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 copy() . . . 96
8.4.3.3 getBuffer() . . . 97
8.4.3.4 getCount() . . . 97
8.4.3.5 getStride() . . . 97
8.4.3.6 getTarget() . . . 98
8.4.3.7 map() [1/2] . . . 98
8.4.3.8 map() [2/2] . . . 98
8.4.3.9 operator=() . . . 99
8.4.3.10 read() . . . 99
8.4.3.11 send() . . . 99
8.4.3.12 unbind() . . . 100
8.4.3.13 unmap() . . . 100
8.5 gg::GgColorTexture 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 GgColorTexture() [1/3] . . . 101
8.5.2.2 GgColorTexture() [2/3] . . . 101
8.5.2.3 GgColorTexture() [3/3] . . . 102
8.5.2.4 ~GgColorTexture() . . . 103
8.5.3 関数詳解 . . . 103
8.5.3.1 load() [1/2] . . . 103
8.5.3.2 load() [2/2] . . . 104
8.6 gg::GgElements 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 GgElements() [1/2] . . . 106
8.6.2.2 GgElements() [2/2] . . . 106
8.6.2.3 ~GgElements() . . . 107
8.6.3 関数詳解 . . . 107
8.6.3.1 draw() . . . 107
8.6.3.2 getIndexBuffer() . . . 108
8.6.3.3 getIndexCount() . . . 108
8.6.3.4 load() . . . 108
8.6.3.5 send() . . . 109
8.7 gg::GgMatrix 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 GgMatrix() [1/4] . . . 112
8.7.2.2 GgMatrix() [2/4] . . . 112
8.7.2.3 GgMatrix() [3/4] . . . 113
8.7.2.4 GgMatrix() [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 get() . . . 254
8.13.3.3 getMode() . . . 254
8.13.3.4 operator=() . . . 254
8.13.3.5 setMode() . . . 254
8.14 gg::GgSimpleObj 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 GgSimpleObj() . . . 255
8.14.2.2 ~GgSimpleObj() . . . 256
8.14.3 関数詳解 . . . 256
8.14.3.1 draw() . . . 256
8.14.3.2 get() . . . 257
8.15 gg::GgSimpleShader 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 GgSimpleShader() [1/4] . . . 259
8.15.2.2 GgSimpleShader() [2/4] . . . 259
8.15.2.3 GgSimpleShader() [3/4] . . . 260
8.15.2.4 GgSimpleShader() [4/4] . . . 260
8.15.2.5 ~GgSimpleShader() . . . 260
8.15.3 関数詳解 . . . 261
8.15.3.1 load() [1/2] . . . 261
8.15.3.2 load() [2/2] . . . 261
8.15.3.3 loadMatrix() [1/4] . . . 262
8.15.3.4 loadMatrix() [2/4] . . . 263
8.15.3.5 loadMatrix() [3/4] . . . 263
8.15.3.6 loadMatrix() [4/4] . . . 264
8.15.3.7 loadModelviewMatrix() [1/4] . . . 264
8.15.3.8 loadModelviewMatrix() [2/4] . . . 265
8.15.3.9 loadModelviewMatrix() [3/4] . . . 265
8.15.3.10 loadModelviewMatrix() [4/4] . . . 265
8.15.3.11 operator=() . . . 266
8.15.3.12 use() [1/13] . . . 266
8.15.3.13 use() [2/13] . . . 266
8.15.3.14 use() [3/13] . . . 267
8.15.3.15 use() [4/13] . . . 268
8.15.3.16 use() [5/13] . . . 268
8.15.3.17 use() [6/13] . . . 269
8.15.3.18 use() [7/13] . . . 269
8.15.3.19 use() [8/13] . . . 270
8.15.3.20 use() [9/13] . . . 270
8.15.3.21 use() [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 getAltlArrow() . . . 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ル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るので、別に GLEW や glad、 GL3W などを導入する必要はありません。

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

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

2.4.1 補助プログラムのドキュメント

Doxygen で生成したドキュメントの HTML 版を html フォルダに、 PDF 版を pdf フォルダに置いています。

2.4.2 補助プログラムの使い方

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

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

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

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

2.4.3 Oculus Rift を使う場合

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

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

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

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

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

2.4.4 Dear ImGui を使う場合

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

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

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

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

そして、OpenGL の描画ループの中で `ImGui::NewFrame();` と `ImGui::Render();` の間に Dear ImGui の API を置いてください。Dear ImGui のウィンドウの実際のレンダリング (`ImGui_ImplOpenGL3_RenderDrawData();` の呼び出し) は `window.swapbuffers()` の中で行っているので、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
  - gg::GgMatrix . . . 109
  - gg::GgVector . . . 299
    - gg::GgQuaternion . . . 182
      - gg::GgTrackball . . . 277
- 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 &gtext="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

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

覚え書き

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

gg.cpp の 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

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

覚え書き

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

name	ファイル名.
normalize	true 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

q	GgQuaternion 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

csrc	コンピュートシェーダのソースプログラムの文字列.
ctext	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

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

vsrc	バーテックスシェーダのソースプログラムの文字列.
fsrc	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).
gsrc	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	フィードバックする varying 変数のリスト (nullptr なら不使用).
vtext	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.
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.
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.

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。

被呼び出し関係図:

7.1.3.8 ggCross() [1/2]

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

GgVector 型の 3 要素の外積.

引数

a	GgVector 型のベクトル.
b	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

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

a	GgVector 型のベクトル.
b	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:

被呼び出し関係図:

7.1.3.12 ggDistance4() [1/2]

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

GgVector 型の 4 要素の距離.

引数

a	GgVector 型の変数.
b	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

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

戻り値

a と b のそれぞれの 4 要素の距離.

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

a	GgVector 型のベクトル.
b	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

a	GgVector 型の変数.
b	GgVector 型の変数.

戻り値

a と b のそれぞれの 4 要素の内積.

gg.h の 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

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

戻り値

a と b のそれぞれの 4 要素の内積.

gg.h の 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

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

戻り値

GgElements 型のポインタ.

覚え書き

メッシュ状に GgElements 形式の三角形データを生成する.

gg.cpp の 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

name	ファイル名.
normalize	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

radius	球の半径.
slices	球の経度方向の分割数.
stacks	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 )
```

楕円状に三角形を生成する.

引数

width	楕円の横幅.
height	楕円の高さ.
slices	楕円の分割数.

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

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

heading	y 軸中心の回転角.
pitch	x 軸中心の回転角.
roll	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

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

被呼び出し関係図:

7.1.3.28 ggInvert() [1/2]

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

逆行列を返す.

引数

m	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

q	GgQuaternion 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

a	GgVector 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

a	GLfloat 型の 4 要素の配列変数.

戻り値

a の 4 要素の長さ.

gg.h の 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

comp	コンピュートシェーダのソースファイル名.

戻り値

プログラムオブジェクトのプログラム名 (作成できなければ 0).

gg.cpp の 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

name	読み込むファイル名.
nz	法線の z 成分の割合.
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).
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).
internal	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

name	読み込むファイル名.
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).
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).
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.
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.

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

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

vert	バーテックスシェーダのソースファイル名.
frag	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).
geom	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	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

name	読み込む Wavefront OBJ ファイル名.
group	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.
material	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.
vert	読み込んだデータの頂点属性.
normalize	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

name	読み込む Wavefront OBJ ファイル名.
group	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.
material	読み込んだデータのポリゴングループごとの材質.
vert	読み込んだデータの頂点属性.
face	読み込んだデータの三角形の頂点インデックス.
normalize	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

image	テクスチャとして読み込むデータ, nullptr ならテクスチャの作成のみを行う.
width	テクスチャとして読み込むデータ image の横の画素数.
height	テクスチャとして読み込むデータ image の縦の画素数.
format	image のフォーマット.
type	image のデータ型.
internal	テクスチャの内部フォーマット.
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.
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.

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

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

戻り値

求めたビュー変換行列.

gg.h の 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

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

戻り値

求めたビュー変換行列.

gg.h の 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

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

戻り値

求めたビュー変換行列.

gg.h の 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

q	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

m	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

q	GgQuaternion 型の四元数.

戻り値

四元数 q を正規化した四元数.

gg.h の 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

a	GgVector 型のベクトル.

戻り値

a の 3 要素を正規化したもの.

覚え書き

戻り値の w (第4) 要素は 0 になる.

gg.h の 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

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

gg.h の 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

a	GgVector 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

a	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

a	GgVector 型の変数.

戻り値

a の 4 要素を正規化した結果.

gg.h の 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

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

gg.h の 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

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

fovy	y 方向の画角.
aspect	縦横比.
zNear	視点から前方面までの位置.
zFar	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

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

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

a	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

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

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:

被呼び出し関係図:

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

q	元の四元数.

戻り値

四元数 q が表す回転に相当する転置した GgMatrix 型の変換行列.

gg.h の 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

name	読み込むファイル名.
image	読み込んだデータを格納する vector.
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.
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.
pFormat	読み込んだファイルの書式 (GL_RED, G_RG, GL_RGB, G_RGBA) の格納先のポインタ, nullptr 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 行目に定義があります。

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

width	矩形の横幅.
height	矩形の高さ.

戻り値

GgTriangles 型のポインタ.

gg.cpp の 5162 行目に定義があります。

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

r	回転軸のベクトルと回転角を表す GgVector 型の変数.

戻り値

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

gg.h の 3249 行目に定義があります。

呼び出し関係図:

7.1.3.69 ggRotate() [2/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

r	回転軸のベクトルを表す GgVector 型の変数.
a	回転角.

戻り値

r を軸に a だけ回転する変換行列.

gg.h の 3225 行目に定義があります。

呼び出し関係図:

7.1.3.70 ggRotate() [3/5]

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

r 方向のベクトルを軸とする回転の変換行列を乗じた結果を返す.

引数

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

戻り値

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

gg.h の 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

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

戻り値

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

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

戻り値

(x, y, z) を軸にさらに a 回転する変換行列.

gg.h の 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

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

戻り値

回転を表す四元数.

gg.h の 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

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

戻り値

回転を表す四元数.

gg.h の 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

a	回転角.

戻り値

z 軸中心に a だけ回転する変換行列.

gg.h の 3184 行目に定義があります。

呼び出し関係図:

7.1.3.79 ggSaveColor()

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

カラーバッファの内容を TGA ファイルに保存する.

引数

name	保存するファイル名.

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 3518 行目に定義があります。

呼び出し関係図:

7.1.3.80 ggSaveDepth()

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

デプスバッファの内容を TGA ファイルに保存する.

引数

name	保存するファイル名.

戻り値

保存に成功すれば true, 失敗すれば false.

gg.cpp の 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

name	保存するファイル名.
buffer	画像データを格納した配列.
width	画像の横の画素数.
height	画像の縦の画素数.
depth	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。

被呼び出し関係図:

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

s	拡大率の GgVector 型の変数.

戻り値

拡大縮小の変換行列.

gg.h の 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

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

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

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

戻り値

q, a を t で内分した四元数.

gg.h の 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
q	GgQuaternion 型の四元数.
t	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:

被呼び出し関係図:

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

t	移動量の GgVector 型の変数.

戻り値

平行移動の変換行列.

gg.h の 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

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

戻り値

平行移動の変換行列.

gg.h の 3097 行目に定義があります。

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

x	x 方向の移動量.
y	y 方向の移動量.
z	z 方向の移動量.
w	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 行目に定義があります。

呼び出し関係図:

7.1.3.92 ggTranspose()

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

転置行列を返す.

引数

m	元の変換行列.

戻り値

m の転置行列.

gg.h の 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

a	GLfloat 型の値.
b	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

v	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

a	GLfloat 型の値.
b	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

v	GgVector 型のベクトル.

戻り値

v の値の符号を反転した結果.

gg.h の 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

a	GLfloat 型の値.
b	GgVector 型のベクトル.

戻り値

a から b の各要素を引いた差.

gg.h の 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

a	GLfloat 型の値.
b	GgVector 型のベクトル.

戻り値

a を b の各要素で割った商.

gg.h の 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 行目に定義があります。

8.1.2 構築子と解体子

8.1.2.1 Config() [1/2]

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

コンストラクタ

Config.cpp の 25 行目に定義があります。

8.1.2.2 Config() [2/2]

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

ファイルから構成データを読み込むコンストラクタ

Config.cpp の 42 行目に定義があります。

呼び出し関係図:

8.1.2.3 ~Config()

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

デストラクタ

Config.cpp の 52 行目に定義があります。

8.1.3 関数詳解

8.1.3.1 getHeight()

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

ウィンドウの横幅を得る

Config.h の 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 行目に定義があります。

このクラス詳解は次のファイルから抽出されました:

- Config.h
- Config.cpp

8.2 Draw クラス

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

公開メンバ関数

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

8.2.1 詳解

図形の描画クラス

Draw.h の 17 行目に定義があります。

8.2.2 構築子と解体子

8.2.2.1 Draw()

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

コンストラクタ

Draw.cpp の 13 行目に定義があります。

8.2.2.2 ~Draw()

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

デストラクタ

Draw.cpp の 29 行目に定義があります。

8.2.3 関数詳解

8.2.3.1 draw()

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

描画する

Draw.cpp の 36 行目に定義があります。

呼び出し関係図:

このクラス詳解は次のファイルから抽出されました:

- Draw.h
- Draw.cpp

8.3 GgApp クラス

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

クラス

- class Window

公開メンバ関数

- GgApp (int major=0, int minor=1)
- GgApp (const GgApp &w)=delete
- 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 行目に定義があります。

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

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

GgApp.cpp の 24 行目に定義があります。

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 行目に定義があります。

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

target	バッファオブジェクトのターゲット.
data	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).
count	データの数.
stride	データの間隔.
usage	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

src_buffer	複写元のバッファオブジェクト名.
src_first	複写元 (src_buffer) の先頭のデータの位置.
dst_first	複写先 (getBuffer()) の先頭のデータの位置.
count	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

first	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.
count	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

data	抽出先の領域の先頭のポインタ.
first	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.
count	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

data	転送元のデータが格納されている領域の先頭のポインタ.
first	転送先のバッファオブジェクトの先頭の要素番号.
count	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

image	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.
width	読み込む画像の横の画素数.
height	読み込む画像の縦の画素数.
format	読み込む画像のフォーマット.
type	読み込む画像のデータ型.
internal	テクスチャの内部フォーマット.
wrap	テクスチャのラッピングモード.
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.

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

name	読み込むファイル名.
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.
wrap	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

image	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.
width	テクスチャの横の画素数.
height	テクスチャの縦の画素数.
format	読み込む画像のフォーマット.
type	読み込む画像のデータ型.
internal	glTexImage2D() に指定するテクスチャの内部フォーマット.
wrap	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).
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.

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

name	読み込むファイル名.
internal	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.
wrap	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

mode	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

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

引数

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

gg.h の 6465 行目に定義があります。

8.6.2.3 ~GgElements()

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

デストラクタ.

gg.h の 6481 行目に定義があります。

8.6.3 関数詳解

8.6.3.1 draw()

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

インデックスを使った三角形の描画.

引数

first	描画を開始する最初の三角形番号.
count	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 行目に定義があります。

呼び出し関係図:

8.6.3.2 getIndexBuffer()

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

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

戻り値

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

gg.h の 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

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

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

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

gg.h の 2147 行目に定義があります。

8.7.2.3 GgMatrix() [3/4]

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

コンストラクタ.

引数

c	GLfloat 型の値.

gg.h の 2162 行目に定義があります。

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

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

left	ウィンドウの左端の位置.
right	ウィンドウの右端の位置.
bottom	ウィンドウの下端の位置.
top	ウィンドウの上端の位置.
zNear	視点から前方面までの位置.
zFar	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 行目に定義があります。

被呼び出し関係図:

8.7.3.6 loadIdentity()

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

単位行列を格納する.

gg.cpp の 2685 行目に定義があります。

被呼び出し関係図:

8.7.3.7 loadInvert() [1/2]

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

逆行列を格納する.

引数

m	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

a	GLfloat 型の 16 要素の変換行列.

戻り値

設定した a の逆行列.

gg.cpp の 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

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

戻り値

設定したビュー変換行列.

gg.h の 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

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

戻り値

設定したビュー変換行列.

gg.h の 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

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

戻り値

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

m	GgMatrix 型の変換行列.

戻り値

設定した m の法線変換行列.

gg.h の 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

a	GLfloat 型の 16 要素の変換行列.

戻り値

設定した m の法線変換行列.

gg.cpp の 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

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

fovy	y 方向の画角.
aspect	縦横比.
zNear	視点から前方面までの位置.
zFar	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

r	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

r	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.
a	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

r	回転軸の方向ベクトルと回転角を格納した GLfloat 型の 4 要素の配列変数, 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

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

戻り値

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

s	拡大率の GgVector 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

s	拡大率の GLfloat 型の配列 (x, y, z).

戻り値

設定した変換行列.

gg.h の 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

x	x 方向の拡大率.
y	y 方向の拡大率.
z	z 方向の拡大率.
w	w 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

t	移動量の GgVector 型の変数.

戻り値

設定した変換行列.

gg.h の 2411 行目に定義があります。

呼び出し関係図:

8.7.3.28 loadTranslate() [2/3]

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

平行移動の変換行列を格納する.

引数

t	移動量の GLfloat 型の配列 (x, y, z).

戻り値

設定した変換行列.

gg.h の 2400 行目に定義があります。

呼び出し関係図:

8.7.3.29 loadTranslate() [3/3]

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

平行移動の変換行列を格納する.

引数

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

戻り値

設定した変換行列.

gg.cpp の 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]

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

転置行列を格納する.

引数

m	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

a	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

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

戻り値

ビュー変換行列を乗じた変換行列.

gg.h の 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

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

戻り値

ビュー変換行列を乗じた変換行列.

gg.h の 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

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

戻り値

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()

