

ゲームグラフィックス特論

構築: Doxygen 1.9.5

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]	24
7.1.3.10 ggDistance3() [1/2]	24
7.1.3.11 ggDistance3() [2/2]	25
7.1.3.12 ggDistance4() [1/2]	26
7.1.3.13 ggDistance4() [2/2]	27

7.1.3.14 ggDot3() [1/2]	28
7.1.3.15 ggDot3() [2/2]	29
7.1.3.16 ggDot4() [1/2]	30
7.1.3.17 ggDot4() [2/2]	30
7.1.3.18 ggElementsMesh()	31
7.1.3.19 ggElementsObj()	32
7.1.3.20 ggElementsSphere()	33
7.1.3.21 ggEllipse()	34
7.1.3.22 ggEulerQuaternion() [1/2]	34
7.1.3.23 ggEulerQuaternion() [2/2]	35
7.1.3.24 ggFrustum()	36
7.1.3.25 ggIdentity()	36
7.1.3.26 ggIdentityQuaternion()	37
7.1.3.27 ggInit()	38
7.1.3.28 ggInvert() [1/2]	38
7.1.3.29 ggInvert() [2/2]	39
7.1.3.30 ggLength3() [1/2]	39
7.1.3.31 ggLength3() [2/2]	40
7.1.3.32 ggLength4() [1/2]	41
7.1.3.33 ggLength4() [2/2]	42
7.1.3.34 ggLoadComputeShader()	42
7.1.3.35 ggLoadHeight()	43
7.1.3.36 ggLoadImage()	44
7.1.3.37 ggLoadShader() [1/2]	45
7.1.3.38 ggLoadShader() [2/2]	45
7.1.3.39 ggLoadSimpleObj() [1/2]	46
7.1.3.40 ggLoadSimpleObj() [2/2]	47
7.1.3.41 ggLoadTexture()	48
7.1.3.42 ggLookat() [1/3]	49
7.1.3.43 ggLookat() [2/3]	49
7.1.3.44 ggLookat() [3/3]	50
7.1.3.45 ggMatrixQuaternion() [1/2]	51
7.1.3.46 ggMatrixQuaternion() [2/2]	52
7.1.3.47 ggNorm()	53
7.1.3.48 ggNormal()	53
7.1.3.49 ggNormalize()	54
7.1.3.50 ggNormalize3() [1/4]	54
7.1.3.51 ggNormalize3() [2/4]	55
7.1.3.52 ggNormalize3() [3/4]	56
7.1.3.53 ggNormalize3() [4/4]	57
7.1.3.54 ggNormalize4() [1/4]	57
7.1.3.55 ggNormalize4() [2/4]	58

7.1.3.56 ggNormalize4() [3/4]	58
7.1.3.57 ggNormalize4() [4/4]	59
7.1.3.58 ggOrthogonal()	59
7.1.3.59 ggPerspective()	60
7.1.3.60 ggPointsCube()	61
7.1.3.61 ggPointsSphere()	62
7.1.3.62 ggQuaternion() [1/2]	62
7.1.3.63 ggQuaternion() [2/2]	63
7.1.3.64 ggQuaternionMatrix()	64
7.1.3.65 ggQuaternionTransposeMatrix()	64
7.1.3.66 ggReadImage()	65
7.1.3.67 ggRectangle()	66
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]	88
8.1.2.2 Config() [2/2]	88
8.1.2.3 ~Config()	88
8.1.3 関数詳解	89
8.1.3.1 getHeight()	89
8.1.3.2 getWidth()	89
8.1.3.3 load()	89
8.1.3.4 save()	90
8.1.4 フレンドと関連関数の詳解	91
8.1.4.1 Menu	91
8.2 Draw クラス	91
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()	92
8.2.3 関数詳解	92
8.2.3.1 draw()	92
8.3 GgApp クラス	92
8.3.1 詳解	93
8.3.2 構築子と解体子	93
8.3.2.1 GgApp() [1/2]	93
8.3.2.2 GgApp() [2/2]	93
8.3.2.3 ~GgApp()	93
8.3.3 関数詳解	94
8.3.3.1 main()	94
8.3.3.2 operator=()	94
8.4 gg::GgBuffer< T > クラステンプレート	95
8.4.1 詳解	95
8.4.2 構築子と解体子	95
8.4.2.1 GgBuffer() [1/2]	95
8.4.2.2 ~GgBuffer()	96
8.4.2.3 GgBuffer() [2/2]	96
8.4.3 関数詳解	96
8.4.3.1 bind()	96

8.4.3.2 <code>copy()</code>	96
8.4.3.3 <code>getBuffer()</code>	97
8.4.3.4 <code>getCount()</code>	97
8.4.3.5 <code>getStride()</code>	97
8.4.3.6 <code>getTarget()</code>	98
8.4.3.7 <code>map() [1/2]</code>	98
8.4.3.8 <code>map() [2/2]</code>	98
8.4.3.9 <code>operator=()</code>	99
8.4.3.10 <code>read()</code>	99
8.4.3.11 <code>send()</code>	99
8.4.3.12 <code>unbind()</code>	100
8.4.3.13 <code>unmap()</code>	100
8.5 <code>gg::GgColorTexture</code> クラス	100
8.5.1 詳解	100
8.5.2 構築子と解体子	101
8.5.2.1 <code>GgColorTexture()</code> [1/3]	101
8.5.2.2 <code>GgColorTexture()</code> [2/3]	101
8.5.2.3 <code>GgColorTexture()</code> [3/3]	102
8.5.2.4 <code>~GgColorTexture()</code>	103
8.5.3 関数詳解	103
8.5.3.1 <code>load() [1/2]</code>	103
8.5.3.2 <code>load() [2/2]</code>	104
8.6 <code>gg::GgElements</code> クラス	105
8.6.1 詳解	106
8.6.2 構築子と解体子	106
8.6.2.1 <code>GgElements()</code> [1/2]	106
8.6.2.2 <code>GgElements()</code> [2/2]	106
8.6.2.3 <code>~GgElements()</code>	107
8.6.3 関数詳解	107
8.6.3.1 <code>draw()</code>	107
8.6.3.2 <code>getIndexBuffer()</code>	108
8.6.3.3 <code>getIndexCount()</code>	108
8.6.3.4 <code>load()</code>	108
8.6.3.5 <code>send()</code>	109
8.7 <code>gg::GgMatrix</code> クラス	109
8.7.1 詳解	112
8.7.2 構築子と解体子	112
8.7.2.1 <code>GgMatrix()</code> [1/4]	112
8.7.2.2 <code>GgMatrix()</code> [2/4]	112
8.7.2.3 <code>GgMatrix()</code> [3/4]	113
8.7.2.4 <code>GgMatrix()</code> [4/4]	113
8.7.3 関数詳解	114

8.7.3.1 frustum()	114
8.7.3.2 get() [1/2]	115
8.7.3.3 get() [2/2]	116
8.7.3.4 invert()	117
8.7.3.5 loadFrustum()	117
8.7.3.6 loadIdentity()	118
8.7.3.7 loadInvert() [1/2]	119
8.7.3.8 loadInvert() [2/2]	119
8.7.3.9 loadLookat() [1/3]	120
8.7.3.10 loadLookat() [2/3]	121
8.7.3.11 loadLookat() [3/3]	121
8.7.3.12 loadNormal() [1/2]	122
8.7.3.13 loadNormal() [2/2]	123
8.7.3.14 loadOrthogonal()	124
8.7.3.15 loadPerspective()	124
8.7.3.16 loadRotate() [1/5]	125
8.7.3.17 loadRotate() [2/5]	126
8.7.3.18 loadRotate() [3/5]	126
8.7.3.19 loadRotate() [4/5]	127
8.7.3.20 loadRotate() [5/5]	128
8.7.3.21 loadRotateX()	129
8.7.3.22 loadRotateY()	129
8.7.3.23 loadRotateZ()	130
8.7.3.24 loadScale() [1/3]	131
8.7.3.25 loadScale() [2/3]	131
8.7.3.26 loadScale() [3/3]	132
8.7.3.27 loadTranslate() [1/3]	133
8.7.3.28 loadTranslate() [2/3]	133
8.7.3.29 loadTranslate() [3/3]	134
8.7.3.30 loadTranspose() [1/2]	135
8.7.3.31 loadTranspose() [2/2]	135
8.7.3.32 lookat() [1/3]	136
8.7.3.33 lookat() [2/3]	137
8.7.3.34 lookat() [3/3]	137
8.7.3.35 normal()	139
8.7.3.36 operator*() [1/3]	139
8.7.3.37 operator*() [2/3]	140
8.7.3.38 operator*() [3/3]	140
8.7.3.39 operator*=(()) [1/2]	141
8.7.3.40 operator*=(()) [2/2]	142
8.7.3.41 operator+() [1/2]	143
8.7.3.42 operator+() [2/2]	143

8.7.3.43 operator+=() [1/2]	144
8.7.3.44 operator+=() [2/2]	145
8.7.3.45 operator-() [1/2]	145
8.7.3.46 operator-() [2/2]	146
8.7.3.47 operator-=(() [1/2]	146
8.7.3.48 operator-=(() [2/2]	147
8.7.3.49 operator/() [1/2]	148
8.7.3.50 operator/() [2/2]	148
8.7.3.51 operator/=(() [1/2]	149
8.7.3.52 operator/=(() [2/2]	150
8.7.3.53 operator=()	150
8.7.3.54 orthogonal()	151
8.7.3.55 perspective()	152
8.7.3.56 projection() [1/4]	153
8.7.3.57 projection() [2/4]	153
8.7.3.58 projection() [3/4]	153
8.7.3.59 projection() [4/4]	154
8.7.3.60 rotate() [1/5]	154
8.7.3.61 rotate() [2/5]	155
8.7.3.62 rotate() [3/5]	155
8.7.3.63 rotate() [4/5]	156
8.7.3.64 rotate() [5/5]	157
8.7.3.65 rotateX()	158
8.7.3.66 rotateY()	158
8.7.3.67 rotateZ()	159
8.7.3.68 scale() [1/3]	160
8.7.3.69 scale() [2/3]	160
8.7.3.70 scale() [3/3]	161
8.7.3.71 translate() [1/3]	162
8.7.3.72 translate() [2/3]	162
8.7.3.73 translate() [3/3]	163
8.7.3.74 transpose()	164
8.8 gg::GgNormalTexture クラス	165
8.8.1 詳解	165
8.8.2 構築子と解体子	165
8.8.2.1 GgNormalTexture() [1/3]	166
8.8.2.2 GgNormalTexture() [2/3]	166
8.8.2.3 GgNormalTexture() [3/3]	166
8.8.2.4 ~GgNormalTexture()	167
8.8.3 関数詳解	167
8.8.3.1 load() [1/2]	167
8.8.3.2 load() [2/2]	168

8.9 gg::GgPoints クラス	169
8.9.1 詳解	170
8.9.2 構築子と解体子	170
8.9.2.1 GgPoints() [1/2]	170
8.9.2.2 GgPoints() [2/2]	170
8.9.2.3 ~GgPoints()	170
8.9.3 関数詳解	171
8.9.3.1 draw()	171
8.9.3.2 getBuffer()	171
8.9.3.3 getCount()	172
8.9.3.4 load()	172
8.9.3.5 send()	172
8.10 gg::GgPointShader クラス	173
8.10.1 詳解	174
8.10.2 構築子と解体子	174
8.10.2.1 GgPointShader() [1/3]	174
8.10.2.2 GgPointShader() [2/3]	174
8.10.2.3 GgPointShader() [3/3]	174
8.10.2.4 ~GgPointShader()	175
8.10.3 関数詳解	175
8.10.3.1 get()	175
8.10.3.2 load() [1/2]	175
8.10.3.3 load() [2/2]	176
8.10.3.4 loadMatrix() [1/2]	177
8.10.3.5 loadMatrix() [2/2]	177
8.10.3.6 loadModelviewMatrix() [1/2]	178
8.10.3.7 loadModelviewMatrix() [2/2]	178
8.10.3.8 loadProjectionMatrix() [1/2]	179
8.10.3.9 loadProjectionMatrix() [2/2]	179
8.10.3.10 unuse()	179
8.10.3.11 use() [1/5]	180
8.10.3.12 use() [2/5]	180
8.10.3.13 use() [3/5]	180
8.10.3.14 use() [4/5]	181
8.10.3.15 use() [5/5]	181
8.11 gg::GgQuaternion クラス	182
8.11.1 詳解	185
8.11.2 構築子と解体子	185
8.11.2.1 GgQuaternion() [1/5]	185
8.11.2.2 GgQuaternion() [2/5]	185
8.11.2.3 GgQuaternion() [3/5]	185
8.11.2.4 GgQuaternion() [4/5]	186

8.11.2.5 GgQuaternion() [5/5]	186
8.11.3 関数詳解	186
8.11.3.1 add() [1/4]	186
8.11.3.2 add() [2/4]	187
8.11.3.3 add() [3/4]	188
8.11.3.4 add() [4/4]	188
8.11.3.5 conjugate()	189
8.11.3.6 divide() [1/4]	190
8.11.3.7 divide() [2/4]	190
8.11.3.8 divide() [3/4]	191
8.11.3.9 divide() [4/4]	192
8.11.3.10 euler() [1/2]	193
8.11.3.11 euler() [2/2]	193
8.11.3.12 get()	194
8.11.3.13 getConjugateMatrix() [1/3]	194
8.11.3.14 getConjugateMatrix() [2/3]	195
8.11.3.15 getConjugateMatrix() [3/3]	196
8.11.3.16 getMatrix() [1/3]	196
8.11.3.17 getMatrix() [2/3]	197
8.11.3.18 getMatrix() [3/3]	198
8.11.3.19 invert()	198
8.11.3.20 load() [1/4]	199
8.11.3.21 load() [2/4]	200
8.11.3.22 load() [3/4]	200
8.11.3.23 load() [4/4]	201
8.11.3.24 loadAdd() [1/4]	201
8.11.3.25 loadAdd() [2/4]	202
8.11.3.26 loadAdd() [3/4]	203
8.11.3.27 loadAdd() [4/4]	203
8.11.3.28 loadConjugate() [1/2]	204
8.11.3.29 loadConjugate() [2/2]	205
8.11.3.30 loadDivide() [1/4]	205
8.11.3.31 loadDivide() [2/4]	206
8.11.3.32 loadDivide() [3/4]	206
8.11.3.33 loadDivide() [4/4]	207
8.11.3.34 loadEuler() [1/2]	208
8.11.3.35 loadEuler() [2/2]	209
8.11.3.36 loadIdentity()	210
8.11.3.37 loadInvert() [1/2]	210
8.11.3.38 loadInvert() [2/2]	211
8.11.3.39 loadMatrix() [1/2]	212
8.11.3.40 loadMatrix() [2/2]	212

8.11.3.41 loadMultiply() [1/4]	213
8.11.3.42 loadMultiply() [2/4]	214
8.11.3.43 loadMultiply() [3/4]	214
8.11.3.44 loadMultiply() [4/4]	215
8.11.3.45 loadNormalize() [1/2]	216
8.11.3.46 loadNormalize() [2/2]	216
8.11.3.47 loadRotate() [1/3]	217
8.11.3.48 loadRotate() [2/3]	218
8.11.3.49 loadRotate() [3/3]	218
8.11.3.50 loadRotateX()	219
8.11.3.51 loadRotateY()	220
8.11.3.52 loadRotateZ()	220
8.11.3.53 loadSlerp() [1/4]	221
8.11.3.54 loadSlerp() [2/4]	222
8.11.3.55 loadSlerp() [3/4]	223
8.11.3.56 loadSlerp() [4/4]	223
8.11.3.57 loadSubtract() [1/4]	224
8.11.3.58 loadSubtract() [2/4]	225
8.11.3.59 loadSubtract() [3/4]	225
8.11.3.60 loadSubtract() [4/4]	226
8.11.3.61 multiply() [1/4]	227
8.11.3.62 multiply() [2/4]	227
8.11.3.63 multiply() [3/4]	227
8.11.3.64 multiply() [4/4]	228
8.11.3.65 norm()	228
8.11.3.66 normalize()	229
8.11.3.67 operator*() [1/3]	230
8.11.3.68 operator*() [2/3]	230
8.11.3.69 operator*() [3/3]	230
8.11.3.70 operator*() [1/3]	231
8.11.3.71 operator*() [2/3]	231
8.11.3.72 operator*() [3/3]	231
8.11.3.73 operator+() [1/3]	232
8.11.3.74 operator+() [2/3]	232
8.11.3.75 operator+() [3/3]	233
8.11.3.76 operator+=() [1/3]	233
8.11.3.77 operator+=() [2/3]	234
8.11.3.78 operator+=() [3/3]	234
8.11.3.79 operator-() [1/3]	234
8.11.3.80 operator-() [2/3]	235
8.11.3.81 operator-() [3/3]	235
8.11.3.82 operator-() [1/3]	236

8.11.3.83 operator-() [2/3]	236
8.11.3.84 operator-() [3/3]	236
8.11.3.85 operator/() [1/3]	237
8.11.3.86 operator/() [2/3]	237
8.11.3.87 operator/() [3/3]	238
8.11.3.88 operator/=(()) [1/3]	238
8.11.3.89 operator/=(()) [2/3]	239
8.11.3.90 operator/=(()) [3/3]	239
8.11.3.91 operator=(()) [1/2]	240
8.11.3.92 operator=(()) [2/2]	240
8.11.3.93 rotate() [1/3]	240
8.11.3.94 rotate() [2/3]	241
8.11.3.95 rotate() [3/3]	242
8.11.3.96 rotateX()	243
8.11.3.97 rotateY()	243
8.11.3.98 rotateZ()	244
8.11.3.99 slerp() [1/2]	244
8.11.3.100 slerp() [2/2]	245
8.11.3.101 subtract() [1/4]	245
8.11.3.102 subtract() [2/4]	246
8.11.3.103 subtract() [3/4]	246
8.11.3.104 subtract() [4/4]	247
8.12 gg::GgShader クラス	248
8.12.1 詳解	248
8.12.2 構築子と解体子	249
8.12.2.1 GgShader() [1/3]	249
8.12.2.2 GgShader() [2/3]	249
8.12.2.3 GgShader() [3/3]	250
8.12.2.4 ~GgShader()	250
8.12.3 関数詳解	250
8.12.3.1 get()	250
8.12.3.2 operator=(())	250
8.12.3.3 unuse()	251
8.12.3.4 use()	251
8.13 gg::GgShape クラス	251
8.13.1 詳解	252
8.13.2 構築子と解体子	252
8.13.2.1 GgShape() [1/2]	252
8.13.2.2 ~GgShape()	252
8.13.2.3 GgShape() [2/2]	253
8.13.3 関数詳解	253
8.13.3.1 draw()	253

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

8.15.3.22 use() [11/13]	271
8.15.3.23 use() [12/13]	271
8.15.3.24 use() [13/13]	272
8.16 gg::GgTexture クラス	272
8.16.1 詳解	273
8.16.2 構築子と解体子	273
8.16.2.1 GgTexture() [1/2]	273
8.16.2.2 ~GgTexture()	274
8.16.2.3 GgTexture() [2/2]	274
8.16.3 関数詳解	274
8.16.3.1 bind()	274
8.16.3.2 getHeight()	274
8.16.3.3 getSize() [1/2]	275
8.16.3.4 getSize() [2/2]	275
8.16.3.5 getTexture()	275
8.16.3.6 getWidth()	276
8.16.3.7 operator=()	276
8.16.3.8 swapRandB()	276
8.16.3.9 unbind()	277
8.17 gg::GgTrackball クラス	277
8.17.1 詳解	278
8.17.2 構築子と解体子	279
8.17.2.1 GgTrackball()	279
8.17.2.2 ~GgTrackball()	279
8.17.3 関数詳解	279
8.17.3.1 begin()	279
8.17.3.2 end()	280
8.17.3.3 get()	280
8.17.3.4 getMatrix()	281
8.17.3.5 getQuaternion()	281
8.17.3.6 getScale() [1/3]	282
8.17.3.7 getScale() [2/3]	282
8.17.3.8 getScale() [3/3]	282
8.17.3.9 getStart() [1/3]	282
8.17.3.10 getStart() [2/3]	283
8.17.3.11 getStart() [3/3]	283
8.17.3.12 motion()	283
8.17.3.13 operator=()	284
8.17.3.14 region() [1/2]	285
8.17.3.15 region() [2/2]	285
8.17.3.16 reset()	286
8.17.3.17 rotate()	286

8.18 gg::GgTriangles クラス	287
8.18.1 詳解	288
8.18.2 構築子と解体子	288
8.18.2.1 GgTriangles() [1/2]	288
8.18.2.2 GgTriangles() [2/2]	288
8.18.2.3 ~GgTriangles()	289
8.18.3 関数詳解	289
8.18.3.1 draw()	289
8.18.3.2 getBuffer()	290
8.18.3.3 getCount()	290
8.18.3.4 load()	290
8.18.3.5 send()	291
8.19 gg::GgUniformBuffer< T > クラステンプレート	291
8.19.1 詳解	292
8.19.2 構築子と解体子	292
8.19.2.1 GgUniformBuffer() [1/3]	292
8.19.2.2 GgUniformBuffer() [2/3]	292
8.19.2.3 GgUniformBuffer() [3/3]	293
8.19.2.4 ~GgUniformBuffer()	293
8.19.3 関数詳解	293
8.19.3.1 bind()	294
8.19.3.2 copy()	294
8.19.3.3 fill()	294
8.19.3.4 getBuffer()	295
8.19.3.5 getCount()	295
8.19.3.6 getStride()	295
8.19.3.7 getTarget()	296
8.19.3.8 load() [1/2]	296
8.19.3.9 load() [2/2]	296
8.19.3.10 map() [1/2]	297
8.19.3.11 map() [2/2]	297
8.19.3.12 read()	297
8.19.3.13 send()	298
8.19.3.14 unbind()	298
8.19.3.15 unmap()	299
8.20 gg::GgVector クラス	299
8.20.1 詳解	300
8.20.2 構築子と解体子	300
8.20.2.1 GgVector() [1/4]	300
8.20.2.2 GgVector() [2/4]	301
8.20.2.3 GgVector() [3/4]	302
8.20.2.4 GgVector() [4/4]	302

8.20.3 関数詳解	302
8.20.3.1 distance3()	303
8.20.3.2 distance4()	303
8.20.3.3 dot3()	304
8.20.3.4 dot4()	304
8.20.3.5 length3()	305
8.20.3.6 length4()	306
8.20.3.7 normalize3()	306
8.20.3.8 normalize4()	307
8.20.3.9 operator*() [1/2]	307
8.20.3.10 operator*() [2/2]	307
8.20.3.11 operator*() [1/2]	308
8.20.3.12 operator*() [2/2]	308
8.20.3.13 operator+() [1/2]	309
8.20.3.14 operator+() [2/2]	309
8.20.3.15 operator+=() [1/2]	309
8.20.3.16 operator+=() [2/2]	310
8.20.3.17 operator-() [1/2]	310
8.20.3.18 operator-() [2/2]	311
8.20.3.19 operator-=() [1/2]	311
8.20.3.20 operator-=() [2/2]	311
8.20.3.21 operator/() [1/2]	312
8.20.3.22 operator/() [2/2]	312
8.20.3.23 operator/() [1/2]	313
8.20.3.24 operator/() [2/2]	313
8.21 gg::GgVertex 構造体	313
8.21.1 詳解	314
8.21.2 構築子と解体子	314
8.21.2.1 GgVertex() [1/4]	315
8.21.2.2 GgVertex() [2/4]	315
8.21.2.3 GgVertex() [3/4]	315
8.21.2.4 GgVertex() [4/4]	316
8.21.3 メンバ詳解	316
8.21.3.1 normal	316
8.21.3.2 position	316
8.22 gg::GgSimpleShader::Light 構造体	317
8.22.1 詳解	317
8.22.2 メンバ詳解	317
8.22.2.1 ambient	318
8.22.2.2 diffuse	318
8.22.2.3 position	318
8.22.2.4 specular	318

8.23 gg::GgSimpleShader::LightBuffer クラス	319
8.23.1 詳解	320
8.23.2 構築子と解体子	320
8.23.2.1 LightBuffer() [1/2]	320
8.23.2.2 LightBuffer() [2/2]	320
8.23.2.3 ~LightBuffer()	321
8.23.3 関数詳解	321
8.23.3.1 load() [1/2]	321
8.23.3.2 load() [2/2]	321
8.23.3.3 loadAmbient() [1/3]	322
8.23.3.4 loadAmbient() [2/3]	322
8.23.3.5 loadAmbient() [3/3]	323
8.23.3.6 loadColor()	323
8.23.3.7 loadDiffuse() [1/3]	324
8.23.3.8 loadDiffuse() [2/3]	324
8.23.3.9 loadDiffuse() [3/3]	324
8.23.3.10 loadPosition() [1/4]	325
8.23.3.11 loadPosition() [2/4]	325
8.23.3.12 loadPosition() [3/4]	326
8.23.3.13 loadPosition() [4/4]	326
8.23.3.14 loadSpecular() [1/3]	327
8.23.3.15 loadSpecular() [2/3]	327
8.23.3.16 loadSpecular() [3/3]	327
8.23.3.17 select()	328
8.24 gg::GgSimpleShader::Material 構造体	328
8.24.1 詳解	329
8.24.2 メンバ詳解	329
8.24.2.1 ambient	329
8.24.2.2 diffuse	330
8.24.2.3 shininess	330
8.24.2.4 specular	330
8.25 gg::GgSimpleShader::MaterialBuffer クラス	330
8.25.1 詳解	331
8.25.2 構築子と解体子	331
8.25.2.1 MaterialBuffer() [1/2]	331
8.25.2.2 MaterialBuffer() [2/2]	332
8.25.2.3 ~MaterialBuffer()	332
8.25.3 関数詳解	332
8.25.3.1 load() [1/2]	332
8.25.3.2 load() [2/2]	333
8.25.3.3 loadAmbient() [1/2]	333
8.25.3.4 loadAmbient() [2/2]	334

8.25.3.5 loadAmbientAndDiffuse() [1/2]	334
8.25.3.6 loadAmbientAndDiffuse() [2/2]	334
8.25.3.7 loadDiffuse() [1/2]	335
8.25.3.8 loadDiffuse() [2/2]	335
8.25.3.9 loadShininess() [1/2]	336
8.25.3.10 loadShininess() [2/2]	336
8.25.3.11 loadSpecular() [1/2]	337
8.25.3.12 loadSpecular() [2/2]	337
8.25.3.13 select()	338
8.26 Menu クラス	338
8.26.1 詳解	338
8.26.2 構築子と解体子	338
8.26.2.1 Menu() [1/2]	339
8.26.2.2 Menu() [2/2]	339
8.26.2.3 ~Menu()	339
8.26.3 関数詳解	339
8.26.3.1 draw()	339
8.26.3.2 operator=()	339
8.26.4 フレンドと関連関数の詳解	339
8.26.4.1 Draw	340
8.27 GgApp::Window クラス	340
8.27.1 詳解	341
8.27.2 構築子と解体子	341
8.27.2.1 Window() [1/2]	341
8.27.2.2 Window() [2/2]	342
8.27.2.3 ~Window()	342
8.27.3 関数詳解	342
8.27.3.1 get()	343
8.27.3.2 getAltArrowX()	343
8.27.3.3 getAltArrowY()	344
8.27.3.4 getAltArrow()	344
8.27.3.5 getArrow() [1/2]	345
8.27.3.6 getArrow() [2/2]	345
8.27.3.7 getArrowX()	346
8.27.3.8 getArrowY()	347
8.27.3.9 getAspect()	348
8.27.3.10 getControlArrow()	348
8.27.3.11 getControlArrowX()	349
8.27.3.12 getControlArrowY()	350
8.27.3.13 getFboHeight()	350
8.27.3.14 getFboSize() [1/2]	351
8.27.3.15 getFboSize() [2/2]	351

8.27.3.16 getFboWidth()	352
8.27.3.17 getHeight()	352
8.27.3.18 getKey()	353
8.27.3.19 getLastKey()	353
8.27.3.20 getMouse() [1/3]	354
8.27.3.21 getMouse() [2/3]	354
8.27.3.22 getMouse() [3/3]	354
8.27.3.23 getMouseX()	355
8.27.3.24 getMouseY()	355
8.27.3.25 getRotation()	355
8.27.3.26 getRotationMatrix()	356
8.27.3.27 getScrollMatrix()	356
8.27.3.28 getShiftArrow()	356
8.27.3.29 getShiftArrowX()	357
8.27.3.30 getShiftArrowY()	358
8.27.3.31 getSize() [1/2]	358
8.27.3.32 getSize() [2/2]	358
8.27.3.33 getTranslation()	359
8.27.3.34 getTranslationMatrix()	359
8.27.3.35 getUserPointer()	360
8.27.3.36 getWheel() [1/3]	360
8.27.3.37 getWheel() [2/3]	360
8.27.3.38 getWheel() [3/3]	361
8.27.3.39 getWheelX()	361
8.27.3.40 getWheelY()	361
8.27.3.41 getWidth()	362
8.27.3.42 operator bool()	362
8.27.3.43 operator=()	362
8.27.3.44 reset()	363
8.27.3.45 resetRotation()	363
8.27.3.46 resetTranslation()	364
8.27.3.47 restoreViewport()	364
8.27.3.48 selectInterface()	364
8.27.3.49 setClose()	364
8.27.3.50 setKeyboardFunc()	366
8.27.3.51 setMouseFunc()	366
8.27.3.52 setResizeFunc()	366
8.27.3.53 setUserPointer()	367
8.27.3.54 setVelocity()	367
8.27.3.55 setWheelFunc()	367
8.27.3.56 shouldClose()	368
8.27.3.57 swapBuffers()	368

8.27.3.58 updateViewport()	368
9 ファイル詳解	369
9.1 Config.cpp ファイル	369
9.1.1 詳解	369
9.1.2 変数詳解	370
9.1.2.1 defaultLight	370
9.2 Config.cpp	370
9.3 Config.h ファイル	372
9.3.1 詳解	373
9.3.2 マクロ定義詳解	374
9.3.2.1 TCHARToUtf8	374
9.3.2.2 Utf8ToTCHAR	374
9.3.3 型定義詳解	374
9.3.3.1 pathChar	374
9.3.3.2 pathString	374
9.4 Config.h	375
9.5 Draw.cpp ファイル	376
9.5.1 詳解	376
9.6 Draw.cpp	376
9.7 Draw.h ファイル	377
9.7.1 詳解	378
9.8 Draw.h	378
9.9 gg.cpp ファイル	378
9.9.1 詳解	379
9.10 gg.cpp	379
9.11 gg.h ファイル	453
9.11.1 詳解	457
9.11.2 マクロ定義詳解	457
9.11.2.1 ggError	458
9.11.2.2 ggFBOError	458
9.12 gg.h	458
9.13 GgApp.cpp ファイル	512
9.13.1 詳解	512
9.14 GgApp.cpp	513
9.15 GgApp.h ファイル	526
9.15.1 詳解	527
9.15.2 マクロ定義詳解	527
9.15.2.1 GG_BUTTON_COUNT	527
9.15.2.2 GG_INTERFACE_COUNT	527
9.15.2.3 GG_USE_IMGUI	527
9.16 GgApp.h	528

9.17 ggsample01.cpp ファイル	535
9.17.1 マクロ定義詳解	536
9.17.1.1 CONFIG_FILE	536
9.17.1.2 PROJECT_NAME	536
9.18 ggsample01.cpp	536
9.19 main.cpp ファイル	537
9.19.1 マクロ定義詳解	537
9.19.1.1 HEADER_STR	538
9.19.2 関数詳解	538
9.19.2.1 main()	538
9.20 main.cpp	538
9.21 Menu.cpp ファイル	539
9.21.1 詳解	539
9.22 Menu.cpp	540
9.23 Menu.h ファイル	541
9.23.1 詳解	542
9.24 Menu.h	542
9.25 parseconfig.h ファイル	543
9.25.1 詳解	544
9.25.2 関数詳解	545
9.25.2.1 getString() [1/3]	545
9.25.2.2 getString() [2/3]	545
9.25.2.3 getString() [3/3]	546
9.25.2.4 getValue() [1/2]	546
9.25.2.5 getValue() [2/2]	547
9.25.2.6 getVector()	548
9.25.2.7 setString() [1/3]	548
9.25.2.8 setString() [2/3]	549
9.25.2.9 setString() [3/3]	549
9.25.2.10 setValue() [1/2]	550
9.25.2.11 setValue() [2/2]	550
9.25.2.12 setVector()	551
9.26 parseconfig.h	552
9.27 README.md ファイル	554
Index	555

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	109
gg::GgMatrix	299
gg::GgVector	182
gg::GgQuaternion	277
gg::GgTrackball	
Config	87
Draw	91
GgApp	92
gg::GgBuffer< T >	95
gg::GgColorTexture	100
gg::GgNormalTexture	165
gg::GgPointShader	173
gg::GgSimpleShader	258
gg::GgShader	248
gg::GgShape	251
gg::GgPoints	169
gg::GgTriangles	287
gg::GgElements	105
gg::GgSimpleObj	255
gg::GgTexture	272
gg::GgUniformBuffer< T >	291
gg::GgUniformBuffer< Light >	291
gg::GgSimpleShader::LightBuffer	319
gg::GgUniformBuffer< Material >	291
gg::GgSimpleShader::MaterialBuffer	330
gg::GgVertex	313
gg::GgSimpleShader::Light	317
gg::GgSimpleShader::Material	328
Menu	338
GgApp::Window	340

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	91
GgApp	92
gg::GgBuffer< T >	95
gg::GgColorTexture	100
gg::GgElements	105
gg::GgMatrix	109
gg::GgNormalTexture	165
gg::GgPoints	169
gg::GgPointShader	173
gg::GgQuaternion	182
gg::GgShader	248
gg::GgShape	251
gg::GgSimpleObj	255
gg::GgSimpleShader	258
gg::GgTexture	272
gg::GgTrackball	277
gg::GgTriangles	287
gg::GgUniformBuffer< T >	291
gg::GgVector	299
gg::GgVertex	313
gg::GgSimpleShader::Light	317
gg::GgSimpleShader::LightBuffer	319
gg::GgSimpleShader::Material	328
gg::GgSimpleShader::MaterialBuffer	330
Menu	338
GgApp::Window	340

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	369
Config.h	372
Draw.cpp	376
Draw.h	377
gg.cpp	378
gg.h	453
GgApp.cpp	512
GgApp.h	526
ggsample01.cpp	535
main.cpp	537
Menu.cpp	539
Menu.h	541
parseconfig.h	543

Chapter 7

名前空間詳解

7.1 gg 名前空間

クラス

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

列挙型

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

関数

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

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

- std::unique_ptr< [GgElements](#) > [ggElementsMesh](#) (GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3]=nullptr)
- std::unique_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 1338 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 2560 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 2604 行目に定義があります。

7.1.3.3 `ggArraysObj()`

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

Wavefront OBJ ファイルを読み込む (Arrays 形式)

引数

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

戻り値

`GgTriangles` 型のポインタ.

覚え書き

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

`gg.cpp` の 5208 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 4732 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 4981 行目に定義があります。

被呼び出し関係図:



7.1.3.6 ggCreateNormalMap()

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

グレースケール画像(8bit)から法線マップのデータを作成する。

引数

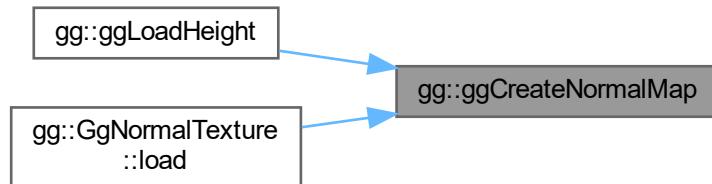
<i>hmap</i>	グレースケール画像のデータ。
<i>width</i>	高さマップのグレースケール画像 <i>hmap</i> の横の画素数。
<i>height</i>	高さマップのグレースケール画像 <i>hmap</i> の縦の画素数。
<i>format</i>	データの書式(GL_RED, GL_RG, GL_RGB, GL_RGBA)。
<small>構築</small> Doxygen	法線の z 成分の割合。
<i>internal</i>	法線マップを格納するテクスチャの内部フォーマット。
<i>nmap</i>	法線マップを格納する vector。

gg.cpp の 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:



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 4803 行目に定義があります。

呼び出し関係図:



7.1.3.8 ggCross() [1/2]

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

GgVector 型の 3 要素の外積.

引数

<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 1976 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 1418 行目に定義があります。

被呼び出し関係図:



7.1.3.10 ggDistance3() [1/2]

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

[GgVector](#) 型の 3 要素の距離.

引数

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

戻り値

a と b の距離.

[gg.h の 2013 行目](#)に定義があります。

呼び出し関係図:



7.1.3.11 ggDistance3() [2/2]

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

3 要素の距離.

引数

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

戻り値

a と b の距離.

[gg.h の 1455 行目](#)に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.12 ggDistance4() [1/2]

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

GgVector 型の 4 要素の距離.

引数

<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 2079 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 1518 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.14 ggDot3() [1/2]

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

GgVector 型の 3 要素の内積.

引数

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

戻り値

a と b のそれぞれの 3 要素の内積.

gg.h の 1990 行目に定義があります。

呼び出し関係図:



7.1.3.15 ggDot3() [2/2]

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

3 要素の内積.

引数

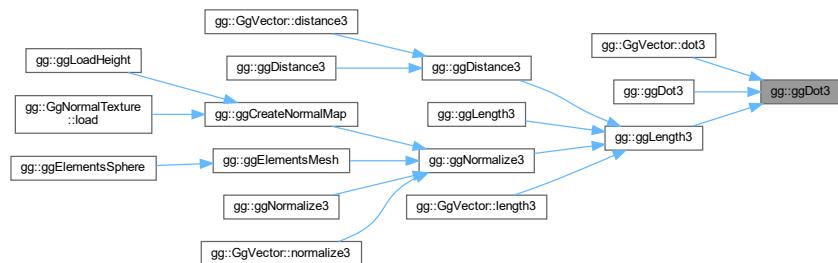
a	GLfloat 型の 3 要素の配列変数.
b	GLfloat 型の 3 要素の配列変数.

戻り値

a と b のそれぞれの 3 要素の内積.

gg.h の 1432 行目に定義があります。

被呼び出し関係図:



7.1.3.16 ggDot4() [1/2]

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

GgVector 型の 4 要素の内積.

引数

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

戻り値

a と *b* のそれぞれの 4 要素の内積.

gg.h の 2056 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

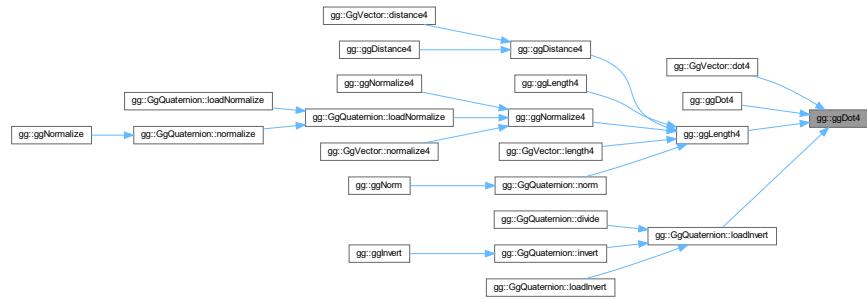
<i>a</i>	GLfloat 型の 4 要素の配列変数.
<i>b</i>	GLfloat 型の 4 要素の配列変数.

戻り値

`a` と `b` のそれぞれの 4 要素の内積.

`gg.h` の 1495 行目に定義があります。

被呼び出し関係図:



7.1.3.18 ggElementsMesh()

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

メッシュ形状を作成する (Elements 形式).

引数

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

戻り値

`GgElements` 型のポインタ.

覚え書き

メッシュ状に `GgElements` 形式の三角形データを生成する.

`gg.cpp` の 5242 行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::unique_ptr< gg::GgElements > gg::ggElementsObj (
    const std::string & name,
    bool normalize = false )
```

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

引数

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

戻り値

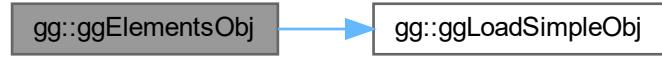
[GgElements](#) 型のポインタ.

覚え書き

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

gg.cpp の 5224 行目に定義があります。

呼び出し関係図:



7.1.3.20 ggElementsSphere()

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

球状に三角形データを生成する (Elements 形式).

引数

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

戻り値

[GgElements](#) 型のポインタ.

覚え書き

球状に [GgElements](#) 形式の三角形データを生成する.

[gg.cpp](#) の 5337 行目に定義があります。

呼び出し関係図:



7.1.3.21 ggEllipse()

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

橢円状に三角形を生成する。

引数

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

戻り値

`GgTriangles` 型のポインタ。

`gg.cpp` の 5183 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

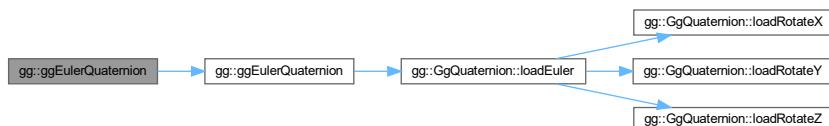
<i>e</i>	オイラー角を表す <code>GLfloat</code> 型の 3 要素の配列変数 (<code>heading</code> , <code>pitch</code> , <code>roll</code>)。
----------	---

戻り値

回転を表す四元数。

`gg.h` の 4646 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

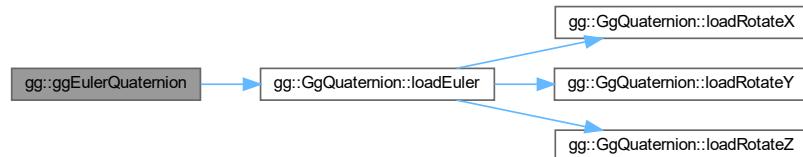
<i>heading</i>	y 軸中心の回転角.
<i>pitch</i>	x 軸中心の回転角.
<i>roll</i>	z 軸中心の回転角.

戻り値

回転を表す四元数.

`gg.h` の 4634 行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 3345 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 3070 行目に定義があります。

呼び出し関係図:



7.1.3.26 ggIdentityQuaternion()

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

単位四元数を返す.

戻り値

単位四元数.

gg.h の 4532 行目に定義があります。

呼び出し関係図:



7.1.3.27 `ggInit()`

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

ゲームグラフィックス特論の都合にもとづく初期化を行う。

覚え書き

WindowsにおいてOpenGL 1.2以降のAPIを有効化する。

`gg.cpp` の 1305 行目に定義があります。

呼び出し関係図:



7.1.3.28 `ggInvert()` [1/2]

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

逆行列を返す。

引数

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

戻り値

`m` の逆行列。

`gg.h` の 3390 行目に定義があります。

呼び出し関係図:



7.1.3.29 `ggInvert()` [2/2]

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

四元数の逆元を求める。

引数

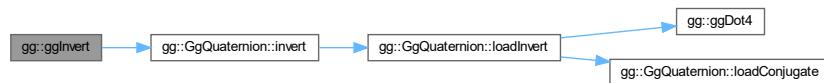
<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

四元数 `q` の逆元。

`gg.h` の 4743 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

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

戻り値

`a` の 3 要素の長さ。

`gg.h` の 2001 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

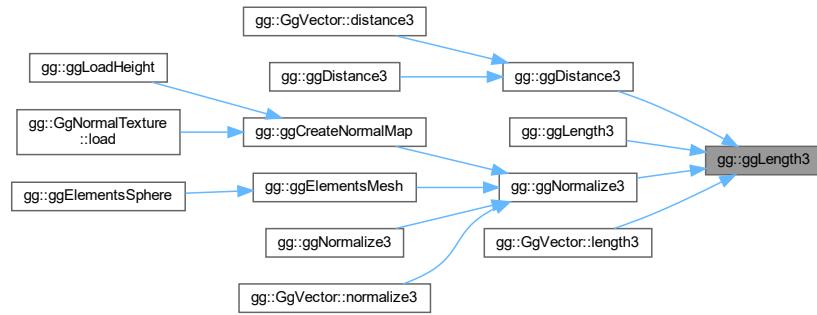
aの3要素の長さ。

gg.h の 1443 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.32 ggLength4() [1/2]

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

GgVector 型の 4 要素の長さ.

引数

a	GgVector 型の変数.
---	----------------

戻り値

a の 4 要素の長さ.

gg.h の 2067 行目に定義があります。

呼び出し関係図:



7.1.3.33 ggLength4() [2/2]

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

GLfloat 型の 4 要素の長さ.

引数

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

戻り値

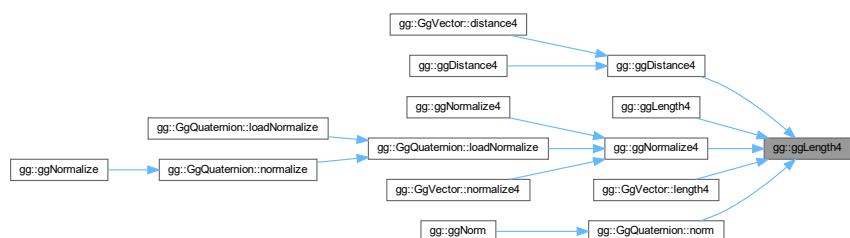
a の 4 要素の長さ.

gg.h の 1506 行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 5022 行目に定義があります。

呼び出し関係図:



7.1.3.35 ggLoadHeight()

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

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

引数

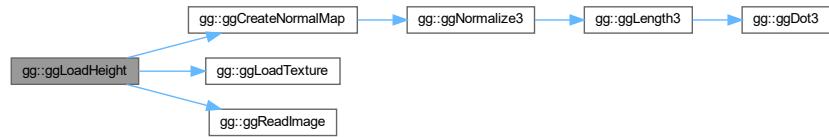
<i>name</i>	読み込むファイル名.
<i>nz</i>	法線の z 成分の割合.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット.

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば0.

[gg.cpp](#) の 3877 行目に定義があります。

呼び出し関係図:



7.1.3.36 ggLoadImage()

```

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

テクスチャを作成して TGA フォーマットの画像ファイルを読み込む.

引数

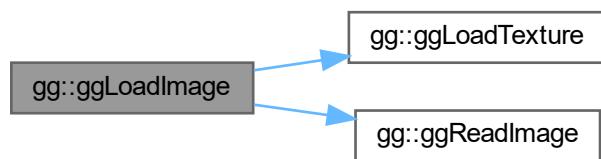
<i>name</i>	読み込むファイル名.
<i>pWidth</i>	読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない).
<i>pHeight</i>	読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない).
<i>internal</i>	glTexImage2D() に指定するテクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる.
<i>wrap</i>	テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE.

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

gg.cpp の 3743 行目に定義があります。

呼び出し関係図:



7.1.3.37 **ggLoadShader()** [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

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

戻り値

プログラムオブジェクトのプログラム名 (作成できなければ 0)。

gg.h の 5134 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>	構築ジオメトリシェーダのソースファイル名 (空文字列なら不使用)。
<i>nvarying</i>	フィードバックする varying 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする varying 変数のリスト (nullptr なら不使用)。

戻り値

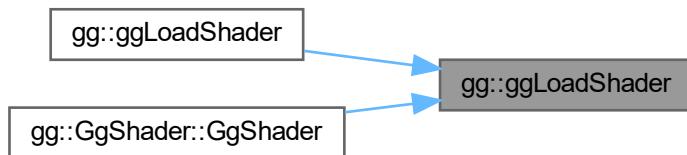
プログラムオブジェクトのプログラム名(作成できなければ0).

gg.cpp の 4948 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.39 ggLoadSimpleObj() [1/2]

```

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

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

引数

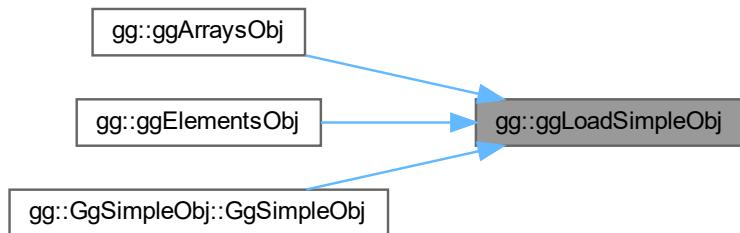
<i>name</i>	読み込む Wavefront OBJ ファイル名.
<i>group</i>	読み込んだデータのポリゴングループごとの最初の三角形の番号と三角形数・材質番号.
<i>material</i>	読み込んだデータのポリゴングループごとの GgSimpleShader::Material 型の材質.
<i>vert</i>	読み込んだデータの頂点属性.
<i>normalize</i>	true なら読み込んだデータの大きさを正規化する.

戻り値

ファイルの読み込みに成功したら true.

gg.cpp の 4555 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 4644 行目に定義があります。

7.1.3.41 ggLoadTexture()

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

テクスチャを作成して確保して画像データをテクスチャとして読み込む。

引数

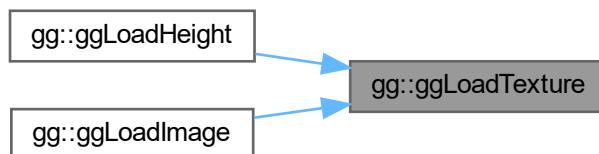
<i>image</i>	テクスチャとして読み込むデータ, <code>nullptr</code> ならテクスチャの作成のみを行う.
<i>width</i>	テクスチャとして読み込むデータ <i>image</i> の横の画素数.
<i>height</i>	テクスチャとして読み込むデータ <i>image</i> の縦の画素数.
<i>format</i>	<i>image</i> のフォーマット.
<i>type</i>	<i>image</i> のデータ型.
<i>internal</i>	テクスチャの内部フォーマット.
<i>wrap</i>	テクスチャのラッピングモード, デフォ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

戻り値

テクスチャの作成に成功すればテクスチャ名, 失敗すれば 0.

`gg.cpp` の 3695 行目に定義があります。

被呼び出し関係図:



7.1.3.42 ggLookat() [1/3]

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

ビュー変換行列を返す。

引数

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

戻り値

求めたビュー変換行列。

`gg.h` の 3305 行目に定義があります。

呼び出し関係図:



7.1.3.43 ggLookat() [2/3]

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

ビュー変換行列を返す。

引数

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

戻り値

求めたビュー変換行列.

gg.h の 3287 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

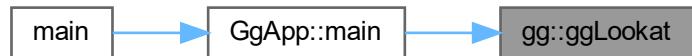
求めたビュー変換行列.

gg.h の 3269 行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

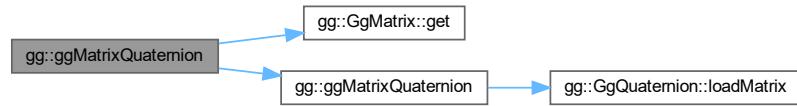
m	GgMatrix 型の変換行列。
---	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4556 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 4544 行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

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

戻り値

四元数 *q* のノルム。

gg.h の 4710 行目に定義があります。

呼び出し関係図:



7.1.3.48 ggNormal()

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

法線変換行列を返す。

引数

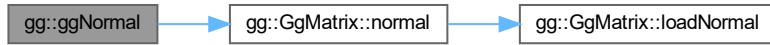
<i>m</i>	元の変換行列。
----------	---------

戻り値

m の法線変換行列。

gg.h の 3401 行目に定義があります。

呼び出し関係図:



7.1.3.49 ggNormalize()

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

正規化した四元数を返す。

引数

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

戻り値

四元数 `q` を正規化した四元数。

`gg.h` の 4721 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

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

戻り値

`a` の 3 要素を正規化したもの.

覚え書き

戻り値の `w` (第4) 要素は 0 になる.

`gg.h` の 2027 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]
```

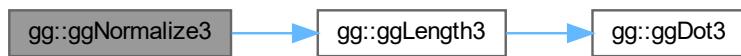
3 要素の正規化.

引数

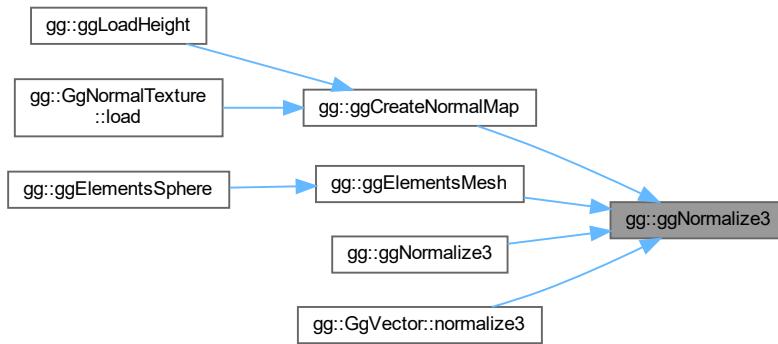
<code>a</code>	GLfloat 型の 3 要素の配列変数.
<code>b</code>	<code>a</code> の 3 要素を正規化した結果を格納する GLfloat 型の 3 要素の配列変数.

`gg.h` の 1467 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



7.1.3.52 `ggNormalize3()` [3/4]

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

`GgVector` 型の 3 要素の正規化.

引数

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

覚え書き

`a` の `w` (第4) 要素は 0 になる.

`gg.h` の 2043 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 1480 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 2090 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

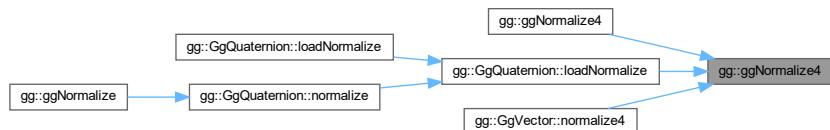
<i>a</i>	GLfloat 型の 4 要素の配列変数。
<i>b</i>	<i>a</i> を正規化した結果を格納する GLfloat 型の 4 要素の配列変数。

gg.h の 1530 行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 2102 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 1544 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 3326 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 3364 行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::unique_ptr< gg::GgPoints > gg::ggPointsCube (
    GLsizei countv,
    GLfloat length = 1.0f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f )
  
```

点群を立方体状に生成する.

引数

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

戻り値

`GgPoints` 型のポインタ.

`gg.cpp` の 5109 行目に定義があります。

7.1.3.61 `ggPointsSphere()`

```
std::unique_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` 型のポインタ.

`gg.cpp` の 5135 行目に定義があります。

7.1.3.62 `ggQuaternion()` [1/2]

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

四元数を返す.

引数

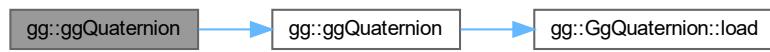
<code>a</code>	GLfloat 型の GLfloat 型の 4 要素の配列変数に格納した四元数.
----------------	--

戻り値

四元数.

gg.h の 4522 行目に定義があります。

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 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

四元数.

gg.h の 4510 行目に定義があります。

呼び出し関係図:



呼び出し関係図:



7.1.3.64 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 4567 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

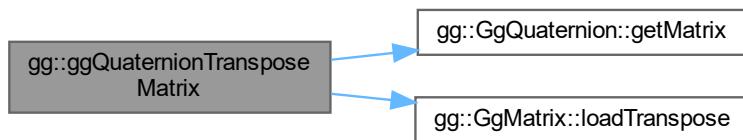
<i>q</i>	元の四元数.
----------	--------

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 4580 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

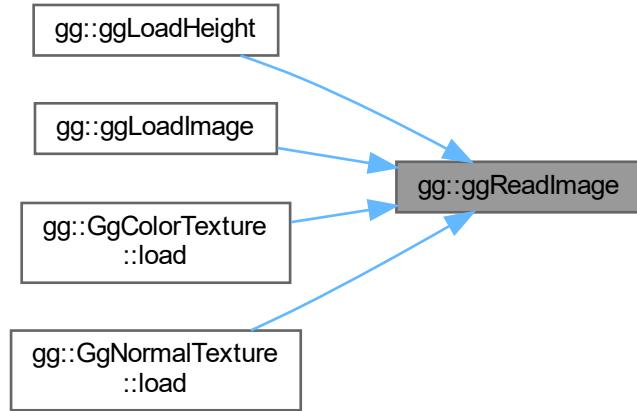
<i>name</i>	読み込むファイル名.
<i>image</i>	読み込んだデータを格納する <code>vector</code> .
<i>pWidth</i>	読み込んだ画像の横の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない.
<i>pHeight</i>	読み込んだ画像の縦の画素数の格納先のポインタ, <code>nullptr</code> なら格納しない.
<i>pFormat</i>	読み込んだファイルの書式 (<code>GL_RED</code> , <code>GL_RG</code> , <code>GL_RGB</code>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

戻り値

読み込みに成功すれば `true`, 失敗すれば `false`.

gg.cpp の 3574 行目に定義がります。

被呼び出し関係図:



7.1.3.67 ggRectangle()

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

矩形状に 2 枚の三角形を生成する。

引数

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

戻り値

GgTriangles 型のポインタ。

gg.cpp の 5162 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 3249 行目に定義がります。

呼び出し関係図:



7.1.3.69 `ggRotate()` [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

<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 3225 行目に定義がります。

呼び出し関係図:



7.1.3.70 ggRotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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 3237 行目に定義があります。

呼び出し関係図:



7.1.3.71 ggRotate() [4/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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

戻り値

r を軸に *a* だけ回転する変換行列.

`gg.h` の 3212 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 3199 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 4621 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 4610 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

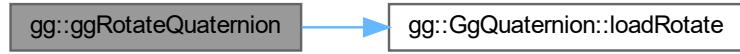
<i>x</i>	軸ベクトルの <i>x</i> 成分.
<i>y</i>	軸ベクトルの <i>y</i> 成分.
<i>z</i>	軸ベクトルの <i>z</i> 成分.
<i>a</i>	回転角.

戻り値

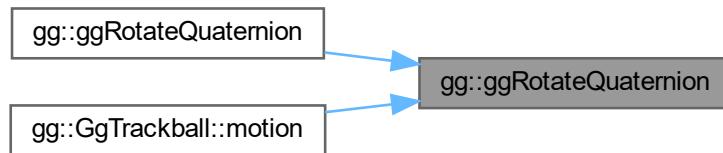
回転を表す四元数.

gg.h の 4597 行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 3160 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 3172 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

<i>a</i>	回転角.
----------	------

戻り値

z 軸中心に *a* だけ回転する変換行列.

[gg.h](#) の 3184 行目に定義がります。

呼び出し関係図:



7.1.3.79 ggSaveColor()

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

カラーバッファの内容を TGA ファイルに保存する.

引数

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

戻り値

保存に成功すれば true, 失敗すれば false.

[gg.cpp](#) の 3518 行目に定義がります。

呼び出し関係図:



7.1.3.80 ggSaveDepth()

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

デプスバッファの内容を TGA ファイルに保存する.

引数

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

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 3544 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

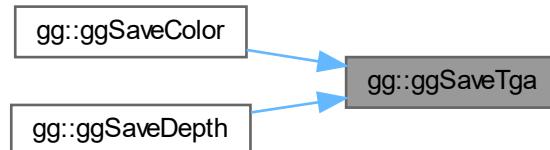
<i>name</i>	保存するファイル名.
<i>buffer</i>	画像データを格納した配列.
<i>width</i>	画像の横の画素数.
<i>height</i>	画像の縦の画素数.
<i>depth</i>	1画素のバイト数.

戻り値

保存に成功すれば `true`, 失敗すれば `false`.

`gg.cpp` の 3428 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 3148 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

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

戻り値

拡大縮小の変換行列.

[gg.h](#) の 3136 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

x	x 方向の拡大率.
y	y 方向の拡大率.
z	z 方向の拡大率.
w	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 3124 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 4673 行目に定義があります。

呼び出し関係図:



7.1.3.86 ggSlerp() [2/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

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

戻り値

q, a を *t* で内分した四元数。

gg.h の 4686 行目に定義があります。

呼び出し関係図:



7.1.3.87 ggSlerp() [3/4]

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

二つの四元数の球面線形補間の結果を返す。

引数

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

戻り値

a, q を t で内分した四元数.

gg.h の 4699 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

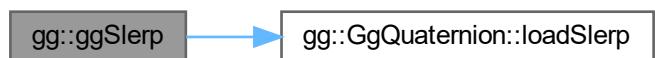
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
b	四元数を格納した GLfloat 型の 4 要素の配列変数.
t	補間パラメータ.

戻り値

a, b を t で内分した四元数.

gg.h の 4659 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 3109 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 <code>GLfloat</code> 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 3097 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 3085 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

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

戻り値

`m` の転置行列.

`gg.h` の 3379 行目に定義があります。

呼び出し関係図:



7.1.3.93 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 1949 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 1902 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 1914 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

<code>v</code>	<code>GgVector</code> 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 1925 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

<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` から `b` の各要素を引いた差.

`gg.h` の 1937 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 1961 行目に定義があります。

7.1.4 変数詳解

7.1.4.1 ggBufferAlignment

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

使用している GPU のバッファアライメント.

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

Chapter 8

クラス詳解

8.1 Config クラス

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

公開メンバ関数

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

フレンド

- `class Menu`

8.1.1 詳解

構成データ

`Config.h` の 40 行目に定義がります。

8.1.2 構築子と解体子

8.1.2.1 Config() [1/2]

Config::Config ()

コンストラクタ

[Config.cpp](#) の 25 行目に定義があります。

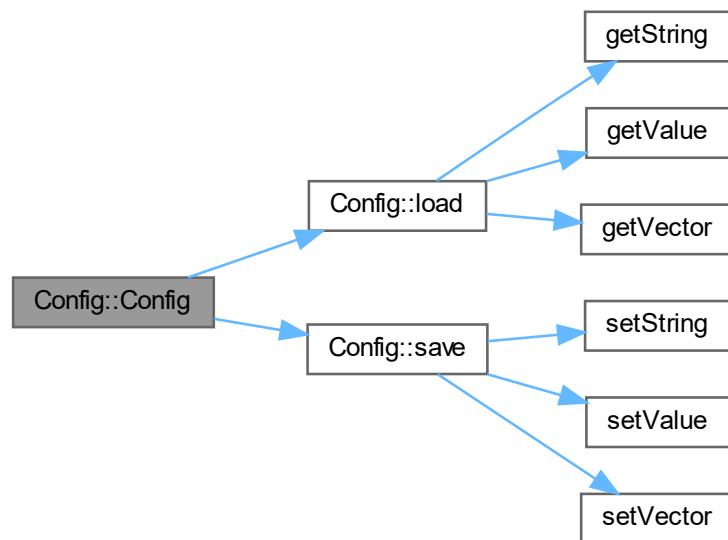
8.1.2.2 Config() [2/2]

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

ファイルから構成データを読み込むコンストラクタ

[Config.cpp](#) の 42 行目に定義があります。

呼び出し関係図:



8.1.2.3 ~Config()

Config::~Config () [virtual]

デストラクタ

[Config.cpp](#) の 52 行目に定義があります。

8.1.3 関数詳解

8.1.3.1 getHeight()

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

ウィンドウの横幅を得る

[Config.h](#) の 91 行目に定義があります。

8.1.3.2 getWidth()

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

ウィンドウの横幅を得る

[Config.h](#) の 83 行目に定義があります。

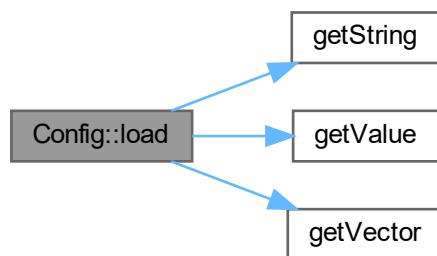
8.1.3.3 load()

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

設定ファイルを読み込む

[Config.cpp](#) の 59 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



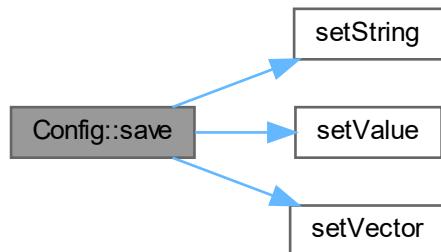
8.1.3.4 save()

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

設定ファイルを書き出す

Config.cpp の 109 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.1.4 フレンドと関連関数の詳解

8.1.4.1 Menu

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

Config.h の 43 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:

- 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 行目に定義があります。

8.2.3 関数詳解

8.2.3.1 draw()

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

描画する

Draw.cpp の 36 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.3 GgApp クラス

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

クラス

- class [Window](#)

公開メンバ関数

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

8.3.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラス

`GgApp.h` の 64 行目に定義がります。

8.3.2 構築子と解体子

8.3.2.1 GgApp() [1/2]

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

コンストラクタ.

引数

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

`GgApp.cpp` の 24 行目に定義がります。

8.3.2.2 GgApp() [2/2]

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

8.3.2.3 ~GgApp()

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

デストラクタ.

`GgApp.cpp` の 71 行目に定義がります。

8.3.3 関数詳解

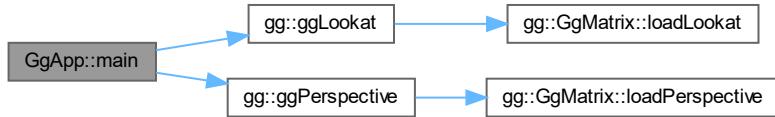
8.3.3.1 main()

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

アプリケーション本体。

[ggsample01.cpp](#) の 28 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.3.3.2 operator=()

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

このクラス詳解は次のファイルから抽出されました:

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

8.4 gg::GgBuffer< T > クラステンプレート

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

公開メンバ関数

- `GgBuffer` (GLenum target, const T *data, GLsizei stride, GLsizei count, GLenum usage)
- `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 5529 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

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

gg.h の 5541 行目に定義があります。

8.4.2.2 ~GgBuffer()

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

gg.h の 5541 行目に定義があります。

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 5634 行目に定義があります。

8.4.3.2 copy()

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

別のバッファオブジェクトからデータを複写する。

引数

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

gg.h の 5739 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 5626 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 5616 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 5606 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 5596 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 5652 行目に定義があります。

8.4.3.8 `map()` [2/2]

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

バッファオブジェクトの指定した範囲をマップする。

引数

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

戻り値

マップしたメモリの先頭のポインタ。

`gg.h` の 5669 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 5712 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 5694 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 5642 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 5682 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

8.5 `gg::GgColorTexture` クラス

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

公開メンバ関数

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

8.5.1 詳解

カラーマップ。

覚え書き

カラー画像を読み込んでテクスチャを作成する。

[gg.h](#) の 5308 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 5318 行目に定義があります。

8.5.2.2 GgColorTexture() [2/3]

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

メモリ上のデータからカラーのテクスチャを作成するコンストラクタ.

引数

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

gg.h の 5334 行目に定義があります。

呼び出し関係図:



8.5.2.3 GgColorTexture() [3/3]

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

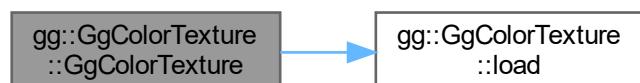
TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成するコンストラクタ.

引数

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

gg.h の 5355 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 5367 行目に定義があります。

8.5.3 関数詳解

8.5.3.1 load() [1/2]

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

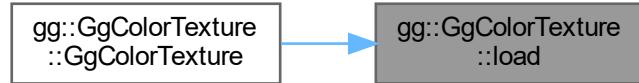
メモリ上のデータを読み込んでテクスチャを作成する.

引数

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

gg.h の 5383 行目に定義があります。

被呼び出し関係図:



8.5.3.2 load() [2/2]

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

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

引数

<i>name</i>	読み込むファイル名。
<i>internal</i>	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 3939 行目に定義があります。

呼び出し関係図:



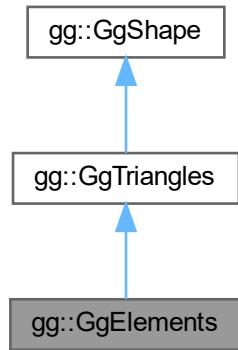
このクラス詳解は次のファイルから抽出されました:

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

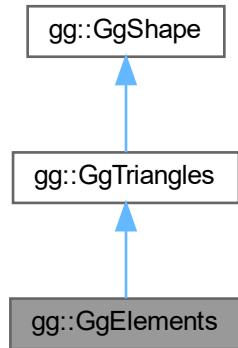
8.6 gg::GgElements クラス

```
#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 6437 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

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

`gg.h` の 6450 行目に定義があります。

8.6.2.2 `GgElements()` [2/2]

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

コンストラクタ.

引数

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

引数

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

gg.h の 6465 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 6481 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 5096 行目に定義がります。

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 6499 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 6489 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 6536 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 6514 行目に定義があります。

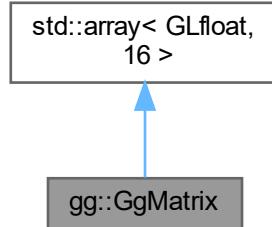
このクラス詳解は次のファイルから抽出されました:

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

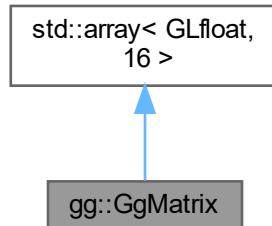
8.7 gg::GgMatrix クラス

```
#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 2110 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 2123 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 2147 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 2162 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

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

gg.h の 2172 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

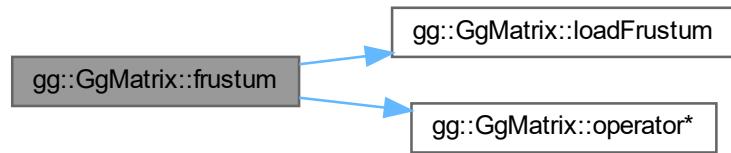
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

透視投影変換行列を乗じた変換行列.

gg.h の 2926 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]
```

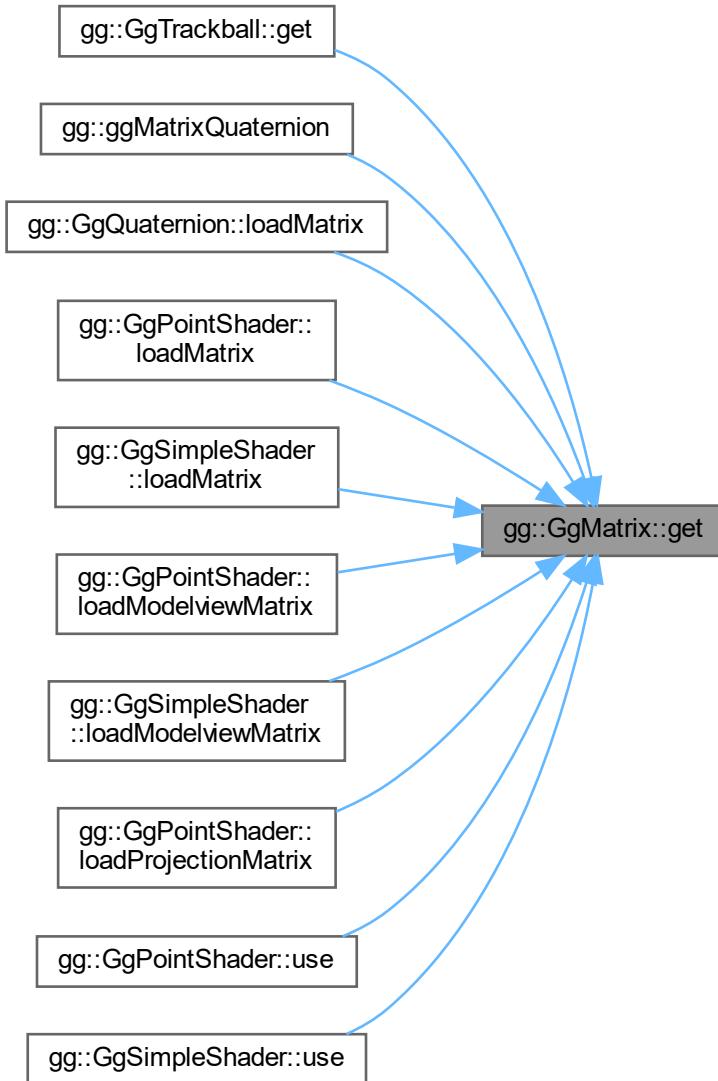
変換行列を取り出す.

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 3049 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 3059 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 2970 行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

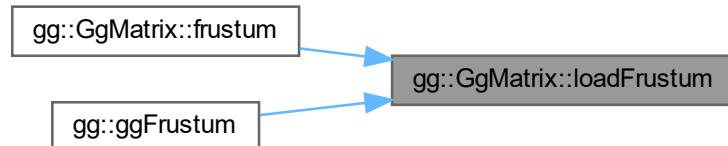
<i>left</i>	ウィンドウの左端の位置.
<i>right</i>	ウィンドウの右端の位置.
<i>bottom</i>	ウィンドウの下端の位置.
<i>top</i>	ウィンドウの上端の位置.
<i>zNear</i>	視点から前方面までの位置.
<i>zFar</i>	視点から後方面までの位置.

戻り値

設定した透視投影変換行列.

`gg.cpp` の 3032 行目に定義がります。

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 2685 行目に定義がります。

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

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

戻り値

設定した *m* の逆行列。

gg.h の 2649 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

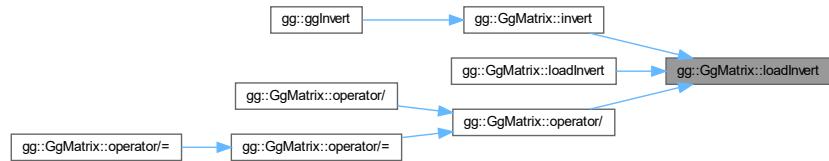
<i>a</i>	GLfloat 型の 16 要素の変換行列。
----------	------------------------

戻り値

設定した *a* の逆行列。

gg.cpp の 2842 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 2569 行目に定義があります。

呼び出し関係図:



8.7.3.10 loadLookat() [2/3]

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

ビュー変換行列を格納する。

引数

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

戻り値

設定したビュー変換行列。

gg.h の 2556 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

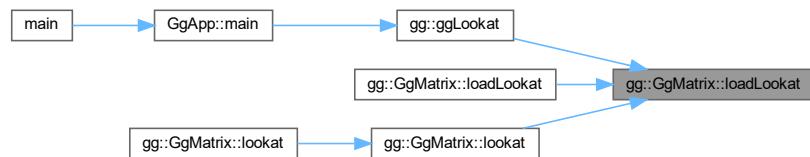
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

設定したビュー変換行列.

`gg.cpp` の 2944 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 2668 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

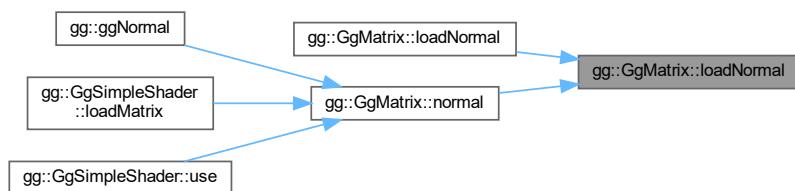
<code>a</code>	GLfloat 型の 16 要素の変換行列。
----------------	------------------------

戻り値

設定した `m` の法線変換行列。

gg.cpp の 2924 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

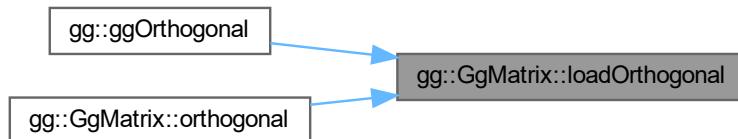
<i>left</i>	ウィンドウの左端の位置。
<i>right</i>	ウィンドウの右端の位置。
<i>bottom</i>	ウィンドウの下端の位置。
<i>top</i>	ウィンドウの上端の位置。
<i>zNear</i>	視点から前方面までの位置。
<i>zFar</i>	視点から後方面までの位置。

戻り値

設定した直交投影変換行列。

gg.cpp の 3002 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 3062 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 2525 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 GgVector 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 2503 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 <code>GLfloat</code> 型の 4 要素の配列変数, 第 4 要素を回転角に用いる.
----------	--

戻り値

設定した変換行列.

`gg.h` の 2514 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 <code>GLfloat</code> 型の 3 要素の配列変数 (<i>x</i> , <i>y</i> , <i>z</i>).
<i>a</i>	回転角.

戻り値

設定した変換行列.

`gg.h` の 2491 行目に定義があります。

呼び出し関係図:



8.7.3.20 loadRotate() [5/5]

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

(x, y, z) 方向のベクトルを軸とする回転の変換行列を格納する。

引数

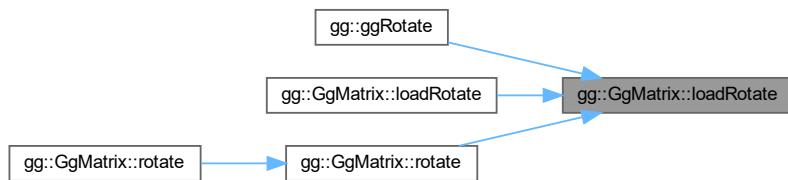
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

設定した変換行列.

gg.cpp の 2778 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

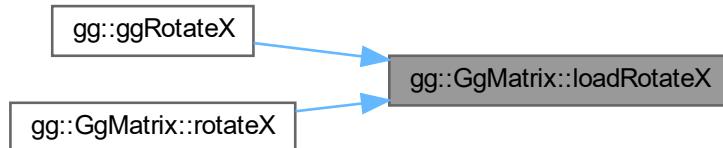
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2730 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

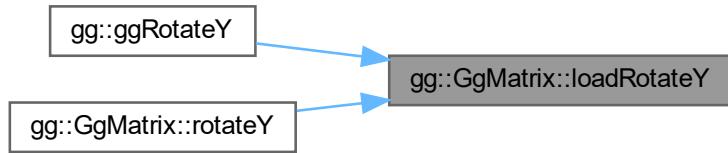
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2746 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

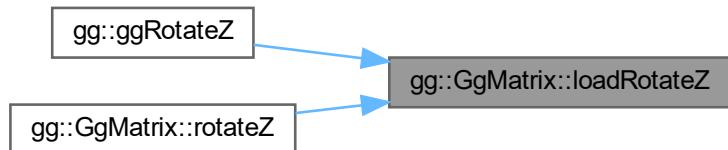
a	回転角.
---	------

戻り値

設定した変換行列。

gg.cpp の 2762 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

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

戻り値

設定した変換行列。

`gg.h` の 2444 行目に定義があります。

呼び出し関係図:

8.7.3.25 `loadScale()` [2/3]

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

拡大縮小の変換行列を格納する。

引数

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

戻り値

設定した変換行列。

`gg.h` の 2433 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

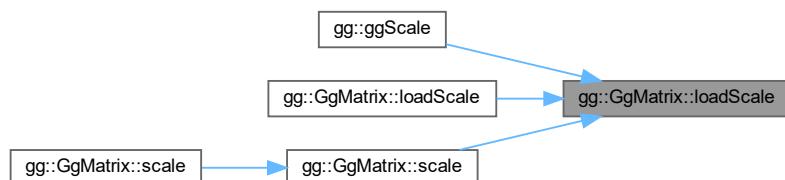
<i>x</i>	<i>x</i> 方向の拡大率。
<i>y</i>	<i>y</i> 方向の拡大率。
<i>z</i>	<i>z</i> 方向の拡大率。
<i>w</i>	<i>w</i> 拡大率のスケールファクタ (= 1.0f)。

戻り値

設定した変換行列。

gg.cpp の 2714 行目に定義があります。

被呼び出し関係図:



8.7.3.27 **loadTranslate()** [1/3]

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

平行移動の変換行列を格納する。

引数

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

戻り値

設定した変換行列。

`gg.h` の 2411 行目に定義がります。

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

戻り値

設定した変換行列。

`gg.h` の 2400 行目に定義がります。

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

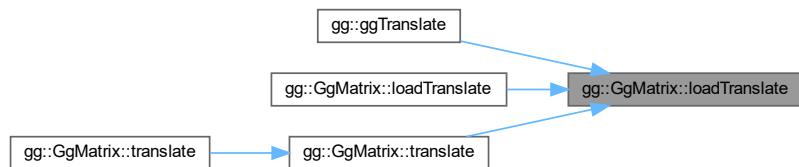
<i>x</i>	<i>x</i> 方向の移動量。
<i>y</i>	<i>y</i> 方向の移動量。
<i>z</i>	<i>z</i> 方向の移動量。
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f)。

戻り値

設定した変換行列。

gg.cpp の 2698 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]

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

転置行列を格納する。

引数

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

戻り値

設定した *m* の転置行列。

gg.h の 2630 行目に定義があります。

呼び出し関係図:



8.7.3.31 loadTranspose() [2/2]

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

転置行列を格納する。

引数

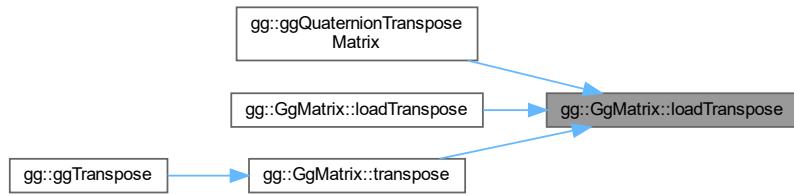
<i>a</i>	GLfloat 型の 16 要素の変換行列。
----------	------------------------

戻り値

設定した *a* の転置行列。

gg.cpp の 2817 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 2889 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 2876 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

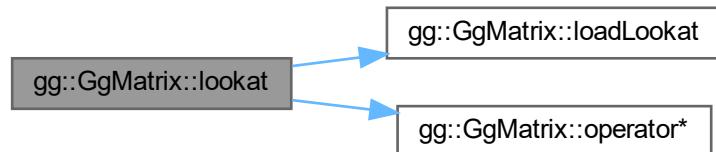
<code>ex</code>	視点の位置の x 座標値.
<code>ey</code>	視点の位置の y 座標値.
<code>ez</code>	視点の位置の z 座標値.
<code>tx</code>	目標点の位置の x 座標値.
<code>ty</code>	目標点の位置の y 座標値.
<code>tz</code>	目標点の位置の z 座標値.
<code>ux</code>	上方向のベクトルの x 成分.
<code>uy</code>	上方向のベクトルの y 成分.
<code>uz</code>	上方向のベクトルの z 成分.

戻り値

ビュー変換行列を乗じた変換行列.

`gg.h` の 2858 行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()

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

法線変換行列を返す。

戻り値

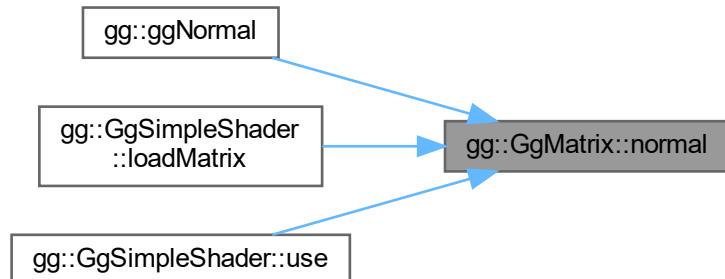
法線変換行列。

gg.h の 2981 行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]

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

変換行列に別の変換行列を乗算した値を返す。

引数

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

戻り値

変換行列に `a` を乗じた `GgMatrix` 型の変換行列.

`gg.h` の 2302 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

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

戻り値

`v` 変換結果の `GgVector` 型のベクトル.

`gg.h` の 3037 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

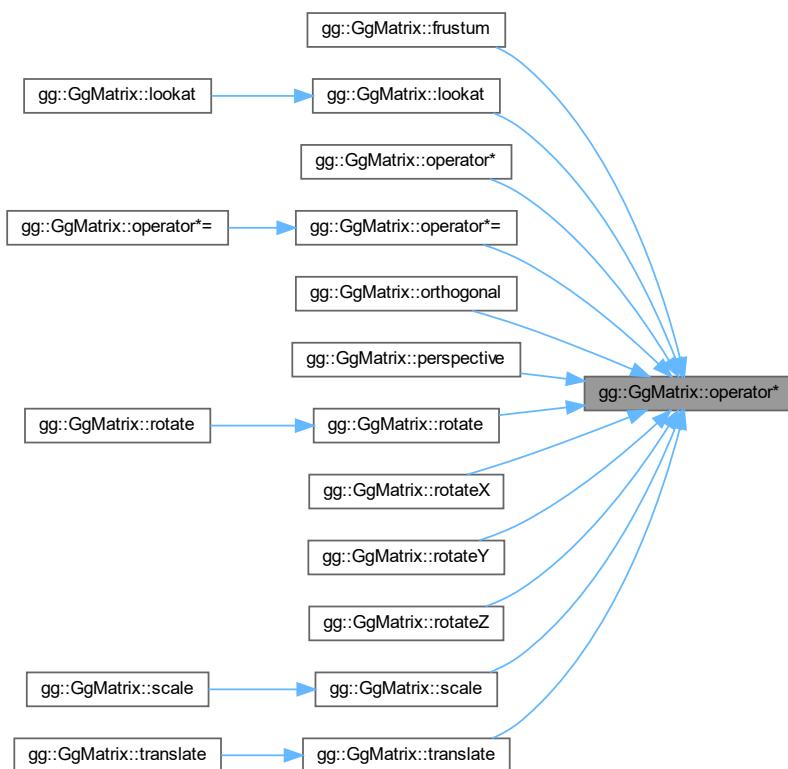
a	変換行列を格納した GLfloat 型の 16 要素の配列変数.
---	----------------------------------

戻り値

変換行列に a を乗じた GgMatrix 型の変換行列.

gg.h の 2289 行目に定義があります。

被呼び出し関係図:



8.7.3.39 operator*=() [1/2]

```

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

```

変換行列に別の変換行列を乗算した結果を格納する.

引数

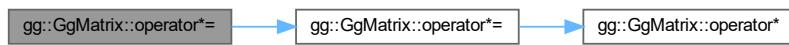
<code>m</code>	<code>GgMatrix</code> 型の変換行列.
----------------	-------------------------------

戻り値

`m` を掛けた変換行列の参照.

`gg.h` の 2325 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 <code>GLfloat</code> 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 2313 行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

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

戻り値

変換行列に *a* を加えた GgMatrix 型の変換行列.

gg.h の 2208 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 <code>GLfloat</code> 型の 16 要素の配列変数.
---	---

戻り値

変換行列に a を加えた `GgMatrix` 型の変換行列.

`gg.h` の 2195 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	<code>GgMatrix</code> 型の変換行列.
---	-------------------------------

戻り値

m を加えた変換行列の参照.

`gg.h` の 2231 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 2219 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 2255 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 2242 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 2278 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 2266 行目に定義があります。

被呼び出し関係図:



8.7.3.49 operator/() [1/2]

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

変換行列を変換行列で除算した値を返す.

引数

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

戻り値

変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2350 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

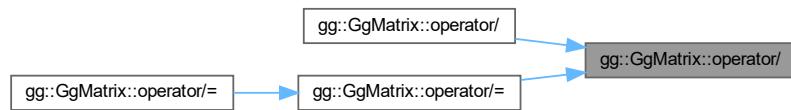
変換行列を *a* で割った GgMatrix 型の変換行列.

gg.h の 2336 行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>	GgMatrix 型の変換行列。
----------	------------------

戻り値

m で割った変換行列の参照。

gg.h の 2373 行目に定義があります。

呼び出し関係図:



8.7.3.52 `operator/()` [2/2]

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

変換行列を配列に格納した変換行列で除算した結果を格納する。

引数

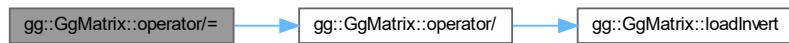
<code>a</code>	変換行列を格納した GLfloat 型の 16 要素の配列変数。
----------------	----------------------------------

戻り値

`a` で割った変換行列の参照。

`gg.h` の 2361 行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

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

戻り値

`a` を代入したこのオブジェクトの参照.

`gg.h` の 2183 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

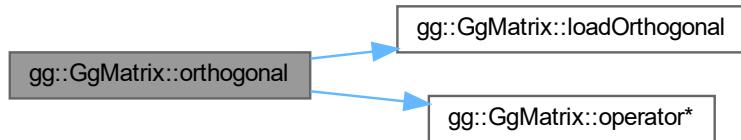
<code>left</code>	ウィンドウの左端の位置.
<code>right</code>	ウィンドウの右端の位置.
<code>bottom</code>	ウィンドウの下端の位置.
<code>top</code>	ウィンドウの上端の位置.
<code>zNear</code>	視点から前方面までの位置.
<code>zFar</code>	視点から後方面までの位置.

戻り値

直交投影変換行列を乗じた変換行列.

`gg.h` の 2905 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

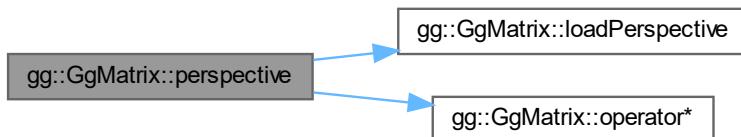
<i>fovy</i>	<i>y</i> 方向の画角。
<i>aspect</i>	縦横比。
<i>zNear</i>	視点から前方面までの位置。
<i>zFar</i>	視点から後方面までの位置。

戻り値

透視投影変換行列を乗じた変換行列。

gg.h の 2945 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 <code>GgVector</code> 型の変数。
<i>v</i>	元のベクトルの <code>GgVector</code> 型の変数。

`gg.h` の 3026 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 <code>GgVector</code> 型の変数。
<i>v</i>	元のベクトルの <code>GLfloat</code> 型の 4 要素の配列変数。

`gg.h` の 3015 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 <code>GLfloat</code> 型の 4 要素の配列変数。
<i>v</i>	元のベクトルの <code>GgVector</code> 型の変数。

`gg.h` の 3004 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 2993 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 2839 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

<code>r</code>	回転軸の方向ベクトルを格納した <code>GgVector</code> 型の変数.
<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 2817 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

<code>r</code>	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 (<code>x, y, z, a</code>).
----------------	--

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 2828 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

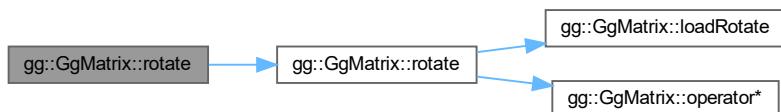
<code>r</code>	回転軸の方向ベクトルを格納した <code>GLfloat</code> 型の 3 要素の配列変数 (<code>x, y, z</code>).
<code>a</code>	回転角.

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 2805 行目に定義がります。

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

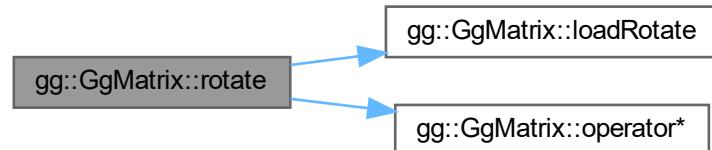
x	回転軸の x 成分.
y	回転軸の y 成分.
z	回転軸の z 成分.
a	回転角.

戻り値

(x, y, z) を軸にさらに a 回転した変換行列.

`gg.h` の 2792 行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

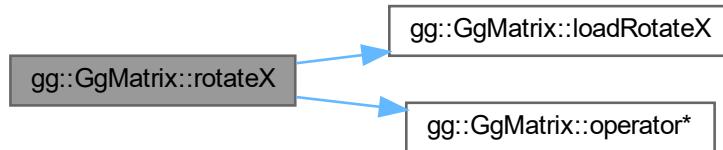
<i>a</i>	回転角.
----------	------

戻り値

x 軸中心にさらに *a* 回転した変換行列。

[gg.h](#) の 2753 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

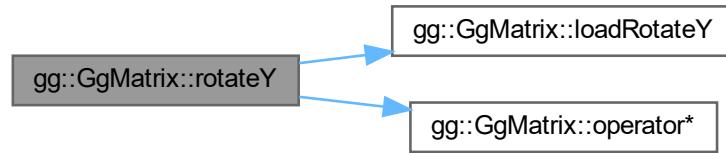
<i>a</i>	回転角.
----------	------

戻り値

y 軸中心にさらに *a* 回転した変換行列。

[gg.h](#) の 2765 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

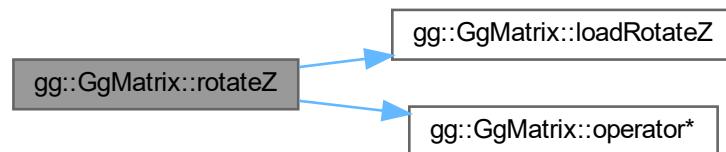
<code>a</code>	回転角.
----------------	------

戻り値

`z` 軸中心にさらに `a` 回転した変換行列。

`gg.h` の 2777 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

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

戻り値

拡大縮小した結果の変換行列。

`gg.h` の 2742 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

<code>s</code>	拡大率の <code>GLfloat</code> 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 2731 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

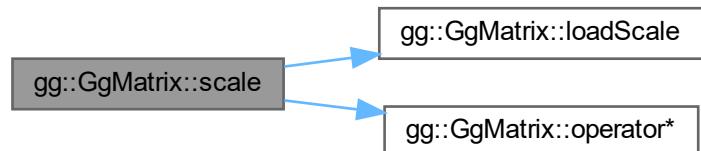
<i>x</i>	<i>x</i> 方向の拡大率.
<i>y</i>	<i>y</i> 方向の拡大率.
<i>z</i>	<i>z</i> 方向の拡大率.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

拡大縮小した結果の変換行列.

gg.h の 2719 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

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

戻り値

平行移動した結果の変換行列.

`gg.h` の 2705 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

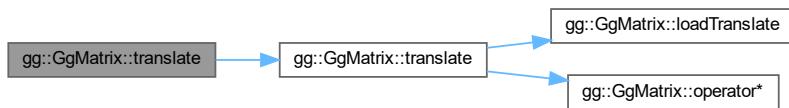
<i>t</i>	移動量の GLfloat 型の 3 要素の配列変数 (x, y, z).
----------	--------------------------------------

戻り値

平行移動した結果の変換行列.

gg.h の 2694 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

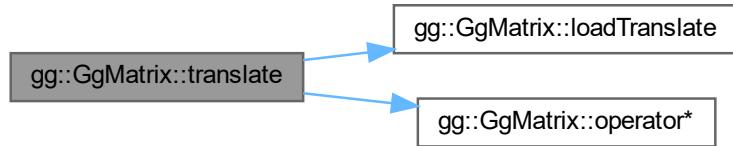
<i>x</i>	<i>x</i> 方向の移動量.
<i>y</i>	<i>y</i> 方向の移動量.
<i>z</i>	<i>z</i> 方向の移動量.
<i>w</i>	<i>w</i> 移動量のスケールファクタ (= 1.0f).

戻り値

平行移動した結果の変換行列.

gg.h の 2682 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]`

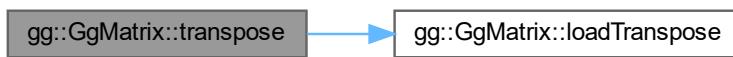
転置行列を返す.

戻り値

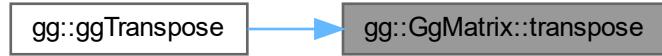
転置行列.

`gg.h` の 2959 行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 5418 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 5428 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]
```

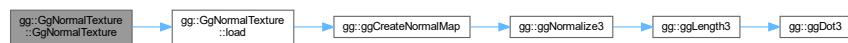
メモリ上のデータから法線マップのテクスチャを作成するコンストラクタ.

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 <i>z</i> 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5442 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 <i>z</i> 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5462 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 5475 行目に定義があります。

8.8.3 関数詳解

8.8.3.1 load() [1/2]

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

メモリ上のデータから法線マップのテクスチャを作成する.

引数

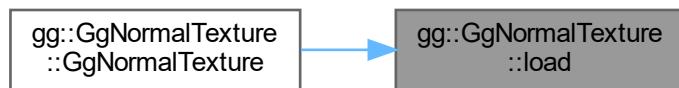
<i>hmap</i>	テクスチャとして用いる画像データ, <code>nullptr</code> ならデータを読み込まない.
<i>width</i>	テクスチャとして用いる画像データの横幅.
<i>height</i>	テクスチャとして用いる画像データの高さ.
<i>format</i>	テクスチャとして用いる画像データのフォーマット (GL_RED, GL_RG, GL_RGB, GL_RGBA).
<i>nz</i>	法線マップの <i>z</i> 成分の値.
<i>internal</i>	テクスチャの内部フォーマット.

gg.h の 5489 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 )
  
```

TGA フォーマットの画像ファイルから高さマップ読み込んで法線マップのテクスチャを作成する。

引数

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

gg.cpp の 3975 行目に定義がります。

呼び出し関係図:



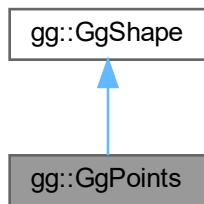
このクラス詳解は次のファイルから抽出されました:

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

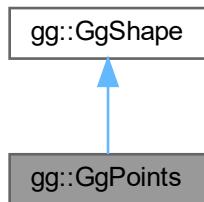
8.9 gg::GgPoints クラス

```
#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 \(const GgVector *pos, GLsizei count, GLenum mode=GL_POINTS, GLenum usage=GL_STATIC_DRAW\)](#)
- 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 6184 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 6195 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 6208 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 6222 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 5053 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 6241 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 6231 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 5040 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 6253 行目に定義があります。

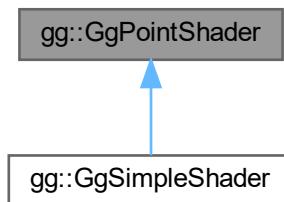
このクラス詳解は次のファイルから抽出されました:

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

8.10 gg::GgPointShader クラス

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

gg::GgPointShader の継承関係図



公開メンバ関数

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

8.10.1 詳解

点のシェーダ.

gg.h の 6788 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 6804 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 6819 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 6838 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 6850 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 7034 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 <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).

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 6897 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

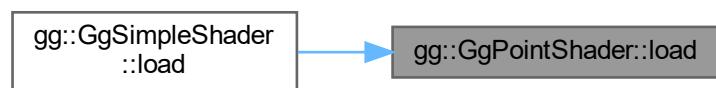
<i>vert</i>	バーテックスシェーダのソースファイル名.
<i>frag</i>	フラグメントシェーダのソースファイル名(空文字列なら不使用).
<i>geom</i>	ジオメトリシェーダのソースファイル名(空文字列なら不使用).
<i>nvarying</i>	フィードバックする <i>varying</i> 変数の数 (0 なら不使用).
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト.

