseconfig.h の 81 行目に定義があります。

被呼び出し関係図:



9.12.2.7 setString() [1/3]

```
template<std::size_t U>
void setString (
    picojson::object & object,
    const std::string & key,
    const std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトに文字列の配列を設定する

テンプレート引数

<i>U</i>	構成ファイルに設定する配列の要素の文字列の数
----------	------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

parseconfig.h の 283 行目に定義があります。

9.12.2.8 setString() [2/3]

```
void setString (
    picojson::object & object,
    const std::string & key,
    const std::string & string ) [inline]
```

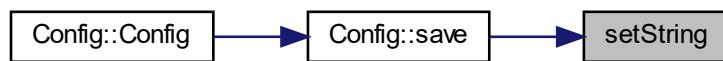
構成ファイルの JSON オブジェクトに文字列を設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>string</i>	設定する文字列

parseconfig.h の 268 行目に定義があります。

被呼び出し関係図:



9.12.2.9 setString() [3/3]

```
void setString (
    picojson::object & object,
    const std::string & key,
    const std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトに文字列のベクトルを設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

parseconfig.h の 308 行目に定義があります。

9.12.2.10 setValue() [1/2]

```
template<typename T , std::size_t U>
void setValue (
    picojson::object & object,
    const std::string & key,
    const std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトに数値の配列を設定する

テンプレート引数

<i>T</i>	構成ファイルに設定する配列の要素の数値のデータ型
<i>U</i>	構成ファイルに設定する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>vector</i>	設定する数値の配列

parseconfig.h の 218 行目に定義があります。

9.12.2.11 setValue() [2/2]

```
template<typename T >
void setValue (
    picojson::object & object,
    const std::string & key,
    const T & scalar )
```

構成ファイルの JSON オブジェクトに数値を設定する

テンプレート引数

<i>T</i>	構成ファイルに設定する数値のデータ型
----------	--------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>scalar</i>	設定する数値

parseconfig.h の 202 行目に定義があります。

9.12.2.12 setVector()

```
void setVector (
    picojson::object & object,
    const std::string & key,
    const GgVector & vector ) [inline]
```

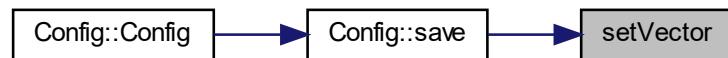
構成ファイルの JSON オブジェクトに 4 要素の数値のベクトルを設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>vector</i>	設定する数値の配列

parseconfig.h の 243 行目に定義があります。

被呼び出し関係図:

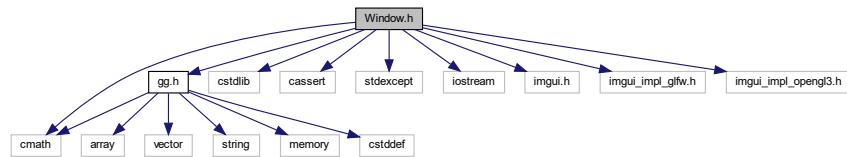


9.13 README.md ファイル

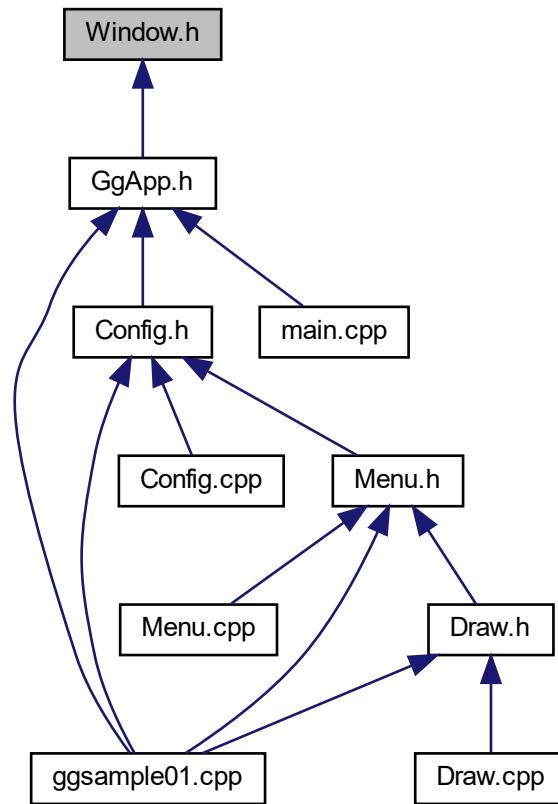
9.14 Window.h ファイル

```
#include "gg.h"
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <stdexcept>
#include <iostream>
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"
```

Window.h の依存先関係図:



被依存関係図:



クラス

- class [Window](#)

マクロ定義

- `#define GG_BUTTON_COUNT 3`
- `#define GG_INTERFACE_COUNT 5`

9.14.1 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

9.14.2 マクロ定義詳解

9.14.2.1 GG_BUTTON_COUNT

```
#define GG_BUTTON_COUNT 3
```

Window.h の 44 行目に定義がります。