```
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]

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

変換行列に別の変換行列を乗算した値を返す.

引数

m	GgMatrix 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

v	元のベクトルの GgVector 型の変数.

戻り値

v 変換結果の GgVector 型のベクトル.

gg.h の 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

m	GgMatrix 型の変換行列.

戻り値

m を掛けた変換行列の参照.

gg.h の 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

a	変換行列を格納した GLfloat 型の 16 要素の配列変数.

戻り値

a を掛けた変換行列の参照.

gg.h の 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

m	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 GLfloat 型の 16 要素の配列変数.

戻り値

変換行列に a を加えた GgMatrix 型の変換行列.

gg.h の 2195 行目に定義があります。

被呼び出し関係図:

8.7.3.43 operator+=() [1/2]

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

変換行列に別の変換行列を加算した結果を格納する.

引数

m	GgMatrix 型の変換行列.

戻り値

m を加えた変換行列の参照.

gg.h の 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

a	変換行列を格納した GLfloat 型の 16 要素の配列変数.

戻り値

変換行列から a を引いた GgMatrix 型の変換行列.

gg.h の 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

m	GgMatrix 型の変換行列.

戻り値

m を引いた変換行列の参照.

gg.h の 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

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

m	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

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

m	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

a	変換行列を格納した GLfloat 型の 16 要素の配列変数.

戻り値

a で割った変換行列の参照.

gg.h の 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

a	GLfloat 型の 16 要素の配列変数.

戻り値

a を代入したこのオブジェクトの参照.

gg.h の 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

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

fovy	y 方向の画角.
aspect	縦横比.
zNear	視点から前方面までの位置.
zFar	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

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

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

gg.h の 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

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

gg.h の 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

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

gg.h の 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

r	回転軸の方向ベクトルと回転角を格納した GgVector 型の変数).

戻り値

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

gg.h の 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

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

戻り値

(r[0], r[1], r[2]) を軸にさらに a 回転した変換行列.

gg.h の 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

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

戻り値

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

gg.h の 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

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

戻り値

(r[0], r[1], r[2]) を軸にさらに a 回転した変換行列.

gg.h の 2805 行目に定義があります。

呼び出し関係図:

8.7.3.64 rotate() [5/5]

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

(x, y, z) 方向のベクトルを軸とする回転変換を乗じた結果を返す.

引数

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

戻り値

(x, y, z) を軸にさらに a 回転した変換行列.

gg.h の 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

a	回転角.

戻り値

x 軸中心にさらに a 回転した変換行列.

gg.h の 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

a	回転角.

戻り値

y 軸中心にさらに a 回転した変換行列.

gg.h の 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

a	回転角.

戻り値

z 軸中心にさらに a 回転した変換行列.

gg.h の 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

s	拡大率の GgVector 型の変数.

戻り値

拡大縮小した結果の変換行列.

gg.h の 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

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

戻り値

拡大縮小した結果の変換行列.

gg.h の 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

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

戻り値

拡大縮小した結果の変換行列.

gg.h の 2719 行目に定義があります。

呼び出し関係図:

被呼び出し関係図:

8.7.3.71 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

t	移動量の GgVector 型の変数.

戻り値

平行移動した結果の変換行列.

gg.h の 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

t	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 行目に定義があります。

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

x	x 方向の移動量.
y	y 方向の移動量.
z	z 方向の移動量.
w	w 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 行目に定義があります。

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

image	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.
width	テクスチャとして用いる画像データの横幅.
height	テクスチャとして用いる画像データの高さ.
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).
nz	法線マップの z 成分の値.
internal	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

name	画像ファイル名.
nz	法線マップの z 成分の値.
internal	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

hmap	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.
width	テクスチャとして用いる画像データの横幅.
height	テクスチャとして用いる画像データの高さ.
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).
nz	法線マップの z 成分の値.
internal	テクスチャの内部フォーマット.

gg.h の 5489 行目に定義があります。

呼び出し関係図:

被呼び出し関係図:

8.8.3.2 load() [2/2]

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

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

引数

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

gg.cpp の 3975 行目に定義があります。

呼び出し関係図:

このクラス詳解は次のファイルから抽出されました:

- gg.h
- gg.cpp

8.9 gg::GgPoints クラス

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

gg::GgPoints の継承関係図

gg::GgPoints 連携図

公開メンバ関数

- GgPoints (GLenum mode=GL_POINTS)
- GgPoints (const GgVector *pos, GLsizei countv, GLenum mode=GL_POINTS, GLenum usage=GL_STATIC↩_DRAW)
- virtual ~GgPoints ()
- 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 行目に定義があります。

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 行目に定義があります。

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

pos	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).
countv	頂点数.
mode	描画する基本図形の種類.
usage	バッファオブジェクトの使い方.

gg.h の 6208 行目に定義があります。

8.9.2.3 ~GgPoints()

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

デストラクタ.

gg.h の 6222 行目に定義があります。

8.9.3 関数詳解

8.9.3.1 draw()

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

点の描画.

引数

first	描画を開始する最初の点の番号.
count	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

pos	頂点の位置データが格納されている領域の先頭のポインタ.
count	頂点のデータの数 (頂点数).
usage	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

pos	転送元の頂点の位置データが格納されている領域の先頭のポインタ.
first	転送先のバッファオブジェクトの先頭の要素番号.
count	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 行目に定義があります。

8.10.2 構築子と解体子

8.10.2.1 GgPointShader() [1/3]

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

コンストラクタ.

gg.h の 6804 行目に定義があります。

8.10.2.2 GgPointShader() [2/3]

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

コンストラクタ.

引数

vert	バーテックスシェーダのソースファイル名.
frag	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).
geom	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	フィードバックする varying 変数のリスト.

gg.h の 6819 行目に定義があります。

8.10.2.3 GgPointShader() [3/3]

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

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

コンストラクタ.

引数

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

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

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 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

vert	バーテックスシェーダのソースファイル名.
frag	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).
geom	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	フィードバックする varying 変数のリスト.

戻り値

プログラムオブジェクトが作成できれば true.

gg.h の 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

mp	GgMatrix 型の投影変換行列.
mv	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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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

mv	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

mv	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

mp	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

mp	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

mp	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

mp	GgMatrix 型の投影変換行列.
mv	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

mp	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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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 行目に定義があります。

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 行目に定義があります。

8.11.2.2 GgQuaternion() [2/5]

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

コンストラクタ.

引数

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

gg.h の 3440 行目に定義があります。

8.11.2.3 GgQuaternion() [3/5]

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

コンストラクタ.

引数

c	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

a	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

v	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

q	GgQuaternion 型の四元数.

戻り値

q を加えた四元数.

gg.h の 3779 行目に定義があります。

呼び出し関係図:

8.11.3.2 add() [2/4]

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

四元数に別の四元数を加算した結果を返す.

引数

v	四元数を格納した GgVector 型の変数.

戻り値

v を加えた四元数.

gg.h の 3768 行目に定義があります。

呼び出し関係図:

8.11.3.3 add() [3/4]

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

四元数に別の四元数を加算した結果を返す.

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数.

戻り値

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

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 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

q	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

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 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

heading	y 軸中心の回転角.
pitch	x 軸中心の回転角.
roll	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

m	回転の変換行列を格納する GgMatrix 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

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 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

m	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

q	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

v	四元数を格納した GgVector 型の変数.

戻り値

設定した四元数.

gg.h の 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.

戻り値

設定した四元数.

gg.h の 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

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

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

q	GgQuaternion 型の四元数.

戻り値

q を加えた四元数.

gg.h の 3584 行目に定義があります。

呼び出し関係図:

8.11.3.25 loadAdd() [2/4]

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

四元数に別の四元数を加算した結果を格納する.

引数

v	四元数を格納した GgVector 型の変数.

戻り値

v を加えた四元数.

gg.h の 3573 行目に定義があります。

呼び出し関係図:

8.11.3.26 loadAdd() [3/4]

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

四元数に別の四元数を加算した結果を格納する.

引数

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

q	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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.