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 6864 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 6964 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 6952 行目に定義があります。

8.10.3.6 loadModelviewMatrix() [1/2]

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

モデルビュー変換行列を設定する。

引数

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

gg::GgSimpleShader で再実装されています。

gg.h の 6941 行目に定義があります。

呼び出し関係図:



8.10.3.7 loadModelviewMatrix() [2/2]

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

モデルビュー変換行列を設定する。

引数

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

gg::GgSimpleShader で再実装されています。

gg.h の 6931 行目に定義があります。

8.10.3.8 loadProjectionMatrix() [1/2]

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

投影変換行列を設定する。

引数

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

gg.h の 6921 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 6911 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 7024 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 6972 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

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

[gg.h](#) の 6993 行目に定義があります。

呼び出し関係図:



8.10.3.13 use() [3/5]

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

投影変換行列とモデルビューを設定してシェーダプログラムの使用を開始する。

引数

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

gg.h の 7016 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 6982 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 7004 行目に定義があります。

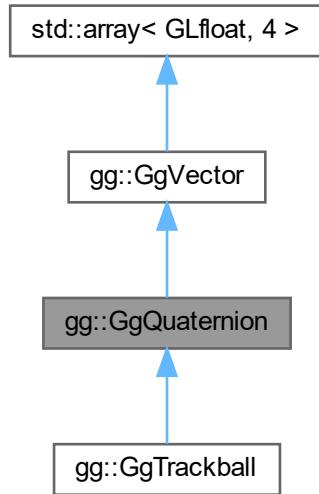
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

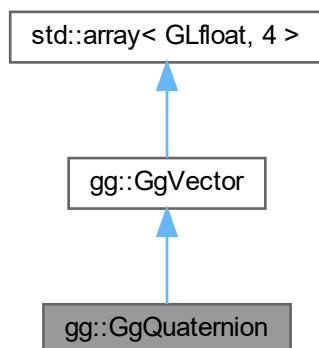
8.11 gg::GgQuaternion クラス

```
#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 3409 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 3428 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

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

gg.h の 3440 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 3450 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 3460 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 3470 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 3779 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 3768 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

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

戻り値

a を加えた四元数.

[gg.h](#) の 3757 行目に定義があります。

呼び出し関係図:



8.11.3.4 add() [4/4]

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

四元数に別の四元数を加算した結果を返す.

引数

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

戻り値

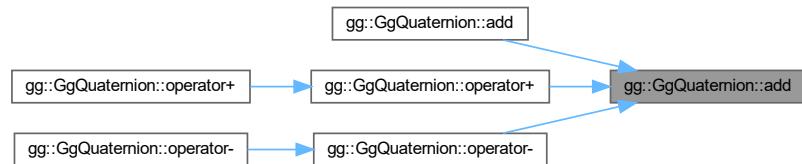
(*x*, *y*, *z*, *w*) を加えた四元数.

gg.h の 3745 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.5 conjugate()

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

共役四元数に変換する。

戻り値

共役四元数。

gg.h の 4402 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.6 divide() [1/4]

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

四元数を別の四元数で除算した結果を返す。

引数

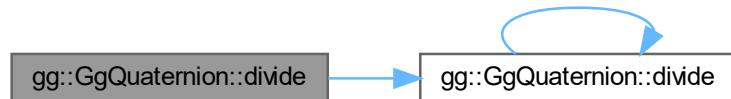
<i>q</i>	GgQuaternion 型の四元数。
----------	---------------------

戻り値

q で割った四元数。

gg.h の 3928 行目に定義があります。

呼び出し関係図:



8.11.3.7 divide() [2/4]

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

四元数を別の四元数で除算した結果を返す。

引数

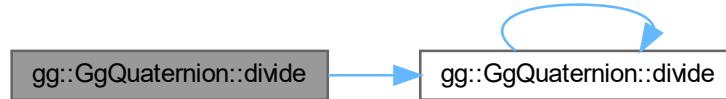
v	四元数を格納した GgVector 型の変数.
---	---

戻り値

v で割った四元数.

[gg.h](#) の 3917 行目に定義があります。

呼び出し関係図:



8.11.3.8 divide() [3/4]

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

四元数を別の四元数で除算した結果を返す.

引数

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

戻り値

a で割った四元数.

[gg.h](#) の 3903 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

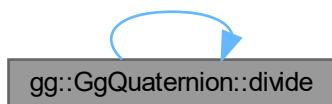
<i>x</i>	割る四元数の <i>x</i> 要素。
<i>y</i>	割る四元数の <i>y</i> 要素。
<i>z</i>	割る四元数の <i>z</i> 要素。
<i>w</i>	割る四元数の <i>w</i> 要素。

戻り値

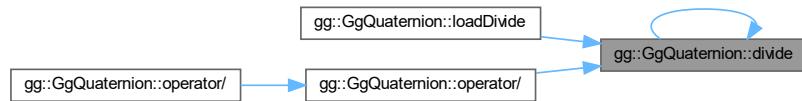
(x, y, z, w) を割った四元数。

[gg.h](#) の 3891 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.10 euler() [1/2]

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

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

引数

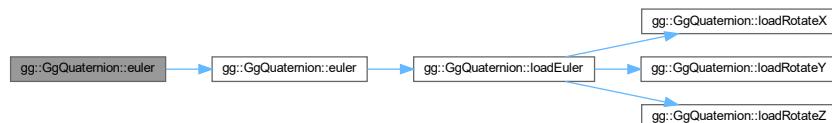
e	オイラー角を表す GLfloat 型の 3 要素の配列変数 (heading, pitch, roll).
----------	---

戻り値

回転した四元数.

gg.h の 4242 行目に定義があります。

呼び出し関係図:



8.11.3.11 euler() [2/2]

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

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

引数

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

戻り値

回転した四元数.

gg.h の 4230 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.12 get()

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

四元数を取り出す。

引数

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

gg.h の 4426 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 4493 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.14 `getConjugateMatrix()` [2/3]

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

四元数の共役が表す回転の変換行列を `m` に求める.

引数

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

`gg.h` の 4483 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

<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 4471 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 4459 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.17 `getMatrix()` [2/3]

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

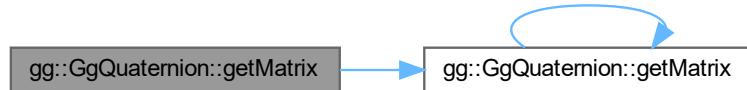
四元数が表す回転の変換行列を `m` に求める.

引数

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

gg.h の 4449 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]
```

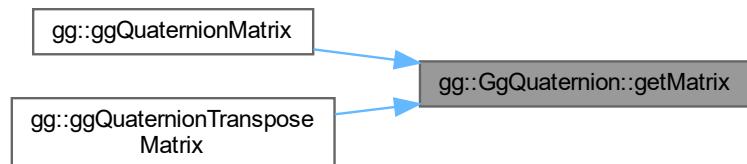
四元数が表す回転の変換行列を a に求める。

引数

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 4439 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 4414 行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 3532 行目に定義があります。

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 3520 行目に定義があります。

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 3509 行目に定義があります。

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

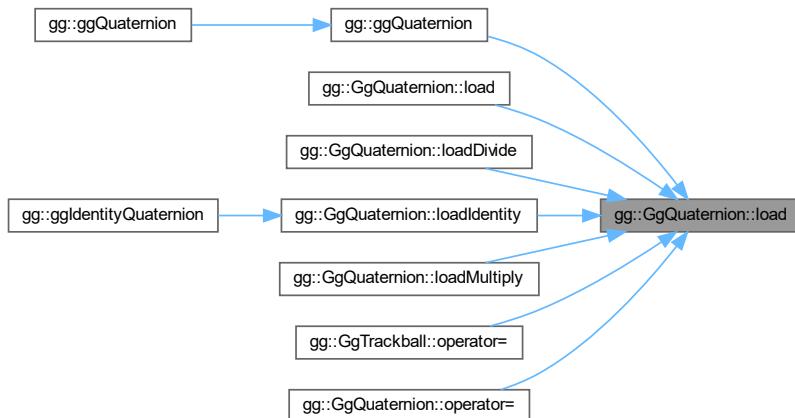
<i>x</i>	四元数の <i>x</i> 要素。
<i>y</i>	四元数の <i>y</i> 要素。
<i>z</i>	四元数の <i>z</i> 要素。
<i>w</i>	四元数の <i>w</i> 要素。

戻り値

設定した四元数。

gg.h の 3494 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 3584 行目に定義がります。

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 3573 行目に定義がります。

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

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 3562 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

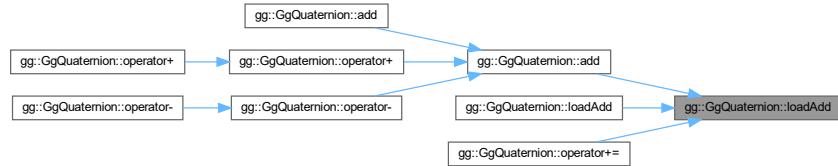
x	加える四元数の x 要素。
y	加える四元数の y 要素。
z	加える四元数の z 要素。
w	加える四元数の w 要素。

戻り値

(x, y, z, w) を加えた四元数.

gg.h の 3547 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 4333 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

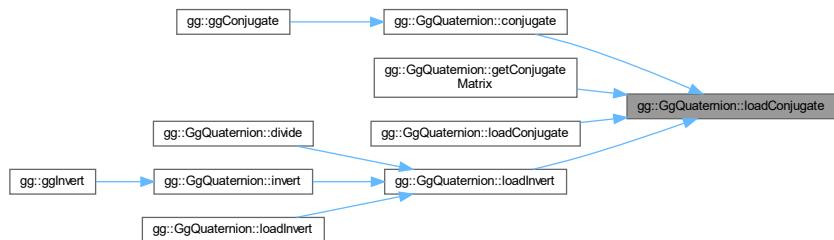
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数。
----------	--------------------------------

戻り値

共役四元数。

gg.cpp の 3272 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 3731 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

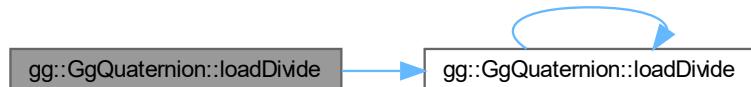
v	四元数を格納した GgVector 型の変数。
---	-------------------------

戻り値

v で割った四元数。

gg.h の 3720 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

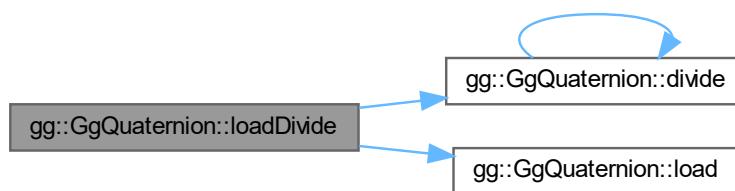
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

a で割った四元数.

gg.h の 3709 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 3697 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.34 loadEuler() [1/2]

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

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

引数

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 4217 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

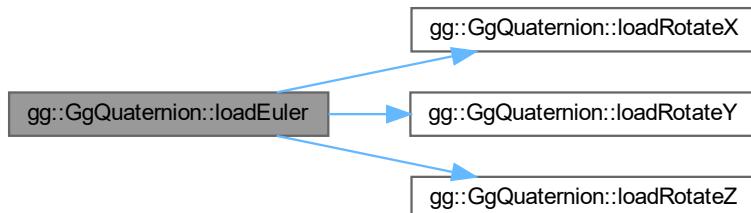
<i>heading</i>	y 軸中心の回転角。
<i>pitch</i>	x 軸中心の回転角。
<i>roll</i>	z 軸中心の回転角。

戻り値

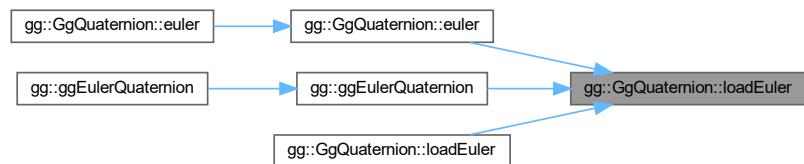
格納した回転を表す四元数。

gg.cpp の 3241 行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

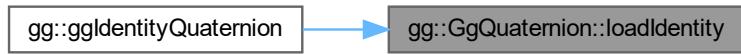
格納された単位元。

`gg.h` の 4067 行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

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

戻り値

四元数の逆元.

gg.h の 4352 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

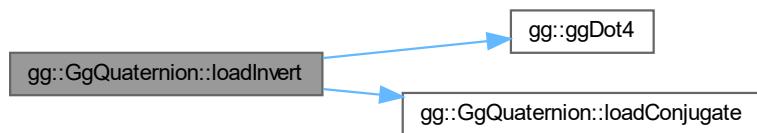
a	四元数を格納した GLfloat 型の 4 要素の配列変数.
---	--------------------------------

戻り値

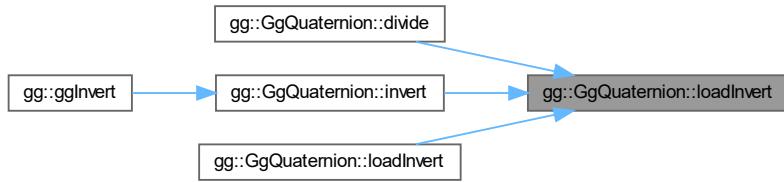
四元数の逆元.

gg.cpp の 3286 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.39 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

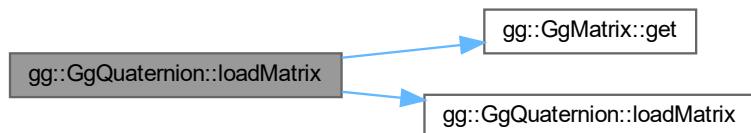
<i>m</i>	Ggmatrix 型の変換行列。
----------	------------------

戻り値

m による回転の変換に相当する四元数。

gg.h の 4057 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

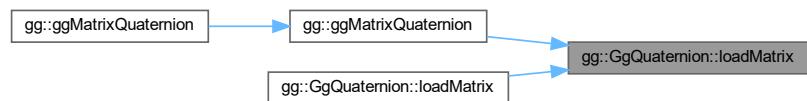
a	GLfloat 型の 16 要素の変換行列.
---	------------------------

戻り値

a による回転の変換に相当する四元数.

gg.h の 4045 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 3683 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 3672 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 3661 行目に定義がります。

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

戻り値

(x, y, z, w) を掛けた四元数。

gg.h の 3649 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.45 loadNormalize() [1/2]

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

引数に指定した四元数を正規化して格納する。

引数

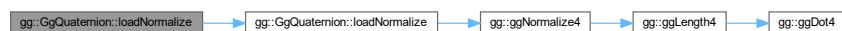
<code>q</code>	<code>GgQuaternion</code> 型の四元数。
----------------	----------------------------------

戻り値

正規化された四元数。

`gg.h` の 4314 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

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

戻り値

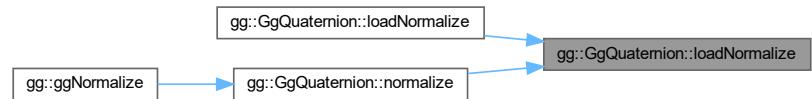
正規化された四元数.

gg.cpp の 3257 行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 4101 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

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

格納された回転を表す四元数。

gg.h の 4090 行目に定義があります。

呼び出し関係図:



8.11.3.49 loadRotate() [3/3]

```
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

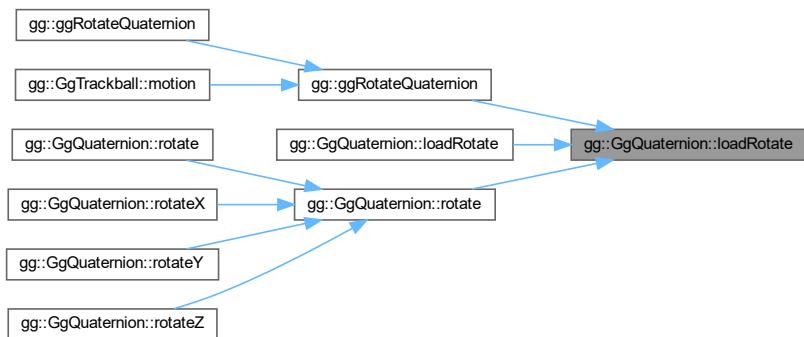
<i>x</i>	軸ベクトルの x 成分.
<i>y</i>	軸ベクトルの y 成分.
<i>z</i>	軸ベクトルの z 成分.
<i>a</i>	回転角.

戻り値

格納された回転を表す四元数.

gg.cpp の 3175 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

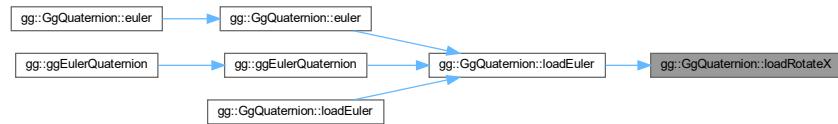
<i>a</i>	回転角.
----------	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3199 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

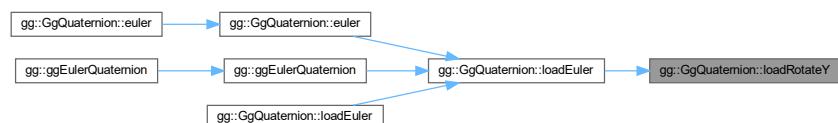
a	回転角。
---	------

戻り値

格納された回転を表す四元数。

gg.cpp の 3213 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

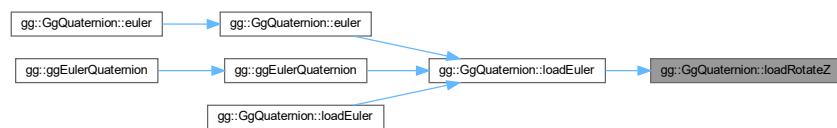
a	回転角.
---	------

戻り値

格納された回転を表す四元数.

gg.cpp の 3227 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

q	GgQuaternion 型の四元数.
r	GgQuaternion 型の四元数.
t	補間パラメータ.

戻り値

格納した q, r を t で内分した四元数.

gg.h の 4269 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>	<code>GgQuaternion</code> 型の四元数。
<i>a</i>	四元数を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
<i>t</i>	補間パラメータ。

戻り値

格納した *q*, *a* を *t* で内分した四元数。

`gg.h` の 4282 行目に定義がります。

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 4295 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

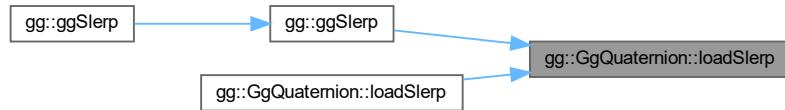
<i>a</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>b</i>	四元数を格納した GLfloat 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

格納した a, b を t で内分した四元数.

gg.h の 4255 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 3635 行目に定義があります。

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

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 3624 行目に定義があります。

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

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 3613 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

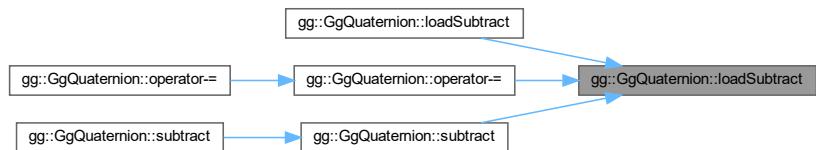
<i>x</i>	引く四元数の x 要素.
<i>y</i>	引く四元数の y 要素.
<i>z</i>	引く四元数の z 要素.
<i>w</i>	引く四元数の w 要素.

戻り値

(*x*, *y*, *z*, *w*) を引いた四元数.

gg.h の 3598 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>	GgQuaternion 型の四元数.
----------	---------------------

戻り値

q を掛けた四元数.

gg.h の 3877 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

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

戻り値

v を掛けた四元数.

gg.h の 3866 行目に定義があります。

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

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

戻り値

`a` を掛けた四元数。

`gg.h` の 3853 行目に定義があります。

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

<code>x</code>	掛ける四元数の <code>x</code> 要素。
<code>y</code>	掛ける四元数の <code>y</code> 要素。
<code>z</code>	掛ける四元数の <code>z</code> 要素。
<code>w</code>	掛ける四元数の <code>w</code> 要素。

戻り値

(x, y, z, w) を掛けた四元数。

`gg.h` の 3841 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 3480 行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 4390 行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 4022 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 4018 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 4014 行目に定義があります。

被呼び出し関係図:

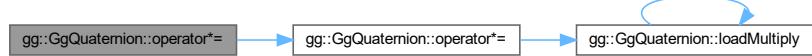


8.11.3.70 operator*=() [1/3]

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

gg.h の 3974 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 3970 行目に定義があります。

呼び出し関係図:

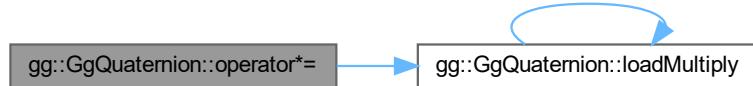


8.11.3.72 operator*=() [3/3]

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

gg.h の 3966 行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 3998 行目に定義があります。

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 3994 行目に定義があります。

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 3990 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.76 operator+=() [1/3]

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

gg.h の 3950 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 3946 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 3942 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 4010 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 4006 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 4002 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:

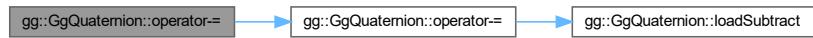


8.11.3.82 operator-() [1/3]

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

gg.h の 3962 行目に定義がります。

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 3958 行目に定義がります。

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 3954 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:

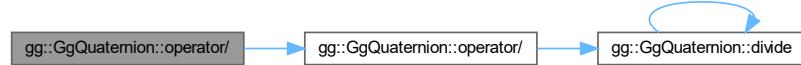


8.11.3.85 operator() [1/3]

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

gg.h の 4034 行目に定義があります。

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 4030 行目に定義があります。

呼び出し関係図:

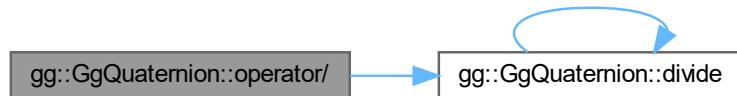


8.11.3.87 operator/() [3/3]

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

gg.h の 4026 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.11.3.88 operator/() [1/3]

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

gg.h の 3986 行目に定義があります。

呼び出し関係図:

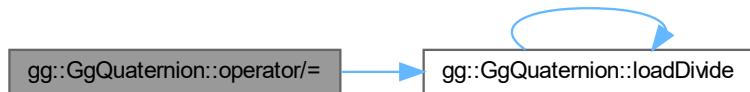


8.11.3.89 `operator/()` [2/3]

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

gg.h の 3982 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 3978 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



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

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

gg.h の 3938 行目に定義があります。

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 3934 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 4163 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

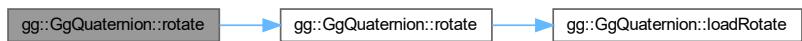
v	軸ベクトルを表す GLfloat 型の 3 要素の配列変数.
a	回転角.

戻り値

回転した四元数.

gg.h の 4152 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

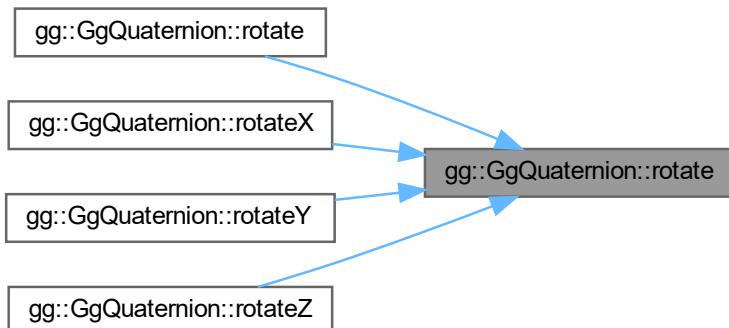
回転した四元数。

`gg.h` の 4139 行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

a	回転角.
---	------

戻り値

回転した四元数.

gg.h の 4174 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 4185 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

<code>a</code>	回転角.
----------------	------

戻り値

回転した四元数.

`gg.h` の 4196 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

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

戻り値

四元数を *q* に対して *t* で内分した結果.

`gg.h` の 4378 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 <code>GLfloat</code> 型の 4 要素の配列変数.
<i>t</i>	補間パラメータ.

戻り値

四元数を *a* に対して *t* で内分した結果.

`gg.h` の 4364 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>	<code>GgQuaternion</code> 型の四元数.
----------	----------------------------------

戻り値

`v` を引いた四元数.

`gg.h` の 3827 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

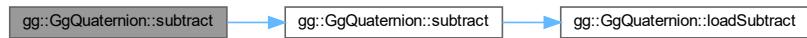
<code>v</code>	四元数を格納した <code>GgVector</code> 型の変数.
----------------	--------------------------------------

戻り値

`v` を引いた四元数.

`gg.h` の 3816 行目に定義があります。

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

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

戻り値

`a` を引いた四元数.

[gg.h](#) の 3805 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

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

戻り値

(x, y, z, w) を引いた四元数.

[gg.h](#) の 3793 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.12 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 6695 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

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

gg.h の 6711 行目に定義があります。

呼び出し関係図:



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 6729 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 6746 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 6779 行目に定義がります。

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 6769 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 6761 行目に定義があります。

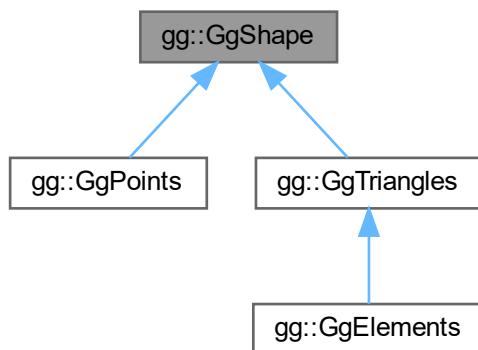
このクラス詳解は次のファイルから抽出されました:

- [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 6099 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

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

`gg.h` の 6114 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 6124 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・コンストラクタは使用禁止。

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。

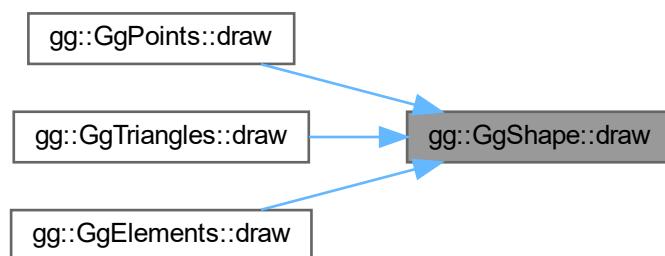
引数

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

[gg.h](#) の 6175 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 6144 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 6164 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 6154 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- gg.h

8.14 gg::GgSimpleObj クラス

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

公開メンバ関数

- **GgSimpleObj** (const std::string &name, bool normalize=false)
- virtual ~**GgSimpleObj** ()
デストラクタ.
- 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 7984 行目に定義があります。

8.14.2 構築子と解体子

8.14.2.1 GgSimpleObj()

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

コンストラクタ.

引数

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

[gg.cpp](#) の 5825 行目に定義がります。

呼び出し関係図:



8.14.2.2 ~GgSimpleObj()

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

デストラクタ.

[gg.h](#) の 8007 行目に定義がります。

8.14.3 関数詳解

8.14.3.1 draw()

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

Wavefront OBJ 形式のデータを描画する手続き.

引数

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

[gg.cpp](#) の 5853 行目に定義がります。

被呼び出し関係図:



8.14.3.2 get()

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

形状データの取り出し.

戻り値

[GgTriangles](#) 型の形状データのポインタ.

[gg.h](#) の 8016 行目に定義があります。

呼び出し関係図:



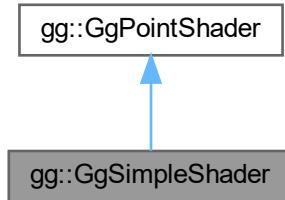
このクラス詳解は次のファイルから抽出されました:

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

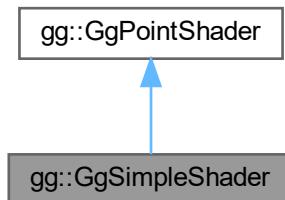
8.15 gg::GgSimpleShader クラス

```
#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 *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 7043 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 7060 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 7077 行目に定義があります。

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 7095 行目に定義があります。

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 7107 行目に定義があります。

8.15.2.5 ~GgSimpleShader()

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

デストラクタ.

gg.h の 7118 行目に定義があります。

8.15.3 関数詳解

8.15.3.1 load() [1/2]

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

シェーダのソースファイルを読み込んでプログラムオブジェクトを作成する。

引数

<i>files</i>	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)。

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 7164 行目に定義があります。

8.15.3.2 load() [2/2]

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

シェーダのソースプログラムの文字列からプログラムオブジェクトを作成する。

引数

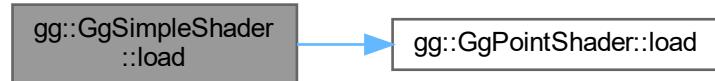
<i>vert</i>	バーテックスシェーダのソースプログラムの文字列。
<i>frag</i>	フラグメントシェーダのソースプログラムの文字列 (空文字列なら不使用)。
<i>geom</i>	ジオメトリシェーダのソースプログラムの文字列 (空文字列なら不使用)。
<i>nvarying</i>	フィードバックする <i>varying</i> 変数の数 (0 なら不使用)。
<i>varyings</i>	フィードバックする <i>varying</i> 変数のリスト (nullptr なら不使用)。

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 5808 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

<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::GgPointShader`を再実装しています。

`gg.h` の 7258 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>	<code>GgMatrix</code> 型の投影変換行列。
<i>mv</i>	<code>GgMatrix</code> 型のモデルビュー変換行列。
<i>mn</i>	<code>GgMatrix</code> 型のモデルビュー変換行列の法線変換行列。

`gg.h` の 7236 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>	<code>GLfloat</code> 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>	<code>GLfloat</code> 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 7247 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 7223 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 7211 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 7191 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 7201 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 7179 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 7125 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 7751 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 7802 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 7780 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 7866 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 7903 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>	<code>GgMatrix</code> 型の投影変換行列。
<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 7936 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 7791 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 7764 行目に定義があります。

8.15.3.20 use() [9/13]

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

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

引数

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

gg.h の 7842 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 7885 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 7920 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 7828 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 7813 行目に定義があります。

呼び出し関係図:



このクラス詳解は次のファイルから抽出されました:

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

8.16 `gg::GgTexture` クラス

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

公開メンバ関数

- **GgTexture** (const GLvoid *image, GLsizei width, GLsizei height, GLenum format=GL_RGB, GLenum type=GL_UNSIGNED_BYTE, GLenum internal=GL_RGBA, GLenum wrap=GL_CLAMP_TO_EDGE, bool swizzle=true)
- virtual ~**GgTexture** ()
- **GgTexture** (const GgTexture &o)=delete
- **GgTexture & operator=** (const GgTexture &o)=delete
- void **bind** () const
- void **unbind** () const
- void **swapRandB** (bool swizzle) const
- const GLsizei & **getWidth** () const
- const GLsizei & **getHeight** () const
- void **getSize** (GLsizei *size) const
- const GLsizei * **getSize** () const
- const GLuint & **getTexture** () const

8.16.1 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 5171 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,
    bool swizzle = true ) [inline]
```

メモリ上のデータからテクスチャを作成するコンストラクタ.

引数

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

gg.h の 5193 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 5211 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 5230 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 5265 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 5286 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]
```

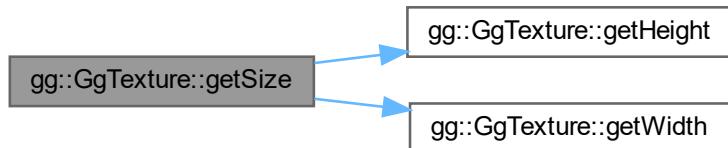
使用しているテクスチャのサイズを取り出す。

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 5275 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 5296 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 5255 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 swapRandB()

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

テクスチャの赤と青を交換する

引数

swizzle	赤と青を交換するなら true
---------	-----------------

gg.cpp の 3917 行目に定義があります。

8.16.3.9 unbind()

```
void gg::GgTexture::unbind ( ) const [inline]
```

テクスチャの使用終了(このテクスチャを使用しなくなったら呼び出す)。

gg.h の 5238 行目に定義があります。

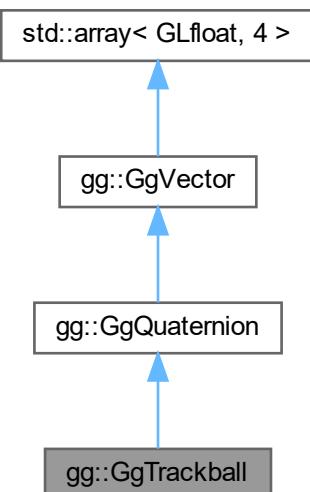
このクラス詳解は次のファイルから抽出されました:

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

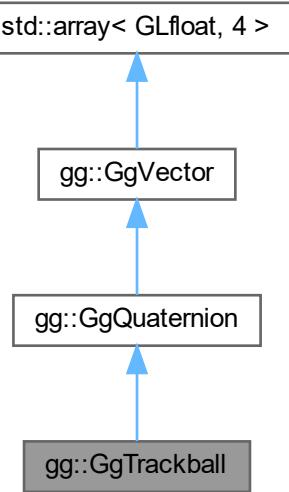
8.17 gg::GgTrackball クラス

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

gg::GgTrackball の継承関係図



gg::GgTrackball 連携図



公開メンバ関数

- `GgTrackball (const GgQuaternion &q=ggIdentityQuaternion())`
- `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 4751 行目に定義があります。

8.17.2 構築子と解体子

8.17.2.1 GgTrackball()

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

コンストラクタ.

引数

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

[gg.h](#) の 4766 行目に定義があります。

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 4774 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

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

覚え書き

マウスのドラッグ開始時 (マウスボタンを押したとき) に呼び出す.

[gg.cpp](#) の 3341 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 3406 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 4946 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 4936 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 4926 行目に定義があります。

8.17.3.6 `getScale()` [1/3]

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

トラックボール処理の換算係数を取り出す。

戻り値

トラックボールの換算係数のポインタ。

[gg.h](#) の 4895 行目に定義があります。

8.17.3.7 `getScale()` [2/3]

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

トラックボール処理の換算係数を取り出す。

引数

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

[gg.h](#) の 4915 行目に定義があります。

8.17.3.8 `getScale()` [3/3]

```
const GLfloat gg::GgTrackball::getScale (
    int direction ) const [inline]
```

トラックボール処理の換算係数を取り出す。

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向。
------------------	-----------------------

[gg.h](#) の 4905 行目に定義があります。

8.17.3.9 `getStart()` [1/3]

```
const GLfloat * gg::GgTrackball::getStart ( ) const [inline]
```

トラックボール処理の開始位置を取り出す。

戻り値

トラックボールの開始位置のポインタ.

gg.h の 4866 行目に定義があります。

8.17.3.10 getStart() [2/3]

```
void gg::GgTrackball::getStart (
    GLfloat * position ) const [inline]
```

トラックボール処理の開始位置を取り出す.

引数

<i>position</i>	トラックボールの開始位置を格納する 2 要素の配列.
-----------------	----------------------------

gg.h の 4884 行目に定義があります。

8.17.3.11 getStart() [3/3]

```
const GLfloat & gg::GgTrackball::getStart (
    int direction ) const [inline]
```

トラックボール処理の開始位置を取り出す.

引数

<i>direction</i>	0 なら x 方向, 1 なら y 方向.
------------------	-----------------------

gg.h の 4874 行目に定義があります。

8.17.3.12 motion()

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

回転の変換行列を計算する.

引数

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

覚え書き

マウスのドラッグ中に呼び出す.

gg.cpp の 3357 行目に定義があります。

呼び出し関係図:



8.17.3.13 operator=()

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

代入.

引数

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

gg.h の 4783 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

w	領域の横幅.
h	領域の高さ.

覚え書き

ウィンドウのリサイズ時に呼び出す。

gg.cpp の 3328 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 4809 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

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

gg.cpp の 3310 行目に定義があります。

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

q	修正分の回転角の四元数。
-----	--------------

gg.cpp の 3385 行目に定義があります。

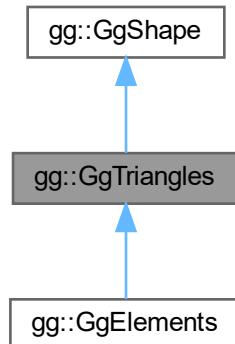
このクラス詳解は次のファイルから抽出されました:

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

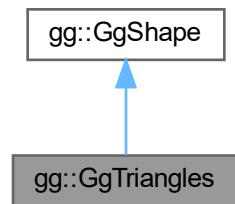
8.18 gg::GgTriangles クラス

```
#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 6340 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

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

`gg.h` の 6353 行目に定義があります。

8.18.2.2 `GgTriangles()` [2/2]

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

コンストラクタ。

引数

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

gg.h の 6366 行目に定義があります。

8.18.2.3 ~GgTriangles()

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

デストラクタ.

gg.h の 6380 行目に定義があります。

8.18.3 関数詳解

8.18.3.1 draw()

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

三角形の描画.

引数

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

gg::GgShapeを再実装しています。

gg::GgElementsで再実装されています。

gg.cpp の 5084 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 6399 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 6389 行目に定義があります。

8.18.3.4 `load()`

```
void gg::GgTriangles::load (
    const GgVertex * vert,
    GLsizei count,
    GLenum usage = GL_STATIC_DRAW )
```

バッファオブジェクトを確保して頂点属性を格納する。

引数

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

gg.cpp の 5065 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 6411 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

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

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

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

公開メンバ関数

- [GgUniformBuffer \(\)](#)
- [GgUniformBuffer \(const T *data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [GgUniformBuffer \(const T &data, GLsizei count, GLenum usage=GL_STATIC_DRAW\)](#)
- [virtual ~GgUniformBuffer \(\)](#)
- [const GLuint & **getTarget** \(\) const](#)
- [GLsizeiptr **getStride** \(\) const](#)
- [const GLsizei & **getCount** \(\) const](#)
- [const GLuint & **getBuffer** \(\) const](#)

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

8.19.1 詳解

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

ユニフォームバッファオブジェクト。

覚え書き

ユニフォーム変数を格納するバッファオブジェクトの基底クラス。

gg.h の 5765 行目に定義があります。

8.19.2 構築子と解体子

8.19.2.1 GgUniformBuffer() [1/3]

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

コンストラクタ。

gg.h の 5768 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 (nullptr ならデータを転送しない).
<i>count</i>	データの数.
<i>usage</i>	バッファオブジェクトの使い方.

gg.h の 5768 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 5768 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 5768 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 5853 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 6064 行目に定義があります。

8.19.3.3 fill()

```
template<typename T >
void gg::GgUniformBuffer< T >::fill (
    const GLvoid * data,
    GLint offset = 0,
    GLsizei size = sizeof(T),
    GLint first = 0,
    GLsizei count = 0 ) const [inline]
```

ユニフォームバッファオブジェクトの全ブロックのメンバーを同じデータを格納する。

引数

<i>data</i>	格納するデータ。
<i>offset</i>	格納先のメンバのブロックの先頭からのバイトオフセット。
<i>size</i>	格納するデータの一個あたりのバイト数。
<i>first</i>	格納先のバッファオブジェクトのブロックの先頭の番号。
<i>count</i>	格納するデータの数。

gg.h の 5981 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 5845 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 5835 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 5825 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 5815 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

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

`gg.h` の 5922 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

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

gg.h の 5903 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 5871 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 5883 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 6016 行目に定義がります。

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 5943 行目に定義がります。

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 5861 行目に定義がります。

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 5891 行目に定義があります。

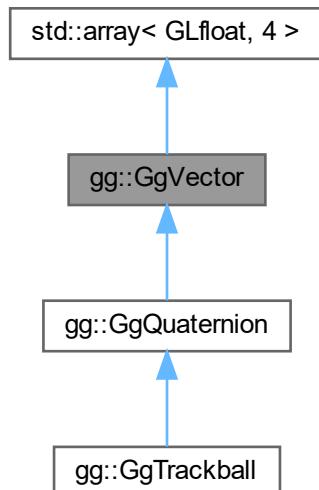
このクラス詳解は次のファイルから抽出されました:

- [gg.h](#)

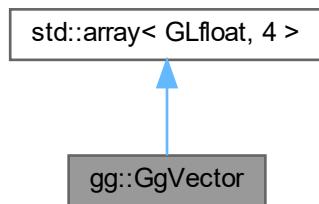
8.20 gg::GgVector クラス

```
#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 1556 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 1563 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 1575 行目に定義がります。

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 1585 行目に定義がります。

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 1595 行目に定義がります。

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

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

戻り値

オブジェクトと v の 3 要素の距離.

gg.h の 1834 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 1878 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 1813 行目に定義があります。

呼び出し関係図:



8.20.3.4 dot4()

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

GgVector 型の 4 要素の内積.

引数

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

戻り値

オブジェクトと `v` のそれぞれの 4 要素の内積.

`gg.h` の 1857 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 1823 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 1867 行目に定義があります。

呼び出し関係図:



8.20.3.7 normalize3()

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

GgVector 型の 4 要素の正規化.

戻り値

GLfloat 型の 4 要素の配列変数.

[gg.h](#) の 1844 行目に定義があります。

呼び出し関係図:



8.20.3.8 normalize4()

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

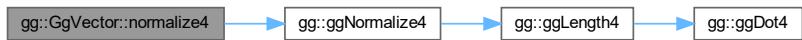
GgVector 型の 4 要素の正規化.

戻り値

GLfloat 型の 4 要素の配列変数.

gg.h の 1888 行目に定義があります。

呼び出し関係図:



8.20.3.9 operator*() [1/2]

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

GgVector 型の積を返す.

引数

v	GgVector 型の変数.
---	----------------

戻り値

オブジェクトの各要素と v の各要素の要素ごとを乗じたオブジェクト.

gg.h の 1710 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>c</code>	<code>GLfloat</code> 型の変数.
----------------	----------------------------

戻り値

オブジェクトの各要素に `c` を乗じたオブジェクト.

`gg.h` の 1721 行目に定義がります。

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 1731 行目に定義がります。

8.20.3.12 `operator*=()` [2/2]

```
GgVector & gg::GgVector::operator*=
  (GLfloat c) [inline]
```

`GgVector` 型の各要素にスカラーを乗じる.

引数

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

戻り値

オブジェクトの各要素に `c` を乗じたオブジェクトの参照.

`gg.h` の 1746 行目に定義がります。

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のオブジェクト.

gg.h の 1606 行目に定義があります。

8.20.3.14 operator+() [2/2]

```
GgVector gg::GgVector::operator+ (
    GLfloat c ) const [inline]
```

GgVector 型の各要素にスカラーを足した和を返す.

引数

c	GLfloat 型の値.
---	--------------

戻り値

a の各要素に b を足した和のオブジェクト.

gg.h の 1617 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>	<code>GgVector</code> 型の変数.
----------------	-----------------------------

戻り値

オブジェクトの各要素に `v` の各要素をそれぞれ加算したオブジェクトの参照.

`gg.h` の 1628 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>	<code>GLfloat</code> 型の値.
----------------	---------------------------

戻り値

オブジェクトの各要素に `c` を足したオブジェクトの参照.

`gg.h` の 1643 行目に定義があります。

8.20.3.17 `operator-()` [1/2]

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

`GgVector` 型の差を返す.

引数

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

戻り値

オブジェクトの各要素と `v` の各要素の要素ごとの差のオブジェクト.

`gg.h` の 1658 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 を引いたオブジェクト.

gg.h の 1669 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

v	GgVector 型の変数.
----------	----------------

戻り値

オブジェクトの各要素に v の各要素をそれぞれ減算したオブジェクトの参照.

gg.h の 1680 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>	GLfloat 型の変数.
----------------	---------------

戻り値

オブジェクトの各要素から `c` を引いたオブジェクトの参照.

`gg.h` の 1695 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>	GgVector 型の変数.
----------------	----------------

戻り値

オブジェクトの各要素を `v` の各要素で要素ごとに割った結果のオブジェクト.

`gg.h` の 1761 行目に定義があります。

8.20.3.22 operator/() [2/2]

```
GgVector gg::GgVector::operator/ (
    GLfloat c ) const [inline]
```

`GgVector` 型の各要素をスカラーで割った商を返す.

引数

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

戻り値

オブジェクトの各要素を `c` で割った商のオブジェクト.

`gg.h` の 1787 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 1772 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

c	GLfloat 型の変数.
---	---------------

戻り値

オブジェクトの各要素を c で割ったオブジェクトの参照.

gg.h の 1798 行目に定義があります。

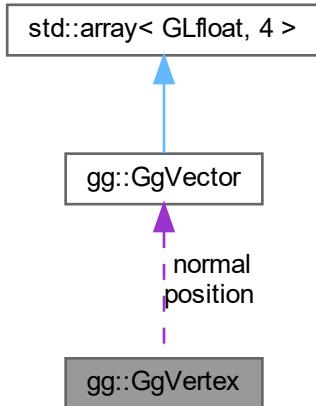
このクラス詳解は次のファイルから抽出されました:

- [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 6279 行目に定義があります。

8.21.2 構築子と解体子

8.21.2.1 GgVertex() [1/4]

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

コンストラクタ。

gg.h の 6290 行目に定義があります。

8.21.2.2 GgVertex() [2/4]

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

コンストラクタ。

引数

<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 6300 行目に定義があります。

8.21.2.3 GgVertex() [3/4]

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

コンストラクタ。

引数

<i>px</i>	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 6316 行目に定義がります。

8.21.2.4 GgVertex() [4/4]

```
gg::GgVertex::GgVertex (
    const GLfloat * pos,
    const GLfloat * norm ) [inline]
```

コンストラクタ.

引数

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

[gg.h](#) の 6331 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 6285 行目に定義がります。

8.21.3.2 position

`GgVector gg::GgVertex::position`

位置.

[gg.h](#) の 6282 行目に定義がります。

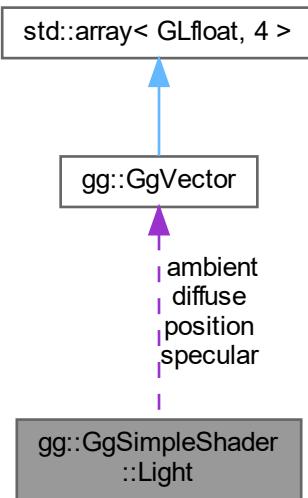
この構造体詳解は次のファイルから抽出されました:

- [gg.h](#)

8.22 gg::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 7266 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 7268 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 7269 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 7271 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 7270 行目に定義があります。

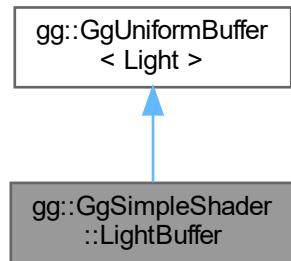
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

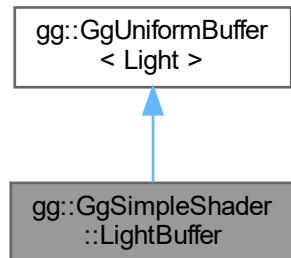
8.23 gg::GgSimpleShader::LightBuffer クラス

```
#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 7277 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, <code>glBufferData()</code> の第 4 引数の <i>usage</i> に渡される。

gg.h の 7289 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 7305 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 7317 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 7509 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 7497 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 5410 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 7343 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 5384 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 5537 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

<code>diffuse</code>	光源の強度の拡散反射光成分を格納した <code>GgVector</code> 型の変数。
<code>first</code>	値を設定する光源データの最初の番号, デフォルトは 0.
<code>count</code>	値を設定する光源データの数, デフォルトは 1.

`gg.cpp` の 5462 行目に定義があります。

8.23.3.8 `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

<code>diffuse</code>	光源の強度の拡散反射光成分を格納した <code>GLfloat</code> 型の 4 要素の配列変数。
<code>first</code>	値を設定する光源データの最初の番号, デフォルトは 0.
<code>count</code>	値を設定する光源データの数, デフォルトは 1.

`gg.h` の 7389 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 5436 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 5591 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 7485 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 7472 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 5565 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 5514 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 7426 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 5488 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 7519 行目に定義があります。

被呼び出し関係図:



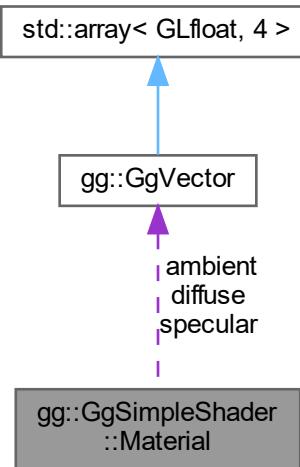
このクラス詳解は次のファイルから抽出されました:

- [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 7530 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 7532 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 7533 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 7535 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 7534 行目に定義があります。

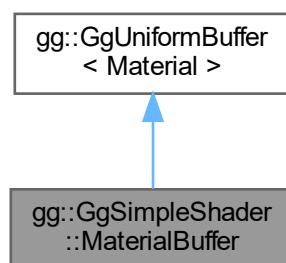
この構造体詳解は次のファイルから抽出されました:

- `gg.h`

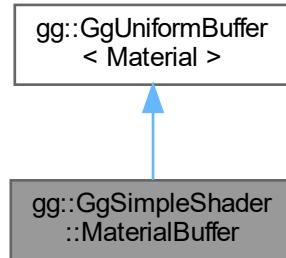
8.25 gg::GgSimpleShader::MaterialBuffer クラス

`#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 7541 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 7553 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 7569 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 7581 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 7730 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 7718 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 7607 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 5617 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 5701 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 g,
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 5675 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 7635 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 5646 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 5791 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 5770 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 7687 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 5744 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 7740 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

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

8.26 Menu クラス

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

公開メンバ関数

- `Menu (const Config &config)`
- `Menu (const Menu &menu)=delete`
- `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 行目に定義があります。

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 13 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 49 行目に定義がります。

8.26.3 関数詳解

8.26.3.1 draw()

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

描画する

Menu.cpp の 74 行目に定義がります。

8.26.3.2 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 GgApp::Window クラス

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

公開メンバ関数

- [Window \(const std::string &title="GLFW Window", int width=640, int height=480, int fullscreen=0, GLFWwindow *share=nullptr\)](#)
- [Window \(const Window &w\)=delete](#)
- [virtual ~Window \(\)](#)
- [Window & operator= \(const Window &w\)=delete](#)
- [auto * get \(\) const](#)
- [void setClose \(int flag=GLFW_TRUE\) const](#)
- [bool shouldClose \(\) const](#)
- [operator bool \(\)](#)
- [void swapBuffers \(\) const](#)
- [void restoreViewport \(\) const](#)
- [void updateViewport \(\)](#)
- [auto getWidth \(\) const](#)
- [auto getHeight \(\) const](#)
- [auto getFboWidth \(\) const](#)
- [auto getFboHeight \(\) const](#)
- [const auto & getSize \(\) const](#)
- [void getSize \(GLsizei *size\) const](#)
- [const auto getFboSize \(\) const](#)
- [void getFboSize \(GLsizei *fboSize\) const](#)
- [auto getAspect \(\) const](#)
- [bool getKey \(int key\) const](#)
- [void selectInterface \(int no\)](#)
- [void setVelocity \(GLfloat vx, GLfloat vy, GLfloat vz=0.1f\)](#)
- [int getLastKey \(\)](#)
- [auto getArrow \(int direction=0, int mods=0\) const](#)
- [auto getArrowX \(int mods=0\) const](#)
- [auto getArrowY \(int mods=0\) const](#)
- [void getArrow \(GLfloat *arrow, int mods=0\) const](#)
- [auto getShiftArrowX \(\) const](#)
- [auto getShiftArrowY \(\) const](#)
- [void getShiftArrow \(GLfloat *shift_arrow\) const](#)
- [auto getControlArrowX \(\) const](#)
- [auto getControlArrowY \(\) const](#)

- void `getControlArrow` (GLfloat *control_arrow) const
- auto `getAltArrowX` () const
- auto `getAltArrowY` () const
- void `getAltArrow` (GLfloat *alt_arrow) const
- const auto * `getMouse` () const
- void `getMouse` (GLfloat *position) const
- auto `getMouse` (int direction) const
- auto `getMouseX` () const
- auto `getMouseY` () const
- const auto * `getWheel` () const
- void `getWheel` (GLfloat *rotation) const
- auto `getWheel` (int direction) const
- auto `getWheelX` () const
- auto `getWheelY` () const
- const auto & `getTranslation` (int button=GLFW_MOUSE_BUTTON_1) const
- auto `getTranslationMatrix` (int button=GLFW_MOUSE_BUTTON_1) const
- auto `getScrollMatrix` (int button=GLFW_MOUSE_BUTTON_1) const
- auto `getRotation` (int button=GLFW_MOUSE_BUTTON_1) const
- auto `getRotationMatrix` (int button=GLFW_MOUSE_BUTTON_1) const
- void `resetRotation` ()
- void `resetTranslation` ()
- void `reset` ()
- void * `getUserPointer` () const
- void `setUserPointer` (void *pointer)
- void `setResizeFunc` (void(*func)(const Window *window, int width, int height))
- void `setKeyboardFunc` (void(*func)(const Window *window, int key, int scancode, int action, int mods))
- void `setMouseFunc` (void(*func)(const Window *window, int button, int action, int mods))
- void `setWheelFunc` (void(*func)(const Window *window, double x, double y))

8.27.1 詳解

ウィンドウ関連の処理。

覚え書き

GLFW を使って OpenGL のウィンドウを操作するラッパークラス。

GgApp.h の 102 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]

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

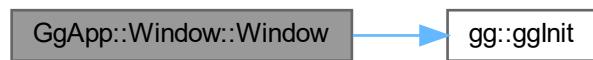
コンストラクタ。

引数

<i>title</i>	ウィンドウタイトルの文字列.
<i>width</i>	開くウィンドウの幅, フルスクリーン時は無視され実際のディスプレイの幅が使われる.
<i>height</i>	開くウィンドウの高さ, フルスクリーン時は無視され実際のディスプレイの高さが使われる.
<i>fullscreen</i>	フルスクリーン表示を行うディスプレイ番号, 0ならフルスクリーン表示を行わない.
<i>share</i>	共有するコンテキスト, <code>nullptr</code> ならコンテキストを共有しない.

[GgApp.cpp](#) の 324 行目に定義があります。

呼び出し関係図:



8.27.2.2 Window() [2/2]

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

8.27.2.3 ~Window()

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

デストラクタ.

[GgApp.h](#) の 227 行目に定義があります。

8.27.3 関数詳解

8.27.3.1 get()

```
auto * GgApp::Window::get ( ) const [inline]
```

ウィンドウの識別子のポインタを取得する。

戻り値

GLFWwindow 型のウィンドウ識別子のポインタ。

[GgApp.h](#) の 246 行目に定義があります。

8.27.3.2 getAltArrowX()

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

ALT キーを押しながら矢印キーを押したときの現在の X 値を得る。

戻り値

ALT キーを押しながら矢印キーを押したときの現在の X 値。

[GgApp.h](#) の 570 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.3 getAltArrowY()

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

ALT キーを押しながら矢印キーを押したときの現在の Y 値を得る。

戻り値

ALT キーを押しながら矢印キーを押したときの現在の Y 値。

[GgApp.h](#) の 580 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.4 getAltArrow()

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

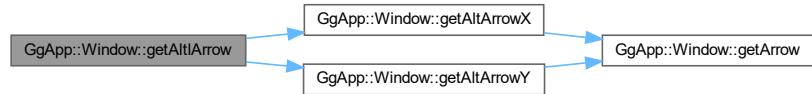
ALT キーを押しながら矢印キーを押したときの現在の値を得る。

引数

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

[GgApp.h](#) の 590 行目に定義があります。

呼び出し関係図:



8.27.3.5 getArrow() [1/2]

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

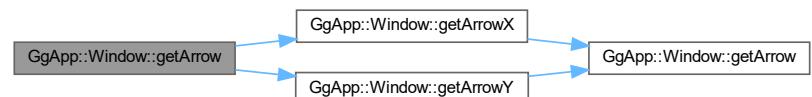
矢印キーの現在の値を得る。

引数

<i>arrow</i>	矢印キーの値を格納する GLfloat[2] の配列。
<i>mods</i>	修飾キーの状態 (0:なし, 1: SHIFT, 2: CTRL, 3: ALT)。

[GgApp.h](#) の 497 行目に定義があります。

呼び出し関係図:



8.27.3.6 getArrow() [2/2]

```
auto GgApp::Window::getArrow (
    int direction = 0,
    int mods = 0 ) const [inline]
```

矢印キーの現在の値を得る。

引数

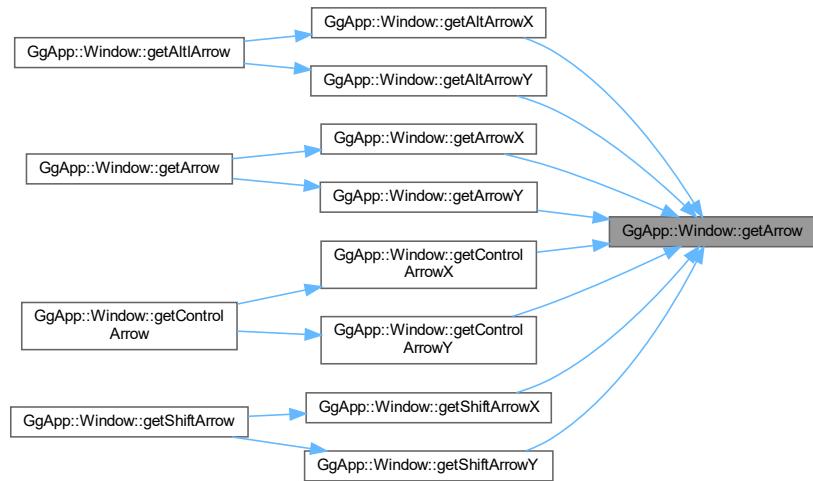
<i>direction</i>	方向 (0: X, 1:Y).
<i>mods</i>	修飾キーの状態 (0:なし, 1, SHIFT, 2: CTRL, 3: ALT).

戻り値

矢印キーの値.

[GgApp.h](#) の 463 行目に定義があります。

被呼び出し関係図:



8.27.3.7 getArrowX()

```
auto GgApp::Window::getArrowX (
    int mods = 0 ) const [inline]
```

矢印キーの現在の X 値を得る.

引数

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

戻り値

矢印キーの X 値.

GgApp.h の 475 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.8 getArrowY()

```
auto GgApp::Window::getArrowY ( int mods = 0 ) const [inline]
```

矢印キーの現在の Y 値を得る。

引数

mods	修飾キーの状態 (0:なし, 1: SHIFT, 2: CTRL, 3: ALT).
------	--

戻り値

矢印キーの Y 値。

GgApp.h の 486 行目に定義がります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.9 `getAspect()`

```
auto GgApp::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。

[GgApp.h](#) の 400 行目に定義があります。

8.27.3.10 `getControlArrow()`

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

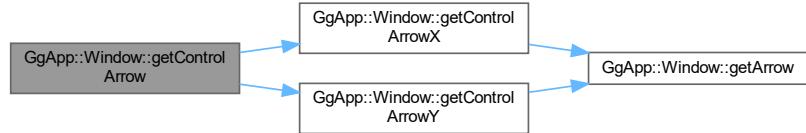
CTRL キーを押しながら矢印キーを押したときの現在の値を得る。

引数

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

[GgApp.h](#) の 559 行目に定義があります。

呼び出し関係図:



8.27.3.11 `getControlArrowX()`

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

CTRL キーを押しながら矢印キーを押したときの現在の X 値を得る.

戻り値

CTRL キーを押しながら矢印キーを押したときの現在の X 値.

[GgApp.h](#) の 539 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.12 `getControlArrowY()`

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

CTRL キーを押しながら矢印キーを押したときの現在の Y 値を得る.

戻り値

CTRL キーを押しながら矢印キーを押したときの現在の Y 値.

[GgApp.h](#) の 549 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.13 `getFboHeight()`

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

FBO の高さを得る.

戻り値

FBO の高さ.

GgApp.h の 348 行目に定義がります。

被呼び出し関係図:



8.27.3.14 getFboSize() [1/2]

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

FBO のサイズを得る.

戻り値

FBO の幅と高さを格納した GLsizei 型の 2 要素の配列.

GgApp.h の 379 行目に定義がります。

8.27.3.15 getFboSize() [2/2]

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

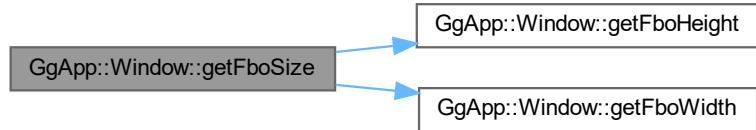
FBO のサイズを得る.

引数

fboSize	FBO の幅と高さを格納した GLsizei 型の 2 要素の配列.
---------	------------------------------------

GgApp.h の 389 行目に定義がります。

呼び出し関係図:



8.27.3.16 `getFboWidth()`

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

FBO の横幅を得る.

戻り値

FBO の横幅.

`GgApp.h` の 338 行目に定義があります。

被呼び出し関係図:



8.27.3.17 `getHeight()`

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

ウィンドウの高さを得る.

戻り値

ウィンドウの高さ.

GgApp.h の 328 行目に定義があります。

被呼び出し関係図:



8.27.3.18 getKey()

```
bool GgApp::Window::getKey ( int key ) const [inline]
```

キーが押されているかどうかを判定する.

戻り値

キーが押されていれば true.

GgApp.h の 410 行目に定義があります。

8.27.3.19 getLastKey()

```
int GgApp::Window::getLastKey ( ) [inline]
```

最後にタイプしたキーを得る.

戻り値

最後にタイプしたキーの文字.

GgApp.h の 448 行目に定義があります。

8.27.3.20 `getMouse()` [1/3]

```
const auto * GgApp::Window::getMouse ( ) const [inline]
```

マウスカーソルの現在位置を得る.

戻り値

マウスカーソルの現在位置を格納した `GLfloat` 型の 2 要素の配列.

[GgApp.h](#) の 601 行目に定義があります。

8.27.3.21 `getMouse()` [2/3]

```
void GgApp::Window::getMouse (
    GLfloat * position ) const [inline]
```

マウスカーソルの現在位置を得る.

引数

<code>position</code>	マウスカーソルの現在位置を格納した <code>GLfloat</code> 型の 2 要素の配列.
-----------------------	--

[GgApp.h](#) の 612 行目に定義があります。

8.27.3.22 `getMouse()` [3/3]

```
auto GgApp::Window::getMouse (
    int direction ) const [inline]
```

マウスカーソルの現在位置を得る.

引数

<code>direction</code>	方向 (0:X, 1:Y).
------------------------	----------------

戻り値

`direction` 方向のマウスカーソルの現在位置.

[GgApp.h](#) の 625 行目に定義があります。

8.27.3.23 getMouseX()

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

マウスカーソルの現在位置の X 座標を得る。

戻り値

`direction` 方向のマウスカーソルの X 方向の現在位置。

[GgApp.h](#) の 636 行目に定義があります。

8.27.3.24 getMouseY()

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

マウスカーソルの現在位置の Y 座標を得る。

戻り値

`direction` 方向のマウスカーソルの Y 方向の現在位置。

[GgApp.h](#) の 647 行目に定義があります。

8.27.3.25 getRotation()

```
auto GgApp::Window::getRotation ( int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボールの回転変換行列を得る。

引数

<code>button</code>	回転変換行列を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------------	--

戻り値

回転を行う GgQuaternion 型の四元数。

[GgApp.h](#) の 769 行目に定義があります。

8.27.3.26 getRotationMatrix()

```
auto GgApp::Window::getRotationMatrix (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボールの回転変換行列を得る。

引数

<i>button</i>	回転変換行列を取得するマウスボタン (GLFW_MOUSE_BUTTON_[1,2]).
---------------	--

戻り値

回転を行う GgMatrix 型の変換行列。

GgApp.h の 782 行目に定義があります。

8.27.3.27 getScrollMatrix()

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

マウスによってオブジェクトの平行移動の変換行列を得る。

引数

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

戻り値

クリッピング座標系で平行移動を行う GgMatrix 型の変換行列。

GgApp.h の 749 行目に定義があります。

8.27.3.28 getShiftArrow()

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

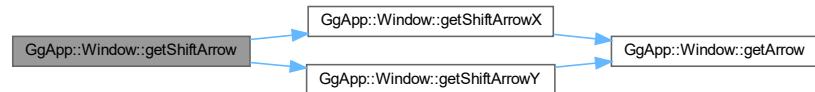
SHIFT キーを押しながら矢印キーを押したときの現在の値を得る。

引数

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

GgApp.h の 528 行目に定義があります。

呼び出し関係図:



8.27.3.29 `getShiftArrowX()`

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

SHIFT キーを押しながら矢印キーを押したときの現在の X 値を得る.

戻り値

SHIFT キーを押しながら矢印キーを押したときの現在の X 値.

GgApp.h の 508 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.30 `getShiftArrowY()`

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

SHIFT キーを押しながら矢印キーを押したときの現在の Y 値を得る。

戻り値

SHIFT キーを押しながら矢印キーを押したときの現在の Y 値。

[GgApp.h](#) の 518 行目に定義があります。

呼び出し関係図:



被呼び出し関係図:



8.27.3.31 `getSize() [1/2]`

```
const auto & GgApp::Window::getSize ( ) const [inline]
```

ウィンドウのサイズを得る。

戻り値

ウィンドウの幅と高さを格納した GLsizei 型の 2 要素の配列の参照。

[GgApp.h](#) の 358 行目に定義があります。

8.27.3.32 `getSize() [2/2]`

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

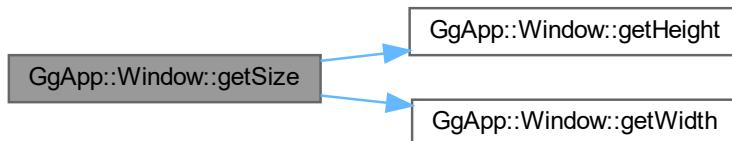
ウィンドウのサイズを得る。

引数

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

[GgApp.h](#) の 368 行目に定義があります。

呼び出し関係図:



8.27.3.33 getTranslation()

```
const auto & GgApp::Window::getTranslation (
    int button = GLFW_MOUSE_BUTTON_1 ) const [inline]
```

トラックボール処理を考慮したマウスによるスクロールの変換行列を得る.

引数

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

戻り値

平行移動量を格納した `GLfloat[3]` の配列のポインタ.

[GgApp.h](#) の 716 行目に定義があります。

8.27.3.34 getTranslationMatrix()

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

マウスによって視点の平行移動の変換行列を得る.

引数

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

戻り値

視点座標系で平行移動を行う `GgMatrix` 型の変換行列.

`GgApp.h` の 729 行目に定義があります。

8.27.3.35 `getUserPointer()`

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

ユーザー ポインタを取り出す.

戻り値

保存されているユーザ ポインタ.

`GgApp.h` の 824 行目に定義があります。

8.27.3.36 `getWheel() [1/3]`

```
const auto * GgApp::Window::getWheel ( ) const [inline]
```

マウスホイールの回転量を得る.

戻り値

マウスホイールの回転量を格納した `GLfloat` 型の 2 要素の配列.

`GgApp.h` の 658 行目に定義があります。

8.27.3.37 `getWheel() [2/3]`

```
void GgApp::Window::getWheel (
    GLfloat * rotation ) const [inline]
```

マウスホイールの回転量を得る.

引数

<i>rotation</i>	マウスホイールの回転量を格納した GLfloat 型の 2 要素の配列.
-----------------	--------------------------------------

GgApp.h の 669 行目に定義がります。

8.27.3.38 getWheel() [3/3]

```
auto GgApp::Window::getWheel (   
    int direction ) const [inline]
```

マウスホイールの回転量を得る.

引数

<i>direction</i>	方向 (0:X, 1:Y).
------------------	----------------

戻り値

direction 方向のマウスホイールの回転量.

GgApp.h の 682 行目に定義がります。

8.27.3.39 getWheelX()

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

マウスホイールの X 方向の回転量を得る.

戻り値

マウスホイールの X 方向の回転量.

GgApp.h の 693 行目に定義がります。

8.27.3.40 getWheelY()

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

マウスホイールの Y 方向の回転量を得る.

戻り値

マウスホイールの Y 方向の回転量.

GgApp.h の 704 行目に定義がります。

8.27.3.41 getWidth()

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

ウィンドウの横幅を得る.

戻り値

ウィンドウの横幅.

[GgApp.h](#) の 318 行目に定義があります。

被呼び出し関係図:



8.27.3.42 operator bool()

```
GgApp::Window::operator bool ( ) [explicit]
```

イベントを取得してループを継続すべきかどうか調べる.

戻り値

ループを継続すべきなら true.

[GgApp.cpp](#) の 417 行目に定義があります。

8.27.3.43 operator=()

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

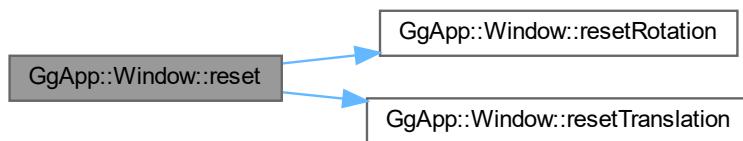
8.27.3.44 reset()

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

トラックボール・マウスホイール・矢印キーの値を初期化する。

[GgApp.h](#) の 810 行目に定義があります。

呼び出し関係図:



8.27.3.45 resetRotation()

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

トラックボール処理を初期化する。

[GgApp.h](#) の 792 行目に定義があります。

被呼び出し関係図:



8.27.3.46 resetTranslation()

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

平行移動量を初期化する。

[GgApp.h](#) の 801 行目に定義があります。

被呼び出し関係図:



8.27.3.47 restoreViewport()

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

ビューポートを元のサイズに復帰する。

[GgApp.h](#) の 287 行目に定義があります。

8.27.3.48 selectInterface()

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

インターフェースを選択する。

引数

no	インターフェース番号.
----	-------------

[GgApp.h](#) の 425 行目に定義があります。

8.27.3.49 setClose()

```
void GgApp::Window::setClose (
    int flag = GLFW_TRUE ) const [inline]
```

ウィンドウのクローズフラグを設定する。

引数

<code>flag</code>	クローズフラグ, 0 (GLFW_FALSE) 以外ならウィンドウを閉じる.
-------------------	--

[GgApp.h](#) の 256 行目に定義があります。

8.27.3.50 `setKeyboardFunc()`

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

ユーザ定義の `keyboard` 関数を設定する。

引数

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

[GgApp.h](#) の 854 行目に定義があります。

8.27.3.51 `setMouseFunc()`

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

ユーザ定義の `mouse` 関数を設定する。

引数

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

[GgApp.h](#) の 864 行目に定義があります。

8.27.3.52 `setResizeFunc()`

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

ユーザ定義の `resize` 関数を設定する。

引数

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

`GgApp.h` の 844 行目に定義がります。

8.27.3.53 `setUserPointer()`

```
void GgApp::Window::setUserPointer ( void * pointer ) [inline]
```

任意のユーザポインタを保存する.

引数

<i>pointer</i>	保存するユーザポインタ.
----------------	--------------

`GgApp.h` の 834 行目に定義がります。

8.27.3.54 `setVelocity()`

```
void GgApp::Window::setVelocity ( GLfloat vx,  
                                 GLfloat vy,  
                                 GLfloat vz = 0.1f ) [inline]
```

マウスの移動速度を設定する.

引数

<i>vx</i>	x 方向の移動速度.
<i>vy</i>	y 方向の移動速度.
<i>vz</i>	z 方向の移動速度.

`GgApp.h` の 438 行目に定義がります。

8.27.3.55 `setWheelFunc()`

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

ユーザ定義の `wheel` 関数を設定する.

引数

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

[GgApp.h](#) の 874 行目に定義があります。

8.27.3.56 `shouldClose()`

```
bool GgApp::Window::shouldClose ( ) const [inline]
```

ウィンドウを閉じるべきかどうか調べる。

戻り値

ウィンドウを閉じるべきなら `true`。

[GgApp.h](#) の 266 行目に定義があります。

8.27.3.57 `swapBuffers()`

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

カラーバッファを入れ替える。

[GgApp.cpp](#) の 467 行目に定義があります。

8.27.3.58 `updateViewport()`

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

ビューポートのサイズを更新する。

[GgApp.cpp](#) の 485 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

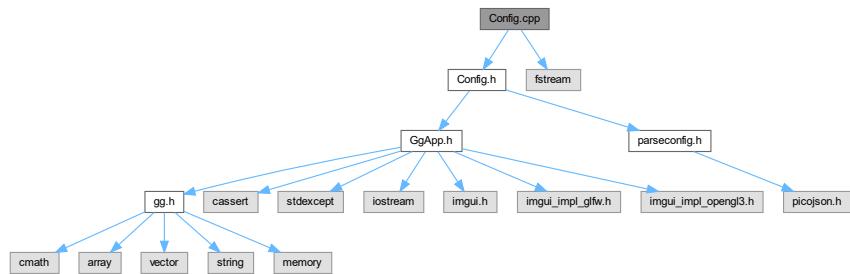
- [GgApp.h](#)
- [GgApp.cpp](#)

Chapter 9

ファイル詳解

9.1 Config.cpp ファイル

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



変数

- constexpr GgSimpleShader::Light defaultLight

9.1.1 詳解

設定構造体の実装

著者

Kohe Tokoi

日付

November 15, 2022

Config.cpp に定義があります。

9.1.2 変数詳解

9.1.2.1 defaultLight

```
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.cpp

[詳解]

```
00001
00008 #include "Config.h"
00009
00010 // 標準ライブラリ
00011 #include <fstream>
00012
00013 // デフォルトの光源データ
00014 constexpr GgSimpleShader::Light defaultLight
00015 {
00016   { 0.2f, 0.2f, 0.2f, 1.0f }, // 環境光成分
00017   { 1.0f, 1.0f, 1.0f, 0.0f }, // 扇散反射光成分
00018   { 1.0f, 1.0f, 1.0f, 0.0f }, // 鏡面反射光成分
00019   { 0.5f, 0.5f, 1.0f, 1.0f } // 光源位置
00020 };
00021
00022 //
00023 // コンストラクタ
00024 //
00025 Config::Config() :
00026   winSize{ 960, 540 },
00027   menuFont{ "Mplus1-Regular.ttf" },
00028   menuFontSize{ 20.0f },
00029   light{ defaultLight },
00030   model{ "logo.obj" },
00031 #if defined(GL_GLES_PROTOTYPES)
00032   shader{ "simple_es3.vert", "simple_es3.frag" }
00033 #else
00034   shader{ "simple.vert", "simple.frag" }
00035 #endif
00036 {
00037 }
00038
00039 //
00040 // ファイルから構成データを読み込むコンストラクタ
00041 //
00042 Config::Config(const std::string& filename) :
00043   Config{}
00044 {
00045   // 構成ファイルが読み込めなかったらデフォルト値の構成ファイルを作る
00046   if (!load(filename)) save(filename);
00047 }
00048
00049 //
00050 // デストラクタ
00051 //
00052 Config::~Config()
00053 {
00054 }
00055
00056 //
00057 // 設定ファイルを読み込む
00058 //
```

```
00059 bool Config::load(const std::string& filename)
00060 {
00061     // 構成ファイルを開く
00062     std::ifstream file{ Utf8ToTChar(filename) };
00063
00064     // 開けなかったらエラー
00065     if (!file) return false;
00066
00067     // JSON の読み込み
00068     picojson::value value;
00069     file >> value;
00070     file.close();
00071
00072     // 構成データの取り出し
00073     const auto& object{ value.get<picojson::object>() };
00074
00075     //
00076     // 構成データの読み込み
00077     //
00078
00079     // ウィンドウサイズ
00080     getValue(object, "window_size", winSize);
00081
00082     // メニューフォント
00083     getString(object, "menu_font", menuFont);
00084
00085     // メニューフォントサイズ
00086     getValue(object, "menu_font_size", menuFontSize);
00087
00088     // 光源
00089     getVector(object, "ambient", light.ambient);
00090     getVector(object, "diffuse", light.diffuse);
00091     getVector(object, "specular", light.specular);
00092     getVector(object, "position", light.position);
00093
00094     // モデル
00095     getString(object, "model", model);
00096
00097     // シェーダ
00098     getString(object, "shader", shader);
00099
00100     // オブジェクトが空だったらエラー
00101     if (object.empty()) return false;
00102
00103     return true;
00104 }
00105
00106 //
00107 // 設定ファイルを書き出す
00108 //
00109 bool Config::save(const std::string& filename) const
00110 {
00111     // 構成ファイルを開く
00112     std::ofstream file{ Utf8ToTChar(filename) };
00113
00114     // 開けなかったらエラー
00115     if (!file) return false;
00116
00117     // 構成データの書き出しに使うオブジェクト
00118     picojson::object object;
00119
00120     //
00121     // 構成データの書き出し
00122     //
00123
00124     // ウィンドウサイズ
00125     setValue(object, "window_size", winSize);
00126
00127     // メニューフォント
00128     setString(object, "menu_font", menuFont);
00129
00130     // メニューフォントサイズ
00131     setValue(object, "menu_font_size", menuFontSize);
00132
00133     // 光源
00134     setVector(object, "ambient", light.ambient);
00135     setVector(object, "diffuse", light.diffuse);
00136     setVector(object, "specular", light.specular);
00137     setVector(object, "position", light.position);
00138
00139     // モデル
00140     setString(object, "model", model);
00141
00142     // シェーダ
00143     setString(object, "shader", shader);
00144
00145     // 構成出データをシリализして JSON で保存
```

```

00146     picojson::value v{ object };
00147     file << v.serialize(true);
00148     file.close();
00149
00150     return true;
00151 }
00152
00153 #if defined(_MSC_VER)
00154 //
00155 // For VC++ MFC Convert UTF-8 to TCHAR, or Convert TCHAR to UTF-8. VC++ MFC用 UTF-8⇒TCHARの変換処理
00156 //
00157 // Original author: mt-u
00158 // Copyright (c) 2013 mt-u
00159 // https://gist.github.com/mt-u/6878251
00160 //
00161 // Modified by: Kohe Tokoi
00162 //
00163
00164 //
00165 // UTF-8 文字列を CString に変換する
00166 //
00167 CString Utf8ToTChar(const std::string& string)
00168 {
00169     // UTF-8 文字列を UTF-16 に変換した後の文字列の長さを求める
00170     const INT length{ MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, NULL, 0) };
00171
00172     // 変換結果の格納に必要な長さのメモリを確保する
00173     std::vector<WCHAR> utf16(length);
00174
00175     // UTF-8 文字列を UTF-16 に変換する
00176     MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, utf16.data(), length);
00177
00178     // 変換した文字列を CString にして返す
00179     return CString{ utf16.data(), static_cast<int>(wcslen(utf16.data())) };
00180 }
00181
00182 //
00183 // Cstring を UTF-8 文字列に変換する
00184 //
00185 std::string TCharToUtf8(const CString& cstring)
00186 {
00187     // 与えられた文字列を一旦 UTF-16 に変換する
00188     CStringW cstringw{ cstring };
00189
00190     // UTF-16 を UTF-8 に変換した後の文字列の長さを求める
00191     const INT length{ WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, NULL, 0, NULL, NULL) };
00192
00193     // 変換結果の格納に必要な長さのメモリを確保する
00194     std::vector<CHAR> utf8(length);
00195
00196     // UTF-16 を UTF-8 に変換する
00197     WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, utf8.data(), length, NULL, NULL);
00198
00199     // 変換した文字列を std::string にして返す
00200     return std::string{ utf8.data(), strlen(utf8.data()) };
00201 }
00202 #endif

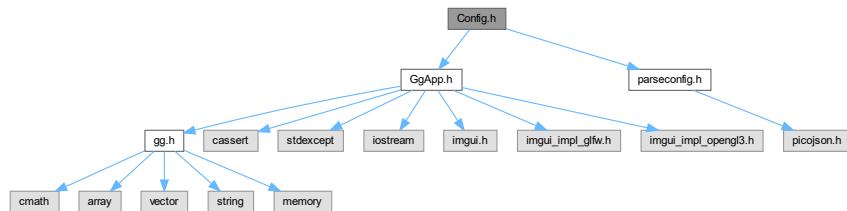
```

9.3 Config.h ファイル

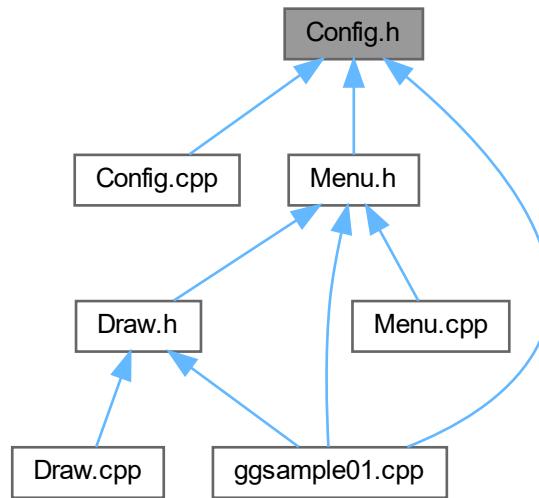
```

#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`

9.3.1 詳解

構成データクラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Config.h](#) に定義があります。

9.3.2 マクロ定義詳解

9.3.2.1 **TCharToUtf8**

```
#define TCHARToUtf8(  
    cstring ) (cstring)
```

[Config.h](#) の 34 行目に定義がります。

9.3.2.2 **Utf8ToTChar**

```
#define Utf8ToTChar(  
    string ) (string)
```

[Config.h](#) の 33 行目に定義がります。

9.3.3 型定義詳解

9.3.3.1 **pathChar**

```
using pathChar = char
```

[Config.h](#) の 32 行目に定義がります。

9.3.3.2 **pathString**

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

[Config.h](#) の 31 行目に定義がります。

9.4 Config.h

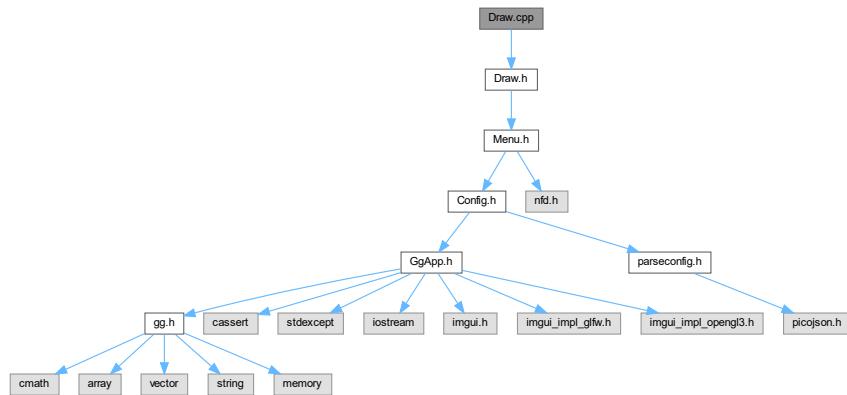
[詳解]

```
00001 #pragma once
00002
00010
00011 // アプリケーションの設定
00012 #include "GgApp.h"
00013
00014 // 構成ファイルの読み取り補助
00015 #include "parseconfig.h"
00016
00017 #if defined(_MSC_VER)
00018 //
00019 // Windows / Visual C++ におけるパス名
00020 //
00021
00022 // 文字コード
00023 #include <atlstr.h>
00024 using pathString = CString;
00025 using pathChar = wchar_t;
00026
00027 // 文字コード変換
00028 extern CString Utf8ToTChar(const std::string& string);
00029 extern std::string TCHARToUtf8(const CString& cstring);
00030 #else
00031 using pathString = std::string;
00032 using pathChar = char;
00033 #define Utf8ToTChar(string) (string)
00034 #define TCHARToUtf8(cstring) (cstring)
00035 #endif
00036
00037 class Config
00038 {
00039     // メニュークラスから参照する
00040     friend class Menu;
00041
00042     // ウィンドウサイズ
00043     std::array<GLsizei, 2> winSize;
00044
00045     // メニューフォント名
00046     std::string menuFont;
00047
00048     // メニューフォントサイズ
00049     float menuFontSize;
00050
00051     // 光源
00052     GgSimpleShader::Light light;
00053
00054     // 形状ファイル名
00055     std::string model;
00056
00057     // シェーダのソースファイル名
00058     std::array<std::string, 3> shader;
00059
00060     // public:
00061
00062     public:
00063
00064     Config();
00065
00066     Config(const std::string& filename);
00067
00068     virtual ~Config();
00069
00070     auto getWidth() const
00071     {
00072         return winSize[0];
00073     }
00074
00075     auto getHeight() const
00076     {
00077         return winSize[1];
00078     }
00079
00080     bool load(const std::string& filename);
00081
00082     bool save(const std::string& filename) const;
00083
00084 };
```

9.5 Draw.cpp ファイル

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

Draw.cpp の依存先関係図:



9.5.1 詳解

図形の描画クラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

Draw.cpp に定義があります。

9.6 Draw.cpp

[詳解]

```

00001
00008 #include "Draw.h"
00009
00010 //
00011 // コンストラクタ
00012 //
00013 Draw::Draw(const Menu& menu) :
00014     light{ *menu.light.get() },
00015     model{ *menu.model.get() },
00016     shader{ *menu.shader.get() }
00017 {
00018     // 背景色を設定する
00019     glClearColor(0.1f, 0.2f, 0.3f, 0.0f);
00020
00021     // 隠面消去処理を設定する
00022     glEnable(GL_DEPTH_TEST);
00023     glEnable(GL_CULL_FACE);
00024 }
```

```

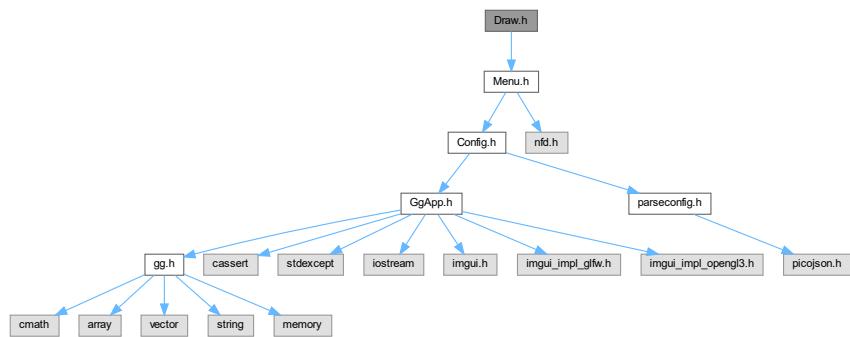
00025
00026 // デストラクタ
00027 // デストラクタ
00028 //
00029 Draw::~Draw()
00030 {
00031 }
00032
00033 // 描画する
00034 // 描画する
00035 //
00036 void Draw::draw(const GgMatrix& mp, const GgMatrix& mv) const
00037 {
00038     // シェーダプログラムを指定する
00039     shader.use(mp, mv, light);
00040
00041     // 図形を描画する
00042     model.draw();
00043 }

```

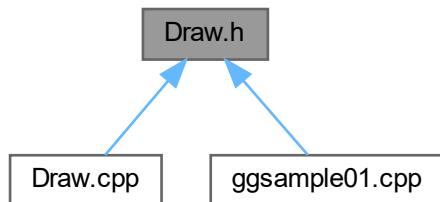
9.7 Draw.h ファイル

#include "Menu.h"

Draw.h の依存先関係図:



被依存関係図:



クラス

- class Draw

9.7.1 詳解

図形の描画クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Draw.h](#) に定義があります。

9.8 Draw.h

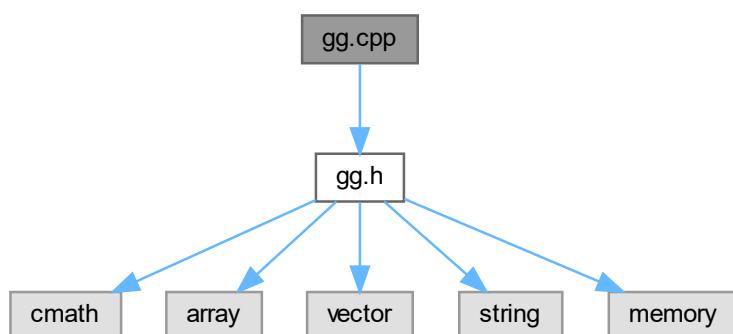
[詳解]

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

9.9 gg.cpp ファイル

#include "gg.h"

gg.cpp の依存先関係図:



9.9.1 詳解

ゲームグラフィックス特論の宿題用補助プログラム GLFW3 版の定義。

著者

Kohe Tokoi

日付

March 31, 2021

gg.cpp に定義があります。

9.10 gg.cpp

【詳解】

```

00001 /*
00002 ** ゲームグラフィックス特論用補助プログラム GLFW3 版
00003 **
00004
00005 Copyright (c) 2011-2022 Kohe Tokoi. All Rights Reserved.
00006
00007 Permission is hereby granted, free of charge, to any person obtaining a copy
00008 of this software and associated documentation files (the "Software"), to deal
00009 in the Software without restriction, including without limitation the rights
00010 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00011 copies or substantial portions of the Software.
00012
00013 The above copyright notice and this permission notice shall be included in
00014 all copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
00019 KOHE TOKOI BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
00020 AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
00021 CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00022
00023 **
00024 */
00025 #include "gg.h"
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037 // 標準ライブラリ
00038 #include <cfloat>
00039 #include <cstdlib>
00040 #include <iostream>
00041 #include <fstream>
00042 #include <sstream>
00043 #include <map>
00044
00045 #define READ_TEXTURE_COORDINATE_FROM_OBJ 0
00046
00047
00048 // Windows (Visual Studio) のとき
00049 #if defined(_MSC_VER)
00050 // リリースビルドではコンソールを出さない
00051 # if !defined(_DEBUG)
00052 # pragma comment(linker, "/subsystem:\\"windows\" /entry:\\"mainCRTStartup\"")
00053 # endif
00054 // リンクするライブラリ
00055 # pragma comment(lib, "lib\\\" GLFW3_PLATFORM \"\\\" GLFW3_CONFIGURATION \"\\\" glfw3.lib")
00056 #endif
00057
00058 // OpenGL 3.2 の API のエントリポイント
00059 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
00060 PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
00061 PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
00062 PFNGLACTIVETEXTUREPROC glActiveTexture;
00063 PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;

```

```

00064 PFNGLATTACHSHADERPROC glAttachShader;
00065 PFNGLBEGINCONDITIONALRENDERNVPROC glBeginConditionalRenderNV;
00066 PFNGLBEGINCONDITIONALRENDERPROC glBeginConditionalRender;
00067 PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00068 PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00069 PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00070 PFNGLBEGINQUERYPROC glBeginQuery;
00071 PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
00072 PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
00073 PFNGLBINDBUFFERBASEPROC glBindBufferBase;
00074 PFNGLBINDBUFFERPROC glBindBuffer;
00075 PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
00076 PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
00077 PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
00078 PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
00079 PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
00080 PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
00081 PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
00082 PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
00083 PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00084 PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00085 PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
00086 PFNGLBINDSAMPLERPROC glBindSampler;
00087 PFNGLBINDSAMPLERSPROC glBindSamplers;
00088 PFNGLBINDTEXTUREPROC glBindTexture;
00089 PFNGLBINDTEXTURESPROC glBindTextures;
00090 PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
00091 PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
00092 PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
00093 PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
00094 PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
00095 PFNGLBLENDBARRIERKHRPROC glBindBarrierKHR;
00096 PFNGLBLENDBARRIERNVPROC glBindBarrierNV;
00097 PFNGLBLENDCOLORPROC glBindColor;
00098 PFNGLBLENDEQUATIONIARBPROC glBindEquationiARB;
00099 PFNGLBLENDEQUATIONIPROC glBindEquationi;
00100 PFNGLBLENDEQUATIONPROC glBindEquation;
00101 PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00102 PFNGLBLENDEQUATIONSEPARATEIPROC glBindEquationSeparatei;
00103 PFNGLBLENDEQUATIONSEPARATEPROC glBindEquationSeparate;
00104 PFNGLBLENDFUNCARBPROC glBindFuncARB;
00105 PFNGLBLENDFUNCIPROC glBindFunci;
00106 PFNGLBLENDFUNCPROC glBindFunc;
00107 PFNGLBLENDFUNCSEPARATEIARBPROC glBindFuncSeparateiARB;
00108 PFNGLBLENDFUNCSEPARATEIPROC glBindFuncSeparatei;
00109 PFNGLBLENDFUNCSEPARATEPROC glBindFuncSeparate;
00110 PFNGLBLENDPARAMETERINVPROC glBindParameteriNV;
00111 PFNGLBLITFRAMEBUFFERPROC glBindFramebuffer;
00112 PFNGLBLITNAMEDFRAMEBUFFERPROC glBindNamedFramebuffer;
00113 PFNGLBUFFERADDRESSRANGEVPROC glBindBufferAddressRangeNV;
00114 PFNGLBUFFERDATAPROC glBindBufferData;
00115 PFNGLBUFFERPAGECOMMITMENTARBPROC glBindBufferPageCommitmentARB;
00116 PFNGLBUFFERSTORAGEPROC glBindBufferStorage;
00117 PFNGLBUFFERSUBDATAPROC glBindBufferSubData;
00118 PFNGLCALLCOMMANDLISTNVPROC glBindCommandListNV;
00119 PFNGLCHECKFRAMEBUFFERSTATUSPROC glBindCheckFramebufferStatus;
00120 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glBindCheckNamedFramebufferStatusEXT;
00121 PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glBindCheckNamedFramebufferStatus;
00122 PFNGLCLAMPCOLORPROC glBindColor;
00123 PFNGLCLEARBUFFERDATAPROC glBindClearBufferData;
00124 PFNGLCLEARBUFFERFIPROC glBindClearBufferfi;
00125 PFNGLCLEARBUFFERFVPROC glBindClearBufferfv;
00126 PFNGLCLEARBUFFERIVPROC glBindClearBufferiv;
00127 PFNGLCLEARBUFFERSUBDATAPROC glBindClearBufferSubData;
00128 PFNGLCLEARBUFFERUIVPROC glBindClearBufferuiv;
00129 PFNGLCLEARCOLORPROC glBindClearColor;
00130 PFNGLCLEARDEPTHFPROC glBindClearDepthf;
00131 PFNGLCLEARDEPTHPROC glBindClearDepth;
00132 PFNGLCLEARNAMEDBUFFERDATAEXTPROC glBindClearNamedBufferDataEXT;
00133 PFNGLCLEARNAMEDBUFFERDATAPROC glBindClearNamedBufferData;
00134 PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glBindClearNamedBufferSubDataEXT;
00135 PFNGLCLEARNAMEDBUFFERSUBDATAPROC glBindClearNamedBufferSubData;
00136 PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glBindClearNamedFramebufferfi;
00137 PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glBindClearNamedFramebufferfv;
00138 PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glBindClearNamedFramebufferiv;
00139 PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glBindClearNamedFramebufferuiv;
00140 PFNGLCLEARPROC glBindClear;
00141 PFNGLCLEARSTENCILPROC glBindClearStencil;
00142 PFNGLCLEARTEXIMAGEPROC glBindClearTexImage;
00143 PFNGLCLEARTEXSUBIMAGEPROC glBindClearTexSubImage;
00144 PFNGLCLIENTATTRIBDEFAULTTEXTPROC glBindClientAttribDefaultEXT;
00145 PFNGLCLIENTWAITSYNCPROC glBindClientWaitSync;
00146 PFNGLCLIPCONTROLPROC glBindClipControl;
00147 PFNGLCOLORFORMATNVPROC glBindColorFormatNV;
00148 PFNGLCOLORMASKIPROC glBindColorMaski;
00149 PFNGLCOLORMASKPROC glBindColorMask;
00150 PFNGLCOMMANDLISTSEGMENTSNVPROC glBindCommandListSegmentsNV;