9.14.2.2 GG_INTERFACE_COUNT

```
#define GG_INTERFACE_COUNT 5
```

Window.h の 49 行目に定義がります。

Index

-ggError
 gg, 18
-ggFBOError
 gg, 19
~Config
 Config, 88
~Draw
 Draw, 91
~GgBuffer
 gg::GgBuffer< T >, 95
~GgColorTexture
 gg::GgColorTexture, 101
~GgElements
 gg::GgElements, 105
~GgNormalTexture
 gg::GgNormalTexture, 165
~GgPointShader
 gg::GgPointShader, 173
~GgPoints
 gg::GgPoints, 168
~GgShader
 gg::GgShader, 248
~GgShape
 gg::GgShape, 251
~GgSimpleObj
 gg::GgSimpleObj, 254
~GgSimpleShader
 gg::GgSimpleShader, 258
~GgTexture
 gg::GgTexture, 272
~GgTrackball
 gg::GgTrackball, 277
~GgTriangles
 gg::GgTriangles, 286
~GgUniformBuffer
 gg::GgUniformBuffer< T >, 291
~LightBuffer
 gg::GgSimpleShader::LightBuffer, 317
~MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 328
~Menu
 Menu, 335
~Window
 Window, 338

add
 gg::GgQuaternion, 184–186
ambient
 gg::GgSimpleShader::Light, 314
 gg::GgSimpleShader::Material, 325

begin
 gg::GgTrackball, 277
bind
 gg::GgBuffer< T >, 95
 gg::GgTexture, 272
 gg::GgUniformBuffer< T >, 291

BindingPoints
 gg, 18

Config, 87
 ~Config, 88
 Config, 87
 load, 88
 Menu, 90
 save, 89
Config.cpp, 359
 defaultLight, 360
Config.h, 360
 pathChar, 362
 pathString, 362
 TCharToUtf8, 362
 Utf8ToTChar, 362

CONFIG_FILE
 ggsample01.cpp, 372

conjugate
 gg::GgQuaternion, 187

copy
 gg::GgBuffer< T >, 95
 gg::GgUniformBuffer< T >, 291

defaultLight
 Config.cpp, 360

diffuse
 gg::GgSimpleShader::Light, 314
 gg::GgSimpleShader::Material, 325

distance3
 gg::GgVector, 299

distance4
 gg::GgVector, 299

divide
 gg::GgQuaternion, 188–190

dot3
 gg::GgVector, 300

dot4
 gg::GgVector, 301

Draw, 90
 ~Draw, 91
 Draw, 91
 draw, 91
 Menu, 335

draw
 Draw, 91
 gg::GgElements, 105
 gg::GgPoints, 169
 gg::GgShape, 251
 gg::GgSimpleObj, 254
 gg::GgTriangles, 287
 Menu, 335
 Draw.cpp, 363
 Draw.h, 363

end
 gg::GgTrackball, 277

euler
 gg::GgQuaternion, 190, 191

fill
 gg::GgUniformBuffer< T >, 292

frustum
 gg::GgMatrix, 112

get
 gg::GgMatrix, 113, 114
 gg::GgPointShader, 173
 gg::GgQuaternion, 192
 gg::GgShader, 248
 gg::GgShape, 252
 gg::GgSimpleObj, 255
 gg::GgTrackball, 278
 Window, 338

getAltArrowX
 Window, 339

getAltArrowY
 Window, 339

getAltArrow
 Window, 339

getArrow
 Window, 340

getArrowX
 Window, 340

getArrowY
 Window, 341

getAspect
 Window, 341

getBuffer
 gg::GgBuffer< T >, 96
 gg::GgPoints, 169
 gg::GgTriangles, 287
 gg::GgUniformBuffer< T >, 292

getConjugateMatrix
 gg::GgQuaternion, 192, 193

getControlArrow
 Window, 341

getControlArrowX
 Window, 342

getControlArrowY
 Window, 342

getCount
 gg::GgBuffer< T >, 96

gg::GgPoints, 169
 gg::GgTriangles, 287
 gg::GgUniformBuffer< T >, 292

getFboHeight
 Window, 342

getFboSize
 Window, 343