戻り値

共役四元数.

gg.cpp の 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

q	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

heading	y 軸中心の回転角.
pitch	x 軸中心の回転角.
roll	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

q	GgQuaternion 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

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 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。

被呼び出し関係図:

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 行目に定義があります。

呼び出し関係図:

8.11.3.44 loadMultiply() [4/4]

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

四元数に別の四元数を乗算した結果を格納する.

引数

x	掛ける四元数の x 要素.
y	掛ける四元数の y 要素.
z	掛ける四元数の z 要素.
w	掛ける四元数の w 要素.

戻り値

(x, y, z, w) を掛けた四元数.

gg.h の 3649 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:

8.11.3.45 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

q	GgQuaternion 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.

戻り値

正規化された四元数.

gg.cpp の 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]

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

(x, y, z) を軸として角度 a 回転する四元数を格納する.

引数

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

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

a	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

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

戻り値

格納した q, a を t で内分した四元数.

gg.h の 4282 行目に定義があります。

呼び出し関係図:

8.11.3.55 loadSlerp() [3/4]

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

球面線形補間の結果を格納する.

引数

a	四元数を格納した GLfloat 型の 4 要素の配列変数.
q	GgQuaternion 型の四元数.
t	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

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

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

q	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

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

戻り値

(x, y, z, w) を引いた四元数.

gg.h の 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

q	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

v	四元数を格納した GgVector 型の変数.

戻り値

v を掛けた四元数.

gg.h の 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

a	四元数を格納した GLfloat 型の 4 要素の配列変数.

戻り値

a を掛けた四元数.

gg.h の 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

x	掛ける四元数の x 要素.
y	掛ける四元数の y 要素.
z	掛ける四元数の z 要素.
w	掛ける四元数の w 要素.

戻り値

(x, y, z, w) を掛けた四元数.

gg.h の 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 行目に定義があります。

呼び出し関係図:

8.11.3.83 operator-=() [2/3]

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

gg.h の 3958 行目に定義があります。

呼び出し関係図:

8.11.3.84 operator-=() [3/3]

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

gg.h の 3954 行目に定義があります。

呼び出し関係図:

被呼び出し関係図:

8.11.3.85 operator/() [1/3]

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

gg.h の 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

a	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

q	GgQuaternion 型の四元数.
t	補間パラメータ.

戻り値

四元数を q に対して t で内分した結果.

gg.h の 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

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

戻り値

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

q	GgQuaternion 型の四元数.

戻り値

q 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

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 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

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 3805 行目に定義があります。

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

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

戻り値

(x, y, z, w) を引いた四元数.

gg.h の 3793 行目に定義があります。

呼び出し関係図:

被呼び出し関係図:

このクラス詳解は次のファイルから抽出されました:

- gg.h
- gg.cpp

8.12 gg::GgShader クラス

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

公開メンバ関数

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

8.12.1 詳解

シェーダの基底クラス.

覚え書き

シェーダのクラスはこのクラスを派生して作る.

gg.h の 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

vert	バーテックスシェーダのソースファイル名.
frag	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).
geom	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	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

files	バーテックスシェーダのソースファイル名の配列.
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	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 行目に定義があります。

8.12.2.3 GgShader() [3/3]

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

コピーコンストラクタは使用禁止.

8.12.2.4 ~GgShader()

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

デストラクタ.

gg.h の 6746 行目に定義があります。

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 行目に定義があります。

8.12.3.2 operator=()

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

代入演算子は使用禁止.

8.12.3.3 unuse()

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

シェーダプログラムの使用を終了する.

gg.h の 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

mode	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

first	描画する最初のアイテム.
count	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

mode	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

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

gg.cpp の 5825 行目に定義があります。

呼び出し関係図:

8.14.2.2 ~GgSimpleObj()

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

デストラクタ.

gg.h の 8007 行目に定義があります。

8.14.3 関数詳解

8.14.3.1 draw()

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

Wavefront OBJ 形式のデータを描画する手続き.

引数

first	描画する最初のパーツ番号.
count	描画するパーツの数, 0 なら全部のパーツを描く.

gg.cpp の 5853 行目に定義があります。

被呼び出し関係図:

8.14.3.2 get()

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

形状データの取り出し.

戻り値

GgTriangles 型の形状データのポインタ.

gg.h の 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

vert	バーテックスシェーダのソースファイル名.
frag	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).
geom	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).
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	フィードバックする varying 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

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

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

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.h の 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

vert	バーテックスシェーダのソースプログラムの文字列.
frag	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）.
geom	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）.
nvarying	フィードバックする varying 変数の数 (0 なら不使用).
varyings	フィードバックする varying 変数のリスト (nullptr なら不使用).

戻り値

プログラムオブジェクトの作成に成功したら true.

gg.cpp の 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

mp	GgMatrix 型の投影変換行列.
mv	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 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

mp	GgMatrix 型の投影変換行列.
mv	GgMatrix 型のモデルビュー変換行列.
mn	GgMatrix 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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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 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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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.
mn	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

mv	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

mv	GgMatrix 型のモデルビュー変換行列.
mn	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

mv	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

mv	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.
mn	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

mp	GgMatrix 型の投影変換行列.
mv	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

mp	GgMatrix 型の投影変換行列.
mv	GgMatrix 型のモデルビュー変換行列.
mn	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

mp	GgMatrix 型の投影変換行列.
mv	GgMatrix 型のモデルビュー変換行列.
mn	GgMatrix 型のモデルビュー変換行列の法線変換行列.
light	光源の特性の gg::LightBuffer 構造体.
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

mp	GgMatrix 型の投影変換行列.
mv	GgMatrix 型のモデルビュー変換行列.
light	光源の特性の gg::LightBuffer 構造体.
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

mp	GgMatrix 型の投影変換行列.
light	光源の特性の gg::LightBuffer 構造体.
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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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.
mn	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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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.
mn	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.
light	光源の特性の gg::LightBuffer 構造体のポインタ.
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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
mv	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.
light	光源の特性の gg::LightBuffer 構造体のポインタ.
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

mp	GLfloat 型の 16 要素の配列変数に格納された投影変換行列.
light	光源の特性の gg::LightBuffer 構造体のポインタ.
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

light	光源の特性の gg::LightBuffer 構造体.
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

light	光源の特性の gg::LightBuffer 構造体のポインタ.
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

image	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.
width	テクスチャの横の画素数.
height	テクスチャの縦の画素数.
format	読み込む画像のフォーマット.
type	画像のデータ型.
internal	テクスチャの内部フォーマット.
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.

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]
```

コピーコンストラクタは使用禁止.

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

q	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

x	現在のマウスの x 座標.
y	現在のマウスの y 座標.

覚え書き

マウスのドラッグ開始時 (マウスボタンを押したとき) に呼び出す.

gg.cpp の 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

x	現在のマウスの x 座標.
y	現在のマウスの y 座標.

覚え書き

マウスのドラッグ終了時 (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

factor	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

direction	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

position	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

direction	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

x	現在のマウスの x 座標.
y	現在のマウスの y 座標.

覚え書き

マウスのドラッグ中に呼び出す.

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

q	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

q	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

mode	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

vert	この図形の頂点属性の配列 (nullptr ならデータを転送しない).
count	頂点数.
mode	描画する基本図形の種類.
usage	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

first	描画を開始する最初の三角形番号.
count	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

vert	頂点属性が格納されている領域の先頭のポインタ.
count	頂点のデータの数 (頂点数).
usage	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

vert	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.
first	転送先のバッファオブジェクトの先頭の要素番号.
count	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

data	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).
count	データの数.
usage	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

data	格納するデータ.
count	格納する数.
usage	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

src_buffer	複写元のバッファオブジェクト名.
src_first	複写元 (buffer) の先頭のデータの位置.
dst_first	複写先 (getBuffer()) の先頭のデータの位置.
count	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

data	格納するデータ.
offset	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.
size	格納するデータの一個あたりのバイト数.
first	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.
count	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

data	格納するデータ.
count	格納する数.
usage	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

data	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).
count	データの数.
usage	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

first	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.
count	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

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

8.19.3.13 send()

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

ユニフォームバッファオブジェクトを確保してユニフォームバッファオブジェクトのブロックごとのメンバを同じデータで埋める.

引数

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

gg.h の 5943 行目に定義があります。

8.19.3.14 unbind()

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

ユニフォームバッファオブジェクトを解放する.

gg.h の 5861 行目に定義があります。

8.19.3.15 unmap()

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

バッファオブジェクトをアンマップする.

gg.h の 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

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

gg.h の 1575 行目に定義があります。

8.20.2.3 GgVector() [3/4]

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

コンストラクタ.

引数

c	GLfloat 型の値.

gg.h の 1585 行目に定義があります。

8.20.2.4 GgVector() [4/4]

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

コンストラクタ.

引数

a	GLfloat 型の 4 要素の配列変数.

gg.h の 1595 行目に定義があります。

8.20.3 関数詳解

8.20.3.1 distance3()

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

GgVector 型の 3 要素の距離.

引数

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

v	GgVector 型のベクトル.

戻り値

オブジェクトと v のそれぞれの 4 要素の内積.

gg.h の 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

c	GLfloat 型の変数.

戻り値

オブジェクトの各要素に c を乗じたオブジェクト.

gg.h の 1721 行目に定義があります。

8.20.3.11 operator*=() [1/2]

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

GgVector 型を乗算する.

引数

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 1731 行目に定義があります。

8.20.3.12 operator*=() [2/2]

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

GgVector 型の各要素にスカラーを乗じる.

引数

c	GgVector 型のベクトル.

戻り値

オブジェクトの各要素に c を乗じたオブジェクトの参照.

gg.h の 1746 行目に定義があります。

8.20.3.13 operator+() [1/2]

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

GgVector 型の和を返す.

引数

v	GgVector 型のベクトル.

戻り値

オブジェクトの各要素と v の各要素の要素ごとの和のオブジェクト.

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

v	GgVector 型の変数.

戻り値

オブジェクトの各要素に v の各要素をそれぞれ加算したオブジェクトの参照.

gg.h の 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

c	GLfloat 型の値.

戻り値

オブジェクトの各要素に c を足したオブジェクトの参照.

gg.h の 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

v	GgVector 型の変数.

戻り値

オブジェクトの各要素と v の各要素の要素ごとの差のオブジェクト.

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

c	GLfloat 型の変数.

戻り値

オブジェクトの各要素から c を引いたオブジェクトの参照.

gg.h の 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

v	GgVector 型の変数.

戻り値

オブジェクトの各要素を v の各要素で要素ごとに割った結果のオブジェクト.

gg.h の 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

c	GLfloat 型の値.

戻り値

オブジェクトの各要素を c で割った商のオブジェクト.

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

pos	GgVector 型の位置データ.
norm	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

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

gg.h の 6316 行目に定義があります。

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

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

gg.h の 6331 行目に定義があります。

8.21.3 メンバ詳解

8.21.3.1 normal

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

法線.

gg.h の 6285 行目に定義があります。

8.21.3.2 position

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

位置.

gg.h の 6282 行目に定義があります。

この構造体詳解は次のファイルから抽出されました:

- gg.h

8.22 gg::GgSimpleShader::Light 構造体

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

gg::GgSimpleShader::Light 連携図

公開変数類

- GgVector ambient
  光源強度の環境光成分.
- GgVector diffuse
  光源強度の拡散反射光成分.
- GgVector specular
  光源強度の鏡面反射光成分.
- GgVector position
  光源の位置.

8.22.1 詳解

三角形に単純な陰影付けを行うシェーダが参照する光源データ.

gg.h の 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

light	GgSimpleShader::Light 型の光源データのポインタ.
count	バッファ中の GgSimpleShader::Light 型の光源データの数.
usage	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 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

light	GgSimpleShader::Light 型の光源データ.
count	バッファ中の GgSimpleShader::Light 型の光源データの数.
usage	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

light	光源の特性の GgSimpleShader::Light 構造体.
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.
count	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

light	光源の特性の GgSimpleShader::Light 構造体のポインタ.
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.
count	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

ambient	光源の強度の環境光成分を格納した GgVector 型の変数.
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.
count	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

ambient	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.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.h の 7343 行目に定義があります。

8.23.3.5 loadAmbient() [3/3]