```

```
00151 PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
00152 PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
00153 PFNGLCOMPILESHADERPROC glCompileShader;
00154 PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
00155 PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
00156 PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
00157 PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
00158 PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
00159 PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
00160 PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
00161 PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
00162 PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
00163 PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
00164 PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
00165 PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
00166 PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
00167 PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
00168 PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
00169 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
00170 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
00171 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00172 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00173 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00174 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00175 PFNGLCONSERVATIVEASTERPARAMETERFNVPROC glConservativeRasterParameterfNV;
00176 PFNGLCONSERVATIVEASTERPARAMETERINVPROC glConservativeRasterParameteriNV;
00177 PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
00178 PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
00179 PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
00180 PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
00181 PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
00182 PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
00183 PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
00184 PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
00185 PFNGLCOPYPATHNVPROC glCopyPathNV;
00186 PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
00187 PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
00188 PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
00189 PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
00190 PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
00191 PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
00192 PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
00193 PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
00194 PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
00195 PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
00196 PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
00197 PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
00198 PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
00199 PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationNV;
00200 PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTableNV;
00201 PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancedNV;
00202 PFNGLCOVERFILLPATHNVPROC glCoverFillPathNV;
00203 PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancedNV;
00204 PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathNV;
00205 PFNGLCREATEBUFFERSPROC glCreateBuffers;
00206 PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
00207 PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
00208 PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
00209 PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
00210 PFNGLCREATEPROGRAMPROC glCreateProgram;
00211 PFNGLCREATEQUERIESPROC glCreateQueries;
00212 PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
00213 PFNGLCREATESAMPLESPROC glCreateSamplers;
00214 PFNGLCREATESHADERPROC glCreateShader;
00215 PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
00216 PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
00217 PFNGLCREATESTATESNVPROC glCreateStatesNV;
00218 PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCleventARB;
00219 PFNGLCREATETEXTURESPROC glCreateTextures;
00220 PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
00221 PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
00222 PFNGLCULLFACEPROC glCullFace;
00223 PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
00224 PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
00225 PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
00226 PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
00227 PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
00228 PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
00229 PFNGLDELETEBUFFERSPROC glDeleteBuffers;
00230 PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
00231 PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
00232 PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
00233 PFNGLDELETEPATHSNVPROC glDeletePathsNV;
00234 PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
00235 PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
00236 PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
00237 PFNGLDELETEPROGRAMPROC glDeleteProgram;
```

```

00238 PFNGLDELETEQUERIESPROC glDeleteQueries;
00239 PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
00240 PFNGLDELETESAMPLESPROC glDeleteSamplers;
00241 PFNGLDELETESHADERPROC glDeleteShader;
00242 PFNGLDELETESTATESNVPROC glDeleteStatesNV;
00243 PFNGLDELETESYNCPROC glDeleteSync;
00244 PFNGLDELETETEXTURESPROC glDeleteTextures;
00245 PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
00246 PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
00247 PFNGLDEPTHFUNCPROC glDepthFunc;
00248 PFNGLDEPTHMASKPROC glDepthMask;
00249 PFNGLDEPTHRANGEARRAYPROC glDepthRangeArrayv;
00250 PFNGLDEPTHRANGEFPROC glDepthRangef;
00251 PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
00252 PFNGLDEPTHRANGEPROC glDepthRange;
00253 PFNGLDETACHSHADERPROC glDetachShader;
00254 PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
00255 PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
00256 PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
00257 PFNGLDISABLEIPROC glDisablei;
00258 PFNGLDISABLEPROC glDisable;
00259 PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexAttribArrayAttribEXT;
00260 PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexAttribArrayAttrib;
00261 PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexAttribArrayEXT;
00262 PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArray;
00263 PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
00264 PFNGLDISPATCHCOMPUTEDIRECTPROC glDispatchComputeIndirect;
00265 PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
00266 PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
00267 PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
00268 PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
00269 PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
00270 PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
00271 PFNGLDRAWARRAYSPROC glDrawArrays;
00272 PFNGLDRAWBUFFERPROC glDrawBuffer;
00273 PFNGLDRAWBUFFERSPROC glDrawBuffers;
00274 PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
00275 PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
00276 PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
00277 PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
00278 PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
00279 PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
00280 PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
00281 PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
00282 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC glDrawElementsInstancedBaseVertexBaseInstance;
00283 PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
00284 PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
00285 PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
00286 PFNGLDRAWELEMENTSPROC glDrawElements;
00287 PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00288 PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00289 PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
00290 PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
00291 PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
00292 PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00293 PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00294 PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00295 PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00296 PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00297 PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00298 PFNGLENABLEIPROC glEnablei;
00299 PFNGLENABLEPROC glEnable;
00300 PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexAttribArrayAttribEXT;
00301 PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00302 PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00303 PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArray;
00304 PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
00305 PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
00306 PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
00307 PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
00308 PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
00309 PFNGLENDQUERYPROC glEndQuery;
00310 PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
00311 PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
00312 PFNGLFENCESYNCNCPROC glFenceSync;
00313 PFNGLFINISHPROC glFinish;
00314 PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00315 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00316 PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
00317 PFNGLFLUSHPROC glFlush;
00318 PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00319 PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00320 PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00321 PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00322 PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00323 PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00324 PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;

```

```
00325 PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC glFramebufferSampleLocationsfvARB;
00326 PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNPROC glFramebufferSampleLocationsfvNV;
00327 PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00328 PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00329 PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00330 PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
00331 PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00332 PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
00333 PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00334 PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00335 PFNGLFRAMEBUFFERTEXTUREPROCP glFramebufferTexture;
00336 PFNGLFRONTFACEPROC glFrontFace;
00337 PFNGLGENBUFFERSPROC glGenBuffers;
00338 PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
00339 PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
00340 PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
00341 PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
00342 PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
00343 PFNGLGENPATHSNVPROC glGenPathsNV;
00344 PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
00345 PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
00346 PFNGLGENQUERIESPROC glGenQueries;
00347 PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
00348 PFNGLGENSAMPLERSPROC glGenSamplers;
00349 PFNGLGENTEXTURESPROC glGenTextures;
00350 PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
00351 PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
00352 PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
00353 PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
00354 PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
00355 PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
00356 PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
00357 PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
00358 PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
00359 PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
00360 PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
00361 PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
00362 PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
00363 PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
00364 PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
00365 PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
00366 PFNGLGETBOOLEANVPROC glGetBooleanv;
00367 PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00368 PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00369 PFNGLGETBUFFERPARAMETERUI64NVPROC glGetBufferParameterui64vNV;
00370 PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00371 PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00372 PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00373 PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
00374 PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
00375 PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
00376 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00377 PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00378 PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00379 PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00380 PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00381 PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00382 PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
00383 PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00384 PFNGLGETDOUBLEVPROC glGetDoublev;
00385 PFNGLGETERRORPROC glGetError;
00386 PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
00387 PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
00388 PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
00389 PFNGLGETFLOATI_VPROC glGetFloati_v;
00390 PFNGLGETFLOATVPROC glGetFloatv;
00391 PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
00392 PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
00393 PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
00394 PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
00395 PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
00396 PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
00397 PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
00398 PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
00399 PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
00400 PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
00401 PFNGLGETINTEGER64VPROC glGetInteger64v;
00402 PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00403 PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00404 PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00405 PFNGLGETINTEGERUI64NVPROC glGetIntegerui64vNV;
00406 PFNGLGETINTEGERVPROC glGetIntegerv;
00407 PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
00408 PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
00409 PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
00410 PFNGLGETMULTISAMPLELEVPROC glGetMultisamplefv;
00411 PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
```

```

00412 PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
00413 PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGenfvEXT;
00414 PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
00415 PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
00416 PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
00417 PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
00418 PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
00419 PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
00420 PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIivEXT;
00421 PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
00422 PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
00423 PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
00424 PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
00425 PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
00426 PFNGLGETNAMEDBUFFERPARAMETERUI64NVPROC glGetNamedBufferParameterui64vNV;
00427 PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
00428 PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
00429 PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
00430 PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
00431 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC glGetNamedFramebufferAttachmentParameterivEXT;
00432 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
00433 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
00434 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
00435 PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
00436 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00437 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00438 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIivEXT;
00439 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00440 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00441 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
00442 PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00443 PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00444 PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00445 PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
00446 PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
00447 PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00448 PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00449 PFNGLGETNTEXIMAGEPROC glGetnTexImage;
00450 PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
00451 PFNGLGETNUNIFORMDVPROC glGetnUniformdv;
00452 PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
00453 PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00454 PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00455 PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
00456 PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
00457 PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00458 PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
00459 PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
00460 PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
00461 PFNGLGETOBJECTLABELPROC glGetObjectLabel;
00462 PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
00463 PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
00464 PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
00465 PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
00466 PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
00467 PFNGLGETPATHMETRICRANGEVPROC glGetPathMetricRangeNV;
00468 PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
00469 PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
00470 PFNGLGETPATHPARAMETERIVVNVPROC glGetPathParameterivNV;
00471 PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
00472 PFNGLGETPERFMONITORCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
00473 PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
00474 PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
00475 PFNGLGETPERFMONITORSAMDPROC glGetPerfMonitorCountersAMD;
00476 PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
00477 PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
00478 PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
00479 PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
00480 PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
00481 PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
00482 PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
00483 PFNGLGETPOINTERI_VEXTPROC glGetPointeri_vEXT;
00484 PFNGLGETPOINTERVPROC glGetPointerv;
00485 PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
00486 PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
00487 PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
00488 PFNGLGETPROGRAMIVPROC glGetProgramiv;
00489 PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
00490 PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
00491 PFNGLGETPROGRAMRESOURCEFVNVPROC glGetProgramResourcefvNV;
00492 PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00493 PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
00494 PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00495 PFNGLGETPROGRAMRESOURCELOCATIONPROC glGetProgramResourceLocation;
00496 PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00497 PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00498 PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;

```

```
00499 PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00500 PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00501 PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00502 PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00503 PFNGLGETQUERYIVPROC glGetQueryiv;
00504 PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
00505 PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
00506 PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00507 PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00508 PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00509 PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00510 PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00511 PFNGLGETSAMPLERPARAMETERUIUVPROC glGetSamplerParameterIuiv;
00512 PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
00513 PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
00514 PFNGLGETSHADERIVPROC glGetShaderiv;
00515 PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
00516 PFNGLGETSHADERSOURCEPROC glGetShaderSource;
00517 PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
00518 PFNGLGETSTRINGIPROC glGetStringi;
00519 PFNGLGETSTRINGPROC glGetString;
00520 PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
00521 PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
00522 PFNGLGETSYNCIVPROC glGetSynciv;
00523 PFNGLGETTEXIMAGEPROC glGetTexImage;
00524 PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00525 PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00526 PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00527 PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00528 PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00529 PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
00530 PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00531 PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
00532 PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00533 PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00534 PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00535 PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
00536 PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
00537 PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
00538 PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
00539 PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
00540 PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
00541 PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
00542 PFNGLGETTEXTUREPARAMETERUIVEXTPROC glGetTextureParameterIuivEXT;
00543 PFNGLGETTEXTUREPARAMETERUIUVPROC glGetTextureParameterIuiv;
00544 PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
00545 PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00546 PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00547 PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
00548 PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00549 PFNGLGETTRANSFORMFEEDBACKI64VPROC glGetTransformFeedbacki64_v;
00550 PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00551 PFNGLGETTRANSFORMFEEDBACKL1VPROC glGetTransformFeedbacki_v;
00552 PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00553 PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00554 PFNGLGETUNIFORMDMDVPROC glGetUniformmdv;
00555 PFNGLGETUNIFORMFVPROC glGetUniformfv;
00556 PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00557 PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
00558 PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
00559 PFNGLGETUNIFORMIVPROC glGetUniformiv;
00560 PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
00561 PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
00562 PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
00563 PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
00564 PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
00565 PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00566 PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
00567 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00568 PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00569 PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00570 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00571 PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00572 PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribbdv;
00573 PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribbfv;
00574 PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribIiv;
00575 PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribIuiv;
00576 PFNGLGETVERTEXATTRIBIVPROC glGetVertexAttribIiv;
00577 PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLdv;
00578 PFNGLGETVERTEXATTRIBL164VNVPROC glGetVertexAttribL164vNV;
00579 PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribLui64vARB;
00580 PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribLui64vNV;
00581 PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribPointerv;
00582 PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrNV;
00583 PFNGLHINTPROC glHint;
00584 PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00585 PFNGLINSETEVENTMARKEREXTPROC glInsertEventMarkerEXT;
```

```
00586 PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00587 PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
00588 PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
00589 PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
00590 PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
00591 PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
00592 PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
00593 PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00594 PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00595 PFNGLISBUFFERPROC glIsBuffer;
00596 PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
00597 PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
00598 PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00599 PFNGLISENABLEDIPROC glIsEnabledi;
00600 PFNGLISENABLEDPROC glIsEnabled;
00601 PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00602 PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
00603 PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
00604 PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00605 PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00606 PFNGLISPATHNVPROC glIsPathNV;
00607 PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00608 PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00609 PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00610 PFNGLISPROGRAMPROC glIsProgram;
00611 PFNGLISQUERYPROC glIsQuery;
00612 PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00613 PFNGLISSAMPLERPROC glIsSampler;
00614 PFNGLISSHADERPROC glIsShader;
00615 PFNGLISSTATEENVPROC glIsStateNV;
00616 PFNGLISSYNCPROC glIsSync;
00617 PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00618 PFNGLISTTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00619 PFNGLISTTEXTUREPROC glIsTexture;
00620 PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00621 PFNGLISVERTEXARRAYPROC glIsVertexArray;
00622 PFNGLLABELOBJECTEXTPROC glLabelObjectEXT;
00623 PFNGLLINEWIDTHPROC glLineWidth;
00624 PFNGLLINKPROGRAMPROC glLinkProgram;
00625 PFNGLLISTDRAWCOMMANDSSTATESCCLIENTNVPROC glListDrawCommandsStatesClientNV;
00626 PFNGLLOGICOPPROC glLogicOp;
00627 PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
00628 PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
00629 PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
00630 PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
00631 PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
00632 PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
00633 PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
00634 PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
00635 PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
00636 PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
00637 PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
00638 PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
00639 PFNGLMAPBUFFERPROC glMapBuffer;
00640 PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
00641 PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
00642 PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
00643 PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
00644 PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
00645 PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00646 PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
00647 PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
00648 PFNGLMATRIXLOADDEXTPROC glMatrixLoadDEXT;
00649 PFNGLMATRIXLOADFEXTPROC glMatrixLoadFEXT;
00650 PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00651 PFNGLMATRIXLOADTRANPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
00652 PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedEXT;
00653 PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
00654 PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
00655 PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
00656 PFNGLMATRIXMULTDEXTPROC glMatrixMultDEXT;
00657 PFNGLMATRIXMULTFEXTPROC glMatrixMultFEXT;
00658 PFNGLMATRIXMULTTRANSPPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
00659 PFNGLMATRIXMULTTRANPOSEDEXTPROC glMatrixMultTransposedEXT;
00660 PFNGLMATRIXMULTTRANPOSEFEXTPROC glMatrixMultTransposefEXT;
00661 PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
00662 PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
00663 PFNGLMATRIXPUSHEXTPROC glMatrixPushEXT;
00664 PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedEXT;
00665 PFNGLMATRIXROTATEFEXTPROC glMatrixRotatefEXT;
00666 PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
00667 PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
00668 PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
00669 PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
00670 PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
00671 PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
00672 PFNGLMEMORYBARRIERPROC glMemoryBarrier;
```

```
00673 PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
00674 PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
00675 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
00676 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
00677 PFNGLMULTIDRAWARRAYSINDIRECTCOUNTRARBPROC glMultiDrawArraysIndirectCountARB;
00678 PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00679 PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
00680 PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00681 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00682 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00683 PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTRARBPROC glMultiDrawElementsIndirectCountARB;
00684 PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00685 PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00686 PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00687 PFNGLMULTITEXCOORDPOINTEREEXTPROC glMultiTexCoordPointerEXT;
00688 PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00689 PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00690 PFNGLMULTITEXENVVEXTPROC glMultiTexEnvvEXT;
00691 PFNGLMULTITEXENVVIVEXTPROC glMultiTexEnvivEXT;
00692 PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00693 PFNGLMULTITEXGENDVEXTPROC glMultiTexGendvEXT;
00694 PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
00695 PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
00696 PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
00697 PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
00698 PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
00699 PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
00700 PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
00701 PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
00702 PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
00703 PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameterIEXT;
00704 PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
00705 PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
00706 PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
00707 PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
00708 PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
00709 PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
00710 PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
00711 PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
00712 PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
00713 PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
00714 PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
00715 PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
00716 PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
00717 PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubData;
00718 PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
00719 PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
00720 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
00721 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
00722 PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
00723 PFNGLNAMEDFRAMEBUFFERPARAMETERIIPROC glNamedFramebufferParameteri;
00724 PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
00725 PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00726 PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00727 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glNamedFramebufferSampleLocationsfvARB;
00728 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFNVNPROC glNamedFramebufferSampleLocationsfvNV;
00729 PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00730 PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00731 PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00732 PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00733 PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00734 PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
00735 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00736 PFNGLNAMEDFRAMEBUFFERTEXTUREREPROC glNamedFramebufferTexture;
00737 PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00738 PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00739 PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
00740 PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00741 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
00742 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00743 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00744 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uivEXT;
00745 PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00746 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
00747 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
00748 PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00749 PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00750 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00751 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00752 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00753 PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00754 PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
00755 PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00756 PFNGLOBJECTLABELPROC glObjectLabel;
00757 PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
00758 PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
```

```

00759 PFNGLPATCHPARAMETERIPROC glPatchParameteri;
00760 PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00761 PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00762 PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00763 PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00764 PFNGLPATHGLYPHINDEXEXARRAYNVPROC glPathGlyphIndexArrayNV;
00765 PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
00766 PFNGLPATHGLYPHRANGENVPROC glPathGlyphRangeNV;
00767 PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
00768 PFNGLPATHMEMORYGLYPHINDEXEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
00769 PFNGLPATHPARAMETERRFNVPROC glPathParameterfvNV;
00770 PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00771 PFNGLPATHPARAMETERINVNVPROC glPathParameterivNV;
00772 PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
00773 PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
00774 PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00775 PFNGLPATHSTRINGNVPROC glPathStringNV;
00776 PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00777 PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00778 PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00779 PFNGLPIXELSTOREFPROC glPixelStoref;
00780 PFNGLPIXELSTOREIPROC glPixelStorei;
00781 PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
00782 PFNGLPOINTPARAMETERFPROC glPointParameterf;
00783 PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
00784 PFNGLPOINTPARAMETERIPROC glPointParameteri;
00785 PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00786 PFNGLPOINTSIZEPROC glPointSize;
00787 PFNGLPOLYGONMODEPROC glPolygonMode;
00788 PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00789 PFNGLPOLYGONOFFSETPROC glPolygonOffset;
00790 PFNGLPOPDEBUGGROUPPROC glPopDebugGroup;
00791 PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00792 PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
00793 PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
00794 PFNGLPROGRAMBINARYPROC glProgramBinary;
00795 PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
00796 PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
00797 PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
00798 PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
00799 PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
00800 PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
00801 PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dv;
00802 PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
00803 PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
00804 PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
00805 PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
00806 PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
00807 PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;
00808 PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
00809 PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
00810 PFNGLPROGRAMUNIFORM1TEXTPROC glProgramUniform1iEXT;
00811 PFNGLPROGRAMUNIFORM1IIPROC glProgramUniform1i;
00812 PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
00813 PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
00814 PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
00815 PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniform1ui64NV;
00816 PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
00817 PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
00818 PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
00819 PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
00820 PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
00821 PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
00822 PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
00823 PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
00824 PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
00825 PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
00826 PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
00827 PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
00828 PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
00829 PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
00830 PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
00831 PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
00832 PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
00833 PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
00834 PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
00835 PFNGLPROGRAMUNIFORM2IIPROC glProgramUniform2i;
00836 PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
00837 PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
00838 PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
00839 PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
00840 PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
00841 PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
00842 PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
00843 PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
00844 PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
00845 PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;

```

```
00846 PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
00847 PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
00848 PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dVEXT;
00849 PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dV;
00850 PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
00851 PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
00852 PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
00853 PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
00854 PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
00855 PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
00856 PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
00857 PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
00858 PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
00859 PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
00860 PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
00861 PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
00862 PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
00863 PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
00864 PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
00865 PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
00866 PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
00867 PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
00868 PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
00869 PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
00870 PFNGLPROGRAMUNIFORM4DDEXTPROC glProgramUniform4dEXT;
00871 PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
00872 PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
00873 PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
00874 PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
00875 PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
00876 PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
00877 PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
00878 PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
00879 PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
00880 PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
00881 PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
00882 PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
00883 PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
00884 PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
00885 PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
00886 PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
00887 PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
00888 PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
00889 PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
00890 PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
00891 PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
00892 PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
00893 PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
00894 PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
00895 PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
00896 PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
00897 PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
00898 PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dEXT;
00899 PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2d;
00900 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
00901 PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
00902 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dVEXT;
00903 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dV;
00904 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00905 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
00906 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
00907 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
00908 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
00909 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
00910 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dVEXT;
00911 PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dV;
00912 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
00913 PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
00914 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dVEXT;
00915 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dV;
00916 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
00917 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
00918 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
00919 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
00920 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
00921 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
00922 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dEXT;
00923 PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dV;
00924 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
00925 PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
00926 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dVEXT;
00927 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dV;
00928 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
00929 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
00930 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dVEXT;
00931 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dV;
00932 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
```

```
00933 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
00934 PFNGLPROGRAMUNIFORMI164NVPROC glProgramUniformi164NV;
00935 PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
00936 PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
00937 PFNGLPUSHCLIENTATTRIBDEFAULTTEXTPROC glPushClientAttribDefaultEXT;
00938 PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
00939 PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
00940 PFNGLQUERYCOUNTERPROC glQueryCounter;
00941 PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
00942 PFNGLREADBUFFERPROC glReadBuffer;
00943 PFNGLREADNPPIXELSARBPROC glReadnPixelsARB;
00944 PFNGLREADNPPIXELSPROC glReadnPixels;
00945 PFNGLREADPIXELSPROC glReadPixels;
00946 PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
00947 PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
00948 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
00949 PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
00950 PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
00951 PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
00952 PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
00953 PFNGLSAMPLEMASKIPROC glSampleMaski;
00954 PFNGLSAMPLERPARAMETERFPROC glSamplerParameterf;
00955 PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
00956 PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIiv;
00957 PFNGLSAMPLERPARAMETERIIPROC glSamplerParameteri;
00958 PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
00959 PFNGLSAMPLERPARAMETERIVPROC glSamplerParameteriv;
00960 PFNGLSCISSORARRAYVPROC glScissorArray;
00961 PFNGLSCISSORINDEXEDPROC glScissorIndexed;
00962 PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
00963 PFNGLSCISSORPROC glScissor;
00964 PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
00965 PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
00966 PFNGLSHADERBINARYPROC glShaderBinary;
00967 PFNGLSHADERSOURCEPROC glShaderSource;
00968 PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
00969 PFNGLSIGNALKVFENCENVPROC glSignalVkFenceNV;
00970 PFNGLSIGNALKVSEMAPHORENVPROC glSignalVkSemaphoreNV;
00971 PFNGLSPECIALESHADERARBPROC glSpecializeShaderARB;
00972 PFNGLSTATECAPTURENVPROC glStateCaptureNV;
00973 PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
00974 PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
00975 PFNGLSTENCILFUNCPROC glStencilFunc;
00976 PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
00977 PFNGLSTENCILMASKPROC glStencilMask;
00978 PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
00979 PFNGLSTENCILLOPPROC glStencilOp;
00980 PFNGLSTENCILLOPSEPARATEPROC glStencilOpSeparate;
00981 PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
00982 PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
00983 PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
00984 PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
00985 PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
00986 PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
00987 PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
00988 PFNGLTEXBUFFERARBPROC glTexBufferARB;
00989 PFNGLTEXBUFFERPROC glTexBuffer;
00990 PFNGLTEXBUFFERRANGEPROC glTexBufferSize;
00991 PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
00992 PFNGLTEXIMAGE1DPROC glTexImage1D;
00993 PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
00994 PFNGLTEXIMAGE2DPROC glTexImage2D;
00995 PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
00996 PFNGLTEXIMAGE3DPROC glTexImage3D;
00997 PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
00998 PFNGLTEXPARAMETERFPROC glTexParameterf;
00999 PFNGLTEXPARAMETERFVPROC glTexParameterfv;
01000 PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
01001 PFNGLTEXPARAMETERIIPROC glTexParameterIi;
01002 PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01003 PFNGLTEXPARAMETERIVPROC glTexParameteriv;
01004 PFNGLTEXTSTORAGE1DPROC glTextStorage1D;
01005 PFNGLTEXTSTORAGE2DMULTISAMPLEPROC glTextStorage2DMultisample;
01006 PFNGLTEXTSTORAGE2DPROC glTextStorage2D;
01007 PFNGLTEXTSTORAGE3DMULTISAMPLEPROC glTextStorage3DMultisample;
01008 PFNGLTEXTSTORAGE3DPROC glTextStorage3D;
01009 PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01010 PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01011 PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01012 PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
01013 PFNGLTEXTUREBARRIERPROC glTextureBarrier;
01014 PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
01015 PFNGLTEXTUREBUFFERPROC glTextureBuffer;
01016 PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01017 PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01018 PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
01019 PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
```

```
01020 PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
01021 PFNGLTEXTUREPAGECOMMITMENTTEXTPROC glTexturePageCommitmentEXT;
01022 PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
01023 PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
01024 PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
01025 PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
01026 PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
01027 PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIiivEXT;
01028 PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiiv;
01029 PFNGLTEXTUREPARAMETERIPROC glTextureParameteri;
01030 PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01031 PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01032 PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterivEXT;
01033 PFNGLTEXTUREPARAMETERIVPROC glTextureParameteriv;
01034 PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
01035 PFNGLTEXTURESTORAGE1DEXTPROC glTextureStorage1DEXT;
01036 PFNGLTEXTURESTORAGE1DPROC glTextureStorage1D;
01037 PFNGLTEXTURESTORAGE2DEXTPROC glTextureStorage2DEXT;
01038 PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
01039 PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
01040 PFNGLTEXTURESTORAGE2DPROC glTextureStorage2D;
01041 PFNGLTEXTURESTORAGE3DEXTPROC glTextureStorage3DEXT;
01042 PFNGLTEXTURESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
01043 PFNGLTEXTURESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
01044 PFNGLTEXTURESTORAGE3DPROC glTextureStorage3D;
01045 PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
01046 PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
01047 PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
01048 PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
01049 PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
01050 PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
01051 PFNGLTEXTUREVIEWPROC glTextureView;
01052 PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
01053 PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
01054 PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
01055 PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
01056 PFNGLUNIFORM1DPROC glUniform1d;
01057 PFNGLUNIFORM1DVPROC glUniform1dv;
01058 PFNGLUNIFORM1FPROC glUniform1f;
01059 PFNGLUNIFORM1FVPROC glUniform1fv;
01060 PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
01061 PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
01062 PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
01063 PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
01064 PFNGLUNIFORM1IPROC glUniform1i;
01065 PFNGLUNIFORM1IVPROC glUniform1iv;
01066 PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01067 PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01068 PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
01069 PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
01070 PFNGLUNIFORM1UIPROC glUniform1ui;
01071 PFNGLUNIFORM1UIVPROC glUniform1uiv;
01072 PFNGLUNIFORM2DPROC glUniform2d;
01073 PFNGLUNIFORM2DVPROC glUniform2dv;
01074 PFNGLUNIFORM2FPROC glUniform2f;
01075 PFNGLUNIFORM2FVPROC glUniform2fv;
01076 PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
01077 PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
01078 PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
01079 PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
01080 PFNGLUNIFORM2IPROC glUniform2i;
01081 PFNGLUNIFORM2IVPROC glUniform2iv;
01082 PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01083 PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01084 PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
01085 PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
01086 PFNGLUNIFORM2UIPROC glUniform2ui;
01087 PFNGLUNIFORM2UIVPROC glUniform2uiv;
01088 PFNGLUNIFORM3DPROC glUniform3d;
01089 PFNGLUNIFORM3DVPROC glUniform3dv;
01090 PFNGLUNIFORM3FPROC glUniform3f;
01091 PFNGLUNIFORM3FVPROC glUniform3fv;
01092 PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
01093 PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
01094 PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
01095 PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
01096 PFNGLUNIFORM3IPROC glUniform3i;
01097 PFNGLUNIFORM3IVPROC glUniform3iv;
01098 PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01099 PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01100 PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
01101 PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
01102 PFNGLUNIFORM3UIVPROC glUniform3uiv;
01103 PFNGLUNIFORM3UIVPROC glUniform3uiv;
01104 PFNGLUNIFORM4DPROC glUniform4d;
01105 PFNGLUNIFORM4DVPROC glUniform4dv;
01106 PFNGLUNIFORM4FPROC glUniform4f;
```

```

01107 PFNGLUNIFORM4FVPROC glUniform4fv;
01108 PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
01109 PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
01110 PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
01111 PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
01112 PFNGLUNIFORM4IPROC glUniform4i;
01113 PFNGLUNIFORM4IVPROC glUniform4iv;
01114 PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01115 PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01116 PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
01117 PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
01118 PFNGLUNIFORM4UIPROC glUniform4ui;
01119 PFNGLUNIFORM4UIVPROC glUniform4uiv;
01120 PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01121 PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01122 PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01123 PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
01124 PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
01125 PFNGLUNIFORMMATRIX2DVPROC glUniformMatrix2dv;
01126 PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
01127 PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
01128 PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
01129 PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
01130 PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
01131 PFNGLUNIFORMMATRIX3DVPROC glUniformMatrix3dv;
01132 PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
01133 PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
01134 PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
01135 PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
01136 PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
01137 PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
01138 PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
01139 PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
01140 PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
01141 PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
01142 PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
01143 PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
01144 PFNGLUNIFORMUI64VNVPROC glUniformui64NV;
01145 PFNGLUNIFORMUI64VNVPROC glUniformui64vNV;
01146 PFNGLUNMAPBUFFERPROC glUnmapBuffer;
01147 PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
01148 PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
01149 PFNGLUSEPROGRAMPROC glUseProgram;
01150 PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
01151 PFNGLUSESHADERPROGRAMEXTPROC glUseShaderProgramEXT;
01152 PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
01153 PFNGLVALIDATEPROGRAMPROC glValidateProgram;
01154 PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
01155 PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
01156 PFNGLVERTEXARRAYATTRIBIFORMATPROC glVertexArrayAttribIFormat;
01157 PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribIFormat;
01158 PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01159 PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01160 PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
01161 PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01162 PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01163 PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01164 PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
01165 PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01166 PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
01167 PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01168 PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
01169 PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
01170 PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01171 PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01172 PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01173 PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01174 PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01175 PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01176 PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01177 PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01178 PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
01179 PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01180 PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
01181 PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
01182 PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
01183 PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
01184 PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
01185 PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
01186 PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
01187 PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
01188 PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
01189 PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
01190 PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
01191 PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2s;
01192 PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
01193 PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;

```

```
01194 PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
01195 PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
01196 PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
01197 PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
01198 PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
01199 PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
01200 PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
01201 PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
01202 PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
01203 PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
01204 PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
01205 PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4Nbv;
01206 PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4Niv;
01207 PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4Nsv;
01208 PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4Nub;
01209 PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4Nubv;
01210 PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4Nuiv;
01211 PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4Nusv;
01212 PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
01213 PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
01214 PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
01215 PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
01216 PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
01217 PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01218 PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01219 PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01220 PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01221 PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01222 PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01223 PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
01224 PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
01225 PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
01226 PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01227 PFNGLVERTEXATTRIBI2IVPROC glVertexAttribI2iv;
01228 PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
01229 PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
01230 PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
01231 PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
01232 PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01233 PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
01234 PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01235 PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
01236 PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
01237 PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
01238 PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
01239 PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
01240 PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
01241 PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
01242 PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
01243 PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
01244 PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
01245 PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
01246 PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
01247 PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
01248 PFNGLVERTEXATTRIBL1I64VNPROC glVertexAttribL1i64vNV;
01249 PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
01250 PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
01251 PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
01252 PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribL1ui64vNV;
01253 PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
01254 PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
01255 PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
01256 PFNGLVERTEXATTRIBL2I64VNVPROC glVertexAttribL2i64vNV;
01257 PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
01258 PFNGLVERTEXATTRIBL2UI64VNVPROC glVertexAttribL2ui64vNV;
01259 PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01260 PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
01261 PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
01262 PFNGLVERTEXATTRIBL3I64VNVPROC glVertexAttribL3i64vNV;
01263 PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
01264 PFNGLVERTEXATTRIBL3UI64VNVPROC glVertexAttribL3ui64vNV;
01265 PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01266 PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
01267 PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
01268 PFNGLVERTEXATTRIBL4I64VNVPROC glVertexAttribL4i64vNV;
01269 PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
01270 PFNGLVERTEXATTRIBL4UI64VNVPROC glVertexAttribL4ui64vNV;
01271 PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
01272 PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
01273 PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
01274 PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
01275 PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
01276 PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
01277 PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
01278 PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
01279 PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
01280 PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
```

```

01281 PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
01282 PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
01283 PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
01284 PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
01285 PFNGLVIEWPORTARRAYVPROC glVertexportArrayv;
01286 PFNGLVIEWPORTINDEXEDFPROC glVertexportIndexedf;
01287 PFNGLVIEWPORTINDEXEDFVPROC glVertexportIndexedfv;
01288 PFNGLVIEWPORTPOSITIONWSCALENVPROC glVertexportPositionWScaleNV;
01289 PFNGLVIEWPORTPROC glVertexport;
01290 PFNGLVIEWPORTSWIZZLENVPROC glVertexportSwizzleNV;
01291 PFNGLWAITSYNCPROC glWaitSync;
01292 PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01293 PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01294 PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01295 #endif
01296
01298 01300 GLint gg::ggBufferAlignment(0);
01301
01302 // ゲームグラフィックス特論の都合にもとづく初期化
01303 // 01304 //
01305 void gg::ggInit()
01306 {
01307 // すでにこの関数が実行されていたら以降の処理を行わない
01308 if (ggBufferAlignment) return;
01309
01310 // macOS 以外で OpenGL 3.2 以降の API を取得する
01311 #if !defined(GL3_PROTOTYPES) & !defined(GL_GLES_PROTOTYPES)
01312 glActiveProgramEXT = PFNGLACTIVEPROGRAMEXTPROC(glfwGetProcAddress("glActiveProgramEXT"));
01313 glActiveShaderProgram = PFNGLACTIVESHADERPROGRAMPROC(glfwGetProcAddress("glActiveShaderProgram"));
01314 glActiveTexture = PFNGLACTIVETEXTUREPROC(glfwGetProcAddress("glActiveTexture"));
01315 glApplyFramebufferAttachmentCMAAINTEL =
01316 PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC(glfwGetProcAddress("glApplyFramebufferAttachmentCMAAINTEL"));
01317 glAttachShader = PFNGLATTACHSHADERPROC(glfwGetProcAddress("glAttachShader"));
01318 glBeginConditionalRender =
01319 PFNGLBEGINCNDITONALRENDERPROC(glfwGetProcAddress("glBeginConditionalRender"));
01320 glBeginConditionalRenderNV =
01321 PFNGLBEGINCNDITONALRENDERNVPROC(glfwGetProcAddress("glBeginConditionalRenderNV"));
01322 glBeginPerfMonitorAMD = PFNGLBEGINPERFMONITORAMDPROC(glfwGetProcAddress("glBeginPerfMonitorAMD"));
01323 glBeginPerfQueryINTEL = PFNGLBEGINPERFQUERYINTELPROC(glfwGetProcAddress("glBeginPerfQueryINTEL"));
01324 glBeginQuery =
01325 PFNGLBEGINQUERYPROC(glfwGetProcAddress("glBeginQuery"));
01326 glBeginQueryIndexed =
01327 PFNGLBEGINQUERYINDEXEDPROC(glfwGetProcAddress("glBeginQueryIndexed"));
01328 glBeginTransformFeedback =
01329 PFNGLBEGINTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBeginTransformFeedback"));
01330 glBindAttribLocation = PFNGLBINDATTRIBLOCATIONPROC(glfwGetProcAddress("glBindAttribLocation"));
01331 glBindBuffer = PFNGLBINDBUFFERPROC(glfwGetProcAddress("glBindBuffer"));
01332 glBindBufferBase = PFNGLBINDBUFFERBASEPROC(glfwGetProcAddress("glBindBufferBase"));
01333 glBindBufferRange = PFNGLBINDBUFFERRANGEPROC(glfwGetProcAddress("glBindBufferRange"));
01334 glBindBuffersBase = PFNGLBINDBUFFERSBASEPROC(glfwGetProcAddress("glBindBuffersBase"));
01335 glBindBuffersRange = PFNGLBINDBUFFERSRANGEPROC(glfwGetProcAddress("glBindBuffersRange"));
01336 glBindFragDataLocation =
01337 PFNGLBINDFRAGDATALOCATIONPROC(glfwGetProcAddress("glBindFragDataLocation"));
01338 glBindFragDataLocationIndexed =
01339 PFNGLBINDFRAGDATALOCATIONINDEXEDPROC(glfwGetProcAddress("glBindFragDataLocationIndexed"));
01340 glBindFramebuffer = PFNGLBINDFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
01341 glBindImageTexture = PFNGLBINDIMAGETEXTUREPROC(glfwGetProcAddress("glBindImageTexture"));
01342 glBindImageTextures =
01343 PFNGLBINDIMAGETEXTURESPROC(glfwGetProcAddress("glBindImageTextures"));
01344 glBindMultiTextureEXT = PFNGLBINDMULTITEXTUREEXTPROC(glfwGetProcAddress("glBindMultiTextureEXT"));
01345 glBindProgramPipeline = PFNGLBINDPROGRAMPIPELINEPROC(glfwGetProcAddress("glBindProgramPipeline"));
01346 glBindRenderbuffer = PFNGLBINDRENDERBUFFERPROC(glfwGetProcAddress("glBindRenderbuffer"));
01347 glBindSampler = PFNGLBINDSAMPLERPROC(glfwGetProcAddress("glBindSampler"));
01348 glBindSamplers =
01349 PFNGLBINDSAMPLERSPROC(glfwGetProcAddress("glBindSamplers"));
01350 glBindTexture = PFNGLBINDTEXTUREPROC(glfwGetProcAddress("glBindTexture"));
01351 glBindTextureUnit =
01352 PFNGLBINDTEXTUREUNITPROC(glfwGetProcAddress("glBindTextureUnit"));
01353 glBindTextures =
01354 PFNGLBINDTEXTURESPROC(glfwGetProcAddress("glBindTextures"));
01355 glBindTransformFeedback =
01356 PFNGLBINDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBindTransformFeedback"));
01357 glBindVertexArray = PFNGLBINDVERTEXARRAYPROC(glfwGetProcAddress("glBindVertexArray"));
01358 glBindVertexBuffer = PFNGLBINDVERTEXBUFFERPROC(glfwGetProcAddress("glBindVertexBuffer"));
01359 glBindVertexBuffers =
01360 PFNGLBINDVERTEXBUFFERSPROC(glfwGetProcAddress("glBindVertexBuffers"));
01361 glBindBarrierKHR = PFNGLBLENDBARRIERKHRPROC(glfwGetProcAddress("glBindBarrierKHR"));
01362 glBindBarrierNV = PFNGLBLENDBARRIERNVPROC(glfwGetProcAddress("glBindBarrierNV"));
01363 glBindColor = PFNGLBLENDCOLORPROC(glfwGetProcAddress("glBindColor"));
01364 glBindEquation = PFNGLBLENEQUATIONPROC(glfwGetProcAddress("glBindEquation"));
01365 glBindEquationSeparate =
01366 PFNGLBLENEQUATIONSEPARATEPROC(glfwGetProcAddress("glBindEquationSeparate"));
01367 glBindEquationSeparatei =
01368 PFNGLBLENEQUATIONSEPARATEIPROC(glfwGetProcAddress("glBindEquationSeparatei"));
01369 glBindEquationSeparateiARB =
01370 PFNGLBLENEQUATIONSEPARATEIARBPROC(glfwGetProcAddress("glBindEquationSeparateiARB"));
01371 glBindEquationi = PFNGLBLENEQUATIONIPROC(glfwGetProcAddress("glBindEquationi"));
01372 glBindEquationiARB =
01373 PFNGLBLENEQUATIONIARBPROC(glfwGetProcAddress("glBindEquationiARB"));
01374 glBindFunc = PFNGLBLENDFUNCPROC(glfwGetProcAddress("glBindFunc"));
01375 glBindFuncSeparate =
01376 PFNGLBLENDFUNCSEPARATEPROC(glfwGetProcAddress("glBindFuncSeparate"));
01377 glBindFuncSeparatei =
01378 PFNGLBLENDFUNCSEPARATEIPROC(glfwGetProcAddress("glBindFuncSeparatei"));
01379 glBindFuncSeparateiARB =

```

```

01360 PFNGLBLENDFUNCSEPARATEIARBPROC(glfwGetProcAddress("glBlendFuncSeparateiARB"));
01361 g1BlendFunci = PFNGLBLENDFUNCIPROC(glfwGetProcAddress("glBlendFunci"));
01362 g1BlendFunciARB = PFNGLBLENDFUNCIARBPROC(glfwGetProcAddress("glBlendFunciARB"));
01363 g1BlendParameteriNV = PFNGLBLENDPARAMETERINVPROC(glfwGetProcAddress("glBlendParameteriNV"));
01364 g1BlitFramebuffer = PFNGLBLITFRAMEBUFFERPROC(glfwGetProcAddress("glBlitFramebuffer"));
01365 g1BlitNamedFramebuffer =
PFNGLBLITNAMEDFRAMEBUFFERPROC(glfwGetProcAddress("glBlitNamedFramebuffer"));
01366 g1BufferAddressRangeNV =
PFNGLBUFFERADDRESSRANGEVPROC(glfwGetProcAddress("glBufferAddressRangeNV"));
01367 g1BufferData = PFNGLBUFFERDATAPROC(glfwGetProcAddress("glBufferData"));
g1BufferPageCommitmentARB =
PFNGLBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glBufferPageCommitmentARB"));
01368 g1BufferStorage = PFNGLBUFFERSTORAGEPROC(glfwGetProcAddress("glBufferStorage"));
01369 g1BufferSubData = PFNGLBUFFERSUBDATAPROC(glfwGetProcAddress("glBufferSubData"));
01370 g1CallCommandListNV = PFNGLCALLCOMMANDLISTNVPROC(glfwGetProcAddress("glCallCommandListNV"));
01371 g1CheckFramebufferStatus =
PFNGLCHECKFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckFramebufferStatus"));
01372 g1CheckNamedFramebufferStatus =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckNamedFramebufferStatus"));
01373 g1CheckNamedFramebufferStatusEXT =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC(glfwGetProcAddress("glCheckNamedFramebufferStatusEXT"));
01374 g1ClampColor = PFNGLCLAMPCOLORPROC(glfwGetProcAddress("glClampColor"));
01375 g1Clear = PFNGLCLEARPROC(glfwGetProcAddress("glClear"));
01376 g1ClearBufferData = PFNGLCLEARBUFFERDATAPROC(glfwGetProcAddress("glClearBufferData"));
01377 g1ClearBufferSubData = PFNGLCLEARBUFFERSUBDATAPROC(glfwGetProcAddress("glClearBufferSubData"));
01378 g1ClearBufferfi = PFNGLCLEARBUFFERFIPROC(glfwGetProcAddress("glClearBufferfi"));
01379 g1ClearBufferfv = PFNGLCLEARBUFFERFVPROC(glfwGetProcAddress("glClearBufferfv"));
01380 g1ClearBufferiv = PFNGLCLEARBUFFERIVPROC(glfwGetProcAddress("glClearBufferiv"));
01381 g1ClearBufferuiv = PFNGLCLEARBUFFERUIVPROC(glfwGetProcAddress("glClearBufferuiv"));
01382 g1ClearColor = PFNGLCLEARCOLORPROC(glfwGetProcAddress("glClearColor"));
01383 g1ClearDepth = PFNGLCLEARDEPTHPROC(glfwGetProcAddress("glClearDepth"));
01384 g1ClearDepthf = PFNGLCLEARDEPTHFPROC(glfwGetProcAddress("glClearDepthf"));
01385 g1ClearNamedBufferData =
PFNGLCLEARNAMEDBUFFERDATAPROC(glfwGetProcAddress("glClearNamedBufferData"));
01386 g1ClearNamedBufferDataEXT =
PFNGLCLEARNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataEXT"));
01387 g1ClearNamedBufferDataSubData =
PFNGLCLEARNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glClearNamedBufferDataSubData"));
01388 g1ClearNamedBufferDataEXT =
PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataSubDataEXT"));
01389 g1ClearNamedFramebufferfi =
PFNGLCLEARNAMEDFRAMEBUFFERFIPROC(glfwGetProcAddress("glClearNamedFramebufferfi"));
01390 g1ClearNamedFramebufferfv =
PFNGLCLEARNAMEDFRAMEBUFFERFVPROC(glfwGetProcAddress("glClearNamedFramebufferfv"));
01391 g1ClearNamedFramebufferiv =
PFNGLCLEARNAMEDFRAMEBUFFERIVPROC(glfwGetProcAddress("glClearNamedFramebufferiv"));
01392 g1ClearNamedFramebufferuiv =
PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC(glfwGetProcAddress("glClearNamedFramebufferuiv"));
01393 g1ClearStencil = PFNGLCLEARSTENCILPROC(glfwGetProcAddress("glClearStencil"));
01394 g1ClearTexImage = PFNGLCLEARTEXIMAGEPROC(glfwGetProcAddress("glClearTexImage"));
01395 g1ClearTexSubImage = PFNGLCLEARTEXSUBIMAGEPROC(glfwGetProcAddress("glClearTexSubImage"));
01396 g1ClientAttribDefaultEXT =
PFNGLCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glClientAttribDefaultEXT"));
01397 g1ClientWaitSync = PFNGLCLIENTWAITSYNCPROC(glfwGetProcAddress("glClientWaitSync"));
01398 g1ClipControl = PFNGLCLIPCONTROLPROC(glfwGetProcAddress("glClipControl"));
01399 g1ColorFormatNV = PFNGLCOLORFORMATNVPROC(glfwGetProcAddress("glColorFormatNV"));
01400 g1ColorMask = PFNGLCOLORMASKPROC(glfwGetProcAddress("glColorMask"));
01401 g1ColorMaski = PFNGLCOLORMASKIPROC(glfwGetProcAddress("glColorMaski"));
01402 g1CommandListSegmentsNV =
PFNGLCOMMANDLISTSEGMENTSNVPROC(glfwGetProcAddress("glCommandListSegmentsNV"));
01403 g1CompileCommandListNV =
PFNGLCOMPILECOMMANDLISTNVPROC(glfwGetProcAddress("glCompileCommandListNV"));
01404 g1CompileShader = PFNGLCOMPILESHADERPROC(glfwGetProcAddress("glCompileShader"));
01405 g1CompileShaderIncludeARB =
PFNGLCOMPILESHADERINCLUDEARBPROC(glfwGetProcAddress("glCompileShaderIncludeARB"));
01406 g1CompressedMultiTexImage1DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage1DEXT"));
01407 g1CompressedMultiTexImage2DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage2DEXT"));
01408 g1CompressedMultiTexImage3DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage3DEXT"));
01409 g1CompressedMultiTexSubImage1DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage1DEXT"));
01410 g1CompressedMultiTexSubImage2DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage2DEXT"));
01411 g1CompressedMultiTexSubImage3DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage3DEXT"));
01412 g1CompressedTexImage1D =
PFNGLCOMPRESSEDTEXIMAGE1DPROC(glfwGetProcAddress("glCompressedTexImage1D"));
01413 g1CompressedTexImage2D =
PFNGLCOMPRESSEDTEXIMAGE2DPROC(glfwGetProcAddress("glCompressedTexImage2D"));
01414 g1CompressedTexImage3D =
PFNGLCOMPRESSEDTEXIMAGE3DPROC(glfwGetProcAddress("glCompressedTexImage3D"));
01415 g1CompressedTexSubImage1D =
PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTexSubImage1D"));
01416 g1CompressedTexSubImage2D =
PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTexSubImage2D"));

```

```

01417 glCompressedTexSubImage3D =
01418 PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTexSubImage3D"));
01419 glCompressedTextureImage1DEXT =
01420 PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureImage1DEXT"));
01421 glCompressedTextureImage2DEXT =
01422 PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureImage2DEXT"));
01423 glCompressedTextureImage3DEXT =
01424 PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureImage3DEXT"));
01425 glCompressedTextureSubImage1D =
01426 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTextureSubImage1D"));
01427 glCompressedTextureSubImage1DEXT =
01428 PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage1DEXT"));
01429 glCompressedTextureSubImage2D =
01430 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTextureSubImage2D"));
01431 glCompressedTextureSubImage2DEXT =
01432 PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage2DEXT"));
01433 glCompressedTextureSubImage3D =
01434 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTextureSubImage3D"));
01435 glCompressedTextureSubImage3DEXT =
01436 PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage3DEXT"));
01437 glConservativeRasterParameteriNV =
01438 PFNGLCONSERVATIVERASTERPARAMETERINVPROC(glfwGetProcAddress("glConservativeRasterParameteriNV"));
01439 glCopyBufferData = PFNGLCOPYBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyBufferData"));
01440 glCopyImageSubData = PFNGLCOPYIMAGESUBDATAPROC(glfwGetProcAddress("glCopyImageSubData"));
01441 glCopyMultiTexImage1DEXT =
01442 PFNGLCOPYMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage1DEXT"));
01443 glCopyMultiTexImage2DEXT =
01444 PFNGLCOPYMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage2DEXT"));
01445 glCopyMultiTexSubImage1DEXT =
01446 PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage1DEXT"));
01447 glCopyMultiTexSubImage2DEXT =
01448 PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage2DEXT"));
01449 glCopyMultiTexSubImage3DEXT =
01450 PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage3DEXT"));
01451 glCopyNamedBufferData =
01452 PFNGLCOPYNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyNamedBufferData"));
01453 glCopyPathNV = PFNGLCOPYPATHNVPROC(glfwGetProcAddress("glCopyPathNV"));
01454 glCopyTexImage1D = PFNGLCOPYTEXIMAGE1DPROC(glfwGetProcAddress("glCopyTexImage1D"));
01455 glCopyTexImage2D = PFNGLCOPYTEXIMAGE2DPROC(glfwGetProcAddress("glCopyTexImage2D"));
01456 glCopyTexImage2DEXT =
01457 PFNGLCOPYTEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTexImage2DEXT"));
01458 glCopyTexSubImage1D =
01459 PFNGLCOPYTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCopyTexSubImage1D"));
01460 glCopyTexSubImage1DEXT =
01461 PFNGLCOPYTEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTexSubImage1DEXT"));
01462 glCopyTexSubImage2D =
01463 PFNGLCOPYTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCopyTexSubImage2D"));
01464 glCopyTexSubImage2DEXT =
01465 PFNGLCOPYTEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTexSubImage2DEXT"));
01466 glCopyTexSubImage3D =
01467 PFNGLCOPYTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCopyTexSubImage3D"));
01468 glCopyTextureImage1DEXT =
01469 PFNGLCOPYTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureImage1DEXT"));
01470 glCopyTextureImage2DEXT =
01471 PFNGLCOPYTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureImage2DEXT"));
01472 glCopyTextureSubImage1D =
01473 PFNGLCOPYTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCopyTextureSubImage1D"));
01474 glCopyTextureSubImage1DEXT =
01475 PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage1DEXT"));
01476 glCopyTextureSubImage2D =
01477 PFNGLCOPYTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCopyTextureSubImage2D"));
01478 glCopyTextureSubImage2DEXT =
01479 PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage2DEXT"));
01480 glCopyTextureSubImage3D =
01481 PFNGLCOPYTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCopyTextureSubImage3D"));
01482 glCopyTextureSubImage3DEXT =
01483 PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage3DEXT"));
01484 glCoverFillPathInstancedNV =
01485 PFNGLCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverFillPathInstancedNV"));
01486 glCoverFillPathNV = PFNGLCOVERFILLPATHNVPROC(glfwGetProcAddress("glCoverFillPathNV"));
01487 glCoverStrokePathInstancedNV =
01488 PFNGLCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverStrokePathInstancedNV"));
01489 glCoverStrokePathNV = PFNGLCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glCoverStrokePathNV"));
01490 glCoverageModulationNV =
01491 PFNGLCOVERAGEMODULATIONNVPROC(glfwGetProcAddress("glCoverageModulationNV"));
01492 glCoverageModulationTableNV =
01493 PFNGLCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glCoverageModulationTableNV"));
01494 glCreateBuffers = PFNGLCREATEBUFFERSPROC(glfwGetProcAddress("glCreateBuffers"));
01495 glCreateCommandListsNV =
01496 PFNGLCREATECOMMANDLISTSNVPROC(glfwGetProcAddress("glCreateCommandListsNV"));
01497 glCreateFramebuffers = PFNGLCREATEFRAMEBUFFERSPROC(glfwGetProcAddress("glCreateFramebuffers"));
01498 glCreatePerfQueryINTEL =
01499 PFNGLCREATEPERFQUERYINTELPROC(glfwGetProcAddress("glCreatePerfQueryINTEL"));
01500 glCreateProgram = PFNGLCREATEPROGRAMPROC(glfwGetProcAddress("glCreateProgram"));
01501 glCreateProgramPipelines =
01502 PFNGLCREATEPROGRAMPIPELINESPROC(glfwGetProcAddress("glCreateProgramPipelines"));
01503 glCreateQueries = PFNGLCREATEQUERIESPROC(glfwGetProcAddress("glCreateQueries"));
01504 glCreateRenderbuffers = PFNGLCREATERENDERBUFFERSPROC(glfwGetProcAddress("glCreateRenderbuffers"));
01505 glCreateSamplers = PFNGLCREATESAMPLERSPROC(glfwGetProcAddress("glCreateSamplers"));
01506 glCreateShader = PFNGLCREATESHADERPROC(glfwGetProcAddress("glCreateShader"));
01507 glCreateShaderProgramEXT =
01508 PFNGLCREATESHADERPROGRAMEXTPROC(glfwGetProcAddress("glCreateShaderProgramEXT"));
01509 glCreateShaderProgramv =
01510 PFNGLCREATESHADERPROGRAMVPROC(glfwGetProcAddress("glCreateShaderProgramv"));

```

```

01469 glCreateStatesNV = PFNGLCREATESTATESNVPROC(glfwGetProcAddress("glCreateStatesNV"));
01470 glCreateSyncFromCLeventARB =
PFNGLCREATESYNCFROMCLEVENTARBPROC(glfwGetProcAddress("glCreateSyncFromCLeventARB"));
01471 glCreateTextures = PFNGLCREATETEXTURESPROC(glfwGetProcAddress("glCreateTextures"));
01472 glCreateTransformFeedbacks =
PFNGLCREATETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glCreateTransformFeedbacks"));
01473 glCreateVertexArrays = PFNGLCREATEVERTEXARRAYSPROC(glfwGetProcAddress("glCreateVertexArrays"));
01474 glCullFace = PFNGLCULLFACEPROC(glfwGetProcAddress("glCullFace"));
01475 glDebugMessageCallback =
PFNGLDEBUGMESSAGECALLBACKPROC(glfwGetProcAddress("glDebugMessageCallback"));
01476 glDebugMessageCallbackARB =
PFNGLDEBUGMESSAGECALLBACKARBPROC(glfwGetProcAddress("glDebugMessageCallbackARB"));
01477 glDebugMessageControl = PFNGLDEBUGMESSAGECONTROLPROC(glfwGetProcAddress("glDebugMessageControl"));
01478 glDebugMessageControlARB =
PFNGLDEBUGMESSAGECONTROLARBPROC(glfwGetProcAddress("glDebugMessageControlARB"));
01479 glDebugMessageInsert = PFNGLDEBUGMESSAGEINSERTPROC(glfwGetProcAddress("glDebugMessageInsert"));
01480 glDebugMessageInsertARB =
PFNGLDEBUGMESSAGEINSERTARBPROC(glfwGetProcAddress("glDebugMessageInsertARB"));
01481 glDeleteBuffers = PFNGLDELETEBUFFERSPROC(glfwGetProcAddress("glDeleteBuffers"));
01482 glDeleteCommandListsNV =
PFNGLDELETECOMMANDLISTSNVPROC(glfwGetProcAddress("glDeleteCommandListsNV"));
01483 glDeleteFramebuffers = PFNGLDELETEFRAMEBUFFERSPROC(glfwGetProcAddress("glDeleteFramebuffers"));
01484 glDeleteNamedStringARB =
PFNGLDELETENAMEDSTRINGARBPROC(glfwGetProcAddress("glDeleteNamedStringARB"));
01485 glDeletePathsNV = PFNGLDELETEPATHSNVPROC(glfwGetProcAddress("glDeletePathsNV"));
01486 glDeletePerfMonitorsAMD =
PFNGLDELETEPERFMONITORSAMDPROC(glfwGetProcAddress("glDeletePerfMonitorsAMD"));
01487 glDeletePerfQueryINTEL =
PFNGLDELETEPERFQUERYINTELPROC(glfwGetProcAddress("glDeletePerfQueryINTEL"));
01488 glDeleteProgram = PFNGLDELETEPROGRAMPROC(glfwGetProcAddress("glDeleteProgram"));
01489 glDeleteProgramPipelines =
PFNGLDELETEPROGRAMPIPELINESPROC(glfwGetProcAddress("glDeleteProgramPipelines"));
01490 glDeleteQueries = PFNGLDELETEQUERIESPROC(glfwGetProcAddress("glDeleteQueries"));
01491 glDeleteRenderbuffers = PFNGLDELETERENDERBUFFERSPROC(glfwGetProcAddress("glDeleteRenderbuffers"));
01492 glDeleteSamplers = PFNGLDELETESAMPLERSPROC(glfwGetProcAddress("glDeleteSamplers"));
01493 glDeleteShader = PFNGLDELETESHADERPROC(glfwGetProcAddress("glDeleteShader"));
01494 glDeleteStatesNV = PFNGLDELETESTATESNVPROC(glfwGetProcAddress("glDeleteStatesNV"));
01495 glDeleteSync = PFNGLDELETESYNCPROC(glfwGetProcAddress("glDeleteSync"));
01496 glDeleteTextures = PFNGLDELETETEXTURESPROC(glfwGetProcAddress("glDeleteTextures"));
01497 glDeleteTransformFeedbacks =
PFNGLDELETETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glDeleteTransformFeedbacks"));
01498 glDeleteVertexArrays = PFNGLDELETEVERTEXARRAYSPROC(glfwGetProcAddress("glDeleteVertexArrays"));
01499 glDepthFunc = PFNGLDEPTHFUNCPROC(glfwGetProcAddress("glDepthFunc"));
01500 glDepthMask = PFNGLDEPTHMASKPROC(glfwGetProcAddress("glDepthMask"));
01501 glDepthRange = PFNGLDEPTHRANGEPROC(glfwGetProcAddress("glDepthRange"));
01502 glDepthRangeArrayv = PFNGLDEPTHRANGEARRAYVPROC(glfwGetProcAddress("glDepthRangeArrayv"));
01503 glDepthRangeIndexed = PFNGLDEPTHRANGEINDEXEDPROC(glfwGetProcAddress("glDepthRangeIndexed"));
01504 glDepthRangef = PFNGLDEPTHRANGEFPROC(glfwGetProcAddress("glDepthRangef"));
01505 glDetachShader = PFNGLDETACHSHADERPROC(glfwGetProcAddress("glDetachShader"));
01506 glDisable = PFNGLDISABLEPROC(glfwGetProcAddress("glDisable"));
01507 glDisableClientStateIndexedEXT =
PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glDisableClientStateIndexedEXT"));
01508 glDisableClientStateiEXT =
PFNGLDISABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glDisableClientStateiEXT"));
01509 glDisableIndexedEXT = PFNGLDISABLEINDEXEDEXTPROC(glfwGetProcAddress("glDisableIndexedEXT"));
01510 glDisableVertexAttribArray =
PFNGLDISABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glDisableVertexAttribArray"));
01511 glDisableVertexAttribArrayAttribEXT =
PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttribEXT"));
01512 glDisableVertexAttribArrayEXT =
PFNGLDISABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glDisableVertexAttribArrayEXT"));
01513 glDisableVertexAttribArrayAttrib =
PFNGLDISABLEVERTEXATTRIBARRAYPROC(glfwGetProcAddress("glDisableVertexAttribArrayAttrib"));
01514 glDisablei = PFNGLDISABLEIPROC(glfwGetProcAddress("glDisablei"));
01515 glDispatchCompute = PFNGLDISPATCHCOMPUTEPROC(glfwGetProcAddress("glDispatchCompute"));
01516 glDispatchComputeGroupSizeARB =
PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC(glfwGetProcAddress("glDispatchComputeGroupSizeARB"));
01517 glDispatchComputeIndirect =
PFNGLDISPATCHCOMPUTEINDIRECTPROC(glfwGetProcAddress("glDispatchComputeIndirect"));
01518 glDrawArrays = PFNGLDRAWARRAYSPROC(glfwGetProcAddress("glDrawArrays"));
01519 glDrawArraysIndirect = PFNGLDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glDrawArraysIndirect"));
01520 glDrawArraysInstanced = PFNGLDRAWARRAYSINSTANCEDPROC(glfwGetProcAddress("glDrawArraysInstanced"));
01521 glDrawArraysInstancedARB =
PFNGLDRAWARRAYSINSTANCEDARBPROC(glfwGetProcAddress("glDrawArraysInstancedARB"));
01522 glDrawArraysInstancedBaseInstance =
PFNGLDRAWARRAYSINSTANCEPROC(glfwGetProcAddress("glDrawArraysInstancedBaseInstance"));
01523 glDrawArraysInstancedEXT =
PFNGLDRAWARRAYSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawArraysInstancedEXT"));
01524 glDrawBuffer = PFNGLDRAWBUFFERPROC(glfwGetProcAddress("glDrawBuffer"));
01525 glDrawBuffers = PFNGLDRAWBUFFERSPROC(glfwGetProcAddress("glDrawBuffers"));
01526 glDrawCommandsAddressNV =
PFNGLDRAWCOMMANDSADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsAddressNV"));
01527 glDrawCommandsNV = PFNGLDRAWCOMMANDSNVPROC(glfwGetProcAddress("glDrawCommandsNV"));
01528 glDrawCommandsStatesAddressNV =
PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsStatesAddressNV"));
01529 glDrawCommandsStatesNV =
PFNGLDRAWCOMMANDSSTATESNVPROC(glfwGetProcAddress("glDrawCommandsStatesNV"));

```

```

01530     glDrawElements = PFNGLDRAWELEMENTSPROC(glfwGetProcAddress("glDrawElements"));
01531     glDrawElementsBaseVertex =
01532     PFNGLDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsBaseVertex"));
01533     glDrawElementsIndirect =
01534     PFNGLDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glDrawElementsIndirect"));
01535     glDrawElementsInstanced =
01536     PFNGLDRAWELEMENTSINSTANCEDPROC(glfwGetProcAddress("glDrawElementsInstanced"));
01537     glDrawElementsInstancedARB =
01538     PFNGLDRAWELEMENTSINSTANCEDARBPROC(glfwGetProcAddress("glDrawElementsInstancedARB"));
01539     glDrawElementsInstancedBaseInstance =
01540     PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseInstance"));
01541     glDrawElementsInstancedBaseVertex =
01542     PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertex"));
01543     glDrawElementsInstancedBaseVertexBaseInstance =
01544     PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertexBaseInstance"));
01545     glDrawElementsInstancedEXT =
01546     PFNGLDRAWELEMENTSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawElementsInstancedEXT"));
01547     glDrawRangeElements = PFNGLDRAWRANGEELEMENTSPROC(glfwGetProcAddress("glDrawRangeElements"));
01548     glDrawRangeElementsBaseVertex =
01549     PFNGLDRAWRANGELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawRangeElementsBaseVertex"));
01550     glDrawTransformFeedback =
01551     PFNGLDRAWTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glDrawTransformFeedback"));
01552     glDrawTransformFeedbackInstanced =
01553     PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackInstanced"));
01554     glDrawTransformFeedbackStream =
01555     PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC(glfwGetProcAddress("glDrawTransformFeedbackStream"));
01556     glDrawTransformFeedbackStreamInstanced =
01557     PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackStreamInstanced"));
01558     glDrawVkImageNV = PFNGLDRAWVKIMAGENVPROC(glfwGetProcAddress("glDrawVkImageNV"));
01559     glEdgeFlagFormatNV = PFNGLEDGEFLAGFORMATNVPROC(glfwGetProcAddress("glEdgeFlagFormatNV"));
01560     glEnable = PFNGLENABLEPROC(glfwGetProcAddress("glEnable"));
01561     glEnableClientStateIndexedEXT =
01562     PFNGLENABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glEnableClientStateIndexedEXT"));
01563     glEnableClientStateiEXT =
01564     PFNGLENABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glEnableClientStateiEXT"));
01565     glEnableIndexedEXT = PFNGLENABLEINDEXEDEXTPROC(glfwGetProcAddress("glEnableIndexedEXT"));
01566     glEnableVertexAttribArrayAttrib =
01567     PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttrib"));
01568     glEnableVertexAttribArrayAttribEXT =
01569     PFNGLENABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttribEXT"));
01570     glEnableVertexAttribArrayEXT =
01571     PFNGLENABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glEnableVertexAttribArrayEXT"));
01572     glEnableVertexAttribArrayAttribArray =
01573     PFNGLENABLEVERTEXATTRIBARRAYPROC(glfwGetProcAddress("glEnableVertexAttribArrayAttribArray"));
01574     glEnablei = PFNGLENABLEIIPROC(glfwGetProcAddress("glEnablei"));
01575     glEndConditionalRender =
01576     PFNGLENDCONDITIONALRENDERPROC(glfwGetProcAddress("glEndConditionalRender"));
01577     glEndConditionalRenderNV =
01578     PFNGLENDCONDITIONALRENDERNVPROC(glfwGetProcAddress("glEndConditionalRenderNV"));
01579     glEndPerfMonitorAMD = PFNGLENDPERFMONITORAMDPROC(glfwGetProcAddress("glEndPerfMonitorAMD"));
01580     glEndPerfQueryINTEL = PFNGLENDPERFQUERYINTELPROC(glfwGetProcAddress("glEndPerfQueryINTEL"));
01581     glEndQuery = PFNGLENDQUERYPROC(glfwGetProcAddress("glEndQuery"));
01582     glEndQueryIndexed = PFNGLENDQUERYINDEXEDPROC(glfwGetProcAddress("glEndQueryIndexed"));
01583     glEndTransformFeedback =
01584     PFNGLENDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glEndTransformFeedback"));
01585     glEvaluateDepthValuesARB =
01586     PFNGLEVALUATEDEPTHVALUESARBPROC(glfwGetProcAddress("glEvaluateDepthValuesARB"));
01587     glFenceSync = PFNGLFENCESYNCPROC(glfwGetProcAddress("glFenceSync"));
01588     glFinish = PFNGLFINISHPROC(glfwGetProcAddress("glFinish"));
01589     glFlush = PFNGLFLUSHPROC(glfwGetProcAddress("glFlush"));
01590     glFlushMappedBufferRange =
01591     PFNGLFLUSHMAPPEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedBufferRange"));
01592     glFlushMappedNamedBufferRange =
01593     PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedNamedBufferRange"));
01594     glFlushMappedNamedBufferRangeEXT =
01595     PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glFlushMappedNamedBufferRangeEXT"));
01596     glFogCoordFormatNV = PFNGLFOGOORDFORMATNVPROC(glfwGetProcAddress("glFogCoordFormatNV"));
01597     glFragmentCoverageColorNV =
01598     PFNGLFRAGMENTCOVERAGECOLORNVPROC(glfwGetProcAddress("glFragmentCoverageColorNV"));
01599     glFramebufferDrawBufferEXT =
01600     PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC(glfwGetProcAddress("glFramebufferDrawBufferEXT"));
01601     glFramebufferDrawBuffersEXT =
01602     PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC(glfwGetProcAddress("glFramebufferDrawBuffersEXT"));
01603     glFramebufferParameteri =
01604     PFNGLFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glFramebufferParameteri"));
01605     glFramebufferReadBufferEXT =
01606     PFNGLFRAMEBUFFERREADBUFFEREXTPROC(glfwGetProcAddress("glFramebufferReadBufferEXT"));
01607     glFramebufferRenderbuffer =
01608     PFNGLFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glFramebufferRenderbuffer"));
01609     glFramebufferSampleLocationsfvARB =
01610     PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvARB"));
01611     glFramebufferSampleLocationsfvNV =
01612     PFNGLFRAMEBUFFERSAMPLELOCATIONSFNVPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvNV"));
01613     glFramebufferTexture = PFNGLFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glFramebufferTexture"));
01614     glFramebufferTexture1D =
01615     PFNGLFRAMEBUFFERTEXTURE1DPROC(glfwGetProcAddress("glFramebufferTexture1D"));
01616     glFramebufferTexture2D =

```

```

01582 PFNGLFRAMEBUFFERTEXTURE2DPROC glfwGetProcAddress("glFramebufferTexture2D"));
01583     glFramebufferTexture3D =
01584 PFNGLFRAMEBUFFERTEXTURE3DPROC glfwGetProcAddress("glFramebufferTexture3D"));
01585     glFramebufferTextureARB =
01586 PFNGLFRAMEBUFFERTEXTUREARBPROC glfwGetProcAddress("glFramebufferTextureARB"));
01587     glFramebufferTextureFaceARB =
01588 PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glfwGetProcAddress("glFramebufferTextureFaceARB"));
01589     glFramebufferTextureLayer =
01590 PFNGLFRAMEBUFFERTEXTURELAYERPROC glfwGetProcAddress("glFramebufferTextureLayer"));
01591     glFramebufferTextureLayerARB =
01592 PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glfwGetProcAddress("glFramebufferTextureLayerARB"));
01593     glFramebufferTextureMultiviewOVR =
01594 PFNGLFRAMEBUFFERTEXTUREMULTIVIEWPROC glfwGetProcAddress("glFramebufferTextureMultiviewOVR"));
01595     glFrontFace = PFNGLFRONTPFACEPROC glfwGetProcAddress("glFrontFace"));
01596     glGenBuffers = PFNGLGENBUFFERSPROC glfwGetProcAddress("glGenBuffers"));
01597     glGenFramebuffers = PFNGLGENFRAMEBUFFERSPROC glfwGetProcAddress("glGenFramebuffers"));
01598     glGenPathsNV = PFNGLGENPATHSNVPROC glfwGetProcAddress("glGenPathsNV"));
01599     glGenPerfMonitorsAMD = PFNGLGENPERFMONITORSAMDPROC glfwGetProcAddress("glGenPerfMonitorsAMD"));
01600     glGenProgramPipelines = PFNGLGENPROGRAMPIPELINESPROC glfwGetProcAddress("glGenProgramPipelines"));
01601     glGenQueries = PFNGLGENQUERIESPROC glfwGetProcAddress("glGenQueries"));
01602     glGenRenderbuffers = PFNGLGENRENDERBUFFERSPROC glfwGetProcAddress("glGenRenderbuffers"));
01603     glGenSamplers = PFNGLGENSAMPLERSPROC glfwGetProcAddress("glGenSamplers"));
01604     glGenTextures = PFNGLGENTEXTURESPROC glfwGetProcAddress("glGenTextures"));
01605     glGenTransformFeedbacks =
01606 PFNGLGENTRANSFORMFEEDBACKSPROC glfwGetProcAddress("glGenTransformFeedbacks"));
01607     glGenVertexArrays = PFNGLGENVERTEXARRAYSPROC glfwGetProcAddress("glGenVertexArrays"));
01608     glGenerateMipmap = PFNGLGENERATEMIPMAPPROC glfwGetProcAddress("glGenerateMipmap"));
01609     glGenerateMultiTexMipmapEXT =
01610 PFNGLGENERATEMULTITEXMIPMAPEXTPROC glfwGetProcAddress("glGenerateMultiTexMipmapEXT"));
01611     glGenerateTextureMipmap =
01612 PFNGLGENERATETEXTUREMIPMAPPROC glfwGetProcAddress("glGenerateTextureMipmap"));
01613     glGenerateTextureMipmapEXT =
01614 PFNGLGENERATETEXTUREMIPMAPEXTPROC glfwGetProcAddress("glGenerateTextureMipmapEXT"));
01615     glGetActiveAtomicCounterBufferiv =
01616 PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glfwGetProcAddress("glGetActiveAtomicCounterBufferiv"));
01617     glGetActiveAttrib = PFNGLGETACTIVEATTRIBPROC glfwGetProcAddress("glGetActiveAttrib"));
01618     glGetActiveSubroutineName =
01619 PFNGLGETACTIVESUBROUTINENAMEPROC glfwGetProcAddress("glGetActiveSubroutineName"));
01620     glGetActiveSubroutineUniformName =
01621 PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glfwGetProcAddress("glGetActiveSubroutineUniformName"));
01622     glGetActiveSubroutineUniformiv =
01623 PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glfwGetProcAddress("glGetActiveSubroutineUniformiv"));
01624     glGetActiveUniform = PFNGLGETACTIVEUNIFORMPROC glfwGetProcAddress("glGetActiveUniform"));
01625     glGetActiveUniformBlockName =
01626 PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glfwGetProcAddress("glGetActiveUniformBlockName"));
01627     glGetActiveUniformBlockiv =
01628 PFNGLGETACTIVEUNIFORMBLOCKIVPROC glfwGetProcAddress("glGetActiveUniformBlockiv"));
01629     glGetActiveUniformName =
01630 PFNGLGETACTIVEUNIFORMNAMEPROC glfwGetProcAddress("glGetActiveUniformName"));
01631     glGetActiveUniformsiv = PFNGLGETACTIVEUNIFORMSIVPROC glfwGetProcAddress("glGetActiveUniformsiv"));
01632     glGetAttachedShaders = PFNGLGETATTACHEDSHADERSPROC glfwGetProcAddress("glGetAttachedShaders"));
01633     glGetAttribLocation = PFNGLGETATTRIBLOCATIONPROC glfwGetProcAddress("glGetAttribLocation"));
01634     glGetBooleanIndexedvEXT =
01635 PFNGLGETBOOLEANINDEXEDVEXTPROC glfwGetProcAddress("glGetBooleanIndexedvEXT"));
01636     glGetBooleani_v = PFNGLGETBOOLEANI_VPROC glfwGetProcAddress("glGetBooleani_v"));
01637     glGetBooleanv = PFNGLGETBOOLEANVPROC glfwGetProcAddress("glGetBooleanv"));
01638     glGetBufferParameteri64v =
01639 PFNGLGETBUFFERPARAMETERI64VPROC glfwGetProcAddress("glGetBufferParameteri64v"));
01640     glGetBufferParameteriv =
01641 PFNGLGETBUFFERPARAMETERIVPROC glfwGetProcAddress("glGetBufferParameteriv"));
01642     glGetBufferParameterui64vNV =
01643 PFNGLGETBUFFERPARAMETERUI64NVPROC glfwGetProcAddress("glGetBufferParameterui64vNV"));
01644     glGetBufferPointerv = PFNGLGETBUFFERPOINTERVPROC glfwGetProcAddress("glGetBufferPointerv"));
01645     glGetBufferSubData = PFNGLGETBUFFERSUBDATAPROC glfwGetProcAddress("glGetBufferSubData"));
01646     glGetCommandHeaderNV = PFNGLGETCOMMANDHEADERNVPROC glfwGetProcAddress("glGetCommandHeaderNV"));
01647     glGetCompressedMultiTexImageEXT =
01648 PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glfwGetProcAddress("glGetCompressedMultiTexImageEXT"));
01649     glGetCompressedTexImage =
01650 PFNGLGETCOMPRESSEDTEXIMAGEPROC glfwGetProcAddress("glGetCompressedTexImage"));
01651     glGetCompressedTextureImage =
01652 PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glfwGetProcAddress("glGetCompressedTextureImage"));
01653     glGetCompressedTextureImageEXT =
01654 PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glfwGetProcAddress("glGetCompressedTextureImageEXT"));
01655     glGetCompressedTextureSubImage =
01656 PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glfwGetProcAddress("glGetCompressedTextureSubImage"));
01657     glGetCoverageModulationTableNV =
01658 PFNGLGETCOVERAGEMODULATIONTABLENVPROC glfwGetProcAddress("glGetCoverageModulationTableNV"));
01659     glGetDebugMessageLog = PFNGLGETDEBUGMESSAGELOGPROC glfwGetProcAddress("glGetDebugMessageLog"));
01660     glGetDebugMessageLogARB =
01661 PFNGLGETDEBUGMESSAGELOGARBPROC glfwGetProcAddress("glGetDebugMessageLogARB"));
01662     glGetDoubleIndexedvEXT =
01663 PFNGLGETDOUBLEINDEXEDVEXTPROC glfwGetProcAddress("glGetDoubleIndexedvEXT"));
01664     glGetDoublei_v = PFNGLGETDOUBLEI_VPROC glfwGetProcAddress("glGetDoublei_v"));
01665     glGetDoublei_vEXT = PFNGLGETDOUBLEI_VEXTPROC glfwGetProcAddress("glGetDoublei_vEXT"));
01666     glGetDoublev = PFNGLGETDOUBLEVPROC glfwGetProcAddress("glGetDoublev"));
01667     glGetError = PFNGLGETERRORPROC glfwGetProcAddress("glGetError"));
01668     glGetFirstPerfQueryIdINTEL =

```

```

01639 PFNGLGETFIRSTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetFirstPerfQueryIdINTEL"));
01640 glGetFloatIndexedvEXT = PFNGLGETFLOATINDEXEDVEXTPROC(glfwGetProcAddress("glGetFloatIndexedvEXT"));
01641 glGetFloati_v = PFNGLGETFLOATI_VPROC(glfwGetProcAddress("glGetFloati_v"));
01642 glGetFloati_vEXT = PFNGLGETFLOATI_VEXTPROC(glfwGetProcAddress("glGetFloati_vEXT"));
01643 glGetFragDataIndex = PFNGLGETFRAGDATAINDEXPROC(glfwGetProcAddress("glGetFragDataIndex"));
01644 glGetFragDataLocation = PFNGLGETFRAGDATALOCATIONPROC(glfwGetProcAddress("glGetFragDataLocation"));
01645 glGetFramebufferAttachmentParameteriv =
01646 PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferAttachmentParameteriv"));
01647 glGetFramebufferParameteriv =
01648 PFNGLGETFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferParameteriv"));
01649 glGetGraphicsResetStatus =
01650 PFNGLGETGRAPHICSRESETSTATUSPROC(glfwGetProcAddress("glGetGraphicsResetStatus"));
01651 glGetGraphicsResetStatusARB =
01652 PFNGLGETGRAPHICSRESETSTATUSARBPROC(glfwGetProcAddress("glGetGraphicsResetStatusARB"));
01653 glGetImageHandleARB = PFNGLGETIMAGEHANDLEARBPROC(glfwGetProcAddress("glGetImageHandleARB"));
01654 glGetImageHandleNV = PFNGLGETIMAGEHANDLENVPROC(glfwGetProcAddress("glGetImageHandleNV"));
01655 glGetInteger64i_v = PFNGLGETINTEGER64I_VPROC(glfwGetProcAddress("glGetInteger64i_v"));
01656 glGetInteger64v = PFNGLGETINTEGER64VPROC(glfwGetProcAddress("glGetInteger64v"));
01657 glGetIntegerIndexedvEXT =
01658 PFNGLGETINTEGERINDEXEDVEXTPROC(glfwGetProcAddress("glGetIntegerIndexedvEXT"));
01659 glGetIntegeri_v = PFNGLGETINTEGERI_VPROC(glfwGetProcAddress("glGetIntegeri_v"));
01660 glGetIntegerui64i_vNV = PFNGLGETINTEGERUI64I_VNVPROC(glfwGetProcAddress("glGetIntegerui64i_vNV"));
01661 glGetIntegerui64vNV = PFNGLGETINTEGERUI64VNVPROC(glfwGetProcAddress("glGetIntegerui64vNV"));
01662 glGetIntegerv = PFNGLGETINTEGERVPROC(glfwGetProcAddress("glGetIntegerv"));
01663 glGetInternalformatSampleivNV =
01664 PFNGLGETINTERNALFORMATSAMPLEIVNVPROC(glfwGetProcAddress("glGetInternalformatSampleivNV"));
01665 glGetInternalformati64v =
01666 PFNGLGETINTERNALFORMATI64VPROC(glfwGetProcAddress("glGetInternalformati64v"));
01667 glGetInternalformativ = PFNGLGETINTERNALFORMATIVPROC(glfwGetProcAddress("glGetInternalformativ"));
01668 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01669 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01670 glGetMultiTexEnvvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvvEXT"));
01671 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01672 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01673 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01674 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01675 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01676 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01677 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01678 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01679 glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01680 glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01681 glGetNamedBufferData =
01682 PFNGLGETNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glGetNamedBufferData"));
01683 glGetNamedBufferDataEXT =
01684 PFNGLGETNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glGetNamedBufferDataEXT"));
01685 glGetNamedFramebufferAttachmentParameteriv =
01686 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameteriv"));
01687 glGetNamedFramebufferAttachmentParameterivEXT =
01688 PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameterivEXT"));
01689 glGetNamedFramebufferParameteriv =
01690 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriv"));
01691 glGetNamedFramebufferParameterivEXT =
01692 PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterivEXT"));
01693 glGetNamedProgramLocalParameteriIivEXT =
01694 PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameteriIivEXT"));
01695 glGetNamedProgramLocalParameteriIuivEXT =
01696 PFNGLGETNAMEDPROGRAMLOCALPARAMETERUIIEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameteriIuivEXT"));
01697 glGetNamedProgramLocalParameterdvEXT =
01698 PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterdvEXT"));
01699 glGetNamedProgramLocalParameterfvEXT =
01700 PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterfvEXT"));
01701 glGetNamedProgramStringEXT =
01702 PFNGLGETNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glGetNamedProgramStringEXT"));
01703 glGetNamedProgramivEXT =
01704 PFNGLGETNAMEDPROGRAMIVEXTPROC(glfwGetProcAddress("glGetNamedProgramivEXT"));

```

```

01693     glGetNamedRenderbufferParameteriv =
01694     PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedRenderbufferParameteriv"));
01695     glGetNamedRenderbufferParameterivEXT =
01696     PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedRenderbufferParameterivEXT"));
01697     glGetStringARB = PFNGLGETNAMEDSTRINGARBPROC(glfwGetProcAddress("glGetStringARB"));
01698     glGetStringivARB = PFNGLGETNAMEDSTRINGIVARBPROC(glfwGetProcAddress("glGetStringivARB"));
01699     glGetNextPerfQueryIdINTEL =
01700     PFNGLGETNEXTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetNextPerfQueryIdINTEL"));
01701     glGetObjectLabel = PFNGLGETOBJECTLABELPROC(glfwGetProcAddress("glGetObjectLabel"));
01702     glGetObjectLabelEXT = PFNGLGETOBJECTLABELEXTPROC(glfwGetProcAddress("glGetObjectLabelEXT"));
01703     glGetObjectPtrLabel = PFNGLGETOBJECTPTRLABELPROC(glfwGetProcAddress("glGetObjectPtrLabel"));
01704     glGetPathCommandsNV = PFNGLGETPATHCOMMANDSNVPROC(glfwGetProcAddress("glGetPathCommandsNV"));
01705     glGetPathCoordsNV = PFNGLGETPATHCOORDSNVPROC(glfwGetProcAddress("glGetPathCoordsNV"));
01706     glGetPathDashArrayNV = PFNGLGETPATHDASHARRAYNVPROC(glfwGetProcAddress("glGetPathDashArrayNV"));
01707     glGetPathLengthNV = PFNGLGETPATHLENGTHNVPROC(glfwGetProcAddress("glGetPathLengthNV"));
01708     glGetPathMetricRangeNV =
01709     PFNGLGETPATHMETRICRANGEPROC(glfwGetProcAddress("glGetPathMetricRangeNV"));
01710     glGetPathMetricsNV = PFNGLGETPATHMETRICSNVPROC(glfwGetProcAddress("glGetPathMetricsNV"));
01711     glGetPathParameterfvNV =
01712     PFNGLGETPATHPARAMETERFVNPROC(glfwGetProcAddress("glGetPathParameterfvNV"));
01713     glGetPathParameterivNV =
01714     PFNGLGETPATHPARAMETERIVNVPROC(glfwGetProcAddress("glGetPathParameterivNV"));
01715     glGetPathSpacingNV = PFNGLGETPATHSPACINGNVPROC(glfwGetProcAddress("glGetPathSpacingNV"));
01716     glGetPerfCounterInfoINTEL =
01717     PFNGLGETPERFCOUNTERINFOINTELPROC(glfwGetProcAddress("glGetPerfCounterInfoINTEL"));
01718     glGetPerfMonitorCounterDataAMD =
01719     PFNGLGETPERFMONITORCOUNTERDATAAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterDataAMD"));
01720     glGetPerfMonitorCounterInfoAMD =
01721     PFNGLGETPERFMONITORCOUNTERINFOAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterInfoAMD"));
01722     glGetPerfMonitorCounterStringAMD =
01723     PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterStringAMD"));
01724     glGetPerfMonitorCountersAMD =
01725     PFNGLGETPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glGetPerfMonitorCountersAMD"));
01726     glGetPerfMonitorGroupStringAMD =
01727     PFNGLGETPERFMONITORGROUPSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupStringAMD"));
01728     glGetPerfMonitorGroupsAMD =
01729     PFNGLGETPERFMONITORGROUPSAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupsAMD"));
01730     glGetPerfQueryDataINTEL =
01731     PFNGLGETPERFQUERYDATAINTELPROC(glfwGetProcAddress("glGetPerfQueryDataINTEL"));
01732     glGetPerfQueryIdByNameINTEL =
01733     PFNGLGETPERFQUERYIDBYNAMEINTELPROC(glfwGetProcAddress("glGetPerfQueryIdByNameINTEL"));
01734     glGetPerfQueryInfoINTEL =
01735     PFNGLGETPERFQUERYINFOINTELPROC(glfwGetProcAddress("glGetPerfQueryInfoINTEL"));
01736     glGetPointerIndexedvEXT =
01737     PFNGLGETPOINTERINDEXEDVEXTPROC(glfwGetProcAddress("glGetPointerIndexedvEXT"));
01738     glGetPointervi_vEXT = PFNGLGETPOINTERI_VEXTPROC(glfwGetProcAddress("glGetPointervi_vEXT"));
01739     glGetPointerv = PFNGLGETPOINTERVERPROC(glfwGetProcAddress("glGetPointerv"));
01740     glGetProgramBinary = PFNGLGETPROGRAMBINARYPROC(glfwGetProcAddress("glGetProgramBinary"));
01741     glGetProgramInfoLog = PFNGLGETPROGRAMINFOLOGPROC(glfwGetProcAddress("glGetProgramInfoLog"));
01742     glGetProgramInterfaceiv =
01743     PFNGLGETPROGRAMINTERFACEIVPROC(glfwGetProcAddress("glGetProgramInterfaceiv"));
01744     glGetProgramPipelineInfoLog =
01745     PFNGLGETPROGRAMPIPELINEINFOLOGPROC(glfwGetProcAddress("glGetProgramPipelineInfoLog"));
01746     glGetProgramPipelineiv =
01747     PFNGLGETPROGRAMPIPELINEIVPROC(glfwGetProcAddress("glGetProgramPipelineiv"));
01748     glGetProgramResourceIndex =
01749     PFNGLGETPROGRAMRESOURCEINDEXPROC(glfwGetProcAddress("glGetProgramResourceIndex"));
01750     glGetProgramResourceLocation =
01751     PFNGLGETPROGRAMRESOURCELOCATIONPROC(glfwGetProcAddress("glGetProgramResourceLocation"));
01752     glGetProgramResourceLocationIndex =
01753     PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC(glfwGetProcAddress("glGetProgramResourceLocationIndex"));
01754     glGetProgram resourceName =
01755     PFNGLGETPROGRAMRESOURCENAMEPROC(glfwGetProcAddress("glGetProgram resourceName"));
01756     glGetProgramResourcefvNV =
01757     PFNGLGETPROGRAMRESOURCEFVNPROC(glfwGetProcAddress("glGetProgramResourcefvNV"));
01758     glGetProgramResourceiv =
01759     PFNGLGETPROGRAMRESOURCEIVPROC(glfwGetProcAddress("glGetProgramResourceiv"));
01760     glGetProgramStageiv = PFNGLGETPROGRAMSTAGEIVPROC(glfwGetProcAddress("glGetProgramStageiv"));
01761     glGetProgramiv = PFNGLGETPROGRAMIVPROC(glfwGetProcAddress("glGetProgramiv"));
01762     glGetQueryBufferObjecti64v =
01763     PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
01764     glGetQueryBufferObjecti =
01765     PFNGLGETQUERYBUFFEROBJECTIVPROC(glfwGetProcAddress("glGetQueryBufferObjectiv"));
01766     glGetQueryBufferObjectui64v =
01767     PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
01768     glGetQueryBufferObjectiuv =
01769     PFNGLGETQUERYBUFFEROBJECTUIVPROC(glfwGetProcAddress("glGetQueryBufferObjectiuv"));
01770     glGetQueryIndexediv = PFNGLGETQUERYINDEXEDIVPROC(glfwGetProcAddress("glGetQueryIndexediv"));
01771     glGetQueryObjecti64v = PFNGLGETQUERYOBJECTI64VPROC(glfwGetProcAddress("glGetQueryObjecti64v"));
01772     glGetQueryObjectiv = PFNGLGETQUERYOBJECTIVPROC(glfwGetProcAddress("glGetQueryObjectiv"));
01773     glGetQueryObjectui64v = PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
01774     glGetQueryObjectiuv = PFNGLGETQUERYOBJECTIUVPROC(glfwGetProcAddress("glGetQueryObjectiuv"));
01775     glGetQueryiv = PFNGLGETQUERYIVPROC(glfwGetProcAddress("glGetQueryiv"));
01776     glGetRenderbufferParameteriv =
01777     PFNGLGETRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetRenderbufferParameteriv"));
01778     glGetSamplerParameterIiv =
01779     PFNGLGETSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glGetSamplerParameterIiv"));

```

```
01748 glGetSamplerParameterIuiv =
PFNGLGETSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glGetSamplerParameterIuiv"));
01749 glGetSamplerParameterfv =
PFNGLGETSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glGetSamplerParameterfv"));
01750 glGetSamplerParameteriv =
PFNGLGETSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glGetSamplerParameteriv"));
01751 glGetShaderInfoLog = PFNGLGETSHADERINFOLOGPROC(glfwGetProcAddress("glGetShaderInfoLog"));
01752 glGetShaderPrecisionFormat =
PFNGLGETSHADERPRECISIONFORMATPROC(glfwGetProcAddress("glGetShaderPrecisionFormat"));
01753 glGetShaderSource = PFNGLGETSHADERSOURCEPROC(glfwGetProcAddress("glGetShaderSource"));
01754 glGetShaderiv = PFNGLGETSHADERIVPROC(glfwGetProcAddress("glGetShaderiv"));
01755 glGetStageIndexNV = PFNGLGETSTAGEINDEXNVPROC(glfwGetProcAddress("glGetStageIndexNV"));
01756 glGetString = PFNGLGETSTRINGPROC(glfwGetProcAddress("glGetString"));
01757 glGetStringi = PFNGLGETSTRINGIPROC(glfwGetProcAddress("glGetStringi"));
01758 glGetSubroutineIndex = PFNGLGETSUBROUTINEINDEXPROC(glfwGetProcAddress("glGetSubroutineIndex"));
01759 glGetSubroutineUniformLocation =
PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetSubroutineUniformLocation"));
01760 glGetSynciv = PFNGLGETSYNCIVPROC(glfwGetProcAddress("glGetSynciv"));
01761 glGetTexImage = PFNGLGETTEXIMAGEPROC(glfwGetProcAddress("glGetTexImage"));
01762 glGetTexLevelParameterfv =
PFNGLGETTEXLEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTexLevelParameterfv"));
01763 glGetTexLevelParameteriv =
PFNGLGETTEXLEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTexLevelParameteriv"));
01764 glGetTexParameterIiv = PFNGLGETTEXPARAMETERIIVPROC(glfwGetProcAddress("glGetTexParameterIiv"));
01765 glGetTexParameterIuiv = PFNGLGETTEXPARAMETERIUIVPROC(glfwGetProcAddress("glGetTexParameterIuiv"));
01766 glGetTexParameterfv = PFNGLGETTEXPARAMETERFVPROC(glfwGetProcAddress("glGetTexParameterfv"));
01767 glGetTexParameteriv = PFNGLGETTEXPARAMETERIVPROC(glfwGetProcAddress("glGetTexParameteriv"));
01768 glGetTextureHandleARB = PFNGLGETTEXTUREHANDLEARBPROC(glfwGetProcAddress("glGetTextureHandleARB"));
01769 glGetTextureHandleNV = PFNGLGETTEXTUREHANDLENVPROC(glfwGetProcAddress("glGetTextureHandleNV"));
01770 glGetTextureImage = PFNGLGETTEXTUREIMAGEPROC(glfwGetProcAddress("glGetTextureImage"));
01771 glGetTextureImageEXT = PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTextureImageEXT"));
01772 glGetTextureLevelParameterfv =
PFNGLGETTEXTURELEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTextureLevelParameterfv"));
01773 glGetTextureLevelParameterfvEXT =
PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterfvEXT"));
01774 glGetTextureLevelParameteriv =
PFNGLGETTEXTURELEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTextureLevelParameteriv"));
01775 glGetTextureLevelParameterivEXT =
PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterivEXT"));
01776 glGetTextureParameterIiv =
PFNGLGETTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glGetTextureParameterIiv"));
01777 glGetTextureParameterIiivEXT =
PFNGLGETTEXTUREPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIiivEXT"));
01778 glGetTextureParameterIuiv =
PFNGLGETTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glGetTextureParameterIuiv"));
01779 glGetTextureParameterIuivEXT =
PFNGLGETTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIuivEXT"));
01780 glGetTextureParameterfv =
PFNGLGETTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glGetTextureParameterfv"));
01781 glGetTextureParameterfvEXT =
PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
01782 glGetTextureParameteriv =
PFNGLGETTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glGetTextureParameteriv"));
01783 glGetTextureParameterivEXT =
PFNGLGETTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureParameterivEXT"));
01784 glGetTextureSamplerHandleARB =
PFNGLGETTEXTURESAMPLERHANDLEARBPROC(glfwGetProcAddress("glGetTextureSamplerHandleARB"));
01785 glGetTextureSamplerHandleNV =
PFNGLGETTEXTURESAMPLERHANDLENVPROC(glfwGetProcAddress("glGetTextureSamplerHandleNV"));
01786 glGetTextureSubImage = PFNGLGETTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetTextureSubImage"));
01787 glGetTransformFeedbackVarying =
PFNGLGETTRANSFORMFEEDBACKVARYINGPROC(glfwGetProcAddress("glGetTransformFeedbackVarying"));
01788 glGetTransformFeedbacki64_v =
PFNGLGETTRANSFORMFEEDBACKI64_VPROC(glfwGetProcAddress("glGetTransformFeedbacki64_v"));
01789 glGetTransformFeedbacki_v =
PFNGLGETTRANSFORMFEEDBACKI_VPROC(glfwGetProcAddress("glGetTransformFeedbacki_v"));
01790 glGetTransformFeedbackiv =
PFNGLGETTRANSFORMFEEDBACKIVPROC(glfwGetProcAddress("glGetTransformFeedbackiv"));
01791 glGetUniformLocation =
PFNGLGETUNIFORMBLOCKINDEXPROC(glfwGetProcAddress("glGetUniformBlockIndex"));
01792 glGetUniformLocation = PFNGLGETUNIFORMINDICESPROC(glfwGetProcAddress("glGetUniformIndices"));
01793 glGetUniformLocation = PFNGLGETUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetUniformLocation"));
01794 glGetUniformLocationuiiv =
PFNGLGETUNIFORMSUBROUTINEUIVPROC(glfwGetProcAddress("glGetUniformSubroutineuiiv"));
01795 glGetUniformLocationmdv = PFNGLGETUNIFORMMDVPROC(glfwGetProcAddress("glGetUniformmdv"));
01796 glGetUniformLocationmfv = PFNGLGETUNIFORMMFVPROC(glfwGetProcAddress("glGetUniformmfv"));
01797 glGetUniformLocation64vARB = PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64vARB"));
01798 glGetUniformLocation64vNV = PFNGLGETUNIFORMI64VNVPROC(glfwGetProcAddress("glGetUniformi64vNV"));
01799 glGetUniformLocationiv = PFNGLGETUNIFORMIVPROC(glfwGetProcAddress("glGetUniformiv"));
01800 glGetUniformLocationui64vARB = PFNGLGETUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetUniformui64vARB"));
01801 glGetUniformLocationui64vNV = PFNGLGETUNIFORMUI64VNVPROC(glfwGetProcAddress("glGetUniformui64vNV"));
01802 glGetUniformLocationuiiv = PFNGLGETUNIFORMUIVPROC(glfwGetProcAddress("glGetUniformuiiv"));
01803 glVertexArrayIndexed64iv =
PFNGLGETVERTEXARRAYINDEXED64IVPROC(glfwGetProcAddress("glGetVertexArrayIndexed64iv"));
01804 glVertexVertexArrayIndexeddiv =
PFNGLGETVERTEXARRAYINDEXEDIVPROC(glfwGetProcAddress("glGetVertexArrayIndexeddiv"));
01805 glVertexVertexArrayIntegeri_vEXT =