getFboWidth
 Window, 343

getHeight
 gg::GgTexture, 272
 Window, 344

getIndexBuffer
 gg::GgElements, 106

getIndexCount
 gg::GgElements, 106

getKey
 Window, 344

getLastKey
 Window, 344

getMatrix
 gg::GgQuaternion, 194, 195
 gg::GgTrackball, 278

getMode
 gg::GgShape, 252

getMouse
 Window, 344, 345

getMouseX
 Window, 345

getMouseY
 Window, 346

getQuaternion
 gg::GgTrackball, 279

getRotation
 Window, 346

getRotationMatrix
 Window, 346

getScale
 gg::GgTrackball, 279, 280

getScrollMatrix
 Window, 347

getShiftArrow
 Window, 347

getShiftArrowX
 Window, 348

getShiftArrowY
 Window, 348

getSize
 gg::GgTexture, 272, 273
 Window, 348

getStart
 gg::GgTrackball, 280, 281

getStride
 gg::GgBuffer< T >, 96
 gg::GgUniformBuffer< T >, 292

getString
 parseconfig.h, 378, 379

getTarget

gg::GgBuffer< T >, 96
gg::GgUniformBuffer< T >, 293
getTexture
 gg::GgTexture, 273
getTranslation
 Window, 349
getTranslationMatrix
 Window, 349
getUserPointer
 Window, 349
getValue
 parseconfig.h, 379, 380
getVector
 parseconfig.h, 380
getWheel
 Window, 350
getWheelX
 Window, 351
getWheelY
 Window, 351
getWidth
 gg::GgTexture, 274
 Window, 351
gg, 15
 _ggError, 18
 _ggFBOError, 19
 BindingPoints, 18
 ggArraysObj, 19
 ggBufferAlignment, 86
 ggConjugate, 20
 ggCreateComputeShader, 20
 ggCreateNormalMap, 21
 ggCreateShader, 22
 ggCross, 23
 ggDistance3, 24
 ggDistance4, 25, 26
 ggDot3, 27, 28
 ggDot4, 28, 29
 ggElementsMesh, 30
 ggElementsObj, 31
 ggElementsSphere, 32
 ggEllipse, 32
 ggEulerQuaternion, 33
 ggFrustum, 34
 ggIdentity, 35
 ggIdentityQuaternion, 36
 ggInit, 36
 ggInvert, 37, 38
 ggLength3, 38, 39
 ggLength4, 40
 ggLoadComputeShader, 41
 ggLoadHeight, 42
 ggLoadImage, 42
 ggLoadShader, 43, 44
 ggLoadSimpleObj, 45, 46
 ggLoadTexture, 46
 ggLookat, 47–49
 ggMatrixQuaternion, 50, 51
 ggNorm, 51
 ggNormal, 52
 ggNormalize, 52
 ggNormalize3, 53–55
 ggNormalize4, 56–58
 ggOrthogonal, 58
 ggPerspective, 59
 ggPointsCube, 60
 ggPointsSphere, 61
 ggQuaternion, 61, 62
 ggQuaternionMatrix, 63
 ggQuaternionTransposeMatrix, 63
 ggReadImage, 64
 ggRectangle, 65
 ggRotate, 65, 67–69
 ggRotateQuaternion, 70, 71
 ggRotateX, 72
 ggRotateY, 73
 ggRotateZ, 73
 ggSaveColor, 74
 ggSaveDepth, 75
 ggSaveTga, 75
 ggScale, 76, 77
 ggSlerp, 78–80
 ggTranslate, 81, 82
 ggTranspose, 83
 LightBindingPoint, 18
 MaterialBindingPoint, 18
 operator*, 83
 operator+, 84
 operator-, 85
 operator/, 85
gg.cpp, 365
gg.h, 365
 ggError, 370
 ggFBOError, 370
gg::GgBuffer< T >, 93
 ~GgBuffer, 95
 bind, 95
 copy, 95
 getBuffer, 96
 getCount, 96
 getStride, 96
 getTarget, 96
 GgBuffer, 94, 95
 map, 97
 operator=, 98
 read, 98
 send, 98
 unbind, 99
 unmap, 99
gg::GgColorTexture, 99
 ~GgColorTexture, 101