```
void gg::GgSimpleShader::LightBuffer::loadAmbient (
            GLfloat r,
            GLfloat g,
            GLfloat b,
            GLfloat a = 1.0f,
            GLint first = 0,
            GLsizei count = 1 ) const
```

光源の強度の環境光成分を設定する.

引数

r	光源の強度の環境光成分の赤成分.
g	光源の強度の環境光成分の緑成分.
b	光源の強度の環境光成分の青成分.
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.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5384 行目に定義があります。

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

color	光源の特性の GgSimpleShader::Light 構造体.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5537 行目に定義があります。

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

diffuse	光源の強度の拡散反射光成分を格納した GgVector 型の変数.
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.
count	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

diffuse	光源の強度の拡散反射光成分を格納した GLfloat 型の 4 要素の配列変数.
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.
count	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

r	光源の強度の拡散反射光成分の赤成分.
g	光源の強度の拡散反射光成分の緑成分.
b	光源の強度の拡散反射光成分の青成分.
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.
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.
count	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

position	光源の位置の同次座標を格納した GgVector 型の変数.
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.
count	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

position	光源の位置の同次座標を格納した GgVector 型の配列.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.h の 7485 行目に定義があります。

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

position	光源の位置の同次座標を格納した GLfloat 型の 4 要素の配列変数.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.h の 7472 行目に定義があります。

8.23.3.13 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

x	光源の位置の x 座標.
y	光源の位置の y 座標.
z	光源の位置の z 座標.
w	光源の位置の w 座標, デフォルトは 1.
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.
count	値を設定する光源データの数, デフォルトは 1.

gg.cpp の 5565 行目に定義があります。

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

specular	光源の強度の鏡面反射光成分を格納した GgVector 型の変数.
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.
count	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

specular	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.
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.
count	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

r	光源の強度の鏡面反射光成分の赤成分.
g	光源の強度の鏡面反射光成分の緑成分.
b	光源の強度の鏡面反射光成分の青成分.
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.
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.
count	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	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

material	GgSimpleShader::Material 型の材質データのポインタ.
count	バッファ中の GgSimpleShader::Material 型の材質データの数.
usage	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 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

material	GgSimpleShader::Material 型の材質データ.
count	バッファ中の GgSimpleShader::Material 型の材質データの数.
usage	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 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

material	光源の特性の GgSimpleShader::Material 構造体.
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.
count	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

material	光源の特性の GgSimpleShader::Material 構造体のポインタ.
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.
count	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

ambient	環境光に対する反射係数を格納した GLfloat 型の 4 要素の配列変数.
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.
count	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

r	環境光に対する反射係数の赤成分.
g	環境光に対する反射係数の緑成分.
b	環境光に対する反射係数の青成分.
a	環境光に対する反射係数の不透明度, デフォルトは 1.
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.
count	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5617 行目に定義があります。

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

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 要素の配列変数.
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.
count	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5701 行目に定義があります。

8.25.3.6 loadAmbientAndDiffuse() [2/2]

```
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse (
            GLfloat r,
```

```
            GLfloat g,
            GLfloat b,
            GLfloat a = 1.0f,
            GLint first = 0,
            GLsizei count = 1 ) const
```

環境光に対する反射係数と拡散反射係数を設定する.

引数

r	環境光に対する反射係数と拡散反射係数の赤成分.
g	環境光に対する反射係数と拡散反射係数の緑成分.
b	環境光に対する反射係数と拡散反射係数の青成分.
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.
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.
count	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

diffuse	拡散反射係数を格納した GLfloat 型の 4 要素の配列変数.
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.
count	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

r	拡散反射係数の赤成分.
g	拡散反射係数の緑成分.
b	拡散反射係数の青成分.
a	拡散反射係数の不透明度, デフォルトは 1.
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.
count	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

shininess	輝き係数.
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.
count	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

shininess	輝き係数.
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.
count	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5770 行目に定義があります。

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

specular	鏡面反射係数を格納した GLfloat 型の 4 要素の配列変数.
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.
count	値を設定する材質データの数, デフォルトは 1.

gg.h の 7687 行目に定義があります。

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

r	鏡面反射係数の赤成分.
g	鏡面反射係数の緑成分.
b	鏡面反射係数の青成分.
a	鏡面反射係数の不透明度, デフォルトは 1.
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.
count	値を設定する材質データの数, デフォルトは 1.

gg.cpp の 5744 行目に定義があります。

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	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 行目に定義があります。

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 行目に定義があります。

8.26.3 関数詳解

8.26.3.1 draw()

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

描画する

Menu.cpp の 74 行目に定義があります。

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 getAltlArrow (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

title	ウィンドウタイトルの文字列.
width	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.
height	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.
fullscreen	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.
share	共有するコンテキスト, nullptr 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 getAltlArrow()

```
void GgApp::Window::getAltlArrow (
            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

alt_arrow	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

arrow	矢印キーの値を格納する GLfloat[2] の配列.
mods	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

direction	方向 (0: X, 1:Y).
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

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 X 値.

GgApp.h の 475 行目に定義があります。

呼び出し関係図:

被呼び出し関係図:

8.27.3.8 getArrowY()

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

矢印キーの現在の Y 値を得る.

引数

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 行目に定義があります。
呼び出し関係図:

被呼び出し関係図:

8.27.3.9 getAspect()

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

ウィンドウの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

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

GgApp.h の 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 行目に定義があります。

被呼び出し関係図:

8.27.3.14 getFboSize() [1/2]

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

FBO のサイズを得る.

戻り値

FBO の幅と高さを格納した GLsizei 型の 2 要素の配列.

GgApp.h の 379 行目に定義があります。

8.27.3.15 getFboSize() [2/2]

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

FBO のサイズを得る.

引数

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 行目に定義があります。

呼び出し関係図:

8.27.3.16 getFboWidth()

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

FBO の横幅を得る.

戻り値

FBO の横幅.

GgApp.h の 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

position	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 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

direction	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

button	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

button	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 GgMarix 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

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

戻り値

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

shift_arrow	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

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

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

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

戻り値

平行移動量を格納した GLfloat[3] の配列のポインタ.

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

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

戻り値

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

rotation	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 行目に定義があります。

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

direction	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 行目に定義があります。

8.27.3.39 getWheelX()

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

マウスホイールの X 方向の回転量を得る.

戻り値

マウスホイールの X 方向の回転量.

GgApp.h の 693 行目に定義があります。

8.27.3.40 getWheelY()

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

マウスホイールの Y 方向の回転量を得る.

戻り値

マウスホイールの Y 方向の回転量.

GgApp.h の 704 行目に定義があります。

8.27.3.41 getWidth()

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

ウィンドウの横幅を得る.

戻り値

ウィンドウの横幅.

GgApp.h の 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

flag	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

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

GgApp.h の 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

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

GgApp.h の 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

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

GgApp.h の 844 行目に定義があります。

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

pointer	保存するユーザポインタ.

GgApp.h の 834 行目に定義があります。

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

vx	x 方向の移動速度.
vy	y 方向の移動速度.
vz	z 方向の移動速度.

GgApp.h の 438 行目に定義があります。

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

func	ユーザ定義の wheel 関数, マウスホイールの操作時に呼び出される.

GgApp.h の 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 }, // 拡散反射光成分
00018   { 1.0f, 1.0f, 1.0f, 0.0f }, // 鏡面反射光成分
00019   { 0.5f, 0.5f, 1.0f, 1.0f }  // 光源位置
00020 };
00021
00022 //
00023 // コンストラクタ
00024 //
00025 Config::Config() :
00026   winSize{ 960, 540 },
00027   menuFont{ "Mplus1-Regular.ttf" },
00028   menuFontSize{ 20.0f },
00029   light{ defaultLight },
00030   model{ "logo.obj" },
00031 #if defined(GL_GLES_PROTOTYPES)
00032   shader{ "simple_es3.vert", "simple_es3.frag" }
00033 #else
00034   shader{ "simple.vert", "simple.frag" }
00035 #endif
00036 {
00037 }
00038
00039 //
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アライズして JSON で保存
```

```
  picojson::value v{ object };
  file << v.serialize(true);
  file.close();

  return true;
}

#if defined(_MSC_VER)
//
// 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
//
//   Original author: mt-u
//   Copyright (c) 2013 mt-u
//   https://gist.github.com/mt-u/6878251
//
//   Modified by: Kohe Tokoi
//

//
// UTF-8 文字列を CString に変換する
//
CString Utf8ToTChar(const std::string& string)
{
  // 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
  const INT length{ MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, NULL, 0) };

  // 変換結果の格納に必要な長さのメモリを確保する
  std::vector<WCHAR> utf16(length);

  // UTF-8 文字列を UTF-16 に変換する
  MultiByteToWideChar(CP_UTF8, 0, string.c_str(), -1, utf16.data(), length);

  // 変換した文字列を CString にして返す
  return CString{ utf16.data(), static_cast<int>(wcslen(utf16.data())) };
}

//
// Cstring を UTF-8 文字列に変換する
//
std::string TCharToUtf8(const CString& cstring)
{
  // 与えられた文字列を一旦 UTF-16 に変換する
  CStringW cstringw{ cstring };

  // UTF-16 を UTF-8 に変換した後のの文字列の長さを求める
  const INT length{ WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, NULL, 0, NULL, NULL) };

  // 変換結果の格納に必要な長さのメモリを確保する
  std::vector<CHAR> utf8(length);

  // UTF-16 を UTF-8 に変換する
  WideCharToMultiByte(CP_UTF8, 0, cstringw, -1, utf8.data(), length, NULL, NULL);

  // 変換した文字列を std::string にして返す
  return std::string{ utf8.data(), strlen(utf8.data()) };
}
#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 行目に定義があります。

9.3.2.2 Utf8ToTChar

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

Config.h の 33 行目に定義があります。

9.3.3 型定義詳解

9.3.3.1 pathChar

```
using pathChar = char
```

Config.h の 32 行目に定義があります。

9.3.3.2 pathString

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

Config.h の 31 行目に定義があります。

9.4 Config.h

[詳解]

```
#pragma once


// アプリケーションの設定
#include "GgApp.h"

// 構成ファイルの読み取り補助
#include "parseconfig.h"

#if defined(_MSC_VER)
//
// Windows / Visual C++ におけるパス名
//

// 文字コード
#include <atlstr.h>
using pathString = CString;
using pathChar = wchar_t;

// 文字コード変換
extern CString Utf8ToTChar(const std::string& string);
extern std::string TCharToUtf8(const CString& cstring);
#else
using pathString = std::string;
using pathChar = char;
#define Utf8ToTChar(string) (string)
#define TCharToUtf8(cstring) (cstring)
#endif

class Config
{
  // メニュークラスから参照する
  friend class Menu;

  // ウィンドウサイズ
  std::array<GLsizei, 2> winSize;

  // メニューフォント名
  std::string menuFont;

  // メニューフォントサイズ
  float menuFontSize;

  // 光源
  GgSimpleShader::Light light;

  // 形状ファイル名
  std::string model;

  // シェーダのソースファイル名
  std::array<std::string, 3> shader;

public:

  Config();

  Config(const std::string& filename);

  virtual ~Config();

  auto getWidth() const
  {
    return winSize[0];
  }

  auto getHeight() const
  {
    return winSize[1];
  }

  bool load(const std::string& filename);

  bool save(const std::string& filename) const;
};
```

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 }
```

```
//
// デストラクタ
//
Draw::~Draw()
{
}

//
// 描画する
//
void Draw::draw(const GgMatrix& mp, const GgMatrix& mv) const
{
  // シェーダプログラムを指定する
  shader.use(mp, mv, light);