```

```

01806 PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegeri_vEXT"));
01806     glGetVertexArrayIntegervEXT =
01807 PFNGLGETVERTEXARRAYINTEGERVEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegervEXT"));
01807     glGetVertexArrayPointeri_vEXT =
01808 PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayPointeri_vEXT"));
01808     glGetVertexArrayPointervEXT =
01809 PFNGLGETVERTEXARRAYPOINTERVEXTPROC(glfwGetProcAddress("glGetVertexArrayPointervEXT"));
01809     glGetVertexArrayiv = PFNGLGETVERTEXARRAYIVPROC(glfwGetProcAddress("glGetVertexArrayiv"));
01810     glGetVertexAttribIiv = PFNGLGETVERTEXATTRIBIIVPROC(glfwGetProcAddress("glGetVertexAttribIiv"));
01811     glGetVertexAttribIuiv = PFNGLGETVERTEXATTRIBUIVPROC(glfwGetProcAddress("glGetVertexAttribIuiv"));
01812     glGetVertexAttribLdv = PFNGLGETVERTEXATTRIBLDVPROC(glfwGetProcAddress("glGetVertexAttribLdv"));
01813     glGetVertexAttribLi64vNV =
01813 PFNGLGETVERTEXATTRIBLI64VNVPROC(glfwGetProcAddress("glGetVertexAttribLi64vNV"));
01814     glGetVertexAttribLui64vARB =
01814 PFNGLGETVERTEXATTRIBLUI64VARBPROC(glfwGetProcAddress("glGetVertexAttribLui64vARB"));
01815     glGetVertexAttribLui64vNV =
01815 PFNGLGETVERTEXATTRIBLUI64VNVPROC(glfwGetProcAddress("glGetVertexAttribLui64vNV"));
01816     glGetVertexAttribPointerv =
01816 PFNGLGETVERTEXATTRIBPOINTERVPROC(glfwGetProcAddress("glGetVertexAttribPointerv"));
01817     glGetVertexAttribBdv = PFNGLGETVERTEXATTRIBBDVPROC(glfwGetProcAddress("glGetVertexAttribBdv"));
01818     glGetVertexAttribBfv = PFNGLGETVERTEXATTRIBBFVPROC(glfwGetProcAddress("glGetVertexAttribBfv"));
01819     glGetVertexAttribBiv = PFNGLGETVERTEXATTRIBBIVPROC(glfwGetProcAddress("glGetVertexAttribBiv"));
01820     glGetVkProcAddrNV = PFNGLGETVKPROCAADDRNVPROC(glfwGetProcAddress("glGetVkProcAddrNV"));
01821     glGetnCompressedTexImage =
01821 PFNGLGETNCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetnCompressedTexImage"));
01822     glGetnCompressedTexImageARB =
01822 PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC(glfwGetProcAddress("glGetnCompressedTexImageARB"));
01823     glGetnTexImage = PFNGLGETNTEXIMAGEPROC(glfwGetProcAddress("glGetnTexImage"));
01824     glGetnTexImageARB = PFNGLGETNTEXIMAGEARBPROC(glfwGetProcAddress("glGetnTexImageARB"));
01825     glGetnUniformdv = PFNGLGETNUNIFORMDVPROC(glfwGetProcAddress("glGetnUniformdv"));
01826     glGetnUniformdvarB = PFNGLGETNUNIFORMDVARBPROC(glfwGetProcAddress("glGetnUniformdvarB"));
01827     glGetnUniformfv = PFNGLGETNUNIFORMFVPROC(glfwGetProcAddress("glGetnUniformfv"));
01828     glGetnUniformfvARB = PFNGLGETNUNIFORMFVARBPROC(glfwGetProcAddress("glGetnUniformfvARB"));
01829     glGetnUniformi64vARB = PFNGLGETNUNIFORMI64VARBPROC(glfwGetProcAddress("glGetnUniformi64vARB"));
01830     glGetnUniformiv = PFNGLGETNUNIFORMIVPROC(glfwGetProcAddress("glGetnUniformiv"));
01831     glGetnUniformivARB = PFNGLGETNUNIFORMIVARBPROC(glfwGetProcAddress("glGetnUniformivARB"));
01832     glGetnUniformui64vARB = PFNGLGETNUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetnUniformui64vARB"));
01833     glGetnUniformuiv = PFNGLGETNUNIFORMUIVPROC(glfwGetProcAddress("glGetnUniformuiv"));
01834     glGetnUniformuivARB = PFNGLGETNUNIFORMUIVARBPROC(glfwGetProcAddress("glGetnUniformuivARB"));
01835     glHint = PFNGLHINTPROC(glfwGetProcAddress("glHint"));
01836     glIndexFormatNV = PFNGLINDEXFORMATNVPROC(glfwGetProcAddress("glIndexFormatNV"));
01837     glInsertEventMarkerEXT =
01837 PFNGLINSERTEVENTMARKEREXTPROC(glfwGetProcAddress("glInsertEventMarkerEXT"));
01838     glInterpolatePathsNV = PFNGLINTERPOLATEPATHSNVPROC(glfwGetProcAddress("glInterpolatePathsNV"));
01839     glInvalidateBufferData =
01839 PFNGLINVALIDATEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
01840     glInvalidateBufferData =
01840 PFNGLINVALIDATEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
01841     glInvalidateFramebuffer =
01841 PFNGLINVALIDATEFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateFramebuffer"));
01842     glInvalidateNamedFramebufferData =
01842 PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferData"));
01843     glInvalidateNamedFramebufferSubData =
01843 PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferSubData"));
01844     glInvalidateSubFramebuffer =
01844 PFNGLINVALIDATESUBFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateSubFramebuffer"));
01845     glInvalidateTexImage = PFNGLINVALIDATETEXIMAGEPROC(glfwGetProcAddress("glInvalidateTexImage"));
01846     glInvalidateTexSubImage =
01846 PFNGLINVALIDATETEXSUBIMAGEPROC(glfwGetProcAddress("glInvalidateTexSubImage"));
01847     glIsBuffer = PFNGLISBUFFERPROC(glfwGetProcAddress("glIsBuffer"));
01848     glIsBufferResidentNV = PFNGLISBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsBufferResidentNV"));
01849     glIsCommandListNV = PFNGLCOMMANDLISTNVPROC(glfwGetProcAddress("glIsCommandListNV"));
01850     glIsEnabled = PFNGLISENABLEDPROC(glfwGetProcAddress("glIsEnabled"));
01851     glIsEnabledIndexedEXT = PFNGLISENABLEDINDEXEDEXTPROC(glfwGetProcAddress("glIsEnabledIndexedEXT"));
01852     glIsEnabledi = PFNGLISENABLEDIPROC(glfwGetProcAddress("glIsEnabledi"));
01853     glIsFramebuffer = PFNGLISFRAMEBUFFERPROC(glfwGetProcAddress("glIsFramebuffer"));
01854     glIsImageHandleResidentARB =
01854 PFNGLISIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsImageHandleResidentARB"));
01855     glIsImageHandleResidentNV =
01855 PFNGLISIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsImageHandleResidentNV"));
01856     glIsNamedBufferResidentNV =
01856 PFNGLISNAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsNamedBufferResidentNV"));
01857     glIsNamedStringARB = PFNGLISNAMEDSTRINGARBPROC(glfwGetProcAddress("glIsNamedStringARB"));
01858     glIsPathNV = PFNGLISPATHNVPROC(glfwGetProcAddress("glIsPathNV"));
01859     glIsPointInFillPathNV = PFNGLISPOINTINFPATHNVPROC(glfwGetProcAddress("glIsPointInFillPathNV"));
01860     glIsPointInStrokePathNV =
01860 PFNGLISPOINTINSTROKEPATHNVPROC(glfwGetProcAddress("glIsPointInStrokePathNV"));
01861     glIsProgram = PFNGLISPROGRAMPROC(glfwGetProcAddress("glIsProgram"));
01862     glIsProgramPipeline = PFNGLISPROGRAMPIPELINEPROC(glfwGetProcAddress("glIsProgramPipeline"));
01863     glIsQuery = PFNGLISQUERYPROC(glfwGetProcAddress("glIsQuery"));
01864     glIsRenderbuffer = PFNGLISRENDERBUFFERPROC(glfwGetProcAddress("glIsRenderbuffer"));
01865     glIsSampler = PFNGLISSAMPLERPROC(glfwGetProcAddress("glIsSampler"));
01866     glIsShader = PFNGLISSHADERPROC(glfwGetProcAddress("glIsShader"));
01867     glIsStateNV = PFNGLISSTATENVPROC(glfwGetProcAddress("glIsStateNV"));
01868     glIsSync = PFNGLISSYNCPROC(glfwGetProcAddress("glIsSync"));
01869     glIsTexture = PFNGLISTEXTUREPROC(glfwGetProcAddress("glIsTexture"));
01870     glIsTextureHandleResidentARB =

```

```

PFNGLISTTEXTUREHANDLERESIDENTARBPROC glfwGetProcAddress("glIsTextureHandleResidentARB"));
01871    glIsTextureHandleResidentNV =
PFNGLISTTEXTUREHANDLERESIDENTNVPROC glfwGetProcAddress("glIsTextureHandleResidentNV");
01872    glIsTransformFeedback = PFNGLISTRANSFORMFEEDBACKPROC glfwGetProcAddress("glIsTransformFeedback");
01873    glIsVertexArray = PFNGLISVERTEXARRAYPROC glfwGetProcAddress("glIsVertexArray");
01874    glLabelObjectEXT = PFNGLLABELOBJECTEXTPROC glfwGetProcAddress("glLabelObjectEXT");
01875    glLineWidth = PFNGLLINEWIDTHPROC glfwGetProcAddress("glLineWidth");
01876    glLinkProgram = PFNGLLINKPROGRAMPROC glfwGetProcAddress("glLinkProgram");
01877    glListDrawCommandsStatesClientNV =
PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glfwGetProcAddress("glListDrawCommandsStatesClientNV");
01878    glLogicOp = PFNGLLOGICOPPROC glfwGetProcAddress("glLogicOp");
01879    glMakeBufferNonResidentNV =
PFNGLMAKEBUFFERNONRESIDENTNVPROC glfwGetProcAddress("glMakeBufferNonResidentNV");
01880    glMakeBufferResidentNV =
PFNGLMAKEBUFFERRESIDENTNVPROC glfwGetProcAddress("glMakeBufferResidentNV");
01881    glMakeImageHandleNonResidentARB =
PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glfwGetProcAddress("glMakeImageHandleNonResidentARB");
01882    glMakeImageHandleNonResidentNV =
PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glfwGetProcAddress("glMakeImageHandleNonResidentNV");
01883    glMakeImageHandleResidentARB =
PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glfwGetProcAddress("glMakeImageHandleResidentARB");
01884    glMakeImageHandleResidentNV =
PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glfwGetProcAddress("glMakeImageHandleResidentNV");
01885    glMakeNamedBufferNonResidentNV =
PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glfwGetProcAddress("glMakeNamedBufferNonResidentNV");
01886    glMakeNamedBufferResidentNV =
PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glfwGetProcAddress("glMakeNamedBufferResidentNV");
01887    glMakeTextureHandleNonResidentARB =
PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glfwGetProcAddress("glMakeTextureHandleNonResidentARB");
01888    glMakeTextureHandleNonResidentNV =
PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glfwGetProcAddress("glMakeTextureHandleNonResidentNV");
01889    glMakeTextureHandleResidentARB =
PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glfwGetProcAddress("glMakeTextureHandleResidentARB");
01890    glMakeTextureHandleResidentNV =
PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glfwGetProcAddress("glMakeTextureHandleResidentNV");
01891    glMapBuffer = PFNGLMAPBUFFERPROC glfwGetProcAddress("glMapBuffer");
01892    glMapBufferRange = PFNGLMAPBUFFERRANGEPROC glfwGetProcAddress("glMapBufferRange");
01893    glMapNamedBuffer = PFNGLMAPNAMEDBUFFERPROC glfwGetProcAddress("glMapNamedBuffer");
01894    glMapNamedBufferEXT = PFNGLMAPNAMEDBUFFEREXTPROC glfwGetProcAddress("glMapNamedBufferEXT");
01895    glMapNamedBufferRange = PFNGLMAPNAMEDBUFFERRANGEPROC glfwGetProcAddress("glMapNamedBufferRange");
01896    glMapNamedBufferRangeEXT =
PFNGLMAPNAMEDBUFFERRANGEEXTPROC glfwGetProcAddress("glMapNamedBufferRangeEXT");
01897    glMatrixFrustumEXT = PFNGLMATRIXFRUSTUMEXTPROC glfwGetProcAddress("glMatrixFrustumEXT");
01898    glMatrixLoad3x2fNV = PFNGLMATRIXLOAD3X2FNVPROC glfwGetProcAddress("glMatrixLoad3x2fNV");
01899    glMatrixLoad3x3fNV = PFNGLMATRIXLOAD3X3FNVPROC glfwGetProcAddress("glMatrixLoad3x3fNV");
01900    glMatrixLoadIdentityEXT =
PFNGLMATRIXLOADIDENTITYEXTPROC glfwGetProcAddress("glMatrixLoadIdentityEXT");
01901    glMatrixLoadTranspose3x3fNV =
PFNGLMATRIXLOADTRANPOSE3X3FNVPROC glfwGetProcAddress("glMatrixLoadTranspose3x3fNV");
01902    glMatrixLoadTransposedEXT =
PFNGLMATRIXLOADTRANPOSEDEXTPROC glfwGetProcAddress("glMatrixLoadTransposedEXT");
01903    glMatrixLoadTransposefEXT =
PFNGLMATRIXLOADTRANPOSEFEXTPROC glfwGetProcAddress("glMatrixLoadTransposefEXT");
01904    glMatrixLoaddEXT = PFNGLMATRIXLOADDEXTPROC glfwGetProcAddress("glMatrixLoaddEXT");
01905    glMatrixLoadfEXT = PFNGLMATRIXLOADFEXTPROC glfwGetProcAddress("glMatrixLoadfEXT");
01906    glMatrixMult3x2fNV = PFNGLMATRIXMULT3X2FNVPROC glfwGetProcAddress("glMatrixMult3x2fNV");
01907    glMatrixMult3x3fNV = PFNGLMATRIXMULT3X3FNVPROC glfwGetProcAddress("glMatrixMult3x3fNV");
01908    glMatrixMultTranspose3x3fNV =
PFNGLMATRIXMULTTRANPOSE3X3FNVPROC glfwGetProcAddress("glMatrixMultTranspose3x3fNV");
01909    glMatrixMultTransposedEXT =
PFNGLMATRIXMULTTRANPOSEDEXTPROC glfwGetProcAddress("glMatrixMultTransposedEXT");
01910    glMatrixMultTransposefEXT =
PFNGLMATRIXMULTTRANPOSEFEXTPROC glfwGetProcAddress("glMatrixMultTransposefEXT");
01911    glMatrixMultdEXT = PFNGLMATRIXMULTDEXTPROC glfwGetProcAddress("glMatrixMultdEXT");
01912    glMatrixMultfEXT = PFNGLMATRIXMULTFEXTPROC glfwGetProcAddress("glMatrixMultfEXT");
01913    glMatrixOrthoEXT = PFNGLMATRIXORTHOEXTPROC glfwGetProcAddress("glMatrixOrthoEXT");
01914    glMatrixPopEXT = PFNGLMATRIXPOPEXTPROC glfwGetProcAddress("glMatrixPopEXT");
01915    glMatrixPushEXT = PFNGLMATRIXPUSHEXTPROC glfwGetProcAddress("glMatrixPushEXT");
01916    glMatrixRotatedEXT = PFNGLMATRIXROTATEDEXTPROC glfwGetProcAddress("glMatrixRotatedEXT");
01917    glMatrixRotatefEXT = PFNGLMATRIXROTATEFEXTPROC glfwGetProcAddress("glMatrixRotatefEXT");
01918    glMatrixScaledEXT = PFNGLMATRIXSCALEDEXTPROC glfwGetProcAddress("glMatrixScaledEXT");
01919    glMatrixScalefEXT = PFNGLMATRIXSCALEFEXTPROC glfwGetProcAddress("glMatrixScalefEXT");
01920    glMatrixTranslatedEXT = PFNGLMATRIXTRANSLATEDEXTPROC glfwGetProcAddress("glMatrixTranslatedEXT");
01921    glMatrixTranslatefEXT = PFNGLMATRIXTRANSLATEFEXTPROC glfwGetProcAddress("glMatrixTranslatefEXT");
01922    glMaxShaderCompilerThreadsARB =
PFNGLMAXSHADERCOMPILERTHREADSARBPROC glfwGetProcAddress("glMaxShaderCompilerThreadsARB");
01923    glMemoryBarrier = PFNGLMEMORYBARRIERPROC glfwGetProcAddress("glMemoryBarrier");
01924    glMemoryBarrierByRegion =
PFNGLMEMORYBARRIERBYREGIONPROC glfwGetProcAddress("glMemoryBarrierByRegion");
01925    glMinSampleShading = PFNGLMINSAMPLESHADINGPROC glfwGetProcAddress("glMinSampleShading");
01926    glMinSampleShadingARB = PFNGLMINSAMPLESHADINGARBPROC glfwGetProcAddress("glMinSampleShadingARB");
01927    glMultiDrawArrays = PFNGLMULTIDRAWARRAYSPROC glfwGetProcAddress("glMultiDrawArrays");
01928    glMultiDrawArraysIndirect =
PFNGLMULTIDRAWARRAYSINDIRECTPROC glfwGetProcAddress("glMultiDrawArraysIndirect");
01929    glMultiDrawArraysIndirectBindlessCountNV =
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glfwGetProcAddress("glMultiDrawArraysIndirectBindlessCountNV");
01930    glMultiDrawArraysIndirectBindlessNV =

```

```

01931 PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessNV"));
01931     glMultiDrawArraysIndirectCountARB =
01932 PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawArraysIndirectCountARB"));
01932     glMultiDrawElements = PFNGLMULTIDRAWELEMENTSPROC(glfwGetProcAddress("glMultiDrawElements"));
01933     glMultiDrawElementsBaseVertex =
01934 PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glMultiDrawElementsBaseVertex"));
01934     glMultiDrawElementsIndirectCountARB =
01935 PFNGLMULTIDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glMultiDrawElementsIndirect"));
01935     glMultiDrawElementsIndirectBindlessCountNV =
01936 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessCountNV"));
01936     glMultiDrawElementsIndirectBindlessNV =
01937 PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessNV"));
01937     glMultiDrawElementsIndirectCountARB =
01938 PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawElementsIndirectCountARB"));
01938     glMultiTexBufferEXT = PFNGLMULTITEXBUFFEREXTPROC(glfwGetProcAddress("glMultiTexBufferEXT"));
01939     glMultiTexCoordPointerEXT =
01940 PFNGLMULTITEXCOORDPOINTERTEXTPROC(glfwGetProcAddress("glMultiTexCoordPointerEXT"));
01940     glMultiTexEnvfEXT = PFNGLMULTITEXENVFEXTPROC(glfwGetProcAddress("glMultiTexEnvfEXT"));
01941     glMultiTexEnvfvEXT = PFNGLMULTITEXENVFVEXTPROC(glfwGetProcAddress("glMultiTexEnvfvEXT"));
01942     glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
01943     glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
01944     glMultiTexGendEXT = PFNGLMULTITEXGENDEXTPROC(glfwGetProcAddress("glMultiTexGendEXT"));
01945     glMultiTexGendvEXT = PFNGLMULTITEXGENDEVENTPROC(glfwGetProcAddress("glMultiTexGendvEXT"));
01946     glMultiTexGenfEXT = PFNGLMULTITEXGENFEXTPROC(glfwGetProcAddress("glMultiTexGenfEXT"));
01947     glMultiTexGenfvEXT = PFNGLMULTITEXGENFVEXTPROC(glfwGetProcAddress("glMultiTexGenfvEXT"));
01948     glMultiTexGeniEXT = PFNGLMULTITEXGENIEXTPROC(glfwGetProcAddress("glMultiTexGeniEXT"));
01949     glMultiTexGenivEXT = PFNGLMULTITEXGENIVEXTPROC(glfwGetProcAddress("glMultiTexGenivEXT"));
01950     glMultiTexImage1DEXT = PFNGLMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexImage1DEXT"));
01951     glMultiTexImage2DEXT = PFNGLMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexImage2DEXT"));
01952     glMultiTexImage3DEXT = PFNGLMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexImage3DEXT"));
01953     glMultiTexParameterIiivEXT =
01954 PFNGLMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIiivEXT"));
01954     glMultiTexParameterIiivEXT =
01955 PFNGLMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIuivEXT"));
01955     glMultiTexParameterIfEXT =
01956 PFNGLMULTITEXPARAMETERFEXTPROC(glfwGetProcAddress("glMultiTexParameterfEXT"));
01956     glMultiTexParameterfvEXT =
01957 PFNGLMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glMultiTexParameterfvEXT"));
01957     glMultiTexParameterieEXT =
01958 PFNGLMULTITEXPARAMETERIEXTPROC(glfwGetProcAddress("glMultiTexParameterieEXT"));
01958     glMultiTexParameterivEXT =
01959 PFNGLMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glMultiTexParameterivEXT"));
01959     glMultiTexRenderbufferEXT =
01960 PFNGLMULTITEXRENDERBUFFEREXTPROC(glfwGetProcAddress("glMultiTexRenderbufferEXT"));
01960     glMultiTexSubImage1DEXT =
01961 PFNGLMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexSubImage1DEXT"));
01961     glMultiTexSubImage2DEXT =
01962 PFNGLMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexSubImage2DEXT"));
01962     glMultiTexSubImage3DEXT =
01963 PFNGLMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexSubImage3DEXT"));
01963     glNamedBufferData = PFNGLNAMEDBUFFERDATAPROC(glfwGetProcAddress("glNamedBufferData"));
01964     glNamedBufferDataEXT = PFNGLNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glNamedBufferDataEXT"));
01965     glNamedBufferPageCommitmentARB =
01966 PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glNamedBufferPageCommitmentARB"));
01966     glNamedBufferPageCommitmentEXT =
01967 PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glNamedBufferPageCommitmentEXT"));
01967     glNamedBufferStorage = PFNGLNAMEDBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedBufferStorage"));
01968     glNamedBufferStorageEXT =
01969 PFNGLNAMEDBUFFERSTORAGEEXTPROC(glfwGetProcAddress("glNamedBufferStorageEXT"));
01969     glNamedBufferSubData = PFNGLNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glNamedBufferSubData"));
01970     glNamedBufferSubDataEXT =
01971 PFNGLNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedBufferSubDataEXT"));
01971     glNamedCopyBufferSubDataEXT =
01972 PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedCopyBufferSubDataEXT"));
01972     glNamedFramebufferDrawBuffer =
01973 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffer"));
01973     glNamedFramebufferDrawBuffers =
01974 PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffers"));
01974     glNamedFramebufferParameteri =
01975 PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glNamedFramebufferParameteri"));
01975     glNamedFramebufferParameteriEXT =
01976 PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC(glfwGetProcAddress("glNamedFramebufferParameteriEXT"));
01976     glNamedFramebufferReadBuffer =
01977 PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC(glfwGetProcAddress("glNamedFramebufferReadBuffer"));
01977     glNamedFramebufferRenderbuffer =
01978 PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glNamedFramebufferRenderbuffer"));
01978     glNamedFramebufferRenderbufferEXT =
01979 PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC(glfwGetProcAddress("glNamedFramebufferRenderbufferEXT"));
01979     glNamedFramebufferSampleLocationsfvARB =
01980 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvARB"));
01980     glNamedFramebufferSampleLocationsfvNV =
01981 PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvNV"));
01981     glNamedFramebufferTexture =
01982 PFNGLNAMEDFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glNamedFramebufferTexture"));
01982     glNamedFramebufferTexture1DEXT =
01983 PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture1DEXT"));
01983     glNamedFramebufferTexture2DEXT =

```

```

01984 PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture2DEXT"));
01984     glNamedFramebufferTexture3DEXT =
01985 PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture3DEXT"));
01985     glNamedFramebufferTextureEXT =
01986 PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureEXT"));
01986     glNamedFramebufferTextureFaceEXT =
01987 PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureFaceEXT"));
01987     glNamedFramebufferTextureLayer =
01988 PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glNamedFramebufferTextureLayer"));
01988     glNamedFramebufferTextureLayerEXT =
01989 PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC(glfwGetProcAddress("glNamedFramebufferTextureLayerEXT"));
01989     glNamedProgramLocalParameter4dEXT =
01990 PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dEXT"));
01990     glNamedProgramLocalParameter4dvEXT =
01991 PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dvEXT"));
01991     glNamedProgramLocalParameter4fEXT =
01992 PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fEXT"));
01992     glNamedProgramLocalParameter4fvEXT =
01993 PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fvEXT"));
01993     glNamedProgramLocalParameterI4iEXT =
01994 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4iEXT"));
01994     glNamedProgramLocalParameterI4ivEXT =
01995 PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4ivEXT"));
01995     glNamedProgramLocalParameterI4uiEXT =
01996 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uiEXT"));
01996     glNamedProgramLocalParameterI4uivEXT =
01997 PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uivEXT"));
01997     glNamedProgramLocalParameters4fvEXT =
01998 PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameters4fvEXT"));
01998     glNamedProgramLocalParametersI4ivEXT =
01999 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4ivEXT"));
01999     glNamedProgramLocalParametersI4uivEXT =
02000 PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4uivEXT"));
02000     glNamedProgramStringEXT =
02001 PFNGLNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glNamedProgramStringEXT"));
02001     glNamedRenderbufferStorage =
02002 PFNGLNAMEDRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedRenderbufferStorage"));
02002     glNamedRenderbufferStorageEXT =
02003 PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageEXT"));
02003     glNamedRenderbufferStorageMultisample =
02004 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisample"));
02004     glNamedRenderbufferStorageMultisampleCoverageEXT =
02005 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERGEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleCoverageEXT"));
02005     glNamedRenderbufferStorageMultisampleEXT =
02006 PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleEXT"));
02006     glNamedStringARB = PFNGLNAMEDSTRINGARBPROC(glfwGetProcAddress("glNamedStringARB"));
02007     glNormalFormatNV = PFNGLNORMALFORMATNVPROC(glfwGetProcAddress("glNormalFormatNV"));
02008     glObjectLabel = PFNGLOBJECTLABELPROC(glfwGetProcAddress("glObjectLabel"));
02008     glObjectLabel = PFNGLOBJECTPTRLABELPROC(glfwGetProcAddress("glObjectPtrLabel"));
02009     glObjectPtrLabel = PFNGLOBJECTPTRLABELPROC(glfwGetProcAddress("glObjectPtrLabel"));
02010     glPatchParameterfv = PFNGLPATCHPARAMETERFVPROC(glfwGetProcAddress("glPatchParameterfv"));
02011     glPatchParameteri = PFNGLPATCHPARAMETERIPROC(glfwGetProcAddress("glPatchParameteri"));
02012     glPatchCommandsNV = PFNGLPATHCOMMANDSNVPROC(glfwGetProcAddress("glPathCommandsNV"));
02013     glPathCoordsNV = PFNGLPATHCOORDSNVPROC(glfwGetProcAddress("glPathCoordsNV"));
02014     glPathCoverDepthFuncNV =
02015 PFNGLPATHCOVERDEPTHFUNCNVPROC(glfwGetProcAddress("glPathCoverDepthFuncNV"));
02015     glPathDashArrayNV = PFNGLPATHDASHARRAYNVPROC(glfwGetProcAddress("glPathDashArrayNV"));
02016     glPathGlyphIndexArrayNV =
02017 PFNGLPATHGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathGlyphIndexArrayNV"));
02017     glPathGlyphIndexRangeNV =
02018 PFNGLPATHGLYPHINDEXRANGENVPROC(glfwGetProcAddress("glPathGlyphIndexRangeNV"));
02018     glPathGlyphRangeNV = PFNGLPATHGLYPHRANGENVPROC(glfwGetProcAddress("glPathGlyphRangeNV"));
02019     glPathGlyphsNV = PFNGLPATHGLYPHSNVPROC(glfwGetProcAddress("glPathGlyphsNV"));
02020     glPathMemoryGlyphIndexArrayNV =
02021 PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathMemoryGlyphIndexArrayNV"));
02021     glPathParameterfNV = PFNGLPATHPARAMETERFNVPROC(glfwGetProcAddress("glPathParameterfNV"));
02022     glPathParameterfvNV = PFNGLPATHPARAMETERFVNVPROC(glfwGetProcAddress("glPathParameterfvNV"));
02023     glPathParameteriNV = PFNGLPATHPARAMETERINVPROC(glfwGetProcAddress("glPathParameteriNV"));
02024     glPathParameterivNV = PFNGLPATHPARAMETERIVNVPROC(glfwGetProcAddress("glPathParameterivNV"));
02025     glPathStencilDepthOffsetNV =
02026 PFNGLPATHSTENCILDEPTHOFFSETNVPROC(glfwGetProcAddress("glPathStencilDepthOffsetNV"));
02026     glPathStencilFuncNV = PFNGLPATHSTENCILFUNCNVPROC(glfwGetProcAddress("glPathStencilFuncNV"));
02027     glPathStringNV = PFNGLPATHSTRINGNVPROC(glfwGetProcAddress("glPathStringNV"));
02028     glPathSubCommandsNV = PFNGLPATHSUBCOMMANDSNVPROC(glfwGetProcAddress("glPathSubCommandsNV"));
02029     glPathSubCoordsNV = PFNGLPATHSUBCOORDSNVPROC(glfwGetProcAddress("glPathSubCoordsNV"));
02030     glPauseTransformFeedback =
02031 PFNGLPAUSETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glPauseTransformFeedback"));
02031     glPixelStoref = PFNGLPIXELSTOREFPROC(glfwGetProcAddress("glPixelStoref"));
02032     glPixelStorei = PFNGLPIXELSTOREIPROC(glfwGetProcAddress("glPixelStorei"));
02033     glPointAlongPathNV = PFNGLPOINTALONGPATHNVPROC(glfwGetProcAddress("glPointAlongPathNV"));
02034     glPointParameterf = PFNGLPOINTPARAMETERFPROC(glfwGetProcAddress("glPointParameterf"));
02035     glPointParameterfv = PFNGLPOINTPARAMETERFVPROC(glfwGetProcAddress("glPointParameterfv"));
02036     glPointParameteri = PFNGLPOINTPARAMETERIPROC(glfwGetProcAddress("glPointParameteri"));
02037     glPointParameteriv = PFNGLPOINTPARAMETERIVPROC(glfwGetProcAddress("glPointParameteriv"));
02038     glPointSize = PFNGLPOINTSIZEPROC(glfwGetProcAddress("glPointSize"));
02039     glPolygonMode = PFNGLPOLYGMODEPROC(glfwGetProcAddress("glPolygonMode"));
02040     glPolygonOffset = PFNGLPOLYGONOFFSETPROC(glfwGetProcAddress("glPolygonOffset"));
02041     glPolygonOffsetClampEXT =

```



```

02153 PFNGLPROGRAMUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glProgramUniformMatrix2fv"));
02153     glProgramUniformMatrix2fvEXT =
02154 PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2fvEXT"));
02154     glProgramUniformMatrix2x3dv =
02155 PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dv"));
02155     glProgramUniformMatrix2x3dvEXT =
02156 PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dvEXT"));
02156     glProgramUniformMatrix2x3fv =
02157 PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fv"));
02157     glProgramUniformMatrix2x3fvEXT =
02158 PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fvEXT"));
02158     glProgramUniformMatrix2x4dv =
02159 PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dv"));
02159     glProgramUniformMatrix2x4dvEXT =
02160 PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dvEXT"));
02160     glProgramUniformMatrix2x4fv =
02161 PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fv"));
02161     glProgramUniformMatrix2x4fvEXT =
02162 PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fvEXT"));
02162     glProgramUniformMatrix3dv =
02163 PFNGLPROGRAMUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glProgramUniformMatrix3dv"));
02163     glProgramUniformMatrix3dvEXT =
02164 PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3dvEXT"));
02164     glProgramUniformMatrix3fv =
02165 PFNGLPROGRAMUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glProgramUniformMatrix3fv"));
02165     glProgramUniformMatrix3fvEXT =
02166 PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3fvEXT"));
02166     glProgramUniformMatrix3x2dv =
02167 PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dv"));
02167     glProgramUniformMatrix3x2dvEXT =
02168 PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dvEXT"));
02168     glProgramUniformMatrix3x2fv =
02169 PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fv"));
02169     glProgramUniformMatrix3x2fvEXT =
02170 PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fvEXT"));
02170     glProgramUniformMatrix3x4dv =
02171 PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dv"));
02171     glProgramUniformMatrix3x4dvEXT =
02172 PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dvEXT"));
02172     glProgramUniformMatrix3x4fv =
02173 PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fv"));
02173     glProgramUniformMatrix3x4fvEXT =
02174 PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fvEXT"));
02174     glProgramUniformMatrix4dv =
02175 PFNGLPROGRAMUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glProgramUniformMatrix4dv"));
02175     glProgramUniformMatrix4dvEXT =
02176 PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4dvEXT"));
02176     glProgramUniformMatrix4fv =
02177 PFNGLPROGRAMUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));
02177     glProgramUniformMatrix4fvEXT =
02178 PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fvEXT"));
02178     glProgramUniformMatrix4x2dv =
02179 PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dv"));
02179     glProgramUniformMatrix4x2dvEXT =
02180 PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dvEXT"));
02180     glProgramUniformMatrix4x2fv =
02181 PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fv"));
02181     glProgramUniformMatrix4x2fvEXT =
02182 PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fvEXT"));
02182     glProgramUniformMatrix4x3dv =
02183 PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dv"));
02183     glProgramUniformMatrix4x3dvEXT =
02184 PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dvEXT"));
02184     glProgramUniformMatrix4x3fv =
02185 PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fv"));
02185     glProgramUniformMatrix4x3fvEXT =
02186 PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fvEXT"));
02186     glProgramUniformui64NV =
02187 PFNGLPROGRAMUNIFORMUI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
02187     glProgramUniformui64vNV =
02188 PFNGLPROGRAMUNIFORMUI64NVNPROC(glfwGetProcAddress("glProgramUniformui64vNV"));
02188     glProvokingVertex = PFNGLPROVOKINGVERTEXPROC(glfwGetProcAddress("glProvokingVertex"));
02189     glPushClientAttribDefaultEXT =
02189 PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glPushClientAttribDefaultEXT"));
02190     glPushDebugGroup = PFNGLPUSHDEBUGGROUPPROC(glfwGetProcAddress("glPushDebugGroup"));
02191     glPushGroupMarkerEXT = PFNGLPUSHGROUPMARKEREXTPROC(glfwGetProcAddress("glPushGroupMarkerEXT"));
02192     glQueryCounter = PFNGLQUERYCOUNTERPROC(glfwGetProcAddress("glQueryCounter"));
02193     glRasterSamplesEXT = PFNGLRASTERSAMPLESEXTPROC(glfwGetProcAddress("glRasterSamplesEXT"));
02194     glReadBuffer = PFNGLREADBUFFERPROC(glfwGetProcAddress("glReadBuffer"));
02195     glReadPixels = PFNGLREADPIXELSPROC(glfwGetProcAddress("glReadPixels"));
02196     glReadnPixels = PFNGLREADNPPIXELSPROC(glfwGetProcAddress("glReadnPixels"));
02197     glReadnPixelsARB = PFNGLREADNPPIXELSBPROC(glfwGetProcAddress("glReadnPixelsARB"));
02198     glReleaseShaderCompiler =
02198 PFNGLRELEASESHADERCOMPILERPROC(glfwGetProcAddress("glReleaseShaderCompiler"));
02199     glRenderbufferStorage = PFNGLRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glRenderbufferStorage"));
02200     glRenderbufferStorageMultisample =
02200 PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glRenderbufferStorageMultisample"));

```

```

02201    glRenderbufferStorageMultisampleCoverageNV =
02202    PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC(glfwGetProcAddress("glRenderbufferStorageMultisampleCoverageNV"));
02203    glResolveDepthValuesNV =
02204    PFNGLRESOLVEDEPTHVALUESNVPROC(glfwGetProcAddress("glResolveDepthValuesNV"));
02205    glResumeTransformFeedback =
02206    PFNGLRESUMETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glResumeTransformFeedback"));
02207    glSampleCoverage = PFNGLSAMPLECOVERAGEPROC(glfwGetProcAddress("glSampleCoverage"));
02208    glSampleMaski = PFNGLSAMPLEMASKIPROC(glfwGetProcAddress("glSampleMaski"));
02209    glSamplerParameterIiv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameterIiv"));
02210    glSamplerParameterIuiv =
02211    PFNGLSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glSamplerParameterIuiv"));
02212    glSamplerParameterf = PFNGLSAMPLERPARAMETERFPROC(glfwGetProcAddress("glSamplerParameterf"));
02213    glSamplerParameterfv = PFNGLSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glSamplerParameterfv"));
02214    glSamplerParameteri = PFNGLSAMPLERPARAMETERIIPROC(glfwGetProcAddress("glSamplerParameteri"));
02215    glSamplerParameteriv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameteriv"));
02216    glScissor = PFNGLSCISSORPROC(glfwGetProcAddress("glScissor"));
02217    glScissorArrayv = PFNGLSCISSORARRAYVPROC(glfwGetProcAddress("glScissorArrayv"));
02218    glScissorIndexed = PFNGLSCISSORINDEXEDPROC(glfwGetProcAddress("glScissorIndexed"));
02219    glScissorIndexedv = PFNGLSCISSORINDEXEDVPROC(glfwGetProcAddress("glScissorIndexedv"));
02220    glSecondaryColorFormatNV =
02221    PFNGLSECONDARYCOLORFORMATNVPROC(glfwGetProcAddress("glSecondaryColorFormatNV"));
02222    glSelectPerfMonitorCountersAMD =
02223    PFNGLSELECTPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glSelectPerfMonitorCountersAMD"));
02224    glShaderBinary = PFNGLSHADERBINARYPROC(glfwGetProcAddress("glShaderBinary"));
02225    glShaderSource = PFNGLSHADERSOURCEPROC(glfwGetProcAddress("glShaderSource"));
02226    glShaderStorageBlockBinding =
02227    PFNGLSHADERSTORAGEBLOCKBINDINGPROC(glfwGetProcAddress("glShaderStorageBlockBinding"));
02228    glSignalVkFenceNV = PFNGLSIGNALLVKFENCENVPROC(glfwGetProcAddress("glSignalVkFenceNV"));
02229    glSignalVkSemaphoreNV = PFNGLSIGNALLVKSEMAPHORENVPROC(glfwGetProcAddress("glSignalVkSemaphoreNV"));
02230    glSpecializeShaderARB = PFNGLSPECIALIZESHADERARBPROC(glfwGetProcAddress("glSpecializeShaderARB"));
02231    glStateCaptureNV = PFNGLSTATECAPTURENVPROC(glfwGetProcAddress("glStateCaptureNV"));
02232    glStencilFillPathInstancedNV =
02233    PFNGLSTENCILFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilFillPathInstancedNV"));
02234    glStencilFillPathNV = PFNGLSTENCILFILLPATHNVPROC(glfwGetProcAddress("glStencilFillPathNV"));
02235    glStencilFunc =
02236    PFNGLSTENCILFUNCPROC(glfwGetProcAddress("glStencilFunc"));
02237    glStencilFuncSeparate =
02238    PFNGLSTENCILFUNCSEPARATEPROC(glfwGetProcAddress("glStencilFuncSeparate"));
02239    glStencilMask =
02240    PFNGLSTENCILMASKPROC(glfwGetProcAddress("glStencilMask"));
02241    glStencilMaskSeparate =
02242    PFNGLSTENCILMASKSEPARATEPROC(glfwGetProcAddress("glStencilMaskSeparate"));
02243    glStencilOp =
02244    PFNGLSTENCILOPPROC(glfwGetProcAddress("glStencilOp"));
02245    glStencilOpSeparate =
02246    PFNGLSTENCILOPSEPARATEPROC(glfwGetProcAddress("glStencilOpSeparate"));
02247    glStencilStrokePathInstancedNV =
02248    PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilStrokePathInstancedNV"));
02249    glStencilStrokePathNV =
02250    PFNGLSTENCILSTROKEPATHNVPROC(glfwGetProcAddress("glStencilStrokePathNV"));
02251    glStencilThenCoverFillPathInstancedNV =
02252    PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathInstancedNV"));
02253    glStencilThenCoverFillPathNV =
02254    PFNGLSTENCILTHENCOVERFILLPATHNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathNV"));
02255    glStencilThenCoverStrokePathInstancedNV =
02256    PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathInstancedNV"));
02257    glStencilThenCoverStrokePathNV =
02258    PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathNV"));
02259    glSubpixelPrecisionBiasNV =
02260    PFNGLSUBPIXELPRECISIONBIASNVPROC(glfwGetProcAddress("glSubpixelPrecisionBiasNV"));
02261    glTexBuffer = PFNGLTEXBUFFERPROC(glfwGetProcAddress("glTexBuffer"));
02262    glTexBufferARB = PFNGLTEXBUFFERARBPROC(glfwGetProcAddress("glTexBufferARB"));
02263    glTexBufferSize = PFNGLTEXBUFFERRANGEPROC(glfwGetProcAddress("glTexBufferSize"));
02264    glTexCoordFormatNV = PFNGLTEXCOORDFORMATNVPROC(glfwGetProcAddress("glTexCoordFormatNV"));
02265    glTexImage1D = PFNGLTEXIMAGE1DPROC(glfwGetProcAddress("glTexImage1D"));
02266    glTexImage2D = PFNGLTEXIMAGE2DPROC(glfwGetProcAddress("glTexImage2D"));
02267    glTexImage2DMultisample =
02268    PFNGLTEXIMAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage2DMultisample"));
02269    glTexImage3D = PFNGLTEXIMAGE3DPROC(glfwGetProcAddress("glTexImage3D"));
02270    glTexImage3DMultisample =
02271    PFNGLTEXIMAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage3DMultisample"));
02272    glTexPageCommitmentARB =
02273    PFNGLTEXPAGECOMMITMENTARBPROC(glfwGetProcAddress("glTexPageCommitmentARB"));
02274    glTexParameterIiv = PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameterIiv"));
02275    glTexParameterIuiv = PFNGLTEXPARAMETERIUIVPROC(glfwGetProcAddress("glTexParameterIuiv"));
02276    glTexParameterf =
02277    PFNGLTEXPARAMETERFPROC(glfwGetProcAddress("glTexParameterf"));
02278    glTexParameterfv =
02279    PFNGLTEXPARAMETERFVPROC(glfwGetProcAddress("glTexParameterfv"));
02280    glTexParameterIi = PFNGLTEXPARAMETERIIPROC(glfwGetProcAddress("glTexParameterIi"));
02281    glTexParameteriv =
02282    PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameteriv"));
02283    glTexStorage1D =
02284    PFNGLTEXSTORAGE1DPROC(glfwGetProcAddress("glTexStorage1D"));
02285    glTexStorage2D =
02286    PFNGLTEXSTORAGE2DPROC(glfwGetProcAddress("glTexStorage2D"));
02287    glTexStorage2DMultisample =
02288    PFNGLTEXSTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage2DMultisample"));
02289    glTexStorage3D =
02290    PFNGLTEXSTORAGE3DPROC(glfwGetProcAddress("glTexStorage3D"));
02291    glTexStorage3DMultisample =
02292    PFNGLTEXSTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage3DMultisample"));
02293    glTexSubImage1D =
02294    PFNGLTEXSUBIMAGE1DPROC(glfwGetProcAddress("glTexSubImage1D"));
02295    glTexSubImage2D =
02296    PFNGLTEXSUBIMAGE2DPROC(glfwGetProcAddress("glTexSubImage2D"));
02297    glTexSubImage3D =
02298    PFNGLTEXSUBIMAGE3DPROC(glfwGetProcAddress("glTexSubImage3D"));
02299    glTextureBarrier =
02300    PFNGLTEXTUREBARRIERPROC(glfwGetProcAddress("glTextureBarrier"));
02301    glTextureBarrierNV =
02302    PFNGLTEXTUREBARRIERNVPROC(glfwGetProcAddress("glTextureBarrierNV"));
02303    glTextureBuffer =
02304    PFNGLTEXTUREBUFFERPROC(glfwGetProcAddress("glTextureBuffer"));
02305    glTextureBufferEXT =
02306    PFNGLTEXTUREBUFFEREXTPROC(glfwGetProcAddress("glTextureBufferEXT"));
02307    glTextureBufferRange =
02308    PFNGLTEXTUREBUFFERRANGEPROC(glfwGetProcAddress("glTextureBufferRange"));

```

```

02269  glTextureBufferRangeEXT =
02270  PFNGLTEXTUREBUFFERRANGEEXTPROC(glfwGetProcAddress("glTextureBufferRangeEXT"));
02271  glTextureImage1DEXT = PFNGLTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glTextureImage1DEXT"));
02272  glTextureImage2DEXT = PFNGLTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glTextureImage2DEXT"));
02273  glTextureImage3DEXT = PFNGLTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glTextureImage3DEXT"));
02274  glTexturePageCommitmentEXT =
02275  PFNGLTEXTUREPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glTexturePageCommitmentEXT"));
02276  glTextureParameterIiiv = PFNGLTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glTextureParameterIiiv"));
02277  glTextureParameterIiivEXT =
02278  PFNGLTEXTUREPARAMETERIIVEXTPROC(glfwGetProcAddress("glTextureParameterIiivEXT"));
02279  glTextureParameterIuiv =
02280  PFNGLTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glTextureParameterIuiv"));
02281  glTextureParameterIuivEXT =
02282  PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
02283  glTextureParameterIf = PFNGLTEXTUREPARAMETERFPROC(glfwGetProcAddress("glTextureParameterf"));
02284  glTextureParameterfEXT =
02285  PFNGLTEXTUREPARAMETERFEXTPROC(glfwGetProcAddress("glTextureParameterfEXT"));
02286  glTextureParameterfv = PFNGLTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glTextureParameterfv"));
02287  glTextureParameterfvEXT =
02288  PFNGLTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glTextureParameterfvEXT"));
02289  glTextureParameteri = PFNGLTEXTUREPARAMETERIPROC(glfwGetProcAddress("glTextureParameteri"));
02290  glTextureParameteriEXT =
02291  PFNGLTEXTUREPARAMETERIEXTPROC(glfwGetProcAddress("glTextureParameteriEXT"));
02292  glTextureParameteriv = PFNGLTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glTextureParameteriv"));
02293  glTextureParameterivEXT =
02294  PFNGLTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glTextureParameterivEXT"));
02295  glTextureRenderbufferEXT =
02296  PFNGLTEXTURERENDERBUFFEREXTPROC(glfwGetProcAddress("glTextureRenderbufferEXT"));
02297  glTextureStorage1D = PFNGLTEXTURERESTORAGE1DPROC(glfwGetProcAddress("glTextureStorage1D"));
02298  glTextureStorage1DEXT = PFNGLTEXTURERESTORAGE1DEXTPROC(glfwGetProcAddress("glTextureStorage1DEXT"));
02299  glTextureStorage2D = PFNGLTEXTURERESTORAGE2DFPROC(glfwGetProcAddress("glTextureStorage2D"));
02300  glTextureStorage2DEXT = PFNGLTEXTURERESTORAGE2DEXTPROC(glfwGetProcAddress("glTextureStorage2DEXT"));
02301  glTextureStorage2DMultisample =
02302  PFNGLTEXTURERESTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage2DMultisample"));
02303  glTextureStorage2DMultisampleEXT =
02304  PFNGLTEXTURERESTORAGE2DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage2DMultisampleEXT"));
02305  glTextureStorage3D = PFNGLTEXTURERESTORAGE3DPROC(glfwGetProcAddress("glTextureStorage3D"));
02306  glTextureStorage3DEXT = PFNGLTEXTURERESTORAGE3DEXTPROC(glfwGetProcAddress("glTextureStorage3DEXT"));
02307  glTextureStorage3DMultisample =
02308  PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage3DMultisample"));
02309  glTextureSubImage1D = PFNGLTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glTextureSubImage1D"));
02310  glTextureSubImage1DEXT =
02311  PFNGLTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glTextureSubImage1DEXT"));
02312  glTextureSubImage2D = PFNGLTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glTextureSubImage2D"));
02313  glTextureSubImage2DEXT =
02314  PFNGLTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glTextureSubImage2DEXT"));
02315  glTextureSubImage3D = PFNGLTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glTextureSubImage3D"));
02316  glTextureSubImage3DEXT =
02317  PFNGLTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glTextureSubImage3DEXT"));
02318  glTextureView = PFNGLTEXTUREVIEWPROC(glfwGetProcAddress("glTextureView"));
02319  glTransformFeedbackBufferBase =
02320  PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC(glfwGetProcAddress("glTransformFeedbackBufferBase"));
02321  glTransformFeedbackBufferRange =
02322  PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC(glfwGetProcAddress("glTransformFeedbackBufferRange"));
02323  glTransformFeedbackVaryings =
02324  PFNGLTRANSFORMFEEDBACKVARYINGSPROC(glfwGetProcAddress("glTransformFeedbackVaryings"));
02325  glTransformPathNV = PFNGLTRANSFORMPATHNVPROC(glfwGetProcAddress("glTransformPathNV"));
02326  glUniform1d = PFNGLUNIFORM1DPROC(glfwGetProcAddress("glUniform1d"));
02327  glUniform1dv = PFNGLUNIFORM1DVPROC(glfwGetProcAddress("glUniform1dv"));
02328  glUniform1f = PFNGLUNIFORM1FPROC(glfwGetProcAddress("glUniform1f"));
02329  glUniform1fv = PFNGLUNIFORM1FVPROC(glfwGetProcAddress("glUniform1fv"));
02330  glUniform1i = PFNGLUNIFORM1IPROC(glfwGetProcAddress("glUniform1i"));
02331  glUniform1i64ARB = PFNGLUNIFORM1I64ARBPROC(glfwGetProcAddress("glUniform1i64ARB"));
02332  glUniform1i64NV = PFNGLUNIFORM1I64NVPROC(glfwGetProcAddress("glUniform1i64NV"));
02333  glUniform1i64vARB = PFNGLUNIFORM1I64VARBPROC(glfwGetProcAddress("glUniform1i64vARB"));
02334  glUniform1i64vNV = PFNGLUNIFORM1I64VNVPROC(glfwGetProcAddress("glUniform1i64vNV"));
02335  glUniform1liv = PFNGLUNIFORM1UIVPROC(glfwGetProcAddress("glUniform1liv"));
02336  glUniform1liv = PFNGLUNIFORM1UIPROC(glfwGetProcAddress("glUniform1liv"));
02337  glUniform1ui64ARB = PFNGLUNIFORM1UI64ARBPROC(glfwGetProcAddress("glUniform1ui64ARB"));
02338  glUniform1ui64NV = PFNGLUNIFORM1UI64NVPROC(glfwGetProcAddress("glUniform1ui64NV"));
02339  glUniform1ui64vARB = PFNGLUNIFORM1UI64VARBPROC(glfwGetProcAddress("glUniform1ui64vARB"));
02340  glUniform1ui64vNV = PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniform1ui64vNV"));
02341  glUniform1uiv = PFNGLUNIFORM1UIVPROC(glfwGetProcAddress("glUniform1uiv"));
02342  glUniform2d = PFNGLUNIFORM2DPROC(glfwGetProcAddress("glUniform2d"));
02343  glUniform2dv = PFNGLUNIFORM2DVPROC(glfwGetProcAddress("glUniform2dv"));
02344  glUniform2f = PFNGLUNIFORM2FPROC(glfwGetProcAddress("glUniform2f"));
02345  glUniform2fv = PFNGLUNIFORM2FVPROC(glfwGetProcAddress("glUniform2fv"));
02346  glUniform2i = PFNGLUNIFORM2IPROC(glfwGetProcAddress("glUniform2i"));
02347  glUniform2i64ARB = PFNGLUNIFORM2I64ARBPROC(glfwGetProcAddress("glUniform2i64ARB"));
02348  glUniform2i64NV = PFNGLUNIFORM2I64NVPROC(glfwGetProcAddress("glUniform2i64NV"));
02349  glUniform2i64vARB = PFNGLUNIFORM2I64VARBPROC(glfwGetProcAddress("glUniform2i64vARB"));
02350  glUniform2i64vNV = PFNGLUNIFORM2I64VNVPROC(glfwGetProcAddress("glUniform2i64vNV"));
02351  glUniform2iv = PFNGLUNIFORM2IVPROC(glfwGetProcAddress("glUniform2iv"));
02352  glUniform2ui = PFNGLUNIFORM2UIPROC(glfwGetProcAddress("glUniform2ui"));
02353  glUniform2ui64ARB = PFNGLUNIFORM2UI64ARBPROC(glfwGetProcAddress("glUniform2ui64ARB"));

```

```

02336 glUniform2ui64NV = PFNGLUNIFORM2UI64NVPROC(glfwGetProcAddress("glUniform2ui64NV"));
02337 glUniform2ui64VARB = PFNGLUNIFORM2UI64VARBPROC(glfwGetProcAddress("glUniform2ui64vARB"));
02338 glUniform2ui64NVN = PFNGLUNIFORM2UI64NVNPROC(glfwGetProcAddress("glUniform2ui64vNV"));
02339 glUniform2uiv = PFNGLUNIFORM2UIUVPROC(glfwGetProcAddress("glUniform2uiv"));
02340 glUniform3d = PFNGLUNIFORM3DPROC(glfwGetProcAddress("glUniform3d"));
02341 glUniform3dv = PFNGLUNIFORM3DVPROC(glfwGetProcAddress("glUniform3dv"));
02342 glUniform3f = PFNGLUNIFORM3FPROC(glfwGetProcAddress("glUniform3f"));
02343 glUniform3fv = PFNGLUNIFORM3FVPROC(glfwGetProcAddress("glUniform3fv"));
02344 glUniform3i = PFNGLUNIFORM3IPROC(glfwGetProcAddress("glUniform3i"));
02345 glUniform3i64ARB = PFNGLUNIFORM3I64ARBPROC(glfwGetProcAddress("glUniform3i64ARB"));
02346 glUniform3i64NV = PFNGLUNIFORM3I64NVPROC(glfwGetProcAddress("glUniform3i64NV"));
02347 glUniform3i64VARB = PFNGLUNIFORM3I64VARBPROC(glfwGetProcAddress("glUniform3i64VARB"));
02348 glUniform3i64vNV = PFNGLUNIFORM3I64VNVPROC(glfwGetProcAddress("glUniform3i64vNV"));
02349 glUniform3iv = PFNGLUNIFORM3IVPROC(glfwGetProcAddress("glUniform3iv"));
02350 glUniform3ui = PFNGLUNIFORM3UIPROC(glfwGetProcAddress("glUniform3ui"));
02351 glUniform3ui64ARB = PFNGLUNIFORM3UI64ARBPROC(glfwGetProcAddress("glUniform3ui64ARB"));
02352 glUniform3ui64NV = PFNGLUNIFORM3UI64VNVPROC(glfwGetProcAddress("glUniform3ui64NV"));
02353 glUniform3ui64VARB = PFNGLUNIFORM3UI64VARBPROC(glfwGetProcAddress("glUniform3ui64VARB"));
02354 glUniform3ui64vNVN = PFNGLUNIFORM3UI64VNVPROC(glfwGetProcAddress("glUniform3ui64vNVN"));
02355 glUniform3uiv = PFNGLUNIFORM3UIUVPROC(glfwGetProcAddress("glUniform3uiv"));
02356 glUniform4d = PFNGLUNIFORM4DPROC(glfwGetProcAddress("glUniform4d"));
02357 glUniform4dv = PFNGLUNIFORM4DVPROC(glfwGetProcAddress("glUniform4dv"));
02358 glUniform4f = PFNGLUNIFORM4FPROC(glfwGetProcAddress("glUniform4f"));
02359 glUniform4fv = PFNGLUNIFORM4FVPROC(glfwGetProcAddress("glUniform4fv"));
02360 glUniform4i = PFNGLUNIFORM4IPROC(glfwGetProcAddress("glUniform4i"));
02361 glUniform4i64ARB = PFNGLUNIFORM4I64ARBPROC(glfwGetProcAddress("glUniform4i64ARB"));
02362 glUniform4i64NV = PFNGLUNIFORM4I64NVPROC(glfwGetProcAddress("glUniform4i64NV"));
02363 glUniform4i64vARB = PFNGLUNIFORM4I64VARBPROC(glfwGetProcAddress("glUniform4i64vARB"));
02364 glUniform4i64vNV = PFNGLUNIFORM4I64VNVPROC(glfwGetProcAddress("glUniform4i64vNV"));
02365 glUniform4iv = PFNGLUNIFORM4IVPROC(glfwGetProcAddress("glUniform4iv"));
02366 glUniform4ui = PFNGLUNIFORM4UIPROC(glfwGetProcAddress("glUniform4ui"));
02367 glUniform4ui64ARB = PFNGLUNIFORM4UI64ARBPROC(glfwGetProcAddress("glUniform4ui64ARB"));
02368 glUniform4ui64NV = PFNGLUNIFORM4UI64VNVPROC(glfwGetProcAddress("glUniform4ui64NV"));
02369 glUniform4ui64VARB = PFNGLUNIFORM4UI64VARBPROC(glfwGetProcAddress("glUniform4ui64VARB"));
02370 glUniform4ui64vNVN = PFNGLUNIFORM4UI64VNVPROC(glfwGetProcAddress("glUniform4ui64vNVN"));
02371 glUniform4uiv = PFNGLUNIFORM4UIUVPROC(glfwGetProcAddress("glUniform4uiv"));
02372 glUniformBlockBinding = PFNGLUNIFORMBLOCKBINDINGPROC(glfwGetProcAddress("glUniformBlockBinding"));
02373 glUniformHandleui64ARB =
PFNGLUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glUniformHandleui64ARB"));
02374 glUniformHandleui64NV = PFNGLUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glUniformHandleui64NV"));
02375 glUniformHandleui64VARB =
PFNGLUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glUniformHandleui64vARB"));
02376 glUniformHandleui64NV =
PFNGLUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glUniformHandleui64vNV"));
02377 glUniformMatrix2dv = PFNGLUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glUniformMatrix2dv"));
02378 glUniformMatrix2fv = PFNGLUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glUniformMatrix2fv"));
02379 glUniformMatrix2x3dv = PFNGLUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glUniformMatrix2x3dv"));
02380 glUniformMatrix2x3fv = PFNGLUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glUniformMatrix2x3fv"));
02381 glUniformMatrix2x4dv = PFNGLUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glUniformMatrix2x4dv"));
02382 glUniformMatrix2x4fv = PFNGLUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glUniformMatrix2x4fv"));
02383 glUniformMatrix3dv = PFNGLUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glUniformMatrix3dv"));
02384 glUniformMatrix3fv = PFNGLUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glUniformMatrix3fv"));
02385 glUniformMatrix3x2dv = PFNGLUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glUniformMatrix3x2dv"));
02386 glUniformMatrix3x2fv = PFNGLUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glUniformMatrix3x2fv"));
02387 glUniformMatrix3x4dv = PFNGLUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glUniformMatrix3x4dv"));
02388 glUniformMatrix3x4fv = PFNGLUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glUniformMatrix3x4fv"));
02389 glUniformMatrix4dv = PFNGLUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glUniformMatrix4dv"));
02390 glUniformMatrix4fv = PFNGLUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glUniformMatrix4fv"));
02391 glUniformMatrix4x2dv = PFNGLUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glUniformMatrix4x2dv"));
02392 glUniformMatrix4x2fv = PFNGLUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glUniformMatrix4x2fv"));
02393 glUniformMatrix4x3dv = PFNGLUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glUniformMatrix4x3dv"));
02394 glUniformMatrix4x3fv = PFNGLUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glUniformMatrix4x3fv"));
02395 glUniformSubroutinesuiv =
PFNGLUNIFORMSUBROUTINESUVPROC(glfwGetProcAddress("glUniformSubroutinesuiv"));
02396 glUniformui64NV = PFNGLUNIFORMUI64NVPROC(glfwGetProcAddress("glUniformui64NV"));
02397 glUniformui64NVN = PFNGLUNIFORMUI64VNVPROC(glfwGetProcAddress("glUniformui64vNV"));
02398 glUnmapBuffer = PFNGLUNMAPBUFFERPROC(glfwGetProcAddress("glUnmapBuffer"));
02399 glUnmapNamedBuffer = PFNGLUNMAPNAMEDBUFFERPROC(glfwGetProcAddress("glUnmapNamedBuffer"));
02400 glUnmapNamedBufferEXT = PFNGLUNMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glUnmapNamedBufferEXT"));
02401 glUseProgram = PFNGLUSEPROGRAMPROC(glfwGetProcAddress("glUseProgram"));
02402 glUseProgramStages = PFNGLUSEPROGRAMSTAGESPROC(glfwGetProcAddress("glUseProgramStages"));
02403 glUseShaderProgramEXT = PFNGLUSESHELDERPROGRAMEXTPROC(glfwGetProcAddress("glUseShaderProgramEXT"));
02404 glValidateProgram = PFNGLVALIDATEPROGRAMPROC(glfwGetProcAddress("glValidateProgram"));
02405 glValidateProgramPipeline =
PFNGLVALIDATEPROGRAMPIPELINEPROC(glfwGetProcAddress("glValidateProgramPipeline"));
02406 glVertexArrayAttribBinding =
PFNGLVERTEXARRAYATTRIBBINDINGPROC(glfwGetProcAddress("glVertexArrayAttribBinding"));
02407 glVertexArrayAttribFormat =
PFNGLVERTEXARRAYATTRIBFORMATPROC(glfwGetProcAddress("glVertexArrayAttribFormat"));
02408 glVertexArrayAttribIFormat =
PFNGLVERTEXARRAYATTRIBIFORMATPROC(glfwGetProcAddress("glVertexArrayAttribIFormat"));
02409 glVertexArrayAttribLFormat =
PFNGLVERTEXARRAYATTRIBLFORMATPROC(glfwGetProcAddress("glVertexArrayAttribLFormat"));
02410 glVertexArrayBindVertexBufferEXT =
PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC(glfwGetProcAddress("glVertexArrayBindVertexBufferEXT"));
02411 glVertexArrayBindingDivisor =
PFNGLVERTEXARRAYBINDINGDIVISORPROC(glfwGetProcAddress("glVertexArrayBindingDivisor"));

```

```

02412     glVertexArrayColorOffsetEXT =
02413     PFNGLVERTEXARRAYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArrayColorOffsetEXT"));
02414     glVertexArrayEdgeFlagOffsetEXT =
02415     PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayEdgeFlagOffsetEXT"));
02416     glVertexArrayElementBuffer =
02417     PFNGLVERTEXARRAYELEMENTBUFFERPROC(glfwGetProcAddress("glVertexArrayElementBuffer"));
02418     glVertexArrayFogCoordOffsetEXT =
02419     PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayFogCoordOffsetEXT"));
02420     glVertexArrayIndexOffsetEXT =
02421     PFNGLVERTEXARRAYINDEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayIndexOffsetEXT"));
02422     glVertexArrayMultiTexCoordOffsetEXT =
02423     PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayMultiTexCoordOffsetEXT"));
02424     glVertexArrayNormalOffsetEXT =
02425     PFNGLVERTEXARRAYNORMALOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayNormalOffsetEXT"));
02426     glVertexArraySecondaryColorOffsetEXT =
02427     PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArraySecondaryColorOffsetEXT"));
02428     glVertexArrayTexCoordOffsetEXT =
02429     PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayTexCoordOffsetEXT"));
02430     glVertexArrayVertexAttribBindingEXT =
02431     PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribBindingEXT"));
02432     glVertexArrayVertexAttribDivisorEXT =
02433     PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribDivisorEXT"));
02434     glVertexArrayVertexAttribFormatEXT =
02435     PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribFormatEXT"));
02436     glVertexArrayVertexAttribIFormatEXT =
02437     PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIFormatEXT"));
02438     glVertexArrayVertexAttribIOffsetEXT =
02439     PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIOffsetEXT"));
02440     glVertexArrayVertexAttribLFormatEXT =
02441     PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLFormatEXT"));
02442     glVertexArrayVertexAttribLOffsetEXT =
02443     PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLOffsetEXT"));
02444     glVertexArrayVertexAttribOffsetEXT =
02445     PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribOffsetEXT"));
02446     glVertexArrayVertexBindingDivisorEXT =
02447     PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexBindingDivisorEXT"));
02448     glVertexArrayVertexBuffer =
02449     PFNGLVERTEXARRAYVERTEXBUFFERPROC(glfwGetProcAddress("glVertexArrayVertexBuffer"));
02450     glVertexArrayVertexBuffers =
02451     PFNGLVERTEXARRAYVERTEXBUFFERSPROC(glfwGetProcAddress("glVertexArrayVertexBuffers"));
02452     glVertexArrayVertexOffsetEXT =
02453     PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexOffsetEXT"));
02454     glVertexAttrib1d = PFNGLVERTEXATTRIB1DPROC(glfwGetProcAddress("glVertexAttrib1d"));
02455     glVertexAttrib1dv = PFNGLVERTEXATTRIB1DVPROC(glfwGetProcAddress("glVertexAttrib1dv"));
02456     glVertexAttrib1f = PFNGLVERTEXATTRIB1FPROC(glfwGetProcAddress("glVertexAttrib1f"));
02457     glVertexAttrib1fv = PFNGLVERTEXATTRIB1FVPROC(glfwGetProcAddress("glVertexAttrib1fv"));
02458     glVertexAttrib1ls = PFNGLVERTEXATTRIB1SPROC(glfwGetProcAddress("glVertexAttrib1ls"));
02459     glVertexAttrib1lsv = PFNGLVERTEXATTRIB1SVPROC(glfwGetProcAddress("glVertexAttrib1lsv"));
02460     glVertexAttrib2d = PFNGLVERTEXATTRIB2DPROC(glfwGetProcAddress("glVertexAttrib2d"));
02461     glVertexAttrib2dv = PFNGLVERTEXATTRIB2DVPROC(glfwGetProcAddress("glVertexAttrib2dv"));
02462     glVertexAttrib2f = PFNGLVERTEXATTRIB2FPROC(glfwGetProcAddress("glVertexAttrib2f"));
02463     glVertexAttrib2fv = PFNGLVERTEXATTRIB2FVPROC(glfwGetProcAddress("glVertexAttrib2fv"));
02464     glVertexAttrib2s = PFNGLVERTEXATTRIB2SPROC(glfwGetProcAddress("glVertexAttrib2s"));
02465     glVertexAttrib2sv = PFNGLVERTEXATTRIB2SVPROC(glfwGetProcAddress("glVertexAttrib2sv"));
02466     glVertexAttrib3d = PFNGLVERTEXATTRIB3DPROC(glfwGetProcAddress("glVertexAttrib3d"));
02467     glVertexAttrib3dv = PFNGLVERTEXATTRIB3DVPROC(glfwGetProcAddress("glVertexAttrib3dv"));
02468     glVertexAttrib3f = PFNGLVERTEXATTRIB3FPROC(glfwGetProcAddress("glVertexAttrib3f"));
02469     glVertexAttrib3fv = PFNGLVERTEXATTRIB3FVPROC(glfwGetProcAddress("glVertexAttrib3fv"));
02470     glVertexAttrib3s = PFNGLVERTEXATTRIB3SPROC(glfwGetProcAddress("glVertexAttrib3s"));
02471     glVertexAttrib3sv = PFNGLVERTEXATTRIB3SVPROC(glfwGetProcAddress("glVertexAttrib3sv"));
02472     glVertexAttrib4nbv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4nbv"));
02473     glVertexAttrib4niv = PFNGLVERTEXATTRIB4NIVPROC(glfwGetProcAddress("glVertexAttrib4niv"));
02474     glVertexAttrib4nsv = PFNGLVERTEXATTRIB4NSVPROC(glfwGetProcAddress("glVertexAttrib4nsv"));
02475     glVertexAttrib4nub = PFNGLVERTEXATTRIB4NUBPROC(glfwGetProcAddress("glVertexAttrib4nub"));
02476     glVertexAttrib4nubv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4nubv"));
02477     glVertexAttrib4nuiv = PFNGLVERTEXATTRIB4NUIVPROC(glfwGetProcAddress("glVertexAttrib4nuiv"));
02478     glVertexAttrib4nusv = PFNGLVERTEXATTRIB4NUSVPROC(glfwGetProcAddress("glVertexAttrib4nusv"));
02479     glVertexAttrib4bv = PFNGLVERTEXATTRIB4BVPROC(glfwGetProcAddress("glVertexAttrib4bv"));
02480     glVertexAttrib4d = PFNGLVERTEXATTRIB4DPROC(glfwGetProcAddress("glVertexAttrib4d"));
02481     glVertexAttrib4dv = PFNGLVERTEXATTRIB4DVPROC(glfwGetProcAddress("glVertexAttrib4dv"));
02482     glVertexAttrib4f = PFNGLVERTEXATTRIB4FPROC(glfwGetProcAddress("glVertexAttrib4f"));
02483     glVertexAttrib4fv = PFNGLVERTEXATTRIB4FVPROC(glfwGetProcAddress("glVertexAttrib4fv"));
02484     glVertexAttrib4iv = PFNGLVERTEXATTRIB4IVPROC(glfwGetProcAddress("glVertexAttrib4iv"));
02485     glVertexAttrib4s = PFNGLVERTEXATTRIB4SPROC(glfwGetProcAddress("glVertexAttrib4s"));
02486     glVertexAttrib4sv = PFNGLVERTEXATTRIB4SVPROC(glfwGetProcAddress("glVertexAttrib4sv"));
02487     glVertexAttrib4ubv = PFNGLVERTEXATTRIB4UBVPROC(glfwGetProcAddress("glVertexAttrib4ubv"));
02488     glVertexAttrib4uiv = PFNGLVERTEXATTRIB4UIVPROC(glfwGetProcAddress("glVertexAttrib4uiv"));
02489     glVertexAttrib4usv = PFNGLVERTEXATTRIB4USVPROC(glfwGetProcAddress("glVertexAttrib4usv"));
02490     glVertexAttribBinding = PFNGLVERTEXATTRIBBINDINGPROC(glfwGetProcAddress("glVertexAttribBinding"));
02491     glVertexAttribDivisor = PFNGLVERTEXATTRIBDIVISORPROC(glfwGetProcAddress("glVertexAttribDivisor"));
02492     glVertexAttribDivisorARB =
02493     PFNGLVERTEXATTRIBDIVISORARBPROC(glfwGetProcAddress("glVertexAttribDivisorARB"));
02494     glVertexAttribFormat =
02495     PFNGLVERTEXATTRIBFORMATPROC(glfwGetProcAddress("glVertexAttribFormat"));
02496     glVertexAttribFormatNV =
02497     PFNGLVERTEXATTRIBFORMATNVPROC(glfwGetProcAddress("glVertexAttribFormatNV"));
02498     glVertexAttribI1i = PFNGLVERTEXATTRIBI1IPROC(glfwGetProcAddress("glVertexAttribI1i"));
02499     glVertexAttribI1iv = PFNGLVERTEXATTRIBI1IVPROC(glfwGetProcAddress("glVertexAttribI1iv"));

```

```

02476 glVertexAttribI1ui = PFNGLVERTEXATTRIBI1UIPROC(glfwGetProcAddress("glVertexAttribI1ui"));
02477 glVertexAttribI1uiv = PFNGLVERTEXATTRIBI1UIVPROC(glfwGetProcAddress("glVertexAttribI1uiv"));
02478 glVertexAttribI2i = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2i"));
02479 glVertexAttribI2iv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2iv"));
02480 glVertexAttribI2ui = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2ui"));
02481 glVertexAttribI2uiv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2uiv"));
02482 glVertexAttribI3i = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3i"));
02483 glVertexAttribI3iv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3iv"));
02484 glVertexAttribI3ui = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3ui"));
02485 glVertexAttribI3uiv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3uiv"));
02486 glVertexAttribI4bv = PFNGLVERTEXATTRIBI4BVPROC(glfwGetProcAddress("glVertexAttribI4bv"));
02487 glVertexAttribI4i = PFNGLVERTEXATTRIBI4IPROC(glfwGetProcAddress("glVertexAttribI4i"));
02488 glVertexAttribI4iv = PFNGLVERTEXATTRIBI4IVPROC(glfwGetProcAddress("glVertexAttribI4iv"));
02489 glVertexAttribI4sv = PFNGLVERTEXATTRIBI4SVPROC(glfwGetProcAddress("glVertexAttribI4sv"));
02490 glVertexAttribI4ubv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4ubv"));
02491 glVertexAttribI4ui = PFNGLVERTEXATTRIBI4UIPROC(glfwGetProcAddress("glVertexAttribI4ui"));
02492 glVertexAttribI4uiv = PFNGLVERTEXATTRIBI4UIVPROC(glfwGetProcAddress("glVertexAttribI4uiv"));
02493 glVertexAttribI4usv = PFNGLVERTEXATTRIBI4USVPROC(glfwGetProcAddress("glVertexAttribI4usv"));
02494 glVertexAttribIFormat = PFNGLVERTEXATTRIBIFORMATPROC(glfwGetProcAddress("glVertexAttribIFormat"));
02495 glVertexAttribIFormatNV =
PFNGLVERTEXATTRIBIFORMATNVPROC(glfwGetProcAddress("glVertexAttribIFormatNV"));
02496 glVertexAttribIPointer =
PFNGLVERTEXATTRIBIPOINTERPROC(glfwGetProcAddress("glVertexAttribIPointer"));
02497 glVertexAttribL1d = PFNGLVERTEXATTRIBL1DPROC(glfwGetProcAddress("glVertexAttribL1d"));
02498 glVertexAttribL1dv = PFNGLVERTEXATTRIBL1DVPROC(glfwGetProcAddress("glVertexAttribL1dv"));
02499 glVertexAttribL1i64NV = PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
02500 glVertexAttribL1i64nv =
PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64nv"));
02501 glVertexAttribL1ui64ARB =
PFNGLVERTEXATTRIBL1UI64ARBPROC(glfwGetProcAddress("glVertexAttribL1ui64ARB"));
02502 glVertexAttribL1ui64NV =
PFNGLVERTEXATTRIBL1UI64NVPROC(glfwGetProcAddress("glVertexAttribL1ui64NV"));
02503 glVertexAttribL1ui64vARB =
PFNGLVERTEXATTRIBL1UI64VARBPROC(glfwGetProcAddress("glVertexAttribL1ui64vARB"));
02504 glVertexAttribL1ui64vNV =
PFNGLVERTEXATTRIBL1UI64VNVPROC(glfwGetProcAddress("glVertexAttribL1ui64vNV"));
02505 glVertexAttribL2d = PFNGLVERTEXATTRIBL2DPROC(glfwGetProcAddress("glVertexAttribL2d"));
02506 glVertexAttribL2dv = PFNGLVERTEXATTRIBL2DVPROC(glfwGetProcAddress("glVertexAttribL2dv"));
02507 glVertexAttribL2i64NV = PFNGLVERTEXATTRIBL2I64NVPROC(glfwGetProcAddress("glVertexAttribL2i64NV"));
02508 glVertexAttribL2i64vNV =
PFNGLVERTEXATTRIBL2I64VNVPROC(glfwGetProcAddress("glVertexAttribL2i64vNV"));
02509 glVertexAttribL2ui64NV =
PFNGLVERTEXATTRIBL2UI64NVPROC(glfwGetProcAddress("glVertexAttribL2ui64NV"));
02510 glVertexAttribL2ui64vNV =
PFNGLVERTEXATTRIBL2UI64VNVPROC(glfwGetProcAddress("glVertexAttribL2ui64vNV"));
02511 glVertexAttribL3d = PFNGLVERTEXATTRIBL3DPROC(glfwGetProcAddress("glVertexAttribL3d"));
02512 glVertexAttribL3dv = PFNGLVERTEXATTRIBL3DVPROC(glfwGetProcAddress("glVertexAttribL3dv"));
02513 glVertexAttribL3i64NV = PFNGLVERTEXATTRIBL3I64NVPROC(glfwGetProcAddress("glVertexAttribL3i64NV"));
02514 glVertexAttribL3i64vNV =
PFNGLVERTEXATTRIBL3I64VNVPROC(glfwGetProcAddress("glVertexAttribL3i64vNV"));
02515 glVertexAttribL3ui64NV =
PFNGLVERTEXATTRIBL3UI64NVPROC(glfwGetProcAddress("glVertexAttribL3ui64NV"));
02516 glVertexAttribL3ui64vNV =
PFNGLVERTEXATTRIBL3UI64VNVPROC(glfwGetProcAddress("glVertexAttribL3ui64vNV"));
02517 glVertexAttribL4d = PFNGLVERTEXATTRIBL4DPROC(glfwGetProcAddress("glVertexAttribL4d"));
02518 glVertexAttribL4dv = PFNGLVERTEXATTRIBL4DVPROC(glfwGetProcAddress("glVertexAttribL4dv"));
02519 glVertexAttribL4i64NV = PFNGLVERTEXATTRIBL4I64NVPROC(glfwGetProcAddress("glVertexAttribL4i64NV"));
02520 glVertexAttribL4i64vNV =
PFNGLVERTEXATTRIBL4I64VNVPROC(glfwGetProcAddress("glVertexAttribL4i64vNV"));
02521 glVertexAttribL4ui64NV =
PFNGLVERTEXATTRIBL4UI64NVPROC(glfwGetProcAddress("glVertexAttribL4ui64NV"));
02522 glVertexAttribL4ui64vNV =
PFNGLVERTEXATTRIBL4UI64VNVPROC(glfwGetProcAddress("glVertexAttribL4ui64vNV"));
02523 glVertexAttribLFormat = PFNGLVERTEXATTRIBLFORMATPROC(glfwGetProcAddress("glVertexAttribLFormat"));
02524 glVertexAttribLFormatNV =
PFNGLVERTEXATTRIBLFORMATNVPROC(glfwGetProcAddress("glVertexAttribLFormatNV"));
02525 glVertexAttribLPointer =
PFNGLVERTEXATTRIBLPOINTERPROC(glfwGetProcAddress("glVertexAttribLPointer"));
02526 glVertexAttribP1ui = PFNGLVERTEXATTRIBP1UIPROC(glfwGetProcAddress("glVertexAttribP1ui"));
02527 glVertexAttribP1uiv = PFNGLVERTEXATTRIBP1UIVPROC(glfwGetProcAddress("glVertexAttribP1uiv"));
02528 glVertexAttribP2ui = PFNGLVERTEXATTRIBP2UIPROC(glfwGetProcAddress("glVertexAttribP2ui"));
02529 glVertexAttribP2uiv = PFNGLVERTEXATTRIBP2UIVPROC(glfwGetProcAddress("glVertexAttribP2uiv"));
02530 glVertexAttribP3ui = PFNGLVERTEXATTRIBP3UIPROC(glfwGetProcAddress("glVertexAttribP3ui"));
02531 glVertexAttribP3uiv = PFNGLVERTEXATTRIBP3UIVPROC(glfwGetProcAddress("glVertexAttribP3uiv"));
02532 glVertexAttribP4ui = PFNGLVERTEXATTRIBP4UIPROC(glfwGetProcAddress("glVertexAttribP4ui"));
02533 glVertexAttribP4uiv = PFNGLVERTEXATTRIBP4UIVPROC(glfwGetProcAddress("glVertexAttribP4uiv"));
02534 glVertexAttribPointer = PFNGLVERTEXATTRIBPOINTERPROC(glfwGetProcAddress("glVertexAttribPointer"));
02535 glVertexBindingDivisor =
PFNGLVERTEXBINDINGDIVISORPROC(glfwGetProcAddress("glVertexBindingDivisor"));
02536 glVertexFormatNV = PFNGLVERTEXFORMATNVPROC(glfwGetProcAddress("glVertexFormatNV"));
02537 glViewport = PFNGLVIEWPORTPROC(glfwGetProcAddress("glViewport"));
02538 glViewportArrayv = PFNGLVIEWPORTARRAYPROC(glfwGetProcAddress("glViewportArrayv"));
02539 glViewportIndexeddf = PFNGLVIEWPORTINDEXEDFPROC(glfwGetProcAddress("glViewportIndexeddf"));
02540 glViewportIndexeddfv = PFNGLVIEWPORTINDEXEDFVPROC(glfwGetProcAddress("glViewportIndexeddfv"));
02541 glViewportPositionWScaleNV =
PFNGLVIEWPORTPOSITIONWSCALENVPROC(glfwGetProcAddress("glViewportPositionWScaleNV"));
02542 glViewportSwizzleNV = PFNGLVIEWPORTSWIZZLENVPROC(glfwGetProcAddress("glViewportSwizzleNV"));

```

```
02543     glWaitSync = PFNGLWAITSYNCPROC(glfwGetProcAddress("glWaitSync"));
02544     glWaitVkSemaphoreNV = PFNGLWAITVKSEMAPHORENVPROC(glfwGetProcAddress("glWaitVkSemaphoreNV"));
02545     glWeightPathsNV = PFNGLWEIGHTPATHSNVPROC(glfwGetProcAddress("glWeightPathsNV"));
02546     glWindowRectanglesEXT = PFNGLWINDOWRECTANGLESEXTPROC(glfwGetProcAddress("glWindowRectanglesEXT"));
02547 #endif
02548
02549 // 使用している GPU のバッファアライメントを調べる
02550     glGetIntegerv(GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT, &ggBufferAlignment);
02551 }
02552
02553 //
02554 // OpenGL のエラーをチェックする
02555 //
02556 // 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
02557 //
02558 // msg エラー発生時に標準エラー出力に出来する文字列. nullptr なら何も出力しない
02559 //
02560 void gg::ggError(const std::string& name, unsigned int line)
02561 {
02562     const GLenum error{ glGetError() };
02563
02564     if (error != GL_NO_ERROR)
02565     {
02566         if (!name.empty())
02567         {
02568             std::cerr << name;
02569             if (line > 0) std::cerr << " (" << line << ")";
02570             std::cerr << ":" << " ";
02571         }
02572
02573         switch (error)
02574         {
02575             case GL_INVALID_ENUM:
02576                 std::cerr << "An unacceptable value is specified for an enumerated argument" << std::endl;
02577                 break;
02578             case GL_INVALID_VALUE:
02579                 std::cerr << "A numeric argument is out of range" << std::endl;
02580                 break;
02581             case GL_INVALID_OPERATION:
02582                 std::cerr << "The specified operation is not allowed in the current state" << std::endl;
02583                 break;
02584             case GL_OUT_OF_MEMORY:
02585                 std::cerr << "There is not enough memory left to execute the command" << std::endl;
02586                 break;
02587             case GL_INVALID_FRAMEBUFFER_OPERATION:
02588                 std::cerr << "The specified operation is not allowed current frame buffer" << std::endl;
02589                 break;
02590             default:
02591                 std::cerr << "An OpenGL error has occurred: " << std::hex << std::showbase << error << std::endl;
02592                 break;
02593         }
02594     }
02595 }
02596
02597 //
02598 // FBO のエラーをチェックする
02599 //
02600 // 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
02601 //
02602 // msg エラー発生時に標準エラー出力に出来する文字列. nullptr なら何も出力しない
02603 //
02604 void gg::ggFBOError(const std::string& name, unsigned int line)
02605 {
02606     const GLenum status{ glCheckFramebufferStatus(GL_FRAMEBUFFER) };
02607
02608     if (status != GL_FRAMEBUFFER_COMPLETE)
02609     {
02610         if (!name.empty())
02611         {
02612             std::cerr << name;
02613             if (line > 0) std::cerr << " (" << line << ")";
02614             std::cerr << ":" << " ";
02615         }
02616
02617         switch (status)
02618         {
02619             case GL_FRAMEBUFFER_UNDEFINED:
02620                 std::cerr << "the default framebuffer does not exist" << std::endl;
02621                 break;
02622             case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
02623                 std::cerr << "Framebuffer incomplete, duplicate attachment" << std::endl;
02624                 break;
02625             case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
02626                 std::cerr << "Framebuffer incomplete, missing attachment" << std::endl;
02627                 break;
02628             case GL_FRAMEBUFFER_UNSUPPORTED:
02629                 std::cerr << "Unsupported framebuffer internal" << std::endl;

```

```

02630     break;
02631     case GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE:
02632         std::cerr << "The value of GL_RENDERBUFFER_SAMPLES is not"
02633         " the same for all attached renderbuffers or,"
02634         " if the attached images are a mix of renderbuffers and textures,"
02635         " the value of GL_RENDERBUFFER_SAMPLES is not zero" << std::endl;
02636     break;
02637 #if !defined(GL_GLES_PROTOTYPES)
02638     case GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS:
02639         std::cerr << "Any framebuffer attachment is layered,"
02640         " and any populated attachment is not layered,"
02641         " or if all populated color attachments are not from textures"
02642         " of the same target" << std::endl;
02643     break;
02644     case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER:
02645         std::cerr << "Framebuffer incomplete, missing draw buffer" << std::endl;
02646     break;
02647     case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
02648         std::cerr << "Framebuffer incomplete, missing read buffer" << std::endl;
02649     break;
02650 #endif
02651     default:
02652         std::cerr << "Programming error; will fail on all hardware: " << std::hex << std::showbase <<
02653         status << std::endl;
02654     break;
02655 }
02656 }
02657
02658 //
02659 // 変換行列：行列とベクトルの積  $c \leftarrow a \times b$ 
02660 //
02661 void gg::GgMatrix::projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02662 {
02663     for (int i = 0; i < 4; ++i)
02664     {
02665         c[i] = a[0 + i] * b[0] + a[4 + i] * b[1] + a[8 + i] * b[2] + a[12 + i] * b[3];
02666     }
02667 }
02668
02669 //
02670 // 変換行列：行列と行列の積  $c \leftarrow a \times b$ 
02671 //
02672 void gg::GgMatrix::multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const
02673 {
02674     for (int i = 0; i < 16; ++i)
02675     {
02676         int j = i & 3, k = i & ~3;
02677
02678         c[i] = a[0 + j] * b[k + 0] + a[4 + j] * b[k + 1] + a[8 + j] * b[k + 2] + a[12 + j] * b[k + 3];
02679     }
02680 }
02681
02682 //
02683 // 変換行列：単位行列を設定する
02684 //
02685 gg::GgMatrix& gg::GgMatrix::loadIdentity()
02686 {
02687     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02688     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02689     data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02690     data()[ 0] = data()[ 5] = data()[10] = data()[15] = 1.0f;
02691
02692     return *this;
02693 }
02694
02695 //
02696 // 変換行列：平行移動変換行列を設定する
02697 //
02698 gg::GgMatrix& gg::GgMatrix::loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02699 {
02700     data()[12] = x;
02701     data()[13] = y;
02702     data()[14] = z;
02703     data()[ 0] = data()[ 5] = data()[10] = data()[15] = w;
02704     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02705     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02706     data()[11] = 0.0f;
02707
02708     return *this;
02709 }
02710
02711 //
02712 // 変換行列：拡大縮小変換行列を設定する
02713 //
02714 gg::GgMatrix& gg::GgMatrix::loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
02715 {

```

```

02716     data()[ 0] = x;
02717     data()[ 5] = y;
02718     data()[10] = z;
02719     data()[15] = w;
02720     data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
02721     data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
02722     data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02723
02724     return *this;
02725 }
02726
02727 /**
02728 // 変換行列：x 軸中心の回転変換行列を設定する
02729 /**
02730 gg::GgMatrix& gg::GgMatrix::loadRotateX(GLfloat a)
02731 {
02732     const GLfloat c{ cosf(a) };
02733     const GLfloat s{ sinf(a) };
02734
02735     data()[ 0] = 1.0f; data()[ 1] = 0.0f; data()[ 2] = 0.0f; data()[ 3] = 0.0f;
02736     data()[ 4] = 0.0f; data()[ 5] = c;     data()[ 6] = s;     data()[ 7] = 0.0f;
02737     data()[ 8] = 0.0f; data()[ 9] = -s;    data()[10] = c;    data()[11] = 0.0f;
02738     data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02739
02740     return *this;
02741 }
02742
02743 /**
02744 // 変換行列：y 軸中心の回転変換行列を設定する
02745 /**
02746 gg::GgMatrix& gg::GgMatrix::loadRotateY(GLfloat a)
02747 {
02748     const GLfloat c{ cosf(a) };
02749     const GLfloat s{ sinf(a) };
02750
02751     data()[ 0] = c;     data()[ 1] = 0.0f; data()[ 2] = -s;    data()[ 3] = 0.0f;
02752     data()[ 4] = 0.0f; data()[ 5] = 1.0f; data()[ 6] = 0.0f; data()[ 7] = 0.0f;
02753     data()[ 8] = s;     data()[ 9] = 0.0f; data()[10] = c;    data()[11] = 0.0f;
02754     data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02755
02756     return *this;
02757 }
02758
02759 /**
02760 // 変換行列：z 軸中心の回転変換行列を設定する
02761 /**
02762 gg::GgMatrix& gg::GgMatrix::loadRotateZ(GLfloat a)
02763 {
02764     const GLfloat c{ cosf(a) };
02765     const GLfloat s{ sinf(a) };
02766
02767     data()[ 0] = c;     data()[ 1] = s;     data()[ 2] = 0.0f; data()[ 3] = 0.0f;
02768     data()[ 4] = -s;    data()[ 5] = c;     data()[ 6] = 0.0f; data()[ 7] = 0.0f;
02769     data()[ 8] = 0.0f; data()[ 9] = 0.0f; data()[10] = 1.0f; data()[11] = 0.0f;
02770     data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;
02771
02772     return *this;
02773 }
02774
02775 /**
02776 // 変換行列：任意軸中心の回転変換行列を設定する
02777 /**
02778 gg::GgMatrix& gg::GgMatrix::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
02779 {
02780     const GLfloat d{ sqrtf(x * x + y * y + z * z) };
02781
02782     if (d > 0.0f)
02783     {
02784         const GLfloat l{ x / d }, m{ y / d }, n{ z / d };
02785         const GLfloat l2{ l * l }, m2{ m * m }, n2{ n * n };
02786         const GLfloat lm{ l * m }, mn{ m * n }, nl{ n * l };
02787         const GLfloat c{ cosf(a) }, cl{ 1.0f - c };
02788         const GLfloat s{ sinf(a) };
02789
02790         data()[ 0] = (1.0f - l2) * c + l * cl + n * s;
02791         data()[ 1] = lm * cl + n * s;
02792         data()[ 2] = nl * cl - m * s;
02793         data()[ 3] = 0.0f;
02794
02795         data()[ 4] = lm * cl - n * s;
02796         data()[ 5] = (1.0f - m2) * c + m * cl + l * s;
02797         data()[ 6] = mn * cl + l * s;
02798         data()[ 7] = 0.0f;
02799
02800         data()[ 8] = nl * cl + m * s;
02801         data()[ 9] = mn * cl - l * s;
02802         data()[10] = (1.0f - n2) * c + n * cl;
02803     }
02804 }

```

```

02803     data()[11] = 0.0f;
02804
02805     data()[12] = 0.0f;
02806     data()[13] = 0.0f;
02807     data()[14] = 0.0f;
02808     data()[15] = 1.0f;
02809 }
02810
02811     return *this;
02812 }
02813
02814 /**
02815 // 変換行列：転置行列を設定する
02816 /**
02817 gg::GgMatrix& gg::GgMatrix::loadTranspose(const GLfloat* marray)
02818 {
02819     data()[ 0] = marray[ 0];
02820     data()[ 1] = marray[ 4];
02821     data()[ 2] = marray[ 8];
02822     data()[ 3] = marray[12];
02823     data()[ 4] = marray[ 1];
02824     data()[ 5] = marray[ 5];
02825     data()[ 6] = marray[ 9];
02826     data()[ 7] = marray[13];
02827     data()[ 8] = marray[ 2];
02828     data()[ 9] = marray[ 6];
02829     data()[10] = marray[10];
02830     data()[11] = marray[14];
02831     data()[12] = marray[ 3];
02832     data()[13] = marray[ 7];
02833     data()[14] = marray[11];
02834     data()[15] = marray[15];
02835
02836     return *this;
02837 }
02838
02839 /**
02840 // 変換行列：逆行行列を設定する
02841 /**
02842 gg::GgMatrix& gg::GgMatrix::loadInvert(const GLfloat* marray)
02843 {
02844     GLfloat lu[20], * plu[4];
02845
02846 // j 行の要素の値の絶対値の最大値を plu[j][4] に求める
02847 for (int j = 0; j < 4; ++j)
02848 {
02849     GLfloat max{ fabsf(* (plu[j] = lu + 5 * j) = *(marray++)) };
02850
02851     for (int i = 0; ++i < 4;)
02852     {
02853         GLfloat a{ fabsf(plu[j][i] = *(marray++)) };
02854         if (a > max) max = a;
02855     }
02856     if (max == 0.0f) return *this;
02857     plu[j][4] = 1.0f / max;
02858 }
02859
02860 // ピボットを考慮した LU 分解
02861 for (int j = 0; j < 4; ++j)
02862 {
02863     GLfloat max{ fabsf(plu[j][j] * plu[j][4]) };
02864     int i = j;
02865
02866     for (int k = j; ++k < 4;)
02867     {
02868         GLfloat a{ fabsf(plu[k][j] * plu[k][4]) };
02869         if (a > max)
02870         {
02871             max = a;
02872             i = k;
02873         }
02874     }
02875     if (i > j)
02876     {
02877         GLfloat* t{ plu[j] };
02878         plu[j] = plu[i];
02879         plu[i] = t;
02880     }
02881     if (plu[j][j] == 0.0f) return *this;
02882     for (int k = j; ++k < 4;)
02883     {
02884         plu[k][j] /= plu[j][j];
02885         for (int i = j; ++i < 4;)
02886         {
02887             plu[k][i] -= plu[j][i] * plu[k][j];
02888         }
02889     }

```

```

02890 }
02891 // LU 分解から逆行列を求める
02892 for (int k = 0; k < 4; ++k)
02893 {
02894     // array に単位行列を設定する
02895     for (int i = 0; i < 4; ++i)
02896     {
02897         data()[i * 4 + k] = (plu[i] == lu + k * 5) ? 1.0f : 0.0f;
02898     }
02899     // lu から逆行列を求める
02900     for (int i = 0; i < 4; ++i)
02901     {
02902         for (int j = i; ++j < 4;)
02903         {
02904             data()[j * 4 + k] -= data()[i * 4 + k] * plu[j][i];
02905         }
02906     }
02907 }
02908 for (int i = 4; --i >= 0;)
02909 {
02910     for (int j = i; ++j < 4;)
02911     {
02912         data()[i * 4 + k] -= plu[i][j] * data()[j * 4 + k];
02913     }
02914     data()[i * 4 + k] /= plu[i][i];
02915 }
02916 }
02917 return *this;
02918 }
02919 }
02920
02921 // 変換行列：法線変換行列を設定する
02922 // 変換行列：法線変換行列を設定する
02923 //
02924 gg::GgMatrix& gg::GgMatrix::loadNormal(const GLfloat* marray)
02925 {
02926     data()[0] = marray[5] * marray[10] - marray[6] * marray[9];
02927     data()[1] = marray[6] * marray[8] - marray[4] * marray[10];
02928     data()[2] = marray[4] * marray[9] - marray[5] * marray[8];
02929     data()[3] = marray[9] * marray[2] - marray[10] * marray[1];
02930     data()[5] = marray[10] * marray[0] - marray[8] * marray[2];
02931     data()[6] = marray[8] * marray[1] - marray[9] * marray[0];
02932     data()[8] = marray[1] * marray[6] - marray[2] * marray[5];
02933     data()[9] = marray[2] * marray[4] - marray[0] * marray[6];
02934     data()[10] = marray[0] * marray[5] - marray[1] * marray[4];
02935     data()[13] = data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
02936     data()[15] = 1.0f;
02937
02938 return *this;
02939 }
02940
02941 // 変換行列：ビュー変換行列を設定する
02942 // 変換行列：ビュー変換行列を設定する
02943 //
02944 gg::GgMatrix& gg::GgMatrix::loadLookat(
02945     GLfloat ex, GLfloat ey, GLfloat ez,
02946     GLfloat tx, GLfloat ty, GLfloat tz,
02947     GLfloat ux, GLfloat uy, GLfloat uz
02948 )
02949 {
02950     // z 軸 = e - t
02951     const GLfloat zx{ ex - tx };
02952     const GLfloat zy{ ey - ty };
02953     const GLfloat zz{ ez - tz };
02954
02955     // x 軸 = u x z 軸
02956     const GLfloat xx{ uy * zz - uz * zy };
02957     const GLfloat xy{ uz * zx - ux * zz };
02958     const GLfloat xz{ ux * zy - uy * zx };
02959
02960     // y 軸 = z 軸 x x 軸
02961     const GLfloat yx{ zy * xz - zz * xy };
02962     const GLfloat yy{ zz * xx - zx * xz };
02963     const GLfloat yz{ zx * xy - zy * xx };
02964
02965     // y 軸の長さをチェック
02966     GLfloat y{ yx * yx + yy * yy + yz * yz };
02967     if (y == 0.0f) return *this;
02968
02969     // x 軸の正規化
02970     const GLfloat x{ sqrtf(xx * xx + xy * xy + xz * xz) };
02971     data()[0] = xx / x;
02972     data()[4] = xy / x;
02973     data()[8] = xz / x;
02974
02975     // y 軸の正規化
02976     y = sqrtf(y);

```

```

02977     data()[ 1] = yx / y;
02978     data()[ 5] = yy / y;
02979     data()[ 9] = yz / y;
02980
02981     // z 軸の正規化
02982     const GLfloat z{ sqrtf(zx * zx + zy * zy + zz * zz) };
02983     data()[ 2] = zx / z;
02984     data()[ 6] = zy / z;
02985     data()[10] = zz / z;
02986
02987     // 平行移動
02988     data()[12] = -(ex * data()[ 0] + ey * data()[ 4] + ez * data()[ 8]);
02989     data()[13] = -(ex * data()[ 1] + ey * data()[ 5] + ez * data()[ 9]);
02990     data()[14] = -(ex * data()[ 2] + ey * data()[ 6] + ez * data()[10]);
02991
02992     // 残り
02993     data()[ 3] = data()[ 7] = data()[11] = 0.0f;
02994     data()[15] = 1.0f;
02995
02996     return *this;
02997 }
02998
02999 //
03000 // 変換行列：平行投影変換行列を設定する
03001 //
03002 gg::GgMatrix& gg::GgMatrix::loadOrthogonal(
03003     GLfloat left, GLfloat right,
03004     GLfloat bottom, GLfloat top,
03005     GLfloat zNear, GLfloat zFar
03006 )
03007 {
03008     const GLfloat dx{ right - left };
03009     const GLfloat dy{ top - bottom };
03010     const GLfloat dz{ zFar - zNear };
03011
03012     if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
03013     {
03014         data()[ 0] = 2.0f / dx;
03015         data()[ 5] = 2.0f / dy;
03016         data()[10] = -2.0f / dz;
03017         data()[12] = -(right + left) / dx;
03018         data()[13] = -(top + bottom) / dy;
03019         data()[14] = -(zFar + zNear) / dz;
03020         data()[15] = 1.0f;
03021         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03022         data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
03023         data()[11] = 0.0f;
03024     }
03025
03026     return *this;
03027 }
03028
03029 //
03030 // 変換行列：透視投影変換行列を設定する
03031 //
03032 gg::GgMatrix& gg::GgMatrix::loadFrustum(
03033     GLfloat left, GLfloat right,
03034     GLfloat bottom, GLfloat top,
03035     GLfloat zNear, GLfloat zFar
03036 )
03037 {
03038     const GLfloat dx{ right - left };
03039     const GLfloat dy{ top - bottom };
03040     const GLfloat dz{ zFar - zNear };
03041
03042     if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
03043     {
03044         data()[ 0] = 2.0f * zNear / dx;
03045         data()[ 5] = 2.0f * zNear / dy;
03046         data()[ 8] = (right + left) / dx;
03047         data()[ 9] = (top + bottom) / dy;
03048         data()[10] = -(zFar + zNear) / dz;
03049         data()[11] = -1.0f;
03050         data()[14] = -2.0f * zFar * zNear / dz;
03051         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03052         data()[ 6] = data()[ 7] = data()[12] = data()[13] =
03053         data()[15] = 0.0f;
03054     }
03055
03056     return *this;
03057 }
03058
03059 //
03060 // 変換行列：画角から透視投影変換行列を設定する
03061 //
03062 gg::GgMatrix& gg::GgMatrix::loadPerspective(
03063     GLfloat fovy, GLfloat aspect,