 GgColorTexture, 100, 101
 load, 101, 102
gg::GgElements, 103
 ~GgElements, 105
 draw, 105

getIndexBuffer, 106
 getIndexCount, 106
 GgElements, 104, 105
 load, 106
 send, 107
 gg::GgMatrix, 107
 frustum, 112
 get, 113, 114
 GgMatrix, 110, 111
 invert, 115
 loadFrustum, 115
 loadIdentity, 116
 loadInvert, 116, 117
 loadLookat, 118, 119
 loadNormal, 120, 121
 loadOrthogonal, 121
 loadPerspective, 122
 loadRotate, 123–126
 loadRotateX, 126
 loadRotateY, 127
 loadRotateZ, 128
 loadScale, 128–130
 loadTranslate, 130–132
 loadTranspose, 132, 133
 lookat, 134, 135
 normal, 136
 operator*, 137, 138
 operator*=, 139, 140
 operator+, 141
 operator+=, 142, 143
 operator-, 143, 144
 operator-=, 144, 145
 operator/, 146
 operator/=, 147
 operator=, 148
 orthogonal, 149
 perspective, 150
 projection, 151, 152
 rotate, 152–154
 rotateX, 155
 rotateY, 156
 rotateZ, 157
 scale, 157–159
 translate, 160, 161
 transpose, 162
 gg::GgNormalTexture, 163
 ~GgNormalTexture, 165
 GgNormalTexture, 163, 164
 load, 165, 166
 gg::GgPoints, 167
 ~GgPoints, 168
 draw, 169
 getBuffer, 169
 getCount, 169
 GgPoints, 168
 load, 170
 send, 170
 gg::GgPointShader, 171
 ~GgPointShader, 173
 get, 173
 GgPointShader, 172
 load, 173, 174
 loadMatrix, 175
 loadModelviewMatrix, 176
 loadProjectionMatrix, 176, 177
 unuse, 177
 use, 177–179
 gg::GgQuaternion, 180
 add, 184–186
 conjugate, 187
 divide, 188–190
 euler, 190, 191
 get, 192
 getConjugateMatrix, 192, 193
 getMatrix, 194, 195
 GgQuaternion, 183, 184
 invert, 195
 load, 196–198
 loadAdd, 199–201
 loadConjugate, 201, 202
 loadDivide, 203–205
 loadEuler, 206
 loadIdentity, 207
 loadInvert, 208, 209
 loadMatrix, 210
 loadMultiply, 212–214
 loadNormalize, 214, 215
 loadRotate, 216, 217
 loadRotateX, 218
 loadRotateY, 218
 loadRotateZ, 219
 loadSlerp, 220–222
 loadSubtract, 222–224
 multiply, 225, 226
 norm, 227
 normalize, 227
 operator*, 228, 229
 operator*=, 229, 230
 operator+, 230, 231
 operator+=, 232
 operator-, 233
 operator-=, 234, 235
 operator/, 235, 236
 operator/=, 237
 operator=, 238
 rotate, 239, 240
 rotateX, 241
 rotateY, 242
 rotateZ, 242
 slerp, 243
 subtract, 244, 245
 gg::GgShader, 247
 ~GgShader, 248
 get, 248
 GgShader, 247, 248
 operator=, 249

unuse, 249
use, 249
gg::GgShape, 250
~GgShape, 251
draw, 251
get, 252
getMode, 252
GgShape, 250, 251
operator=, 252
setMode, 253
gg::GgSimpleObj, 253
~GgSimpleObj, 254
draw, 254
get, 255
GgSimpleObj, 253
gg::GgSimpleShader, 256
~GgSimpleShader, 258
GgSimpleShader, 257, 258
load, 259
loadMatrix, 260, 261
loadModelviewMatrix, 262, 263
operator=, 264
use, 264–270
gg::GgSimpleShader::Light, 313
ambient, 314
diffuse, 314
position, 314
specular, 314
gg::GgSimpleShader::LightBuffer, 315