  // 図形を描画する
  model.draw();
}
```

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
00034
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
00046 #define READ_TEXTURE_COORDINATE_FROM_OBJ 0
00047
00048 // Windows (Visual Studio) のとき
00049 #if defined(_MSC_VER)
00050 // リリースビルドではコンソールを出さない
00051 #  if !defined(_DEBUG)
00052 #    pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
00053 #  endif
00054 // リンクするライブラリ
00055 #  pragma comment(lib, "lib\\" GLFW3_PLATFORM "\\" GLFW3_CONFIGURATION "\\glfw3.lib")
00056 #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;
```

```
PFNGLATTACHSHADERPROC glAttachShader;
PFNGLBEGINCONDITIONALRENDERNVPROC glBeginConditionalRenderNV;
PFNGLBEGINCONDITIONALRENDERPROC glBeginConditionalRender;
PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
PFNGLBEGINQUERYPROC glBeginQuery;
PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
PFNGLBINDBUFFERBASEPROC glBindBufferBase;
PFNGLBINDBUFFERPROC glBindBuffer;
PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
PFNGLBINDSAMPLERPROC glBindSampler;
PFNGLBINDSAMPLERSPROC glBindSamplers;
PFNGLBINDTEXTUREPROC glBindTexture;
PFNGLBINDTEXTURESPROC glBindTextures;
PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
PFNGLBLENDBARRIERKHRPROC glBlendBarrierKHR;
PFNGLBLENDBARRIERNVPROC glBlendBarrierNV;
PFNGLBLENDCOLORPROC glBlendColor;
PFNGLBLENDEQUATIONIARBPROC glBlendEquationiARB;
PFNGLBLENDEQUATIONIPROC glBlendEquationi;
PFNGLBLENDEQUATIONPROC glBlendEquation;
PFNGLBLENDEQUATIONSEPARATEIARBPROC glBlendEquationSeparateiARB;
PFNGLBLENDEQUATIONSEPARATEIPROC glBlendEquationSeparatei;
PFNGLBLENDEQUATIONSEPARATEPROC glBlendEquationSeparate;
PFNGLBLENDFUNCIARBPROC glBlendFunciARB;
PFNGLBLENDFUNCIPROC glBlendFunci;
PFNGLBLENDFUNCPROC glBlendFunc;
PFNGLBLENDFUNCSEPARATEIARBPROC glBlendFuncSeparateiARB;
PFNGLBLENDFUNCSEPARATEIPROC glBlendFuncSeparatei;
PFNGLBLENDFUNCSEPARATEPROC glBlendFuncSeparate;
PFNGLBLENDPARAMETERINVPROC glBlendParameteriNV;
PFNGLBLITFRAMEBUFFERPROC glBlitFramebuffer;
PFNGLBLITNAMEDFRAMEBUFFERPROC glBlitNamedFramebuffer;
PFNGLBUFFERADDRESSRANGENVPROC glBufferAddressRangeNV;
PFNGLBUFFERDATAPROC glBufferData;
PFNGLBUFFERPAGECOMMITMENTARBPROC glBufferPageCommitmentARB;
PFNGLBUFFERSTORAGEPROC glBufferStorage;
PFNGLBUFFERSUBDATAPROC glBufferSubData;
PFNGLCALLCOMMANDLISTNVPROC glCallCommandListNV;
PFNGLCHECKFRAMEBUFFERSTATUSPROC glCheckFramebufferStatus;
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glCheckNamedFramebufferStatusEXT;
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glCheckNamedFramebufferStatus;
PFNGLCLAMPCOLORPROC glClampColor;
PFNGLCLEARBUFFERDATAPROC glClearBufferData;
PFNGLCLEARBUFFERFIPROC glClearBufferfi;
PFNGLCLEARBUFFERFVPROC glClearBufferfv;
PFNGLCLEARBUFFERIVPROC glClearBufferiv;
PFNGLCLEARBUFFERSUBDATAPROC glClearBufferSubData;
PFNGLCLEARBUFFERUIVPROC glClearBufferuiv;
PFNGLCLEARCOLORPROC glClearColor;
PFNGLCLEARDEPTHFPROC glClearDepthf;
PFNGLCLEARDEPTHPROC glClearDepth;
PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferuiv;
PFNGLCLEARPROC glClear;
PFNGLCLEARSTENCILPROC glClearStencil;
PFNGLCLEARTEXIMAGEPROC glClearTexImage;
PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
PFNGLCLIENTATTRIBDEFAULTEXTPROC glClientAttribDefaultEXT;
PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
PFNGLCLIPCONTROLPROC glClipControl;
PFNGLCOLORFORMATNVPROC glColorFormatNV;
PFNGLCOLORMASKIPROC glColorMaski;
PFNGLCOLORMASKPROC glColorMask;
PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
```

```
PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
PFNGLCOMPILESHADERPROC glCompileShader;
PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
PFNGLCONSERVATIVERASTERPARAMETERFNVPROC glConservativeRasterParameterfNV;
PFNGLCONSERVATIVERASTERPARAMETERINVPROC glConservativeRasterParameteriNV;
PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
PFNGLCOPYPATHNVPROC glCopyPathNV;
PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationNV;
PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTableNV;
PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancedNV;
PFNGLCOVERFILLPATHNVPROC glCoverFillPathNV;
PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancedNV;
PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathNV;
PFNGLCREATEBUFFERSPROC glCreateBuffers;
PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
PFNGLCREATEPROGRAMPROC glCreateProgram;
PFNGLCREATEQUERIESPROC glCreateQueries;
PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
PFNGLCREATESAMPLERSPROC glCreateSamplers;
PFNGLCREATESHADERPROC glCreateShader;
PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
PFNGLCREATESTATESNVPROC glCreateStatesNV;
PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCLeventARB;
PFNGLCREATETEXTURESPROC glCreateTextures;
PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
PFNGLCULLFACEPROC glCullFace;
PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
PFNGLDELETEBUFFERSPROC glDeleteBuffers;
PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
PFNGLDELETEPATHSNVPROC glDeletePathsNV;
PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
PFNGLDELETEPROGRAMPROC glDeleteProgram;
```

```
PFNGLDELETEQUERIESPROC glDeleteQueries;
PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
PFNGLDELETESAMPLERSPROC glDeleteSamplers;
PFNGLDELETESHADERPROC glDeleteShader;
PFNGLDELETESTATESNVPROC glDeleteStatesNV;
PFNGLDELETESYNCPROC glDeleteSync;
PFNGLDELETETEXTURESPROC glDeleteTextures;
PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
PFNGLDEPTHFUNCPROC glDepthFunc;
PFNGLDEPTHMASKPROC glDepthMask;
PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
PFNGLDEPTHRANGEFPROC glDepthRangef;
PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
PFNGLDEPTHRANGEPROC glDepthRange;
PFNGLDETACHSHADERPROC glDetachShader;
PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
PFNGLDISABLEIPROC glDisablei;
PFNGLDISABLEPROC glDisable;
PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexArrayAttribEXT;
PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexArrayAttrib;
PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexArrayEXT;
PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArray;
PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
PFNGLDISPATCHCOMPUTEINDIRECTPROC glDispatchComputeIndirect;
PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
PFNGLDRAWARRAYSPROC glDrawArrays;
PFNGLDRAWBUFFERPROC glDrawBuffer;
PFNGLDRAWBUFFERSPROC glDrawBuffers;
PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC glDrawElementsInstancedBaseVertexBaseInstance;
PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
PFNGLDRAWELEMENTSPROC glDrawElements;
PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
PFNGLENABLEIPROC glEnablei;
PFNGLENABLEPROC glEnable;
PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexArrayAttribEXT;
PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexArrayAttrib;
PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexArrayEXT;
PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArray;
PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
PFNGLENDQUERYPROC glEndQuery;
PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
PFNGLFENCESYNCPROC glFenceSync;
PFNGLFINISHPROC glFinish;
PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
PFNGLFLUSHPROC glFlush;
PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
```

```
PFNGLFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glFramebufferSampleLocationsfvARB;
PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
PFNGLFRONTFACEPROC glFrontFace;
PFNGLGENBUFFERSPROC glGenBuffers;
PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
PFNGLGENPATHSNVPROC glGenPathsNV;
PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
PFNGLGENQUERIESPROC glGenQueries;
PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
PFNGLGENSAMPLERSPROC glGenSamplers;
PFNGLGENTEXTURESPROC glGenTextures;
PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
PFNGLGETBOOLEANVPROC glGetBooleanv;
PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
PFNGLGETBUFFERPARAMETERUI64VNVPROC glGetBufferParameterui64vNV;
PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
PFNGLGETDOUBLEVPROC glGetDoublev;
PFNGLGETERRORPROC glGetError;
PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
PFNGLGETFLOATI_VPROC glGetFloati_v;
PFNGLGETFLOATVPROC glGetFloatv;
PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
PFNGLGETINTEGER64VPROC glGetInteger64v;
PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
PFNGLGETINTEGERUI64VNVPROC glGetIntegerui64vNV;
PFNGLGETINTEGERVPROC glGetIntegerv;
PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
```

```
PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGendvEXT;
PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIivEXT;
PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
PFNGLGETMULTITEXPARAMETERIVEXTPROC glGetMultiTexParameterivEXT;
PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC glGetNamedBufferParameterui64vNV;
PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC glGetNamedFramebufferAttachmentParameterivEXT;
PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIivEXT;
PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
PFNGLGETNTEXIMAGEPROC glGetnTexImage;
PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
PFNGLGETNUNIFORMDVPROC glGetnUniformdv;
PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
PFNGLGETOBJECTLABELPROC glGetObjectLabel;
PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
PFNGLGETPATHMETRICRANGENVPROC glGetPathMetricRangeNV;
PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
PFNGLGETPOINTERI_VEXTPROC glGetPointeri_vEXT;
PFNGLGETPOINTERVPROC glGetPointerv;
PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
PFNGLGETPROGRAMIVPROC glGetProgramiv;
PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
PFNGLGETPROGRAMRESOURCEFVNVPROC glGetProgramResourcefvNV;
PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC glGetProgramResourceLocationIndex;
PFNGLGETPROGRAMRESOURCELOCATIONPROC glGetProgramResourceLocation;
PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
```

```
PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
PFNGLGETQUERYIVPROC glGetQueryiv;
PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
PFNGLGETSHADERIVPROC glGetShaderiv;
PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
PFNGLGETSHADERSOURCEPROC glGetShaderSource;
PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
PFNGLGETSTRINGIPROC glGetStringi;
PFNGLGETSTRINGPROC glGetString;
PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
PFNGLGETSYNCIVPROC glGetSynciv;
PFNGLGETTEXIMAGEPROC glGetTexImage;
PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
PFNGLGETTRANSFORMFEEDBACKI64_VPROC glGetTransformFeedbacki64_v;
PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
PFNGLGETUNIFORMDVPROC glGetUniformdv;
PFNGLGETUNIFORMFVPROC glGetUniformfv;
PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
PFNGLGETUNIFORMIVPROC glGetUniformiv;
PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribdv;
PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribfv;
PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribIiv;
PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribIuiv;
PFNGLGETVERTEXATTRIBIVPROC glGetVertexAttribiv;
PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLdv;
PFNGLGETVERTEXATTRIBLI64VNVPROC glGetVertexAttribLi64vNV;
PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribLui64vARB;
PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribLui64vNV;
PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribPointerv;
PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrNV;
PFNGLHINTPROC glHint;
PFNGLINDEXFORMATNVPROC glIndexFormatNV;
PFNGLINSERTEVENTMARKEREXTPROC glInsertEventMarkerEXT;
```

```
PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
PFNGLISBUFFERPROC glIsBuffer;
PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
PFNGLISENABLEDIPROC glIsEnabledi;
PFNGLISENABLEDPROC glIsEnabled;
PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
PFNGLISPATHNVPROC glIsPathNV;
PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
PFNGLISPROGRAMPROC glIsProgram;
PFNGLISQUERYPROC glIsQuery;
PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
PFNGLISSAMPLERPROC glIsSampler;
PFNGLISSHADERPROC glIsShader;