```

```

03064     GLfloat zNear, GLfloat zFar
03065 )
03066 {
03067     const GLfloat dz{ zFar - zNear };
03068
03069     if (dz != 0.0f)
03070     {
03071         data()[ 5] = 1.0f / tanf(fovy * 0.5f);
03072         data()[ 0] = data()[ 5] / aspect;
03073         data()[10] = -(zFar + zNear) / dz;
03074         data()[11] = -1.0f;
03075         data()[14] = -2.0f * zFar * zNear / dz;
03076         data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
03077             data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
03078             data()[12] = data()[13] = data()[15] = 0.0f;
03079     }
03080
03081     return *this;
03082 }
03083
03084 //
03085 // 四元数:GgQuaternion 型の四元数 p, q の積を r に求める
03086 //
03087 void gg::GgQuaternion::multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const
03088 {
03089     r[0] = p[1] * q[2] - p[2] * q[1] + p[0] * q[3] + p[3] * q[0];
03090     r[1] = p[2] * q[0] - p[0] * q[2] + p[1] * q[3] + p[3] * q[1];
03091     r[2] = p[0] * q[1] - p[1] * q[0] + p[2] * q[3] + p[3] * q[2];
03092     r[3] = p[3] * q[3] - p[0] * q[0] - p[1] * q[1] - p[2] * q[2];
03093 }
03094
03095 //
03096 // 四元数:GgQuaternion 型の四元数 q が表す変換行列を m に求める
03097 //
03098 void gg::GgQuaternion::toMatrix(GLfloat* m, const GLfloat* q) const
03099 {
03100     const GLfloat xx{ q[0] * q[0] * 2.0f };
03101     const GLfloat yy{ q[1] * q[1] * 2.0f };
03102     const GLfloat zz{ q[2] * q[2] * 2.0f };
03103     const GLfloat xy{ q[0] * q[1] * 2.0f };
03104     const GLfloat yz{ q[1] * q[2] * 2.0f };
03105     const GLfloat zx{ q[2] * q[0] * 2.0f };
03106     const GLfloat xw{ q[0] * q[3] * 2.0f };
03107     const GLfloat yw{ q[1] * q[3] * 2.0f };
03108     const GLfloat zw{ q[2] * q[3] * 2.0f };
03109
03110     m[ 0] = 1.0f - yy - zz;
03111     m[ 1] = xy + zw;
03112     m[ 2] = zx - yw;
03113     m[ 4] = xy - zw;
03114     m[ 5] = 1.0f - zz - xx;
03115     m[ 6] = yz + xw;
03116     m[ 8] = zx + yw;
03117     m[ 9] = yz - xw;
03118     m[10] = 1.0f - xx - yy;
03119     m[ 3] = m[ 7] = m[11] = m[12] = m[13] = m[14] = 0.0f;
03120     m[15] = 1.0f;
03121 }
03122
03123 //
03124 // 四元数:回転変換行列 a が表す四元数を q に求める
03125 //
03126 void gg::GgQuaternion::toQuaternion(GLfloat* q, const GLfloat* a) const
03127 {
03128     const GLfloat tr{ a[0] + a[5] + a[10] + a[15] };
03129
03130     if (tr > 0.0f)
03131     {
03132         q[3] = sqrtf(tr) * 0.5f;
03133         q[0] = (a[6] - a[9]) * 0.25f / q[3];
03134         q[1] = (a[8] - a[2]) * 0.25f / q[3];
03135         q[2] = (a[1] - a[4]) * 0.25f / q[3];
03136     }
03137 }
03138
03139 //
03140 // 四元数:球面線形補間 p に q と r を t で補間した四元数を求める
03141 //
03142 void gg::GgQuaternion::slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const
03143 {
03144     const GLfloat qr{ ggDot3(q, r) };
03145     const GLfloat ss{ 1.0f - qr * qr };
03146
03147     if (ss == 0.0f)
03148     {
03149         if (p != q)
03150         {

```

```

03151     p[0] = q[0];
03152     p[1] = q[1];
03153     p[2] = q[2];
03154     p[3] = q[3];
03155   }
03156 }
03157 else
03158 {
03159   const GLfloat sp{ sqrtf(ss) };
03160   const GLfloat ph{ acosf(qr) };
03161   const GLfloat pt{ ph * t };
03162   const GLfloat t1{ sinf(pt) / sp };
03163   const GLfloat t0{ sinf(ph - pt) / sp };
03164
03165   p[0] = q[0] * t0 + r[0] * t1;
03166   p[1] = q[1] * t0 + r[1] * t1;
03167   p[2] = q[2] * t0 + r[2] * t1;
03168   p[3] = q[3] * t0 + r[3] * t1;
03169 }
03170 }
03171 //
03172 // 四元数: (x, y, z) を軸とし角度 a 回転する四元数を求める
03173 //
03174 gg::GgQuaternion& gg::GgQuaternion::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03175 {
03176   const GLfloat l(x * x + y * y + z * z);
03177
03178   if (l != 0.0)
03179   {
03180     GLfloat s{ sinf(a *= 0.5f) / sqrtf(l) };
03181
03182     data()[0] = x * s;
03183     data()[1] = y * s;
03184     data()[2] = z * s;
03185   }
03186   else
03187   {
03188     data()[0] = data()[1] = data()[2] = 0.0f;
03189   }
03190   data()[3] = cosf(a);
03191
03192   return *this;
03193 }
03194 }
03195 //
03196 // x 軸中心に角度 a 回転する四元数を格納する
03197 //
03198 gg::GgQuaternion& gg::GgQuaternion::loadRotateX(GLfloat a)
03199 {
03200   const GLfloat t{ a * 0.5f };
03201
03202   data()[0] = sinf(t);
03203   data()[3] = cosf(t);
03204   data()[1] = data()[2] = 0.0f;
03205
03206   return *this;
03207 }
03208 }
03209 //
03210 // y 軸中心に角度 a 回転する四元数を格納する
03211 //
03212 gg::GgQuaternion& gg::GgQuaternion::loadRotateY(GLfloat a)
03213 {
03214   const GLfloat t{ a * 0.5f };
03215
03216   data()[1] = sinf(t);
03217   data()[3] = cosf(t);
03218   data()[0] = data()[2] = 0.0f;
03219
03220   return *this;
03221 }
03222 }
03223 //
03224 // z 軸中心に角度 a 回転する四元数を格納する
03225 //
03226 gg::GgQuaternion& gg::GgQuaternion::loadRotateZ(GLfloat a)
03227 {
03228   const GLfloat t{ a * 0.5f };
03229
03230   data()[2] = sinf(t);
03231   data()[3] = cosf(t);
03232   data()[0] = data()[1] = 0.0f;
03233
03234   return *this;
03235 }
03236 }
03237

```

```
03238 //  
03239 // 四元数：オイラー角 (heading, pitch, roll) にもとづいて四元数を求める  
03240 //  
03241 gg::GgQuaternion& gg::GgQuaternion::loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll)  
03242 {  
03243     GgQuaternion h, p, r;  
03244  
03245     h.loadRotateY(heading);  
03246     p.loadRotateX(pitch);  
03247     r.loadRotateZ(roll);  
03248  
03249     *this = h * p * r;  
03250  
03251     return *this;  
03252 }  
03253  
03254 //  
03255 // 四元数：正規化して格納する  
03256 //  
03257 gg::GgQuaternion& gg::GgQuaternion::loadNormalize(const GLfloat* a)  
03258 {  
03259     data()[0] = a[0];  
03260     data()[1] = a[1];  
03261     data()[2] = a[2];  
03262     data()[3] = a[3];  
03263  
03264     ggNormalize4(data());  
03265  
03266     return *this;  
03267 }  
03268  
03269 //  
03270 // 四元数：共役四元数を格納する  
03271 //  
03272 gg::GgQuaternion& gg::GgQuaternion::loadConjugate(const GLfloat* a)  
03273 {  
03274     // w 要素を反転する  
03275     data()[0] = a[0];  
03276     data()[1] = a[1];  
03277     data()[2] = a[2];  
03278     data()[3] = -a[3];  
03279  
03280     return *this;  
03281 }  
03282  
03283 //  
03284 // 四元数：逆元を格納する  
03285 //  
03286 gg::GgQuaternion& gg::GgQuaternion::loadInvert(const GLfloat* a)  
03287 {  
03288     // ノルムの二乗を求める  
03289     const GLfloat l(ggDot4(a, a));  
03290  
03291     if (l > 0.0f)  
03292     {  
03293         // 共役四元数を求める  
03294         GgQuaternion r;  
03295         r.loadConjugate(a);  
03296  
03297         // ノルムの二乗で割る  
03298         data()[0] = r[0] / l;  
03299         data()[1] = r[1] / l;  
03300         data()[2] = r[2] / l;  
03301         data()[3] = r[3] / l;  
03302     }  
03303  
03304     return *this;  
03305 }  
03306  
03307 //  
03308 // 簡易トラックボール処理：リセット  
03309 //  
03310 void gg::GgTrackball::reset(const GgQuaternion& q)  
03311 {  
03312     // ドラッグ中ではない  
03313     drag = false;  
03314  
03315     // 単位クオーターニオンに初期値を与える  
03316     this->load(cq = q);  
03317  
03318     // 回転行列を初期化する  
03319     static_cast<GgQuaternion*>(this)->getMatrix(rt);  
03320 }  
03321  
03322 //  
03323 // 簡易トラックボール処理：トラックボールする領域の設定  
03324 //
```

```

03325 //  Reshape コールバック (resize) の中に実行する
03326 //  (w, h) ウィンドウサイズ
03327 //
03328 void gg::GgTrackball::region(GLfloat w, GLfloat h)
03329 {
03330     // マウスボインタ位置のウィンドウ内の相対的位置への換算用
03331     scale[0] = 2.0f / w;
03332     scale[1] = 2.0f / h;
03333 }
03334
03335 //
03336 // 簡易トラックボール処理: ドラッグ開始時の処理
03337 //
03338 // マウスボタンを押したときに実行する
03339 // (x, y) 現在のマウス位置
03340 //
03341 void gg::GgTrackball::begin(GLfloat x, GLfloat y)
03342 {
03343     // ドラッグ開始
03344     drag = true;
03345
03346     // ドラッグ開始点を記録する
03347     start[0] = x;
03348     start[1] = y;
03349 }
03350
03351 //
03352 // 簡易トラックボール処理: ドラッグ中の処理
03353 //
03354 // マウスのドラッグ中に実行する
03355 // (x, y) 現在のマウス位置
03356 //
03357 void gg::GgTrackball::motion(GLfloat x, GLfloat y)
03358 {
03359     // ドラッグ中でなければ何もしない
03360     if (!drag) return;
03361
03362     // マウスボインタの位置のドラッグ開始位置からの変位
03363     const GLfloat d[] = { (x - start[0]) * scale[0], (y - start[1]) * scale[1] };
03364
03365     // マウスボインタの位置のドラッグ開始位置からの距離
03366     const GLfloat a = hypotf(d[0], d[1]);
03367
03368     // マウスボインタの位置がドラッグ開始位置から移動していれば
03369     if (a > 0.0)
03370     {
03371         // 現在の回転の四元数に作った四元数を掛けて合成する
03372         this->load(ggRotateQuaternion(d[1], d[0], 0.0f, a * 3.1415926536f) * cq);
03373
03374         // 合成した四元数から回転の変換行列を求める
03375         static_cast<GgQuaternion*>(this)->getMatrix(rt);
03376     }
03377 }
03378
03379 //
03380 // 簡易トラックボール処理: 回転角の修正
03381 //
03382 // 現在の回転角を修正する
03383 // q 修正分の回転角を表す四元数
03384 //
03385 void gg::GgTrackball::rotate(const GgQuaternion& q)
03386 {
03387     // ドラッグ中なら何もしない
03388     if (drag) return;
03389
03390     // 保存されている四元数に修正分の四元数を掛けて合成する
03391     this->load(q * cq);
03392
03393     // 合成した四元数から回転の変換行列を求める
03394     static_cast<GgQuaternion*>(this)->getMatrix(rt);
03395
03396     // 誤差を吸収するために正規化して保存する
03397     cq = normalize();
03398 }
03399
03400 //
03401 // 簡易トラックボール処理: 停止時の処理
03402 //
03403 // マウスボタンを離したときに実行する
03404 // (x, y) 現在のマウス位置
03405 //
03406 void gg::GgTrackball::end(GLfloat x, GLfloat y)
03407 {
03408     // ドラッグ終了点における回転を求める
03409     motion(x, y);
03410
03411     // 誤差を吸収するために正規化して保存する

```

```
03412 cq = normalize();
03413
03414 // ドラッグ終了
03415 drag = false;
03416 }
03417
03418 //
03419 // 配列に格納された画像の内容を TGA ファイルに保存する
03420 //
03421 // name ファイル名
03422 // buffer 画像データ
03423 // width 画像の横の画素数
03424 // height 画像の縦の画素数
03425 // depth 画像の 1 画素のバイト数
03426 // 戻り値 保存に成功すれば true, 失敗すれば false
03427 //
03428 bool gg::ggSaveTga(
03429 const std::string& name,
03430 const void* buffer,
03431 unsigned int width,
03432 unsigned int height,
03433 unsigned int depth
03434 )
03435 {
03436 // ファイルを開く
03437 std::ofstream file{ name, std::ios::binary };
03438
03439 // ファイルが開けなかったら戻る
03440 if (file.fail()) return false;
03441
03442 // 画像のヘッダ
03443 const unsigned char type{ static_cast<unsigned char>(depth == 0 ? 0 : depth < 3 ? 3 : 2) };
03444 const unsigned char alpha{ static_cast<unsigned char>(depth == 2 || depth == 4 ? 8 : 0) };
03445 const unsigned char header[18]
03446 {
03447 0,           // ID length
03448 0,           // Color map type (none)
03449 type,        // Image Type (2:RGB, 3:Grayscale)
03450 0, 0,         // Offset into the color map table
03451 0, 0,         // Number of color map entries
03452 0,           // Number of a color map entry bits per pixel
03453 0, 0,         // Horizontal image position
03454 0, 0,         // Vertical image position
03455 static_cast<unsigned char>(width & 0xff),
03456 static_cast<unsigned char>(width >> 8),
03457 static_cast<unsigned char>(height & 0xff),
03458 static_cast<unsigned char>(height >> 8),
03459 static_cast<unsigned char>(depth * 8),
03460 alpha         // Image descriptor
03461 };
03462
03463 // ヘッダを書き込む
03464 file.write(reinterpret_cast<const char*>(header), sizeof header);
03465
03466 // ヘッダの書き込みに失敗したら戻る
03467 if (file.bad())
03468 {
03469 file.close();
03470 return false;
03471 }
03472
03473 // データを書き込む
03474 const unsigned int size{ width * height * depth };
03475 if (type == 2)
03476 {
03477 // フルカラー
03478 std::vector<char> temp(size);
03479 for (GLuint i = 0; i < size; i += depth)
03480 {
03481 temp[i + 2] = static_cast<const char*>(buffer)[i + 0];
03482 temp[i + 1] = static_cast<const char*>(buffer)[i + 1];
03483 temp[i + 0] = static_cast<const char*>(buffer)[i + 2];
03484 if (depth == 4) temp[i + 3] = static_cast<const char*>(buffer)[i + 3];
03485 }
03486 file.write(temp.data(), size);
03487 }
03488 else if (type == 3)
03489 {
03490 // グレースケール
03491 file.write(static_cast<const char*>(buffer), size);
03492 }
03493
03494 // フッタを書き込む
03495 constexpr char footer[] = "\0\0\0\0\0\0\0\0\0TRUEVISION-XFILE.";
03496 file.write(footer, sizeof footer);
03497
03498 // データの書き込みに失敗したら戻る
```

```

03499  if (file.bad())
03500  {
03501      file.close();
03502      return false;
03503  }
03504
03505  // ファイルを閉じる
03506  file.close();
03507
03508  // データの書き込みに成功した
03509  return true;
03510 }
03511
03512 //
03513 // カラーバッファの内容を TGA ファイルに保存する
03514 //
03515 //   name 保存するファイル名
03516 //   戻り値 保存に成功すれば true, 失敗すれば false
03517 //
03518 bool gg::ggSaveColor(const std::string& name)
03519 {
03520     // 現在のビューポートのサイズを得る
03521     GLint viewport[4];
03522     glGetIntegerv(GL_VIEWPORT, viewport);
03523
03524     // ビューポートのサイズ分のメモリを確保する
03525     std::vector<GLubyte> buffer(viewport[2] * viewport[3] * 3);
03526
03527     // 画面表示の完了を待つ
03528     glFinish();
03529
03530     // カラーバッファを読み込む
03531     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03532         GL_RGB, GL_UNSIGNED_BYTE, buffer.data());
03533
03534     // 読み込んだデータをファイルに書き込む
03535     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 3);
03536 }
03537
03538 //
03539 // デプスバッファの内容を TGA ファイルに保存する
03540 //
03541 //   name 保存するファイル名
03542 //   戻り値 保存に成功すれば true, 失敗すれば false
03543 //
03544 bool gg::ggSaveDepth(const std::string& name)
03545 {
03546     // 現在のビューポートのサイズを得る
03547     GLint viewport[4];
03548     glGetIntegerv(GL_VIEWPORT, viewport);
03549
03550     // ビューポートのサイズ分のメモリを確保する
03551     std::vector<GLubyte> buffer(viewport[2] * viewport[3]);
03552
03553     // 画面表示の完了を待つ
03554     glFinish();
03555
03556     // デプスバッファを読み込む
03557     glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
03558         GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, buffer.data());
03559
03560     // 読み込んだデータをファイルに書き込む
03561     return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 1);
03562 }
03563
03564 //
03565 // TGA ファイル (8/16/24/32bit) を読み込む
03566 //
03567 //   name 読み込むファイル名
03568 //   pWidth 読み込んだファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)
03569 //   pHeight 読み込んだファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)
03570 //   pFormat 読み込んだファイルのフォーマットの格納先のポインタ (nullptr なら格納しない)
03571 //   image 読み込んだ画像を格納する vector
03572 //   戻り値 読み込みに成功すれば true, 失敗すれば false
03573 //
03574 bool gg::ggReadImage(
03575     const std::string& name,
03576     std::vector<GLubyte>& image,
03577     GLsizei* pWidth,
03578     GLsizei* pHeight,
03579     GLenum* pFormat
03580 )
03581 {
03582     // ファイルを開く
03583     std::ifstream file{ name, std::ios::binary };
03584
03585     // ファイルが開けなかったら戻る

```

```
03586 if (file.fail()) return false;
03587 // ヘッダを読み込む
03588 unsigned char header[18];
03589 file.read(reinterpret_cast<char*>(header), sizeof header);
03590
03591 // ヘッダの読み込みに失敗したら戻る
03592 if (file.bad())
03593 {
03594     file.close();
03595     return false;
03596 }
03597
03598 // 深度
03599 const int depth{ header[16] / 8 };
03600 switch (depth)
03601 {
03602     case 1:
03603         *pFormat = GL_RED;
03604         break;
03605     case 2:
03606         *pFormat = GL_RG;
03607         break;
03608     case 3:
03609         *pFormat = GL_RGB;
03610         break;
03611     case 4:
03612         *pFormat = GL_RGBA;
03613         break;
03614     default:
03615         // 取り扱えないフォーマットだったら戻る
03616         file.close();
03617         return false;
03618     }
03619
03620 // 画像の縦横の画素数
03621 *pWidth = header[13] << 8 | header[12];
03622 *pHeight = header[15] << 8 | header[14];
03623
03624 // データサイズ
03625 const int size{ *pWidth * *pHeight * depth };
03626 if (size < 2) return false;
03627
03628 // 読み込みに使うメモリを確保する
03629 image.resize(size);
03630
03631 // データを読み込む
03632 if (header[2] & 8)
03633 {
03634     // RLE
03635     int p{ 0 };
03636     char c;
03637     while (file.get(c))
03638     {
03639         if (c & 0x80)
03640         {
03641             // run-length packet
03642             const int count{ (c & 0x7f) + 1 };
03643             if (p + depth * count > size) break;
03644             char temp[4];
03645             file.read(temp, depth);
03646             for (int i = 0; i < count; ++i)
03647             {
03648                 for (int j = 0; j < depth;) image[p++] = temp[j++];
03649             }
03650         }
03651     }
03652     else
03653     {
03654         // raw packet
03655         const int count{ (c + 1) * depth };
03656         if (p + count > size) break;
03657         file.read(reinterpret_cast<char*>(image.data() + p), count);
03658         p += count;
03659     }
03660 }
03661 }
03662 else
03663 {
03664     // 非圧縮
03665     file.read(reinterpret_cast<char*>(image.data()), size);
03666 }
03667 // 読み込みに失敗したら戻る
03668 if (file.bad())
03669 {
03670     file.close();
03671     return false;
03672 }
```

```

03673 }
03674 // ファイルを閉じる
03675 file.close();
03677
03678 // ファイルの読み込みに成功した
03679 return true;
03680 }
03681
03682 //
03683 // テクスチャを作成して画像を読み込む
03684 //
03685 // image 画像データ, nullptr ならメモリの確保だけを行う
03686 // width 画像の横の画素数
03687 // height 画像の縦の画素数
03688 // format 画像データのフォーマット
03689 // type 画像のデータ型
03690 // internal テクスチャの内部フォーマット
03691 // wrap テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE
03692 // swizzle true ならテクスチャの赤と青を入れ替える, デフォルトは true
03693 // 戻り値 テクスチャ名
03694 //
03695 GLuint gg::ggLoadTexture(
03696     const GLvoid* image,
03697     GLsizei width,
03698     GLsizei height,
03699     GLenum format,
03700     GLenum type,
03701     GLenum internal,
03702     GLenum wrap,
03703     bool swizzle
03704 )
03705 {
03706 // テクスチャオブジェクト
03707 const GLuint tex{ [] { glGenTextures(1, &tex); return tex; } () };
03708 glBindTexture(GL_TEXTURE_2D, tex);
03709
03710 // アルファチャンネルがついていれば 4 バイト境界に設定する
03711 glPixelStorei(GL_UNPACK_ALIGNMENT, format == GL_RGBA ? 4 : 1);
03712
03713 // テクスチャを割り当てる
03714 glTexImage2D(GL_TEXTURE_2D, 0, internal, width, height, 0, format, type, image);
03715
03716 // バイナリ (ミップマップなし), エッジでクランプ
03717 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
03718 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
03719 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap);
03720 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);
03721
03722 if (swizzle)
03723 {
03724 // テクスチャのサンプリング時に赤と青を入れ替える
03725 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, GL_BLUE);
03726 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, GL_RED);
03727 }
03728
03729 // テクスチャ名を返す
03730 return tex;
03731 }
03732
03733 //
03734 // テクスチャを作成して TGA フォーマットの画像ファイルを読み込む
03735 //
03736 // name TGA ファイル名
03737 // pWidth 読みだした画像ファイルの横の画素数の格納先のポインタ (nullptr なら格納しない)
03738 // pHeight 読みだした画像ファイルの縦の画素数の格納先のポインタ (nullptr なら格納しない)
03739 // internal テクスチャの内部フォーマット, 0 なら外部フォーマットに合わせる.
03740 // wrap テクスチャのラッピングモード, デフォルトは GL_CLAMP_TO_EDGE
03741 // 戻り値 テクスチャ名
03742 //
03743 GLuint gg::ggLoadImage(
03744     const std::string& name,
03745     GLsizei* pWidth,
03746     GLsizei* pHeight,
03747     GLenum internal,
03748     GLenum wrap
03749 )
03750 {
03751 // 画像データ
03752 std::vector<GLubyte> image;
03753
03754 // 画像サイズ
03755 GLsizei width, height;
03756
03757 // 画像フォーマット
03758 GLenum format;
03759

```

```
03760 // 画像を読み込む
03761 ggReadImage(name, image, &width, &height, &format);
03762
03763 // 画像が読み込めなかつたら戻る
03764 if (image.empty()) return 0;
03765
03766 // internal == 0 なら内部フォーマットを読み込んだファイルに合わせる
03767 if (internal == 0) internal = format;
03768
03769 // テクスチャに読み込む
03770 const GLuint tex(ggLoadTexture(image.data(), width, height,
03771 format, GL_UNSIGNED_BYTE, internal, wrap, true));
03772
03773 // 画像サイズを返す
03774 if (pWidth) *pWidth = width;
03775 if (pHeight) *pHeight = height;
03776
03777 // テクスチャ名を返す
03778 return tex;
03779 }
0380
0381 //
0382 // 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
0383 //
0384 // width 高さマップのグレースケール画像 hmap の横の画素数
0385 // height 高さマップのグレースケール画像のデータ hmap の縦の画素数
0386 // stride データの間隔
0387 // hmap グレースケール画像のデータ
0388 // nz 法線の z 成分の割合
0389 // internal テクスチャの内部フォーマット
0390 // nmap 法線マップを格納する vector
0391 //
0392 void gg::ggCreateNormalMap(
0393     const GLubyte* hmap,
0394     GLsizei width,
0395     GLsizei height,
0396     GLenum format,
0397     GLfloat nz,
0398     GLenum internal,
0399     std::vector<GgVector>& nmap
0400 )
0401 {
0402     // メモリサイズ
0403     const GLsizei size{ width * height };
0404
0405     // 法線マップのメモリを確保する
0406     nmap.resize(size);
0407
0408     // 画素のバイト数
0409     GLint stride;
0410     switch (format)
0411     {
0412         case GL_RED:
0413             stride = 1;
0414             break;
0415         case GL_RG:
0416             stride = 2;
0417             break;
0418         case GL_RGB:
0419             stride = 3;
0420             break;
0421         case GL_RGBA:
0422             stride = 4;
0423             break;
0424         default:
0425             stride = 1;
0426             break;
0427     }
0428
0429     // 法線マップの作成
0430     for (GLsizei i = 0; i < size; ++i)
0431     {
0432         const int x{ i % width };
0433         const int y{ i - x };
0434         const int u0{ (y + (x - 1 + width) % width) * stride };
0435         const int u1{ (y + (x + 1) % width) * stride };
0436         const int v0{ ((y - width + size) % size + x) * stride };
0437         const int v1{ ((y + width) % size + x) * stride };
0438
0439         // 隣接する画素との値の差を法線の成分に用いる
0440         nmap[i][0] = static_cast<GLfloat>(hmap[u1] - hmap[u0]);
0441         nmap[i][1] = static_cast<GLfloat>(hmap[v1] - hmap[v0]);
0442         nmap[i][2] = nz;
0443         nmap[i][3] = hmap[i * stride];
0444
0445         // 法線ベクトルを正規化する
0446         ggNormalize3(nmap[i]);
0447     }
0448 }
```

```

03847    }
03848
03849 // 内部フォーマットが浮動小数点テクスチャでなければ [0,1] に正規化する
03850 if (
03851     internal != GL_RGB16F &&
03852     internal != GL_RGBA16F &&
03853     internal != GL_RGB32F &&
03854     internal != GL_RGBA32F
03855 )
03856 {
03857     for (GLsizei i = 0; i < size; ++i)
03858     {
03859         nmap[i][0] = nmap[i][0] * 0.5f + 0.5f;
03860         nmap[i][1] = nmap[i][1] * 0.5f + 0.5f;
03861         nmap[i][2] = nmap[i][2] * 0.5f + 0.5f;
03862         nmap[i][3] *= 0.0039215686f; // == 1/255
03863     }
03864 }
03865 }
03866
03867 //
03868 // TGA 画像ファイルの高さマップ読み込んで法線マップのテクスチャを作成する
03869 //
03870 // name TGA ファイル名
03871 // nz 作成した法線の z 成分の割合
03872 // pWidth 読み込んだ画像の横の画素数の格納先のポインタ (nullptr なら格納しない)
03873 // pHeight 読み込んだ画像の縦の画素数の格納先のポインタ (nullptr なら格納しない)
03874 // internal テクスチャの内部フォーマット
03875 // 戻り値 テクスチャ名
03876 //
03877 GLuint gg::ggLoadHeight(
03878     const std::string& name,
03879     GLfloat nz,
03880     GLsizei* pWidth,
03881     GLsizei* pHeight,
03882     GLenum internal
03883 )
03884 {
03885     // 画像データ
03886     std::vector<GLubyte> hmap;
03887
03888     // 画像サイズ
03889     GLsizei width, height;
03890
03891     // 画像フォーマット
03892     GLenum format;
03893
03894     // 高さマップの画像を読み込む
03895     ggReadImage(name, hmap, &width, &height, &format);
03896
03897     // 画像が読み込めなかつたら戻る
03898     if (hmap.empty()) return 0;
03899
03900     // 法線マップ
03901     std::vector<GgVector> nmap;
03902
03903     // 法線マップを作成する
03904     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
03905
03906     // 画像サイズを返す
03907     if (pWidth) *pWidth = width;
03908     if (pHeight) *pHeight = height;
03909
03910     // テクスチャを作成して返す
03911     return ggLoadTexture(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal, GL_REPEAT);
03912 }
03913
03914 //
03915 // テクスチャの赤と青を交換する
03916 //
03917 void gg::GgTexture::swapRandB(bool swap) const
03918 {
03919     const auto swizzle
03920     {
03921         swap ?
03922             std::pair<GLint, GLint>{ GL_BLUE, GL_RED } :
03923             std::pair<GLint, GLint>{ GL_RED, GL_BLUE }
03924     };
03925
03926     glBindTexture(GL_TEXTURE_2D, texture);
03927     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, swizzle.first);
03928     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, swizzle.second);
03929     glBindTexture(GL_TEXTURE_2D, 0);
03930 }
03931
03932 //
03933 // TGA フォーマットの画像ファイルを読み込んでカラーのテクスチャを作成する

```

```
03934 //  
03935 // name 読み込むファイル名  
03936 // internal glTexImage2D() に指定するテクスチャの内部フォーマット. 0 なら外部フォーマットに合わせる  
03937 // 戻り値 テクスチャの作成に成功すれば true, 失敗すれば false  
03938 //  
03939 void gg::GgColorTexture::load(  
03940     const std::string& name,  
03941     GLenum internal,  
03942     GLenum wrap  
03943 )  
03944 {  
03945     // 画像データ  
03946     std::vector<GLubyte> image;  
03947  
03948     // 画像サイズ  
03949     GLsizei width, height;  
03950  
03951     // 画像フォーマット  
03952     GLenum format;  
03953  
03954     // 画像を読み込む  
03955     ggReadImage(name, image, &width, &height, &format);  
03956  
03957     // internal == 0 なら内部フォーマットを読み込んだファイルに合わせる  
03958     if (internal == 0) internal = format;  
03959  
03960     // テクスチャを作成する  
03961     texture = std::make_shared<GgTexture>(image.data(), width, height,  
03962         format, GL_UNSIGNED_BYTE, internal, wrap, true);  
03963 }  
03964  
03965 //  
03966 // TGA フォーマットの画像ファイルから高さマップ読み込んで法線マップのテクスチャを作成する  
03967 //  
03968 // name 画像ファイル名  
03969 // width テクスチャとして用いる画像データの横幅  
03970 // height テクスチャとして用いる画像データの高さ  
03971 // format テクスチャとして用いる画像データのフォーマット (GL_RED, GL_RG, GL_RGB, GL_RGBA)  
03972 // nz 法線マップの z 成分の値  
03973 // internal テクスチャの内部フォーマット  
03974 //  
03975 void gg::GgNormalTexture::load(  
03976     const std::string& name,  
03977     GLfloat nz,  
03978     GLenum internal  
03979 )  
03980 {  
03981     // 高さマップ  
03982     std::vector<GLubyte> hmap;  
03983  
03984     // 画像サイズ  
03985     GLsizei width, height;  
03986  
03987     // 画像フォーマット  
03988     GLenum format;  
03989  
03990     // 高さマップの画像を読み込む  
03991     ggReadImage(name, hmap, &width, &height, &format);  
03992  
03993     // 法線マップ  
03994     std::vector<GgVector> nmap;  
03995  
03996     // 法線マップを作成する  
03997     ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);  
03998 }  
03999  
04000  
04001  
04002 //  
04003 // OBJ ファイルの読み込みに使うデータ型と関数  
04004 //  
04005 namespace gg  
04006 {  
04007     // GLfloat 型の 2 要素のベクトル  
04008     using vec2 = std::array<GLfloat, 2>;  
04009  
04010     // GLfloat 型の 3 要素のベクトル  
04011     using vec3 = std::array<GLfloat, 3>;  
04012  
04013     // 三角形データ  
04014     struct fidx  
04015     {  
04016         GLuint p[3];           // 頂点座標番号  
04017         GLuint n[3];           // 頂点法線番号  
04018         GLuint t[3];           // テクスチャ座標番号  
04019         bool smooth;          // スムーズシェーディングの有無  
04020     };  
04021
```

```

04022 // ポリゴン グループ
04023 struct fgrp
04024 {
04025     GLuint nextgroup; // 次のポリゴン グループの最初の三角形番号
04026     GLuint mtlno; // このポリゴン グループの材質番号
04027
04028     // コンストラクタ
04029     fgrp(GLuint nextgroup, GLuint mtlno) :
04030         nextgroup(nextgroup),
04031         mtlno(mtlno)
04032     {
04033     }
04034 };
04035
04036 // デフォルトの材質
04037 constexpr GgSimpleShader::Material defaultMaterial
04038 {
04039     { 0.1f, 0.1f, 0.1f, 1.0f },
04040     { 0.6f, 0.6f, 0.6f, 0.0f },
04041     { 0.3f, 0.3f, 0.3f, 0.0f },
04042     60.0f
04043 };
04044
04045 // デフォルトの材質名
04046 constexpr char defaultMaterialName[] = "_default_";
04047
04048 //
04049 // Alias OBJ 形式の MTL ファイルを読み込む
04050 //
04051 // mtlpath MTL ファイルのパス名
04052 // mtl 読み込んだ材質名をキー、材質番号を値にした map
04053 // material 材質データ
04054 //
04055 static bool ggLoadMtl(const std::string& mtlpath,
04056     std::map<std::string, GLuint>& mtl,
04057     std::vector<GgSimpleShader::Material>& material)
04058 {
04059     // MTL ファイルが無ければ戻る
04060     std::ifstream mtlfile{ mtlpath, std::ios::binary };
04061     if (!mtlfile)
04062     {
04063 #if defined(DEBUG)
04064         std::cerr << "Warning: Can't open MTL file: " << mtlpath << std::endl;
04065 #endif
04066     return false;
04067     }
04068
04069 // 一行読み込み用のバッファ
04070 std::string mtlline;
04071
04072 // 材質名 (ループの外に置く)
04073 std::string mtlname{ defaultMaterialName };
04074
04075 // 現在の材質番号を登録する
04076 mtl[mtlname] = static_cast<GLuint>(material.size());
04077
04078 // 現在の材質にデフォルトの材質を設定する
04079 material.emplace_back(defaultMaterial);
04080
04081 // 材質データを読み込む
04082 while (std::getline(mtlfile, mtlline))
04083 {
04084     // 空行は読み飛ばす
04085     if (mtlline == "") continue;
04086
04087     // 最後の文字が '\r' なら
04088     if (*(mtlline.end() - 1) == '\r')
04089     {
04090         // 最後の文字を削除する
04091         mtlline.erase(mtlline.end() - 1, mtlline.end());
04092
04093         // 空行になったら読み飛ばす
04094         if (mtlline == "") continue;
04095     }
04096
04097     // 読み込んだ行を文字列ストリームにする
04098     std::istringstream mtlstr{ mtlline };
04099
04100     // オペレータ
04101     std::string mtlop;
04102
04103     // 文字列ストリームから材質パラメータの種類を取り出す
04104     mtlstr >> mtlop;
04105
04106     // '#' で始まる場合はコメントとして行末まで読み飛ばす
04107     if (mtlop[0] == '#') continue;
04108

```

```
04109     if (mtlop == "newmtl")
04110     {
04111         // 新規作成する材質名を取り出す
04112         mtlstr >> mtlname;
04113
04114         // 材質名が既に存在するかどうか調べる
04115         const auto m{ mtl.find(mtlname) };
04116         if (m == mtl.end())
04117         {
04118             // 存在しないので新規作成する材質の番号をその材質名に割り当てる
04119             mtl[mtlname] = static_cast<GLuint>(material.size());
04120
04121             // 新規作成する材質にデフォルトの材質を設定しておく
04122             material.emplace_back(defaultMaterial);
04123         }
04124
04125 #if defined(DEBUG)
04126     std::cerr << "newmtl: " << mtlname << std::endl;
04127 #endif
04128 }
04129 else if (mtlop == "Ka")
04130 {
04131     // 環境光の反射係数を登録する
04132     mtlstr
04133         >> material.back().ambient[0]
04134         >> material.back().ambient[1]
04135         >> material.back().ambient[2];
04136 }
04137 else if (mtlop == "Kd")
04138 {
04139     // 拡散反射係数を登録する
04140     mtlstr
04141         >> material.back().diffuse[0]
04142         >> material.back().diffuse[1]
04143         >> material.back().diffuse[2];
04144 }
04145 else if (mtlop == "Ks")
04146 {
04147     // 鏡面反射係数を登録する
04148     mtlstr
04149         >> material.back().specular[0]
04150         >> material.back().specular[1]
04151         >> material.back().specular[2];
04152 }
04153 else if (mtlop == "Ns")
04154 {
04155     // 輝き係数を登録する
04156     GLfloat shininess;
04157     mtlstr >> shininess;
04158     material.back().shininess = shininess * 0.1f;
04159 }
04160 else if (mtlop == "d")
04161 {
04162     // 不透明度を登録する
04163     mtlstr >> material.back().ambient[3];
04164 }
04165 }
04166
04167 // MTL ファイルの読み込みに失敗したら戻る
04168 if (mtlfile.bad())
04169 {
04170 #if defined(DEBUG)
04171     std::cerr << "Warning: Can't read MTL file: " << mtlpath << std::endl;
04172 #endif
04173
04174 // MTL ファイルを閉じる
04175 mtlfile.close();
04176
04177 // MTL ファイルが読み込みに失敗したので戻る
04178 return false;
04179 }
04180
04181 // MTL ファイルを閉じる
04182 mtlfile.close();
04183
04184 // MTL ファイルの読み込みに成功した
04185 return true;
04186 }
04187
04188 //
04189 // Alias OBJ 形式のファイルを解析する
04190 //
04191 //     name Alias OBJ 形式のファイルのファイル名
04192 //     group 同じ材質を割り当てるポリゴングループ
04193 //     mtl 読み込んだ材質名をキーにした map
04194 //     pos 頂点の位置
04195 //     norm 頂点の法線
```

```

04196 //  tex 頂点のテクスチャ座標
04197 //  face 三角形のデータ
04198 //
04199 static bool ggParseObj(
04200     const std::string& name,
04201     std::vector<fgrp>& group,
04202     std::vector<GgSimpleShader::Material>& material,
04203     std::vector<vec3>& pos,
04204     std::vector<vec3>& norm,
04205     std::vector<vec2>& tex,
04206     std::vector<fidx>& face,
04207     bool normalize
04208 )
04209 {
04210     // ファイルパスからディレクトリ名を取り出す
04211     const std::string path{ name };
04212     const size_t base{ path.find_last_of("//\\") };
04213     const std::string dirname{ (base == std::string::npos) ? "" : path.substr(0, base + 1) };
04214
04215     // OBJ ファイルを読み込む
04216     std::ifstream file{ path };
04217
04218     // 読み込みに失敗したら戻る
04219     if (file.fail())
04220     {
04221 #if defined(DEBUG)
04222         std::cerr << "Error: Can't open OBJ file: " << path << std::endl;
04223 #endif
04224     }
04225
04226     // ポリゴングループの最初の三角形番号
04227     GLsizei startgroup(static_cast<GLsizei>(group.size()));
04228
04229     // スムーズシェーディングのスイッチ
04230     bool smooth{ false };
04231
04232     // 材質のテーブル
04233     std::map<std::string, GLuint> mtl;
04234
04235     // 現在の材質名 (ループの外で宣言する)
04236     std::string mtlname;
04237
04238     // 座標値の最小値・最大値
04239     vec3 bmin{ FLT_MAX }, bmax{ -FLT_MAX };
04240
04241     // 一行読み込み用のバッファ
04242     std::string line;
04243
04244     // データの読み込み
04245     while (std::getline(file, line))
04246     {
04247         // 空行は読み飛ばす
04248         if (line == "") continue;
04249
04250         // 最後の文字が '\r' なら
04251         if (*(line.end() - 1) == '\r')
04252         {
04253             // 最後の文字を削除する
04254             line.erase(line.end() - 1, line.end());
04255
04256             // 空行になつたら読み飛ばす
04257             if (line == "") continue;
04258         }
04259
04260         // 一行を文字列ストリームに入れる
04261         std::istringstream str(line);
04262
04263         // 最初のトークンを命令 (op) とみなす
04264         std::string op;
04265         str >> op;
04266
04267         if (op[0] == '#') continue;
04268
04269         if (op == "v")
04270         {
04271             // 頂点位置
04272             vec3 v;
04273
04274             // 頂点位置はスペースで区切られている
04275             str >> v[0] >> v[1] >> v[2];
04276
04277             // 頂点位置を記録する
04278             pos.emplace_back(v);
04279
04280             // 頂点位置の最小値と最大値を求める (AABB)
04281             for (int i = 0; i < 3; ++i)

```

```
04283     {
04284         bmin[i] = std::min(bmin[i], v[i]);
04285         bmax[i] = std::max(bmax[i], v[i]);
04286     }
04287 }
04288 else if (op == "vt")
04289 {
04290     // テクスチャ座標
04291     vec2 t;
04292
04293     // 頂点位置はスペースで区切られている
04294     str >> t[0] >> t[1];
04295
04296     // テクスチャ座標を記録する
04297     tex.emplace_back(t);
04298 }
04299 else if (op == "vn")
04300 {
04301     // 頂点法線
04302     vec3 n;
04303
04304     // 頂点法線はスペースで区切られている
04305     str >> n[0] >> n[1] >> n[2];
04306
04307     // 頂点法線を記録する
04308     norm.emplace_back(n);
04309 }
04310 else if (op == "f")
04311 {
04312     // 三角形データ
04313     fidx f;
04314
04315     // スムースシェーディング
04316     f.smooth = smooth;
04317
04318     // 三頂点のそれぞれについて
04319     for (int i = 0; i < 3; ++i)
04320     {
04321         // 1項目取り出す
04322         std::string s;
04323         str >> s;
04324
04325         // 文字の位置
04326         auto c{ s.begin() };
04327
04328         // テクスチャ座標と法線の番号は未定義を表す 0 にしておく
04329         f.p[i] = f.t[i] = f.n[i] = 0;
04330
04331         // 項目の最初の要素は頂点座標番号
04332         while (c != s.end() && isdigit(*c)) f.p[i] = f.p[i] * 10 + *c++ - '0';
04333         if (c == s.end()) break;
04334         if (*c++ != '/') continue;
04335
04336         // 二つ目の項目はテクスチャ座標
04337         while (c != s.end() && isdigit(*c)) f.t[i] = f.t[i] * 10 + *c++ - '0';
04338         if (c == s.end()) break;
04339         if (*c++ != '/') continue;
04340
04341         // 三つ目の項目は法線番号
04342         while (c != s.end() && isdigit(*c)) f.n[i] = f.n[i] * 10 + *c++ - '0';
04343     }
04344
04345     // 三角形データを登録する
04346     face.emplace_back(f);
04347 }
04348 else if (op == "s")
04349 {
04350     // '1' だったらスムースシェーディング有効
04351     std::string s;
04352     str >> s;
04353     smooth = s == "1";
04354 }
04355 else if (op == "usemtl")
04356 {
04357     // 次のポリゴン グループの最初の三角形番号
04358     const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04359
04360     // ポリゴン グループに三角形が存在すれば
04361     if (nextgroup > startgroup)
04362     {
04363         // ポリゴン グループの三角形数と材質番号を記録する
04364         group.emplace_back(nextgroup, mtl[mtlname]);
04365
04366         // 次のポリゴン グループの開始番号を保存しておく
04367         startgroup = nextgroup;
04368     }
04369 }
```

```

04370     // 次に usemtl が来るまで材質名を保持する
04371     str >> mtlname;
04372
04373     // 材質の存在チェック
04374     if (mtl.find(mtlname) == mtl.end())
04375     {
04376 #if defined(DEBUG)
04377         std::cerr << "Warning: Undefined material: " << mtlname << std::endl;
04378 #endif
04379
04380     // デフォルトの材質を割り当てておく
04381     mtlname = defaultMaterialName;
04382 }
04383 #if defined(DEBUG)
04384     else std::cerr << "usemtl: " << mtlname << std::endl;
04385 #endif
04386 }
04387 else if (op == "mtllib")
04388 {
04389     // MTL ファイルのパス名を作る
04390     str >> std::ws;
04391     std::string mtlpath;
04392     std::getline(str, mtlpath);
04393
04394     // MTL ファイルを読み込む
04395     ggLoadMtl(dirname + mtlpath, mtl, material);
04396 }
04397
04398 // OBJ ファイルの読み込みに失敗したら戻る
04399 if (file.bad())
04400 {
04401 #if defined(DEBUG)
04402     std::cerr << "Error: Can't read OBJ file: " << path << std::endl;
04403 #endif
04404     file.close();
04405     return false;
04406 }
04407
04408 // ファイルを閉じる
04409 file.close();
04410
04411 // 最後のポリゴングループの次の三角形番号
04412 const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
04413 if (nextgroup > startgroup)
04414 {
04415     // 最後のポリゴングループの三角形数と材質を記録する
04416     group.emplace_back(nextgroup, static_cast<GLuint>(mtl[mtlname]));
04417 }
04418
04419 // スムーズシェーディングしない三角形の頂点を追加する
04420 for (auto& f : face)
04421 {
04422     if (!f.smooth)
04423     {
04424         // 三頂点のそれぞれについて
04425         for (int i = 0; i < 3; ++i)
04426         {
04427             // 新しい頂点座標を生成する (std::array の要素は emplace_back できない)
04428             pos.emplace_back(pos[f.p[i] - 1]);
04429             f.p[i] = static_cast<GLuint>(pos.size());
04430
04431             if (f.t[i] > 0)
04432             {
04433                 // 新しいテクスチャ座標を生成する
04434                 tex.emplace_back(tex[f.t[i] - 1]);
04435                 f.t[i] = static_cast<GLuint>(tex.size());
04436             }
04437
04438             if (f.n[i] > 0)
04439             {
04440                 // 新しい法線を生成する
04441                 norm.emplace_back(norm[f.n[i] - 1]);
04442                 f.n[i] = static_cast<GLuint>(norm.size());
04443             }
04444         }
04445     }
04446 }
04447 }
04448
04449 // 法線データがなければ算出しておく
04450 if (norm.empty())
04451 {
04452     // 法線データ数の初期値は頂点数と同じでスムーズシェーディングのために初期値は 0
04453     norm.resize(pos.size(), { 0.0f, 0.0f, 0.0f });
04454
04455     // 面の法線の算出と頂点法線の算出
04456     for (auto& f : face)

```

```

04457     {
04458         // 顶点座標番号
04459         const GLuint v0{ f.p[0] - 1 };
04460         const GLuint v1{ f.p[1] - 1 };
04461         const GLuint v2{ f.p[2] - 1 };
04462
04463         // v1 - v0, v2 - v0 を求める
04464         const GLfloat d1[] { pos[v1][0] - pos[v0][0], pos[v1][1] - pos[v0][1], pos[v1][2] - pos[v0][2] };
04465         const GLfloat d2[] { pos[v2][0] - pos[v0][0], pos[v2][1] - pos[v0][1], pos[v2][2] - pos[v0][2] };
04466
04467         // 外積により面法線を求める
04468         vec3 n;
04469         ggCross(n.data(), d1, d2);
04470
04471         if (f.smooth)
04472         {
04473             // スムースシェーディングを行うときは
04474             for (int i = 0; i < 3; ++i)
04475             {
04476                 // 面法線を頂点法線に積算する
04477                 norm[v0][i] += n[i];
04478                 norm[v1][i] += n[i];
04479                 norm[v2][i] += n[i];
04480
04481                 // 面の各頂点の法線番号は頂点番号と同じにする
04482                 f.n[i] = f.p[i];
04483             }
04484         }
04485         else
04486         {
04487             // 面法線を最初の頂点に保存する
04488             norm[v0] = n;
04489             f.n[0] = f.p[0];
04490
04491             // 2 頂点追加
04492             for (int i = 1; i < 3; ++i)
04493             {
04494                 norm.emplace_back(n);
04495                 f.n[i] = static_cast<GLuint>(norm.size());
04496             }
04497         }
04498     }
04499
04500     // 頂点の法線ベクトルを正規化する
04501     for (auto& n : norm) ggNormalize3(n.data());
04502
04503
04504     // 図形の正規化
04505     if (normalize)
04506     {
04507         // 図形の大きさ
04508         const GLfloat sx{ bmax[0] - bmin[0] };
04509         const GLfloat sy{ bmax[1] - bmin[1] };
04510         const GLfloat sz{ bmax[2] - bmin[2] };
04511
04512         // 図形のスケール
04513         GLfloat s{ sx };
04514         if (sy > s) s = sy;
04515         if (sz > s) s = sz;
04516         const GLfloat scale{ s != 0.0f ? 2.0f / s : 1.0f };
04517
04518         // 図形の中心位置
04519         const GLfloat cx{ (bmax[0] + bmin[0]) * 0.5f };
04520         const GLfloat cy{ (bmax[1] + bmin[1]) * 0.5f };
04521         const GLfloat cz{ (bmax[2] + bmin[2]) * 0.5f };
04522
04523         // 図形の大きさと位置を正規化する
04524         for (auto& p : pos)
04525         {
04526             p[0] = (p[0] - cx) * scale;
04527             p[1] = (p[1] - cy) * scale;
04528             p[2] = (p[2] - cz) * scale;
04529         }
04530     }
04531
04532 #if defined(DEBUG)
04533     std::cerr
04534     << "[" << name << "]\n(Parsed) Group: " << group.size() << ", Material: " << mtl.size()
04535     << ", Pos: " << pos.size() << ", Norm: " << norm.size() << ", Tex: " << tex.size()
04536     << ", Face: " << face.size() << "\n";
04537 #endif
04538
04539     // OBJ ファイルの読み込み成功
04540     return true;
04541 }

```

```

04542 }
04544
04545 // 三角形分割された Alias OBJ 形式のファイルと MTL ファイルを読み込む (Arrays 形式)
04547 //
04548 //   name 読み込むOBJ ファイル名
04549 //   group 読み込んだデータの各ポリゴングループの最初の三角形番号と三角形数
04550 //   material 読み込んだデータのポリゴングループごとの材質
04551 //   vert 読み込んだデータの頂点属性
04552 //   normalize true ならサイズを正規化する
04553 //   戻り値 読み込みに成功したら true
04554 //
04555 bool gg::ggLoadSimpleObj(const std::string& name,
04556   std::vector<std::array<GLuint, 3>>& group,
04557   std::vector<GgSimpleShader::Material>& material,
04558   std::vector<GgVertex>& vert,
04559   bool normalize)
04560 {
04561   // 読み込み用の一時記憶領域
04562   std::vector<fgroup> tgroup;
04563   std::vector<vec3> tpos;
04564   std::vector<vec3> tnorm;
04565   std::vector<vec2> ttex;
04566   std::vector<fidx> tface;
04567
04568   // OBJ ファイルを解析する
04569   if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04570
04571   // 頂点属性データのメモリを確保する
04572   vert.reserve(vert.size() + tface.size() * 3);
04573
04574   // ポリゴングループデータのメモリを確保する
04575   group.reserve(group.size() + tgroup.size());
04576   material.reserve(material.size() + tgroup.size());
04577
04578   // ポリゴングループの最初の三角形番号
04579   GLuint startgroup{ 0 };
04580
04581   // ポリゴングループデータの作成
04582   for (auto& g : tgroup)
04583   {
04584     // このポリゴングループの最初の頂点番号と頂点数・材質番号
04585     std::array<GLuint, 3> v;
04586
04587     // このポリゴングループの最初の頂点番号を保存する
04588     v[0] = static_cast<GLuint>(vert.size());
04589
04590     // 三角形ごとの頂点データの作成
04591     for (GLuint j = startgroup; j < g.nextgroup; ++j)
04592     {
04593       // 処理対象の三角形
04594       auto& f = tface[j];
04595
04596       // 三頂点のそれぞれについて
04597       for (int i = 0; i < 3; ++i)
04598       {
04599         // テクスチャ座標
04600         vec2 tex{ 0.0f, 0.0f };
04601         if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04602
04603         // 頂点法線
04604         vec3 norm{ 0.0f, 0.0f, 0.0f };
04605         if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04606
04607         // 頂点属性の追加
04608         vert.emplace_back(tpos[f.p[i] - 1].data(), norm.data());
04609       }
04610     }
04611
04612     // このポリゴングループの頂点数を保存する
04613     v[1] = static_cast<GLuint>(vert.size()) - v[0];
04614     v[2] = g.mtlno;
04615
04616     // このポリゴングループの最初の頂点番号と頂点数・材質番号を登録する
04617     group.emplace_back(v);
04618
04619     // 次のポリゴングループの最初の三角形番号を求める
04620     startgroup = g.nextgroup;
04621   }
04622
04623 #if defined(DEBUG)
04624   std::cerr
04625     << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04626     << ", Vertex: " << vert.size() << "\n";
04627 #endif
04628
04629   // OBJ ファイルの読み込み成功

```

```

04630     return true;
04631 }
04632
04633 //
04634 // 三角形分割された Alias OBJ 形式のファイルと MTL ファイルを読み込む (Elements 形式)
04635 //
04636 //   name 読み込むOBJ ファイル名
04637 //   group 読み込んだデータの各ポリゴングループの最初の三角形番号と三角形数
04638 //   material 読み込んだデータのポリゴングループごとの材質
04639 //   vert 読み込んだデータの頂点属性
04640 //   face 読み込んだデータの三角形の頂点インデックス
04641 //   normalize true ならサイズを正規化する
04642 //   戻り値 読み込みに成功したら true
04643 //
04644 bool gg::ggLoadSimpleObj(const std::string& name,
04645     std::vector<std::array<GLuint, 3>>& group,
04646     std::vector<GgSimpleShader::Material>& material,
04647     std::vector<GgVertex>& vert,
04648     std::vector<GLuint>& face,
04649     bool normalize)
04650 {
04651     // 読み込み用の一時記憶領域
04652     std::vector<fgrp> tgroup;
04653     std::vector<vec3> tpos;
04654     std::vector<vec3> tnorm;
04655     std::vector<vec2> ttex;
04656     std::vector<fidx> tface;
04657
04658     // OBJ ファイルを解析する
04659     if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;
04660
04661     // 頂点属性データの最初の頂点番号
04662     const GLuint vertbase{ static_cast<GLuint>(vert.size()) };
04663
04664     // 頂点属性データのメモリを確保する
04665     vert.resize(vertbase + tpos.size());
04666
04667     // 三角形データのメモリを確保する
04668     face.reserve(face.size() + tface.size());
04669
04670     // ポリゴングループデータのメモリを確保する
04671     group.reserve(group.size() + tgroup.size());
04672     material.reserve(material.size() + tgroup.size());
04673
04674     // ポリゴングループの最初の三角形番号
04675     GLuint startgroup{ 0 };
04676
04677     // ポリゴングループデータの作成
04678     for (auto& g : tgroup)
04679     {
04680         // このポリゴングループの最初の頂点番号
04681         const GLuint first{ static_cast<GLuint>(face.size()) };
04682
04683         // 三角形ごとの頂点データの作成
04684         for (GLuint j = startgroup; j < g.nextgroup; ++j)
04685         {
04686             // 処理対象の三角形
04687             auto& f{ tface[j] };
04688
04689             // 三頂点のそれぞれについて
04690             for (int i = 0; i < 3; ++i)
04691             {
04692                 // 追加する三角形データの頂点番号
04693                 const GLuint q{ f.p[i] - 1 + vertbase };
04694
04695                 // 三角形データの追加
04696                 face.emplace_back(q);
04697
04698                 // テクスチャ座標番号
04699                 vec2 tex{ 0.0f, 0.0f };
04700                 if (f.t[i] > 0) tex = ttex[f.t[i] - 1];
04701
04702                 // 頂点法線番号
04703                 vec3 norm{ 0.0f, 0.0f, 0.0f };
04704                 if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];
04705
04706                 // 頂点の格納
04707                 vert[q] = GgVertex(tpos[f.p[i] - 1].data(), norm.data());
04708             }
04709         }
04710
04711         // このポリゴングループの最初の三角形番号と三角形数・材質番号を登録する
04712         group.emplace_back(std::array<GLuint, 3>{ first, static_cast<GLuint>(face.size()) - first, g.mtlno });
04713
04714         // 次のポリゴングループの最初の三角形番号を求める
04715         startgroup = g.nextgroup;

```

```

04716  }
04717
04718 #if defined(DEBUG)
04719   std::cerr
04720     << "(Stored) Group: " << group.size() << ", Material: " << material.size()
04721     << ", Vertex: " << vert.size() << ", Face: " << face.size() << "\n";
04722 #endif
04723
04724 // OBJ ファイルの読み込み成功
04725   return true;
04726 }
04727
04728 //
04729 // シエーダオブジェクトのコンパイル結果を表示する
04730 //
04731 static GLboolean printShaderInfoLog(GLuint shader, const std::string& str)
04732 {
04733   // コンパイル結果を取得する
04734   GLint status;
04735   glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
04736 #if defined(DEBUG)
04737   if (status == GL_FALSE) std::cerr << "Compile Error in " << str << std::endl;
04738 #endif
04739
04740   // シエーダのコンパイル時のログの長さを取得する
04741   GLsizei bufSize;
04742   glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &bufSize);
04743
04744   if (bufSize > 1)
04745   {
04746     // シエーダのコンパイル時のログの内容を取得する
04747     std::vector<GLchar> infoLog(bufSize);
04748     GLsizei length;
04749     glGetShaderInfoLog(shader, bufSize, &length, infoLog.data());
04750 #if defined(DEBUG)
04751   std::cerr << infoLog.data();
04752 #endif
04753   }
04754
04755   // コンパイル結果を返す
04756   return static_cast<GLboolean>(status);
04757 }
04758
04759 //
04760 // プログラムオブジェクトのリンク結果を表示する
04761 //
04762 static GLboolean printProgramInfoLog(GLuint program)
04763 {
04764   // リンク結果を取得する
04765   GLint status;
04766   glGetProgramiv(program, GL_LINK_STATUS, &status);
04767 #if defined(DEBUG)
04768   if (status == GL_FALSE) std::cerr << "Link Error." << std::endl;
04769 #endif
04770
04771   // シエーダのリンク時のログの長さを取得する
04772   GLsizei bufSize;
04773   glGetProgramiv(program, GL_INFO_LOG_LENGTH, &bufSize);
04774
04775   // シエーダのリンク時のログの内容を取得する
04776   if (bufSize > 1)
04777   {
04778     std::vector<GLchar> infoLog(bufSize);
04779     GLsizei length;
04780     glGetProgramInfoLog(program, bufSize, &length, infoLog.data());
04781 #if defined(DEBUG)
04782   std::cerr << infoLog.data() << std::endl;
04783 #endif
04784   }
04785
04786   // リンク結果を返す
04787   return static_cast<GLboolean>(status);
04788 }
04789
04790 //
04791 // シエ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
04792 //
04793 //  vsrc パーテックスシェーダのソースプログラムの文字列
04794 //  fsrc フラグメントシェーダのソースプログラムの文字列 (nullptr なら不使用)
04795 //  gsrc ジオメトリシェーダのソースプログラムの文字列 (nullptr なら不使用)
04796 //  nvarying フィードバックする varying 変数の数 (0 なら不使用)
04797 //  varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04798 //  vtext パーテックスシェーダのコンパイル時のメッセージに追加する文字列
04799 //  ftext フラグメントシェーダのコンパイル時のメッセージに追加する文字列
04800 //  gtext ジオメトリシェーダのコンパイル時のメッセージに追加する文字列
04801 //  gtext 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)
04802 //

```

```
04803 GLuint gg::ggCreateShader(
04804     const std::string& vsrc,
04805     const std::string& fsrc,
04806     const std::string& gsrc,
04807     GLint nvarying,
04808     const char* const* varyings,
04809     const std::string& vtext,
04810     const std::string& ftext,
04811     const std::string& gtext)
04812 {
04813     // シェーダプログラムの作成
04814     const GLuint program{ glCreateProgram() };
04815
04816     if (program > 0)
04817     {
04818         bool status = true;
04819
04820         if (!vsrc.empty())
04821         {
04822             // パーテックスシェーダのシェーダオブジェクトを作成する
04823             const GLuint vertShader{ glCreateShader(GL_VERTEX_SHADER) };
04824             const GLchar* vsrcp{ vsrc.c_str() };
04825             glShaderSource(vertShader, 1, &vsrcp, nullptr);
04826             glCompileShader(vertShader);
04827
04828             // パーテックスシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04829             if (printShaderInfoLog(vertShader, vtext))
04830                 glAttachShader(program, vertShader);
04831             else
04832                 status = false;
04833             glDeleteShader(vertShader);
04834         }
04835
04836         if (!fsrc.empty())
04837         {
04838             // フラグメントシェーダのシェーダオブジェクトを作成する
04839             const GLuint fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
04840             const GLchar* fsrcc{ fsrc.c_str() };
04841             glShaderSource(fragShader, 1, &fsrcc, nullptr);
04842             glCompileShader(fragShader);
04843
04844             // フラグメントシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04845             if (printShaderInfoLog(fragShader, ftext))
04846                 glAttachShader(program, fragShader);
04847             else
04848                 status = false;
04849             glDeleteShader(fragShader);
04850         }
04851
04852         if (!gsrc.empty())
04853         {
04854 #if defined(GL_GLES_PROTOTYPES)
04855 # if defined(DEBUG)
04856             std::cerr << gtext << ": The geometry is not supported." << std::endl;
04857             status = false;
04858 # endif
04859 #else
04860             // ジオメトリシェーダのシェーダオブジェクトを作成する
04861             const GLuint geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
04862             const GLchar* gsrcp{ gsrc.c_str() };
04863             glShaderSource(geomShader, 1, &gsrcp, nullptr);
04864             glCompileShader(geomShader);
04865
04866             // ジオメトリシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04867             if (printShaderInfoLog(geomShader, gtext))
04868                 glAttachShader(program, geomShader);
04869             else
04870                 status = false;
04871             glDeleteShader(geomShader);
04872 #endif
04873     }
04874
04875     // 全てのシェーダオブジェクトのコンパイルに成功したら
04876     if (status)
04877     {
04878         // feedback に使う varying 変数を指定する
04879         if (nvarying > 0)
04880             glTransformFeedbackVaryings(program, nvarying, varyings, GL_SEPARATE_ATTRIBS);
04881
04882         // シェーダプログラムをリンクする
04883         glLinkProgram(program);
04884
04885         // リンクに成功したらプログラムオブジェクト名を返す
04886         if (printProgramInfoLog(program) != GL_FALSE) return program;
04887     }
04888 }
```

```

04890 // プログラムオブジェクトが作成できなかった
04891 glDeleteProgram(program);
04892 return 0;
04893 }
04894
04895 //
04896 // シェーダのソースファイルを読み込んだ vector を返す
04897 //
04898 // name ソースファイル名
04899 // src 読み込んだソースファイルの文字列
04900 // 戻り値 読み込みの成功すれば true, 失敗したら false
04901 //
04902 static bool readShaderSource(const std::string& name, std::string& src)
04903 {
04904 // ファイル名が nullptr ならそのまま戻る
04905 if (name.empty()) return true;
04906
04907 // ソースファイルを開く
04908 std::ifstream file{ name, std::ios::binary };
04909 if (file.fail())
04910 {
04911 // ファイルが開けなければエラーで戻る
04912 #if defined(DEBUG)
04913 std::cerr << "Error: Can't open source file: " << name << std::endl;
04914 #endif
04915 return false;
04916 }
04917
04918 // ファイル全体を文字列として読み込む
04919 std::istreambuf_iterator<char> it{ file };
04920 std::istreambuf_iterator<char> last;
04921 src = std::string(it, last);
04922
04923 // ファイルがうまく読み込めなければ戻る
04924 if (file.bad())
04925 {
04926 #if defined(DEBUG)
04927 std::cerr << "Error: Could not read source file: " << name << std::endl;
04928 #endif
04929 file.close();
04930 return false;
04931 }
04932
04933 // ファイルを閉じて戻る
04934 file.close();
04935 return true;
04936 }
04937
04938 //
04939 // 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
04940 //
04941 // vert パーテックスシェーダのソースファイル名
04942 // frag フラグメントシェーダのソースファイル名 (nullptr なら不使用)
04943 // geom ジオメトリシェーダのソースファイル名 (nullptr なら不使用)
04944 // nvarying フィードバックする varying 変数の数 (0 なら不使用)
04945 // varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
04946 // 戻り値 シェーダプログラムのプログラム名 (作成できなければ 0)
04947 //
04948 GLuint gg::ggLoadShader(
04949 const std::string& vert,
04950 const std::string& frag,
04951 const std::string& geom,
04952 GLint nvarying,
04953 const char* const* varyings
04954 )
04955 {
04956 // 読み込んだシェーダのソースプログラム
04957 std::string vsrc, fsrc, gsrc;
04958
04959 // 読み込んだ結果
04960 bool status;
04961
04962 // シェーダのソースファイルを読み込む
04963 status = readShaderSource(vert, vsrc);
04964 status = readShaderSource(frag, fsrc) && status;
04965 status = readShaderSource(geom, gsrc) && status;
04966
04967 // 全てのソースファイルが読み込めていなければエラー
04968 if (!status) return 0;
04969
04970 // プログラムオブジェクトを作成する
04971 return ggCreateShader(vsrc, fsrc, gsrc, nvarying, varyings, vert, frag, geom);
04972 }
04973
04974 #if !defined(__APPLE__) && !defined(GL_GLES_PROTOTYPES)
04975 //
04976 // 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

```

```

04977 //
04978 //   csrc コンピュートシェーダのソースプログラムの文字列
04979 //   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)
04980 //
04981 GLuint gg::ggCreateComputeShader(const std::string& csrc, const std::string& ctext)
04982 {
04983   // シェーダプログラムの作成
04984   const GLuint program{ glCreateProgram() };
04985
04986   if (program > 0)
04987   {
04988     // ソースプログラムの文字列が空だったら 0 を返す
04989     if (csrc.empty()) return 0;
04990
04991     // コンピュートシェーダのシェーダオブジェクトを作成する
04992     const GLuint compShader{ glCreateShader(GL_COMPUTE_SHADER) };
04993     const GLchar* csrcp{ csrc.c_str() };
04994     glShaderSource(compShader, 1, &csrcp, nullptr);
04995     glCompileShader(compShader);
04996
04997     // コンピュートシェーダのシェーダオブジェクトをプログラムオブジェクトに組み込む
04998     if (printShaderInfoLog(compShader, ctext)) glAttachShader(program, compShader);
04999     glDeleteShader(compShader);
05000
05001     // シェーダプログラムをリンクする
05002     glLinkProgram(program);
05003
05004     // プログラムオブジェクトが作成できなければ 0 を返す
05005     if (printProgramInfoLog(program) == GL_FALSE)
05006     {
05007       glDeleteProgram(program);
05008       return 0;
05009     }
05010   }
05011
05012   // プログラムオブジェクトを返す
05013   return program;
05014 }
05015
05016 //
05017 // 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
05018 //
05019 //   comp コンピュートシェーダのソースファイル名
05020 //   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)
05021 //
05022 GLuint gg::ggLoadComputeShader(const std::string& comp)
05023 {
05024   // シェーダのソースファイルを読み込む
05025   std::string csrc;
05026   if (readShaderSource(comp, csrc))
05027   {
05028     // プログラムオブジェクトを作成する
05029     return ggCreateComputeShader(csrc.data(), comp);
05030   }
05031
05032   // プログラムオブジェクト作成失敗
05033   return 0;
05034 }
05035 #endif
05036
05037 //
05038 // 点: データ作成
05039 //
05040 void gg::GgPoints::load(const GgVector* pos, GLsizei count, GLenum usage)
05041 {
05042   // 頂点バッファオブジェクトを作成する
05043   position = std::make_shared<GgBuffer<GgVector>>(GL_ARRAY_BUFFER, pos,
05044     static_cast<GLsizei>(sizeof(GgVector)), count, usage);
05045
05046   // このバッファオブジェクトは index == 0 の in 変数から入力する
05047   glVertexAttribPointer(0, static_cast<GLint>(pos->size()), GL_FLOAT, GL_FALSE, 0, 0);
05048   glEnableVertexAttribArray(0);
05049
05050 //
05051 // 点: 描画
05052 //
05053 void gg::GgPoints::draw(GLint first, GLsizei count) const
05054 {
05055   // 頂点配列オブジェクトを指定する
05056   GgShape::draw(first, count);
05057
05058   // 図形を描画する
05059   glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05060 }
05061
05062 //

```

```

05063 // 三角形: データ作成
05064 //
05065 void gg::GgTriangles::load(const GgVertex* vert, GLsizei count, GLenum usage)
05066 {
05067     // 頂点バッファオブジェクトを作成する
05068     vertex = std::make_unique<GgBuffer<GgVertex>>(GL_ARRAY_BUFFER, vert,
05069     static_cast<GLsizei>(sizeof(GgVertex)), count, usage);
05070     // 頂点の位置は index == 0 の in 変数から入力する
05071     glVertexAttribPointer(0, static_cast<GLint>(vert->position.size()), GL_FLOAT, GL_FALSE,
05072     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, position));
05073     glEnableVertexAttribArray(0);
05074     // 頂点の法線は index == 1 の in 変数から入力する
05075     glVertexAttribPointer(1, static_cast<GLint>(vert->normal.size()), GL_FLOAT, GL_FALSE,
05076     sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, normal));
05077     glEnableVertexAttribArray(1);
05078 }
05079 }
05080
05081 //
05082 // 三角形: 描画
05083 //
05084 void gg::GgTriangles::draw(GLint first, GLsizei count) const
05085 {
05086     // 頂点配列オブジェクトを指定する
05087     GgShape::draw(first, count);
05088
05089     // 図形を描画する
05090     glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
05091 }
05092
05093 //
05094 // オブジェクト: 描画
05095 //
05096 void gg::GgElements::draw(GLint first, GLsizei count) const
05097 {
05098     // 頂点配列オブジェクトを指定する
05099     GgShape::draw(first, count);
05100
05101     // 図形を描画する
05102     glDrawElements(getMode(), count > 0 ? count : getIndexCount() - first,
05103     GL_UNSIGNED_INT, static_cast<GLuint*>(0) + first);
05104 }
05105
05106 //
05107 // 点群を立方体状に生成する
05108 //
05109 std::unique_ptr<gg::GgPoints> gg::ggPointsCube(GLsizei count, GLfloat length, GLfloat cx, GLfloat cy,
05110     GLfloat cz)
05111 {
05112     // メモリを確保する
05113     std::vector<GgVector> pos(count);
05114
05115     // 点を生成する
05116     for (GLsizei v = 0; v < count; ++v)
05117     {
05118         const GgVector p
05119         {
05120             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cx,
05121             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cy,
05122             (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cz,
05123             1.0f
05124         };
05125         pos.emplace_back(p);
05126     }
05127
05128     // 点データの GgPoints オブジェクトを作成して返す
05129     return std::make_unique<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05130 }
05131
05132 //
05133 // 点群を球状に生成する
05134 //
05135 std::unique_ptr<gg::GgPoints> gg::ggPointsSphere(GLsizei count, GLfloat radius,
05136     GLfloat cx, GLfloat cy, GLfloat cz)
05137 {
05138     // メモリを確保する
05139     std::vector<GgVector> pos(count);
05140
05141     // 点を生成する
05142     for (GLsizei v = 0; v < count; ++v)
05143     {
05144         const GLfloat r{ radius * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) };
05145         const GLfloat t{ 6.28318530718f * static_cast<GLfloat>(rand()) / (static_cast<GLfloat>(RAND_MAX) +
05146             1.0f) };
05147         const GLfloat cp{ 2.0f * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 1.0f };
05148     }
05149 }

```

```
05147     const GLfloat sp{ sqrtf(1.0f - cp * cp) };
05148     const GLfloat ct{ cosf(t) };
05149     const GLfloat st{ sinf(t) };
05150     const GgVector p{ r * sp * ct + cx, r * sp * st + cy, r * cp + cz, 1.0f };
05151     pos.emplace_back(p);
05152 }
05154
05155 // 点データの GgPoints オブジェクトを作成して返す
05156 return std::make_unique<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
05157 }
05158
05159 //
05160 // 矩形状に 2 枚の三角形を生成する
05161 //
05162 std::unique_ptr<gg::GgTriangles> gg::ggRectangle(GLfloat width, GLfloat height)
05163 {
05164     // 頂点属性
05165     std::array<gg::GgVertex, 4> vert;
05166
05167     // 頂点位置と法線を求める
05168     for (int v = 0; v < 4; ++v)
05169     {
05170         const GLfloat x{ ((v & 1) * 2 - 1) * width };
05171         const GLfloat y{ ((v & 2) - 1) * height };
05172
05173         vert[v] = gg::GgVertex{ x, y, 0.0f, 0.0f, 0.0f, 1.0f };
05174     }
05175
05176     // 矩形の GgTriangles オブジェクトを作成する
05177     return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLE_STRIP);
05178 }
05179
05180 //
05181 // 楕円状に三角形を生成する
05182 //
05183 std::unique_ptr<gg::GgTriangles> gg::ggEllipse(GLfloat width, GLfloat height, GLuint slices)
05184 {
05185     // 楕円のスケール
05186     constexpr GLfloat scale{ 0.5f };
05187
05188     // 作業用のメモリ
05189     std::vector<gg::GgVertex> vert;
05190
05191     // 頂点位置と法線を求める
05192     for (GLuint v = 0; v < slices; ++v)
05193     {
05194         const GLfloat t{ 6.28318530717f * static_cast<GLfloat>(v) / static_cast<GLfloat>(slices) };
05195         const GLfloat x{ cosf(t) * width * scale };
05196         const GLfloat y{ sinf(t) * height * scale };
05197
05198         vert.emplace_back(x, y, 0.0f, 0.0f, 0.0f, 1.0f);
05199     }
05200
05201     // GgTriangles オブジェクトを作成する
05202     return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLE_FAN);
05203 }
05204
05205 //
05206 // Wavefront OBJ ファイルを読み込む (Arrays 形式)
05207 //
05208 std::unique_ptr<gg::GgTriangles> gg::ggArraysObj(const std::string& name, bool normalize)
05209 {
05210     std::vector<std::array<GLuint, 3>> group;
05211     std::vector<gg::GgSimpleShader::Material> material;
05212     std::vector<gg::GgVertex> vert;
05213
05214     // ファイルを読み込む
05215     if (!ggLoadSimpleObj(name, group, material, vert, normalize)) return nullptr;
05216
05217     // GgTriangles オブジェクトを作成する
05218     return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()), GL_TRIANGLES);
05219 }
05220
05221 //
05222 // Wavefront OBJ ファイルを読み込む (Elements 形式)
05223 //
05224 std::unique_ptr<gg::GgElements> gg::ggElementsObj(const std::string& name, bool normalize)
05225 {
05226     std::vector<std::array<GLuint, 3>> group;
05227     std::vector<gg::GgSimpleShader::Material> material;
05228     std::vector<gg::GgVertex> vert;
05229     std::vector<GLuint> face;
05230 }
```

```

05231 // ファイルを読み込む
05232 if (!ggLoadSimpleObj(name, group, material, vert, face, normalize)) return nullptr;
05233
05234 // GgElements オブジェクトを作成する
05235 return std::make_unique<gg::GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05236     face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05237 }
05238
05239 //
05240 // メッシュ形状を作成する (Elements 形式)
05241 //
05242 std::unique_ptr<gg::GgElements> gg::ggElementsMesh(GLuint slices, GLuint stacks, const
05243 GLfloat *pos)[3], const GLfloat *norm)[3])
05244 {
05245     // 頂点属性
05246     std::vector<gg::GgVertex> vert((slices + 1) * (stacks + 1));
05247
05248     // 頂点の法線を求める
05249     for (GLuint j = 0; j <= stacks; ++j)
05250     {
05251         for (GLuint i = 0; i <= slices; ++i)
05252         {
05253             // 処理対象の頂点番号
05254             const GLuint k{ j * (slices + 1) + i };
05255
05256             // 頂点の法線
05257             gg::GgVector tnorm;
05258             tnorm[3] = 0.0f;
05259
05260             if (norm)
05261             {
05262                 tnorm[0] = norm[k][0];
05263                 tnorm[1] = norm[k][1];
05264                 tnorm[2] = norm[k][2];
05265             }
05266             else
05267             {
05268                 // 処理対象の頂点の周囲の頂点番号
05269                 const GLuint kim{ i > 0 ? k - 1 : k };
05270                 const GLuint kip{ i < slices ? k + 1 : k };
05271                 const GLuint kjm{ j > 0 ? k - slices - 1 : k };
05272                 const GLuint kjp{ j < stacks ? k + slices + 1 : k };
05273
05274                 // 接線ベクトル
05275                 const std::array<GLfloat, 3> t
05276                 {
05277                     pos[kip][0] - pos[kim][0],
05278                     pos[kip][1] - pos[kim][1],
05279                     pos[kip][2] - pos[kim][2]
05280                 };
05281
05282                 // 駒接線ベクトル
05283                 const std::array<GLfloat, 3> b
05284                 {
05285                     pos[kjp][0] - pos[kjm][0],
05286                     pos[kjp][1] - pos[kjm][1],
05287                     pos[kjp][2] - pos[kjm][2]
05288                 };
05289
05290                 // 法線
05291                 tnorm[0] = t[1] * b[2] - t[2] * b[1];
05292                 tnorm[1] = t[2] * b[0] - t[0] * b[2];
05293                 tnorm[2] = t[0] * b[1] - t[1] * b[0];
05294
05295                 // 法線の正規化
05296                 ggNormalize3(tnorm);
05297             }
05298
05299             // 頂点の位置
05300             const gg::GgVector tpos{ pos[k][0], pos[k][1], pos[k][2], 1.0f };
05301
05302             // 頂点属性の保存
05303             vert.emplace_back(tpos, tnorm);
05304         }
05305
05306             // 頂点のインデックス (三角形データ)
05307             std::vector<GLuint> face;
05308
05309             // 頂点のインデックスを求める
05310             for (GLuint j = 0; j < stacks; ++j)
05311             {
05312                 for (GLuint i = 0; i < slices; ++i)
05313                 {
05314                     // 処理対象のマスク
05315                     const GLuint k{ (slices + 1) * j + i };
05316

```

```

05317     // マスの上半分の三角形
05318     face.emplace_back(k);
05319     face.emplace_back(k + slices + 2);
05320     face.emplace_back(k + 1);
05321
05322     // マスのお下半分の三角形
05323     face.emplace_back(k);
05324     face.emplace_back(k + slices + 1);
05325     face.emplace_back(k + slices + 2);
05326 }
05327 }
05328
05329 // GgElements オブジェクトを作成する
05330 return std::make_unique<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05331     face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05332 }
05333
05334 //
05335 // 球状に三角形データを生成する (Elements 形式)
05336 //
05337 std::unique_ptr<gg::GgElements> gg::ggElementsSphere(GLfloat radius, int slices, int stacks)
05338 {
05339     // 頂点の位置と法線
05340     std::vector<GLfloat> p, n;
05341
05342     // 頂点の位置と法線を求める
05343     for (int j = 0; j <= stacks; ++j)
05344     {
05345         const GLfloat t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };
05346         const GLfloat ph{ 3.1415926536f * t };
05347         const GLfloat y{ cosf(ph) };
05348         const GLfloat r{ sinf(ph) };
05349
05350         for (int i = 0; i <= slices; ++i)
05351         {
05352             const GLfloat s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };
05353             const GLfloat th{ -2.0f * 3.1415926536f * s };
05354             const GLfloat x{ r * cosf(th) };
05355             const GLfloat z{ r * sinf(th) };
05356
05357             // 頂点の座標値
05358             p.emplace_back(x * radius);
05359             p.emplace_back(y * radius);
05360             p.emplace_back(z * radius);
05361
05362             // 頂点の法線
05363             n.emplace_back(x);
05364             n.emplace_back(y);
05365             n.emplace_back(z);
05366         }
05367     }
05368
05369 // GgElements オブジェクトを作成する
05370 return ggElementsMesh(slices, stacks, reinterpret_cast<GLfloat(*)[3]>(p.data()),
05371     reinterpret_cast<GLfloat(*)[3]>(p.data()));
05372 }
05373
05374 //
05375 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の環境光成分を設定する
05376 //
05377 // r 光源の強度の環境光成分の赤成分
05378 // g 光源の強度の環境光成分の緑成分
05379 // b 光源の強度の環境光成分の青成分
05380 // a 光源の強度の環境光成分の不透明度、デフォルトは 1
05381 // first 値を設定する光源データの最初の番号、デフォルトは 0
05382 // count 値を設定する光源データの数、デフォルトは 1
05383 //
05384 void gg::GgSimpleShader::LightBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05385     GLint first, GLsizei count) const
05386 {
05387     // データを格納するバッファオブジェクトの先頭のポインタ
05388     char* const start{ static_cast<char*>(map(first, count)) };
05389     for (GLsizei i = 0; i < count; ++i)
05390     {
05391         // バッファオブジェクトの i 番目のブロックのポインタ
05392         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05393
05394         // 光源の環境光成分を設定する
05395         light->ambient[0] = r;
05396         light->ambient[1] = g;
05397         light->ambient[2] = b;
05398         light->ambient[3] = a;
05399     }
05400     unmap();
05401 }
05402
05403 //

```

```

05404 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の環境光成分を設定する
05405 //
05406 //    ambient 光源の強度の環境光成分
05407 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05408 //    count 値を設定する光源データの数, デフォルトは 1
05409 //
05410 void gg::GgSimpleShader::LightBuffer::loadAmbient(const GgVector& ambient,
05411     GLint first, GLsizei count) const
05412 {
05413     // データを格納するバッファオブジェクトの先頭のポインタ
05414     char* const start{ static_cast<char*>(map(first, count)) };
05415     for (GLsizei i = 0; i < count; ++i)
05416     {
05417         // バッファオブジェクトの i 番目のブロックのポインタ
05418         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05419
05420         // 光源の強度の環境光成分を設定する
05421         light->ambient = ambient;
05422     }
05423     unmap();
05424 }
05425
05426 //
05427 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の拡散反射光成分を設定する
05428 //
05429 //    r 光源の強度の拡散反射光成分の赤成分
05430 //    g 光源の強度の拡散反射光成分の緑成分
05431 //    b 光源の強度の拡散反射光成分の青成分
05432 //    a 光源の強度の拡散反射光成分の不透明度, デフォルトは 1
05433 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05434 //    count 値を設定する光源データの数, デフォルトは 1
05435 //
05436 void gg::GgSimpleShader::LightBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05437     GLint first, GLsizei count) const
05438 {
05439     // データを格納するバッファオブジェクトの先頭のポインタ
05440     char* const start{ static_cast<char*>(map(first, count)) };
05441     for (GLsizei i = 0; i < count; ++i)
05442     {
05443         // バッファオブジェクトの i 番目のブロックのポインタ
05444         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05445
05446         // 光源の拡散反射光成分を設定する
05447         light->diffuse[0] = r;
05448         light->diffuse[1] = g;
05449         light->diffuse[2] = b;
05450         light->diffuse[3] = a;
05451     }
05452     unmap();
05453 }
05454
05455 //
05456 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の鏡面反射光成分を設定する
05457 //
05458 //    ambient 光源の強度の鏡面反射光成分
05459 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05460 //    count 値を設定する光源データの数, デフォルトは 1
05461 //
05462 void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& diffuse,
05463     GLint first, GLsizei count) const
05464 {
05465     // データを格納するバッファオブジェクトの先頭のポインタ
05466     char* const start{ static_cast<char*>(map(first, count)) };
05467     for (GLsizei i = 0; i < count; ++i)
05468     {
05469         // バッファオブジェクトの i 番目のブロックのポインタ
05470         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05471
05472         // 光源の強度の鏡面反射光成分を設定する
05473         light->diffuse = diffuse;
05474     }
05475     unmap();
05476 }
05477
05478 //
05479 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の鏡面反射光成分を設定する
05480 //
05481 //    r 光源の強度の鏡面反射光成分の赤成分
05482 //    g 光源の強度の鏡面反射光成分の緑成分
05483 //    b 光源の強度の鏡面反射光成分の青成分
05484 //    a 光源の強度の鏡面反射光成分の不透明度, デフォルトは 1
05485 //    first 値を設定する光源データの最初の番号, デフォルトは 0
05486 //    count 値を設定する光源データの数, デフォルトは 1
05487 //
05488 void gg::GgSimpleShader::LightBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05489     GLint first, GLsizei count) const
05490 {

```

```

05491 // データを格納するバッファオブジェクトの先頭のポインタ
05492 char* const start{ static_cast<char*>(map(first, count)) };
05493 for (GLsizei i = 0; i < count; ++i)
05494 {
05495     // バッファオブジェクトの i 番目のブロックのポインタ
05496     Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05497
05498     // 光源の鏡面反射光成分を設定する
05499     light->specular[0] = r;
05500     light->specular[1] = g;
05501     light->specular[2] = b;
05502     light->specular[3] = a;
05503 }
05504 unmap();
05505 }
05506
05507 //
05508 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の強度の鏡面反射光成分を設定する
05509 //
05510 // ambient 光源の強度の鏡面反射光成分
05511 // first 値を設定する光源データの最初の番号, デフォルトは 0
05512 // count 値を設定する光源データの数, デフォルトは 1
05513 //
05514 void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& specular,
05515 GLint first, GLsizei count) const
05516 {
05517     // データを格納するバッファオブジェクトの先頭のポインタ
05518     char* const start{ static_cast<char*>(map(first, count)) };
05519     for (GLsizei i = 0; i < count; ++i)
05520     {
05521         // バッファオブジェクトの i 番目のブロックのポインタ
05522         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05523
05524         // 光源の強度の鏡面反射光成分を設定する
05525         light->specular = specular;
05526     }
05527 unmap();
05528 }
05529
05530 //
05531 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の色を設定するか位置は変更しない
05532 //
05533 // material 光源の特性の GgSimpleShader::Light 構造体
05534 // first 値を設定する光源データの最初の番号, デフォルトは 0
05535 // count 値を設定する光源データの数, デフォルトは 1
05536 //
05537 void gg::GgSimpleShader::LightBuffer::loadColor(const Light& color,
05538 GLint first, GLsizei count) const
05539 {
05540     // データを格納するバッファオブジェクトの先頭のポインタ
05541     char* const start{ static_cast<char*>(map(first, count)) };
05542     for (GLsizei i = 0; i < count; ++i)
05543     {
05544         // バッファオブジェクトの i 番目のブロックのポインタ
05545         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05546
05547         // 光源の色を設定する
05548         light->ambient = color.ambient;
05549         light->diffuse = color.diffuse;
05550         light->specular = color.specular;
05551     }
05552 unmap();
05553 }
05554
05555 //
05556 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の位置を設定する
05557 //
05558 // x 光源の位置の x 座標
05559 // y 光源の位置の y 座標
05560 // z 光源の位置の z 座標
05561 // w 光源の位置の w 座標, デフォルトは 1
05562 // first 値を設定する光源データの最初の番号, デフォルトは 0
05563 // count 値を設定する光源データの数, デフォルトは 1
05564 //
05565 void gg::GgSimpleShader::LightBuffer::loadPosition(GLfloat x, GLfloat y, GLfloat z, GLfloat w,
05566 GLint first, GLsizei count) const
05567 {
05568     // データを格納するバッファオブジェクトの先頭のポインタ
05569     char* const start{ static_cast<char*>(map(first, count)) };
05570     for (GLsizei i = 0; i < count; ++i)
05571     {
05572         // バッファオブジェクトの i 番目のブロックのポインタ
05573         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05574
05575         // 光源の位置を設定する
05576         light->position[0] = x;
05577         light->position[1] = y;

```

```

05578     light->position[2] = z;
05579     light->position[3] = w;
05580 }
05581 unmap();
05582 }
05583
05584 //
05585 // 三角形に単純な陰影付けを行うシェーダが参照する光源データ：光源の位置を設定する
05586 //
05587 // position 光源の位置
05588 // first 値を設定する光源データの最初の番号, デフォルトは 0
05589 // count 値を設定する光源データの数, デフォルトは 1
05590 //
05591 void gg::GgSimpleShader::LightBuffer::loadPosition(const GgVector& position,
05592 GLint first, GLsizei count) const
05593 {
05594     // データを格納するバッファオブジェクトの先頭のポインタ
05595     char* const start{ static_cast<char*>(map(first, count)) };
05596     for (GLsizei i = 0; i < count; ++i)
05597     {
05598         // バッファオブジェクトの i 番目のブロックのポインタ
05599         Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };
05600
05601         // 光源の位置を設定する
05602         light->position = position;
05603     }
05604     unmap();
05605 }
05606
05607 //
05608 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数を設定する
05609 //
05610 // r 環境光に対する反射係数の赤成分
05611 // g 環境光に対する反射係数の緑成分
05612 // b 環境光に対する反射係数の青成分
05613 // a 環境光に対する反射係数の不透明度, デフォルトは 1
05614 // first 値を設定する材質データの最初の番号, デフォルトは 0
05615 // count 値を設定する材質データの数, デフォルトは 1
05616 //
05617 void gg::GgSimpleShader::MaterialBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05618 GLint first, GLsizei count) const
05619 {
05620     // データを格納するバッファオブジェクトの先頭のポインタ
05621     char* const start{ static_cast<char*>(map(first, count)) };
05622     for (GLsizei i = 0; i < count; ++i)
05623     {
05624         // バッファオブジェクトの i 番目のブロックのポインタ
05625         Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05626
05627         // 環境光に対する反射係数を設定する
05628         material->ambient[0] = r;
05629         material->ambient[1] = g;
05630         material->ambient[2] = b;
05631         material->ambient[3] = a;
05632     }
05633     unmap();
05634 }
05635
05636 //
05637 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：拡散反射係数を設定する
05638 //
05639 // r 拡散反射係数の赤成分
05640 // g 拡散反射係数の緑成分
05641 // b 拡散反射係数の青成分
05642 // a 拡散反射係数の不透明度, デフォルトは 1
05643 // first 値を設定する材質データの最初の番号, デフォルトは 0
05644 // count 値を設定する材質データの数, デフォルトは 1
05645 //
05646 void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05647 GLint first, GLsizei count) const
05648 {
05649     // データを格納するバッファオブジェクトの先頭のポインタ
05650     char* const start{ static_cast<char*>(map(first, count)) };
05651     for (GLsizei i = 0; i < count; ++i)
05652     {
05653         // バッファオブジェクトの i 番目のブロックのポインタ
05654         Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05655
05656         // 拡散反射係数を設定する
05657         material->diffuse[0] = r;
05658         material->diffuse[1] = g;
05659         material->diffuse[2] = b;
05660         material->diffuse[3] = a;
05661     }
05662     unmap();
05663 }
05664

```

```

05665 //
05666 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する
05667 //
05668 //   r 環境光に対する反射係数と拡散反射係数の赤成分
05669 //   g 環境光に対する反射係数と拡散反射係数の緑成分
05670 //   b 環境光に対する反射係数と拡散反射係数の青成分
05671 //   a 環境光に対する反射係数と拡散反射係数の不透明度、デフォルトは 1
05672 //   first 値を設定する材質データの最初の番号、デフォルトは 0
05673 //   count 値を設定する材質データの数、デフォルトは 1
05674 //
05675 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(GLfloat r, GLfloat g, GLfloat b,
05676   GLfloat a,
05677   GLint first, GLsizei count) const
05678 {
05679   // データを格納するバッファオブジェクトの先頭のポインタ
05680   char* const start{ static_cast<char*>(map(first, count)) };
05681   for (GLsizei i = 0; i < count; ++i)
05682   {
05683     // バッファオブジェクトの i 番目のブロックのポインタ
05684     Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05685     // 環境光に対する反射係数と拡散反射係数を設定する
05686     material->ambient[0] = material->diffuse[0] = r;
05687     material->ambient[1] = material->diffuse[1] = g;
05688     material->ambient[2] = material->diffuse[2] = b;
05689     material->ambient[3] = material->diffuse[3] = a;
05690   }
05691   unmap();
05692 }
05693
05694 //
05695 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：環境光に対する反射係数と拡散反射係数を設定する
05696 //
05697 //   color 環境光に対する反射係数と拡散反射係数を格納した GLfloat 型の 4 要素の配列
05698 //   first 値を設定する材質データの最初の番号、デフォルトは 0
05699 //   count 値を設定する材質データの数、デフォルトは 1
05700 //
05701 void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GLfloat* color,
05702   GLint first, GLsizei count) const
05703 {
05704   // ambient 要素のバイトオフセット
05705   constexpr GLint ambientOffset{ offsetof(Material, ambient) };
05706
05707   // ambient 要素のバイト数
05708   constexpr size_t ambientSize{ sizeof(Material::diffuse) };
05709
05710   // diffuse 要素のバイトオフセット
05711   constexpr GLint diffuseOffset{ offsetof(Material, diffuse) };
05712
05713   // diffuse 要素のバイト数
05714   constexpr size_t diffuseSize{ sizeof(Material::diffuse) };
05715
05716   // 元のデータの先頭
05717   const char* source{ reinterpret_cast<const char*>(color) };
05718
05719   // first 番目のブロックから count 値の ambient 要素と diffuse 要素に値を設定する
05720   bind();
05721   for (GLsizei i = 0; i < count; ++i)
05722   {
05723     // 格納先
05724     const GLsizeiptr destination{ getStride() * (first + i) };
05725
05726     // first + i 番目のブロックの ambient 要素に値を設定する
05727     glBufferSubData(getTarget(), destination + ambientOffset, ambientSize, source + i * ambientSize);
05728
05729     // first + i 番目のブロックの diffuse 要素に値を設定する
05730     glBufferSubData(getTarget(), destination + diffuseOffset, diffuseSize, source + i * diffuseSize);
05731   }
05732 }
05733
05734 //
05735 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：鏡面反射係数を設定する
05736 //
05737 //   r 鏡面反射係数の赤成分
05738 //   g 鏡面反射係数の緑成分
05739 //   b 鏡面反射係数の青成分
05740 //   a 鏡面反射係数の不透明度、デフォルトは 1
05741 //   first 値を設定する材質データの最初の番号、デフォルトは 0
05742 //   count 値を設定する材質データの数、デフォルトは 1
05743 //
05744 void gg::GgSimpleShader::MaterialBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
05745   GLint first, GLsizei count) const
05746 {
05747   // データを格納するバッファオブジェクトの先頭のポインタ
05748   char* const start{ static_cast<char*>(map(first, count)) };
05749   for (GLsizei i = 0; i < count; ++i)
05750   {

```

```

05751 // バッファオブジェクトの i 番目のブロックのポインタ
05752 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05753
05754 // 鏡面反射係数を設定する
05755 material->specular[0] = r;
05756 material->specular[1] = g;
05757 material->specular[2] = b;
05758 material->specular[3] = a;
05759 }
05760 unmap();
05761 }
05762
05763 //
05764 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：輝き係数を設定する
05765 //
05766 // shininess 輝き係数
05767 // first 値を設定する材質データの最初の番号, デフォルトは 0
05768 // count 値を設定する材質データの数, デフォルトは 1
05769 //
05770 void gg::GgSimpleShader::MaterialBuffer::loadShininess(GLfloat shininess,
05771 GLint first, GLsizei count) const
05772 {
05773 // データを格納するバッファオブジェクトの先頭のポインタ
05774 char* const start{ static_cast<char*>(map(first, count)) };
05775 for (GLsizei i = 0; i < count; ++i)
05776 {
05777 // バッファオブジェクトの i 番目のブロックのポインタ
05778 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05779 material->shininess = shininess;
05780 }
05781 unmap();
05782 }
05783
05784 //
05785 // 三角形に単純な陰影付けを行うシェーダが参照する材質データ：輝き係数を設定する
05786 //
05787 // shininess 輝き係数
05788 // first 値を設定する材質データの最初の番号, デフォルトは 0
05789 // count 値を設定する材質データの数, デフォルトは 1
05790 //
05791 void gg::GgSimpleShader::MaterialBuffer::loadShininess(const GLfloat* shininess,
05792 GLint first, GLsizei count) const
05793 {
05794 // データを格納するバッファオブジェクトの先頭のポインタ
05795 char* const start{ static_cast<char*>(map(first, count)) };
05796 for (GLsizei i = 0; i < count; ++i)
05797 {
05798 // バッファオブジェクトの i 番目のブロックのポインタ
05799 Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
05800 material->shininess = shininess[i];
05801 }
05802 unmap();
05803 }
05804
05805 //
05806 // 三角形に単純な陰影付けを行うシェーダ：シェーダのソースファイルの読み込み
05807 //
05808 bool gg::GgSimpleShader::load(const std::string& vert, const std::string& frag, const std::string&
05809 geom,
05810 GLint nvarying, const char* const* varyings)
05811 {
05812 if (!GgPointShader::load(vert, frag, geom, nvarying, varyings)) return false;
05813
05814 mnLoc = glGetUniformLocation(get(), "mn");
05815 lightIndex = glGetUniformBlockIndex(get(), "Light");
05816 glUniformBlockBinding(get(), lightIndex, 0);
05817 materialIndex = glGetUniformLocation(get(), "Material");
05818 glUniformBlockBinding(get(), materialIndex, 1);
05819 return true;
05820 }
05821
05822 //
05823 // Wavefront OBJ 形式のデータ：コンストラクタ
05824 //
05825 gg::GgSimpleObj::GgSimpleObj(const std::string& name, bool normalize)
05826 {
05827 // 作業用のメモリ
05828 std::vector<GgSimpleShader::Material> mat;
05829 std::vector<GgVertex> vert;
05830 std::vector<GLuint> face;
05831
05832 // グループのデータのメモリを確保する
05833 group = std::make_shared<std::vector<std::array<GLuint, 3>>>();
05834
05835 // ファイルを読み込む
05836 if (ggLoadSimpleObj(name, *group, mat, vert, face, normalize))