~LightBuffer, 317
LightBuffer, 316
load, 317
loadAmbient, 318, 319
loadColor, 319
loadDiffuse, 319, 320
loadPosition, 321, 322
loadSpecular, 322, 323
select, 324
gg::GgSimpleShader::Material, 324
ambient, 325
diffuse, 325
shininess, 326
specular, 326
gg::GgSimpleShader::MaterialBuffer, 326
~MaterialBuffer, 328
load, 328, 329
loadAmbient, 329
loadAmbientAndDiffuse, 330
loadDiffuse, 331
loadShininess, 332
loadSpecular, 333
MaterialBuffer, 327, 328
select, 333
gg::GgTexture, 270
~GgTexture, 272
bind, 272
getHeight, 272
getSize, 272, 273
getTexture, 273
getWidth, 274
GgTexture, 271, 272
operator=, 274
unbind, 274
gg::GgTrackball, 275
~GgTrackball, 277
begin, 277
end, 277
get, 278
getMatrix, 278
getQuaternion, 279
getScale, 279, 280
getStart, 280, 281
GgTrackball, 276
motion, 281
operator=, 281
region, 282, 283
reset, 283
rotate, 284
gg::GgTriangles, 284
~GgTriangles, 286
draw, 287
getBuffer, 287
getCount, 287
GgTriangles, 286
load, 288
send, 288
gg::GgUniformBuffer< T >, 289
~GgUniformBuffer, 291
bind, 291
copy, 291
fill, 292
getBuffer, 292
getCount, 292
getStride, 292
getTarget, 293
GgUniformBuffer, 290
load, 293
map, 294
read, 295
send, 295
unbind, 296
unmap, 296
gg::GgVector, 296
distance3, 299
distance4, 299
dot3, 300
dot4, 301
GgVector, 298, 299
length3, 301
length4, 302
normalize3, 302
normalize4, 303
operator*, 303, 304
operator*=, 304, 305
operator+, 305
operator+=, 306

operator-, 307
 operator-=, 307, 308
 operator/, 308, 309
 operator/=, 309
gg::GgVertex, 310
 GgVertex, 311, 312
 normal, 312
 position, 312
GG_BUTTON_COUNT
 Window.h, 386
GG_INTERFACE_COUNT
 Window.h, 386
GG_USE_IMGUI
 GgApp.h, 371
GgApp, 92
 main, 92
GgApp.h, 370
 GG_USE_IMGUI, 371
ggArraysObj
 gg, 19
GgBuffer
 gg::GgBuffer< T >, 94, 95
ggBufferAlignment
 gg, 86
GgColorTexture
 gg::GgColorTexture, 100, 101
ggConjugate
 gg, 20
ggCreateComputeShader
 gg, 20
ggCreateNormalMap
 gg, 21
ggCreateShader
 gg, 22
ggCross
 gg, 23
ggDistance3
 gg, 24
ggDistance4
 gg, 25, 26
ggDot3
 gg, 27, 28
ggDot4
 gg, 28, 29
GgElements
 gg::GgElements, 104, 105
ggElementsMesh
 gg, 30
ggElementsObj
 gg, 31
ggElementsSphere
 gg, 32
ggEllipse
 gg, 32
ggError
 gg.h, 370
ggEulerQuaternion
 gg, 33

ggFBOError
 gg.h, 370
ggFrustum
 gg, 34
ggIdentity
 gg, 35
ggIdentityQuaternion
 gg, 36
ggInit
 gg, 36
ggInvert
 gg, 37, 38
ggLength3
 gg, 38, 39
ggLength4
 gg, 40
ggLoadComputeShader
 gg, 41
ggLoadHeight
 gg, 42
ggLoadImage
 gg, 42
ggLoadShader
 gg, 43, 44
ggLoadSimpleObj
 gg, 45, 46
ggLoadTexture
 gg, 46
ggLookat
 gg, 47–49
GgMatrix
 gg::GgMatrix, 110, 111
ggMatrixQuaternion
 gg, 50, 51
ggNorm
 gg, 51
ggNormal
 gg, 52
ggNormalize
 gg, 52
ggNormalize3
 gg, 53–55
ggNormalize4
 gg, 56–58
GgNormalTexture