PFNGLISSTATENVPROC glIsStateNV;
PFNGLISSYNCPROC glIsSync;
PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
PFNGLISTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
PFNGLISTEXTUREPROC glIsTexture;
PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
PFNGLISVERTEXARRAYPROC glIsVertexArray;
PFNGLLABELOBJECTEXTPROC glLabelObjectEXT;
PFNGLLINEWIDTHPROC glLineWidth;
PFNGLLINKPROGRAMPROC glLinkProgram;
PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
PFNGLLOGICOPPROC glLogicOp;
PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
PFNGLMAPBUFFERPROC glMapBuffer;
PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
PFNGLMATRIXLOADDEXTPROC glMatrixLoaddEXT;
PFNGLMATRIXLOADFEXTPROC glMatrixLoadfEXT;
PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedEXT;
PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
PFNGLMATRIXMULTDEXTPROC glMatrixMultdEXT;
PFNGLMATRIXMULTFEXTPROC glMatrixMultfEXT;
PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefEXT;
PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
PFNGLMATRIXPUSHEXTPROC glMatrixPushEXT;
PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedEXT;
PFNGLMATRIXROTATEFEXTPROC glMatrixRotatefEXT;
PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
PFNGLMEMORYBARRIERPROC glMemoryBarrier;
```

```
PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC glMultiDrawArraysIndirectCountARB;
PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC glMultiDrawElementsIndirectCountARB;
PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
PFNGLMULTITEXCOORDPOINTEREXTPROC glMultiTexCoordPointerEXT;
PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
PFNGLMULTITEXENVIEXTPROC glMultiTexEnviEXT;
PFNGLMULTITEXENVIVEXTPROC glMultiTexEnvivEXT;
PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
PFNGLMULTITEXGENDVEXTPROC glMultiTexGendvEXT;
PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameteriEXT;
PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubDataEXT;
PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC glNamedFramebufferParameteri;
PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glNamedFramebufferSampleLocationsfvARB;
PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
PFNGLNAMEDFRAMEBUFFERTEXTUREPROC glNamedFramebufferTexture;
PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uivEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC
       glNamedRenderbufferStorageMultisampleCoverageEXT;
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
PFNGLNORMALFORMATNVPROC glNormalFormatNV;
PFNGLOBJECTLABELPROC glObjectLabel;
PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
```

```
PFNGLPATCHPARAMETERIPROC glPatchParameteri;
PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
PFNGLPATHCOORDSNVPROC glPathCoordsNV;
PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
PFNGLPATHGLYPHINDEXARRAYNVPROC glPathGlyphIndexArrayNV;
PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
PFNGLPATHGLYPHRANGENVPROC glPathGlyphRangeNV;
PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
PFNGLPATHPARAMETERFNVPROC glPathParameterfNV;
PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
PFNGLPATHSTRINGNVPROC glPathStringNV;
PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
PFNGLPIXELSTOREFPROC glPixelStoref;
PFNGLPIXELSTOREIPROC glPixelStorei;
PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
PFNGLPOINTPARAMETERFPROC glPointParameterf;
PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
PFNGLPOINTPARAMETERIPROC glPointParameteri;
PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
PFNGLPOINTSIZEPROC glPointSize;
PFNGLPOLYGONMODEPROC glPolygonMode;
PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
PFNGLPOLYGONOFFSETPROC glPolygonOffset;
PFNGLPOPDEBUGGROUPPROC glPopDebugGroup;
PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
PFNGLPROGRAMBINARYPROC glProgramBinary;
PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dv;
PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;
PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniform1iEXT;
PFNGLPROGRAMUNIFORM1IPROC glProgramUniform1i;
PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniform1ui64NV;
PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
```

```
PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dvEXT;
PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dv;
PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dv;
PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dvEXT;
PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dv;
PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dv;
PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dv;
PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
```

```
PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
PFNGLPROGRAMUNIFORMUI64VNVPROC glProgramUniformui64vNV;
PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC glPushClientAttribDefaultEXT;
PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
PFNGLQUERYCOUNTERPROC glQueryCounter;
PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
PFNGLREADBUFFERPROC glReadBuffer;
PFNGLREADNPIXELSARBPROC glReadnPixelsARB;
PFNGLREADNPIXELSPROC glReadnPixels;
PFNGLREADPIXELSPROC glReadPixels;
PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
PFNGLSAMPLEMASKIPROC glSampleMaski;
PFNGLSAMPLERPARAMETERFPROC glSamplerParameterf;
PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIiv;
PFNGLSAMPLERPARAMETERIPROC glSamplerParameteri;
PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
PFNGLSAMPLERPARAMETERIVPROC glSamplerParameteriv;
PFNGLSCISSORARRAYVPROC glScissorArrayv;
PFNGLSCISSORINDEXEDPROC glScissorIndexed;
PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
PFNGLSCISSORPROC glScissor;
PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
PFNGLSHADERBINARYPROC glShaderBinary;
PFNGLSHADERSOURCEPROC glShaderSource;
PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
PFNGLSIGNALVKFENCENVPROC glSignalVkFenceNV;
PFNGLSIGNALVKSEMAPHORENVPROC glSignalVkSemaphoreNV;
PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
PFNGLSTATECAPTURENVPROC glStateCaptureNV;
PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
PFNGLSTENCILFUNCPROC glStencilFunc;
PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
PFNGLSTENCILMASKPROC glStencilMask;
PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
PFNGLSTENCILOPPROC glStencilOp;
PFNGLSTENCILOPSEPARATEPROC glStencilOpSeparate;
PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
PFNGLTEXBUFFERARBPROC glTexBufferARB;
PFNGLTEXBUFFERPROC glTexBuffer;
PFNGLTEXBUFFERRANGEPROC glTexBufferRange;
PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
PFNGLTEXIMAGE1DPROC glTexImage1D;
PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
PFNGLTEXIMAGE2DPROC glTexImage2D;
PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
PFNGLTEXIMAGE3DPROC glTexImage3D;
PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
PFNGLTEXPARAMETERFPROC glTexParameterf;
PFNGLTEXPARAMETERFVPROC glTexParameterfv;
PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
PFNGLTEXPARAMETERIPROC glTexParameteri;
PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
PFNGLTEXPARAMETERIVPROC glTexParameteriv;
PFNGLTEXSTORAGE1DPROC glTexStorage1D;
PFNGLTEXSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
PFNGLTEXSTORAGE2DPROC glTexStorage2D;
PFNGLTEXSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
PFNGLTEXSTORAGE3DPROC glTexStorage3D;
PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
PFNGLTEXTUREBARRIERPROC glTextureBarrier;
PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
PFNGLTEXTUREBUFFERPROC glTextureBuffer;
PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
```

```
PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
PFNGLTEXTUREPAGECOMMITMENTEXTPROC glTexturePageCommitmentEXT;
PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIivEXT;
PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiv;
PFNGLTEXTUREPARAMETERIPROC glTextureParameteri;
PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterivEXT;
PFNGLTEXTUREPARAMETERIVPROC glTextureParameteriv;
PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
PFNGLTEXTURESTORAGE1DEXTPROC glTextureStorage1DEXT;
PFNGLTEXTURESTORAGE1DPROC glTextureStorage1D;
PFNGLTEXTURESTORAGE2DEXTPROC glTextureStorage2DEXT;
PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
PFNGLTEXTURESTORAGE2DPROC glTextureStorage2D;
PFNGLTEXTURESTORAGE3DEXTPROC glTextureStorage3DEXT;
PFNGLTEXTURESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
PFNGLTEXTURESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
PFNGLTEXTURESTORAGE3DPROC glTextureStorage3D;
PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
PFNGLTEXTUREVIEWPROC glTextureView;
PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
PFNGLUNIFORM1DPROC glUniform1d;
PFNGLUNIFORM1DVPROC glUniform1dv;
PFNGLUNIFORM1FPROC glUniform1f;
PFNGLUNIFORM1FVPROC glUniform1fv;
PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
PFNGLUNIFORM1IPROC glUniform1i;
PFNGLUNIFORM1IVPROC glUniform1iv;
PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
PFNGLUNIFORM1UIPROC glUniform1ui;
PFNGLUNIFORM1UIVPROC glUniform1uiv;
PFNGLUNIFORM2DPROC glUniform2d;
PFNGLUNIFORM2DVPROC glUniform2dv;
PFNGLUNIFORM2FPROC glUniform2f;
PFNGLUNIFORM2FVPROC glUniform2fv;
PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
PFNGLUNIFORM2IPROC glUniform2i;
PFNGLUNIFORM2IVPROC glUniform2iv;
PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
PFNGLUNIFORM2UIPROC glUniform2ui;
PFNGLUNIFORM2UIVPROC glUniform2uiv;
PFNGLUNIFORM3DPROC glUniform3d;
PFNGLUNIFORM3DVPROC glUniform3dv;
PFNGLUNIFORM3FPROC glUniform3f;
PFNGLUNIFORM3FVPROC glUniform3fv;
PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
PFNGLUNIFORM3IPROC glUniform3i;
PFNGLUNIFORM3IVPROC glUniform3iv;
PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
PFNGLUNIFORM3UIPROC glUniform3ui;
PFNGLUNIFORM3UIVPROC glUniform3uiv;
PFNGLUNIFORM4DPROC glUniform4d;
PFNGLUNIFORM4DVPROC glUniform4dv;
PFNGLUNIFORM4FPROC glUniform4f;
```

```
PFNGLUNIFORM4FVPROC glUniform4fv;
PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
PFNGLUNIFORM4IPROC glUniform4i;
PFNGLUNIFORM4IVPROC glUniform4iv;
PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
PFNGLUNIFORM4UIPROC glUniform4ui;
PFNGLUNIFORM4UIVPROC glUniform4uiv;
PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
PFNGLUNIFORMMATRIX2DVPROC glUniformMatrix2dv;
PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
PFNGLUNIFORMMATRIX3DVPROC glUniformMatrix3dv;
PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
PFNGLUNIFORMUI64NVPROC glUniformui64NV;
PFNGLUNIFORMUI64VNVPROC glUniformui64vNV;
PFNGLUNMAPBUFFERPROC glUnmapBuffer;
PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
PFNGLUSEPROGRAMPROC glUseProgram;
PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
PFNGLUSESHADERPROGRAMEXTPROC glUseShaderProgramEXT;
PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
PFNGLVALIDATEPROGRAMPROC glValidateProgram;
PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
PFNGLVERTEXARRAYATTRIBIFORMATPROC glVertexArrayAttribIFormat;
PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
PFNGLVERTEXATTRIB2SPROC glVertexAttrib2s;
PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
```

```
PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4Nbv;
PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4Niv;
PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4Nsv;
PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4Nub;
PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4Nubv;
PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4Nuiv;
PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4Nusv;
PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
PFNGLVERTEXATTRIBI2IVPROC glVertexAttribI2iv;
PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
PFNGLVERTEXATTRIBL1I64VNVPROC glVertexAttribL1i64vNV;
PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribL1ui64vNV;
PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
PFNGLVERTEXATTRIBL2I64VNVPROC glVertexAttribL2i64vNV;
PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
PFNGLVERTEXATTRIBL2UI64VNVPROC glVertexAttribL2ui64vNV;
PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
PFNGLVERTEXATTRIBL3I64VNVPROC glVertexAttribL3i64vNV;
PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