```

```

05837  {
05838  // 頂点バッファオブジェクトを作成する
05839  data = std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
05840  face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
05841
05842  // 描画するオブジェクトを切り替えるために頂点配列オブジェクトを閉じておく
05843  glBindVertexArray(0);
05844
05845  // 材質データを設定する
05846  material = std::make_shared<GgSimpleShader::MaterialBuffer>(mat.data(),
05847  static_cast<GLsizei>(mat.size()));
05848 }
05849
05850 //
05851 // Wavefront OBJ 形式のデータ：図形の描画
05852 //
05853 void gg::GgSimpleObj::draw(GLint first, GLsizei count) const
05854 {
05855  // 保持しているグループの数
05856  const GLsizei ng{ static_cast<GLsizei>(group->size()) };
05857
05858  // 描画する最後のグループの次
05859  GLsizei last(count <= 0 ? ng : first + count);
05860  if (last > ng) last = ng;
05861
05862  for (GLsizei i = first; i < last; ++i)
05863  {
05864    // グループのデータ
05865    const auto& g{ (*group)[i] };
05866
05867    // 材質を設定する
05868    material->select(g[2]);
05869
05870    // 図形を描画する
05871    data->draw(g[0], g[1]);
05872  }
05873 }

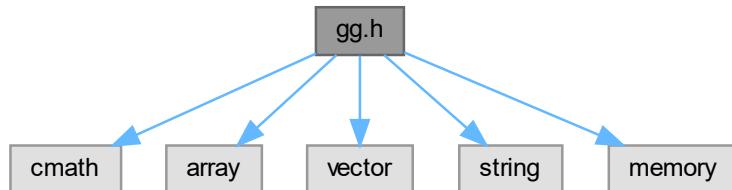
```

9.11 gg.h ファイル

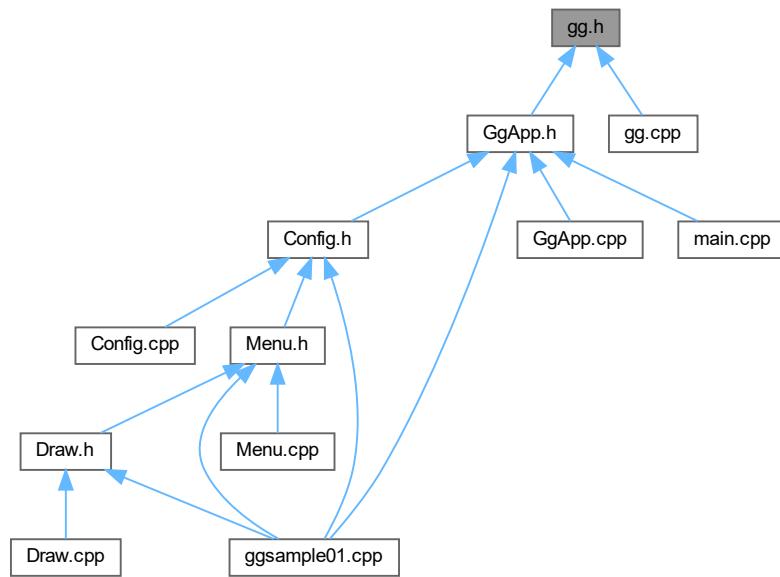
```

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

- namespace `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, bool swizzle=true)
- GLuint `gg::ggLoadImage` (const std::string &name, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=0, GLenum wrap=GL_CLAMP_TO_EDGE)
- void `gg::ggCreateNormalMap` (const GLubyte *hmap, GLsizei width, GLsizei height, GLenum format, GLfloat nz, GLenum internal, std::vector< GgVector > &nmap)
- GLuint `gg::ggLoadHeight` (const std::string &name, GLfloat nz, GLsizei *pWidth=nullptr, GLsizei *pHeight=nullptr, GLenum internal=GL_RGBA)
- GLuint `gg::ggCreateShader` (const std::string &vsrc, const std::string &fsrc="", const std::string &gsrc="", GLint nvarying=0, const char *const *varyings=nullptr, const std::string &vtext="vertex shader", const std::string &ftext="fragment shader", const std::string >ext="geometry shader")
- GLuint `gg::ggLoadShader` (const std::string &vert, const std::string &frag="", const std::string &geom="", GLint nvarying=0, const char *const *varyings=nullptr)

- GLuint `gg::ggLoadShader` (const std::array< std::string, 3 > &files, GLint nvarying=0, const char *const *varyings=nullptr)
- GLuint `gg::ggCreateComputeShader` (const std::string &csrc, const std::string &ctext="compute shader")
- GLuint `gg::ggLoadComputeShader` (const std::string &comp)
- std::unique_ptr< GgPoints > `gg::ggPointsCube` (GLsizei countv, GLfloat length=1.0f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::unique_ptr< GgPoints > `gg::ggPointsSphere` (GLsizei countv, GLfloat radius=0.5f, GLfloat cx=0.0f, GLfloat cy=0.0f, GLfloat cz=0.0f)
- std::unique_ptr< GgTriangles > `gg::ggRectangle` (GLfloat width=1.0f, GLfloat height=1.0f)
- std::unique_ptr< GgTriangles > `gg::ggEllipse` (GLfloat width=1.0f, GLfloat height=1.0f, GLuint slices=16)
- std::unique_ptr< GgTriangles > `gg::ggArraysObj` (const std::string &name, bool normalize=false)
- std::unique_ptr< GgElements > `gg::ggElementsObj` (const std::string &name, bool normalize=false)
- std::unique_ptr< GgElements > `gg::ggElementsMesh` (GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3]=nullptr)
- std::unique_ptr< GgElements > `gg::ggElementsSphere` (GLfloat radius=1.0f, int slices=16, int stacks=8)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, bool normalize=false)
- bool `gg::ggLoadSimpleObj` (const std::string &name, std::vector< std::array< GLuint, 3 > > &group, std::vector< GgSimpleShader::Material > &material, std::vector< GgVertex > &vert, std::vector< GLuint > &face, bool normalize=false)

変数

- GLint `gg::ggBufferAlignment`

使用している GPU のバッファアライメント。

9.11.1 詳解

ゲームグラフィックス特論の宿題用補助プログラム GLFW3 版の宣言。

著者

Kohe Tokoi

日付

March 31, 2021

`gg.h` に定義があります。

9.11.2 マクロ定義詳解

9.11.2.1 ggError

```
#define ggError( )
```

OpenGL のエラーの発生を検知したときにソースファイル名と行番号を表示する。

覚え書き

このマクロを置いた場所（より前）で OpenGL のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

gg.h の 1381 行目に定義があります。

9.11.2.2 ggFBOError

```
#define ggFBOError( )
```

FBO のエラーの発生を検知したときにソースファイル名と行番号を表示する。

覚え書き

このマクロを置いた場所（より前）で FBO のエラーが発生していた時に、このマクロを置いたソースファイル名と行番号を出力する。リリースビルド時には無視される。

gg.h の 1408 行目に定義があります。

9.12 gg.h

〔詳解〕

```
00001 #pragma once
00002
00036
00037 // macOS で "OpenGL deprecated" の警告を出さない
00038 #if defined(__APPLE__)
00039 # define GL_SILENCE_DEPRECATION
00040 #endif
00041
00042 // フレームワークに GLFW 3 を使う
00043 #if defined(IMGUI_IMPL_OPENGL_ES2)
00044 # define GLFW_INCLUDE_ES2
00045 #elif defined(IMGUI_IMPL_OPENGL_ES3)
00046 # define GLFW_INCLUDE_ES3
00047 #else
00048 # define GLFW_INCLUDE_GLCOREARB
00049 #endif
00050 #include <GLFW/glfw3.h>
00051
00052 // Windows (Visual Studio) のとき
00053 #if defined(_MSC_VER)
00054 // 非推奨の警告を出さない
00055 # pragma warning(disable:4996)
00056 // 数学ライブラリの定数を使う
00057 # define _USE_MATH_DEFINES
00058 // MIN() / MAX マクロは使わない
00059 # define NOMINMAX
00060 // APIENTRY マクロは使わない
00061 # undef APIENTRY
00062 // デバッグビルドかどうか調べる
00063 # if defined(_DEBUG)
00064 #     if !defined(DEBUG)
```

```
00065 #     define DEBUG
00066 #     endif
00067 #     define GLFW3_CONFIGURATION "Debug"
00068 #   else
00069 #     if !defined(NDEBUG)
00070 #       define NDEBUG
00071 #     endif
00072 #     define GLFW3_CONFIGURATION "Release"
00073 #   endif
00074 // プラットフォームを調べる
00075 # if defined(_WIN64)
00076 #   define GLFW3_PLATFORM "x64"
00077 # else
00078 #   define GLFW3_PLATFORM "Win32"
00079 # endif
00080 #endif
00081
00082 // OpenGL 3.2 の API のエントリポイント
00083 #if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
00084     extern PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
00085     extern PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
00086     extern PFNGLACTIVETEXTUREPROC glActiveTexture;
00087     extern PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
00088     extern PFNGLATTACHSHADERPROC glAttachShader;
00089     extern PFNGLBEGINCIONALRENDERNVPROC glBeginConditionalRenderNV;
00090     extern PFNGLBEGINCIONALRENDERPROC glBeginConditionalRender;
00091     extern PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
00092     extern PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
00093     extern PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
00094     extern PFNGLBEGINQUERYPROC glBeginQuery;
00095     extern PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
00096     extern PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
00097     extern PFNGLBINDBUFFERBASEPROC glBindBufferBase;
00098     extern PFNGLBINDBUFFERPROC glBindBuffer;
00099     extern PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
00100    extern PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
00101    extern PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
00102    extern PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
00103    extern PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
00104    extern PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
00105    extern PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
00106    extern PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
00107    extern PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
00108    extern PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
00109    extern PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
00110    extern PFNGLBINDSAMPLERPROC glBindSampler;
00111    extern PFNGLBINDSAMPLERSPROC glBindSamplers;
00112    extern PFNGLBINDTEXTUREPROC glBindTexture;
00113    extern PFNGLBINDTEXTURESPROC glBindTextures;
00114    extern PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
00115    extern PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
00116    extern PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
00117    extern PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
00118    extern PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
00119    extern PFNGLBLENDBARRIERKHRPROC glBindBarrierKHR;
00120    extern PFNGLBLENDBARRIERNVPROC glBindBarrierNV;
00121    extern PFNGLBLENDCOLORPROC glBindColor;
00122    extern PFNGLBLENDEQUATIONIARBPROC glBindEquationiARB;
00123    extern PFNGLBLENDEQUATIONIPROC glBindEquationi;
00124    extern PFNGLBLENDEQUATIONPROC glBindEquation;
00125    extern PFNGLBLENDEQUATIONSEPARATEIARBPROC glBindEquationSeparateiARB;
00126    extern PFNGLBLENDEQUATIONSEPARATEIPROC glBindEquationSeparatei;
00127    extern PFNGLBLENDEQUATIONSEPARATEPROC glBindEquationSeparate;
00128    extern PFNGLBLENDFUNCIARBPROC glBindFunciARB;
00129    extern PFNGLBLENDFUNCIPROC glBindFunci;
00130    extern PFNGLBLENDFUNCPROC glBindFunc;
00131    extern PFNGLBLENDFUNCSEPARATEIARBPROC glBindFuncSeparateiARB;
00132    extern PFNGLBLENDFUNCSEPARATEIPROC glBindFuncSeparatei;
00133    extern PFNGLBLENDFUNCSEPARATEPROC glBindFuncSeparate;
00134    extern PFNGLBLENDPARAMETERINVPROC glBindParameteriINV;
00135    extern PFNGLBLITFRAMEBUFFERPROC glBindFramebuffer;
00136    extern PFNGLBLITNAMEDFRAMEBUFFERPROC glBindNamedFramebuffer;
00137    extern PFNGLBUFFERADDRESSRANGEVPROC glBindBufferAddressRangeNV;
00138    extern PFNGLBUFFERDATAPROC glBindBufferData;
00139    extern PFNGLBUFFERPAGECOMMITMENTARBPROC glBindBufferPageCommitmentARB;
00140    extern PFNGLBUFFERSTORAGEPROC glBindBufferStorage;
00141    extern PFNGLBUFFERSUBDATAPROC glBindBufferSubData;
00142    extern PFNGLCALLCOMMANDLISTNVPROC glBindCallCommandListNV;
00143    extern PFNGLCHECKFRAMEBUFFERSTATUSPROC glBindCheckFramebufferStatus;
00144    extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glBindCheckNamedFramebufferStatusEXT;
00145    extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glBindCheckNamedFramebufferStatus;
00146    extern PFNGLCLAMPCOLORPROC glBindClampColor;
00147    extern PFNGLCLEARBUFFERDATAPROC glBindClearBufferData;
00148    extern PFNGLCLEARBUFFERFIPROC glBindClearBufferfi;
00149    extern PFNGLCLEARBUFFERFVPROC glBindClearBufferfv;
00150    extern PFNGLCLEARBUFFERIVPROC glBindClearBufferiv;
00151    extern PFNGLCLEARBUFFERSUBDATAPROC glBindClearBufferSubData;
```

```

00152 extern PFNGLCLEARBUFFERUIVPROC glClearBufferui;
00153 extern PFNGLCLEARCOLORPROC glClearColor;
00154 extern PFNGLCLEARDEPTHFPROC glClearDepthf;
00155 extern PFNGLCLEARDEPTHPROC glClearDepth;
00156 extern PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
00157 extern PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
00158 extern PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
00159 extern PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
00160 extern PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
00161 extern PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
00162 extern PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
00163 extern PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferui;
00164 extern PFNGLCLEARPROC glClear;
00165 extern PFNGLCLEARSTENCILPROC glClearStencil;
00166 extern PFNGLCLEARTEXIMAGEPROC glClearTexImage;
00167 extern PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
00168 extern PFNGLCLIENTATTRIBDEFAULTEXTPROC glClientAttribDefaultEXT;
00169 extern PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
00170 extern PFNGLCLIPCONTROLPROC glClipControl;
00171 extern PFNGLCOLORFORMATNVPROC glColorFormatNV;
00172 extern PFNGLCOLORMASKIPROC glColorMaski;
00173 extern PFNGLCOLORMASKPROC glColorMask;
00174 extern PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
00175 extern PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
00176 extern PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
00177 extern PFNGLCOMPILESHADERPROC glCompileShader;
00178 extern PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
00179 extern PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
00180 extern PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
00181 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
00182 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
00183 extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
00184 extern PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
00185 extern PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
00186 extern PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
00187 extern PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
00188 extern PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
00189 extern PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
00190 extern PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
00191 extern PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
00192 extern PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
00193 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
00194 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
00195 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
00196 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
00197 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
00198 extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
00199 extern PFNGLCONSERVATERASTERPARAMETERFNVPROC glConservativeRasterParameterfnv;
00200 extern PFNGLCONSERVATERASTERPARAMETERINVPROC glConservativeRasterParameterinv;
00201 extern PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
00202 extern PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
00203 extern PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
00204 extern PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
00205 extern PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
00206 extern PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
00207 extern PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
00208 extern PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
00209 extern PFNGLCOPYPATHNVPROC glCopyPathnv;
00210 extern PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
00211 extern PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
00212 extern PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
00213 extern PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
00214 extern PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
00215 extern PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
00216 extern PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
00217 extern PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
00218 extern PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
00219 extern PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
00220 extern PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
00221 extern PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
00222 extern PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
00223 extern PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationnv;
00224 extern PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTablenv;
00225 extern PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancednv;
00226 extern PFNGLCOVERFILLPATHNVPROC glCoverFillPathnv;
00227 extern PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancednv;
00228 extern PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathnv;
00229 extern PFNGLCREATEBUFFERSPROC glCreateBuffers;
00230 extern PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsnv;
00231 extern PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
00232 extern PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryIntel;
00233 extern PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
00234 extern PFNGLCREATEPROGRAMPROC glCreateProgram;
00235 extern PFNGLCREATEQUERIESPROC glCreateQueries;
00236 extern PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
00237 extern PFNGLCREATESAMPLERSPROC glCreateSamplers;
00238 extern PFNGLCREATESHADERPROC glCreateShader;

```

```
00239 extern PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
00240 extern PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
00241 extern PFNGLCREATESTATESNVPROC glCreateStatesNV;
00242 extern PFNGLCREATESTATESYNCFROMCLEVENTARBPROC glCreateSyncFromCleventARB;
00243 extern PFNGLCREATETEXTURESPROC glCreateTextures;
00244 extern PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
00245 extern PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
00246 extern PFNGLCULLFACEPROC glCullFace;
00247 extern PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
00248 extern PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
00249 extern PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
00250 extern PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
00251 extern PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
00252 extern PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
00253 extern PFNGLDELETEBUFFERSPROC glDeleteBuffers;
00254 extern PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
00255 extern PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
00256 extern PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
00257 extern PFNGLDELETEPATHSNVPROC glDeletePathsNV;
00258 extern PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
00259 extern PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
00260 extern PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
00261 extern PFNGLDELETEPROGRAMPROC glDeleteProgram;
00262 extern PFNGLDELETEQUERIESPROC glDeleteQueries;
00263 extern PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
00264 extern PFNGLDELETESAMPLERSPROC glDeleteSamplers;
00265 extern PFNGLDELETESHADERPROC glDeleteShader;
00266 extern PFNGLDELETETESTESNVPROC glDeleteStatesNV;
00267 extern PFNGLDELETETESTESYNCPROC glDeleteSync;
00268 extern PFNGLDELETETEXTURESPROC glDeleteTextures;
00269 extern PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
00270 extern PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
00271 extern PFNGLDEPTHFUNCPROC glDepthFunc;
00272 extern PFNGLDEPTHMASKPROC glDepthMask;
00273 extern PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
00274 extern PFNGLDEPTHRANGEFPROC glDepthRangef;
00275 extern PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
00276 extern PFNGLDEPTHRANGEPROC glDepthRange;
00277 extern PFNGLDETACHSHADERPROC glDetachShader;
00278 extern PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
00279 extern PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
00280 extern PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
00281 extern PFNGLDISABLEIPROC glDisablei;
00282 extern PFNGLDISABLEPROC glDisable;
00283 extern PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexAttribArrayAttribEXT;
00284 extern PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexAttribArrayAttrib;
00285 extern PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexAttribArrayEXT;
00286 extern PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArrayArray;
00287 extern PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
00288 extern PFNGLDISPATCHCOMPUTEDIIRECTPROC glDispatchComputeIndirect;
00289 extern PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
00290 extern PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
00291 extern PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
00292 extern PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
00293 extern PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
00294 extern PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstanced;
00295 extern PFNGLDRAWARRAYSPROC glDrawArrays;
00296 extern PFNGLDRAWBUFFERPROC glDrawBuffer;
00297 extern PFNGLDRAWBUFFERSPROC glDrawBuffers;
00298 extern PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
00299 extern PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
00300 extern PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
00301 extern PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
00302 extern PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
00303 extern PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
00304 extern PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
00305 extern PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
00306 extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC
    glDrawElementsInstancedBaseVertexBaseInstance;
00307 extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
00308 extern PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
00309 extern PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
00310 extern PFNGLDRAWELEMENTSPROC glDrawElements;
00311 extern PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
00312 extern PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
00313 extern PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
00314 extern PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
00315 extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
00316 extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
00317 extern PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
00318 extern PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
00319 extern PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
00320 extern PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
00321 extern PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
00322 extern PFNGLENABLEIPROC glEnablei;
00323 extern PFNGLENABLEPROC glEnable;
00324 extern PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexArrayAttribEXT;
```

```

00325 extern PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexAttribArrayAttrib;
00326 extern PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexAttribArrayEXT;
00327 extern PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArrayAttribArray;
00328 extern PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
00329 extern PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
00330 extern PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
00331 extern PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
00332 extern PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
00333 extern PFNGLENDQUERYPROC glEndQuery;
00334 extern PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
00335 extern PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
00336 extern PFNGLFENCESYNCPROC glFenceSync;
00337 extern PFNGLFINISHPROC glFinish;
00338 extern PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
00339 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
00340 extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
00341 extern PFNGLFLUSHPROC glFlush;
00342 extern PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
00343 extern PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
00344 extern PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
00345 extern PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
00346 extern PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
00347 extern PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
00348 extern PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
00349 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSVARBPROC glFramebufferSampleLocationsfvARB;
00350 extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
00351 extern PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
00352 extern PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
00353 extern PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
00354 extern PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
00355 extern PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
00356 extern PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
00357 extern PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
00358 extern PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
00359 extern PFNGLFRAMEBUFFERTEXTUREREPROC glFramebufferTexture;
00360 extern PFNGLFRONTFACEPROC glFrontFace;
00361 extern PFNGLGENBUFFERSPROC glGenBuffers;
00362 extern PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
00363 extern PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
00364 extern PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
00365 extern PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
00366 extern PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
00367 extern PFNGLGENPATHSNVPROC glGenPathsNV;
00368 extern PFNGLGENPERFMONITORAMDPROC glGenPerfMonitorsAMD;
00369 extern PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
00370 extern PFNGLGENQUERIESPROC glGenQueries;
00371 extern PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
00372 extern PFNGLGENSAMPLERSPROC glGenSamplers;
00373 extern PFNGLGENTEXTURESPROC glGenTextures;
00374 extern PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
00375 extern PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
00376 extern PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
00377 extern PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
00378 extern PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
00379 extern PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
00380 extern PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
00381 extern PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
00382 extern PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
00383 extern PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
00384 extern PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
00385 extern PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
00386 extern PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
00387 extern PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
00388 extern PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
00389 extern PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
00390 extern PFNGLGETBOOLEANVPROC glGetBooleanv;
00391 extern PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
00392 extern PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
00393 extern PFNGLGETBUFFERPARAMETERUI64VNVPROC glGetBufferParameterui64vNV;
00394 extern PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
00395 extern PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
00396 extern PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
00397 extern PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
00398 extern PFNGLGETCOMPRESSEDTEXTIMAGEPROC glGetCompressedTexImage;
00399 extern PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImageEXT;
00400 extern PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
00401 extern PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
00402 extern PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
00403 extern PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
00404 extern PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
00405 extern PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
00406 extern PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
00407 extern PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
00408 extern PFNGLGETDOUBLEVPROC glGetDoublev;
00409 extern PFNGLGETERRORPROC glGetError;
00410 extern PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
00411 extern PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;

```

```

00412 extern PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
00413 extern PFNGLGETFLOATI_VPROC glGetFloati_v;
00414 extern PFNGLGETFLOATVPROC glGetFloatv;
00415 extern PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
00416 extern PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
00417 extern PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
00418 extern PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
00419 extern PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
00420 extern PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
00421 extern PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
00422 extern PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
00423 extern PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
00424 extern PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
00425 extern PFNGLGETINTEGER64VPROC glGetInteger64v;
00426 extern PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
00427 extern PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
00428 extern PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
00429 extern PFNGLGETINTEGERUI64NVPROC glGetIntegerui64NV;
00430 extern PFNGLGETINTEGERVPROC glGetIntegerv;
00431 extern PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
00432 extern PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
00433 extern PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
00434 extern PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
00435 extern PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
00436 extern PFNGLGETMULTITEXENVVEXTPROC glGetMultiTexEnvvEXT;
00437 extern PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGendvEXT;
00438 extern PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
00439 extern PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
00440 extern PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
00441 extern PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
00442 extern PFNGLGETMULTITEXLEVELPARAMETERIVEVEXTPROC glGetMultiTexLevelParameterivEXT;
00443 extern PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
00444 extern PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIivEXT;
00445 extern PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
00446 extern PFNGLGETMULTITEXPARAMETERIVEVEXTPROC glGetMultiTexParameterivEXT;
00447 extern PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
00448 extern PFNGLGETNAMEDBUFFERPARAMETERIVEVEXTPROC glGetNamedBufferParameterivEXT;
00449 extern PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
00450 extern PFNGLGETNAMEDBUFFERPARAMETERUI64NVPROC glGetNamedBufferParameterui64NV;
00451 extern PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
00452 extern PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
00453 extern PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
00454 extern PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
00455 extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEVEXTPROC
    glGetNamedFramebufferAttachmentParameterivEXT;
00456 extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
00457 extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEVEXTPROC glGetNamedFramebufferParameterivEXT;
00458 extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
00459 extern PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
00460 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
00461 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
00462 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIivEXT;
00463 extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
00464 extern PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
00465 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEVEXTPROC glGetNamedRenderbufferParameterivEXT;
00466 extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
00467 extern PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
00468 extern PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
00469 extern PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
00470 extern PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
00471 extern PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
00472 extern PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
00473 extern PFNGLGETNTEXIMAGEPROC glGetnTexImage;
00474 extern PFNGLGETNUNIFORMDVPROC glGetnUniformdvARB;
00475 extern PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00476 extern PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00477 extern PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
00478 extern PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
00479 extern PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
00480 extern PFNGLGETNUNIFORMVPROC glGetnUniformv;
00481 extern PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
00482 extern PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuiARB;
00483 extern PFNGLGETNUNIFORMUIVPROC glGetnUniformiui;
00484 extern PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
00485 extern PFNGLGETOBJECTLABELPROC glGetObjectLabel;
00486 extern PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
00487 extern PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
00488 extern PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
00489 extern PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
00490 extern PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
00491 extern PFNGLGETPATHMETRICRANGEENVPROC glGetPathMetricRangeENV;
00492 extern PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
00493 extern PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
00494 extern PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
00495 extern PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
00496 extern PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
00497 extern PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;

```

```

00498 extern PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
00499 extern PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
00500 extern PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
00501 extern PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
00502 extern PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
00503 extern PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
00504 extern PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
00505 extern PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
00506 extern PFNGLGETPOINTERINDEXDEVEXTPROC glGetPointerIndexedvEXT;
00507 extern PFNGLGETPOINTERI_VEXTPROC glGetPointeri_VEXT;
00508 extern PFNGLGETPOINTERVPROC glGetPointerv;
00509 extern PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
00510 extern PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
00511 extern PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
00512 extern PFNGLGETPROGRAMIVPROC glGetProgramiv;
00513 extern PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
00514 extern PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
00515 extern PFNGLGETPROGRAMRESOURCEFVNPROC glGetProgramResourcefvNV;
00516 extern PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
00517 extern PFNGLGETPROGRAMRESOURCIEIVPROC glGetProgramResourceiv;
00518 extern PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC glGetProgramResourceLocationIndex;
00519 extern PFNGLGETPROGRAMRESOURCELOCATIONPROC glGetProgramResourceLocation;
00520 extern PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
00521 extern PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
00522 extern PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
00523 extern PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
00524 extern PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
00525 extern PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
00526 extern PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
00527 extern PFNGLGETQUERYIVPROC glGetQueryiv;
00528 extern PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
00529 extern PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
00530 extern PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
00531 extern PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
00532 extern PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
00533 extern PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
00534 extern PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
00535 extern PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
00536 extern PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
00537 extern PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
00538 extern PFNGLGETSHADERIVPROC glGetShaderiv;
00539 extern PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
00540 extern PFNGLGETSHADERSOURCEPROC glGetShaderSource;
00541 extern PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
00542 extern PFNGLGETSTRINGIPROC glGetStringi;
00543 extern PFNGLGETSTRINGPROC glGetString;
00544 extern PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
00545 extern PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
00546 extern PFNGLGETSYNCIVPROC glGetSynciv;
00547 extern PFNGLGETTEXIMAGEPROC glGetTexImage;
00548 extern PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
00549 extern PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
00550 extern PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
00551 extern PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
00552 extern PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
00553 extern PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
00554 extern PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
00555 extern PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
00556 extern PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
00557 extern PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
00558 extern PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
00559 extern PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
00560 extern PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
00561 extern PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
00562 extern PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
00563 extern PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
00564 extern PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
00565 extern PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
00566 extern PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
00567 extern PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
00568 extern PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
00569 extern PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
00570 extern PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
00571 extern PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
00572 extern PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
00573 extern PFNGLGETTRANSFORMFEEDBACKI64VPROC glGetTransformFeedbacki64v;
00574 extern PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
00575 extern PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
00576 extern PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
00577 extern PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
00578 extern PFNGLGETUNIFORMDVPROC glGetUniformdv;
00579 extern PFNGLGETUNIFORMFVPROC glGetUniformfv;
00580 extern PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
00581 extern PFNGLGETUNIFORMI64NVPROC glGetUniformi64vNV;
00582 extern PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
00583 extern PFNGLGETUNIFORMIVPROC glGetUniformiv;
00584 extern PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;

```

```
00585 extern PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
00586 extern PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
00587 extern PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
00588 extern PFNGLGETUNIFORMUIVPROC glGetUniformmuiv;
00589 extern PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
00590 extern PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
00591 extern PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
00592 extern PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
00593 extern PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
00594 extern PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
00595 extern PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
00596 extern PFNGLGETVERTEXATTRIBDPROC glGetVertexAttribd;
00597 extern PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribfv;
00598 extern PFNGLGETVERTEXATTRIBIIIVPROC glGetVertexAttribIiiv;
00599 extern PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribIuiv;
00600 extern PFNGLGETVERTEXATTRIBLPROC glGetVertexAttribl;
00601 extern PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLdv;
00602 extern PFNGLGETVERTEXATTRIBL64VNVPROC glGetVertexAttribLi64vNV;
00603 extern PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribLui64vARB;
00604 extern PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribLui64vNV;
00605 extern PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribPointerv;
00606 extern PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrnv;
00607 extern PFNGLHINTPROC glHint;
00608 extern PFNGLINDEXFORMATNVPROC glIndexFormatNV;
00609 extern PFNGLINSETEVENTMARKEREXTPROC glInsertEventMarkerEXT;
00610 extern PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
00611 extern PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
00612 extern PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
00613 extern PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
00614 extern PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
00615 extern PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
00616 extern PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
00617 extern PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
00618 extern PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
00619 extern PFNGLISBUFFERPROC glIsBuffer;
00620 extern PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
00621 extern PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
00622 extern PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
00623 extern PFNGLISENABLEDIPROC glIsEnabledi;
00624 extern PFNGLISENABLEDPROC glIsEnabled;
00625 extern PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
00626 extern PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
00627 extern PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
00628 extern PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
00629 extern PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
00630 extern PFNGLISPATHNVPROC glIsPathNV;
00631 extern PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
00632 extern PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
00633 extern PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
00634 extern PFNGLISPROGRAMPROC glIsProgram;
00635 extern PFNGLISQUERYPROC glIsQuery;
00636 extern PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
00637 extern PFNGLISSAMPLERPROC glIsSampler;
00638 extern PFNGLISSHADERPROC glIsShader;
00639 extern PFNGLISSTATENVPROC glIsStateNV;
00640 extern PFNGLISSYNCPROC glIsSync;
00641 extern PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
00642 extern PFNGLISTTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
00643 extern PFNGLISTTEXTUREPROC glIsTexture;
00644 extern PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
00645 extern PFNGLISVERTEXARRAYPROC glIsVertexArray;
00646 extern PFNGLLABELOBJECTEXTPROC glLabelObjectEXT;
00647 extern PFNGLLINEWIDTHPROC glLineWidth;
00648 extern PFNGLLINKPROGRAMPROC glLinkProgram;
00649 extern PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
00650 extern PFNGLLOGICOPPROC glLogicOp;
00651 extern PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
00652 extern PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
00653 extern PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
00654 extern PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
00655 extern PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
00656 extern PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
00657 extern PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
00658 extern PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
00659 extern PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
00660 extern PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
00661 extern PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
00662 extern PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
00663 extern PFNGLMAPBUFFERPROC glMapBuffer;
00664 extern PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
00665 extern PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
00666 extern PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
00667 extern PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
00668 extern PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
00669 extern PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
00670 extern PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fnv;
00671 extern PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fnv;
```

```

00672 extern PFNGLMATRIXLOADDEXTPROC glMatrixLoaddEXT;
00673 extern PFNGLMATRIXLOADFEXTPROC glMatrixLoadfEXT;
00674 extern PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
00675 extern PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
00676 extern PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedEXT;
00677 extern PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
00678 extern PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
00679 extern PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
00680 extern PFNGLMATRIXMULTDEXTPROC glMatrixMultdEXT;
00681 extern PFNGLMATRIXMULTFEXTPROC glMatrixMultfEXT;
00682 extern PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
00683 extern PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
00684 extern PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefEXT;
00685 extern PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
00686 extern PFNGLMATRIXXPOPEXTPROC glMatrixPopEXT;
00687 extern PFNGLMATRIXXPUSHEXTPROC glMatrixPushEXT;
00688 extern PFNGLMATRIXXROTATEDEXTPROC glMatrixRotatedEXT;
00689 extern PFNGLMATRIXXROTATEFEXTPROC glMatrixRotatefEXT;
00690 extern PFNGLMATRIXXSCALEDEXTPROC glMatrixScaledEXT;
00691 extern PFNGLMATRIXXSCALEFEXTPROC glMatrixScalefEXT;
00692 extern PFNGLMATRIXXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
00693 extern PFNGLMATRIXXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
00694 extern PFNGLMAXSHADERCOMPILERTHREADSARB glMaxShaderCompilerThreadsARB;
00695 extern PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
00696 extern PFNGLMEMORYBARRIERPROC glMemoryBarrier;
00697 extern PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
00698 extern PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
00699 extern PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
00700 extern PFNGLMULTIDRAWARRAYSINDIRECTCOUNTNVPROC glMultiDrawArraysIndirectBindlessNV;
00701 extern PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC glMultiDrawArraysIndirectCountARB;
00702 extern PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
00703 extern PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
00704 extern PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
00705 extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
00706 extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
00707 extern PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC glMultiDrawElementsIndirectCountARB;
00708 extern PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
00709 extern PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
00710 extern PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
00711 extern PFNGLMULTITEXCOORDPOINTEREXTPROC glMultiTexCoordPointerEXT;
00712 extern PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
00713 extern PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
00714 extern PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvEXT;
00715 extern PFNGLMULTITEXENVIVEXTPROC glMultiTexEnvivEXT;
00716 extern PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
00717 extern PFNGLMULTITEXGENDEXTPROC glMultiTexGendvEXT;
00718 extern PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
00719 extern PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
00720 extern PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
00721 extern PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
00722 extern PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
00723 extern PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
00724 extern PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
00725 extern PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
00726 extern PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
00727 extern PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameteriEXT;
00728 extern PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameteriIivEXT;
00729 extern PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameteriIuivEXT;
00730 extern PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
00731 extern PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
00732 extern PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
00733 extern PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
00734 extern PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
00735 extern PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
00736 extern PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
00737 extern PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
00738 extern PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
00739 extern PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
00740 extern PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
00741 extern PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubData;
00742 extern PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
00743 extern PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
00744 extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
00745 extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
00746 extern PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
00747 extern PFNGLNAMEDFRAMEBUFFERPARAMETERIIPROC glNamedFramebufferParameteri;
00748 extern PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
00749 extern PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
00750 extern PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
00751 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
00752 extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
00753 extern PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
00754 extern PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
00755 extern PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
00756 extern PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
00757 extern PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
00758 extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;

```

```

00759 extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
00760 extern PFNGLNAMEDFRAMEBUFFERTEXTUREPROC glNamedFramebufferTexture;
00761 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
00762 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
00763 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
00764 extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
00765 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
00766 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
00767 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
00768 extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uivEXT;
00769 extern PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
00770 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
00771 extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
00772 extern PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
00773 extern PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
00774 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERGEEXTPROC
    glNamedRenderbufferStorageMultisampleCoverageEXT;
00775 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
00776 extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
00777 extern PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
00778 extern PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
00779 extern PFNGLNORMALFORMATNVPROC glNormalFormatNV;
00780 extern PFNGLOBJECTLABELPROC glObjectLabel;
00781 extern PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
00782 extern PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
00783 extern PFNGLPATCHPARAMETERIPROC glPatchParameteri;
00784 extern PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
00785 extern PFNGLPATHCOORDSNVPROC glPathCoordsNV;
00786 extern PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
00787 extern PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
00788 extern PFNGLPATHGLYPHINDEXARRAYNVPROC glPathGlyphIndexArrayNV;
00789 extern PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
00790 extern PFNGLPATHGLYPHRANGENVPROC glPathGlyphRangeNV;
00791 extern PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
00792 extern PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
00793 extern PFNGLPATHPARAMETERFNVPROC glPathParameterfv;
00794 extern PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
00795 extern PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
00796 extern PFNGLPATHPARAMETERINVNVPROC glPathParameterivNV;
00797 extern PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
00798 extern PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
00799 extern PFNGLPATHSTRINGNVPROC glPathStringNV;
00800 extern PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
00801 extern PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
00802 extern PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
00803 extern PFNGLPIXELSTOREFPROC glPixelStoref;
00804 extern PFNGLPIXELSTOREIPROC glPixelStorei;
00805 extern PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
00806 extern PFNGLPOINTPARAMETERFPROC glPointParameterf;
00807 extern PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
00808 extern PFNGLPOINTPARAMETERIPROC glPointParameteri;
00809 extern PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
00810 extern PFNGLPOINTSIZEPROC glPointSize;
00811 extern PFNGLPOLYGONMODEPROC glPolygonMode;
00812 extern PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
00813 extern PFNGLPOLYGONOFFSETPROC glPolygonOffset;
00814 extern PFNGLPOPODEBUGGROUPPROC glPopDebugGroup;
00815 extern PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
00816 extern PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
00817 extern PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
00818 extern PFNGLPROGRAMBINARYPROC glProgramBinary;
00819 extern PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
00820 extern PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
00821 extern PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
00822 extern PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
00823 extern PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
00824 extern PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dVEXT;
00825 extern PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dV;
00826 extern PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
00827 extern PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
00828 extern PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
00829 extern PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
00830 extern PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniformi64ARB;
00831 extern PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniformi164NV;
00832 extern PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniformi164vARB;
00833 extern PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniformi164vNV;
00834 extern PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniformiEXT;
00835 extern PFNGLPROGRAMUNIFORM1IPROC glProgramUniformi;
00836 extern PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
00837 extern PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
00838 extern PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniformui64ARB;
00839 extern PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniformui64NV;
00840 extern PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniformui64vARB;
00841 extern PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniformui64vNV;
00842 extern PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniformuiEXT;
00843 extern PFNGLPROGRAMUNIFORM1UIPROC glProgramUniformui;
00844 extern PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniformuiVEXT;

```

```

00845 extern PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1ui;
00846 extern PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
00847 extern PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
00848 extern PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
00849 extern PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
00850 extern PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
00851 extern PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
00852 extern PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
00853 extern PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
00854 extern PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
00855 extern PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
00856 extern PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
00857 extern PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
00858 extern PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
00859 extern PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
00860 extern PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
00861 extern PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
00862 extern PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
00863 extern PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
00864 extern PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
00865 extern PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
00866 extern PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
00867 extern PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
00868 extern PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
00869 extern PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
00870 extern PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
00871 extern PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
00872 extern PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
00873 extern PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
00874 extern PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
00875 extern PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
00876 extern PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
00877 extern PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
00878 extern PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
00879 extern PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
00880 extern PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
00881 extern PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
00882 extern PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
00883 extern PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
00884 extern PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
00885 extern PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
00886 extern PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
00887 extern PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
00888 extern PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
00889 extern PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
00890 extern PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
00891 extern PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
00892 extern PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
00893 extern PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
00894 extern PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
00895 extern PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
00896 extern PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
00897 extern PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
00898 extern PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
00899 extern PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
00900 extern PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
00901 extern PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
00902 extern PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
00903 extern PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
00904 extern PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
00905 extern PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
00906 extern PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
00907 extern PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
00908 extern PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
00909 extern PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
00910 extern PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
00911 extern PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
00912 extern PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
00913 extern PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
00914 extern PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
00915 extern PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
00916 extern PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
00917 extern PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
00918 extern PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniformHandleui64ARB;
00919 extern PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
00920 extern PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
00921 extern PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
00922 extern PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dvEXT;
00923 extern PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
00924 extern PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
00925 extern PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
00926 extern PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
00927 extern PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
00928 extern PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
00929 extern PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
00930 extern PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
00931 extern PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;

```

```

00932 extern PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
00933 extern PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
00934 extern PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
00935 extern PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dv;
00936 extern PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
00937 extern PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
00938 extern PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
00939 extern PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dv;
00940 extern PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
00941 extern PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
00942 extern PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
00943 extern PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dv;
00944 extern PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
00945 extern PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
00946 extern PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix4dvEXT;
00947 extern PFNGLPROGRAMUNIFORMMATRIX4DVEPROC glProgramUniformMatrix4dv;
00948 extern PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
00949 extern PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
00950 extern PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
00951 extern PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dv;
00952 extern PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
00953 extern PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
00954 extern PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
00955 extern PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dv;
00956 extern PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
00957 extern PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
00958 extern PFNGLPROGRAMUNIFORMI64NVPROC glProgramUniformi64NV;
00959 extern PFNGLPROGRAMUNIFORMI64NVPROC glProgramUniformi64NV;
00960 extern PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
00961 extern PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC glPushClientAttribDefaultEXT;
00962 extern PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
00963 extern PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
00964 extern PFNGLQUERYCOUNTERPROC glQueryCounter;
00965 extern PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
00966 extern PFNGLREADBUFFERPROC glReadBuffer;
00967 extern PFNGLREADNPIXELSLARBPROC glReadnPixelsARB;
00968 extern PFNGLREADNPIXELSPROC glReadnPixels;
00969 extern PFNGLREADPIXELSPROC glReadPixels;
00970 extern PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
00971 extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
00972 extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
00973 extern PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
00974 extern PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
00975 extern PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
00976 extern PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
00977 extern PFNGLSAMPLEMASKIPROC glSampleMaski;
00978 extern PFNGLSAMPLERPARAMETERFFPROC glSamplerParameterf;
00979 extern PFNGLSAMPLERPARAMETERRFVPROC glSamplerParameterfv;
00980 extern PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameteriiv;
00981 extern PFNGLSAMPLERPARAMETERIIPROC glSamplerParameteri;
00982 extern PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
00983 extern PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameteriv;
00984 extern PFNGLSCISSORARRAYVPROC glScissorArrayv;
00985 extern PFNGLSCISSORINDEXEDPROC glScissorIndexed;
00986 extern PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
00987 extern PFNGLSCISSORPROC glScissor;
00988 extern PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
00989 extern PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
00990 extern PFNGLSHADERBINARYPROC glShaderBinary;
00991 extern PFNGLSHADERSOURCEPROC glShaderSource;
00992 extern PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
00993 extern PFNGLSIGNALKVFENCENVPROC glSignalVkFenceNV;
00994 extern PFNGLSIGNALKSEMAPHORENVPROC glSignalVkSemaphoreNV;
00995 extern PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
00996 extern PFNGLSTATECAPTURENVPROC glStateCaptureNV;
00997 extern PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
00998 extern PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
00999 extern PFNGLSTENCILFUNCPROC glStencilFunc;
01000 extern PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
01001 extern PFNGLSTENCILMASKPROC glStencilMask;
01002 extern PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
01003 extern PFNGLSTENCILOPPROC glStencilOp;
01004 extern PFNGLSTENCILOPSEPARATEPROC glStencilOpSeparate;
01005 extern PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
01006 extern PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
01007 extern PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
01008 extern PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
01009 extern PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
01010 extern PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
01011 extern PFNGLSUBPIXELPRECISIONIASNVPROC glSubpixelPrecisionBiasNV;
01012 extern PFNGLTEXBUFFERARBPROC glTexBufferARB;
01013 extern PFNGLTEXBUFFERPROC glTexBuffer;
01014 extern PFNGLTEXBUFFERRANGEPROC glTexBufferSize;
01015 extern PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
01016 extern PFNGLTEXIMAGE1DPROC glTexImage1D;
01017 extern PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
01018 extern PFNGLTEXIMAGE2DPROC glTexImage2D;

```

```

01019 extern PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
01020 extern PFNGLTEXIMAGE3DPROC glTexImage3D;
01021 extern PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
01022 extern PFNGLTEXPARAMETERFPROC glTexParameterf;
01023 extern PFNGLTEXPARAMETERFVPROC glTexParameterfv;
01024 extern PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
01025 extern PFNGLTEXPARAMETERIIPROC glTexParameterIi;
01026 extern PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
01027 extern PFNGLTEXPARAMETERIVPROC glTexParameteriv;
01028 extern PFNGLTEXSTORAGE1DPROC glTexStorage1D;
01029 extern PFNGLTEXSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
01030 extern PFNGLTEXSTORAGE2DPROC glTexStorage2D;
01031 extern PFNGLTEXSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
01032 extern PFNGLTEXSTORAGE3DPROC glTexStorage3D;
01033 extern PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
01034 extern PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
01035 extern PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
01036 extern PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
01037 extern PFNGLTEXTUREBARRIERPROC glTextureBarrier;
01038 extern PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
01039 extern PFNGLTEXTUREBUFFERPROC glTextureBuffer;
01040 extern PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
01041 extern PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
01042 extern PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
01043 extern PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
01044 extern PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
01045 extern PFNGLTEXTUREPAGECOMMITMENTEXTPROC glTexturePageCommitmentEXT;
01046 extern PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
01047 extern PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
01048 extern PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
01049 extern PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
01050 extern PFNGLTEXTUREPARAMETERIEEXTPROC glTextureParameteriEXT;
01051 extern PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIivEXT;
01052 extern PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiv;
01053 extern PFNGLTEXTUREPARAMETERIIPROC glTextureParameteri;
01054 extern PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01055 extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01056 extern PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
01057 extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
01058 extern PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
01059 extern PFNGLTEXTURERESTORAGE1DEXTPROC glTextureStorage1DEXT;
01060 extern PFNGLTEXTURERESTORAGE1DPROC glTextureStorage1D;
01061 extern PFNGLTEXTURERESTORAGE2DEXTPROC glTextureStorage2DEXT;
01062 extern PFNGLTEXTURERESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
01063 extern PFNGLTEXTURERESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
01064 extern PFNGLTEXTURERESTORAGE2DPROC glTextureStorage2D;
01065 extern PFNGLTEXTURERESTORAGE3DEXTPROC glTextureStorage3DEXT;
01066 extern PFNGLTEXTURERESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
01067 extern PFNGLTEXTURERESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
01068 extern PFNGLTEXTURERESTORAGE3DPROC glTextureStorage3D;
01069 extern PFNGLTEXTURESUBIMAGE1DDEXTPROC glTextureSubImage1DEXT;
01070 extern PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
01071 extern PFNGLTEXTURESUBIMAGE2DDEXTPROC glTextureSubImage2DEXT;
01072 extern PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
01073 extern PFNGLTEXTURESUBIMAGE3DDEXTPROC glTextureSubImage3DEXT;
01074 extern PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
01075 extern PFNGLTEXTUREVIEWPROC glTextureView;
01076 extern PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
01077 extern PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
01078 extern PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
01079 extern PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
01080 extern PFNGLUNIFORM1DPROC glUniform1d;
01081 extern PFNGLUNIFORM1DVPROC glUniform1dv;
01082 extern PFNGLUNIFORM1FPROC glUniform1f;
01083 extern PFNGLUNIFORM1FVPROC glUniform1fv;
01084 extern PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
01085 extern PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
01086 extern PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
01087 extern PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
01088 extern PFNGLUNIFORM1IIPROC glUniform1i;
01089 extern PFNGLUNIFORM1IVPROC glUniform1iv;
01090 extern PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
01091 extern PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
01092 extern PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
01093 extern PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
01094 extern PFNGLUNIFORM1UIIPROC glUniform1ui;
01095 extern PFNGLUNIFORM1UIVPROC glUniform1uiv;
01096 extern PFNGLUNIFORM2DPROC glUniform2d;
01097 extern PFNGLUNIFORM2DVPROC glUniform2dv;
01098 extern PFNGLUNIFORM2FPROC glUniform2f;
01099 extern PFNGLUNIFORM2FVPROC glUniform2fv;
01100 extern PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
01101 extern PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
01102 extern PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
01103 extern PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
01104 extern PFNGLUNIFORM2IIPROC glUniform2i;
01105 extern PFNGLUNIFORM2IVPROC glUniform2iv;

```

```

01106 extern PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
01107 extern PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
01108 extern PFNGLUNIFORM2UI64VARBPROC glUniform2ui64VARB;
01109 extern PFNGLUNIFORM2UI64VNVPROC glUniform2ui64VN;
01110 extern PFNGLUNIFORM2UIPROC glUniform2ui;
01111 extern PFNGLUNIFORM2UIVPROC glUniform2uiv;
01112 extern PFNGLUNIFORM3DPROC glUniform3d;
01113 extern PFNGLUNIFORM3DVPROC glUniform3dv;
01114 extern PFNGLUNIFORM3FPROC glUniform3f;
01115 extern PFNGLUNIFORM3FVPROC glUniform3fv;
01116 extern PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
01117 extern PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
01118 extern PFNGLUNIFORM3I64VARBPROC glUniform3i64VARB;
01119 extern PFNGLUNIFORM3I64VNVPROC glUniform3i64VN;
01120 extern PFNGLUNIFORM3IPROC glUniform3i;
01121 extern PFNGLUNIFORM3IVPROC glUniform3iv;
01122 extern PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
01123 extern PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
01124 extern PFNGLUNIFORM3UI64VARBPROC glUniform3ui64VARB;
01125 extern PFNGLUNIFORM3UI64VNVPROC glUniform3ui64VN;
01126 extern PFNGLUNIFORM3UIPROC glUniform3ui;
01127 extern PFNGLUNIFORM3UIVPROC glUniform3uiv;
01128 extern PFNGLUNIFORM4DPROC glUniform4d;
01129 extern PFNGLUNIFORM4DVPROC glUniform4dv;
01130 extern PFNGLUNIFORM4FPROC glUniform4f;
01131 extern PFNGLUNIFORM4FVPROC glUniform4fv;
01132 extern PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
01133 extern PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
01134 extern PFNGLUNIFORM4I64VARBPROC glUniform4i64VARB;
01135 extern PFNGLUNIFORM4I64VNVPROC glUniform4i64VN;
01136 extern PFNGLUNIFORM4IPROC glUniform4i;
01137 extern PFNGLUNIFORM4IVPROC glUniform4iv;
01138 extern PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
01139 extern PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
01140 extern PFNGLUNIFORM4UI64VARBPROC glUniform4ui64VARB;
01141 extern PFNGLUNIFORM4UI64VNVPROC glUniform4ui64VN;
01142 extern PFNGLUNIFORM4UIPROC glUniform4ui;
01143 extern PFNGLUNIFORM4UIVPROC glUniform4uiv;
01144 extern PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
01145 extern PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
01146 extern PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
01147 extern PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64VARB;
01148 extern PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64VN;
01149 extern PFNGLUNIFORMMATRIX2DPROC glUniformMatrix2dv;
01150 extern PFNGLUNIFORMMATRIX2FPROC glUniformMatrix2fv;
01151 extern PFNGLUNIFORMMATRIX2X3DPROC glUniformMatrix2x3dv;
01152 extern PFNGLUNIFORMMATRIX2X3FPROC glUniformMatrix2x3fv;
01153 extern PFNGLUNIFORMMATRIX2X4DPROC glUniformMatrix2x4dv;
01154 extern PFNGLUNIFORMMATRIX2X4FPROC glUniformMatrix2x4fv;
01155 extern PFNGLUNIFORMMATRIX3DPROC glUniformMatrix3dv;
01156 extern PFNGLUNIFORMMATRIX3FPROC glUniformMatrix3fv;
01157 extern PFNGLUNIFORMMATRIX3X2DPROC glUniformMatrix3x2dv;
01158 extern PFNGLUNIFORMMATRIX3X2FPROC glUniformMatrix3x2fv;
01159 extern PFNGLUNIFORMMATRIX3X4DPROC glUniformMatrix3x4dv;
01160 extern PFNGLUNIFORMMATRIX3X4FPROC glUniformMatrix3x4fv;
01161 extern PFNGLUNIFORMMATRIX4DPROC glUniformMatrix4dv;
01162 extern PFNGLUNIFORMMATRIX4FPROC glUniformMatrix4fv;
01163 extern PFNGLUNIFORMMATRIX4X2DPROC glUniformMatrix4x2dv;
01164 extern PFNGLUNIFORMMATRIX4X2FPROC glUniformMatrix4x2fv;
01165 extern PFNGLUNIFORMMATRIX4X3DPROC glUniformMatrix4x3dv;
01166 extern PFNGLUNIFORMMATRIX4X3FPROC glUniformMatrix4x3fv;
01167 extern PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
01168 extern PFNGLUNIFORMUI64NVPROC glUniformui64NV;
01169 extern PFNGLUNIFORMUI64VNVPROC glUniformui64VN;
01170 extern PFNGLUNMAPBUFFERPROC glUnmapBuffer;
01171 extern PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
01172 extern PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
01173 extern PFNGLUSEPROGRAMPROC glUseProgram;
01174 extern PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
01175 extern PFNGLUSESHEADERPROGRAMEXTPROC glUseShaderProgramEXT;
01176 extern PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
01177 extern PFNGLVALIDATEPROGRAMPROC glValidateProgram;
01178 extern PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
01179 extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
01180 extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribIFormat;
01181 extern PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
01182 extern PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
01183 extern PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
01184 extern PFNGLVERTEXARRAYCOLOROFFSETSETEXTPROC glVertexArrayColorOffsetEXT;
01185 extern PFNGLVERTEXARRAYEDGEFLAGOFFSETSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
01186 extern PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
01187 extern PFNGLVERTEXARRAYFOGCOORDOFFSETSETEXTPROC glVertexArrayFogCoordOffsetEXT;
01188 extern PFNGLVERTEXARRAYINDEXOFFSETSETEXTPROC glVertexArrayIndexOffsetEXT;
01189 extern PFNGLVERTEXARRAYMULTITEXCOORDOFFSETSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
01190 extern PFNGLVERTEXARRAYNORMALOFFSETSETEXTPROC glVertexArrayNormalOffsetEXT;
01191 extern PFNGLVERTEXARRAYSECONDARYCOLOROFFSETSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
01192 extern PFNGLVERTEXARRAYTEXCOORDOFFSETSETEXTPROC glVertexArrayTexCoordOffsetEXT;

```

```

01193 extern PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
01194 extern PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
01195 extern PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
01196 extern PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
01197 extern PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
01198 extern PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
01199 extern PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
01200 extern PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
01201 extern PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
01202 extern PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
01203 extern PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
01204 extern PFNGLVERTEXARRAYVERTEXEXOFSETEXTPROC glVertexArrayVertexOffsetEXT;
01205 extern PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
01206 extern PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
01207 extern PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
01208 extern PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
01209 extern PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
01210 extern PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
01211 extern PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
01212 extern PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
01213 extern PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
01214 extern PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
01215 extern PFNGLVERTEXATTRIB2SPROC glVertexAttrib2s;
01216 extern PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
01217 extern PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
01218 extern PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
01219 extern PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
01220 extern PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
01221 extern PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
01222 extern PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
01223 extern PFNGLVERTEXATTRIB4BPROC glVertexAttrib4b;
01224 extern PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
01225 extern PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
01226 extern PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
01227 extern PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
01228 extern PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
01229 extern PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4Nbv;
01230 extern PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4Niv;
01231 extern PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4Nsv;
01232 extern PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4Nub;
01233 extern PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4Nubv;
01234 extern PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4Nuiv;
01235 extern PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4Nusv;
01236 extern PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
01237 extern PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
01238 extern PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
01239 extern PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
01240 extern PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
01241 extern PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
01242 extern PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
01243 extern PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
01244 extern PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
01245 extern PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
01246 extern PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
01247 extern PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
01248 extern PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
01249 extern PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
01250 extern PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
01251 extern PFNGLVERTEXATTRIBI2IVPROC glVertexAttribI2iv;
01252 extern PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
01253 extern PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
01254 extern PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
01255 extern PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
01256 extern PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
01257 extern PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
01258 extern PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
01259 extern PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
01260 extern PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
01261 extern PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
01262 extern PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
01263 extern PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
01264 extern PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
01265 extern PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
01266 extern PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
01267 extern PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
01268 extern PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
01269 extern PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
01270 extern PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
01271 extern PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
01272 extern PFNGLVERTEXATTRIBL1I64NVNPROC glVertexAttribL1i64vNV;
01273 extern PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
01274 extern PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
01275 extern PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
01276 extern PFNGLVERTEXATTRIBL1UI64NVNPROC glVertexAttribL1ui64vNV;
01277 extern PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
01278 extern PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
01279 extern PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;

```

```

01280 extern PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64vNV;
01281 extern PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
01282 extern PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64vNV;
01283 extern PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01284 extern PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
01285 extern PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
01286 extern PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
01287 extern PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64vNV;
01288 extern PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01289 extern PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
01290 extern PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
01291 extern PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
01292 extern PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64vNV;
01293 extern PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
01294 extern PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64vNV;
01295 extern PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
01296 extern PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
01297 extern PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
01298 extern PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
01299 extern PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
01300 extern PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
01301 extern PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
01302 extern PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
01303 extern PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
01304 extern PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
01305 extern PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
01306 extern PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
01307 extern PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
01308 extern PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
01309 extern PFNGLVIEWPORTARRAYVPROC glViewportArrayv;
01310 extern PFNGLVIEWPORTINDEXEDFPROC glViewportIndexedf;
01311 extern PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;
01312 extern PFNGLVIEWPORTPOSITIONWSCALENVPROC glViewportPositionWScaleNV;
01313 extern PFNGLVIEWPORTPROC glViewport;
01314 extern PFNGLVIEWPORTSWIZZLENVPROC glViewportSwizzleNV;
01315 extern PFNGLWAITSYNCPROC glWaitSync;
01316 extern PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
01317 extern PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
01318 extern PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
01319 #endif
01320
01322
01323 // 標準ライブラリ
01324 #include <cmath>
01325 #include <array>
01326 #include <vector>
01327 #include <string>
01328 #include <memory>
01329
01330 namespace gg
01331 {
01332     enum BindingPoints
01333     {
01334         LightBindingPoint = 0,
01335         MaterialBindingPoint,
01336     };
01337
01338     extern GLint ggBufferAlignment;
01339
01340     extern void ggInit();
01341
01342     extern void _ggError(const std::string& name = "", unsigned int line = 0);
01343
01344 #if defined(DEBUG)
01345     # define ggError() gg::_ggError(__FILE__, __LINE__)
01346 #else
01347     # define ggError()
01348 #endif
01349
01350     extern void _ggFBOError(const std::string& name = "", unsigned int line = 0);
01351
01352 #if defined(DEBUG)
01353     # define ggFBOError() gg::_ggFBOError(__FILE__, __LINE__)
01354 #else
01355     # define ggFBOError()
01356 #endif
01357
01358     inline void ggCross(GLfloat* c, const GLfloat* a, const GLfloat* b)
01359     {
01360         c[0] = a[1] * b[2] - a[2] * b[1];
01361         c[1] = a[2] * b[0] - a[0] * b[2];
01362         c[2] = a[0] * b[1] - a[1] * b[0];
01363     }
01364
01365     inline GLfloat ggDot3(const GLfloat* a, const GLfloat* b)
01366     {
01367         return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
01368     }

```

```

01435 }
01436
01437 inline GLfloat ggLength3(const GLfloat* a)
01438 {
01439     return sqrtf(ggDot3(a, a));
01440 }
01441
01442 inline GLfloat ggDistance3(const GLfloat* a, const GLfloat* b)
01443 {
01444     const GLfloat c[] { a[0] - b[0], a[1] - b[1], a[2] - b[2], 0.0f };
01445     return ggLength3(c);
01446 }
01447
01448 inline void ggNormalize3(const GLfloat* a, GLfloat* b)
01449 {
01450     const GLfloat l { ggLength3(a) };
01451     b[0] = a[0] / l;
01452     b[1] = a[1] / l;
01453     b[2] = a[2] / l;
01454 }
01455
01456 inline void ggNormalize3(GLfloat* a)
01457 {
01458     const GLfloat l { ggLength3(a) };
01459     a[0] /= l;
01460     a[1] /= l;
01461     a[2] /= l;
01462 }
01463
01464 inline GLfloat ggDot4(const GLfloat* a, const GLfloat* b)
01465 {
01466     return a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + a[3] * b[3];
01467 }
01468
01469 inline GLfloat ggLength4(const GLfloat* a)
01470 {
01471     return sqrtf(ggDot4(a, a));
01472 }
01473
01474 inline GLfloat ggDistance4(const GLfloat* a, const GLfloat* b)
01475 {
01476     const GLfloat c[] { a[0] - b[0], a[1] - b[1], a[2] - b[2], a[3] - b[3] };
01477     return ggLength4(c);
01478 }
01479
01480 inline void ggNormalize4(const GLfloat* a, GLfloat* b)
01481 {
01482     const GLfloat l { ggLength4(a) };
01483     b[0] = a[0] / l;
01484     b[1] = a[1] / l;
01485     b[2] = a[2] / l;
01486     b[3] = a[3] / l;
01487 }
01488
01489 inline void ggNormalize4(GLfloat* a)
01490 {
01491     const GLfloat l { ggLength4(a) };
01492     a[0] /= l;
01493     a[1] /= l;
01494     a[2] /= l;
01495     a[3] /= l;
01496 }
01497
01498 class GgVector : public std::array<GLfloat, 4>
01499 {
01500     public:
01501
01502     GgVector()
01503     {
01504     }
01505
01506     constexpr GgVector(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3) :
01507         std::array<GLfloat, 4> { v0, v1, v2, v3 }
01508     {
01509     }
01510
01511     constexpr GgVector(GLfloat c) :
01512         GgVector { c, c, c, c }
01513     {
01514     }
01515
01516     constexpr GgVector(const GLfloat* a) :
01517         GgVector { a[0], a[1], a[2], a[3] }
01518     {
01519     }
01520
01521     GgVector operator+(const GgVector& v) const

```

```
01607  {
01608      return GgVector{ data()[0] + v[0], data()[1] + v[1], data()[2] + v[2], data()[3] + v[3] };
01609  }
01610
01617  GgVector operator+(GLfloat c) const
01618  {
01619     return GgVector{ data()[0] + c, data()[1] + c, data()[2] + c, data()[3] + c };
01620  }
01621
01628  GgVector& operator+=(const GgVector& v)
01629  {
01630     data()[0] += v[0];
01631     data()[1] += v[1];
01632     data()[2] += v[2];
01633     data()[3] += v[3];
01634     return *this;
01635  }
01636
01643  GgVector& operator+=(GLfloat c)
01644  {
01645     data()[0] += c;
01646     data()[1] += c;
01647     data()[2] += c;
01648     data()[3] += c;
01649     return *this;
01650  }
01651
01658  GgVector operator-(const GgVector& v) const
01659  {
01660     return GgVector{ data()[0] - v[0], data()[1] - v[1], data()[2] - v[2], data()[3] - v[3] };
01661  }
01662
01669  GgVector operator-(GLfloat c) const
01670  {
01671     return GgVector{ data()[0] - c, data()[1] - c, data()[2] - c, data()[3] - c };
01672  }
01673
01680  GgVector& operator-=(const GgVector& v)
01681  {
01682     data()[0] -= v[0];
01683     data()[1] -= v[1];
01684     data()[2] -= v[2];
01685     data()[3] -= v[3];
01686     return *this;
01687  }
01688
01695  GgVector& operator-=(GLfloat c)
01696  {
01697     data()[0] -= c;
01698     data()[1] -= c;
01699     data()[2] -= c;
01700     data()[3] -= c;
01701     return *this;
01702  }
01703
01710  GgVector operator*(const GgVector& v) const
01711  {
01712     return GgVector{ data()[0] * v[0], data()[1] * v[1], data()[2] * v[2], data()[3] * v[3] };
01713  }
01714
01721  GgVector operator*(GLfloat c) const
01722  {
01723     return GgVector{ data()[0] * c, data()[1] * c, data()[2] * c, data()[3] * c };
01724  }
01725
01731  GgVector& operator*=(const GgVector& v)
01732  {
01733     data()[0] *= v[0];
01734     data()[1] *= v[1];
01735     data()[2] *= v[2];
01736     data()[3] *= v[3];
01737     return *this;
01738  }
01739
01746  GgVector& operator*=(GLfloat c)
01747  {
01748     data()[0] *= c;
01749     data()[1] *= c;
01750     data()[2] *= c;
01751     data()[3] *= c;
01752     return *this;
01753  }
01754
01761  GgVector operator/(const GgVector& v) const
01762  {
01763     return GgVector{ data()[0] / v[0], data()[1] / v[1], data()[2] / v[2], data()[3] / v[3] };
01764  }
```

```

01765
01772     GgVector& operator/=(GgVector& v)
01773     {
01774         data()[0] /= v[0];
01775         data()[1] /= v[1];
01776         data()[2] /= v[2];
01777         data()[3] /= v[3];
01778         return *this;
01779     }
01780
01787     GgVector operator/(GLfloat c) const
01788     {
01789         return GgVector{ data()[0] / c, data()[1] / c, data()[2] / c, data()[3] / c };
01790     }
01791
01798     GgVector& operator/=(GLfloat c)
01799     {
01800         data()[0] /= c;
01801         data()[1] /= c;
01802         data()[2] /= c;
01803         data()[3] /= c;
01804         return *this;
01805     }
01806
01813     GLfloat dot3(const GgVector& v) const
01814     {
01815         return ggDot3(data(), v.data());
01816     }
01817
01823     GLfloat length3() const
01824     {
01825         return ggLength3(data());
01826     }
01827
01834     GLfloat distance3(const GgVector& v) const
01835     {
01836         return ggDistance3(data(), v.data());
01837     }
01838
01844     GgVector normalize3() const
01845     {
01846         GgVector b;
01847         ggNormalize3(data(), b.data());
01848         return b;
01849     }
01850
01857     GLfloat dot4(const GgVector& v) const
01858     {
01859         return ggDot4(data(), v.data());
01860     }
01861
01867     GLfloat length4() const
01868     {
01869         return ggLength4(data());
01870     }
01871
01878     GLfloat distance4(const GgVector& v) const
01879     {
01880         return ggDistance4(data(), v.data());
01881     }
01882
01888     GgVector normalize4() const
01889     {
01890         GgVector b;
01891         ggNormalize4(data(), b.data());
01892         return b;
01893     }
01894 };
01895
01902     inline const GgVector& operator+(const GgVector& v)
01903     {
01904         return v;
01905     }
01906
01914     inline GgVector operator+(GLfloat a, const GgVector& b)
01915     {
01916         return GgVector{ a + b[0], a + b[1], a + b[2], a + b[3] };
01917     }
01918
01925     inline const GgVector operator-(const GgVector& v)
01926     {
01927         return GgVector{ -v[0], -v[1], -v[2], -v[3] };
01928     }
01929
01937     inline GgVector operator-(GLfloat a, const GgVector& b)
01938     {
01939         return GgVector{ a - b[0], a - b[1], a - b[2], a - b[3] };
01940     }

```

```
01940 }
01941
01949 inline GgVector operator*(GLfloat a, const GgVector& b)
01950 {
01951     return GgVector{ a * b[0], a * b[1], a * b[2], a * b[3] };
01952 }
01953
01961 inline GgVector operator/(GLfloat a, const GgVector& b)
01962 {
01963     return GgVector{ a / b[0], a / b[1], a / b[2], a / b[3] };
01964 }
01965
01976 inline GgVector ggCross(const GgVector& a, const GgVector& b)
01977 {
01978     GgVector c;
01979     ggCross(c.data(), a.data(), b.data());
01980     return c;
01981 }
01982
01990 inline GLfloat ggDot3(const GgVector& a, const GgVector& b)
01991 {
01992     return ggDot3(a.data(), b.data());
01993 }
01994
02001 inline GLfloat ggLength3(const GgVector& a)
02002 {
02003     return ggLength3(a.data());
02004 }
02005
02013 inline GLfloat ggDistance3(const GgVector& a, const GgVector& b)
02014 {
02015     return ggDistance3(a.data(), b.data());
02016 }
02017
02027 inline GgVector ggNormalize3(const GgVector& a)
02028 {
02029     GgVector b;
02030     ggNormalize3(a.data(), b.data());
02031     b.data()[3] = 0.0f;
02032     return b;
02033 }
02034
02043 inline void ggNormalize3(GgVector* a)
02044 {
02045     ggNormalize3(a->data());
02046     a->data()[3] = 0.0f;
02047 }
02048
02056 inline GLfloat ggDot4(const GgVector& a, const GgVector& b)
02057 {
02058     return ggDot4(a.data(), b.data());
02059 }
02060
02067 inline GLfloat ggLength4(const GgVector& a)
02068 {
02069     return ggLength4(a.data());
02070 }
02071
02079 inline GLfloat ggDistance4(const GgVector& a, const GgVector& b)
02080 {
02081     return ggDistance4(a.data(), b.data());
02082 }
02083
02090 inline GgVector ggNormalize4(const GgVector& a)
02091 {
02092     GgVector b;
02093     ggNormalize4(a.data(), b.data());
02094     return b;
02095 }
02096
02102 inline void ggNormalize4(GgVector* a)
02103 {
02104     ggNormalize4(a->data());
02105 }
02106
02110 class GgMatrix : public std::array<GLfloat, 16>
02111 {
02112     // 行列 a とベクトル b の積をベクトル c に代入する
02113     void projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02114
02115     // 行列 a と行列 b の積を行列 c に代入する
02116     void multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const;
02117
02118 public:
02119
02123     GgMatrix()
02124 {
```

```

02125     }
02126
02127     constexpr GgMatrix(
02128         GLfloat m00, GLfloat m01, GLfloat m02, GLfloat m03,
02129         GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13,
02130         GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23,
02131         GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33
02132     ) :
02133         std::array<GLfloat, 16>{ m00, m10, m20, m30, m01, m11, m21, m31, m02, m12, m22, m32, m03, m13,
02134         m23, m33 }
02135     {
02136     }
02137
02138     constexpr GgMatrix(GLfloat c) :
02139         GgMatrix{ c, c }
02140     {
02141     }
02142
02143     GgMatrix(const GLfloat* a)
02144     {
02145         operator=(a);
02146     }
02147
02148     GgMatrix& operator=(const GLfloat* a)
02149     {
02150         std::copy(a, a + size(), data());
02151         return this;
02152     }
02153
02154     GgMatrix operator+(const GLfloat* a) const
02155     {
02156         GgMatrix t;
02157         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] + a[i];
02158         return t;
02159     }
02160
02161     GgMatrix operator+(const GgMatrix& m) const
02162     {
02163         return operator+(m.data());
02164     }
02165
02166     GgMatrix& operator+=(const GLfloat* a)
02167     {
02168         for (std::size_t i = 0; i < size(); ++i) data()[i] += a[i];
02169         return this;
02170     }
02171
02172     GgMatrix& operator+=(const GgMatrix& m)
02173     {
02174         return operator+=(m.data());
02175     }
02176
02177     GgMatrix operator-(const GLfloat* a) const
02178     {
02179         GgMatrix t;
02180         for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] - a[i];
02181         return t;
02182     }
02183
02184     GgMatrix operator-(const GgMatrix& m) const
02185     {
02186         return operator-(m.data());
02187     }
02188
02189     GgMatrix& operator+=(const GgMatrix& m)
02190     {
02191         for (std::size_t i = 0; i < size(); ++i) data()[i] -= m[i];
02192         return this;
02193     }
02194
02195     GgMatrix operator*(const GLfloat* a) const
02196     {
02197         GgMatrix t;
02198         multiply(t.data(), data(), a);
02199         return t;
02200     }
02201
02202     GgMatrix operator*(const GgMatrix& m) const
02203     {
02204         return operator*(m.data());
02205     }
02206
02207     GgMatrix& operator+=(const GgMatrix& m)
02208     {
02209         for (std::size_t i = 0; i < size(); ++i) data()[i] += m[i];
02210         return this;
02211     }
02212
02213     GgMatrix& operator+=(const GgMatrix& m)
02214     {
02215         for (std::size_t i = 0; i < size(); ++i) data()[i] -= m[i];
02216         return this;
02217     }
02218
02219     GgMatrix& operator+=(const GgMatrix& m)
02220     {
02221         for (std::size_t i = 0; i < size(); ++i) data()[i] *= m[i];
02222         return this;
02223     }
02224
02225     GgMatrix& operator+=(const GgMatrix& m)
02226     {
02227         for (std::size_t i = 0; i < size(); ++i) data()[i] /= m[i];
02228         return this;
02229     }
02230
02231     GgMatrix& operator+=(const GgMatrix& m)
02232     {
02233         for (std::size_t i = 0; i < size(); ++i) data()[i] = m[i];
02234         return this;
02235     }
02236
02237     GgMatrix& operator+=(const GgMatrix& m)
02238     {
02239         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02240         return this;
02241     }
02242
02243     GgMatrix& operator+=(const GgMatrix& m)
02244     {
02245         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02246         return this;
02247     }
02248
02249     GgMatrix& operator+=(const GgMatrix& m)
02250     {
02251         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02252         return this;
02253     }
02254
02255     GgMatrix& operator+=(const GgMatrix& m)
02256     {
02257         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02258         return this;
02259     }
02260
02261     GgMatrix& operator+=(const GgMatrix& m)
02262     {
02263         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02264         return this;
02265     }
02266
02267     GgMatrix& operator+=(const GgMatrix& m)
02268     {
02269         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02270         return this;
02271     }
02272
02273     GgMatrix& operator+=(const GgMatrix& m)
02274     {
02275         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02276         return this;
02277     }
02278
02279     GgMatrix& operator+=(const GgMatrix& m)
02280     {
02281         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02282         return this;
02283     }
02284
02285     GgMatrix& operator+=(const GgMatrix& m)
02286     {
02287         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02288         return this;
02289     }
02290
02291     GgMatrix& operator+=(const GgMatrix& m)
02292     {
02293         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02294         return this;
02295     }
02296
02297     GgMatrix& operator+=(const GgMatrix& m)
02298     {
02299         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02300         return this;
02301     }
02302
02303     GgMatrix& operator+=(const GgMatrix& m)
02304     {
02305         for (std::size_t i = 0; i < size(); ++i) data()[i] = 1.0f / m[i];
02306         return this;
02307     }

```

```
02313     GgMatrix& operator*=(const GLfloat* a)
02314     {
02315         *this = operator*(a);
02316         return *this;
02317     }
02318
02319     GgMatrix& operator*=(const GgMatrix& m)
02320     {
02321         return operator*=(m.data());
02322     }
02323
02324     GgMatrix operator/(const GLfloat* a) const
02325     {
02326         GgMatrix i, t;
02327         i.loadInvert(a);
02328         multiply(t.data(), data(), i.data());
02329         return t;
02330     }
02331
02332     GgMatrix operator/(const GgMatrix& m) const
02333     {
02334         return operator/(m.data());
02335     }
02336
02337     GgMatrix& operator/=(const GLfloat* a)
02338     {
02339         *this = operator/(a);
02340         return *this;
02341     }
02342
02343     GgMatrix& operator/=(const GgMatrix& m)
02344     {
02345         return operator/=(m.data());
02346     }
02347
02348     GgMatrix& loadIdentity();
02349
02350     GgMatrix& loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02351
02352     GgMatrix& loadTranslate(const GLfloat* t)
02353     {
02354         return loadTranslate(t[0], t[1], t[2]);
02355     }
02356
02357     GgMatrix& loadTranslate(const GgVector& t)
02358     {
02359         return loadTranslate(t[0], t[1], t[2], t[3]);
02360     }
02361
02362     GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);
02363
02364     GgMatrix& loadScale(const GLfloat* s)
02365     {
02366         return loadScale(s[0], s[1], s[2]);
02367     }
02368
02369     GgMatrix& loadScale(const GgVector& s)
02370     {
02371         return loadScale(s[0], s[1], s[2], s[3]);
02372     }
02373
02374     GgMatrix& loadRotateX(GLfloat a);
02375
02376     GgMatrix& loadRotateY(GLfloat a);
02377
02378     GgMatrix& loadRotateZ(GLfloat a);
02379
02380     GgMatrix& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
02381
02382     GgMatrix& loadRotate(const GLfloat* r, GLfloat a)
02383     {
02384         return loadRotate(r[0], r[1], r[2], a);
02385     }
02386
02387     GgMatrix& loadRotate(const GgVector& r, GLfloat a)
02388     {
02389         return loadRotate(r[0], r[1], r[2], a);
02390     }
02391
02392     GgMatrix& loadRotate(const GLfloat* r)
02393     {
02394         return loadRotate(r[0], r[1], r[2], r[3]);
02395     }
02396
02397     GgMatrix& loadRotate(const GgVector& r)
02398     {
02399         return loadRotate(r[0], r[1], r[2], r[3]);
02400     }
```

```

02528     }
02529
02544     GgMatrix& loadLookat(GLfloat ex, GLfloat ey, GLfloat ez,
02545         GLfloat tx, GLfloat ty, GLfloat tz,
02546         GLfloat ux, GLfloat uy, GLfloat uz);
02547
02556     GgMatrix& loadLookat(const GLfloat* e, const GLfloat* t, const GLfloat* u)
02557     {
02558         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02559     }
02560
02569     GgMatrix& loadLookat(const GgVector& e, const GgVector& t, const GgVector& u)
02570     {
02571         return loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02572     }
02573
02585     GgMatrix& loadOrthogonal(GLfloat left, GLfloat right,
02586         GLfloat bottom, GLfloat top,
02587         GLfloat zNear, GLfloat zFar);
02588
02600     GgMatrix& loadFrustum(GLfloat left, GLfloat right,
02601         GLfloat bottom, GLfloat top,
02602         GLfloat zNear, GLfloat zFar);
02603
02613     GgMatrix& loadPerspective(GLfloat fovy, GLfloat aspect,
02614         GLfloat zNear, GLfloat zFar);
02615
02622     GgMatrix& loadTranspose(const GLfloat* a);
02623
02630     GgMatrix& loadTranspose(const GgMatrix& m)
02631     {
02632         return loadTranspose(m.data());
02633     }
02634
02641     GgMatrix& loadInvert(const GLfloat* a);
02642
02649     GgMatrix& loadInvert(const GgMatrix& m)
02650     {
02651         return loadInvert(m.data());
02652     }
02653
02660     GgMatrix& loadNormal(const GLfloat* a);
02661
02668     GgMatrix& loadNormal(const GgMatrix& m)
02669     {
02670         return loadNormal(m.data());
02671     }
02672
02682     GgMatrix translate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02683     {
02684         GgMatrix m;
02685         return operator*(m.loadTranslate(x, y, z, w));
02686     }
02687
02694     GgMatrix translate(const GLfloat* t) const
02695     {
02696         return translate(t[0], t[1], t[2]);
02697     }
02698
02705     GgMatrix translate(const GgVector& t) const
02706     {
02707         return translate(t[0], t[1], t[2], t[3]);
02708     }
02709
02719     GgMatrix scale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f) const
02720     {
02721         GgMatrix m;
02722         return operator*(m.loadScale(x, y, z, w));
02723     }
02724
02731     GgMatrix scale(const GLfloat* s) const
02732     {
02733         return scale(s[0], s[1], s[2]);
02734     }
02735
02742     GgMatrix scale(const GgVector& s) const
02743     {
02744         return scale(s[0], s[1], s[2], s[3]);
02745     }
02746
02753     GgMatrix rotateX(GLfloat a) const
02754     {
02755         GgMatrix m;
02756         return operator*(m.loadRotateX(a));
02757     }
02758
02765     GgMatrix rotateY(GLfloat a) const

```

```
02766  {
02767      GgMatrix m;
02768      return operator*(m.loadRotateY(a));
02769  }
02770
02771  GgMatrix rotateZ(GLfloat a) const
02772  {
02773      GgMatrix m;
02774      return operator*(m.loadRotateZ(a));
02775  }
02776
02777  GgMatrix rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
02778  {
02779      GgMatrix m;
02780      return operator*(m.loadRotate(x, y, z, a));
02781  }
02782
02783  GgMatrix rotate(const GLfloat* r, GLfloat a) const
02784  {
02785      return rotate(r[0], r[1], r[2], a);
02786  }
02787
02788  GgMatrix rotate(const GgVector& r, GLfloat a) const
02789  {
02790      return rotate(r[0], r[1], r[2], a);
02791  }
02792
02793  GgMatrix rotate(const GLfloat* r) const
02794  {
02795      return rotate(r[0], r[1], r[2], r[3]);
02796  }
02797
02798  GgMatrix rotate(const GgVector& r) const
02799  {
02800      return rotate(r[0], r[1], r[2], r[3]);
02801  }
02802
02803  GgMatrix lookat(
02804      GLfloat ex, GLfloat ey, GLfloat ez,
02805      GLfloat tx, GLfloat ty, GLfloat tz,
02806      GLfloat ux, GLfloat uy, GLfloat uz
02807  ) const
02808  {
02809      GgMatrix m;
02810      return operator*(m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz));
02811  }
02812
02813  GgMatrix lookat(const GLfloat* e, const GLfloat* t, const GLfloat* u) const
02814  {
02815      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02816  }
02817
02818  GgMatrix lookat(const GgVector& e, const GgVector& t, const GgVector& u) const
02819  {
02820      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
02821  }
02822
02823  GgMatrix orthogonal(
02824      GLfloat left, GLfloat right,
02825      GLfloat bottom, GLfloat top,
02826      GLfloat zNear, GLfloat zFar
02827  ) const
02828  {
02829      GgMatrix m;
02830      return operator*(m.loadOrthogonal(left, right, bottom, top, zNear, zFar));
02831  }
02832
02833  GgMatrix frustum(
02834      GLfloat left, GLfloat right,
02835      GLfloat bottom, GLfloat top,
02836      GLfloat zNear, GLfloat zFar
02837  ) const
02838  {
02839      GgMatrix m;
02840      return operator*(m.loadFrustum(left, right, bottom, top, zNear, zFar));
02841  }
02842
02843  GgMatrix perspective(
02844      GLfloat fovy, GLfloat aspect,
02845      GLfloat zNear, GLfloat zFar
02846  ) const
02847  {
02848      GgMatrix m;
02849      return operator*(m.loadPerspective(fovy, aspect, zNear, zFar));
02850  }
02851
02852  GgMatrix transpose() const
```

```

02960  {
02961      GgMatrix t;
02962      return t.loadTranspose(*this);
02963  }
02964
02970  GgMatrix invert() const
02971  {
02972      GgMatrix t;
02973      return t.loadInvert(*this);
02974  }
02975
02981  GgMatrix normal() const
02982  {
02983      GgMatrix t;
02984      return t.loadNormal(*this);
02985  }
02986
02993  void projection(GLfloat* c, const GLfloat* v) const
02994  {
02995      projection(c, data(), v);
02996  }
02997
03004  void projection(GLfloat* c, const GgVector& v) const
03005  {
03006      projection(c, v.data());
03007  }
03008
03015  void projection(GgVector& c, const GLfloat* v) const
03016  {
03017      projection(c.data(), v);
03018  }
03019
03026  void projection(GgVector& c, const GgVector& v) const
03027  {
03028      projection(c.data(), v.data());
03029  }
03030
03037  GgVector operator*(const GgVector& v) const
03038  {
03039      GgVector c;
03040      projection(c, v);
03041      return c;
03042  }
03043
03049  const GLfloat* get() const
03050  {
03051      return data();
03052  }
03053
03059  void get(GLfloat* a) const
03060  {
03061      std::copy(data(), data() + size(), a);
03062  }
03063 };
03064
03070  inline GgMatrix ggIdentity()
03071  {
03072      GgMatrix t;
03073      return t.loadIdentity();
03074  };
03075
03085  inline GgMatrix ggTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03086  {
03087      GgMatrix m;
03088      return m.loadTranslate(x, y, z, w);
03089  }
03090
03097  inline GgMatrix ggTranslate(const GLfloat* t)
03098  {
03099      GgMatrix m;
03100      return m.loadTranslate(t[0], t[1], t[2]);
03101  }
03102
03109  inline GgMatrix ggTranslate(const GgVector& t)
03110  {
03111      GgMatrix m;
03112      return m.loadTranslate(t[0], t[1], t[2], t[3]);
03113  }
03114
03124  inline GgMatrix ggScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
03125  {
03126      GgMatrix m;
03127      return m.loadScale(x, y, z, w);
03128  }
03129
03136  inline GgMatrix ggScale(const GLfloat* s)
03137  {

```

```
03138     GgMatrix m;
03139     return m.loadScale(s[0], s[1], s[2]);
03140 }
03141
03148 inline GgMatrix ggScale(const GgVector& s)
03149 {
03150     GgMatrix m;
03151     return m.loadScale(s[0], s[1], s[2], s[3]);
03152 }
03153
03160 inline GgMatrix ggRotateX(GLfloat a)
03161 {
03162     GgMatrix m;
03163     return m.loadRotateX(a);
03164 }
03165
03172 inline GgMatrix ggRotateY(GLfloat a)
03173 {
03174     GgMatrix m;
03175     return m.loadRotateY(a);
03176 }
03177
03184 inline GgMatrix ggRotateZ(GLfloat a)
03185 {
03186     GgMatrix m;
03187     return m.loadRotateZ(a);
03188 }
03189
03199 inline GgMatrix ggRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
03200 {
03201     GgMatrix m;
03202     return m.loadRotate(x, y, z, a);
03203 }
03204
03212 inline GgMatrix ggRotate(const GLfloat* r, GLfloat a)
03213 {
03214     GgMatrix m;
03215     return m.loadRotate(r[0], r[1], r[2], a);
03216 }
03217
03225 inline GgMatrix ggRotate(const GgVector& r, GLfloat a)
03226 {
03227     GgMatrix m;
03228     return m.loadRotate(r[0], r[1], r[2], a);
03229 }
03230
03237 inline GgMatrix ggRotate(const GLfloat* r)
03238 {
03239     GgMatrix m;
03240     return m.loadRotate(r[0], r[1], r[2], r[3]);
03241 }
03242
03249 inline GgMatrix ggRotate(const GgVector& r)
03250 {
03251     GgMatrix m;
03252     return m.loadRotate(r[0], r[1], r[2], r[3]);
03253 }
03254
03269 inline GgMatrix ggLookat(
03270     GLfloat ex, GLfloat ey, GLfloat ez,      // 視点の位置
03271     GLfloat tx, GLfloat ty, GLfloat tz,      // 目標点の位置
03272     GLfloat ux, GLfloat uy, GLfloat uz       // 上方向のベクトル
03273 )
03274 {
03275     GgMatrix m;
03276     return m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz);
03277 }
03278
03287 inline GgMatrix ggLookat(
03288     const GLfloat* e,                      // 視点の位置
03289     const GLfloat* t,                      // 目標点の位置
03290     const GLfloat* u,                      // 上方向のベクトル
03291 )
03292 {
03293     GgMatrix m;
03294     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03295 }
03296
03305 inline GgMatrix ggLookat(
03306     const GgVector& e,                      // 視点の位置
03307     const GgVector& t,                      // 目標点の位置
03308     const GgVector& u,                      // 上方向のベクトル
03309 )
03310 {
03311     GgMatrix m;
03312     return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
03313 }
```

```

03314
03326     inline GgMatrix ggOrthogonal(GLfloat left, GLfloat right,
03327         GLfloat bottom, GLfloat top,
03328         GLfloat zNear, GLfloat zFar)
03329     {
03330         GgMatrix m;
03331         return m.loadOrthogonal(left, right, bottom, top, zNear, zFar);
03332     }
03333
03345     inline GgMatrix ggFrustum(
03346         GLfloat left, GLfloat right,
03347         GLfloat bottom, GLfloat top,
03348         GLfloat zNear, GLfloat zFar
03349     )
03350     {
03351         GgMatrix m;
03352         return m.loadFrustum(left, right, bottom, top, zNear, zFar);
03353     }
03354
03364     inline GgMatrix ggPerspective(
03365         GLfloat fovy, GLfloat aspect,
03366         GLfloat zNear, GLfloat zFar
03367     )
03368     {
03369         GgMatrix m;
03370         return m.loadPerspective(fovy, aspect, zNear, zFar);
03371     }
03372
03379     inline GgMatrix ggTranspose(const GgMatrix& m)
03380     {
03381         return m.transpose();
03382     }
03383
03390     inline GgMatrix ggInvert(const GgMatrix& m)
03391     {
03392         return m.invert();
03393     }
03394
03401     inline GgMatrix ggNormal(const GgMatrix& m)
03402     {
03403         return m.normal();
03404     }
03405
03409     class GgQuaternion : public GgVector
03410     {
03411         // GgQuaternion 型の四元数 p と四元数 q の積を四元数 r に求める
03412         void multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const;
03413
03414         // GgQuaternion 型の四元数 q が表す回転の変換行列を m に求める
03415         void toMatrix(GLfloat* m, const GLfloat* q) const;
03416
03417         // 回転の変換行列 m が表す四元数を q に求める
03418         void toQuaternion(GLfloat* q, const GLfloat* m) const;
03419
03420         // 球面線形補間 q と r を t で補間した四元数を p に求める
03421         void slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const;
03422
03423     public:
03424
03428         GgQuaternion()
03429     {
03430     }
03431
03440         constexpr GgQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w) :
03441             GgVector{ x, y, z, w }
03442     {
03443     }
03444
03450         constexpr GgQuaternion(GLfloat c) :
03451             GgQuaternion{ c, c, c, c }
03452     {
03453     }
03454
03460         GgQuaternion(const GLfloat* a) :
03461             GgQuaternion{ a[0], a[1], a[2], a[3] }
03462     {
03463     }
03464
03470         GgQuaternion(const GgVector& v) :
03471             GgQuaternion{ v[0], v[1], v[2], v[3] }
03472     {
03473     }
03474
03480         GLfloat norm() const
03481     {
03482         return ggLength4(*this);
03483     }

```

```
03484     GgQuaternion& load(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03494     {
03495         data()[0] = x;
03496         data()[1] = y;
03497         data()[2] = z;
03498         data()[3] = w;
03499         return *this;
03500     }
03502
03509     GgQuaternion& load(const GLfloat* a)
03510     {
03511         return load(a[0], a[1], a[2], a[3]);
03512     }
03513
03520     GgQuaternion& load(const GgVector& v)
03521     {
03522         static_cast<GgVector>(*this) = v;
03523         return *this;
03524     }
03525
03532     GgQuaternion& load(const GgQuaternion& q)
03533     {
03534         *this = q;
03535         return *this;
03536     }
03537
03547     GgQuaternion& loadAdd(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03548     {
03549         data()[0] += x;
03550         data()[1] += y;
03551         data()[2] += z;
03552         data()[3] += w;
03553         return *this;
03554     }
03555
03562     GgQuaternion& loadAdd(const GLfloat* a)
03563     {
03564         return loadAdd(a[0], a[1], a[2], a[3]);
03565     }
03566
03573     GgQuaternion& loadAdd(const GgVector& v)
03574     {
03575         return loadAdd(v[0], v[1], v[2], v[3]);
03576     }
03577
03584     GgQuaternion& loadAdd(const GgQuaternion& q)
03585     {
03586         return loadAdd(q[0], q[1], q[2], q[3]);
03587     }
03588
03598     GgQuaternion& loadSubtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03599     {
03600         data()[0] -= x;
03601         data()[1] -= y;
03602         data()[2] -= z;
03603         data()[3] -= w;
03604         return *this;
03605     }
03606
03613     GgQuaternion& loadSubtract(const GLfloat* a)
03614     {
03615         return loadSubtract(a[0], a[1], a[2], a[3]);
03616     }
03617
03624     GgQuaternion& loadSubtract(const GgVector& v)
03625     {
03626         return loadSubtract(v[0], v[1], v[2], v[3]);
03627     }
03628
03635     GgQuaternion& loadSubtract(const GgQuaternion& q)
03636     {
03637         return loadSubtract(q[0], q[1], q[2], q[3]);
03638     }
03639
03649     GgQuaternion& loadMultiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03650     {
03651         const GLfloat a[] { x, y, z, w };
03652         return loadMultiply(a);
03653     }
03654
03661     GgQuaternion& loadMultiply(const GLfloat* a)
03662     {
03663         return load(multiply(a));
03664     }
03665
03672     GgQuaternion& loadMultiply(const GgVector& v)
```

```

03673 {
03674     return loadMultiply(v.data());
03675 }
03676
03677 GgQuaternion& loadMultiply(const GgQuaternion& q)
03678 {
03679     return loadMultiply(q.data());
03680 }
03681
03682 GgQuaternion& loadDivide(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03683 {
03684     const GLfloat a[] { x, y, z, w };
03685     return loadDivide(a);
03686 }
03687
03688 GgQuaternion& loadDivide(GLfloat* a)
03689 {
03690     return loadDivide(a);
03691 }
03692
03693 GgQuaternion& loadDivide(const GgVector& v)
03694 {
03695     return loadDivide(v.data());
03696 }
03697
03698 GgQuaternion& loadDivide(const GgQuaternion& q)
03699 {
03700     return loadDivide(q.data());
03701 }
03702
03703 GgQuaternion add(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03704 {
03705     GgQuaternion s { *this };
03706     return s.loadAdd(x, y, z, w);
03707 }
03708
03709 GgQuaternion add(const GLfloat* a) const
03710 {
03711     return add(a[0], a[1], a[2], a[3]);
03712 }
03713
03714 GgQuaternion add(const GgVector& v) const
03715 {
03716     return add(v[0], v[1], v[2], v[3]);
03717 }
03718
03719 GgQuaternion add(const GgQuaternion& q) const
03720 {
03721     return add(q[0], q[1], q[2], q[3]);
03722 }
03723
03724 GgQuaternion subtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03725 {
03726     GgQuaternion s { *this };
03727     return s.loadSubtract(x, y, z, w);
03728 }
03729
03730 GgQuaternion subtract(const GLfloat* a) const
03731 {
03732     return subtract(a[0], a[1], a[2], a[3]);
03733 }
03734
03735 GgQuaternion subtract(const GgVector& v) const
03736 {
03737     return subtract(v[0], v[1], v[2], v[3]);
03738 }
03739
03740 GgQuaternion subtract(const GgQuaternion& q) const
03741 {
03742     return subtract(q[0], q[1], q[2], q[3]);
03743 }
03744
03745 GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03746 {
03747     const GLfloat a[] { x, y, z, w };
03748     return multiply(a);
03749 }
03750
03751 GgQuaternion multiply(const GLfloat* a) const
03752 {
03753     GgQuaternion s;
03754     multiply(s.data(), data(), a);
03755     return s;
03756 }
03757
03758 GgQuaternion multiply(const GgVector& v) const
03759 {

```

```
03868     return multiply(v.data());
03869 }
03870
03871     GgQuaternion multiply(const GgQuaternion& q) const
03872 {
03873     return multiply(q.data());
03874 }
03875
03876     GgQuaternion divide(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03877 {
03878     const GLfloat a[] { x, y, z, w };
03879     return divide(a);
03880 }
03881
03882     GgQuaternion divide(const GLfloat* a) const
03883 {
03884     GgQuaternion s, ia;
03885     ia.loadInvert(a);
03886     multiply(s.data(), data(), ia.data());
03887     return s;
03888 }
03889
03890     GgQuaternion divide(const GgVector& v) const
03891 {
03892     return divide(v.data());
03893 }
03894
03895     GgQuaternion divide(const GgQuaternion& q) const
03896 {
03897     return divide(q.data());
03898 }
03899
03900 // 演算子
03901     GgQuaternion& operator=(const GLfloat* a)
03902 {
03903     return load(a);
03904 }
03905     GgQuaternion& operator=(const GgVector& v)
03906 {
03907     return load(v);
03908 }
03909     GgQuaternion& operator+=(const GLfloat* a)
03910 {
03911     return loadAdd(a);
03912 }
03913     GgQuaternion& operator+=(const GgVector& v)
03914 {
03915     return loadAdd(v);
03916 }
03917     GgQuaternion& operator+=(const GgQuaternion& q)
03918 {
03919     return loadAdd(q);
03920 }
03921
03922     GgQuaternion& operator-=(const GLfloat* a)
03923 {
03924     return loadSubtract(a);
03925 }
03926     GgQuaternion& operator-=(const GgVector& v)
03927 {
03928     return loadSubtract(v);
03929 }
03930     GgQuaternion& operator-=(const GgQuaternion& q)
03931 {
03932     return operator-=(q.data());
03933 }
03934
03935     GgQuaternion& operator*=(const GLfloat* a)
03936 {
03937     return loadMultiply(a);
03938 }
03939     GgQuaternion& operator*=(const GgVector& v)
03940 {
03941     return loadMultiply(v);
03942 }
03943     GgQuaternion& operator*=(const GgQuaternion& q)
03944 {
03945     return operator*=(q.data());
03946 }
03947
03948     GgQuaternion& operator/=(const GLfloat* a)
03949 {
03950     return loadDivide(a);
03951 }
03952     GgQuaternion& operator/=(const GgVector& v)
03953 {
03954     return loadDivide(v);
03955 }
03956     GgQuaternion& operator/=(const GgQuaternion& q)
03957 {
03958     return operator/=(q.data());
03959 }
03960
03961     GgQuaternion& operator/=(const GgQuaternion& q)
03962 {
03963     return operator/=(q.data());
03964 }
03965     GgQuaternion& operator*=(const GgQuaternion& q)
03966 {
03967     return operator*=(q.data());
03968 }
03969     GgQuaternion& operator*=(const GgVector& v)
03970 {
03971     return loadMultiply(v);
03972 }
03973     GgQuaternion& operator*=(const GgQuaternion& q)
03974 {
03975     return operator*=(q.data());
03976 }
03977     GgQuaternion& operator/=(const GgQuaternion& q)
03978 {
03979     return loadDivide(q);
03980 }
03981     GgQuaternion& operator/=(const GgVector& v)
03982 {
03983     return loadDivide(v);
03984 }
03985     GgQuaternion& operator/=(const GgQuaternion& q)
03986 {
03987 }
```