 gg::GgNormalTexture, 163, 164
ggOrthogonal
 gg, 58
ggPerspective
 gg, 59
GgPoints
 gg::GgPoints, 168
ggPointsCube
 gg, 60
GgPointShader
 gg::GgPointShader, 172
ggPointsSphere
 gg, 61

GgQuaternion
 gg::GgQuaternion, 183, 184

ggQuaternion
 gg, 61, 62

ggQuaternionMatrix
 gg, 63

ggQuaternionTransposeMatrix
 gg, 63

ggReadImage
 gg, 64

ggRectangle
 gg, 65

ggRotate
 gg, 65, 67–69

ggRotateQuaternion
 gg, 70, 71

ggRotateX
 gg, 72

ggRotateY
 gg, 73

ggRotateZ
 gg, 73

ggsample01.cpp, 372
 CONFIG_FILE, 372
 PROJECT_NAME, 372

ggSaveColor
 gg, 74

ggSaveDepth
 gg, 75

ggSaveTga
 gg, 75

ggScale
 gg, 76, 77

GgShader
 gg::GgShader, 247, 248

GgShape
 gg::GgShape, 250, 251

GgSimpleObj
 gg::GgSimpleObj, 253

GgSimpleShader
 gg::GgSimpleShader, 257, 258

ggSlerp
 gg, 78–80

GgTexture
 gg::GgTexture, 271, 272

GgTrackball
 gg::GgTrackball, 276

ggTranslate
 gg, 81, 82

ggTranspose
 gg, 83

GgTriangles
 gg::GgTriangles, 286

GgUniformBuffer
 gg::GgUniformBuffer< T >, 290

GgVector
 gg::GgVector, 298, 299

GgVertex
 gg::GgVertex, 311, 312

HEADER_STR
 main.cpp, 373

initialize
 Window, 351

invert
 gg::GgMatrix, 115
 gg::GgQuaternion, 195

length3
 gg::GgVector, 301

length4
 gg::GgVector, 302

LightBindingPoint
 gg, 18

LightBuffer
 gg::GgSimpleShader::LightBuffer, 316

load
 Config, 88
 gg::GgColorTexture, 101, 102
 gg::GgElements, 106
 gg::GgNormalTexture, 165, 166
 gg::GgPoints, 170
 gg::GgPointShader, 173, 174
 gg::GgQuaternion, 196–198
 gg::GgSimpleShader, 259
 gg::GgSimpleShader::LightBuffer, 317
 gg::GgSimpleShader::MaterialBuffer, 328, 329
 gg::GgTriangles, 288
 gg::GgUniformBuffer< T >, 293

loadAdd
 gg::GgQuaternion, 199–201

loadAmbient
 gg::GgSimpleShader::LightBuffer, 318, 319
 gg::GgSimpleShader::MaterialBuffer, 329

loadAmbientAndDiffuse
 gg::GgSimpleShader::MaterialBuffer, 330

loadColor
 gg::GgSimpleShader::LightBuffer, 319

loadConjugate
 gg::GgQuaternion, 201, 202

loadDiffuse
 gg::GgSimpleShader::LightBuffer, 319, 320
 gg::GgSimpleShader::MaterialBuffer, 331

loadDivide
 gg::GgQuaternion, 203–205

loadEuler
 gg::GgQuaternion, 206

loadFrustum
 gg::GgMatrix, 115

loadIdentity
 gg::GgMatrix, 116
 gg::GgQuaternion, 207

loadInvert
 gg::GgMatrix, 116, 117
 gg::GgQuaternion, 208, 209

loadLookat

gg::GgMatrix, 118, 119
loadMatrix
 gg::GgPointShader, 175
 gg::GgQuaternion, 210
 gg::GgSimpleShader, 260, 261
loadModelviewMatrix
 gg::GgPointShader, 176
 gg::GgSimpleShader, 262, 263
loadMultiply
 gg::GgQuaternion, 212–214
loadNormal
 gg::GgMatrix, 120, 121
loadNormalize
 gg::GgQuaternion, 214, 215
loadOrthogonal
 gg::GgMatrix, 121
loadPerspective
 gg::GgMatrix, 122
loadPosition
 gg::GgSimpleShader::LightBuffer, 321, 322
loadProjectionMatrix
 gg::GgPointShader, 176, 177