PFNGLVERTEXATTRIBL3UI64VNVPROC glVertexAttribL3ui64vNV;
PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
PFNGLVERTEXATTRIBL4I64VNVPROC glVertexAttribL4i64vNV;
PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
PFNGLVERTEXATTRIBL4UI64VNVPROC glVertexAttribL4ui64vNV;
PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
```

```
PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
PFNGLVIEWPORTARRAYVPROC glViewportArrayv;
PFNGLVIEWPORTINDEXEDFPROC glViewportIndexedf;
PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;
PFNGLVIEWPORTPOSITIONWSCALENVPROC glViewportPositionWScaleNV;
PFNGLVIEWPORTPROC glViewport;
PFNGLVIEWPORTSWIZZLENVPROC glViewportSwizzleNV;
PFNGLWAITSYNCPROC glWaitSync;
PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
#endif


GLint gg::ggBufferAlignment(0);

//
// ゲームグラフィックス特論の都合にもとづく初期化
//
void gg::ggInit()
{
  // 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
  if (ggBufferAlignment) return;

  // macOS 以外で OpenGL 3.2 以降の API を取得する
#if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
  glActiveProgramEXT = PFNGLACTIVEPROGRAMEXTPROC(glfwGetProcAddress("glActiveProgramEXT"));
  glActiveShaderProgram = PFNGLACTIVESHADERPROGRAMPROC(glfwGetProcAddress("glActiveShaderProgram"));
  glActiveTexture = PFNGLACTIVETEXTUREPROC(glfwGetProcAddress("glActiveTexture"));
  glApplyFramebufferAttachmentCMAAINTEL = PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC(glfwGetProcAddress("glApplyFramebufferAttachmentCMAAINTEL"));
  glAttachShader = PFNGLATTACHSHADERPROC(glfwGetProcAddress("glAttachShader"));
  glBeginConditionalRender = PFNGLBEGINCONDITIONALRENDERPROC(glfwGetProcAddress("glBeginConditionalRender"));
  glBeginConditionalRenderNV = PFNGLBEGINCONDITIONALRENDERNVPROC(glfwGetProcAddress("glBeginConditionalRenderNV"));
  glBeginPerfMonitorAMD = PFNGLBEGINPERFMONITORAMDPROC(glfwGetProcAddress("glBeginPerfMonitorAMD"));
  glBeginPerfQueryINTEL = PFNGLBEGINPERFQUERYINTELPROC(glfwGetProcAddress("glBeginPerfQueryINTEL"));
  glBeginQuery = PFNGLBEGINQUERYPROC(glfwGetProcAddress("glBeginQuery"));
  glBeginQueryIndexed = PFNGLBEGINQUERYINDEXEDPROC(glfwGetProcAddress("glBeginQueryIndexed"));
  glBeginTransformFeedback = PFNGLBEGINTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBeginTransformFeedback"));
  glBindAttribLocation = PFNGLBINDATTRIBLOCATIONPROC(glfwGetProcAddress("glBindAttribLocation"));
  glBindBuffer = PFNGLBINDBUFFERPROC(glfwGetProcAddress("glBindBuffer"));
  glBindBufferBase = PFNGLBINDBUFFERBASEPROC(glfwGetProcAddress("glBindBufferBase"));
  glBindBufferRange = PFNGLBINDBUFFERRANGEPROC(glfwGetProcAddress("glBindBufferRange"));
  glBindBuffersBase = PFNGLBINDBUFFERSBASEPROC(glfwGetProcAddress("glBindBuffersBase"));
  glBindBuffersRange = PFNGLBINDBUFFERSRANGEPROC(glfwGetProcAddress("glBindBuffersRange"));
  glBindFragDataLocation = PFNGLBINDFRAGDATALOCATIONPROC(glfwGetProcAddress("glBindFragDataLocation"));
  glBindFragDataLocationIndexed = PFNGLBINDFRAGDATALOCATIONINDEXEDPROC(glfwGetProcAddress("glBindFragDataLocationIndexed"));
  glBindFramebuffer = PFNGLBINDFRAMEBUFFERPROC(glfwGetProcAddress("glBindFramebuffer"));
  glBindImageTexture = PFNGLBINDIMAGETEXTUREPROC(glfwGetProcAddress("glBindImageTexture"));
  glBindImageTextures = PFNGLBINDIMAGETEXTURESPROC(glfwGetProcAddress("glBindImageTextures"));
  glBindMultiTextureEXT = PFNGLBINDMULTITEXTUREEXTPROC(glfwGetProcAddress("glBindMultiTextureEXT"));
  glBindProgramPipeline = PFNGLBINDPROGRAMPIPELINEPROC(glfwGetProcAddress("glBindProgramPipeline"));
  glBindRenderbuffer = PFNGLBINDRENDERBUFFERPROC(glfwGetProcAddress("glBindRenderbuffer"));
  glBindSampler = PFNGLBINDSAMPLERPROC(glfwGetProcAddress("glBindSampler"));
  glBindSamplers = PFNGLBINDSAMPLERSPROC(glfwGetProcAddress("glBindSamplers"));
  glBindTexture = PFNGLBINDTEXTUREPROC(glfwGetProcAddress("glBindTexture"));
  glBindTextureUnit = PFNGLBINDTEXTUREUNITPROC(glfwGetProcAddress("glBindTextureUnit"));
  glBindTextures = PFNGLBINDTEXTURESPROC(glfwGetProcAddress("glBindTextures"));
  glBindTransformFeedback = PFNGLBINDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glBindTransformFeedback"));
  glBindVertexArray = PFNGLBINDVERTEXARRAYPROC(glfwGetProcAddress("glBindVertexArray"));
  glBindVertexBuffer = PFNGLBINDVERTEXBUFFERPROC(glfwGetProcAddress("glBindVertexBuffer"));
  glBindVertexBuffers = PFNGLBINDVERTEXBUFFERSPROC(glfwGetProcAddress("glBindVertexBuffers"));
  glBlendBarrierKHR = PFNGLBLENDBARRIERKHRPROC(glfwGetProcAddress("glBlendBarrierKHR"));
  glBlendBarrierNV = PFNGLBLENDBARRIERNVPROC(glfwGetProcAddress("glBlendBarrierNV"));
  glBlendColor = PFNGLBLENDCOLORPROC(glfwGetProcAddress("glBlendColor"));
  glBlendEquation = PFNGLBLENDEQUATIONPROC(glfwGetProcAddress("glBlendEquation"));
  glBlendEquationSeparate = PFNGLBLENDEQUATIONSEPARATEPROC(glfwGetProcAddress("glBlendEquationSeparate"));
  glBlendEquationSeparatei = PFNGLBLENDEQUATIONSEPARATEIPROC(glfwGetProcAddress("glBlendEquationSeparatei"));
  glBlendEquationSeparateiARB = PFNGLBLENDEQUATIONSEPARATEIARBPROC(glfwGetProcAddress("glBlendEquationSeparateiARB"));
  glBlendEquationi = PFNGLBLENDEQUATIONIPROC(glfwGetProcAddress("glBlendEquationi"));
  glBlendEquationiARB = PFNGLBLENDEQUATIONIARBPROC(glfwGetProcAddress("glBlendEquationiARB"));
  glBlendFunc = PFNGLBLENDFUNCPROC(glfwGetProcAddress("glBlendFunc"));
  glBlendFuncSeparate = PFNGLBLENDFUNCSEPARATEPROC(glfwGetProcAddress("glBlendFuncSeparate"));
  glBlendFuncSeparatei = PFNGLBLENDFUNCSEPARATEIPROC(glfwGetProcAddress("glBlendFuncSeparatei"));
  glBlendFuncSeparateiARB =
```

```
PFNGLBLENDFUNCSEPARATEIARBPROC(glfwGetProcAddress("glBlendFuncSeparateiARB"));
 glBlendFunci = PFNGLBLENDFUNCIPROC(glfwGetProcAddress("glBlendFunci"));
 glBlendFunciARB = PFNGLBLENDFUNCIARBPROC(glfwGetProcAddress("glBlendFunciARB"));
 glBlendParameteriNV = PFNGLBLENDPARAMETERINVPROC(glfwGetProcAddress("glBlendParameteriNV"));
 glBlitFramebuffer = PFNGLBLITFRAMEBUFFERPROC(glfwGetProcAddress("glBlitFramebuffer"));
 glBlitNamedFramebuffer =
PFNGLBLITNAMEDFRAMEBUFFERPROC(glfwGetProcAddress("glBlitNamedFramebuffer"));
 glBufferAddressRangeNV =
PFNGLBUFFERADDRESSRANGENVPROC(glfwGetProcAddress("glBufferAddressRangeNV"));
 glBufferData = PFNGLBUFFERDATAPROC(glfwGetProcAddress("glBufferData"));
 glBufferPageCommitmentARB =
PFNGLBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glBufferPageCommitmentARB"));
 glBufferStorage = PFNGLBUFFERSTORAGEPROC(glfwGetProcAddress("glBufferStorage"));
 glBufferSubData = PFNGLBUFFERSUBDATAPROC(glfwGetProcAddress("glBufferSubData"));
 glCallCommandListNV = PFNGLCALLCOMMANDLISTNVPROC(glfwGetProcAddress("glCallCommandListNV"));
 glCheckFramebufferStatus =
PFNGLCHECKFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckFramebufferStatus"));
 glCheckNamedFramebufferStatus =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC(glfwGetProcAddress("glCheckNamedFramebufferStatus"));
 glCheckNamedFramebufferStatusEXT =
PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC(glfwGetProcAddress("glCheckNamedFramebufferStatusEXT"));
 glClampColor = PFNGLCLAMPCOLORPROC(glfwGetProcAddress("glClampColor"));
 glClear = PFNGLCLEARPROC(glfwGetProcAddress("glClear"));
 glClearBufferData = PFNGLCLEARBUFFERDATAPROC(glfwGetProcAddress("glClearBufferData"));
 glClearBufferSubData = PFNGLCLEARBUFFERSUBDATAPROC(glfwGetProcAddress("glClearBufferSubData"));
 glClearBufferfi = PFNGLCLEARBUFFERFIPROC(glfwGetProcAddress("glClearBufferfi"));
 glClearBufferfv = PFNGLCLEARBUFFERFVPROC(glfwGetProcAddress("glClearBufferfv"));
 glClearBufferiv = PFNGLCLEARBUFFERIVPROC(glfwGetProcAddress("glClearBufferiv"));
 glClearBufferuiv = PFNGLCLEARBUFFERUIVPROC(glfwGetProcAddress("glClearBufferuiv"));
 glClearColor = PFNGLCLEARCOLORPROC(glfwGetProcAddress("glClearColor"));
 glClearDepth = PFNGLCLEARDEPTHPROC(glfwGetProcAddress("glClearDepth"));
 glClearDepthf = PFNGLCLEARDEPTHFPROC(glfwGetProcAddress("glClearDepthf"));
 glClearNamedBufferData =
PFNGLCLEARNAMEDBUFFERDATAPROC(glfwGetProcAddress("glClearNamedBufferData"));
 glClearNamedBufferDataEXT =
PFNGLCLEARNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferDataEXT"));
 glClearNamedBufferSubData =
PFNGLCLEARNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glClearNamedBufferSubData"));
 glClearNamedBufferSubDataEXT =
PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glClearNamedBufferSubDataEXT"));
 glClearNamedFramebufferfi =
PFNGLCLEARNAMEDFRAMEBUFFERFIPROC(glfwGetProcAddress("glClearNamedFramebufferfi"));
 glClearNamedFramebufferfv =
PFNGLCLEARNAMEDFRAMEBUFFERFVPROC(glfwGetProcAddress("glClearNamedFramebufferfv"));
 glClearNamedFramebufferiv =
PFNGLCLEARNAMEDFRAMEBUFFERIVPROC(glfwGetProcAddress("glClearNamedFramebufferiv"));
 glClearNamedFramebufferuiv =
PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC(glfwGetProcAddress("glClearNamedFramebufferuiv"));
 glClearStencil = PFNGLCLEARSTENCILPROC(glfwGetProcAddress("glClearStencil"));
 glClearTexImage = PFNGLCLEARTEXIMAGEPROC(glfwGetProcAddress("glClearTexImage"));
 glClearTexSubImage = PFNGLCLEARTEXSUBIMAGEPROC(glfwGetProcAddress("glClearTexSubImage"));
 glClientAttribDefaultEXT =
PFNGLCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glClientAttribDefaultEXT"));
 glClientWaitSync = PFNGLCLIENTWAITSYNCPROC(glfwGetProcAddress("glClientWaitSync"));
 glClipControl = PFNGLCLIPCONTROLPROC(glfwGetProcAddress("glClipControl"));
 glColorFormatNV = PFNGLCOLORFORMATNVPROC(glfwGetProcAddress("glColorFormatNV"));
 glColorMask = PFNGLCOLORMASKPROC(glfwGetProcAddress("glColorMask"));
 glColorMaski = PFNGLCOLORMASKIPROC(glfwGetProcAddress("glColorMaski"));
 glCommandListSegmentsNV =
PFNGLCOMMANDLISTSEGMENTSNVPROC(glfwGetProcAddress("glCommandListSegmentsNV"));
 glCompileCommandListNV =
PFNGLCOMPILECOMMANDLISTNVPROC(glfwGetProcAddress("glCompileCommandListNV"));
 glCompileShader = PFNGLCOMPILESHADERPROC(glfwGetProcAddress("glCompileShader"));
 glCompileShaderIncludeARB =
PFNGLCOMPILESHADERINCLUDEARBPROC(glfwGetProcAddress("glCompileShaderIncludeARB"));
 glCompressedMultiTexImage1DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage1DEXT"));
 glCompressedMultiTexImage2DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage2DEXT"));
 glCompressedMultiTexImage3DEXT =
PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexImage3DEXT"));
 glCompressedMultiTexSubImage1DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage1DEXT"));
 glCompressedMultiTexSubImage2DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage2DEXT"));
 glCompressedMultiTexSubImage3DEXT =
PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedMultiTexSubImage3DEXT"));
 glCompressedTexImage1D =
PFNGLCOMPRESSEDTEXIMAGE1DPROC(glfwGetProcAddress("glCompressedTexImage1D"));
 glCompressedTexImage2D =
PFNGLCOMPRESSEDTEXIMAGE2DPROC(glfwGetProcAddress("glCompressedTexImage2D"));
 glCompressedTexImage3D =
PFNGLCOMPRESSEDTEXIMAGE3DPROC(glfwGetProcAddress("glCompressedTexImage3D"));
 glCompressedTexSubImage1D =
PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTexSubImage1D"));
 glCompressedTexSubImage2D =
PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTexSubImage2D"));
```

```
glCompressedTexSubImage3D = PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTexSubImage3D"));
glCompressedTextureImage1DEXT = PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureImage1DEXT"));
glCompressedTextureImage2DEXT = PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureImage2DEXT"));
glCompressedTextureImage3DEXT = PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureImage3DEXT"));
glCompressedTextureSubImage1D = PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCompressedTextureSubImage1D"));
glCompressedTextureSubImage1DEXT = PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage1DEXT"));
glCompressedTextureSubImage2D = PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCompressedTextureSubImage2D"));
glCompressedTextureSubImage2DEXT = PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage2DEXT"));
glCompressedTextureSubImage3D = PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCompressedTextureSubImage3D"));
glCompressedTextureSubImage3DEXT = PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCompressedTextureSubImage3DEXT"));