```
03988     return operator/=(q.data());
03989 }
03990 GgQuaternion operator+(const GLfloat* a) const
03991 {
03992     return add(a);
03993 }
03994 GgQuaternion operator+(const GgVector& v) const
03995 {
03996     return add(v);
03997 }
03998 GgQuaternion operator+(const GgQuaternion& q) const
03999 {
04000     return operator+(q.data());
04001 }
04002 GgQuaternion operator-(const GLfloat* a) const
04003 {
04004     return add(a);
04005 }
04006 GgQuaternion operator-(const GgVector& v) const
04007 {
04008     return add(v);
04009 }
04010 GgQuaternion operator-(const GgQuaternion& q) const
04011 {
04012     return operator-(q.data());
04013 }
04014 GgQuaternion operator*(const GLfloat* a) const
04015 {
04016     return multiply(a);
04017 }
04018 GgQuaternion operator*(const GgVector& v) const
04019 {
04020     return multiply(v);
04021 }
04022 GgQuaternion operator*(const GgQuaternion& q) const
04023 {
04024     return operator*(q.data());
04025 }
04026 GgQuaternion operator/(const GLfloat* a) const
04027 {
04028     return divide(a);
04029 }
04030 GgQuaternion operator/(const GgVector& v) const
04031 {
04032     return divide(v);
04033 }
04034 GgQuaternion operator/(const GgQuaternion& q) const
04035 {
04036     return operator/(q.data());
04037 }
04038
04039 GgQuaternion& loadMatrix(const GLfloat* a)
04040 {
04041     toQuaternion(data(), a);
04042     return *this;
04043 }
04044
04045 GgQuaternion& loadMatrix(const GgMatrix& m)
04046 {
04047     return loadMatrix(m.get());
04048 }
04049
04050
04051 GgQuaternion& loadIdentity()
04052 {
04053     return load(0.0f, 0.0f, 0.0f, 1.0f);
04054 }
04055
04056 GgQuaternion& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);
04057
04058 GgQuaternion& loadRotate(const GLfloat* v, GLfloat a)
04059 {
04060     return loadRotate(v[0], v[1], v[2], a);
04061 }
04062
04063 GgQuaternion& loadRotate(const GLfloat* v)
04064 {
04065     return loadRotate(v[0], v[1], v[2], v[3]);
04066 }
04067
04068 GgQuaternion& loadRotateX(GLfloat a);
04069
04070 GgQuaternion& loadRotateY(GLfloat a);
04071
04072 GgQuaternion& loadRotateZ(GLfloat a);
04073
04074 GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
04075 {
04076 }
```

```
04141     GgQuaternion q;
04142     return multiply(q.loadRotate(x, y, z, a));
04143 }
04144
04152     GgQuaternion rotate(const GLfloat* v, GLfloat a) const
04153 {
04154     return rotate(v[0], v[1], v[2], a);
04155 }
04156
04163     GgQuaternion rotate(const GLfloat* v) const
04164 {
04165     return rotate(v[0], v[1], v[2], v[3]);
04166 }
04167
04174     GgQuaternion rotateX(GLfloat a) const
04175 {
04176     return rotate(1.0f, 0.0f, 0.0f, a);
04177 }
04178
04185     GgQuaternion rotateY(GLfloat a) const
04186 {
04187     return rotate(0.0f, 1.0f, 0.0f, a);
04188 }
04189
04196     GgQuaternion rotateZ(GLfloat a) const
04197 {
04198     return rotate(0.0f, 0.0f, 1.0f, a);
04199 }
04200
04209     GgQuaternion& loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll);
04210
04217     GgQuaternion& loadEuler(const GLfloat* e)
04218 {
04219     return loadEuler(e[0], e[1], e[2]);
04220 }
04221
04230     GgQuaternion euler(GLfloat heading, GLfloat pitch, GLfloat roll) const
04231 {
04232     GgQuaternion r;
04233     return multiply(r.loadEuler(heading, pitch, roll));
04234 }
04235
04242     GgQuaternion euler(const GLfloat* e) const
04243 {
04244     return euler(e[0], e[1], e[2]);
04245 }
04246
04255     GgQuaternion& loadSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04256 {
04257     slerp(data(), a, b, t);
04258     return *this;
04259 }
04260
04269     GgQuaternion& loadSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04270 {
04271     return loadSlerp(q.data(), r.data(), t);
04272 }
04273
04282     GgQuaternion& loadSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04283 {
04284     return loadSlerp(q.data(), a, t);
04285 }
04286
04295     GgQuaternion& loadSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04296 {
04297     return loadSlerp(a, q.data(), t);
04298 }
04299
04306     GgQuaternion& loadNormalize(const GLfloat* a);
04307
04314     GgQuaternion& loadNormalize(const GgQuaternion& q)
04315 {
04316     return loadNormalize(q.data());
04317 }
04318
04325     GgQuaternion& loadConjugate(const GLfloat* a);
04326
04333     GgQuaternion& loadConjugate(const GgQuaternion& q)
04334 {
04335     return loadConjugate(q.data());
04336 }
04337
04344     GgQuaternion& loadInvert(const GLfloat* a);
04345
04352     GgQuaternion& loadInvert(const GgQuaternion& q)
04353 {
04354     return loadInvert(q.data());
```



```
04515
04522 inline GgQuaternion ggQuaternion(const GLfloat* a)
04523 {
04524     return ggQuaternion(a[0], a[1], a[2], a[3]);
04525 }
04526
04532 inline GgQuaternion ggIdentityQuaternion()
04533 {
04534     GgQuaternion q;
04535     return q.loadIdentity();
04536 }
04537
04544 inline GgQuaternion ggMatrixQuaternion(const GLfloat* a)
04545 {
04546     GgQuaternion q;
04547     return q.loadMatrix(a);
04548 }
04549
04556 inline GgQuaternion ggMatrixQuaternion(const GgMatrix& m)
04557 {
04558     return ggMatrixQuaternion(m.get());
04559 }
04560
04567 inline GgMatrix ggQuaternionMatrix(const GgQuaternion& q)
04568 {
04569     GLfloat m[16];
04570     q.getMatrix(m);
04571     return GgMatrix{ m };
04572 }
04573
04580 inline GgMatrix ggQuaternionTransposeMatrix(const GgQuaternion& q)
04581 {
04582     GLfloat m[16];
04583     q.getMatrix(m);
04584     GgMatrix t;
04585     return t.loadTranspose(m);
04586 }
04587
04597 inline GgQuaternion ggRotateQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
04598 {
04599     GgQuaternion q;
04600     return q.loadRotate(x, y, z, a);
04601 }
04602
04610 inline GgQuaternion ggRotateQuaternion(const GLfloat* v, GLfloat a)
04611 {
04612     return ggRotateQuaternion(v[0], v[1], v[2], a);
04613 }
04614
04621 inline GgQuaternion ggRotateQuaternion(const GLfloat* v)
04622 {
04623     return ggRotateQuaternion(v[0], v[1], v[2], v[3]);
04624 }
04625
04634 inline GgQuaternion ggEulerQuaternion(GLfloat heading, GLfloat pitch, GLfloat roll)
04635 {
04636     GgQuaternion q;
04637     return q.loadEuler(heading, pitch, roll);
04638 }
04639
04646 inline GgQuaternion ggEulerQuaternion(const GLfloat* e)
04647 {
04648     return ggEulerQuaternion(e[0], e[1], e[2]);
04649 }
04650
04659 inline GgQuaternion ggSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
04660 {
04661     GgQuaternion r;
04662     return r.loadSlerp(a, b, t);
04663 }
04664
04673 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
04674 {
04675     return ggSlerp(q.data(), r.data(), t);
04676 }
04677
04686 inline GgQuaternion ggSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
04687 {
04688     return ggSlerp(q.data(), a, t);
04689 }
04690
04699 inline GgQuaternion ggSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
04700 {
04701     return ggSlerp(a, q.data(), t);
04702 }
04703
04710 inline GLfloat ggNorm(const GgQuaternion& q)
```

```

04711  {
04712    return q.norm();
04713  }
04714
04721  inline GgQuaternion ggNormalize(const GgQuaternion& q)
04722  {
04723    return q.normalize();
04724  }
04725
04732  inline GgQuaternion ggConjugate(const GgQuaternion& q)
04733  {
04734    return q.conjugate();
04735  }
04736
04743  inline GgQuaternion ggInvert(const GgQuaternion& q)
04744  {
04745    return q.invert();
04746  }
04747
04751  class GgTrackball : public GgQuaternion
04752  {
04753    bool drag;          // ドラッグ中か否か
04754    GLfloat start[2]; // ドラッグ開始位置
04755    GLfloat scale[2]; // マウスの絶対位置→ウィンドウ内での相対位置の換算係数
04756    GgQuaternion cq; // 回軸の初期値 (四元数)
04757    GgMatrix rt;      // 回軸の変換行列
04758
04759  public:
04760
04766  GgTrackball(const GgQuaternion& q = ggIdentityQuaternion())
04767  {
04768    reset(q);
04769  }
04770
04774  virtual ~GgTrackball()
04775  {
04776  }
04777
04783  GgTrackball& operator=(const GgQuaternion& q)
04784  {
04785    this->load(q);
04786    return *this;
04787  }
04788
04798  void region(GLfloat w, GLfloat h);
04799
04809  void region(int w, int h)
04810  {
04811    region(static_cast<GLfloat>(w), static_cast<GLfloat>(h));
04812  }
04813
04823  void begin(GLfloat x, GLfloat y);
04824
04834  void motion(GLfloat x, GLfloat y);
04835
04841  void rotate(const GgQuaternion& q);
04842
04852  void end(GLfloat x, GLfloat y);
04853
04859  void reset(const GgQuaternion& q = ggIdentityQuaternion());
04860
04866  const GLfloat* getStart() const
04867  {
04868    return start;
04869  }
04870
04874  const GLfloat& getStart(int direction) const
04875  {
04876    return start[direction];
04877  }
04878
04884  void getStart(GLfloat* position) const
04885  {
04886    position[0] = start[0];
04887    position[1] = start[1];
04888  }
04889
04895  const GLfloat* getScale() const
04896  {
04897    return scale;
04898  }
04899
04905  const GLfloat getScale(int direction) const
04906  {
04907    return scale[direction];
04908  }
04909

```

```
04915     void getScale(GLfloat* factor) const
04916     {
04917         factor[0] = scale[0];
04918         factor[1] = scale[1];
04919     }
04920
04926     const GgQuaternion& getQuaternion() const
04927     {
04928         return *this;
04929     }
04930
04936     const GgMatrix& getMatrix() const
04937     {
04938         return rt;
04939     }
04940
04946     const GLfloat* get() const
04947     {
04948         return rt.get();
04949     }
04950 };
04951
04962     extern bool ggSaveTga(
04963         const std::string& name,
04964         const void* buffer,
04965         unsigned int width,
04966         unsigned int height,
04967         unsigned int depth
04968 );
04969
04976     extern bool ggSaveColor(const std::string& name);
04977
04984     extern bool ggSaveDepth(const std::string& name);
04985
04996     extern bool ggReadImage(
04997         const std::string& name,
04998         std::vector<GLubyte>& image,
04999         GLsizei* pWidth,
05000         GLsizei* pHeight,
05001         GLenum* pFormat
05002 );
05003
05017     extern GLuint ggLoadTexture(
05018         const GLvoid* image,
05019         GLsizei width,
05020         GLsizei height,
05021         GLenum format = GL_RGB,
05022         GLenum type = GL_UNSIGNED_BYTE,
05023         GLenum internal = GL_RGB,
05024         GLenum wrap = GL_CLAMP_TO_EDGE,
05025         bool swizzle = true
05026 );
05027
05038     extern GLuint ggLoadImage(
05039         const std::string& name,
05040         GLsizei* pWidth = nullptr,
05041         GLsizei* pHeight = nullptr,
05042         GLenum internal = 0,
05043         GLenum wrap = GL_CLAMP_TO_EDGE
05044 );
05045
05057     extern void ggCreateNormalMap(
05058         const GLubyte* hmap,
05059         GLsizei width,
05060         GLsizei height,
05061         GLenum format,
05062         GLfloat nz,
05063         GLenum internal,
05064         std::vector<GgVector>& nmap
05065 );
05066
05077     extern GLuint ggLoadHeight(
05078         const std::string& name,
05079         GLfloat nz,
05080         GLsizei* pWidth = nullptr,
05081         GLsizei* pHeight = nullptr,
05082         GLenum internal = GL_RGBA
05083 );
05084
05098     extern GLuint ggCreateShader(
05099         const std::string& vsrc,
05100         const std::string& fsrc = "",
05101         const std::string& gsrc = "",
05102         GLint nvarying = 0,
05103         const char* const* varyings = nullptr,
05104         const std::string& vtext = "vertex shader",
05105         const std::string& ftext = "fragment shader",
```

```

05106     const std::string& gtext = "geometry shader");
05107
05118 extern GLuint ggLoadShader(
05119     const std::string& vert,
05120     const std::string& frag = "",
05121     const std::string& geom = "",
05122     GLint nvarying = 0,
05123     const char* const* varyings = nullptr
05124 );
05125
05134 inline GLuint ggLoadShader(
05135     const std::array<std::string, 3>& files,
05136     GLint nvarying = 0,
05137     const char* const* varyings = nullptr
05138 )
05139 {
05140     return ggLoadShader(files[0], files[1], files[2], nvarying, varyings);
05141 }
05142
05143 #if !defined(__APPLE__)
05151 extern GLuint ggCreateComputeShader(
05152     const std::string& csrc,
05153     const std::string& ctext = "compute shader"
05154 );
05155
05162 extern GLuint ggLoadComputeShader(const std::string& comp);
05163 #endif
05164
05171 class GgTexture
05172 {
05173     // テクスチャ名
05174     GLuint texture;
05175
05176     // テクスチャの縦横の画素数
05177     GLsizei size[2];
05178
05179 public:
05180
05193     GgTexture(
05194         const GLvoid* image,
05195         GLsizei width,
05196         GLsizei height,
05197         GLenum format = GL_RGB,
05198         GLenum type = GL_UNSIGNED_BYTE,
05199         GLenum internal = GL_RGBA,
05200         GLenum wrap = GL_CLAMP_TO_EDGE,
05201         bool swizzle = true
05202     ) :
05203         texture{ ggLoadTexture(image, width, height, format, type, internal, wrap, swizzle) },
05204         size{ width, height }
05205     {
05206     }
05207
05211     virtual ~GgTexture()
05212     {
05213         glBindTexture(GL_TEXTURE_2D, 0);
05214         glDeleteTextures(1, &texture);
05215     }
05216
05220     GgTexture(const GgTexture& o) = delete;
05221
05225     GgTexture& operator=(const GgTexture& o) = delete;
05226
05230     void bind() const
05231     {
05232         glBindTexture(GL_TEXTURE_2D, texture);
05233     }
05234
05238     void unbind() const
05239     {
05240         glBindTexture(GL_TEXTURE_2D, 0);
05241     }
05242
05248     void swapRandB(bool swizzle) const;
05249
05255     const GLsizei& getWidth() const
05256     {
05257         return size[0];
05258     }
05259
05265     const GLsizei& getHeight() const
05266     {
05267         return size[1];
05268     }
05269
05275     void getSize(GLsizei* size) const
05276     {

```

```
05277     size[0] = getWidth();
05278     size[1] = getHeight();
05279 }
05280
05286 const GLsizei* getSize() const
05287 {
05288     return size;
05289 }
05290
05296 const GLuint& getTexture() const
05297 {
05298     return texture;
05299 }
05300
05301 };
05308 class GgColorTexture
05309 {
05310     // テクスチャ
05311     std::shared_ptr<GgTexture> texture;
05312
05313 public:
05314     GgColorTexture()
05315     {
05320     }
05321
05334     GgColorTexture(
05335         const GLvoid* image,
05336         GLsizei width,
05337         GLsizei height,
05338         GLenum format = GL_RGB,
05339         GLenum type = GL_UNSIGNED_BYTE,
05340         GLenum internal = GL_RGB,
05341         GLenum wrap = GL_CLAMP_TO_EDGE,
05342         bool swizzle = true
05343     )
05344     {
05345         load(image, width, height, format, type, internal, wrap, swizzle);
05346     }
05347
05355     GgColorTexture(
05356         const std::string& name,
05357         GLenum internal = 0,
05358         GLenum wrap = GL_CLAMP_TO_EDGE
05359     )
05360     {
05361         load(name, internal, wrap);
05362     }
05363
05367     virtual ~GgColorTexture()
05368     {
05369     }
05370
05383     void load(
05384         const GLvoid* image,
05385         GLsizei width,
05386         GLsizei height,
05387         GLenum format = GL_RGB,
05388         GLenum type = GL_UNSIGNED_BYTE,
05389         GLenum internal = GL_RGB,
05390         GLenum wrap = GL_CLAMP_TO_EDGE,
05391         bool swizzle = true
05392     )
05393     {
05394         // テクスチャを作成する
05395         texture = std::make_shared<GgTexture>(image, width, height, format, type, internal, wrap,
05396         swizzle);
05397     }
05405     void load(
05406         const std::string& name,
05407         GLenum internal = 0,
05408         GLenum wrap = GL_CLAMP_TO_EDGE
05409     );
05410 };
05411
05418 class GgNormalTexture
05419 {
05420     // テクスチャ
05421     std::shared_ptr<GgTexture> texture;
05422
05423 public:
05424     GgNormalTexture()
05425     {
05430     }
05431
```

```

05442 GgNormalTexture(
05443     const GLubyte* image,
05444     GLsizei width,
05445     GLsizei height,
05446     GLenum format = GL_RED,
05447     GLfloat nz = 1.0f,
05448     GLenum internal = GL_RGBA
05449 )
05450 {
05451     // 法線マップのテクスチャを作成する
05452     load(image, width, height, format, nz, internal);
05453 }
05454
05462 GgNormalTexture(
05463     const std::string& name,
05464     GLfloat nz = 1.0f,
05465     GLenum internal = GL_RGBA
05466 )
05467 {
05468     // 法線マップのテクスチャを作成する
05469     load(name, nz, internal);
05470 }
05471
05475 virtual ~GgNormalTexture()
05476 {
05477 }
05478
05489 void load(
05490     const GLubyte* hmap,
05491     GLsizei width,
05492     GLsizei height,
05493     GLenum format = GL_RED,
05494     GLfloat nz = 1.0f,
05495     GLenum internal = GL_RGBA
05496 )
05497 {
05498     // 法線マップ
05499     std::vector<GgVector> nmap;
05500
05501     // 法線マップを作成する
05502     ggCreateNormalMap(hmap, width, height, format, nz, internal, nmap);
05503
05504     // テクスチャを作成する
05505     texture = std::make_shared<GgTexture>(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal,
05506     GL_REPEAT);
05507 }
05515 void load(
05516     const std::string& name,
05517     GLfloat nz = 1.0f,
05518     GLenum internal = GL_RGBA
05519 );
05520 };
05521
05528 template <typename T>
05529 class GgBuffer
05530 {
05531     // ターゲット
05532     const GLenum target;
05533
05534     // 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
05535     const GLsizei stride;
05536
05537     // データの数
05538     const GLsizei count;
05539
05540     // バッファオブジェクト
05541     const GLuint buffer;
05542
05543 public:
05544
05554     GgBuffer<T>(
05555         GLenum target,
05556         const T* data,
05557         GLsizei stride,
05558         GLsizei count,
05559         GLenum usage
05560     ) :
05561     target{ target },
05562     stride{ stride },
05563     count{ count },
05564     buffer{ [] { GLuint buffer; glGenBuffers(1, &buffer); return buffer; } () }
05565 {
05566     // バッファオブジェクトのメモリを確保してデータを転送する
05567     glBindBuffer(target, buffer);
05568     glBufferData(target, getStride()* count, data, usage);
05569 }

```

```

05570
05574     virtual ~GgBuffer<T>()
05575     {
05576         // バッファオブジェクトを削除する
05577         glBindBuffer(target, 0);
05578         glDeleteBuffers(1, &buffer);
05579     }
05580
05584     GgBuffer<T> (const GgBuffer<T>& o) = delete;
05585
05589     GgBuffer<T>& operator=(const GgBuffer<T>& o) = delete;
05590
05596     const GLuint& getTarget() const
05597     {
05598         return target;
05599     }
05600
05606     GLsizeiptr getStride() const
05607     {
05608         return static_cast<GLsizeiptr>(stride);
05609     }
05610
05616     const GLsizei& getCount() const
05617     {
05618         return count;
05619     }
05620
05626     const GLuint& getBuffer() const
05627     {
05628         return buffer;
05629     }
05630
05634     void bind() const
05635     {
05636         glBindBuffer(target, buffer);
05637     }
05638
05642     void unbind() const
05643     {
05644         glBindBuffer(target, 0);
05645     }
05646
05652     void* map() const
05653     {
05654         glBindBuffer(target, buffer);
05655 #if defined(GL_GLES_PROTOTYPES)
05656         return glMapBufferRange(target, 0, getStride() * count, GL_MAP_WRITE_BIT);
05657 #else
05658         return glMapBuffer(target, GL_WRITE_ONLY);
05659 #endif
05660     }
05661
05669     void* map(GLint first, GLsizei count) const
05670     {
05671         // count が 0 なら全データをマップする
05672         if (count == 0) count = getCount();
05673         if (first + count > getCount()) count = getCount() - first;
05674
05675         glBindBuffer(target, buffer);
05676         return glMapBufferRange(target, getStride() * first, getStride() * count, GL_MAP_WRITE_BIT);
05677     }
05678
05682     void unmap() const
05683     {
05684         glUnmapBuffer(target);
05685     }
05686
05694     void send(const T* data, GLint first, GLsizei count) const
05695     {
05696         // count が 0 なら全データを転送する
05697         if (count == 0) count = getCount();
05698         if (first + count > getCount()) count = getCount() - first;
05699
05700         // データを既存のバッファオブジェクトに転送する
05701         glBindBuffer(target, buffer);
05702         glBufferSubData(target, getStride() * first, getStride() * count, data);
05703     }
05704
05712     void read(T* data, GLint first, GLsizei count) const
05713     {
05714         // count が 0 なら全データを抽出する
05715         if (count == 0) count = getCount();
05716         if (first + count > getCount()) count = getCount() - first;
05717
05718         // データをバッファオブジェクトから抽出する
05719         glBindBuffer(target, buffer);
05720 #if defined(GL_GLES_PROTOTYPES)

```

```

05721     const GLsizeiptr begin{ getStride() * first };
05722     const GLsizeiptr range{ getStride() * count };
05723     T* const source{ glMapBufferRange(target, begin, range, GL_MAP_READ_BIT) };
05724     std::copy(source, source + count, data);
05725     glUnmapBuffer(target);
05726 #else
05727     glGetBufferSubData(target, getStride() * first, getStride() * count, data);
05728 #endif
05729 }
05730
05739 void copy(GLuint src_buffer, GLint src_first = 0, GLint dst_first = 0, GLsizei count = 0) const
05740 {
05741     // count が 0 なら全データを複写する
05742     if (count == 0) count = getCount();
05743     if (src_first + count > getCount()) count = getCount() - src_first;
05744     if (dst_first + count > getCount()) count = getCount() - dst.first;
05745
05746     // データの間隔
05747     const GLsizeiptr stride{ getStride() };
05748
05749     glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
05750     glBindBuffer(GL_COPY_WRITE_BUFFER, buffer);
05751     glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
05752         stride * src_first, stride * dst.first, stride * count);
05753     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
05754     glBindBuffer(GL_COPY_READ_BUFFER, 0);
05755 }
05756 };
05757
05764 template <typename T>
05765 class GgUniformBuffer
05766 {
05767     // ユニフォームバッファオブジェクト
05768     std::shared_ptr<GgBuffer<T>> uniform;
05769
05770 public:
05771
05775     GgUniformBuffer<T>()
05776     {
05777     }
05778
05786     GgUniformBuffer<T> (const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05787     {
05788         load(data, count, usage);
05789     }
05790
05798     GgUniformBuffer<T> (const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05799     {
05800         load(data, count, usage);
05801     }
05802
05806     virtual ~GgUniformBuffer<T>()
05807     {
05808     }
05809
05815     const GLuint& getTarget() const
05816     {
05817         return uniform->getTarget();
05818     }
05819
05825     GLsizeiptr getStride() const
05826     {
05827         return uniform->getStride();
05828     }
05829
05835     const GLsizei& getCount() const
05836     {
05837         return uniform->getCount();
05838     }
05839
05845     const GLuint& getBuffer() const
05846     {
05847         return uniform->getBuffer();
05848     }
05849
05853     void bind() const
05854     {
05855         uniform->bind();
05856     }
05857
05861     void unbind() const
05862     {
05863         uniform->unbind();
05864     }
05865
05871     void* map() const
05872     {

```

```
05873     return uniform->map();
05874 }
05875
05883 void* map(GLint first, GLsizei count) const
05884 {
05885     return uniform->map(first, count);
05886 }
05887
05891 void unmap() const
05892 {
05893     uniform->unmap();
05894 }
05895
05903 void load(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05904 {
05905     // バッファオブジェクト上のデータの間隔
05906     const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) *
05907     ggBufferAlignment };
05908
05909     // ユニフォームバッファオブジェクトを確保する
05910     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05911
05912     // 確保したユニフォームバッファオブジェクトにデータを転送する
05913     if (data) send(data, 0, sizeof(T), 0, count);
05914 }
05915
05922 void load(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
05923 {
05924     // バッファオブジェクト上のデータの間隔
05925     const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) *
05926     ggBufferAlignment };
05927
05928     // ユニフォームバッファオブジェクトを確保する
05929     uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);
05930
05931     // 確保したユニフォームバッファオブジェクトにデータを転送する
05932     fill(&data, 0, sizeof(T), 0, count);
05933 }
05943 void send(
05944     const GLvoid* data,
05945     GLint offset = 0,
05946     GLsizei size = sizeof(T),
05947     GLint first = 0,
05948     GLsizei count = 0
05949 ) const
05950 {
05951     // count が 0 なら全データを転送する
05952     if (count == 0) count = getCount();
05953     if (first + count > getCount()) count = getCount() - first;
05954
05955     // 転送元のデータの先頭
05956     const char* source{ reinterpret_cast<const char*>(data) };
05957
05958     // ターゲット
05959     const GLuint target{ getTarget() };
05960
05961     // データの間隔
05962     const GLsizeiptr stride{ getStride() };
05963
05964     // first 番目のブロックから count 個の各ブロックの先頭から offset バイトの位置にデータを転送する
05965     bind();
05966     for (GLsizei i = 0; i < count; ++i)
05967     {
05968         glBindBufferSubData(target, stride * (first + i) + offset, size, source + size * i);
05969     }
05970 }
05971
05981 void fill(
05982     const GLvoid* data,
05983     GLint offset = 0,
05984     GLsizei size = sizeof(T),
05985     GLint first = 0,
05986     GLsizei count = 0
05987 ) const
05988 {
05989     // count が 0 なら全データを転送する
05990     if (count == 0) count = getCount();
05991     if (first + count > getCount()) count = getCount() - first;
05992
05993     // ターゲット
05994     const GLuint target{ getTarget() };
05995
05996     // データの間隔
05997     const GLsizeiptr stride{ getStride() };
05998
05999     // first 番目のブロックから count 個の各ブロックの先頭から offset バイトの位置にデータを転送する
```

```

06000     bind();
06001     for (GLsizei i = 0; i < count; ++i)
06002     {
06003         glBufferSubData(target, stride * (first + i) + offset, size, data);
06004     }
06005
06006
06016 void read(
06017     GLvoid* data,
06018     GLint offset = 0,
06019     GLsizei size = sizeof(T),
06020     GLint first = 0,
06021     GLsizei count = 0
06022 ) const
06023 {
06024     // count が 0 なら全データを転送する
06025     if (count == 0) count = getCount();
06026     if (first + count > getCount()) count = getCount() - first;
06027
06028     // 抽出先のデータの先頭
06029     char* const destination{ reinterpret_cast<char*>(data) };
06030
06031     // ターゲット
06032     const GLuint target{ getTarget() };
06033
06034     // データの間隔
06035     const GLsizeiptr stride{ getStride() };
06036
06037     // データをユニフォームバッファオブジェクトから抽出する
06038     bind();
06039 #if defined(GL_GLES_PROTOTYPES)
06040     const char* const source{ glMapBufferRange(target, stride * first, stride * count,
06041         GL_MAP_READ_BIT) };
06042     for (GLsizei i = 0; i < count; ++i)
06043     {
06044         const char* const begin{ source + stride * i + offset };
06045         const char* const end{ begin + size };
06046         std::copy(begin, end, destination + sizeof(T) * i);
06047     }
06048     glUnmapBuffer(target);
06049 #else
06050     for (GLsizei i = 0; i < count; ++i)
06051     {
06052         glGetBufferSubData(target, stride * (first + i) + offset, size, destination + sizeof(T) * i);
06053     }
06054 #endif
06055 }
06064 void copy(
06065     GLuint srcBuffer,
06066     GLint srcFirst = 0,
06067     GLint dstFirst = 0,
06068     GLsizei count = 0
06069 ) const
06070 {
06071     // count が 0 なら全データを複写する
06072     if (count == 0) count = getCount();
06073     if (srcFirst + count > getCount()) count = getCount() - srcFirst;
06074     if (dstFirst + count > getCount()) count = getCount() - dstFirst;
06075
06076     // データの間隔
06077     const GLsizeiptr stride{ getStride() };
06078
06079     // ユニフォームバッファオブジェクトではブロックごとに転送する
06080     glBindBuffer(GL_COPY_READ_BUFFER, srcBuffer);
06081     glBindBuffer(GL_COPY_WRITE_BUFFER, getBuffer());
06082     for (GLsizei i = 0; i < count; ++i)
06083     {
06084         glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
06085             stride * (srcFirst + i), stride * (dstFirst + i), sizeof(T));
06086     }
06087     glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
06088     glBindBuffer(GL_COPY_READ_BUFFER, 0);
06089 }
06090 };
06091
06099 class GgShape
06100 {
06101     // 頂点配列オブジェクト
06102     const GLuint vao;
06103
06104     // 基本图形の種類
06105     GLenum mode;
06106
06107     public:
06108     GgShape(GLenum mode = 0) :
06114

```

```
06115     vao{ [] { GLuint vao; glGenVertexArrays(1, &vao); return vao; } () },
06116     mode{ mode }
06117 {
06118     glBindVertexArray(vao);
06119 }
06120
06124     virtual ~GgShape()
06125 {
06126     glBindVertexArray(0);
06127     glDeleteVertexArrays(1, &vao);
06128 }
06129
06133     GgShape(const GgShape& o) = delete;
06134
06138     GgShape& operator=(const GgShape& o) = delete;
06139
06144     const GLuint& get() const
06145 {
06146     return vao;
06147 }
06148
06154     void setMode(GLenum mode)
06155 {
06156     this->mode = mode;
06157 }
06158
06164     const GLenum& getMode() const
06165 {
06166     return this->mode;
06167 }
06168
06175     virtual void draw(GLint first = 0, GLsizei count = 0) const
06176 {
06177     glBindVertexArray(vao);
06178 }
06179 };
06180
06184     class GgPoints
06185     : public GgShape
06186 {
06187     // 頂点バッファオブジェクト
06188     std::shared_ptr<GgBuffer<GgVector>> position;
06189
06190 public:
06191
06195     GgPoints(GLenum mode = GL_POINTS) :
06196         GgShape(mode)
06197     {
06198     }
06199
06208     GgPoints(
06209         const GgVector* pos,
06210         GLsizei countv,
06211         GLenum mode = GL_POINTS,
06212         GLenum usage = GL_STATIC_DRAW
06213     ) :
06214     GgPoints(mode)
06215     {
06216         load(pos, countv, usage);
06217     }
06218
06222     virtual ~GgPoints()
06223     {
06224     }
06225
06231     const GLsizei& getCount() const
06232     {
06233         return position->getCount();
06234     }
06235
06241     const GLuint& getBuffer() const
06242     {
06243         return position->getBuffer();
06244     }
06245
06253     void send(const GgVector* pos, GLint first = 0, GLsizei count = 0) const
06254     {
06255         position->send(pos, first, count);
06256     }
06257
06265     void load(const GgVector* pos, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06266
06273     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06274 }
06275
06279     struct GgVertex
06280 {
```

```

06282     GgVector position;
06283
06285     GgVector normal;
06286
06287     GgVertex()
06288     {
06289     }
06290
06291     GgVertex(const GgVector& pos, const GgVector& norm) :
06292         position(pos),
06293         normal(norm)
06294     {
06295     }
06296
06297     GgVertex(
06298         GLfloat px, GLfloat py, GLfloat pz,
06299         GLfloat nx, GLfloat ny, GLfloat nz
06300     ) :
06301         position{ px, py, pz, 1.0f },
06302         normal{ nx, ny, nz, 0.0f }
06303     {
06304     }
06305
06306     GgVertex(const GLfloat* pos, const GLfloat* norm) :
06307         GgVertex(pos[0], pos[1], pos[2], norm[0], norm[1], norm[2])
06308     {
06309     }
06310
06311     GgTriangles()
06312     : public GgShape
06313     {
06314         // 頂点属性
06315         std::unique_ptr<GgBuffer<GgVertex>> vertex;
06316
06317     public:
06318
06319         GgTriangles(GLenum mode = GL_TRIANGLES) :
06320             GgShape(mode)
06321         {
06322         }
06323
06324         GgTriangles(
06325             const GgVertex* vert,
06326             GLsizei count,
06327             GLenum mode = GL_TRIANGLES,
06328             GLenum usage = GL_STATIC_DRAW
06329         ) :
06330             GgTriangles(mode)
06331         {
06332             load(vert, count, usage);
06333         }
06334
06335         virtual ~GgTriangles()
06336         {
06337         }
06338
06339         const GLsizei& getCount() const
06340         {
06341             return vertex->getCount();
06342         }
06343
06344         const GLuint& getBuffer() const
06345         {
06346             return vertex->getBuffer();
06347         }
06348
06349         void send(const GgVertex* vert, GLint first = 0, GLsizei count = 0) const
06350         {
06351             vertex->send(vert, first, count);
06352         }
06353
06354         void load(const GgVertex* vert, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06355
06356         virtual void draw(GLint first = 0, GLsizei count = 0) const;
06357     };
06358
06359     class GgElements
06360     : public GgTriangles
06361     {
06362         // インデックスを格納する頂点バッファオブジェクト
06363         std::unique_ptr<GgBuffer<GLuint>> index;
06364
06365     public:
06366
06367         GgElements(GLenum mode = GL_TRIANGLES) :
06368             GgTriangles(mode)
06369         {
06370             index = std::make_unique<GgBuffer<GLuint>>();
06371             vertex = std::make_unique<GgBuffer<GgVertex>>();
06372             index->reserve(GL_TRIANGLES * 3);
06373             vertex->reserve(GL_TRIANGLES * 3);
06374             index->clear();
06375             vertex->clear();
06376         }
06377
06378         virtual ~GgElements()
06379         {
06380             index->~GgBuffer();
06381             vertex->~GgBuffer();
06382         }
06383
06384         const GLuint& getIndex() const
06385         {
06386             return index->getBuffer();
06387         }
06388
06389         const GgBuffer<GgVertex>& getVertex() const
06390         {
06391             return vertex->getBuffer();
06392         }
06393
06394         void draw(GLint first = 0, GLsizei count = 0) const
06395         {
06396             vertex->send(index->getBuffer(), first, count);
06397         }
06398
06399         void send(GLint first = 0, GLsizei count = 0) const
06400         {
06401             index->send(vertex->getBuffer(), first, count);
06402         }
06403
06404         void load(GLint first = 0, GLsizei count = 0) const
06405         {
06406             index->load(vertex->getBuffer(), first, count);
06407         }
06408
06409         virtual void draw(GLint first = 0, GLsizei count = 0) const;
06410     };
06411
06412     class GgElementsList
06413     : public GgElements
06414     {
06415         std::vector<GgElements> elements;
06416
06417         void draw(GLint first = 0, GLsizei count = 0) const
06418         {
06419             for (const GgElements& element : elements)
06420                 element.draw(first, count);
06421         }
06422
06423         void send(GLint first = 0, GLsizei count = 0) const
06424         {
06425             for (const GgElements& element : elements)
06426                 element.send(first, count);
06427         }
06428
06429         void load(GLint first = 0, GLsizei count = 0) const
06430         {
06431             for (const GgElements& element : elements)
06432                 element.load(first, count);
06433         }
06434
06435         void draw(GLint first = 0, GLsizei count = 0) const
06436         {
06437             for (const GgElements& element : elements)
06438                 element.draw(first, count);
06439         }
06440
06441         void send(GLint first = 0, GLsizei count = 0) const
06442         {
06443             for (const GgElements& element : elements)
06444                 element.send(first, count);
06445         }
06446
06447         void load(GLint first = 0, GLsizei count = 0) const
06448         {
06449             for (const GgElements& element : elements)
06450                 element.load(first, count);
06451         }
06452     };
06453
06454     class GgElementsListList
06455     : public GgElementsList
06456     {
06457         std::vector<GgElementsList> elements;
06458
06459         void draw(GLint first = 0, GLsizei count = 0) const
06460         {
06461             for (const GgElementsList& element : elements)
06462                 element.draw(first, count);
06463         }
06464
06465         void send(GLint first = 0, GLsizei count = 0) const
06466         {
06467             for (const GgElementsList& element : elements)
06468                 element.send(first, count);
06469         }
06470
06471         void load(GLint first = 0, GLsizei count = 0) const
06472         {
06473             for (const GgElementsList& element : elements)
06474                 element.load(first, count);
06475         }
06476
06477         void draw(GLint first = 0, GLsizei count = 0) const
06478         {
06479             for (const GgElementsList& element : elements)
06480                 element.draw(first, count);
06481         }
06482
06483         void send(GLint first = 0, GLsizei count = 0) const
06484         {
06485             for (const GgElementsList& element : elements)
06486                 element.send(first, count);
06487         }
06488
06489         void load(GLint first = 0, GLsizei count = 0) const
06490         {
06491             for (const GgElementsList& element : elements)
06492                 element.load(first, count);
06493         }
06494     };
06495
06496     class GgElementsListListList
06497     : public GgElementsListList
06498     {
06499         std::vector<GgElementsListList> elements;
06500
06501         void draw(GLint first = 0, GLsizei count = 0) const
06502         {
06503             for (const GgElementsListList& element : elements)
06504                 element.draw(first, count);
06505         }
06506
06507         void send(GLint first = 0, GLsizei count = 0) const
06508         {
06509             for (const GgElementsListList& element : elements)
06510                 element.send(first, count);
06511         }
06512
06513         void load(GLint first = 0, GLsizei count = 0) const
06514         {
06515             for (const GgElementsListList& element : elements)
06516                 element.load(first, count);
06517         }
06518
06519         void draw(GLint first = 0, GLsizei count = 0) const
06520         {
06521             for (const GgElementsListList& element : elements)
06522                 element.draw(first, count);
06523         }
06524
06525         void send(GLint first = 0, GLsizei count = 0) const
06526         {
06527             for (const GgElementsListList& element : elements)
06528                 element.send(first, count);
06529         }
06530
06531         void load(GLint first = 0, GLsizei count = 0) const
06532         {
06533             for (const GgElementsListList& element : elements)
06534                 element.load(first, count);
06535         }
06536     };
06537
06538     class GgElementsListListListList
06539     : public GgElementsListListList
06540     {
06541         std::vector<GgElementsListListList> elements;
06542
06543         void draw(GLint first = 0, GLsizei count = 0) const
06544         {
06545             for (const GgElementsListListList& element : elements)
06546                 element.draw(first, count);
06547         }
06548
06549         void send(GLint first = 0, GLsizei count = 0) const
06550         {
06551             for (const GgElementsListListList& element : elements)
06552                 element.send(first, count);
06553         }
06554
06555         void load(GLint first = 0, GLsizei count = 0) const
06556         {
06557             for (const GgElementsListListList& element : elements)
06558                 element.load(first, count);
06559         }
06560
06561         void draw(GLint first = 0, GLsizei count = 0) const
06562         {
06563             for (const GgElementsListListList& element : elements)
06564                 element.draw(first, count);
06565         }
06566
06567         void send(GLint first = 0, GLsizei count = 0) const
06568         {
06569             for (const GgElementsListListList& element : elements)
06570                 element.send(first, count);
06571         }
06572
06573         void load(GLint first = 0, GLsizei count = 0) const
06574         {
06575             for (const GgElementsListListList& element : elements)
06576                 element.load(first, count);
06577         }
06578     };
06579
06580     class GgElementsListListListListList
06581     : public GgElementsListListListList
06582     {
06583         std::vector<GgElementsListListListList> elements;
06584
06585         void draw(GLint first = 0, GLsizei count = 0) const
06586         {
06587             for (const GgElementsListListListList& element : elements)
06588                 element.draw(first, count);
06589         }
06590
06591         void send(GLint first = 0, GLsizei count = 0) const
06592         {
06593             for (const GgElementsListListListList& element : elements)
06594                 element.send(first, count);
06595         }
06596
06597         void load(GLint first = 0, GLsizei count = 0) const
06598         {
06599             for (const GgElementsListListListList& element : elements)
06600                 element.load(first, count);
06601         }
06602
06603         void draw(GLint first = 0, GLsizei count = 0) const
06604         {
06605             for (const GgElementsListListListList& element : elements)
06606                 element.draw(first, count);
06607         }
06608
06609         void send(GLint first = 0, GLsizei count = 0) const
06610         {
06611             for (const GgElementsListListListList& element : elements)
06612                 element.send(first, count);
06613         }
06614
06615         void load(GLint first = 0, GLsizei count = 0) const
06616         {
06617             for (const GgElementsListListListList& element : elements)
06618                 element.load(first, count);
06619         }
06620     };
06621
06622     class GgElementsListListListListListList
06623     : public GgElementsListListListListList
06624     {
06625         std::vector<GgElementsListListListListList> elements;
06626
06627         void draw(GLint first = 0, GLsizei count = 0) const
06628         {
06629             for (const GgElementsListListListListList& element : elements)
06630                 element.draw(first, count);
06631         }
06632
06633         void send(GLint first = 0, GLsizei count = 0) const
06634         {
06635             for (const GgElementsListListListListList& element : elements)
06636                 element.send(first, count);
06637         }
06638
06639         void load(GLint first = 0, GLsizei count = 0) const
06640         {
06641             for (const GgElementsListListListListList& element : elements)
06642                 element.load(first, count);
06643         }
06644
06645         void draw(GLint first = 0, GLsizei count = 0) const
06646         {
06647             for (const GgElementsListListListListList& element : elements)
06648                 element.draw(first, count);
06649         }
06650
06651         void send(GLint first = 0, GLsizei count = 0) const
06652         {
06653             for (const GgElementsListListListListList& element : elements)
06654                 element.send(first, count);
06655         }
06656
06657         void load(GLint first = 0, GLsizei count = 0) const
06658         {
06659             for (const GgElementsListListListListList& element : elements)
06660                 element.load(first, count);
06661         }
06662     };
06663
06664     class GgElementsListListListListListListList
06665     : public GgElementsListListListListListList
06666     {
06667         std::vector<GgElementsListListListListListList> elements;
06668
06669         void draw(GLint first = 0, GLsizei count = 0) const
06670         {
06671             for (const GgElementsListListListListListList& element : elements)
06672                 element.draw(first, count);
06673         }
06674
06675         void send(GLint first = 0, GLsizei count = 0) const
06676         {
06677             for (const GgElementsListListListListListList& element : elements)
06678                 element.send(first, count);
06679         }
06680
06681         void load(GLint first = 0, GLsizei count = 0) const
06682         {
06683             for (const GgElementsListListListListListList& element : elements)
06684                 element.load(first, count);
06685         }
06686
06687         void draw(GLint first = 0, GLsizei count = 0) const
06688         {
06689             for (const GgElementsListListListListListList& element : elements)
06690                 element.draw(first, count);
06691         }
06692
06693         void send(GLint first = 0, GLsizei count = 0) const
06694         {
06695             for (const GgElementsListListListListListList& element : elements)
06696                 element.send(first, count);
06697         }
06698
06699         void load(GLint first = 0, GLsizei count = 0) const
06700         {
06701             for (const GgElementsListListListListListList& element : elements)
06702                 element.load(first, count);
06703         }
06704     };
06705
06706     class GgElementsListListListListListListListList
06707     : public GgElementsListListListListListListList
06708     {
06709         std::vector<GgElementsListListListListListListList> elements;
06710
06711         void draw(GLint first = 0, GLsizei count = 0) const
06712         {
06713             for (const GgElementsListListListListListListList& element : elements)
06714                 element.draw(first, count);
06715         }
06716
06717         void send(GLint first = 0, GLsizei count = 0) const
06718         {
06719             for (const GgElementsListListListListListListList& element : elements)
06720                 element.send(first, count);
06721         }
06722
06723         void load(GLint first = 0, GLsizei count = 0) const
06724         {
06725             for (const GgElementsListListListListListListList& element : elements)
06726                 element.load(first, count);
06727         }
06728
06729         void draw(GLint first = 0, GLsizei count = 0) const
06730         {
06731             for (const GgElementsListListListListListListList& element : elements)
06732                 element.draw(first, count);
06733         }
06734
06735         void send(GLint first = 0, GLsizei count = 0) const
06736         {
06737             for (const GgElementsListListListListListListList& element : elements)
06738                 element.send(first, count);
06739         }
06740
06741         void load(GLint first = 0, GLsizei count = 0) const
06742         {
06743             for (const GgElementsListListListListListListList& element : elements)
06744                 element.load(first, count);
06745         }
06746     };
06747
06748     class GgElementsListListListListListListListListList
06749     : public GgElementsListListListListListListListList
06750     {
06751         std::vector<GgElementsListListListListListListListList> elements;
06752
06753         void draw(GLint first = 0, GLsizei count = 0) const
06754         {
06755             for (const GgElementsListListListListListListListList& element : elements)
06756                 element.draw(first, count);
06757         }
06758
06759         void send(GLint first = 0, GLsizei count = 0) const
06760         {
06761             for (const GgElementsListListListListListListListList& element : elements)
06762                 element.send(first, count);
06763         }
06764
06765         void load(GLint first = 0, GLsizei count = 0) const
06766         {
06767             for (const GgElementsListListListListListListListList& element : elements)
06768                 element.load(first, count);
06769         }
06770
06771         void draw(GLint first = 0, GLsizei count = 0) const
06772         {
06773             for (const GgElementsListListListListListListListList& element : elements)
06774                 element.draw(first, count);
06775         }
06776
06777         void send(GLint first = 0, GLsizei count = 0) const
06778         {
06779             for (const GgElementsListListListListListListListList& element : elements)
06780                 element.send(first, count);
06781         }
06782
06783         void load(GLint first = 0, GLsizei count = 0) const
06784         {
06785             for (const GgElementsListListListListListListListList& element : elements)
06786                 element.load(first, count);
06787         }
06788     };
06789
06790     class GgElementsListListListListListListListListListList
06791     : public GgElementsListListListListListListListListList
06792     {
06793         std::vector<GgElementsListListListListListListListListList> elements;
06794
06795         void draw(GLint first = 0, GLsizei count = 0) const
06796         {
06797             for (const GgElementsListListListListListListListListList& element : elements)
06798                 element.draw(first, count);
06799         }
06800
06801         void send(GLint first = 0, GLsizei count = 0) const
06802         {
06803             for (const GgElementsListListListListListListListListList& element : elements)
06804                 element.send(first, count);
06805         }
06806
06807         void load(GLint first = 0, GLsizei count = 0) const
06808         {
06809             for (const GgElementsListListListListListListListListList& element : elements)
06810                 element.load(first, count);
06811         }
06812
06813         void draw(GLint first = 0, GLsizei count = 0) const
06814         {
06815             for (const GgElementsListListListListListListListListList& element : elements)
06816                 element.draw(first, count);
06817         }
06818
06819         void send(GLint first = 0, GLsizei count = 0) const
06820         {
06821             for (const GgElementsListListListListListListListListList& element : elements)
06822                 element.send(first, count);
06823         }
06824
06825         void load(GLint first = 0, GLsizei count = 0) const
06826         {
06827             for (const GgElementsListListListListListListListListList& element : elements)
06828                 element.load(first, count);
06829         }
06830     };
06831
06832     class GgElementsListListListListListListListListListListList
06833     : public GgElementsListListListListListListListListListList
06834     {
06835         std::vector<GgElementsListListListListListListListListListList> elements;
06836
06837         void draw(GLint first = 0, GLsizei count = 0) const
06838         {
06839             for (const GgElementsListListListListListListListListListList& element : elements)
06840                 element.draw(first, count);
06841         }
06842
06843         void send(GLint first = 0, GLsizei count = 0) const
06844         {
06845             for (const GgElementsListListListListListListListListListList& element : elements)
06846                 element.send(first, count);
06847         }
06848
06849         void load(GLint first = 0, GLsizei count = 0) const
06850         {
06851             for (const GgElementsListListListListListListListListListList& element : elements)
06852                 element.load(first, count);
06853         }
06854
06855         void draw(GLint first = 0, GLsizei count = 0) const
06856         {
06857             for (const GgElementsListListListListListListListListListList& element : elements)
06858                 element.draw(first, count);
06859         }
06860
06861         void send(GLint first = 0, GLsizei count = 0) const
06862         {
06863             for (const GgElementsListListListListListListListListListList& element : elements)
06864                 element.send(first, count);
06865         }
06866
06867         void load(GLint first = 0, GLsizei count = 0) const
06868         {
06869             for (const GgElementsListListListListListListListListListList& element : elements)
06870                 element.load(first, count);
06871         }
06872     };
06873
06874     class GgElementsListListListListListListListListListListListList
06875     : public GgElementsListListListListListListListListListListList
06876     {
06877         std::vector<GgElementsListListListListListListListListListListList> elements;
06878
06879         void draw(GLint first = 0, GLsizei count = 0) const
06880         {
06881             for (const GgElementsListListListListListListListListListListList& element : elements)
06882                 element.draw(first, count);
06883         }
06884
06885         void send(GLint first = 0, GLsizei count = 0) const
06886         {
06887             for (const GgElementsListListListListListListListListListListList& element : elements)
06888                 element.send(first, count);
06889         }
06890
06891         void load(GLint first = 0, GLsizei count = 0) const
06892         {
06893             for (const GgElementsListListListListListListListListListListList& element : elements)
06894                 element.load(first, count);
06895         }
06896
06897         void draw(GLint first = 0, GLsizei count = 0) const
06898         {
06899             for (const GgElementsListListListListListListListListListListList& element : elements)
07000                 element.draw(first, count);
07001         }
07002
07003         void send(GLint first = 0, GLsizei count = 0) const
07004         {
07005             for (const GgElementsListListListListListListListListListListList& element : elements)
07006                 element.send(first, count);
07007         }
07008
07009         void load(GLint first = 0, GLsizei count = 0) const
07010         {
07011             for (const GgElementsListListListListListListListListListListList& element : elements)
07012                 element.load(first, count);
07013         }
07014     };
07015
07016     class GgElementsListListListListListListListListListListListListList
07017     : public GgElementsListListListListListListListListListListListList
07018     {
07019         std::vector<GgElementsListListListListListListListListListListListList> elements;
07020
07021         void draw(GLint first = 0, GLsizei count = 0) const
07022         {
07023             for (const GgElementsListListListListListListListListListListListList& element : elements)
07024                 element.draw(first, count);
07025         }
07026
07027         void send(GLint first = 0, GLsizei count = 0) const
07028         {
07029             for (const GgElementsListListListListListListListListListListListList& element : elements)
07030                 element.send(first, count);
07031         }
07032
07033         void load(GLint first = 0, GLsizei count = 0) const
07034         {
07035             for (const GgElementsListListListListListListListListListListListList& element : elements)
07036                 element.load(first, count);
07037         }
07038
07039         void draw(GLint first = 0, GLsizei count = 0) const
07040         {
07041             for (const GgElementsListListListListListListListListListListListList& element : elements)
07042                 element.draw(first, count);
07043         }
07044
07045         void send(GLint first = 0, GLsizei count = 0) const
07046         {
07047             for (const GgElementsListListListListListListListListListListListList& element : elements)
07048                 element.send(first, count);
07049         }
07050
07051         void load(GLint first = 0, GLsizei count = 0) const
07052         {
07053             for (const GgElementsListListListListListListListListListListListList& element : elements)
07054                 element.load(first, count);
07055         }
07056     };
07057
07058     class GgElementsListListListListListListListListListListListListListList
07059     : public GgElementsListListListListListListListListListListListListList
07060     {
07061         std::vector<GgElementsListListListListListListListListListListListListList> elements;
07062
07063         void draw(GLint first = 0, GLsizei count = 0) const
07064         {
07065             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07066                 element.draw(first, count);
07067         }
07068
07069         void send(GLint first = 0, GLsizei count = 0) const
07070         {
07071             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07072                 element.send(first, count);
07073         }
07074
07075         void load(GLint first = 0, GLsizei count = 0) const
07076         {
07077             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07078                 element.load(first, count);
07079         }
07080
07081         void draw(GLint first = 0, GLsizei count = 0) const
07082         {
07083             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07084                 element.draw(first, count);
07085         }
07086
07087         void send(GLint first = 0, GLsizei count = 0) const
07088         {
07089             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07090                 element.send(first, count);
07091         }
07092
07093         void load(GLint first = 0, GLsizei count = 0) const
07094         {
07095             for (const GgElementsListListListListListListListListListListListListList& element : elements)
07096                 element.load(first, count);
07097         }
07098     };
07099
07100     class GgElementsListListListListListListListListListListListListListListList
07101     : public GgElementsListListListListListListListListListListListListListList
07102     {
07103         std::vector<GgElementsListListListListListListListListListListListListListList> elements;
07104
07105         void draw(GLint first = 0, GLsizei count = 0) const
07106         {
07107             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07108                 element.draw(first, count);
07109         }
07110
07111         void send(GLint first = 0, GLsizei count = 0) const
07112         {
07113             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07114                 element.send(first, count);
07115         }
07116
07117         void load(GLint first = 0, GLsizei count = 0) const
07118         {
07119             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07120                 element.load(first, count);
07121         }
07122
07123         void draw(GLint first = 0, GLsizei count = 0) const
07124         {
07125             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07126                 element.draw(first, count);
07127         }
07128
07129         void send(GLint first = 0, GLsizei count = 0) const
07130         {
07131             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07132                 element.send(first, count);
07133         }
07134
07135         void load(GLint first = 0, GLsizei count = 0) const
07136         {
07137             for (const GgElementsListListListListListListListListListListListListListList& element : elements)
07138                 element.load(first, count);
07139         }
07140     };
07141
07142     class GgElementsListListListListListListListListListListListListListListListList
07143     : public GgElementsListListListListListListListListListListListListListListList
07144     {
07145         std::vector<GgElementsListListListListListListListListListListListListListListList> elements;
07146
07147         void draw(GLint first = 0, GLsizei count = 0) const
07148         {
07149             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07150                 element.draw(first, count);
07151         }
07152
07153         void send(GLint first = 0, GLsizei count = 0) const
07154         {
07155             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07156                 element.send(first, count);
07157         }
07158
07159         void load(GLint first = 0, GLsizei count = 0) const
07160         {
07161             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07162                 element.load(first, count);
07163         }
07164
07165         void draw(GLint first = 0, GLsizei count = 0) const
07166         {
07167             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07168                 element.draw(first, count);
07169         }
07170
07171         void send(GLint first = 0, GLsizei count = 0) const
07172         {
07173             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07174                 element.send(first, count);
07175         }
07176
07177         void load(GLint first = 0, GLsizei count = 0) const
07178         {
07179             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07180                 element.load(first, count);
07181         }
07182     };
07183
07184     class GgElementsListListListListListListListListListListListListListListListListList
07185     : public GgElementsListListListListListListListListListListListListListListListList
07186     {
07187         std::vector<GgElementsListListListListListListListListListListListListListListListList> elements;
07188
07189         void draw(GLint first = 0, GLsizei count = 0) const
07190         {
07191             for (const GgElementsListListListListListListListListListListListListListListListList& element : elements)
07192                 element.draw(first, count);
07193         }
07194
07195         void send(GLint first = 0, GLsizei count = 0) const
07196         {
07197             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07198                 element.send(first, count);
07199         }
07200
07201         void load(GLint first = 0, GLsizei count = 0) const
07202         {
07203             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07204                 element.load(first, count);
07205         }
07206
07207         void draw(GLint first = 0, GLsizei count = 0) const
07208         {
07209             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07210                 element.draw(first, count);
07211         }
07212
07213         void send(GLint first = 0, GLsizei count = 0) const
07214         {
07215             for (const GgElementsListListListListListListListListListListListListListListList& element : elements)
07216                 element.send(first, count);
072
```

```
06452  {
06453  }
06454
06455  GgElements(
06456      const GgVertex* vert,
06457      GLsizei countv,
06458      const GLuint* face,
06459      GLsizei countf,
06460      GLenum mode = GL_TRIANGLES,
06461      GLenum usage = GL_STATIC_DRAW
06462  ) :
06463      GgElements(mode)
06464  {
06465      load(vert, countv, face, countf, usage);
06466  }
06467
06468  virtual ~GgElements()
06469  {
06470  }
06471
06472  const GLsizei& getIndexCount() const
06473  {
06474      return index->getCount();
06475  }
06476
06477  const GLuint& getIndexBuffer() const
06478  {
06479      return index->getBuffer();
06480  }
06481
06482  void send(
06483      const GgVertex* vert,
06484      GLuint firstv,
06485      GLsizei countv,
06486      const GLuint* face = nullptr,
06487      GLuint firstf = 0,
06488      GLsizei countf = 0
06489  ) const
06490  {
06491      GgTriangles::send(vert, firstv, countv);
06492      if (face != nullptr && countf > 0) index->send(face, firstf, countf);
06493  }
06494
06495  void load(
06496      const GgVertex* vert,
06497      GLsizei countv,
06498      const GLuint* face,
06499      GLsizei countf,
06500      GLenum usage = GL_STATIC_DRAW
06501  )
06502  {
06503      // 頂点バッファオブジェクトを作成する
06504      GgTriangles::load(vert, countv, usage);
06505
06506      // インデックスの頂点バッファオブジェクトを作成する
06507      index = std::make_unique<GgBuffer<GLuint>>(GL_ELEMENT_ARRAY_BUFFER, face,
06508          static_cast<GLsizei>(sizeof(GLuint)), countf, usage);
06509  }
06510
06511  virtual void draw(GLint first = 0, GLsizei count = 0) const;
06512  };
06513
06514  extern std::unique_ptr<GgPoints> ggPointsCube(
06515      GLsizei countv,
06516      GLfloat length = 1.0f,
06517      GLfloat cx = 0.0f,
06518      GLfloat cy = 0.0f,
06519      GLfloat cz = 0.0f
06520  );
06521
06522  extern std::unique_ptr<GgPoints> ggPointsSphere(
06523      GLsizei countv,
06524      GLfloat radius = 0.5f,
06525      GLfloat cx = 0.0f,
06526      GLfloat cy = 0.0f,
06527      GLfloat cz = 0.0f
06528  );
06529
06530  extern std::unique_ptr<GgTriangles> ggRectangle(
06531      GLfloat width = 1.0f,
06532      GLfloat height = 1.0f
06533  );
06534
06535  extern std::unique_ptr<GgTriangles> ggEllipse(
06536      GLfloat width = 1.0f,
06537      GLfloat height = 1.0f,
06538      GLuint slices = 16
```

```

06620 );
06621
06633 extern std::unique_ptr<GgTriangles> ggArraysObj(
06634     const std::string& name,
06635     bool normalize = false
06636 );
06637
06649 extern std::unique_ptr<GgElements> ggElementsObj(
06650     const std::string& name,
06651     bool normalize = false
06652 );
06653
06666 extern std::unique_ptr<GgElements> ggElementsMesh(
06667     GLuint slices,
06668     GLuint stacks,
06669     const GLfloat(*pos)[3],
06670     const GLfloat(*norm)[3] = nullptr
06671 );
06672
06683 extern std::unique_ptr<GgElements> ggElementsSphere(
06684     GLfloat radius = 1.0f,
06685     int slices = 16,
06686     int stacks = 8
06687 );
06688
06695 class GgShader
06696 {
06697     // プログラム名
06698     const GLuint program;
06699
06700 public:
06701
06711     GgShader(
06712         const std::string& vert,
06713         const std::string& frag = "",
06714         const std::string& geom = "",
06715         int nvarying = 0,
06716         const char* const* varyings = nullptr
06717     ) :
06718         program(ggLoadShader(vert, frag, geom, nvarying, varyings))
06719     {
06720     }
06721
06729     GgShader(
06730         const std::array<std::string, 3>& files,
06731         int nvarying = 0,
06732         const char* const* varyings = nullptr
06733     ) :
06734         GgShader(files[0], files[1], files[2], nvarying, varyings)
06735     {
06736     }
06737
06741     GgShader(const GgShader& o) = delete;
06742
06746     virtual ~GgShader()
06747     {
06748         // 参照しているオブジェクトが一つだけならシェーダを削除する
06749         glUseProgram(0);
06750         glDeleteProgram(program);
06751     }
06752
06756     GgShader& operator=(const GgShader& o) = delete;
06757
06761     void use() const
06762     {
06763         glUseProgram(program);
06764     }
06765
06769     void unuse() const
06770     {
06771         glUseProgram(0);
06772     }
06773
06779     GLuint get() const
06780     {
06781         return program;
06782     }
06783 };
06784
06788 class GgPointShader
06789 {
06790     // シェーダー
06791     std::shared_ptr<GgShader> shader;
06792
06793     // 投影変換行列の uniform 変数の場所
06794     GLint mpLoc;
06795

```

```
06796 // モデルビュー変換行列の uniform 変数の場所
06797 GLint mvLoc;
06798
06799 public:
06800
06804 GgPointShader() :
06805     mpLoc{ -1 },
06806     mvLoc{ -1 }
06807 {
06808 }
06809
06819 GgPointShader(
06820     const std::string& vert,
06821     const std::string& frag = ",
06822     const std::string& geom = ",
06823     GLint nvarying = 0,
06824     const char* const* varyings = nullptr
06825 ) :
06826     GgPointShader()
06827 {
06828     load(vert, frag, geom, nvarying, varyings);
06829 }
06830
06838 GgPointShader(
06839     const std::array<std::string, 3>& files,
06840     int nvarying = 0,
06841     const char* const* varyings = nullptr
06842 ) :
06843     GgPointShader(files[0], files[1], files[2], nvarying, varyings)
06844 {
06845 }
06846
06850 virtual ~GgPointShader()
06851 {
06852 }
06853
06864 bool load(
06865     const std::string& vert,
06866     const std::string& frag = ",
06867     const std::string& geom = ",
06868     GLint nvarying = 0,
06869     const char* const* varyings = nullptr
06870 )
06871 {
06872     // シェーダを作成する
06873     shader = std::make_shared<GgShader>(vert, frag, geom, nvarying, varyings);
06874
06875     // プログラム名を取り出す
06876     const GLuint program(shader->get());
06877
06878     // プログラムオブジェクトが作成できていなければ戻る
06879     if (program == 0) return false;
06880
06881     // 変換行列の uniform 変数の場所
06882     mpLoc = glGetUniformLocation(program, "mp");
06883     mvLoc = glGetUniformLocation(program, "mv");
06884
06885     // プログラムオブジェクトの作成に成功した
06886     return true;
06887 }
06888
06897 bool load(
06898     const std::array<std::string, 3>& files,
06899     GLint nvarying = 0,
06900     const char* const* varyings = nullptr
06901 )
06902 {
06903     return load(files[0], files[1], files[2], nvarying, varyings);
06904 }
06905
06911 virtual void loadProjectionMatrix(const GLfloat* mp) const
06912 {
06913     glUniformMatrix4fv(mpLoc, 1, GL_FALSE, mp);
06914 }
06915
06921 virtual void loadProjectionMatrix(const GgMatrix& mp) const
06922 {
06923     loadProjectionMatrix(mp.get());
06924 }
06925
06931 virtual void loadModelviewMatrix(const GLfloat* mv) const
06932 {
06933     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, mv);
06934 }
06935
06941 virtual void loadModelviewMatrix(const GgMatrix& mv) const
06942 {
```

```

06943     loadModelviewMatrix(mv.get());
06944 }
06945
06952     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
06953 {
06954     loadProjectionMatrix(mp);
06955     loadModelviewMatrix(mv);
06956 }
06957
06964     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
06965 {
06966     loadMatrix(mp.get(), mv.get());
06967 }
06968
06972     virtual void use() const
06973 {
06974     shader->use();
06975 }
06976
06982     void use(const GLfloat* mp) const
06983 {
06984     use();
06985     loadProjectionMatrix(mp);
06986 }
06987
06993     void use(const GgMatrix& mp) const
06994 {
06995     use(mp.get());
06996 }
06997
07004     void use(const GLfloat* mp, const GLfloat* mv) const
07005 {
07006     use(mp);
07007     loadModelviewMatrix(mv);
07008 }
07009
07016     void use(const GgMatrix& mp, const GgMatrix& mv) const
07017 {
07018     use(mp.get(), mv.get());
07019 }
07020
07024     void unuse() const
07025 {
07026     shader->unuse();
07027 }
07028
07034     GLuint get() const
07035 {
07036     return shader->get();
07037 }
07038
07043     class GgSimpleShader
07044     : public GgPointShader
07045     {
07046         // 材質データの uniform block のインデックス
07047         GLint materialIndex;
07048
07049         // 光源データの uniform block のインデックス
07050         GLint lightIndex;
07051
07052         // モデルビュー変換の法線変換行列の uniform 変数の場所
07053         GLint mnLoc;
07054
07055     public:
07056
07060         GgSimpleShader() :
07061             GgPointShader(),
07062             materialIndex{ -1 },
07063             lightIndex{ -1 },
07064             mnLoc{ -1 }
07065     }
07066
07067
07077         GgSimpleShader(
07078             const std::string& vert,
07079             const std::string& frag = "",
07080             const std::string& geom = "",
07081             GLint nvarying = 0,
07082             const char* const* varyings = nullptr
07083         )
07084     {
07085         load(vert, frag, geom, nvarying, varyings);
07086     }
07087
07095         GgSimpleShader(
07096             const std::array<std::string, 3>& files,

```

```
07097     GLint nvarying = 0,
07098     const char* const* varyings = nullptr
07099 }
07100     GgSimpleShader(files[0], files[1], files[2], nvarying, varyings)
07101 {
07102 }
07103
07104
07105     GgSimpleShader(const GgSimpleShader& o) :
07106     GgPointShader(o),
07107     materialIndex{ o.materialIndex },
07108     lightIndex{ o.lightIndex },
07109     mnLoc{ o.mnLoc }
07110 {
07111 }
07112 }
07113 }
07114
07115     virtual ~GgSimpleShader()
07116 {
07117 }
07118
07119     GgSimpleShader& operator=(const GgSimpleShader& o)
07120 {
07121     if (&o != this)
07122     {
07123         GgPointShader::operator=(o);
07124         materialIndex = o.materialIndex;
07125         lightIndex = o.lightIndex;
07126         mnLoc = o.mnLoc;
07127     }
07128
07129     return *this;
07130 }
07131
07132     bool load(
07133         const std::string& vert,
07134         const std::string& frag = "",
07135         const std::string& geom = "",
07136         GLint nvarying = 0,
07137         const char* const* varyings = nullptr
07138     );
07139
07140     bool load(
07141         const std::array<std::string, 3>& files,
07142         GLint nvarying = 0,
07143         const char* const* varyings = nullptr
07144     );
07145
07146     {
07147         return load(files[0], files[1], files[2], nvarying, varyings);
07148     }
07149
07150     virtual void loadModelviewMatrix(const GLfloat* mv, const GLfloat* mn) const
07151     {
07152         GgPointShader::loadModelviewMatrix(mv);
07153         glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07154     }
07155
07156     virtual void loadModelviewMatrix(const GgMatrix& mv, const GgMatrix& mn) const
07157     {
07158         loadModelviewMatrix(mv.get(), mn.get());
07159     }
07160
07161     virtual void loadModelviewMatrix(const GLfloat* mv) const
07162     {
07163         loadModelviewMatrix(mv, GgMatrix(mv).normal().get());
07164     }
07165
07166     virtual void loadModelviewMatrix(const GgMatrix& mv) const
07167     {
07168         loadModelviewMatrix(mv.get());
07169     }
07170
07171     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07172     {
07173         GgPointShader::loadMatrix(mp, mv);
07174         glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
07175     }
07176
07177     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07178     {
07179         loadMatrix(mp.get(), mv.get(), mn.get());
07180     }
07181
07182     virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
07183     {
07184         loadMatrix(mp, mv, GgMatrix(mv).normal());
07185     }
07186
07187     virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
```



```
07472     void loadPosition(const GLfloat* position, GLint first = 0, GLsizei count = 1) const
07473     {
07474         // first 番目のブロックから count 個の position 要素に値を設定する
07475         send(position, offsetof(Light, position), sizeof(Light::position), first, count);
07476     }
07477
07478     void loadPosition(const GgVector* position, GLint first = 0, GLsizei count = 1) const
07479     {
07480         loadPosition(position->data(), first, count);
07481     }
07482
07483     void load(const Light* light, GLint first = 0, GLsizei count = 1) const
07484     {
07485         send(light, 0, sizeof(Light), first, count);
07486     }
07487
07488     void load(const Light& light, GLint first = 0, GLsizei count = 1) const
07489     {
07490         load(&light, first, count);
07491     }
07492
07493     void select(GLint i = 0) const
07494     {
07495         // バッファオブジェクトの i 番目のブロックの位置
07496         const GLintptr offset(static_cast<GLintptr>(getStride()) * i);
07497         glBindBufferRange(getTarget(), LightBindingPoint, getBuffer(), offset, sizeof(Light));
07498     }
07499
07500     struct Material
07501     {
07502         GgVector ambient;
07503         GgVector diffuse;
07504         GgVector specular;
07505         GLfloat shininess;
07506     };
07507
07508     class MaterialBuffer
07509         : public GgUniformBuffer<Material>
07510     {
07511     public:
07512
07513         MaterialBuffer(
07514             const Material* material = nullptr,
07515             GLsizei count = 1,
07516             GLenum usage = GL_STATIC_DRAW
07517         ) :
07518             GgUniformBuffer<Material>(material, count, usage)
07519         {
07520
07521         MaterialBuffer(
07522             const Material& material,
07523             GLsizei count = 1,
07524             GLenum usage = GL_STATIC_DRAW
07525         ) :
07526             GgUniformBuffer<Material>(material, count, usage)
07527         {
07528
07529         virtual ~MaterialBuffer()
07530         {
07531
07532         void loadAmbient(
07533             GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07534             GLint first = 0, GLsizei count = 1
07535         ) const;
07536
07537         void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07538         {
07539             // first 番目のブロックから count 個のambient 要素に値を設定する
07540             send(ambient, offsetof(Material, ambient), sizeof(Material::ambient), first, count);
07541         }
07542
07543         void loadDiffuse(
07544             GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07545             GLint first = 0, GLsizei count = 1
07546         ) const;
07547
07548         void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07549         {
07550             // first 番目のブロックから count 個の diffuse 要素に値を設定する
07551             send(diffuse, offsetof(Material, diffuse), sizeof(Material::diffuse), first, count);
07552         }
07553
07554         void loadAmbientAndDiffuse()
```

```

07652     GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07653     GLint first = 0, GLsizei count = 1
07654 ) const;
07655
07663     void loadAmbientAndDiffuse(const GLfloat* color, GLint first = 0, GLsizei count = 1) const;
07664
07675     void loadSpecular(
07676         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07677         GLint first = 0, GLsizei count = 1
07678 ) const;
07679
07687     void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07688     {
07689         // first 番目のブロックから count 個の specular 要素に値を設定する
07690         send(specular, offsetof(Material, specular), sizeof(Material::specular), first, count);
07691     }
07692
07700     void loadShininess(GLfloat shininess, GLint first = 0, GLsizei count = 1) const;
07701
07709     void loadShininess(const GLfloat* shininess, GLint first = 0, GLsizei count = 1) const;
07710
07718     void load(const Material* material, GLint first = 0, GLsizei count = 1) const
07719     {
07720         send(material, 0, sizeof(Material), first, count);
07721     }
07722
07730     void load(const Material& material, GLint first = 0, GLsizei count = 1) const
07731     {
07732         load(&material, first, count);
07733     }
07734
07740     void select(GLint i = 0) const
07741     {
07742         // バッファオブジェクトの i 番目のブロックの位置
07743         const GLintptr offset{ static_cast<GLintptr>(getStride()) * i };
07744         glBindBufferRange(getTarget(), MaterialBindingPoint, getBuffer(), offset, sizeof(Material));
07745     }
07746 };
07747
07751     void use() const
07752     {
07753         // プログラムオブジェクトは基底クラスで指定する
07754         GgPointShader::use();
07755     }
07756
07764     void use(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
07765     {
07766         // プログラムオブジェクトを指定する
07767         use();
07768
07769         // 変換行列を設定する
07770         loadMatrix(mp, mv, mn);
07771     }
07772
07780     void use(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
07781     {
07782         use(mp.get(), mv.get(), mn.get());
07783     }
07784
07791     void use(const GLfloat* mp, const GLfloat* mv) const
07792     {
07793         use(mp, mv, GgMatrix(mv).normal().get());
07794     }
07795
07802     void use(const GgMatrix& mp, const GgMatrix& mv) const
07803     {
07804         use(mp, mv, mv.normal());
07805     }
07806
07813     void use(const LightBuffer* light, GLint i = 0) const
07814     {
07815         // プログラムオブジェクトを指定する
07816         use();
07817
07818         // 光源を設定する
07819         light->select(i);
07820     }
07821
07828     void use(const LightBuffer& light, GLint i = 0) const
07829     {
07830         use(&light, i);
07831     }
07832
07842     void use(
07843         const GLfloat* mp,
07844         const GLfloat* mv,
07845         const GLfloat* mn,

```

```
07846     const LightBuffer* light,
07847     GLint i = 0
07848 } const
07849 {
07850     // 光源を指定してプログラムオブジェクトを指定する
07851     use(light, i);
07852
07853     // 変換行列を設定する
07854     loadMatrix(mp, mv, mn);
07855 }
07856
07857 void use(
07858     const GgMatrix& mp,
07859     const GgMatrix& mv,
07860     const GgMatrix& mn,
07861     const LightBuffer& light,
07862     GLint i = 0
07863 ) const
07864 {
07865     use(mp.get(), mv.get(), mn.get(), &light, i);
07866 }
07867
07868 void use(
07869     const GLfloat* mp,
07870     const GLfloat* mv,
07871     const LightBuffer* light,
07872     GLint i = 0
07873 ) const
07874 {
07875     use(mp.get(), mv.get(), GgMatrix(mv).normal().get(), light, i);
07876 }
07877
07878 void use(
07879     const GgMatrix& mp,
07880     const GgMatrix& mv,
07881     const LightBuffer& light,
07882     GLint i = 0
07883 ) const
07884 {
07885     use(mp, mv, GgMatrix(mv).normal().get(), light, i);
07886 }
07887
07888 void use(
07889     const GgMatrix& mp,
07890     const GgMatrix& mv,
07891     const LightBuffer& light,
07892     GLint i = 0
07893 ) const
07894 {
07895     use(mp, mv, mv.normal(), light, i);
07896 }
07897
07898 void use(const GLfloat* mp, const LightBuffer* light, GLint i = 0) const
07899 {
07900     // 光源を指定してプログラムオブジェクトを指定する
07901     use(light, i);
07902
07903     // 投影変換行列を設定する
07904     loadProjectionMatrix(mp);
07905 }
07906
07907 void use(const GgMatrix& mp, const LightBuffer& light, GLint i = 0) const
07908 {
07909     // 光源を指定してプログラムオブジェクトを指定する
07910     use(mp.get(), &light, i);
07911 }
07912
07913 };
07914
07915 extern bool ggLoadSimpleObj(
07916     const std::string& name,
07917     std::vector<std::array<GLuint, 3>>& group,
07918     std::vector<GgSimpleShader::Material>& material,
07919     std::vector<GgVertex>& vert,
07920     bool normalize = false
07921 );
07922
07923 extern bool ggLoadSimpleObj(
07924     const std::string& name,
07925     std::vector<std::array<GLuint, 3>>& group,
07926     std::vector<GgSimpleShader::Material>& material,
07927     std::vector<GgVertex>& vert,
07928     std::vector<GLuint>& face,
07929     bool normalize = false
07930 );
07931
07932 class GgSimpleObj
07933 {
07934     // 同じ材質を割り当てるポリゴングループごとの三角形数
07935     std::shared_ptr<std::vector<std::array<GLuint, 3>>> group;
07936
07937     // ポリゴングループごとの材質のユニフォームバッファ
07938     std::shared_ptr<GgSimpleShader::MaterialBuffer> material;
07939
07940     // この図形の形状データ
07941     std::shared_ptr<GgElements> data;
07942
07943     public:
```

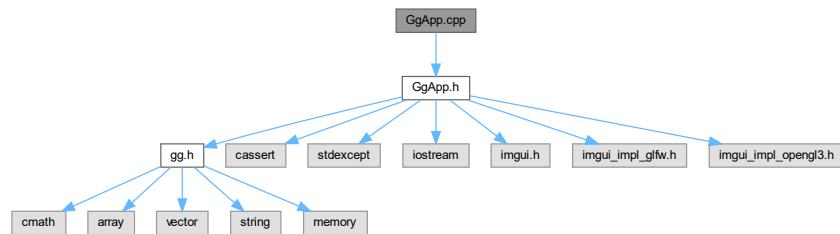
```

07996
08003     GgSimpleObj(const std::string& name, bool normalize = false);
08004
08007     virtual ~GgSimpleObj()
08008     {
08009     }
08010
08016     const GgTriangles* get() const
08017     {
08018         return data.get();
08019     }
08020
08027     virtual void draw(GLint first = 0, GLsizei count = 0) const;
08028 }
08029

```

9.13 GgApp.cpp ファイル

#include "GgApp.h"
GgApp.cpp の依存先関係図:



9.13.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラスの実装

著者

Kohe Tokoi

日付

November 22, 2022

GgApp.cpp に定義があります。

9.14 GgApp.cpp

[詳解]

```

00001
00008 #include "GgApp.h"
00009
00010 //
00011 // GLFW のエラー表示
00012 //
00013 static void glfwErrorCallback(int error, const char* description)
00014 {
00015 #if defined(__aarch64__)
00016     if (error == 65544) return;
00017 #endif
00018     throw std::runtime_error(description);
00019 }
00020
00021 //
00022 // GgApp クラスのコンストラクタ
00023 //
00024 GgApp::GgApp(int major, int minor)
00025 {
00026     // GLFW のエラー処理関数を登録する
00027     glfwSetErrorCallback(glfwErrorCallback);
00028
00029     // GLFW を初期化する
00030     if (glfwInit() == GLFW_FALSE) throw std::runtime_error("Can't initialize GLFW");
00031
00032     // OpenGL の major 番号が指定されていれば
00033     if (major > 0)
00034     {
00035         // OpenGL のバージョンを指定する
00036         glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, major);
00037         glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, minor);
00038
00039 #if defined(GL_GLES_PROTOTYPES)
00040     // OpenGL ES 3 のコンテキストを指定する
00041     glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
00042     glfwWindowHint(GLFW_CONTEXT_CREATION_API, GLFW_EGL_CONTEXT_API);
00043 #else
00044     // OpenGL Version 3.2 以降なら
00045     if (major * 10 + minor >= 32)
00046     {
00047         // Core Profile を選択する (macOS の都合)
00048         glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
00049         glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
00050     }
00051 #endif
00052 }
00053
00054 #if defined(GG_USE_OCU盧US_RIFT)
00055     // Oculus Rift では SRGB でレンダリングする
00056     glfwWindowHint(GLFW_SRGB_CAPABLE, GL_TRUE);
00057 #endif
00058
00059 #if defined(IMGUI_VERSION)
00060     // ImGui のバージョンをチェックする
00061     IMGUI_CHECKVERSION();
00062
00063     // ImGui のコンテキストを作成する
00064     ImGui::CreateContext();
00065 #endif
00066 }
00067
00068 //
00069 // デストラクタ
00070 //
00071 GgApp::~GgApp()
00072 {
00073 #if defined(IMGUI_VERSION)
00074     // Shutdown Platform/Renderer bindings
00075     ImGui_ImplOpenGL3_Shutdown();
00076     ImGui_ImplGlfw_Shutdown();
00077     ImGui::DestroyContext();
00078 #endif
00079
00080     // プログラム終了時に GLFW を終了する
00081     glfwTerminate();
00082 }
00083
00084 //
00085 // マウスや矢印キーによる平行移動量を初期化する
00086 //
00087 void GgApp::Window::HumanInterface::resetTranslation()
00088 {

```

```

00089 // 平行移動量を初期化する
00090 for (auto& t : translation)
00091 {
00092     std::fill(t.begin(), t.end(), GgVector{ 0.0f, 0.0f, 0.0f, 1.0f });
00093 }
00094
00095 // 矢印キーの設定値を初期化する
00096 std::fill(arrow.begin(), arrow.end(), std::array<int, 2>{ 0, 0 });
00097
00098 // マウスホイールの回転量を初期化する
00099 std::fill(wheel.begin(), wheel.end(), 0.0f);
00100 }
00101
00102 //
00103 // 平行移動量と回転量を更新する (X, Y のみ, Z は wheel() で計算する)
00104 //
00105 void GgApp::Window::HumanInterface::calcTranslation(int button, const std::array<GLfloat, 3>&
00106   velocity)
00107 {
00108     // マウスの相対変位
00109     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00110     const auto dx{ (mouse[0] - rotation[button].getStart(0)) * rotation[button].getScale(0) };
00111     const auto dy{ (rotation[button].getStart(1) - mouse[1]) * rotation[button].getScale(1) };
00112
00113     // 平行移動量
00114     auto& t{ translation[button] };
00115
00116     // 平行移動量の更新
00117     t[1][0] = dx * velocity[0] + t[0][0];
00118     t[1][1] = dy * velocity[1] + t[0][1];
00119
00120     // 回転量の更新
00121     rotation[button].motion(mouse[0], mouse[1]);
00122 }
00123
00124 // ウィンドウのサイズ変更時の処理
00125 //
00126 void GgApp::Window::resize(GLFWwindow* window, int width, int height)
00127 {
00128     // このインスタンスの this ポインタを得る
00129     auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00130
00131     if (instance)
00132     {
00133         // ウィンドウのサイズを保存する
00134         instance->size[0] = width;
00135         instance->size[1] = height;
00136
00137         // トラックボール処理の範囲を設定する
00138         for (auto& current_if : instance->interfaceData)
00139         {
00140             for (auto& t : current_if.rotation)
00141             {
00142                 t.region(width, height);
00143             }
00144         }
00145
00146         // ビューポートを更新する
00147         instance->updateViewport();
00148
00149         // ユーザー定義のコールバック関数の呼び出し
00150         if (instance->resizeFunc) (*instance->resizeFunc)(instance, width, height);
00151     }
00152 }
00153
00154 //
00155 // キーボードをタイプした時の処理
00156 //
00157 void GgApp::Window::keyboard(GLFWwindow* window, int key, int scancode, int action, int mods)
00158 {
00159     #if defined(IMGUI_VERSION)
00160         // ImGui がキーボードを使うときはキーボードの処理を行わない
00161         if (ImGui::GetIO().WantCaptureKeyboard) return;
00162     #endif
00163
00164     // このインスタンスの this ポインタを得る
00165     auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00166
00167     if (instance && action)
00168     {
00169         // ユーザー定義のコールバック関数の呼び出し
00170         if (instance->keyboardFunc) (*instance->keyboardFunc)(instance, key, scancode, action, mods);
00171
00172         // 対象のユーザインターフェース
00173         auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00174

```

```
00175     switch (key)
00176     {
00177     case GLFW_KEY_HOME:
00178         // ト ラック ボールを初期化する
00179         instance->resetRotation();
00180         [[fallthrough]];
00181
00182     case GLFW_KEY_END:
00183         // 平行移動量を初期化する
00184         instance->resetTranslation();
00185         break;
00186
00187     case GLFW_KEY_UP:
00188
00189         if (mods & GLFW_MOD_SHIFT)
00190             current_if.arrow[1][1]++;
00191         else if (mods & GLFW_MOD_CONTROL)
00192             current_if.arrow[2][1]++;
00193         else if (mods & GLFW_MOD_ALT)
00194             current_if.arrow[3][1]++;
00195         else
00196             current_if.arrow[0][1]++;
00197         break;
00198
00199     case GLFW_KEY_DOWN:
00200
00201         if (mods & GLFW_MOD_SHIFT)
00202             current_if.arrow[1][1]--;
00203         else if (mods & GLFW_MOD_CONTROL)
00204             current_if.arrow[2][1]--;
00205         else if (mods & GLFW_MOD_ALT)
00206             current_if.arrow[3][1]--;
00207         else
00208             current_if.arrow[0][1]--;
00209         break;
00210
00211     case GLFW_KEY_RIGHT:
00212
00213         if (mods & GLFW_MOD_SHIFT)
00214             current_if.arrow[1][0]++;
00215         else if (mods & GLFW_MOD_CONTROL)
00216             current_if.arrow[2][0]++;
00217         else if (mods & GLFW_MOD_ALT)
00218             current_if.arrow[3][0]++;
00219         else
00220             current_if.arrow[0][0]++;
00221         break;
00222
00223     case GLFW_KEY_LEFT:
00224
00225         if (mods & GLFW_MOD_SHIFT)
00226             current_if.arrow[1][0]--;
00227         else if (mods & GLFW_MOD_CONTROL)
00228             current_if.arrow[2][0]--;
00229         else if (mods & GLFW_MOD_ALT)
00230             current_if.arrow[3][0]--;
00231         else
00232             current_if.arrow[0][0]--;
00233         break;
00234
00235     default:
00236         break;
00237     }
00238
00239     current_if.lastKey = key;
00240
00241 }
00242 }
00243 }
00244
00245 //
00246 // マウスボタンを操作したときの処理
00247 //
00248 void GgApp::Window::mouse(GLFWwindow* window, int button, int action, int mods)
00249 {
00250 #if defined(IMGUI_VERSION)
00251     // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00252     if (ImGui::GetIO().WantCaptureMouse) return;
00253 #endif
00254
00255     // このインスタンスの this ポインタを得る
00256     auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00257
00258     // マウスボタンの状態を記録する
00259     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00260     instance->status[button] = action != GLFW_RELEASE;
00261 }
```

```

00262  if (instance)
00263  {
00264      // ユーザー定義のコールバック関数の呼び出し
00265      if (instance->mouseFunc) (*instance->mouseFunc)(instance, button, action, mods);
00266
00267      // 対象のユーザインターフェース
00268      auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00269
00270      // マウスの現在位置を得る
00271      const auto x{ current_if.mouse[0] };
00272      const auto y{ current_if.mouse[1] };
00273
00274      if (x < 0 || x >= instance->size[0] || y < 0 || y >= instance->size[1]) return;
00275
00276      if (action)
00277      {
00278          // ドラッグ開始
00279          current_if.rotation[button].begin(x, y);
00280      }
00281      else
00282      {
00283          // ドラッグ終了
00284          current_if.translation[button][0] = current_if.translation[button][1];
00285          current_if.rotation[button].end(x, y);
00286      }
00287  }
00288 }
00289
00290 /**
00291 // マウスホイールを操作した時の処理
00292 /**
00293 void GgApp::Window::wheel(GLFWwindow* window, double x, double y)
00294 {
00295 #if defined(IMGUI_VERSION)
00296     // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00297     if (ImGui::GetIO().WantCaptureMouse) return;
00298 #endif
00299
00300 // このインスタンスの this ポインタを得る
00301 auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };
00302
00303 if (instance)
00304 {
00305     // ユーザー定義のコールバック関数の呼び出し
00306     if (instance->wheelFunc) (*instance->wheelFunc)(instance, x, y);
00307
00308     // 対象のユーザインターフェース
00309     auto& current_if{ instance->interfaceData[instance->interfaceNo] };
00310
00311     // マウスホイールの回転量の保存
00312     current_if.wheel[0] += static_cast<GLfloat>(x);
00313     current_if.wheel[1] += static_cast<GLfloat>(y);
00314
00315     // マウスによる平行移動量の z 値の更新
00316     const auto z{ current_if.wheel[1] * instance->velocity[2] };
00317     for (auto& t : current_if.translation) t[1][2] = z;
00318 }
00319 }
00320
00321 /**
00322 // Window クラスのコンストラクタ
00323 /**
00324 GgApp::Window::Window(const std::string& title, int width, int height, int fullscreen, GLFWwindow* share) :
00325     window{ nullptr },
00326     size{ width, height },
00327     fboSize{ width, height },
00328 #if defined(IMGUI_VERSION)
00329     menubarHeight{ 0 },
00330 #endif
00331     aspect{ 1.0f },
00332     velocity{ 1.0f, 1.0f, 0.1f },
00333     status{ false },
00334     interfaceNo{ 0 },
00335     userPointer{ nullptr },
00336     resizeFunc{ nullptr },
00337     keyboardFunc{ nullptr },
00338     mouseFunc{ nullptr },
00339     wheelFunc{ nullptr }
00340 {
00341     // ディスプレイの情報
00342     GLFWmonitor* monitor{ nullptr };
00343
00344     // フルスクリーン表示
00345     if (fullscreen > 0)
00346     {
00347         // 接続されているモニタの数を数える

```

```
00348     int mcount;
00349     auto** const monitors{ glfwGetMonitors(&mcount) };
00350
00351     // セカンダリモニタがあればそれを使う
00352     if (fullscreen > mcount) fullscreen = mcount;
00353     monitor = monitors[fullscreen - 1];
00354
00355     // モニタのモードを調べる
00356     const auto* mode{ glfwGetVideoMode(monitor) };
00357
00358     // ウィンドウのサイズをディスプレイのサイズにする
00359     width = mode->width;
00360     height = mode->height;
00361 }
00362
00363 // GLFW のウィンドウを作成する
00364 window = glfwCreateWindow(width, height, title.c_str(), monitor, share);
00365
00366 // ウィンドウが作成できなければエラー
00367 if (!window) throw std::runtime_error("Unable to open the GLFW window.");
00368
00369 // 現在のウィンドウを処理対象にする
00370 glfwMakeContextCurrent(window);
00371
00372 // ゲームグラフィックス特論の都合による初期化を行う
00373 ggInit();
00374
00375 // このインスタンスの this ポインタを記録しておく
00376 glfwSetWindowUserPointer(window, this);
00377
00378 // キーボードを操作した時の処理を登録する
00379 glfwSetKeyCallback(window, keyboard);
00380
00381 // マウスボタンを操作したときの処理を登録する
00382 glfwSetMouseButtonCallback(window, mouse);
00383
00384 // マウスホイール操作時に呼び出す処理を登録する
00385 glfwSetScrollCallback(window, wheel);
00386
00387 // ウィンドウのサイズ変更時に呼び出す処理を登録する
00388 glfwSetFramebufferSizeCallback(window, resize);
00389
00390 // 垂直同期タイミングに合わせる
00391 glfwSwapInterval(1);
00392
00393 // 実際のフレームバッファのサイズを取得する
00394 glfwGetFramebufferSize(window, &width, &height);
00395
00396 // ビューポートと投影変換行列を初期化する
00397 resize(window, width, height);
00398
00399 #if defined(ImGui_VERSION)
00400 // 最初のウィンドウを開いたとき
00401 static bool firstTime{ true };
00402 if (firstTime)
00403 {
00404     // Setup Platform/Renderer bindings
00405     ImGui_ImplGlfw_InitForOpenGL(window, true);
00406     ImGui_ImplOpenGL3_Init(nullptr);
00407
00408     // 実行済みであることを記録する
00409     firstTime = false;
00410 }
00411 #endif
00412 }
00413
00414 //
00415 // イベントを取得してループを継続すべきかどうか調べる
00416 //
00417 GgApp::Window::operator bool()
00418 {
00419     // イベントを取り出す
00420     glfwPollEvents();
00421
00422     // ウィンドウを閉じるべきなら false を返す
00423     if (shouldClose()) return false;
00424
00425     // 対象のユーザインターフェース
00426     auto& current_if{ interfaceData[interfaceNo] };
00427
00428 #if defined(ImGui_VERSION)
00429 // ImGui の新規フレームを作成する
00430 ImGui_ImplOpenGL3_NewFrame();
00431 ImGui_ImplGlfw_NewFrame();
00432
00433 // ImGui の状態を取り出す
00434 const ImGuiIO& io{ ImGui::GetIO() };
00435 }
```

```

00435 // ImGui がマウスを使うときは Window クラスのマウス位置を更新しない
00436 if (io.WantCaptureMouse) return true;
00437
00438 // マウスの位置を更新する
00439 current_if.mouse = std::array<GLfloat, 2>{ io.MousePos.x, io.MousePos.y };
00440
00441 #else
00442 // マウスの現在位置を調べる
00443 double x, y;
00444 glfwGetCursorPos(window, &x, &y);
00445
00446 // マウスの位置を更新する
00447 current_if.mouse = std::array<GLfloat, 2>{ static_cast<GLfloat>(x), static_cast<GLfloat>(y) };
00448 #endif
00449
00450 // マウスドラッグ
00451 for (int button = GLFW_MOUSE_BUTTON_1; button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT; ++button)
00452 {
00453     // マウスボタンを押していたら
00454     if (status[button])
00455     {
00456         // 現在位置と平行移動量を更新する
00457         current_if.calcTranslation(button, velocity);
00458     }
00459 }
00460
00461 return true;
00462 }
00463
00464 //
00465 // カラーバッファを入れ替える
00466 //
00467 void GgApp::Window::swapBuffers() const
00468 {
00469 #if defined(IMGUI_VERSION)
00470     // ImGui の描画データがあればフレームをレンダリングする
00471     ImDrawData* data{ ImGui::GetDrawData() };
00472     if (data) ImGui_ImplOpenGL3_RenderDrawData(data);
00473 #endif
00474
00475 // エラーチェック
00476 ggError();
00477
00478 // カラーバッファを入れ替える
00479 glfwSwapBuffers(window);
00480 }
00481
00482 //
00483 // ビューポートのサイズを更新する
00484 //
00485 void GgApp::Window::updateViewport()
00486 {
00487     // フレームバッファの大きさを求める
00488     glfwGetFramebufferSize(window, &fboSize[0], &fboSize[1]);
00489
00490 #if defined(IMGUI_VERSION)
00491     // フレームバッファの高さからメニューバーの高さを減じる
00492     fboSize[1] -= menubarHeight;
00493 #endif
00494
00495 // ウィンドウのアスペクト比を保存する
00496 aspect = static_cast<GLfloat>(fboSize[0]) / static_cast<GLfloat>(fboSize[1]);
00497
00498 // ビューポートを設定する
00499 restoreViewport();
00500 }
00501
00502 #if defined(GG_USE_OOCULUS_RIFT)
00503 # if OVR_PRODUCT_VERSION > 0
00504 //
00505 // グラフィックスカードのデフォルトの LUID を得る
00506 //
00507 ovrGraphicsLuid GgApp::Oculus::GetDefaultAdapterLuid()
00508 {
00509     ovrGraphicsLuid luid = ovrGraphicsLuid();
00510
00511 # if defined(_MSC_VER)
00512     IDXGIFactory* factory{ nullptr };
00513
00514     if (SUCCEEDED(CreateDXGIFactory(IID_PPV_ARGS(&factory))))
00515     {
00516         IDXGIAdapter* adapter{ nullptr };
00517
00518         if (SUCCEEDED(factory->EnumAdapters(0, &adapter)))
00519         {
00520             DXGI_ADAPTER_DESC desc;
00521

```

```
00522     adapter->GetDesc(&desc);
00523     memcpy(&luid, &desc.AdapterLuid, sizeof luid);
00524     adapter->Release();
00525 }
00526
00527     factory->Release();
00528 }
00529 # endif
00530
00531     return luid;
00532 }
00533
00534 //
00535 // グラフィックスカードの LUID の比較
00536 //
00537 int GgApp::Oculus::Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs)
00538 {
00539     return memcmp(&lhs, &rhs, sizeof(ovrGraphicsLuid));
00540 }
00541 # endif
00542
00543 //
00544 // コンストラクタ
00545 //
00546 GgApp::Oculus::Oculus() :
00547     session{ nullptr },
00548     oculusFbo{ 0 },
00549     screen{ -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, },
00550     mirrorFbo{ 0 },
00551     window{ nullptr },
00552 # if OVR_PRODUCT_VERSION > 0
00553     frameIndex{ 0 },
00554     oculusDepth{ 0 },
00555     mirrorWidth{ 1280 },
00556     mirrorHeight{ 640 },
00557 # endif
00558     mirrorTexture{ nullptr }
00559 {
00560 }
00561
00562 //
00563 // Oculus Rift のセッションを作成する
00564 //
00565 GgApp::Oculus& GgApp::Oculus::initialize(const Window& window)
00566 {
00567     // Oculus Rift のコンテキスト
00568     static Oculus oculus;
00569
00570     // 既に Oculus Rift のセッションが作成されていたら参照を返す
00571     if (oculus.session) return oculus;
00572
00573     // 最初に呼び出したときだけ実行する
00574     static bool firstTime{ true };
00575     if (firstTime)
00576     {
00577         // Oculus Rift (LibOVR) を初期化する
00578         ovrInitParams initParams{ ovrInit_RequestVersion, OVR_MINOR_VERSION, NULL, 0, 0 };
00579         if (OVR_FAILURE(ovr_Initialize(&initParams)))
00580             throw std::runtime_error("Can't initialize LibOVR");
00581
00582         // アプリケーションの終了時に LibOVR を終了する
00583         atexit(ovr_Shutdown);
00584
00585         // 実行済みであることを記録する
00586         firstTime = false;
00587     }
00588
00589     // Oculus Rift のセッションを作成する
00590     ovrGraphicsLuid luid;
00591     if (OVR_FAILURE(ovr_Create(&oculus.session, &luid)))
00592         throw std::runtime_error("Can't create Oculus Rift session");
00593
00594 # if OVR_PRODUCT_VERSION > 0
00595     // デフォルトのグラフィックスアダプタが使われているか確かめる
00596     if (Compare(luid, GetDefaultAdapterLuid()))
00597         throw std::runtime_error("Graphics adapter is not default");
00598 # endif
00599
00600     // session が無効ならエラー
00601     if (!oculus.session) std::runtime_error("Unable to use the Oculus Rift.");
00602
00603     // ミラー表示を行うウィンドウを設定する
00604     oculus.window = &window;
00605
00606     // Oculus Rift の情報を取り出す
00607     oculus.hmdDesc = ovr_GetHmdDesc(oculus.session);
00608
```