loadRotate
 gg::GgMatrix, 123–126
 gg::GgQuaternion, 216, 217
loadRotateX
 gg::GgMatrix, 126
 gg::GgQuaternion, 218
loadRotateY
 gg::GgMatrix, 127
 gg::GgQuaternion, 218
loadRotateZ
 gg::GgMatrix, 128
 gg::GgQuaternion, 219
loadScale
 gg::GgMatrix, 128–130
loadShininess
 gg::GgSimpleShader::MaterialBuffer, 332
loadSlerp
 gg::GgQuaternion, 220–222
loadSpecular
 gg::GgSimpleShader::LightBuffer, 322, 323
 gg::GgSimpleShader::MaterialBuffer, 333
loadSubtract
 gg::GgQuaternion, 222–224
loadTranslate
 gg::GgMatrix, 130–132
loadTranspose
 gg::GgMatrix, 132, 133
lookat
 gg::GgMatrix, 134, 135
main
 GgApp, 92
 main.cpp, 373
main.cpp, 373
 HEADER_STR, 373
 main, 373
map
 gg::GgBuffer< T >, 97
 gg::GgUniformBuffer< T >, 294
MaterialBindingPoint
 gg, 18
MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 327, 328
Menu, 334
 ~Menu, 335
 Config, 90
 Draw, 335
 draw, 335
 Menu, 334, 335
 operator=, 335
Menu.cpp, 374
 menuFont, 375
 menuFontSize, 375
Menu.h, 375
menuFont
 Menu.cpp, 375
menuFontSize
 Menu.cpp, 375
motion
 gg::GgTrackball, 281
multiply
 gg::GgQuaternion, 225, 226
norm
 gg::GgQuaternion, 227
normal
 gg::GgMatrix, 136
 gg::GgVertex, 312
normalize
 gg::GgQuaternion, 227
normalize3
 gg::GgVector, 302
normalize4
 gg::GgVector, 303
operator bool
 Window, 352
operator*
 gg, 83
 gg::GgMatrix, 137, 138
 gg::GgQuaternion, 228, 229
 gg::GgVector, 303, 304
operator*=
 gg::GgMatrix, 139, 140
 gg::GgQuaternion, 229, 230
 gg::GgVector, 304, 305
operator+
 gg, 84
 gg::GgMatrix, 141
 gg::GgQuaternion, 230, 231
 gg::GgVector, 305
operator+=
 gg::GgMatrix, 142, 143
 gg::GgQuaternion, 232
 gg::GgVector, 306
operator-

gg, 85
gg::GgMatrix, 143, 144
gg::GgQuaternion, 233
gg::GgVector, 307
operator-=
 gg::GgMatrix, 144, 145
 gg::GgQuaternion, 234, 235
 gg::GgVector, 307, 308
operator/
 gg, 85
 gg::GgMatrix, 146
 gg::GgQuaternion, 235, 236
 gg::GgVector, 308, 309
operator/=
 gg::GgMatrix, 147
 gg::GgQuaternion, 237
 gg::GgVector, 309
operator=
 gg::GgBuffer< T >, 98
 gg::GgMatrix, 148
 gg::GgQuaternion, 238
 gg::GgShader, 249
 gg::GgShape, 252
 gg::GgSimpleShader, 264
 gg::GgTexture, 274
 gg::GgTrackball, 281
 Menu, 335
 Window, 352
orthogonal
 gg::GgMatrix, 149

parseconfig.h, 376
 getString, 378, 379
 getValue, 379, 380
 getVector, 380
 setString, 381, 382
 setValue, 382, 383
 setVector, 383
pathChar
 Config.h, 362
pathString
 Config.h, 362
perspective
 gg::GgMatrix, 150
position
 gg::GgSimpleShader::Light, 314
 gg::GgVertex, 312
PROJECT_NAME
 ggsample01.cpp, 372
projection
 gg::GgMatrix, 151, 152

read
 gg::GgBuffer< T >, 98
 gg::GgUniformBuffer< T >, 295
README.md, 384
region
 gg::GgTrackball, 282, 283
reset
 gg::GgTrackball, 283
 Window, 352
 resetRotation