```

00609 # if defined(_DEBUG)
00610 // Oculus Rift の情報を表示する
00611 std::cerr
00612 << "\nProduct name: " << oculus.hmdDesc.ProductName
00613 << "\nResolution: " << oculus.hmdDesc.Resolution.w << " x " << oculus.hmdDesc.Resolution.h
00614 << "\nDefault Fov: (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].LeftTan
00615 << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].DownTan
00616 << ") - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].RightTan
00617 << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].UpTan
00618 << ")\n" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].LeftTan
00619 << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].DownTan
00620 << ") - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].RightTan
00621 << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].UpTan
00622 << "\nMaximum Fov: (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].LeftTan
00623 << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].DownTan
00624 << ") - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].RightTan
00625 << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].UpTan
00626 << ")\n" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].LeftTan
00627 << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].DownTan
00628 << ") - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].RightTan
00629 << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].UpTan
00630 << ")\n" << std::endl;
00631 # endif
00632
00633 // Oculus Rift に転送する描画データを作成する
00634 # if OVR_PRODUCT_VERSION > 0
00635 oculus.layerData.Header.Type = ovrLayerType_EyeFov;
00636 # else
00637 oculus.layerData.Header.Type = ovrLayerType_EyeFovDepth;
00638 # endif
00639
00640 // OpenGL なので左下が原点
00641 oculus.layerData.Header.Flags = ovrLayerFlag_TextureOriginAtBottomLeft;
00642
00643 // Oculus Rift のレンダリングに使う FBO を作成する
00644 glGenFramebuffers(ovrEye_Count, oculus.oculusFbo);
00645
00646 // 全ての目について
00647 for (int eye = 0; eye < ovrEye_Count; ++eye)
00648 {
00649     // Oculus Rift の視野を取得する
00650     const auto& fov{ oculus.hmdDesc.DefaultEyeFov[ovrEyeType(eye)] };
00651
00652     // Oculus Rift 用の FBO のサイズを求める
00653     const auto textureSize{ ovr_GetFovTextureSize(oculus.session, ovrEyeType(eye), fov, 1.0f) };
00654
00655     // Oculus Rift のスクリーンのサイズを保存する
00656     oculus.screen[eye][0] = -fov.LeftTan;
00657     oculus.screen[eye][1] = fov.RightTan;
00658     oculus.screen[eye][2] = -fov.DownTan;
00659     oculus.screen[eye][3] = fov.UpTan;
00660
00661 # if OVR_PRODUCT_VERSION > 0
00662
00663     // 描画データに視野を設定する
00664     oculus.layerData.Fov[eye] = fov;
00665
00666     // 描画データにビューポートを設定する
00667     oculus.layerData.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00668     oculus.layerData.Viewport[eye].Size = textureSize;
00669
00670     // Oculus Rift 用の FBO のカラーバッファとして使うテクスチャセットの特性
00671     const ovrTextureSwapChainDesc colorDesc
00672     {
00673         ovrTexture_2D, // Type
00674         OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00675         1, // ArraySize
00676         textureSize.w, // Width
00677         textureSize.h, // Height
00678         1, // MipLevels
00679         1, // SampleCount
00680         ovrFalse, // StaticImage
00681         0, 0
00682     };
00683
00684     // Oculus Rift 用の FBO のレンダーターゲットとして使うテクスチャチェインを作成する
00685     oculus.layerData.ColorTexture[eye] = nullptr;
00686     if (OVR_SUCCESS(ovr_CreateTextureSwapChainGL(oculus.session, &colorDesc,
00687     &oculus.layerData.ColorTexture[eye])))
00688     {
00689         // 作成したテクスチャチェインの長さを取得する
00690         int length(0);
00691         if (OVR_SUCCESS(ovr_GetTextureSwapChainLength(oculus.session, oculus.layerData.ColorTexture[eye],
00692         &length)))
00693         {
00694             // テクスチャチェインの個々の要素について
00695             for (int i = 0; i < length; ++i)

```

```

00694     {
00695         // テクスチャのパラメータを設定する
00696         GLuint texId{ 0 };
00697         ovr_GetTextureSwapChainBufferGL(oculus.session, oculus.layerData.ColorTexture[eye], i,
00698             &texId);
00699         glBindTexture(GL_TEXTURE_2D, texId);
00700         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00701         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00702         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00703         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00704     }
00705
00706     // Oculus Rift 用の FBO のデプスバッファとして使うテクスチャを作成する
00707     glGenTextures(1, oculus.oculusDepth + eye);
00708     glBindTexture(GL_TEXTURE_2D, oculus.oculusDepth[eye]);
00709     glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h, 0,
00710         GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
00711     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00712     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00713     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00714     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00715 }
00716 # else
00717
00718     // 描画データに視野を設定する
00719     oculus.layerData.EyeFov.Fov[eye] = fov;
00720
00721     // 描画データにビューポートを設定する
00722     oculus.layerData.EyeFov.Viewport[eye].Pos = OVR::Vector2i(0, 0);
00723     oculus.layerData.EyeFov.Viewport[eye].Size = textureSize;
00724
00725     // Oculus Rift 用の FBO のカラーバッファとして使うテクスチャセットを作成する
00726     ovrSwapTextureSet* colorTexture{ nullptr };
00727     ovr_CreateSwapTextureSetGL(oculus.session, GL_SRGB8_ALPHA8, textureSize.w, textureSize.h,
00728         &colorTexture);
00729     oculus.layerData.EyeFov.ColorTexture[eye] = colorTexture;
00730
00731     // Oculus Rift 用の FBO のデプスバッファとして使うテクスチャセットを作成する
00732     ovrSwapTextureSet* depthTexture{ nullptr };
00733     ovr_CreateSwapTextureSetGL(oculus.session, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h,
00734         &depthTexture);
00735     oculus.layerData.EyeFovDepth.DepthTexture[eye] = depthTexture;
00736
00737     // Oculus Rift のレンズ補正等の設定値を取得する
00738     oculus.eyeRenderDesc[eye] = ovr_GetRenderDesc(oculus.session, ovrEyeType(eye), fov);
00739
00740 # endif
00741 }
00742
00743 // 姿勢のトラッキングにおける床の高さを 0 に設定する
00744 ovr_SetTrackingOriginType(oculus.session, ovrTrackingOrigin_FloorLevel);
00745
00746 // ミラー表示用の FBO を作成する
00747 const GLsizei* size{ oculus.window->getSize() };
00748 const ovrMirrorTextureDesc mirrorDesc
00749 {
00750     OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
00751     oculus.mirrorWidth = size[0], // Width
00752     oculus.mirrorHeight = size[1], // Height
00753     0 // Flags
00754 };
00755
00756 // ミラー表示用の FBO のカラーバッファとして使うテクスチャを作成する
00757 if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, &mirrorDesc, &oculus.mirrorTexture)))
00758 {
00759     // 作成したテクスチャのテクスチャ名を得る
00760     GLuint texId{ 0 };
00761     if (OVR_SUCCESS(ovr_GetMirrorTextureBufferGL(oculus.session, oculus.mirrorTexture, &texId)))
00762     {
00763         // ミラー表示用の FBO を作成してテクスチャをカラーバッファとして組み込む
00764         glGenFramebuffers(1, &oculus.mirrorFbo);
00765         glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00766         glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texId, 0);
00767         glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
00768         glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00769     }
00770 }
00771
00772 # else
00773
00774     // 作成したテクスチャのテクスチャ名を得る
00775     if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, GL_SRGB8_ALPHA8, width, height,
00776         reinterpret_cast<ovrTexture**>(&mirrorTexture))))

```

```

00776  {
00777      // ミラー表示用の FBO を作成してテクスチャをカラーバッファとして組み込む
00778      oculus.mirrorFbo = 0;
00779      glGenFramebuffers(1, &oculus.mirrorFbo);
00780      glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
00781      glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
00782          mirrorTexture->OGL.TexId, 0);
00783      glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
00784  }
00785
00786 # endif
00787
00788 // Oculus Rift にレンダリングするときは sRGB カラースペースを使う
00789 glEnable(GL_FRAMEBUFFER_SRGB);
00790
00791 // フロントバッファに描く
00792 glDrawBuffer(GL_FRONT);
00793
00794 // Oculus Rift への表示では垂直同期タイミングに合わせない
00795 glfwSwapInterval(0);
00796
00797 return oculus;
00798 }
00799
00800 //
00801 // Oculus Rift のセッションを破棄する
00802 //
00803 void GgApp::Oculus::terminate()
00804 {
00805     // session が無効なら何もしない
00806     if (!session) return;
00807
00808     // ミラー表示用の FBO を作っていたら削除する
00809     if (mirrorFbo)
00810     {
00811         glDeleteFramebuffers(1, &mirrorFbo);
00812         mirrorFbo = 0;
00813     }
00814
00815     // ミラー表示用の FBO のカラーバッファ用のテクスチャを作っていたら削除する
00816     if (mirrorTexture)
00817     {
00818 # if OVR_PRODUCT_VERSION > 0
00819         ovr_DestroyMirrorTexture(session, mirrorTexture);
00820 # else
00821         glDeleteTextures(1, &mirrorTexture->OGL.TexId);
00822         ovr_DestroyMirrorTexture(session, reinterpret_cast<ovrTexture*>(mirrorTexture));
00823 # endif
00824         mirrorTexture = nullptr;
00825     }
00826
00827     // 全ての目について
00828     for (int eye = 0; eye < ovrEye_Count; ++eye)
00829     {
00830         // Oculus Rift へのレンダリング用の FBO を削除する
00831         glDeleteFramebuffers(1, oculusFbo + eye);
00832         oculusFbo[eye] = 0;
00833
00834 # if OVR_PRODUCT_VERSION > 0
00835
00836         // レンダリングターゲットに使ったテクスチャを削除する
00837         if (layerData.ColorTexture[eye])
00838         {
00839             ovr_DestroyTextureSwapChain(session, layerData.ColorTexture[eye]);
00840             layerData.ColorTexture[eye] = nullptr;
00841         }
00842
00843         // デプスバッファとして使ったテクスチャを削除する
00844         glDeleteTextures(1, oculusDepth + eye);
00845         oculusDepth[eye] = 0;
00846
00847 # else
00848
00849         // レンダリングターゲットに使ったテクスチャを削除する
00850         auto* const colorTexture(layerData.EyeFov.ColorTexture[eye]);
00851         for (int i = 0; i < colorTexture->TextureCount; ++i)
00852         {
00853             const auto* const ctex(reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[i]));
00854             glDeleteTextures(1, &ctex->OGL.TexId);
00855         }
00856         ovr_DestroySwapTextureSet(session, colorTexture);
00857
00858         // デプスバッファとして使ったテクスチャを削除する
00859         auto* const depthTexture(layerData.EyeFovDepth.DepthTexture[eye]);
00860         for (int i = 0; i < depthTexture->TextureCount; ++i)
00861         {

```

```
00862     const auto* const dtex(reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[i]));
00863     glDeleteTextures(1, &dtex->OGL.TexId);
00864 }
00865 ovr_DestroySwapTextureSet(session, depthTexture);
00866
00867 # endif
00868 }
00869
00870 // Oculus Rift のセッションを破棄する
00871 ovr_Destroy(session);
00872 session = nullptr;
00873
00874 // カラースペースを元に戻す
00875 glDisable(GL_FRAMEBUFFER_SRGB);
00876
00877 // バックバッファに描く
00878 glDrawBuffer(GL_BACK);
00879
00880 // 垂直同期タイミングに合わせる
00881 glfwSwapInterval(1);
00882 }
00883
00884 //
00885 // Oculus Rift による描画開始
00886 //
00887 bool GgApp::Oculus::begin()
00888 {
00889 # if OVR_PRODUCT_VERSION > 0
00890
00891 // セッションの状態を取得する
00892 ovrSessionStatus sessionStatus;
00893 ovr_GetSessionStatus(session, &sessionStatus);
00894
00895 // アプリケーションが終了を要求しているときはウィンドウのクローズフラグを立てる
00896 if (sessionStatus.ShouldQuit) window->setClose(GLFW_TRUE);
00897
00898 // Oculus Rift に表示されていないときは戻る
00899 if (!sessionStatus.IsVisible) return false;
00900
00901 // 現在の状態をトラッキングの原点にする
00902 if (sessionStatus.ShouldRecenter) ovr_RecenterTrackingOrigin(session);
00903
00904 // HmdToEyeOffset などは実行時に変化するので毎フレーム ovr_GetRenderDesc() で ovrEyeRenderDesc を取得する
00905 const ovrEyeRenderDesc eyeRenderDesc[]
00906 {
00907     ovr_GetRenderDesc(session, ovrEyeType(0), hmdDesc.DefaultEyeFov[0]),
00908     ovr_GetRenderDesc(session, ovrEyeType(1), hmdDesc.DefaultEyeFov[1])
00909 };
00910
00911 // Oculus Rift のスクリーンのヘッドトラッキング位置からの変位を取得する
00912 const ovrPosef hmdToEyePose[]
00913 {
00914     eyeRenderDesc[0].HmdToEyePose,
00915     eyeRenderDesc[1].HmdToEyePose
00916 };
00917
00918 // 視点の姿勢情報を取得する
00919 ovr_GetEyePoses(session, frameIndex, ovrTrue, hmdToEyePose, &layerData.RenderPose,
&layerData.SensorSampleTime);
00920
00921 # else
00922
00923 // フレームのタイミング計測開始
00924 const auto ftiming(ovr_GetPredictedDisplayTime(session, 0));
00925
00926 // sensorSampleTime の取得は可能な限り ovr_GetTrackingState() の近くで行う
00927 layerData.EyeFov.SensorSampleTime = ovr_GetTimeInSeconds();
00928
00929 // ヘッドトラッキングの状態を取得する
00930 const auto hmdState(ovr_GetTrackingState(session, ftiming, ovrTrue));
00931
00932 // Oculus Rift のスクリーンのヘッドトラッキング位置からの変位を取得する
00933 const ovrVector3f hmdToEyeViewOffset[]
00934 {
00935     eyeRenderDesc[0].HmdToEyeViewOffset,
00936     eyeRenderDesc[1].HmdToEyeViewOffset
00937 };
00938
00939 // 視点の姿勢情報を求める
00940 ovr_CalcEyePoses(hmdState.HeadPose.ThePose, hmdToEyeViewOffset, eyePose);
00941
00942 # endif
00943
00944 return true;
00945 }
00946
00947 //
```

```

00948 // Oculus Rift の描画する目の指定
00949 //
00950 void GgApp::Oculus::select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation)
00951 {
00952 # if OVR_PRODUCT_VERSION > 0
00953
00954 // Oculus Rift にレンダリングする FBO に切り替える
00955 if (layerData.ColorTexture[eye])
00956 {
00957 // FBO のカラー バッファに使う現在のテクスチャのインデックスを取得する
00958 int curIndex;
00959 ovr_GetTextureSwapChainCurrentIndex(session, layerData.ColorTexture[eye], &curIndex);
00960
00961 // FBO のカラー バッファに使うテクスチャを取得する
00962 GLuint curTexId;
00963 ovr_GetTextureSwapChainBufferGL(session, layerData.ColorTexture[eye], curIndex, &curTexId);
00964
00965 // FBO を設定する
00966 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
00967 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, curTexId, 0);
00968 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, oculusDepth[eye], 0);
00969
00970 // ビューポートを設定する
00971 const auto& vp{ layerData.Viewport[eye] };
00972 glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
00973 }
00974
00975 // Oculus Rift の片目の位置と回軸を取得する
00976 const auto& p{ layerData.RenderPose[eye].Position };
00977 const auto& o{ layerData.RenderPose[eye].Orientation };
00978
00979 # else
00980
00981 // レンダーターゲットに描画する前にレンダーターゲットのインデックスをインクリメントする
00982 auto* const colorTexture{ layerData.EyeFov.ColorTexture[eye] };
00983 colorTexture->CurrentIndex = (colorTexture->CurrentIndex + 1) % colorTexture->TextureCount;
00984 auto* const depthTexture{ layerData.EyeFovDepth.DepthTexture[eye] };
00985 depthTexture->CurrentIndex = (depthTexture->CurrentIndex + 1) % depthTexture->TextureCount;
00986
00987 // レンダーターゲットを切り替える
00988 glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
00989 const auto& ctex{
00990 reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[colorTexture->CurrentIndex]) ;
00991 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ctex->OGL.TexId, 0);
00992 const auto& dtex{
00993 reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[depthTexture->CurrentIndex]) ;
00994 glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, dtex->OGL.TexId, 0);
00995
00996 // ビューポートを設定する
00997 const auto& vp{ layerData.EyeFov.Viewport[eye] };
00998 glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
00999
01000 // Oculus Rift の片目の位置と回軸を取得する
01001 const auto& p{ eyePose[eye].Position };
01002 const auto& o{ eyePose[eye].Orientation };
01003
01004 // Oculus Rift のスクリーンの大きさを返す
01005 screen[0] = this->screen[eye][0];
01006 screen[1] = this->screen[eye][1];
01007 screen[2] = this->screen[eye][2];
01008 screen[3] = this->screen[eye][3];
01009
01010 // Oculus Rift の位置を返す
01011 position[0] = p.x;
01012 position[1] = p.y;
01013 position[2] = p.z;
01014
01015 // Oculus Rift の方向を返す
01016 orientation[0] = o.x;
01017 orientation[1] = o.y;
01018 orientation[2] = o.z;
01019 orientation[3] = o.w;
01020 }
01021
01022 //
01023 // Time Warp 处理に使う投影変換行列の成分の設定 (DK1, DK2)
01024 //
01025 void GgApp::Oculus::timewarp(const GgMatrix& projection)
01026 {
01027 # if OVR_PRODUCT_VERSION < 1
01028 // TimeWarp に使う変換行列の成分を設定する
01029 auto& posTimewarpProjectionDesc{ layerData.EyeFovDepth.ProjectionDesc };
01030 posTimewarpProjectionDesc.Projection22 = (projection.get()[4 * 2 + 2] + projection.get()[4 * 3 + 2])
01031 * 0.5f;
01032 posTimewarpProjectionDesc.Projection23 = projection.get()[4 * 2 + 3] * 0.5f;

```

```

01032     posTimewarpProjectionDesc.Projection32 = projection.get()[4 * 3 + 2];
01033 # endif
01034 }
01035
01036 // 
01037 // 図形の描画を完了する (CV1 以降)
01038 //
01039 void GgApp::Oculus::commit(int eye)
01040 {
01041 # if OVR_PRODUCT_VERSION > 0
01042     // GL_COLOR_ATTACHMENT0 に割り当てられたテクスチャが wglDXUnlockObjectsNV() によって
01043     // アンロックされるために次のフレームの処理において無効な GL_COLOR_ATTACHMENT0 が
01044     // FBO に結合されるのを避ける
01045     glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
01046     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
01047     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, 0, 0);
01048
01049     // 保留中の変更を layerData.ColorTexture[eye] に反映しインデックスを更新する
01050     ovrCommitTextureSwapChain(session, layerData.ColorTexture[eye]);
01051 # endif
01052 }
01053
01054 //
01055 // フレームを転送する
01056 //
01057 bool GgApp::Oculus::submit(bool mirror)
01058 {
01059     // エラーチェック
01060     ggError();
01061
01062 # if OVR_PRODUCT_VERSION > 0
01063     // 描画データを oculus Rift に転送する
01064     const auto* const layers{ &layerData.Header };
01065     if (OVR_FAILURE(ovrSubmitFrame(session, frameIndex++, nullptr, &layers, 1))) return false;
01066 # else
01067     // Oculus Rift 上の描画位置と拡大率を求める
01068     ovrViewScaleDesc viewScaleDesc;
01069     viewScaleDesc.HmdSpaceToWorldScaleInMeters = 1.0f;
01070     viewScaleDesc.HmdToEyeViewOffset[0] = eyeRenderDesc[0].HmdToEyeViewOffset;
01071     viewScaleDesc.HmdToEyeViewOffset[1] = eyeRenderDesc[1].HmdToEyeViewOffset;
01072
01073     // 描画データを更新する
01074     layerData.EyeFov.RenderPose[0] = eyePose[0];
01075     layerData.EyeFov.RenderPose[1] = eyePose[1];
01076
01077     // 描画データを oculus Rift に転送する
01078     const auto* const layers{ &layerData.Header };
01079     if (OVR_FAILURE(ovrSubmitFrame(session, 0, &viewScaleDesc, &layers, 1))) return false;
01080 # endif
01081
01082     // ミラー表示
01083     if (mirror)
01084     {
01085 # if OVR_PRODUCT_VERSION > 0
01086         const auto& sx1{ mirrorWidth };
01087         const auto& sy1{ mirrorHeight };
01088 # else
01089         const auto& sx1{ mirrorTexture->OGL.Header.TextureSize.w };
01090         const auto& sy1{ mirrorTexture->OGL.Header.TextureSize.h };
01091 # endif
01092
01093     // ミラー表示のウィンドウのサイズ
01094     GLsizei size[2];
01095     window->getSize(size);
01096
01097     // ミラー表示の表示領域
01098     GLint dx0{ 0 }, dx1{ size[0] }, dy0{ 0 }, dy1{ size[1] };
01099
01100     // ミラー表示が「ウィンドウからはみ出ない」ようにする
01101     if ((size[0] *= sy1) < (size[1] *= sx1))
01102     {
01103         const GLint tyl{ size[0] / sx1 };
01104         dy0 = (dy1 - tyl) / 2;
01105         dy1 = dy0 + tyl;
01106     }
01107     else
01108     {
01109         const GLint tx1{ size[1] / sy1 };
01110         dx0 = (dx1 - tx1) / 2;
01111         dx1 = dx0 + tx1;
01112     }
01113
01114     // レンダリング結果をミラー表示用のフレームバッファにも転送する
01115     glBindFramebuffer(GL_READ_FRAMEBUFFER, mirrorFbo);
01116     glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
01117     glBindFramebuffer(0, sy1, sx1, 0, dx0, dy0, dx1, dy1, GL_COLOR_BUFFER_BIT, GL_NEAREST);
01118     glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);

```

```

01119     // 残っている OpenGL コマンドを実行する
01120     glFlush();
01121 }
01123
01124     return true;
01125 }
01126 #endif

```

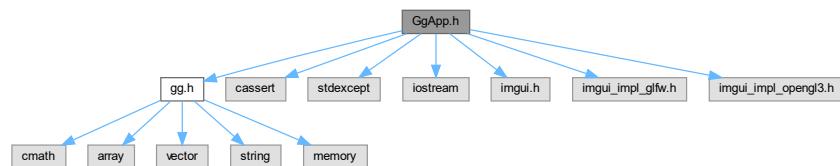
9.15 GgApp.h ファイル

```

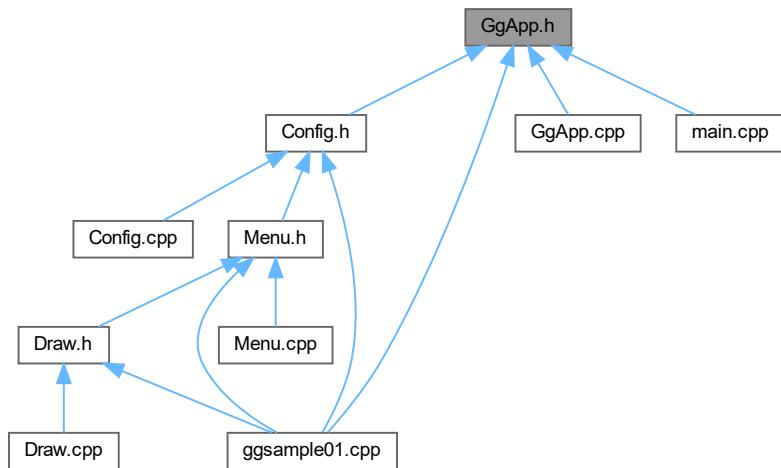
#include "gg.h"
#include <cassert>
#include <stdexcept>
#include <iostream>
#include "imgui.h"
#include "imgui_impl_glfw.h"
#include "imgui_impl_opengl3.h"

```

GgApp.h の依存先関係図:



被依存関係図:



クラス

- class [GgApp](#)
- class [GgApp::Window](#)

マクロ定義

- `#define GG_USE_IMGUI`
- `#define GG_BUTTON_COUNT 3`
- `#define GG_INTERFACE_COUNT 5`

9.15.1 詳解

ゲームグラフィックス特論宿題アプリケーションクラスの定義

著者

Kohe Tokoi

日付

November 22, 2022

[GgApp.h](#) に定義があります。

9.15.2 マクロ定義詳解

9.15.2.1 GG_BUTTON_COUNT

```
#define GG_BUTTON_COUNT 3
```

[GgApp.h](#) の 19 行目に定義があります。

9.15.2.2 GG_INTERFACE_COUNT

```
#define GG_INTERFACE_COUNT 5
```

[GgApp.h](#) の 24 行目に定義があります。

9.15.2.3 GG_USE_IMGUI

```
#define GG_USE_IMGUI
```

[GgApp.h](#) の 12 行目に定義があります。

9.16 GgApp.h

[詳解]

```

00001 #pragma once
00002
00010
00011 // Dear ImGui を使うなら
00012 #define GG_USE_IMGUI
00013
00014 // Oculus Rift を使うなら
00015 // #define GG_USE_OOCULUS_RIFT
00016
00017 // 使用するマウスのボタン数
00018 #if !defined(GG_BUTTON_COUNT)
00019 # define GG_BUTTON_COUNT 3
00020 #endif
00021
00022 // 使用するユーザインターフェースの数
00023 #if !defined(GG_INTERFACE_COUNT)
00024 # define GG_INTERFACE_COUNT 5
00025 #endif
00026
00027 // 補助プログラム
00028 #include "gg.h"
00029 using namespace gg;
00030
00031 // 標準ライブラリ
00032 #include <cassert>
00033 #include <stdexcept>
00034 #include <iostream>
00035
00036 // ImGui の組み込み
00037 #if defined(GG_USE_IMGUI)
00038 # include "imgui.h"
00039 # include "imgui_impl_glfw.h"
00040 # include "imgui_impl_opengl3.h"
00041 #endif
00042
00043 // Oculus Rift SDK ライブラリ (LibOVR) の組み込み
00044 #if defined(GG_USE_OOCULUS_RIFT)
00045 # if defined(_MSC_VER)
00046 # define GLFW_EXPOSE_NATIVE_WIN32
00047 # define GLFW_EXPOSE_NATIVE_WGL
00048 # include <GLFW/glfw3native.h>
00049 # define OVR_OS_WIN32
00050 # undef APIENTRY
00051 # pragma comment(lib, "LibOVR.lib")
00052 # endif
00053 # include <OVR_CAPI_GL.h>
00054 # include <Extras/OVR_Math.h>
00055 # if OVR_PRODUCT_VERSION > 0
00056 # include <dxgi.h> // GetDefaultAdapterLuid のため
00057 # pragma comment(lib, "dxgi.lib")
00058 # endif
00059 #endif
00060
00064 class GgApp
00065 {
00066 public:
00067
00074 GgApp(int major = 0, int minor = 1);
00075
00076 //
00077 // コピーコンストラクタは封じる
00078 //
00079 GgApp(const GgApp& w) = delete;
00080
00084 virtual ~GgApp();
00085
00086 //
00087 // 代入演算子は封じる
00088 //
00089 GgApp& operator=(const GgApp& w) = delete;
00090
00094 int main(int argc, const char* const* argv);
00095
00102 class Window
00103 {
00104 // ウィンドウの識別子
00105 GLFWwindow* window;
00106
00107 // ビューポートの横幅と高さ
00108 std::array<GLsizei, 2> size;
00109
00110 // フレームバッファの横幅と高さ

```

```
00111     std::array<GLsizei, 2> fboSize;
00112
00113 #if defined(IMGUI_VERSION)
00114     // メニューバーの高さ
00115     GLsizei menubarHeight;
00116 #endif
00117
00118     // ビューポートのアスペクト比
00119     GLfloat aspect;
00120
00121     // マウスの移動速度[X/Y/Z]
00122     std::array<GLfloat, 3> velocity;
00123
00124     // マウスボタンの状態
00125     std::array<bool, GG_BUTTON_COUNT> status;
00126
00127     // ユーザインターフェースのデータ構造
00128     struct HumanInterface
00129     {
00130         // 最後にタイプしたキー
00131         int lastKey;
00132
00133         // 矢印キー
00134         std::array<std::array<int, 2>, 4> arrow;
00135
00136         // マウスの現在位置
00137         std::array<GLfloat, 2> mouse;
00138
00139         // マウスホイールの回転量
00140         std::array<GLfloat, 2> wheel;
00141
00142         // 平行移動量[ボタン][直前/更新][X/Y/Z]
00143         std::array<std::array<GGVector, 2>, GG_BUTTON_COUNT> translation;
00144
00145         // トラックボール
00146         std::array<GGTrackball, GG_BUTTON_COUNT> rotation;
00147
00148         // コンストラクタ
00149         HumanInterface() :
00150             lastKey{ 0 },
00151             arrow{},
00152             mouse{},
00153             wheel{},
00154             translation{}
00155         {
00156             resetTranslation();
00157         }
00158
00159         // マウスや矢印キーによる平行移動量を初期化する
00160         // 
00161         void resetTranslation();
00162
00163         // 平行移動量と回転量を更新する (X, Y のみ, Z は wheel() で計算する)
00164         // 
00165         void calcTranslation(int button, const std::array<GLfloat, 3>& velocity);
00166     };
00167
00168     // ヒューマンインターフェースデバイスのデータ
00169     std::array<HumanInterface, GG_INTERFACE_COUNT> interfaceData;
00170
00171     // ヒューマンインターフェースデバイスの番号
00172     int interfaceNo;
00173
00174     // 
00175     // ユーザー定義のコールバック関数へのポインタ
00176     // 
00177     void* userPointer;
00178
00179     void (*resizeFunc)(const Window* window, int width, int height);
00180     void (*keyboardFunc)(const Window* window, int key, int scancode, int action, int mods);
00181     void (*mouseFunc)(const Window* window, int button, int action, int mods);
00182     void (*wheelFunc)(const Window* window, double x, double y);
00183
00184     // 
00185     // ウィンドウのサイズ変更時の処理
00186     // 
00187     static void resize(GLFWwindow* window, int width, int height);
00188
00189     // 
00190     // キーボードをタイプした時の処理
00191     // 
00192     static void keyboard(GLFWwindow* window, int key, int scancode, int action, int mods);
00193
00194     // 
00195     // マウスボタンを操作したときの処理
00196     // 
00197     //
```

```

00198     static void mouse(GLFWwindow* window, int button, int action, int mods);
00199
00200     //
00201     // マウスホイールを操作した時の処理
00202     //
00203     static void wheel(GLFWwindow* window, double x, double y);
00204
00205 public:
00206
00216     Window(const std::string& title = "GLFW Window", int width = 640, int height = 480,
00217         int fullscreen = 0, GLFWwindow* share = nullptr);
00218
00219     //
00220     // コピーコンストラクタは封じる
00221     //
00222     Window(const Window& w) = delete;
00223
00227     virtual ~Window()
00228     {
00229         // ウィンドウが作成されていなければ戻る
00230         if (!window) return;
00231
00232         // ウィンドウを破棄する
00233         glfwDestroyWindow(window);
00234     }
00235
00236     //
00237     // 代入演算子は封じる
00238     //
00239     Window& operator=(const Window& w) = delete;
00240
00246     auto* get() const
00247     {
00248         return window;
00249     }
00250
00256     void setClose(int flag = GLFW_TRUE) const
00257     {
00258         glfwSetWindowShouldClose(window, flag);
00259     }
00260
00266     bool shouldClose() const
00267     {
00268         // ウィンドウを閉じるべきなら true を返す
00269         return glfwWindowShouldClose(window) != GLFW_FALSE;
00270     }
00271
00277     explicit operator bool();
00278
00282     void swapBuffers() const;
00283
00287     void restoreViewport() const
00288     {
00289         glViewport(0, 0, fboSize[0], fboSize[1]);
00290     }
00291
00295     void updateViewport();
00296
00297 #if defined(IMGUI_VERSION)
00303     void setMenubarHeight(GLsizei height)
00304     {
00305         // メニューバーの高さを保存する
00306         menubarHeight = height;
00307
00308         // ビューポートを復帰する
00309         updateViewport();
00310     }
00311 #endif
00312
00318     auto getWidth() const
00319     {
00320         return size[0];
00321     }
00322
00328     auto getHeight() const
00329     {
00330         return size[1];
00331     }
00332
00338     auto getFboWidth() const
00339     {
00340         return fboSize[0];
00341     }
00342
00348     auto getFboHeight() const
00349     {
00350         return fboSize[1];
00350

```

```
00351      }
00352
00353     const auto& getSize() const
00354     {
00355         return size;
00356     }
00357
00358     void getSize(GLsizei* size) const
00359     {
00360         size[0] = getWidth();
00361         size[1] = getHeight();
00362     }
00363
00364     const auto getFboSize() const
00365     {
00366         return fboSize.data();
00367     }
00368
00369     void getFboSize(GLsizei* fboSize) const
00370     {
00371         fboSize[0] = getFboWidth();
00372         fboSize[1] = getFboHeight();
00373     }
00374
00375     auto getAspect() const
00376     {
00377         return aspect;
00378     }
00379
00380     bool getKey(int key) const
00381     {
00382 #if defined(IMGUI_VERSION)
00383         // ImGui カ#キーボードを使うときはキーボードの処理を行わない
00384         if (ImGui::GetIO().WantCaptureKeyboard) return false;
00385 #endif
00386
00387         return glfwGetKey(window, key) != GLFW_RELEASE;
00388     }
00389
00390     void selectInterface(int no)
00391     {
00392         assert(static_cast<size_t>(no) < interfaceData.size());
00393         interfaceNo = no;
00394     }
00395
00396     void setVelocity(GLfloat vx, GLfloat vy, GLfloat vz = 0.1f)
00397     {
00398         velocity = std::array<GLfloat, 3>{ vx, vy, vz };
00399     }
00400
00401     int getLastKey()
00402     {
00403         auto& current_if{ interfaceData[interfaceNo] };
00404         const int key{ current_if.lastKey };
00405         current_if.lastKey = 0;
00406         return key;
00407     }
00408
00409     auto getArrow(int direction = 0, int mods = 0) const
00410     {
00411         const auto& current_if{ interfaceData[interfaceNo] };
00412         return static_cast<GLfloat>(current_if.arrow[mods & 3][direction & 1]);
00413     }
00414
00415     auto getArrowX(int mods = 0) const
00416     {
00417         return getArrow(0, mods);
00418     }
00419
00420     auto getArrowY(int mods = 0) const
00421     {
00422         return getArrow(1, mods);
00423     }
00424
00425     void getArrow(GLfloat* arrow, int mods = 0) const
00426     {
00427         arrow[0] = getArrowX(mods);
00428         arrow[1] = getArrowY(mods);
00429     }
00430
00431     auto getShiftArrowX() const
00432     {
00433         return getArrow(0, 1);
00434     }
00435
00436     auto getShiftArrowY() const
00437     {
```

```
00520     return getArrow(1, 1);
00521 }
00522
00528 void getShiftArrow(GLfloat* shift_arrow) const
00529 {
00530     shift_arrow[0] = getShiftArrowX();
00531     shift_arrow[1] = getShiftArrowY();
00532 }
00533
00539 auto getControlArrowX() const
00540 {
00541     return getArrow(0, 2);
00542 }
00543
00549 auto getControlArrowY() const
00550 {
00551     return getArrow(1, 2);
00552 }
00553
00559 void getControlArrow(GLfloat* control_arrow) const
00560 {
00561     control_arrow[0] = getControlArrowX();
00562     control_arrow[1] = getControlArrowY();
00563 }
00564
00570 auto getAltArrowX() const
00571 {
00572     return getArrow(0, 3);
00573 }
00574
00580 auto getAltArrowY() const
00581 {
00582     return getArrow(1, 3);
00583 }
00584
00590 void getAlt1Arrow(GLfloat* alt_arrow) const
00591 {
00592     alt_arrow[0] = getAltArrowX();
00593     alt_arrow[1] = getAltArrowY();
00594 }
00595
00601 const auto* getMouse() const
00602 {
00603     const auto& current_if{ interfaceData[interfaceNo] };
00604     return current_if.mouse.data();
00605 }
00606
00612 void getMouse(GLfloat* position) const
00613 {
00614     const auto& current_if{ interfaceData[interfaceNo] };
00615     position[0] = current_if.mouse[0];
00616     position[1] = current_if.mouse[1];
00617 }
00618
00625 auto getMouse(int direction) const
00626 {
00627     const auto& current_if{ interfaceData[interfaceNo] };
00628     return current_if.mouse[direction & 1];
00629 }
00630
00636 auto getMouseX() const
00637 {
00638     const auto& current_if{ interfaceData[interfaceNo] };
00639     return current_if.mouse[0];
00640 }
00641
00647 auto getMouseY() const
00648 {
00649     const auto& current_if{ interfaceData[interfaceNo] };
00650     return current_if.mouse[1];
00651 }
00652
00658 const auto* getWheel() const
00659 {
00660     const auto& current_if{ interfaceData[interfaceNo] };
00661     return current_if.wheel.data();
00662 }
00663
00669 void getWheel(GLfloat* rotation) const
00670 {
00671     const auto& current_if{ interfaceData[interfaceNo] };
00672     rotation[0] = current_if.wheel[0];
00673     rotation[1] = current_if.wheel[1];
00674 }
00675
00682 auto getWheel(int direction) const
00683 {
```

```

00684     const auto& current_if{ interfaceData[interfaceNo] };
00685     return current_if.wheel[direction & 1];
00686 }
00687
00693 auto getWheelX() const
00694 {
00695     const auto& current_if{ interfaceData[interfaceNo] };
00696     return current_if.wheel[0];
00697 }
00698
00704 auto getWheelY() const
00705 {
00706     const auto& current_if{ interfaceData[interfaceNo] };
00707     return current_if.wheel[1];
00708 }
00709
00716 const auto& getTranslation(int button = GLFW_MOUSE_BUTTON_1) const
00717 {
00718     const auto& current_if{ interfaceData[interfaceNo] };
00719     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00720     return current_if.translation[button][1];
00721 }
00722
00729 auto getTranslationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00730 {
00731     const auto& current_if{ interfaceData[interfaceNo] };
00732     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00733     const auto& t{ current_if.translation[button][1] };
00734     GgMatrix m;
00735     m[ 1] = m[ 2] = m[ 3] = m[ 4] = m[ 6] = m[ 7] = m[ 8] = m[ 9] = m[11] = 0.0f;
00736     m[ 0] = m[ 5] = m[10] = m[15] = 1.0f;
00737     m[12] = t[0];
00738     m[13] = t[1];
00739     m[14] = t[2];
00740     return m;
00741 }
00742
00749 auto getScrollMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00750 {
00751     const auto& current_if{ interfaceData[interfaceNo] };
00752     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00753     const auto& t{ current_if.translation[button][1] };
00754     GgMatrix m;
00755     m[ 0] = m[ 5] = t[2] + 1.0f;
00756     m[ 1] = m[ 2] = m[ 3] = m[ 4] = m[ 6] = m[ 7] = m[ 8] = m[ 9] = m[11] = m[14] = 0.0f;
00757     m[10] = m[15] = 1.0f;
00758     m[12] = t[0];
00759     m[13] = t[1];
00760     return m;
00761 }
00762
00769 auto getRotation(int button = GLFW_MOUSE_BUTTON_1) const
00770 {
00771     const auto& current_if{ interfaceData[interfaceNo] };
00772     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00773     return current_if.rotation[button].getQuaternion();
00774 }
00775
00782 auto getRotationMatrix(int button = GLFW_MOUSE_BUTTON_1) const
00783 {
00784     const auto& current_if{ interfaceData[interfaceNo] };
00785     assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
00786     return current_if.rotation[button].getMatrix();
00787 }
00788
00792 void resetRotation()
00793 {
00794     // トラックボールを初期化する
00795     for (auto& tb : interfaceData[interfaceNo].rotation) tb.reset();
00796 }
00797
00801 void resetTranslation()
00802 {
00803     // 現在のインターフェースの平行移動量を初期化する
00804     interfaceData[interfaceNo].resetTranslation();
00805 }
00806
00810 void reset()
00811 {
00812     // トラックボール処理を初期化する
00813     resetRotation();
00814
00815     // 平行移動量を初期化する
00816     resetTranslation();
00817 }
00818
00824 void* getUserPointer() const

```

```

00825     {
00826         return userPointer;
00827     }
00828
00834     void setUserPointer(void* pointer)
00835     {
00836         userPointer = pointer;
00837     }
00838
00844     void setResizeFunc(void (*func) (const Window* window, int width, int height))
00845     {
00846         resizeFunc = func;
00847     }
00848
00854     void setKeyboardFunc(void (*func) (const Window* window, int key, int scancode, int action, int
00855     mods))
00856     {
00857         keyboardFunc = func;
00858     }
00864     void setMouseFunc(void (*func) (const Window* window, int button, int action, int mods))
00865     {
00866         mouseFunc = func;
00867     }
00868
00874     void setWheelFunc(void (*func) (const Window* window, double x, double y))
00875     {
00876         wheelFunc = func;
00877     }
00878 };
00879
00880 #if defined(GG_USE_OOCULUS_RIFT)
00881     class Oculus
00882     {
00883         // Oculus Rift のセッション
00884         ovrSession session;
00885
00886         // Oculus Rift の状態
00887         ovrHmdDesc hmdDesc;
00888
00889         // Oculus Rift へのレンダリングに使う FBO
00890         GLuint oculusFbo[ovrEye_Count];
00891
00892         // Oculus Rift のスクリーンのサイズ
00893         GLfloat screen[ovrEye_Count][4];
00894
00895         // ミラー表示用の FBO
00896         GLuint mirrorFbo;
00897
00898         // Oculus Rift のミラー表示を行うウィンドウ
00899         const Window* window;
00900
00901 # if OVR_PRODUCT_VERSION > 0
00902
00903         // Oculus Rift に送る描画データ
00904         ovrLayerEyeFov layerData;
00905
00906         // Oculus Rift にレンダリングするフレームの番号
00907         long long frameIndex;
00908
00909         // Oculus Rift へのレンダリングに使う FBO のデブステクスチャ
00910         GLuint oculusDepth[ovrEye_Count];
00911
00912         // ミラー表示用の FBO のサイズ
00913         int mirrorWidth, mirrorHeight;
00914
00915         // ミラー表示用の FBO のカラー テクスチャ
00916         ovrMirrorTexture mirrorTexture;
00917
00918         // グラフィックスカードのデフォルトの LUID を得る
00919         static ovrGraphicsLuid GetDefaultAdapterLuid();
00920
00921         // グラフィックスカードの LUID の比較
00922         static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);
00923
00924         //
00925         // グラフィックスカードのデフォルトの LUID を得る
00926         //
00927         static ovrGraphicsLuid GetDefaultAdapterLuid();
00928
00929         //
00930         // グラフィックスカードの LUID の比較
00931         //
00932         static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);
00933
00934 # else
00935
00936         // Oculus Rift に送る描画データ
00937         ovrLayerUnion layerData;
00938
00939         // Oculus Rift のレンダリング情報
00940         ovrEyeRenderDesc eyeRenderDesc[ovrEye_Count];
00941

```

```

00942     // Oculus Rift の視点情報
00943     ovrPosef eyePose[ovrEye_Count];
00944
00945     // ミラー表示用の FBO のカラーテクスチャ
00946     ovrGLTexture* mirrorTexture;
00947
00948 # endif
00949
00950     //
00951     // コンストラクタ
00952     //
00953     Oculus();
00954
00955     //
00956     // デストラクタ
00957     //
00958     virtual ~Oculus() = default;
00959
00960 public:
00961
00962     // シングルトンなのでコピー・ムーブ禁止
00963     Oculus(const Oculus&) = delete;
00964     Oculus& operator=(const Oculus&) = delete;
00965     Oculus(Oculus&&) = delete;
00966     Oculus& operator=(Oculus&&) = delete;
00967
00973     static Oculus& initialize(const Window& window);
00974
00978     void terminate();
00979
00985     bool begin();
00986
00995     void select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation);
01002     void timewarp(const GgMatrix& projection);
01003
01009     void commit(int eye);
01010
01017     bool submit(bool mirror = true);
01018 };
01019 #endif
01020 };

```

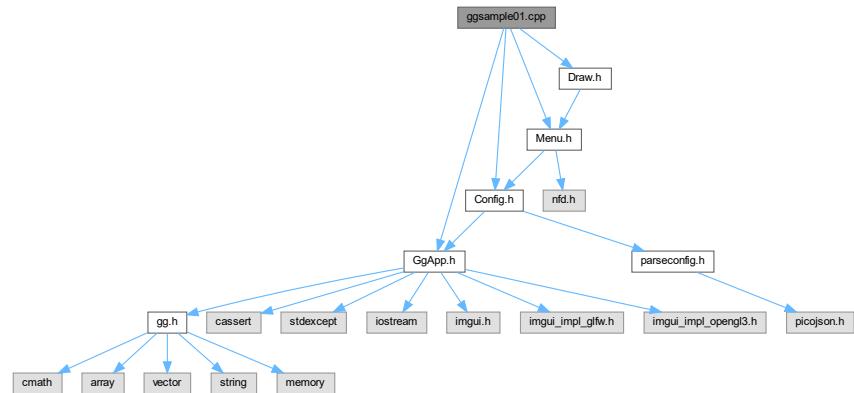
9.17 ggsample01.cpp ファイル

```

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

```

ggsample01.cpp の依存先関係図:



マクロ定義

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

9.17.1 マクロ定義詳解

9.17.1.1 CONFIG_FILE

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

`ggsample01.cpp` の 13 行目に定義があります。

9.17.1.2 PROJECT_NAME

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

`ggsample01.cpp` の 8 行目に定義があります。

9.18 ggsample01.cpp

[詳解]

```
00001 //
00002 // ゲームグラフィックス特論宿題アプリケーション
00003 //
00004 #include "GgApp.h"
00005
00006 // プロジェクト名
00007 #if !defined(PROJECT_NAME)
00008 # define PROJECT_NAME "ggsample01"
00009 #endif
00010
00011 // 構成ファイル名
00012 #if !defined(CONFIG_FILE)
00013 # define CONFIG_FILE PROJECT_NAME ".config.json"
00014 #endif
00015
00016 // 構成データ
00017 #include "Config.h"
00018
00019 // メニューの描画
00020 #include "Menu.h"
00021
00022 // 図形の描画
00023 #include "Draw.h"
00024
00025 //
00026 // アプリケーション本体
00027 //
00028 int GgApp::main(int argc, const char* const* argv)
00029 {
00030     // 設定を読み込む
00031     const Config config{ CONFIG_FILE };
00032
00033     // ウィンドウを作成する
00034     Window window{ argc > 1 ? argv[1] : PROJECT_NAME, config.getWidth(), config.getHeight() };
00035
00036     // メニューを初期化する
00037     Menu menu{ config };
```

```

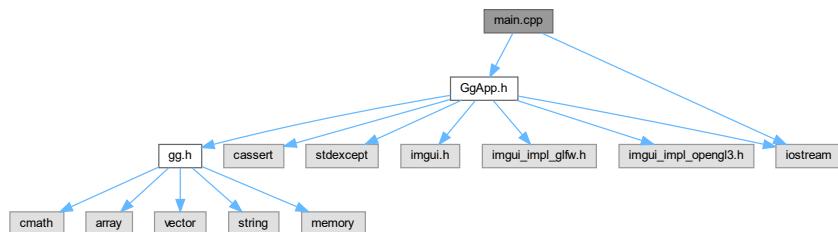
00038
00039 // 図形の描画の設定を行う
00040 Draw draw{ menu };
00041
00042 // ピュー変換行列を設定する
00043 const GgMatrix mv{ ggLookat(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f) };
00044
00045 // ウィンドウが開いている間繰り返す
00046 while (window)
00047 {
00048 // ウィンドウを消去する
00049 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00050
00051 // メニューを表示する
00052 menu.draw();
00053
00054 // オブジェクトの回転の変換行列を設定する
00055 const auto& mr{ window.getRotationMatrix(0) };
00056
00057 // 視点の平行移動の変換行列を設定する
00058 const auto& mt{ window.getTranslationMatrix(1) };
00059
00060 // 投影変換行列を設定する
00061 const GgMatrix&& mp{ ggPerspective(0.5f, window.getAspect(), 1.0f, 15.0f) };
00062
00063 // 図形を描画する
00064 draw.draw(mp, mt * mv * mr);
00065
00066 // カラーバッファを入れ替えてイベントを取り出す
00067 window.swapBuffers();
00068 }
00069
00070 return 0;
00071 }

```

9.19 main.cpp ファイル

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

main.cpp の依存先関係図:



マクロ定義

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

関数

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

9.19.1 マクロ定義詳解

9.19.1.1 HEADER_STR

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

main.cpp の 14 行目に定義がります。

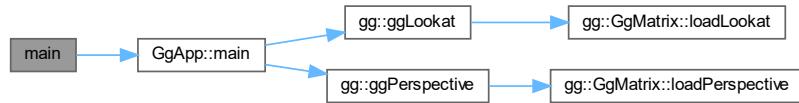
9.19.2 関数詳解

9.19.2.1 main()

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

main.cpp の 19 行目に定義がります。

呼び出し関係図:



9.20 main.cpp

[詳解]

```
00001 //
00002 // ゲームグラフィックス特論宿題アプリケーション
00003 //
00004 #include "GgApp.h"
00005
00006 // MessageBox の準備
00007 #if defined(_MSC_VER)
00008 # include <atistr.h>
00009 #elif defined(_APPLE__)
00010 # include <CoreFoundation/CoreFoundation.h>
00011 #else
00012 # include <iostream>
00013 #endif
00014 #define HEADER_STR "ゲームグラフィックス特論"
00015 //
00016 // メインプログラム
00017 //
00018 int main(int argc, const char* const* argv) try
00019 {
00020     // アプリケーションのオブジェクトを生成する
00021     #if defined(GL_GLES_PROTOTYPES)
00022         GgApp app(3, 1);
00023     #else
00024         GgApp app(4, 1);
00025     #endif
00026
00027     // アプリケーションを実行する
00028     return app.main(argc, argv);
00029 }
```

```

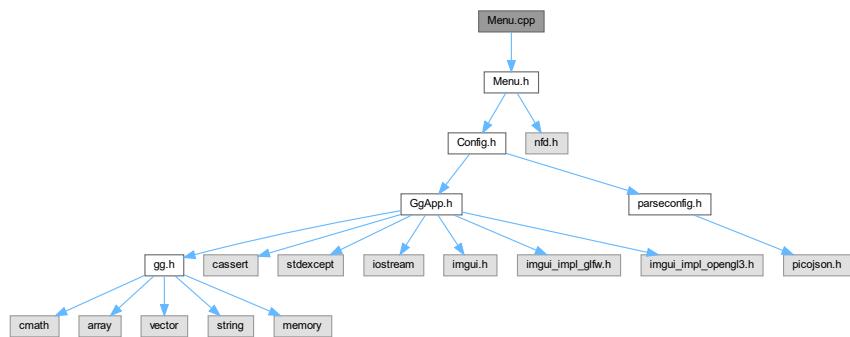
00030 }
00031 catch (const std::runtime_error &e)
00032 {
00033     // エラーメッセージを表示する
00034 #if defined(_MSC_VER)
00035     MessageBox(NULL, CString(e.what()), TEXT(H HEADER_STR), MB_ICONERROR);
00036 #elif defined(_APPLE_)
00037     // the following code is copied from
00038     // http://blog.jorgearimany.com/2010/05/messagebox-from-windows-to-mac.html
00039     CFStringRef msg_ref = CFStringCreateWithCString(NULL, e.what(), kCFStringEncodingUTF8);
00040
00041     // result code from the message box
00042     CFOptionFlags result;
00043
00044     // launch the message box
00045     CFUserNotificationDisplayAlert(
00046         0,                                     // no timeout
00047         kCFUserNotificationNoteAlertLevel, // change it depending message_type flags ( MB_ICONASTERISK.... etc.)
00048         NULL,                                // icon url, use default, you can change it depending
00049         NULL,                                // not used
00050         NULL,                                // localization of strings
00051         CFSTR(H HEADER_STR),                // header text
00052         msg_ref,                             // message text
00053         NULL,                                // default "ok" text in button
00054         NULL,                                // alternate button title
00055         NULL,                                // other button title, null--> no other button
00056         &result                                // response flags
00057     );
00058
00059     // Clean up the strings
00060     CFRelease(msg_ref);
00061 #else
00062     std::cerr << H HEADER_STR << ":" << e.what() << '\n';
00063 #endif
00064
00065     // プログラムを終了する
00066     return EXIT_FAILURE;
00067 }

```

9.21 Menu.cpp ファイル

#include "Menu.h"

Menu.cpp の依存先関係図:



9.21.1 詳解

メニューの描画クラスの実装

著者

Kohe Tokoi

日付

November 15, 2022

[Menu.cpp](#) に定義があります。

9.22 Menu.cpp

[詳解]

```

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

```

```

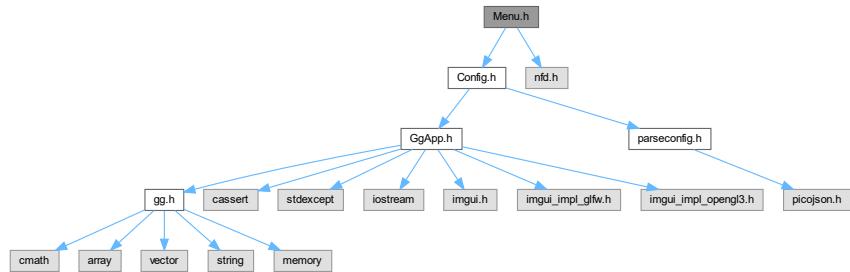
00069 }
00070
00071 // 描画する
00072 // メニューの表示位置を決定する
00073 //
00074 void Menu::draw()
00075 {
00076 #if defined(IMGUI_VERSION)
00077 //
00078 // ImGui によるユーザインターフェース
00079 //
00080 // ImGui のフレームを準備する
00081 ImGui::NewFrame();
00082
00083 // メニュー表示開始
00084 ImGui::Begin(u8"コントロールパネル");
00085
00086 // 光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

```

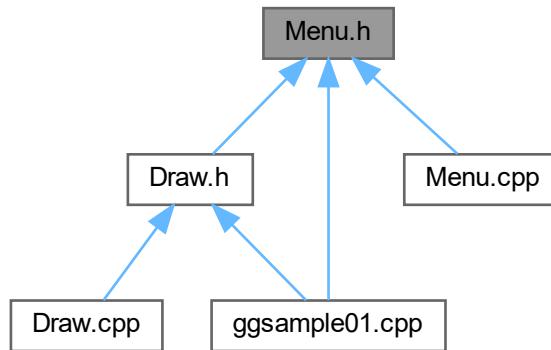
9.23 Menu.h ファイル

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

Menu.h の依存先関係図:



被依存関係図:



クラス

- class [Menu](#)

9.23.1 詳解

メニューの描画クラスの定義

著者

Kohe Tokoi

日付

November 15, 2022

[Menu.h](#) に定義があります。

9.24 Menu.h

[詳解]

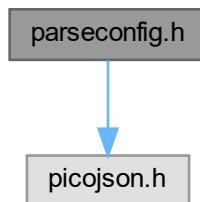
```

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

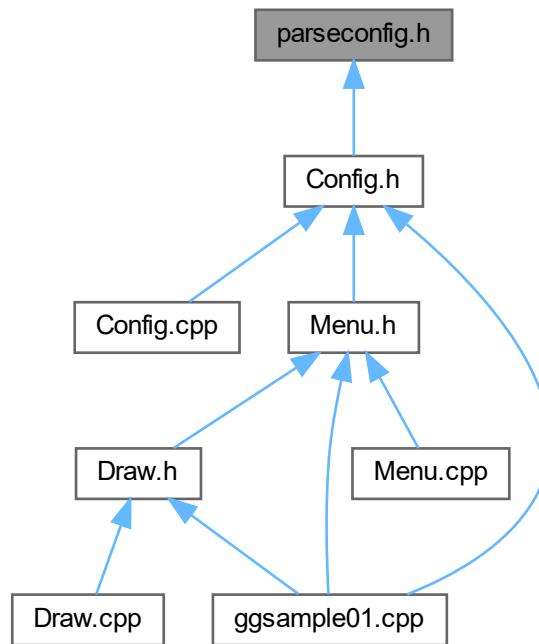
```
00023     friend class Draw;
00024
00025     // オリジナルの構成データ
00026     const Config& defaults;
00027
00028     // 構成データのコピー
00029     Config settings;
00030
00031     // CAD データ
00032     std::unique_ptr<const GgSimpleObj> model;
00033
00034     // 光源データ
00035     std::unique_ptr<const GgSimpleShader::LightBuffer>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

9.25 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)

9.25.1 詳解

構成ファイルの読み取り補助

著者

Kohe Tokoi

日付

November 15, 2022

[parseconfig.h](#) に定義があります。

9.25.2 関数詳解

9.25.2.1 `getString()` [1/3]

```
template<std::size_t U>
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトから文字列の配列を取得する

テンプレート引数

<i>U</i>	構成ファイルから取得する配列の要素の文字列の数
----------	-------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

[parseconfig.h](#) の 138 行目に定義があります。

9.25.2.2 `getString()` [2/3]

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

構成ファイルの JSON オブジェクトから文字列を取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>string</i>	取得した文字列を格納する変数

[parseconfig.h](#) の 114 行目に定義があります。

被呼び出し関係図:



9.25.2.3 getString() [3/3]

```
bool getString (
    const picojson::object & object,
    const std::string & key,
    std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトから文字列のベクトルを取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>strings</i>	取得した文字列の配列を格納する変数

[parseconfig.h](#) の 171 行目に定義があります。

9.25.2.4 getValue() [1/2]

```
template<typename T, std::size_t U>
bool getValue (
    const picojson::object & object,
    const std::string & key,
    std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトから数値の配列を取得する

テンプレート引数

<i>T</i>	構成ファイルから取得する配列の要素の数値のデータ型
<i>U</i>	構成ファイルから取得する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>vector</i>	取得した数値の配列を格納する変数

parseconfig.h の 48 行目に定義があります。

9.25.2.5 getValue() [2/2]

```
template<typename T >
bool getValue (
    const picojson::object & object,
    const std::string & key,
    T & scalar )
```

構成ファイルの JSON オブジェクトから数値を取得する

テンプレート引数

<i>T</i>	構成ファイルから取得する数値のデータ型
----------	---------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>scalar</i>	取得した JSON オブジェクトの数値を格納する変数

parseconfig.h の 23 行目に定義があります。

被呼び出し関係図:



9.25.2.6 getVector()

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

構成ファイルの JSON オブジェクトから 4 要素の数値のベクトルを取得する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	取得する JSON オブジェクトのキー
<i>vector</i>	取得した数値の配列を格納する変数

[parseconfig.h](#) の 81 行目に定義があります。

被呼び出し関係図:



9.25.2.7 setString() [1/3]

```
template<std::size_t U>
void setString (
    picojson::object & object,
    const std::string & key,
    const std::array< std::string, U > & strings )
```

構成ファイルの JSON オブジェクトに文字列の配列を設定する

テンプレート引数

<i>U</i>	構成ファイルに設定する配列の要素の文字列の数
----------	------------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

parseconfig.h の 283 行目に定義があります。

9.25.2.8 setString() [2/3]

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

構成ファイルの JSON オブジェクトに文字列を設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>string</i>	設定する文字列

parseconfig.h の 268 行目に定義があります。

被呼び出し関係図:



9.25.2.9 setString() [3/3]

```
void setString (
    picojson::object & object,
    const std::string & key,
    const std::vector< std::string > & strings ) [inline]
```

構成ファイルの JSON オブジェクトに文字列のベクトルを設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>strings</i>	設定する文字列の配列

[parseconfig.h](#) の 308 行目に定義があります。

9.25.2.10 setValue() [1/2]

```
template<typename T, std::size_t U>
void setValue (
    picojson::object & object,
    const std::string & key,
    const std::array< T, U > & vector )
```

構成ファイルの JSON オブジェクトに数値の配列を設定する

テンプレート引数

<i>T</i>	構成ファイルに設定する配列の要素の数値のデータ型
<i>U</i>	構成ファイルに設定する配列の要素の数値の数

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>vector</i>	設定する数値の配列

[parseconfig.h](#) の 218 行目に定義があります。

9.25.2.11 setValue() [2/2]

```
template<typename T>
void setValue (
    picojson::object & object,
    const std::string & key,
    const T & scalar )
```

構成ファイルの JSON オブジェクトに数値を設定する

テンプレート引数

<i>T</i>	構成ファイルに設定する数値のデータ型
----------	--------------------

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>scalar</i>	設定する数値

parseconfig.h の 202 行目に定義があります。

被呼び出し関係図:



9.25.2.12 setVector()

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

構成ファイルの JSON オブジェクトに 4 要素の数値のベクトルを設定する

引数

<i>object</i>	構成ファイルの JSON オブジェクト
<i>key</i>	設定する JSON オブジェクトのキー
<i>vector</i>	設定する数値の配列

parseconfig.h の 243 行目に定義があります。

被呼び出し関係図:



9.26 parseconfig.h

[詳解]

```

00001 #pragma once
00002
00010
00011 // JSON
00012 #include "picojson.h"
00013
00022 template <typename T>
00023 bool getValue(const picojson::object& object,
00024     const std::string& key, T& scalar)
00025 {
00026     // key に一致するオブジェクトを探す
00027     const auto&& value{ object.find(key) };
00028
00029     // オブジェクトが無いか数値でなかったら戻る
00030     if (value == object.end() || !value->second.is<double>()) return false;
00031
00032     // 数値として格納する
00033     scalar = static_cast<T>(value->second.get<double>());
00034
00035     return true;
00036 }
00037
00047 template <typename T, std::size_t U>
00048 bool getValue(const picojson::object& object,
00049     const std::string& key, std::array<T, U>& vector)
00050 {
00051     // key に一致するオブジェクトを探す
00052     const auto&& value{ object.find(key) };
00053
00054     // オブジェクトが無いか配列でなかったら戻る
00055     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00056
00057     // 配列を取り出す
00058     const auto& array{ value->second.get<picojson::array>() };
00059
00060     // 配列の要素数とデータの格納先の要素数の少ない方の数
00061     const auto n{ std::min(U, array.size()) };
00062
00063     // 配列の要素について
00064     for (std::size_t i = 0; i < n; ++i)
00065     {
00066         // 要素が数値なら格納する
00067         if (array[i].is<double>()) vector[i] = static_cast<T>(array[i].get<double>());
00068     }
00069
00070     return true;
00071 }
00072
00080 inline
00081 bool getVector(const picojson::object& object,
00082     const std::string& key, GgVector& vector)
00083 {
00084     // key に一致するオブジェクトを探す
00085     const auto&& value{ object.find(key) };
00086
00087     // オブジェクトが無いか配列でなかったら戻る
00088     if (value == object.end() || !value->second.is<picojson::array>()) return false;
00089
00090     // 配列を取り出す
00091     const auto& array{ value->second.get<picojson::array>() };
00092
00093     // 配列の要素数とデータの格納先の要素数の少ない方の数
00094     const auto n{ std::min(sizeof(GgVector) / sizeof(GLfloat), array.size()) };
00095
00096     // 配列の要素について
00097     for (std::size_t i = 0; i < n; ++i)
00098     {
00099         // 要素が数値なら格納する
00100         if (array[i].is<double>()) vector[i] = static_cast<GLfloat>(array[i].get<double>());
00101     }
00102
00103     return true;
00104 }
00105
00113 inline
00114 bool getString(const picojson::object& object,
00115     const std::string& key, std::string& string)
00116 {
00117     // key に一致するオブジェクトを探す
00118     const auto&& value{ object.find(key) };
00119
00120     // オブジェクトが無いか文字列でなかったら戻る

```

```
00121 if (value == object.end() || !value->second.is<std::string>()) return false;
00122 // 文字列として格納する
00123 string = value->second.get<std::string>();
00124
00125 return true;
00126 }
00127 }
00128
00129 template <std::size_t U>
00130 bool getString(const picojson::object& object,
00131 const std::string& key, std::array<std::string, U>& strings)
00132 {
00133 // key に一致するオブジェクトを探す
00134 const auto& value{ object.find(key) };
00135
00136 // オブジェクトが無いか配列でなかったら戻る
00137 if (value == object.end() || !value->second.is<picojson::array>()) return false;
00138
00139 // 配列を取り出す
00140 const auto& array{ value->second.get<picojson::array>() };
00141
00142 // 配列の要素数とデータの格納先の要素数の少ない方の数
00143 const auto n{ std::min(U, array.size()) };
00144
00145 // 配列の要素について
00146 for (std::size_t i = 0; i < n; ++i)
00147 {
00148 // 要素が文字列なら文字列として格納する
00149 strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
00150 }
00151
00152 // 配列の要素について
00153 for (std::size_t i = 0; i < n; ++i)
00154 {
00155 // 要素が文字列なら文字列として格納する
00156 strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
00157 }
00158
00159 return true;
00160 }
00161 }
00162
00163 inline
00164 bool getString(const picojson::object& object,
00165 const std::string& key, std::vector<std::string>& strings)
00166 {
00167 // key に一致するオブジェクトを探す
00168 const auto& value{ object.find(key) };
00169
00170 // オブジェクトが無いか配列でなかったら戻る
00171 if (value == object.end() || !value->second.is<picojson::array>()) return false;
00172
00173 // 配列を取り出す
00174 const auto& array{ value->second.get<picojson::array>() };
00175
00176 // 配列のすべての要素について
00177 for (auto& element : array)
00178 {
00179 // 要素が文字列なら文字列として格納する
00180 strings.emplace_back(element.is<std::string>() ? element.get<std::string>() : "");
00181 }
00182
00183 return true;
00184 }
00185 }
00186
00187 template <typename T>
00188 void setValue(picojson::object& object,
00189 const std::string& key, const T& scalar)
00190 {
00191 object.emplace(key, picojson::value(static_cast<double>(scalar)));
00192 }
00193
00194 template <typename T, std::size_t U>
00195 void setValue(picojson::object& object,
00196 const std::string& key, const std::array<T, U>& vector)
00197 {
00198 // picojson の配列
00199 picojson::array array;
00200
00201 // 配列のすべての要素について
00202 for (const auto& element : vector)
00203 {
00204 // 要素を picojson::array に追加する
00205 array.emplace_back(picojson::value(static_cast<double>(element)));
00206 }
00207
00208 // オブジェクトに追加する
00209 object.emplace(key, array);
00210 }
00211
00212 inline
00213 void setVector(picojson::object& object,
00214 const std::string& key, const GgVector& vector)
00215 {
00216 // picojson の配列
```

```
00247 picojson::array array;
00248 // ベクトルのすべての要素について
00249 for (const auto& element : vector)
00250 {
00251     // 要素を picojson::array に追加する
00252     array.emplace_back(picojson::value(static_cast<double>(element)));
00253 }
00254
00255 // オブジェクトに追加する
00256 object.emplace(key, array);
00257 }
00258 }
00259
00260 inline
00261 void setString(picojson::object& object,
00262     const std::string& key, const std::string& string)
00263 {
00264     object.emplace(key, picojson::value(string));
00265 }
00266
00267 template <std::size_t U>
00268 void setString(picojson::object& object,
00269     const std::string& key, const std::array<std::string, U>& strings)
00270 {
00271     // picojson の配列
00272     picojson::array array;
00273
00274     // 配列のすべての要素について
00275     for (const auto& string : strings)
00276     {
00277         // 要素を picojson::array に追加する
00278         array.emplace_back(picojson::value(string));
00279     }
00280
00281     // オブジェクトに追加する
00282     object.emplace(key, array);
00283 }
00284
00285
00286 inline
00287 void setString(picojson::object& object,
00288     const std::string& key, const std::vector<std::string>& strings)
00289 {
00290     // picojson の配列
00291     picojson::array array;
00292
00293     // 配列のすべての要素について
00294     for (auto& string : strings)
00295     {
00296         // 要素を picojson::array に追加する
00297         array.emplace_back(picojson::value(string));
00298     }
00299
00300     // オブジェクトに追加する
00301     object.emplace(key, array);
00302 }
00303
00304
00305 inline
00306 void setString(picojson::object& object,
00307     const std::string& key, const std::vector<std::string>& strings)
00308 {
00309     // picojson の配列
00310     picojson::array array;
00311
00312     // 配列のすべての要素について
00313     for (auto& string : strings)
00314     {
00315         // 要素を picojson::array に追加する
00316         array.emplace_back(picojson::value(string));
00317     }
00318
00319     // オブジェクトに追加する
00320     object.emplace(key, array);
00321 }
00322
00323 }
```

9.27 README.md ファイル

Index

-ggError
 gg, 18
-ggFBOError
 gg, 19
~Config
 Config, 88
~Draw
 Draw, 91
~GgApp
 GgApp, 93
~GgBuffer
 gg::GgBuffer< T >, 96
~GgColorTexture
 gg::GgColorTexture, 102
~GgElements
 gg::GgElements, 107
~GgNormalTexture
 gg::GgNormalTexture, 167
~GgPointShader
 gg::GgPointShader, 175
~GgPoints
 gg::GgPoints, 170
~GgShader
 gg::GgShader, 250
~GgShape
 gg::GgShape, 252
~GgSimpleObj
 gg::GgSimpleObj, 256
~GgSimpleShader
 gg::GgSimpleShader, 260
~GgTexture
 gg::GgTexture, 274
~GgTrackball
 gg::GgTrackball, 279
~GgTriangles
 gg::GgTriangles, 289
~GgUniformBuffer
 gg::GgUniformBuffer< T >, 293
~LightBuffer
 gg::GgSimpleShader::LightBuffer, 321
~MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 332
~Menu
 Menu, 339
~Window
 GgApp::Window, 342
add
 gg::GgQuaternion, 186–188
ambient
 gg::GgSimpleShader::Light, 317
 gg::GgSimpleShader::Material, 329
begin
 gg::GgTrackball, 279
bind
 gg::GgBuffer< T >, 96
 gg::GgTexture, 274
 gg::GgUniformBuffer< T >, 293
BindingPoints
 gg, 18
Config, 87
 ~Config, 88
 Config, 87, 88
 getHeight, 89
 getWidth, 89
 load, 89
 Menu, 91
 save, 90
Config.cpp, 369
 defaultLight, 370
Config.h, 372
 pathChar, 374
 pathString, 374
 TCharToUtf8, 374
 Utf8ToTChar, 374
CONFIGFILE
 ggsample01.cpp, 536
conjugate
 gg::GgQuaternion, 189
copy
 gg::GgBuffer< T >, 96
 gg::GgUniformBuffer< T >, 294
defaultLight
 Config.cpp, 370
diffuse
 gg::GgSimpleShader::Light, 318
 gg::GgSimpleShader::Material, 329
distance3
 gg::GgVector, 302
distance4
 gg::GgVector, 303
divide
 gg::GgQuaternion, 190–192
dot3
 gg::GgVector, 304
dot4
 gg::GgVector, 304

Draw, 91
 ~Draw, 91
 Draw, 91
 draw, 92
 Menu, 339
 draw
 Draw, 92
 gg::GgElements, 107
 gg::GgPoints, 171
 gg::GgShape, 253
 gg::GgSimpleObj, 256
 gg::GgTriangles, 289
 Menu, 339
 Draw.cpp, 376
 Draw.h, 377
 end
 gg::GgTrackball, 280
 euler
 gg::GgQuaternion, 192, 193
 fill
 gg::GgUniformBuffer< T >, 294
 frustum
 gg::GgMatrix, 114
 get
 gg::GgMatrix, 115, 116
 gg::GgPointShader, 175
 gg::GgQuaternion, 194
 gg::GgShader, 250
 gg::GgShape, 253
 gg::GgSimpleObj, 257
 gg::GgTrackball, 280
 GgApp::Window, 342
 getAltArrowX
 GgApp::Window, 343
 getAltArrowY
 GgApp::Window, 343
 getAltArrow
 GgApp::Window, 344
 getArrow
 GgApp::Window, 345
 getArrowX
 GgApp::Window, 346
 getArrowY
 GgApp::Window, 347
 getAspect
 GgApp::Window, 348
 getBuffer
 gg::GgBuffer< T >, 97
 gg::GgPoints, 171
 gg::GgTriangles, 290
 gg::GgUniformBuffer< T >, 295
 getConjugateMatrix
 gg::GgQuaternion, 194–196
 getControlArrow
 GgApp::Window, 348
 getControlArrowX
 GgApp::Window, 349
 getControlArrowY
 GgApp::Window, 349
 getCount
 gg::GgBuffer< T >, 97
 gg::GgPoints, 171
 gg::GgTriangles, 290
 gg::GgUniformBuffer< T >, 295
 getFboHeight
 GgApp::Window, 350
 getFboSize
 GgApp::Window, 351
 getFboWidth
 GgApp::Window, 352
 getHeight
 Config, 89
 gg::GgTexture, 274
 GgApp::Window, 352
 getIndexBuffer
 gg::GgElements, 108
 getIndexCount
 gg::GgElements, 108
 getKey
 GgApp::Window, 353
 getLastKey
 GgApp::Window, 353
 getMatrix
 gg::GgQuaternion, 196–198
 gg::GgTrackball, 281
 getMode
 gg::GgShape, 254
 getMouse
 GgApp::Window, 353, 354
 getMouseX
 GgApp::Window, 354
 getMouseY
 GgApp::Window, 355
 getQuaternion
 gg::GgTrackball, 281
 getRotation
 GgApp::Window, 355
 getRotationMatrix
 GgApp::Window, 355
 getScale
 gg::GgTrackball, 281, 282
 getScrollMatrix
 GgApp::Window, 356
 getShiftArrow
 GgApp::Window, 356
 getShiftArrowX
 GgApp::Window, 357
 getShiftArrowY
 GgApp::Window, 357
 getSize
 gg::GgTexture, 274, 275
 GgApp::Window, 358
 getStart
 gg::GgTrackball, 282, 283

getStride
 gg::GgBuffer< T >, 97
 gg::GgUniformBuffer< T >, 295

getString
 parseconfig.h, 545, 546

getTarget
 gg::GgBuffer< T >, 97
 gg::GgUniformBuffer< T >, 295

getTexture
 gg::GgTexture, 275

getTranslation
 GgApp::Window, 359

getTranslationMatrix
 GgApp::Window, 359

getUserPointer
 GgApp::Window, 360

getValue
 parseconfig.h, 546, 547

getVector
 parseconfig.h, 548

getWheel
 GgApp::Window, 360, 361

getWheelX
 GgApp::Window, 361

getWheelY
 GgApp::Window, 361

getWidth
 Config, 89
 gg::GgTexture, 276
 GgApp::Window, 361

gg, 15
 _ggError, 18
 _ggFBOError, 19
 BindingPoints, 18
 ggArraysObj, 19
 ggBufferAlignment, 86
 ggConjugate, 20
 ggCreateComputeShader, 20
 ggCreateNormalMap, 21
 ggCreateShader, 22
 ggCross, 23, 24
 ggDistance3, 24, 25
 ggDistance4, 26
 ggDot3, 28, 29
 ggDot4, 30
 ggElementsMesh, 31
 ggElementsObj, 32
 ggElementsSphere, 33
 ggEllipse, 33
 ggEulerQuaternion, 34
 ggFrustum, 35
 ggIdentity, 36
 ggIdentityQuaternion, 37
 ggInit, 37
 ggInvert, 38, 39
 ggLength3, 39, 40
 ggLength4, 41
 ggLoadComputeShader, 42

 ggLoadHeight, 43
 ggLoadImage, 44
 ggLoadShader, 45
 ggLoadSimpleObj, 46, 47
 ggLoadTexture, 47
 ggLookat, 48–50
 ggMatrixQuaternion, 51, 52
 ggNorm, 52
 ggNormal, 53
 ggNormalize, 54
 ggNormalize3, 54–56
 ggNormalize4, 57–59
 ggOrthogonal, 59
 ggPerspective, 60
 ggPointsCube, 61
 ggPointsSphere, 62
 ggQuaternion, 62, 63
 ggQuaternionMatrix, 64
 ggQuaternionTransposeMatrix, 64
 ggReadImage, 65
 ggRectangle, 66
 ggRotate, 66–69
 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, 378
 gg.h, 453
 ggError, 457
 ggFBOError, 458

 gg::GgBuffer< T >, 95
 ~GgBuffer, 96
 bind, 96
 copy, 96
 getBuffer, 97
 getCount, 97
 getStride, 97
 getTarget, 97
 GgBuffer, 95, 96
 map, 98
 operator=, 99
 read, 99
 send, 99
 unbind, 100
 unmap, 100

gg::GgColorTexture, 100
 ~GgColorTexture, 102
 GgColorTexture, 101, 102
 load, 103, 104
 gg::GgElements, 105
 ~GgElements, 107
 draw, 107
 getIndexBuffer, 108
 getIndexCount, 108
 GgElements, 106
 load, 108
 send, 109
 gg::GgMatrix, 109
 frustum, 114
 get, 115, 116
 GgMatrix, 112, 113
 invert, 117
 loadFrustum, 117
 loadIdentity, 118
 loadInvert, 118, 119
 loadLookat, 120, 121
 loadNormal, 122, 123
 loadOrthogonal, 123
 loadPerspective, 124
 loadRotate, 125–128
 loadRotateX, 128
 loadRotateY, 129
 loadRotateZ, 130
 loadScale, 130–132
 loadTranslate, 132–134
 loadTranspose, 134, 135
 lookat, 136, 137
 normal, 138
 operator*, 139, 140
 operator*+, 141, 142
 operator+, 143
 operator+=, 144, 145
 operator-, 145, 146
 operator-=, 146, 147
 operator/, 148
 operator/=, 149
 operator=, 150
 orthogonal, 151
 perspective, 152
 projection, 152–154
 rotate, 154–156
 rotateX, 157
 rotateY, 158
 rotateZ, 159
 scale, 159–161
 translate, 162, 163
 transpose, 164
 gg::GgNormalTexture, 165
 ~GgNormalTexture, 167
 GgNormalTexture, 165, 166
 load, 167, 168
 gg::GgPoints, 169
 ~GgPoints, 170

draw, 171
 getBuffer, 171
 getCount, 171
 GgPoints, 170
 load, 172
 send, 172
 gg::GgPointShader, 173
 ~GgPointShader, 175
 get, 175
 GgPointShader, 174
 load, 175, 176
 loadMatrix, 177
 loadModelviewMatrix, 178
 loadProjectionMatrix, 178, 179
 unuse, 179
 use, 179–181
 gg::GgQuaternion, 182
 add, 186–188
 conjugate, 189
 divide, 190–192
 euler, 192, 193
 get, 194
 getConjugateMatrix, 194–196
 getMatrix, 196–198
 GgQuaternion, 185, 186
 invert, 198
 load, 199, 200
 loadAdd, 201–203
 loadConjugate, 204
 loadDivide, 205–207
 loadEuler, 208, 209
 loadIdentity, 209
 loadInvert, 210, 211
 loadMatrix, 212
 loadMultiply, 213–215
 loadNormalize, 216
 loadRotate, 217, 218
 loadRotateX, 219
 loadRotateY, 220
 loadRotateZ, 220
 loadSlerp, 221–223
 loadSubtract, 224–226
 multiply, 226–228
 norm, 228
 normalize, 229
 operator*, 229, 230
 operator*+, 230, 231
 operator+, 232
 operator+=, 233, 234
 operator-, 234, 235
 operator-=, 235, 236
 operator/, 237
 operator/=, 238, 239
 operator=, 239, 240
 rotate, 240, 241
 rotateX, 242
 rotateY, 243
 rotateZ, 244

slerp, 244, 245
 subtract, 245–247

gg::GgShader, 248
 ~GgShader, 250
 get, 250
 GgShader, 249, 250
 operator=, 250
 unuse, 250
 use, 251

gg::GgShape, 251
 ~GgShape, 252
 draw, 253
 get, 253
 getMode, 254
 GgShape, 252
 operator=, 254
 setMode, 254

gg::GgSimpleObj, 255
 ~GgSimpleObj, 256
 draw, 256
 get, 257
 GgSimpleObj, 255

gg::GgSimpleShader, 258
 ~GgSimpleShader, 260
 GgSimpleShader, 259, 260
 load, 261
 loadMatrix, 262, 263
 loadModelviewMatrix, 264, 265
 operator=, 266
 use, 266–272

gg::GgSimpleShader::Light, 317
 ambient, 317
 diffuse, 318
 position, 318
 specular, 318

gg::GgSimpleShader::LightBuffer, 319
 ~LightBuffer, 321
 LightBuffer, 320
 load, 321
 loadAmbient, 322, 323
 loadColor, 323
 loadDiffuse, 323, 324
 loadPosition, 325, 326
 loadSpecular, 326, 327
 select, 328

gg::GgSimpleShader::Material, 328
 ambient, 329
 diffuse, 329
 shininess, 330
 specular, 330

gg::GgSimpleShader::MaterialBuffer, 330
 ~MaterialBuffer, 332
 load, 332, 333
 loadAmbient, 333
 loadAmbientAndDiffuse, 334
 loadDiffuse, 335
 loadShininess, 336
 loadSpecular, 337

 MaterialBuffer, 331, 332
 select, 337

gg::GgTexture, 272
 ~GgTexture, 274
 bind, 274
 getHeight, 274
 getSize, 274, 275
 getTexture, 275
 getWidth, 276
 GgTexture, 273, 274
 operator=, 276
 swapRandB, 276
 unbind, 277

gg::GgTrackball, 277
 ~GgTrackball, 279
 begin, 279
 end, 280
 get, 280
 getMatrix, 281
 getQuaternion, 281
 getScale, 281, 282
 getStart, 282, 283
 GgTrackball, 279
 motion, 283
 operator=, 284
 region, 284, 285
 reset, 286
 rotate, 286

gg::GgTriangles, 287
 ~GgTriangles, 289
 draw, 289
 getBuffer, 290
 getCount, 290
 GgTriangles, 288
 load, 290
 send, 291

gg::GgUniformBuffer< T >, 291
 ~GgUniformBuffer, 293
 bind, 293
 copy, 294
 fill, 294
 getBuffer, 295
 getCount, 295
 getStride, 295
 getTarget, 295
 GgUniformBuffer, 292, 293
 load, 296
 map, 297
 read, 297
 send, 298
 unbind, 298
 unmap, 298

gg::GgVector, 299
 distance3, 302
 distance4, 303
 dot3, 304
 dot4, 304
 GgVector, 300, 302

length3, 305
 length4, 305
 normalize3, 306
 normalize4, 306
 operator*, 307
 operator*=₃₀₈
 operator+, 309
 operator+=, 309, 310
 operator-, 310, 311
 operator-=, 311
 operator/, 312
 operator/=, 313
 gg::GgVertex, 313
 GgVertex, 314–316
 normal, 316
 position, 316
 GG_BUTTON_COUNT
 GgApp.h, 527
 GG_INTERFACE_COUNT
 GgApp.h, 527
 GG_USE_IMGUI
 GgApp.h, 527
 GgApp, 92
 ~GgApp, 93
 GgApp, 93
 main, 94
 operator=, 94
 GgApp.cpp, 512
 GgApp.h, 526
 GG_BUTTON_COUNT, 527
 GG_INTERFACE_COUNT, 527
 GG_USE_IMGUI, 527
 GgApp::Window, 340
 ~Window, 342
 get, 342
 getAltArrowX, 343
 getAltArrowY, 343
 getAltArrow, 344
 getArrow, 345
 getArrowX, 346
 getArrowY, 347
 getAspect, 348
 getControlArrow, 348
 getControlArrowX, 349
 getControlArrowY, 349
 getFboHeight, 350
 getFboSize, 351
 getFboWidth, 352
 getHeight, 352
 getKey, 353
 getLastKey, 353
 getMouse, 353, 354
 getMouseX, 354
 getMouseY, 355
 getRotation, 355
 getRotationMatrix, 355
 getScrollMatrix, 356
 getShiftArrow, 356
 getShiftArrowX, 357
 getShiftArrowY, 357
 getSize, 358
 getTranslation, 359
 getTranslationMatrix, 359
 getUserPointer, 360
 getWheel, 360, 361
 getWheelX, 361
 getWheelY, 361
 getWidth, 361
 operator bool, 362
 operator=, 362
 reset, 362
 resetRotation, 363
 resetTranslation, 363
 restoreViewport, 364
 selectInterface, 364
 setClose, 364
 setKeyboardFunc, 366
 setMouseFunc, 366
 setResizeFunc, 366
 setUserPointer, 367
 setVelocity, 367
 setWheelFunc, 367
 shouldClose, 368
 swapBuffers, 368
 updateViewport, 368
 Window, 341, 342
 ggArraysObj
 gg, 19
 GgBuffer
 gg::GgBuffer< T >, 95, 96
 ggBufferAlignment
 gg, 86
 GgColorTexture
 gg::GgColorTexture, 101, 102
 ggConjugate
 gg, 20
 ggCreateComputeShader
 gg, 20
 ggCreateNormalMap
 gg, 21
 ggCreateShader
 gg, 22
 ggCross
 gg, 23, 24
 ggDistance3
 gg, 24, 25
 ggDistance4
 gg, 26
 ggDot3
 gg, 28, 29
 ggDot4
 gg, 30
 GgElements
 gg::GgElements, 106
 ggElementsMesh
 gg, 31

ggElementsObj
 gg, 32
ggElementsSphere
 gg, 33
ggEllipse
 gg, 33
ggError
 gg.h, 457
ggEulerQuaternion
 gg, 34
ggFBOError
 gg.h, 458
ggFrustum
 gg, 35
ggIdentity
 gg, 36
ggIdentityQuaternion
 gg, 37
ggInit
 gg, 37
ggInvert
 gg, 38, 39
ggLength3
 gg, 39, 40
ggLength4
 gg, 41
ggLoadComputeShader
 gg, 42
ggLoadHeight
 gg, 43
ggLoadImage
 gg, 44
ggLoadShader
 gg, 45
ggLoadSimpleObj
 gg, 46, 47
ggLoadTexture
 gg, 47
ggLookat
 gg, 48–50
GgMatrix
 gg::GgMatrix, 112, 113
ggMatrixQuaternion
 gg, 51, 52
ggNorm
 gg, 52
ggNormal
 gg, 53
ggNormalize
 gg, 54
ggNormalize3
 gg, 54–56
ggNormalize4
 gg, 57–59
GgNormalTexture
 gg::GgNormalTexture, 165, 166
ggOrthogonal
 gg, 59
ggPerspective
 gg, 60
GgPoints
 gg::GgPoints, 170
ggPointsCube
 gg, 61
GgPointShader
 gg::GgPointShader, 174
ggPointsSphere
 gg, 62
GgQuaternion
 gg::GgQuaternion, 185, 186
ggQuaternion
 gg, 62, 63
ggQuaternionMatrix
 gg, 64
ggQuaternionTransposeMatrix
 gg, 64
ggReadImage
 gg, 65
ggRectangle
 gg, 66
ggRotate
 gg, 66–69
ggRotateQuaternion
 gg, 70, 71
ggRotateX
 gg, 72
ggRotateY
 gg, 73
ggRotateZ
 gg, 73
ggsample01.cpp, 535
 CONFIG_FILE, 536
 PROJECT_NAME, 536
ggSaveColor
 gg, 74
ggSaveDepth
 gg, 75
ggSaveTga
 gg, 75
ggScale
 gg, 76, 77
GgShader
 gg::GgShader, 249, 250
GgShape
 gg::GgShape, 252
GgSimpleObj
 gg::GgSimpleObj, 255
GgSimpleShader
 gg::GgSimpleShader, 259, 260
ggSlerp
 gg, 78–80
GgTexture
 gg::GgTexture, 273, 274
GgTrackball
 gg::GgTrackball, 279
ggTranslate

gg, 81, 82
 ggTranspose
 gg, 83
 GgTriangles
 gg::GgTriangles, 288
 GgUniformBuffer
 gg::GgUniformBuffer< T >, 292, 293
 GgVector
 gg::GgVector, 300, 302
 GgVertex
 gg::GgVertex, 314–316

HEADER_STR
 main.cpp, 537

invert
 gg::GgMatrix, 117
 gg::GgQuaternion, 198

length3
 gg::GgVector, 305
 length4
 gg::GgVector, 305

LightBindingPoint
 gg, 18
 LightBuffer
 gg::GgSimpleShader::LightBuffer, 320

load
 Config, 89
 gg::GgColorTexture, 103, 104
 gg::GgElements, 108
 gg::GgNormalTexture, 167, 168
 gg::GgPoints, 172
 gg::GgPointShader, 175, 176
 gg::GgQuaternion, 199, 200
 gg::GgSimpleShader, 261
 gg::GgSimpleShader::LightBuffer, 321
 gg::GgSimpleShader::MaterialBuffer, 332, 333
 gg::GgTriangles, 290
 gg::GgUniformBuffer< T >, 296

loadAdd
 gg::GgQuaternion, 201–203

loadAmbient
 gg::GgSimpleShader::LightBuffer, 322, 323
 gg::GgSimpleShader::MaterialBuffer, 333

loadAmbientAndDiffuse
 gg::GgSimpleShader::MaterialBuffer, 334

loadColor
 gg::GgSimpleShader::LightBuffer, 323

loadConjugate
 gg::GgQuaternion, 204

loadDiffuse
 gg::GgSimpleShader::LightBuffer, 323, 324
 gg::GgSimpleShader::MaterialBuffer, 335

loadDivide
 gg::GgQuaternion, 205–207

loadEuler
 gg::GgQuaternion, 208, 209

loadFrustum

 gg::GgMatrix, 117
 loadIdentity
 gg::GgMatrix, 118
 gg::GgQuaternion, 209

loadInvert
 gg::GgMatrix, 118, 119
 gg::GgQuaternion, 210, 211

loadLookat
 gg::GgMatrix, 120, 121

loadMatrix
 gg::GgPointShader, 177
 gg::GgQuaternion, 212
 gg::GgSimpleShader, 262, 263

loadModelviewMatrix
 gg::GgPointShader, 178
 gg::GgSimpleShader, 264, 265

loadMultiply
 gg::GgQuaternion, 213–215

loadNormal
 gg::GgMatrix, 122, 123

loadNormalize
 gg::GgQuaternion, 216

loadOrthogonal
 gg::GgMatrix, 123

loadPerspective
 gg::GgMatrix, 124

loadPosition
 gg::GgSimpleShader::LightBuffer, 325, 326

loadProjectionMatrix
 gg::GgPointShader, 178, 179

loadRotate
 gg::GgMatrix, 125–128
 gg::GgQuaternion, 217, 218

loadRotateX
 gg::GgMatrix, 128
 gg::GgQuaternion, 219

loadRotateY
 gg::GgMatrix, 129
 gg::GgQuaternion, 220

loadRotateZ
 gg::GgMatrix, 130
 gg::GgQuaternion, 220

loadScale
 gg::GgMatrix, 130–132

loadShininess
 gg::GgSimpleShader::MaterialBuffer, 336

loadSlerp
 gg::GgQuaternion, 221–223

loadSpecular
 gg::GgSimpleShader::LightBuffer, 326, 327
 gg::GgSimpleShader::MaterialBuffer, 337

loadSubtract
 gg::GgQuaternion, 224–226

loadTranslate
 gg::GgMatrix, 132–134

loadTranspose
 gg::GgMatrix, 134, 135

lookat

gg::GgMatrix, 136, 137
main
 GgApp, 94
 main.cpp, 538
main.cpp, 537
 HEADER_STR, 537
 main, 538
map
 gg::GgBuffer< T >, 98
 gg::GgUniformBuffer< T >, 297
MaterialBindingPoint
 gg, 18
MaterialBuffer
 gg::GgSimpleShader::MaterialBuffer, 331, 332
Menu, 338
 ~Menu, 339
 Config, 91
 Draw, 339
 draw, 339
 Menu, 338, 339
 operator=, 339
Menu.cpp, 539
Menu.h, 541
motion
 gg::GgTrackball, 283
multiply
 gg::GgQuaternion, 226–228
norm
 gg::GgQuaternion, 228
normal
 gg::GgMatrix, 138
 gg::GgVertex, 316
normalize
 gg::GgQuaternion, 229
normalize3
 gg::GgVector, 306
normalize4
 gg::GgVector, 306
operator bool
 GgApp::Window, 362
operator*
 gg, 83
 gg::GgMatrix, 139, 140
 gg::GgQuaternion, 229, 230
 gg::GgVector, 307
operator*=
 gg::GgMatrix, 141, 142
 gg::GgQuaternion, 230, 231
 gg::GgVector, 308
operator+
 gg, 84
 gg::GgMatrix, 143
 gg::GgQuaternion, 232
 gg::GgVector, 309
operator+=
 gg::GgMatrix, 144, 145
 gg::GgQuaternion, 233, 234
 gg::GgVector, 309, 310
operator-
 gg, 85
 gg::GgMatrix, 145, 146
 gg::GgQuaternion, 234, 235
 gg::GgVector, 310, 311
operator-=
 gg::GgMatrix, 146, 147
 gg::GgQuaternion, 235, 236
 gg::GgVector, 311
operator/
 gg, 85
 gg::GgMatrix, 148
 gg::GgQuaternion, 237
 gg::GgVector, 312
operator/=
 gg::GgMatrix, 149
 gg::GgQuaternion, 238, 239
 gg::GgVector, 313
operator=
 gg::GgBuffer< T >, 99
 gg::GgMatrix, 150
 gg::GgQuaternion, 239, 240
 gg::GgShader, 250
 gg::GgShape, 254
 gg::GgSimpleShader, 266
 gg::GgTexture, 276
 gg::GgTrackball, 284
 GgApp, 94
 GgApp::Window, 362
 Menu, 339
orthogonal
 gg::GgMatrix, 151
parseconfig.h, 543
 getString, 545, 546
 getValue, 546, 547
 getVector, 548
 setString, 548, 549
 setValue, 550
 setVector, 551
pathChar
 Config.h, 374
pathString
 Config.h, 374
perspective
 gg::GgMatrix, 152
position
 gg::GgSimpleShader::Light, 318
 gg::GgVertex, 316
PROJECT_NAME
 ggsample01.cpp, 536
projection
 gg::GgMatrix, 152–154
read
 gg::GgBuffer< T >, 99
 gg::GgUniformBuffer< T >, 297

README.md, 554
 region
 gg::GgTrackball, 284, 285
 reset
 gg::GgTrackball, 286
 GgApp::Window, 362
 resetRotation
 GgApp::Window, 363
 resetTranslation
 GgApp::Window, 363
 restoreViewport
 GgApp::Window, 364
 rotate
 gg::GgMatrix, 154–156
 gg::GgQuaternion, 240, 241
 gg::GgTrackball, 286
 rotateX
 gg::GgMatrix, 157
 gg::GgQuaternion, 242
 rotateY
 gg::GgMatrix, 158
 gg::GgQuaternion, 243
 rotateZ
 gg::GgMatrix, 159
 gg::GgQuaternion, 244
 save
 Config, 90
 scale
 gg::GgMatrix, 159–161
 select
 gg::GgSimpleShader::LightBuffer, 328
 gg::GgSimpleShader::MaterialBuffer, 337
 selectInterface
 GgApp::Window, 364
 send
 gg::GgBuffer< T >, 99
 gg::GgElements, 109
 gg::GgPoints, 172
 gg::GgTriangles, 291
 gg::GgUniformBuffer< T >, 298
 setClose
 GgApp::Window, 364
 setKeyboardFunc
 GgApp::Window, 366
 setMode
 gg::GgShape, 254
 setMouseFunc
 GgApp::Window, 366
 setResizeFunc
 GgApp::Window, 366
 setString
 parseconfig.h, 548, 549
 setUserPointer
 GgApp::Window, 367
 setValue
 parseconfig.h, 550
 setVector
 parseconfig.h, 551
 setVelocity
 GgApp::Window, 367
 setWheelFunc
 GgApp::Window, 367
 shininess
 gg::GgSimpleShader::Material, 330
 shouldClose
 GgApp::Window, 368
 slerp
 gg::GgQuaternion, 244, 245
 specular
 gg::GgSimpleShader::Light, 318
 gg::GgSimpleShader::Material, 330
 subtract
 gg::GgQuaternion, 245–247
 swapBuffers
 GgApp::Window, 368
 swapRandB
 gg::GgTexture, 276
 TCharToUtf8
 Config.h, 374
 translate
 gg::GgMatrix, 162, 163
 transpose
 gg::GgMatrix, 164
 unbind
 gg::GgBuffer< T >, 100
 gg::GgTexture, 277
 gg::GgUniformBuffer< T >, 298
 unmap
 gg::GgBuffer< T >, 100
 gg::GgUniformBuffer< T >, 298
 unuse
 gg::GgPointShader, 179
 gg::GgShader, 250
 updateViewport
 GgApp::Window, 368
 use
 gg::GgPointShader, 179–181
 gg::GgShader, 251
 gg::GgSimpleShader, 266–272
 Utf8ToTChar
 Config.h, 374
 Window
 GgApp::Window, 341, 342