 Window, 352
 resetTranslation
 Window, 353
 restoreViewport
 Window, 353
 rotate
 gg::GgMatrix, 152–154
 gg::GgQuaternion, 239, 240
 gg::GgTrackball, 284
 rotateX
 gg::GgMatrix, 155
 gg::GgQuaternion, 241
 rotateY
 gg::GgMatrix, 156
 gg::GgQuaternion, 242
 rotateZ
 gg::GgMatrix, 157
 gg::GgQuaternion, 242
 save
 Config, 89
 scale
 gg::GgMatrix, 157–159
 select
 gg::GgSimpleShader::LightBuffer, 324
 gg::GgSimpleShader::MaterialBuffer, 333
 selectInterface
 Window, 353
send
 gg::GgBuffer< T >, 98
 gg::GgElements, 107
 gg::GgPoints, 170
 gg::GgTriangles, 288
 gg::GgUniformBuffer< T >, 295
setClose
 Window, 353
setKeyboardFunc
 Window, 355
setMenubarHeight
 Window, 355
setMode
 gg::GgShape, 253
setMouseFunc
 Window, 355
setResizeFunc
 Window, 356
setString
 parseconfig.h, 381, 382
setUserPointer
 Window, 356
setValue
 parseconfig.h, 382, 383
setVector
 parseconfig.h, 383
setVelocity
 Window, 356

setWheelFunc
 Window, 357

shininess
 gg::GgSimpleShader::Material, 326

shouldClose
 Window, 357

slerp
 gg::GgQuaternion, 243

specular
 gg::GgSimpleShader::Light, 314
 gg::GgSimpleShader::Material, 326

subtract
 gg::GgQuaternion, 244, 245

swapBuffers
 Window, 357

TCharToUtf8
 Config.h, 362

translate
 gg::GgMatrix, 160, 161

transpose
 gg::GgMatrix, 162

unbind
 gg::GgBuffer< T >, 99
 gg::GgTexture, 274
 gg::GgUniformBuffer< T >, 296

unmap
 gg::GgBuffer< T >, 99
 gg::GgUniformBuffer< T >, 296

unuse
 gg::GgPointShader, 177
 gg::GgShader, 249

use
 gg::GgPointShader, 177–179
 gg::GgShader, 249
 gg::GgSimpleShader, 264–270

Utf8ToTChar
 Config.h, 362

Window, 336

 ~Window, 338

 get, 338

 getAltArrowX, 339

 getAltArrowY, 339

 getAltArrow, 339

 getArrow, 340

 getArrowX, 340

 getArrowY, 341

 getAspect, 341

 getControlArrow, 341

 getControlArrowX, 342

 getControlArrowY, 342

 getFboHeight, 342

 getFboSize, 343

 getFboWidth, 343

 getHeight, 344

 getKey, 344

 getLastKey, 344

 getMouse, 344, 345

 getMouseX, 345

 getMouseY, 346

 getRotation, 346

 getRotationMatrix, 346

 getScrollMatrix, 347

 getShiftArrow, 347

 getShiftArrowX, 348

 getShiftArrowY, 348

 getSize, 348

 getTranslation, 349

 getTranslationMatrix, 349

 getUserPointer, 349

 getWheel, 350

 getWheelX, 351

 getWheelY, 351

 getWidth, 351

 initialize, 351

 operator bool, 352

 operator=, 352

 reset, 352

 resetRotation, 352

 resetTranslation, 353

 restoreViewport, 353

 selectInterface, 353

 setClose, 353

 setKeyboardFunc, 355

 setMenubarHeight, 355

 setMouseFunc, 355

 setResizeFunc, 356

 setUserPointer, 356

 setVelocity, 356

 setWheelFunc, 357

 shouldClose, 357

 swapBuffers, 357

 Window, 337, 338

Window.h, 384

 GG_BUTTON_COUNT, 386

 GG_INTERFACE_COUNT, 386