glConservativeRasterParameterfNV = PFNGLCONSERVATIVERASTERPARAMETERFNVPROC(glfwGetProcAddress("glConservativeRasterParameterfNV"));
glConservativeRasterParameteriNV = PFNGLCONSERVATIVERASTERPARAMETERINVPROC(glfwGetProcAddress("glConservativeRasterParameteriNV"));
glCopyBufferSubData = PFNGLCOPYBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyBufferSubData"));
glCopyImageSubData = PFNGLCOPYIMAGESUBDATAPROC(glfwGetProcAddress("glCopyImageSubData"));
glCopyMultiTexImage1DEXT = PFNGLCOPYMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage1DEXT"));
glCopyMultiTexImage2DEXT = PFNGLCOPYMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexImage2DEXT"));
glCopyMultiTexSubImage1DEXT = PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage1DEXT"));
glCopyMultiTexSubImage2DEXT = PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage2DEXT"));
glCopyMultiTexSubImage3DEXT = PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyMultiTexSubImage3DEXT"));
glCopyNamedBufferSubData = PFNGLCOPYNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glCopyNamedBufferSubData"));
glCopyPathNV = PFNGLCOPYPATHNVPROC(glfwGetProcAddress("glCopyPathNV"));
glCopyTexImage1D = PFNGLCOPYTEXIMAGE1DPROC(glfwGetProcAddress("glCopyTexImage1D"));
glCopyTexImage2D = PFNGLCOPYTEXIMAGE2DPROC(glfwGetProcAddress("glCopyTexImage2D"));
glCopyTexSubImage1D = PFNGLCOPYTEXSUBIMAGE1DPROC(glfwGetProcAddress("glCopyTexSubImage1D"));
glCopyTexSubImage2D = PFNGLCOPYTEXSUBIMAGE2DPROC(glfwGetProcAddress("glCopyTexSubImage2D"));
glCopyTexSubImage3D = PFNGLCOPYTEXSUBIMAGE3DPROC(glfwGetProcAddress("glCopyTexSubImage3D"));
glCopyTextureImage1DEXT = PFNGLCOPYTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureImage1DEXT"));
glCopyTextureImage2DEXT = PFNGLCOPYTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureImage2DEXT"));
glCopyTextureSubImage1D = PFNGLCOPYTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glCopyTextureSubImage1D"));
glCopyTextureSubImage1DEXT = PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage1DEXT"));
glCopyTextureSubImage2D = PFNGLCOPYTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glCopyTextureSubImage2D"));
glCopyTextureSubImage2DEXT = PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage2DEXT"));
glCopyTextureSubImage3D = PFNGLCOPYTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glCopyTextureSubImage3D"));
glCopyTextureSubImage3DEXT = PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glCopyTextureSubImage3DEXT"));
glCoverFillPathInstancedNV = PFNGLCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverFillPathInstancedNV"));
glCoverFillPathNV = PFNGLCOVERFILLPATHNVPROC(glfwGetProcAddress("glCoverFillPathNV"));
glCoverStrokePathInstancedNV = PFNGLCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glCoverStrokePathInstancedNV"));
glCoverStrokePathNV = PFNGLCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glCoverStrokePathNV"));
glCoverageModulationNV = PFNGLCOVERAGEMODULATIONNVPROC(glfwGetProcAddress("glCoverageModulationNV"));
glCoverageModulationTableNV = PFNGLCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glCoverageModulationTableNV"));
glCreateBuffers = PFNGLCREATEBUFFERSPROC(glfwGetProcAddress("glCreateBuffers"));
glCreateCommandListsNV = PFNGLCREATECOMMANDLISTSNVPROC(glfwGetProcAddress("glCreateCommandListsNV"));
glCreateFramebuffers = PFNGLCREATEFRAMEBUFFERSPROC(glfwGetProcAddress("glCreateFramebuffers"));
glCreatePerfQueryINTEL = PFNGLCREATEPERFQUERYINTELPROC(glfwGetProcAddress("glCreatePerfQueryINTEL"));
glCreateProgram = PFNGLCREATEPROGRAMPROC(glfwGetProcAddress("glCreateProgram"));
glCreateProgramPipelines = PFNGLCREATEPROGRAMPIPELINESPROC(glfwGetProcAddress("glCreateProgramPipelines"));
glCreateQueries = PFNGLCREATEQUERIESPROC(glfwGetProcAddress("glCreateQueries"));
glCreateRenderbuffers = PFNGLCREATERENDERBUFFERSPROC(glfwGetProcAddress("glCreateRenderbuffers"));
glCreateSamplers = PFNGLCREATESAMPLERSPROC(glfwGetProcAddress("glCreateSamplers"));
glCreateShader = PFNGLCREATESHADERPROC(glfwGetProcAddress("glCreateShader"));
glCreateShaderProgramEXT = PFNGLCREATESHADERPROGRAMEXTPROC(glfwGetProcAddress("glCreateShaderProgramEXT"));
glCreateShaderProgramv = PFNGLCREATESHADERPROGRAMVPROC(glfwGetProcAddress("glCreateShaderProgramv"));
```

```
glCreateStatesNV = PFNGLCREATESTATESNVPROC(glfwGetProcAddress("glCreateStatesNV"));
glCreateSyncFromCLeventARB = PFNGLCREATESYNCFROMCLEVENTARBPROC(glfwGetProcAddress("glCreateSyncFromCLeventARB"));
glCreateTextures = PFNGLCREATETEXTURESPROC(glfwGetProcAddress("glCreateTextures"));
glCreateTransformFeedbacks = PFNGLCREATETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glCreateTransformFeedbacks"));
glCreateVertexArrays = PFNGLCREATEVERTEXARRAYSPROC(glfwGetProcAddress("glCreateVertexArrays"));
glCullFace = PFNGLCULLFACEPROC(glfwGetProcAddress("glCullFace"));
glDebugMessageCallback = PFNGLDEBUGMESSAGECALLBACKPROC(glfwGetProcAddress("glDebugMessageCallback"));
glDebugMessageCallbackARB = PFNGLDEBUGMESSAGECALLBACKARBPROC(glfwGetProcAddress("glDebugMessageCallbackARB"));
glDebugMessageControl = PFNGLDEBUGMESSAGECONTROLPROC(glfwGetProcAddress("glDebugMessageControl"));
glDebugMessageControlARB = PFNGLDEBUGMESSAGECONTROLARBPROC(glfwGetProcAddress("glDebugMessageControlARB"));
glDebugMessageInsert = PFNGLDEBUGMESSAGEINSERTPROC(glfwGetProcAddress("glDebugMessageInsert"));
glDebugMessageInsertARB = PFNGLDEBUGMESSAGEINSERTARBPROC(glfwGetProcAddress("glDebugMessageInsertARB"));
glDeleteBuffers = PFNGLDELETEBUFFERSPROC(glfwGetProcAddress("glDeleteBuffers"));
glDeleteCommandListsNV = PFNGLDELETECOMMANDLISTSNVPROC(glfwGetProcAddress("glDeleteCommandListsNV"));
glDeleteFramebuffers = PFNGLDELETEFRAMEBUFFERSPROC(glfwGetProcAddress("glDeleteFramebuffers"));
glDeleteNamedStringARB = PFNGLDELETENAMEDSTRINGARBPROC(glfwGetProcAddress("glDeleteNamedStringARB"));
glDeletePathsNV = PFNGLDELETEPATHSNVPROC(glfwGetProcAddress("glDeletePathsNV"));
glDeletePerfMonitorsAMD = PFNGLDELETEPERFMONITORSAMDPROC(glfwGetProcAddress("glDeletePerfMonitorsAMD"));
glDeletePerfQueryINTEL = PFNGLDELETEPERFQUERYINTELPROC(glfwGetProcAddress("glDeletePerfQueryINTEL"));
glDeleteProgram = PFNGLDELETEPROGRAMPROC(glfwGetProcAddress("glDeleteProgram"));
glDeleteProgramPipelines = PFNGLDELETEPROGRAMPIPELINESPROC(glfwGetProcAddress("glDeleteProgramPipelines"));
glDeleteQueries = PFNGLDELETEQUERIESPROC(glfwGetProcAddress("glDeleteQueries"));
glDeleteRenderbuffers = PFNGLDELETERENDERBUFFERSPROC(glfwGetProcAddress("glDeleteRenderbuffers"));
glDeleteSamplers = PFNGLDELETESAMPLERSPROC(glfwGetProcAddress("glDeleteSamplers"));
glDeleteShader = PFNGLDELETESHADERPROC(glfwGetProcAddress("glDeleteShader"));
glDeleteStatesNV = PFNGLDELETESTATESNVPROC(glfwGetProcAddress("glDeleteStatesNV"));
glDeleteSync = PFNGLDELETESYNCPROC(glfwGetProcAddress("glDeleteSync"));
glDeleteTextures = PFNGLDELETETEXTURESPROC(glfwGetProcAddress("glDeleteTextures"));
glDeleteTransformFeedbacks = PFNGLDELETETRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glDeleteTransformFeedbacks"));
glDeleteVertexArrays = PFNGLDELETEVERTEXARRAYSPROC(glfwGetProcAddress("glDeleteVertexArrays"));
glDepthFunc = PFNGLDEPTHFUNCPROC(glfwGetProcAddress("glDepthFunc"));
glDepthMask = PFNGLDEPTHMASKPROC(glfwGetProcAddress("glDepthMask"));
glDepthRange = PFNGLDEPTHRANGEPROC(glfwGetProcAddress("glDepthRange"));
glDepthRangeArrayv = PFNGLDEPTHRANGEARRAYVPROC(glfwGetProcAddress("glDepthRangeArrayv"));
glDepthRangeIndexed = PFNGLDEPTHRANGEINDEXEDPROC(glfwGetProcAddress("glDepthRangeIndexed"));
glDepthRangef = PFNGLDEPTHRANGEFPROC(glfwGetProcAddress("glDepthRangef"));
glDetachShader = PFNGLDETACHSHADERPROC(glfwGetProcAddress("glDetachShader"));
glDisable = PFNGLDISABLEPROC(glfwGetProcAddress("glDisable"));
glDisableClientStateIndexedEXT = PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glDisableClientStateIndexedEXT"));
glDisableClientStateiEXT = PFNGLDISABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glDisableClientStateiEXT"));
glDisableIndexedEXT = PFNGLDISABLEINDEXEDEXTPROC(glfwGetProcAddress("glDisableIndexedEXT"));
glDisableVertexArrayAttrib = PFNGLDISABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glDisableVertexArrayAttrib"));
glDisableVertexArrayAttribEXT = PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glDisableVertexArrayAttribEXT"));
glDisableVertexArrayEXT = PFNGLDISABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glDisableVertexArrayEXT"));
glDisableVertexAttribArray = PFNGLDISABLEVERTEXATTRIBARRAYPROC(glfwGetProcAddress("glDisableVertexAttribArray"));
glDisablei = PFNGLDISABLEIPROC(glfwGetProcAddress("glDisablei"));
glDispatchCompute = PFNGLDISPATCHCOMPUTEPROC(glfwGetProcAddress("glDispatchCompute"));
glDispatchComputeGroupSizeARB = PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC(glfwGetProcAddress("glDispatchComputeGroupSizeARB"));
glDispatchComputeIndirect = PFNGLDISPATCHCOMPUTEINDIRECTPROC(glfwGetProcAddress("glDispatchComputeIndirect"));
glDrawArrays = PFNGLDRAWARRAYSPROC(glfwGetProcAddress("glDrawArrays"));
glDrawArraysIndirect = PFNGLDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glDrawArraysIndirect"));
glDrawArraysInstanced = PFNGLDRAWARRAYSINSTANCEDPROC(glfwGetProcAddress("glDrawArraysInstanced"));
glDrawArraysInstancedARB = PFNGLDRAWARRAYSINSTANCEDARBPROC(glfwGetProcAddress("glDrawArraysInstancedARB"));
glDrawArraysInstancedBaseInstance = PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawArraysInstancedBaseInstance"));
glDrawArraysInstancedEXT = PFNGLDRAWARRAYSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawArraysInstancedEXT"));
glDrawBuffer = PFNGLDRAWBUFFERPROC(glfwGetProcAddress("glDrawBuffer"));
glDrawBuffers = PFNGLDRAWBUFFERSPROC(glfwGetProcAddress("glDrawBuffers"));
glDrawCommandsAddressNV = PFNGLDRAWCOMMANDSADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsAddressNV"));
glDrawCommandsNV = PFNGLDRAWCOMMANDSNVPROC(glfwGetProcAddress("glDrawCommandsNV"));
glDrawCommandsStatesAddressNV = PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC(glfwGetProcAddress("glDrawCommandsStatesAddressNV"));
glDrawCommandsStatesNV = PFNGLDRAWCOMMANDSSTATESNVPROC(glfwGetProcAddress("glDrawCommandsStatesNV"));
```

```
 glDrawElements = PFNGLDRAWELEMENTSPROC(glfwGetProcAddress("glDrawElements"));
 glDrawElementsBaseVertex = PFNGLDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsBaseVertex"));
 glDrawElementsIndirect = PFNGLDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glDrawElementsIndirect"));
 glDrawElementsInstanced = PFNGLDRAWELEMENTSINSTANCEDPROC(glfwGetProcAddress("glDrawElementsInstanced"));
 glDrawElementsInstancedARB = PFNGLDRAWELEMENTSINSTANCEDARBPROC(glfwGetProcAddress("glDrawElementsInstancedARB"));
 glDrawElementsInstancedBaseInstance = PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseInstance"));
 glDrawElementsInstancedBaseVertex = PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertex"));
 glDrawElementsInstancedBaseVertexBaseInstance = PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC(glfwGetProcAddress("glDrawElementsInstancedBaseVertexBaseInstance
 glDrawElementsInstancedEXT = PFNGLDRAWELEMENTSINSTANCEDEXTPROC(glfwGetProcAddress("glDrawElementsInstancedEXT"));
 glDrawRangeElements = PFNGLDRAWRANGEELEMENTSPROC(glfwGetProcAddress("glDrawRangeElements"));
 glDrawRangeElementsBaseVertex = PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glDrawRangeElementsBaseVertex"));
 glDrawTransformFeedback = PFNGLDRAWTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glDrawTransformFeedback"));
 glDrawTransformFeedbackInstanced = PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackInstanced"));
 glDrawTransformFeedbackStream = PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC(glfwGetProcAddress("glDrawTransformFeedbackStream"));
 glDrawTransformFeedbackStreamInstanced = PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC(glfwGetProcAddress("glDrawTransformFeedbackStreamInstanced"));
 glDrawVkImageNV = PFNGLDRAWVKIMAGENVPROC(glfwGetProcAddress("glDrawVkImageNV"));
 glEdgeFlagFormatNV = PFNGLEDGEFLAGFORMATNVPROC(glfwGetProcAddress("glEdgeFlagFormatNV"));
 glEnable = PFNGLENABLEPROC(glfwGetProcAddress("glEnable"));
 glEnableClientStateIndexedEXT = PFNGLENABLECLIENTSTATEINDEXEDEXTPROC(glfwGetProcAddress("glEnableClientStateIndexedEXT"));
 glEnableClientStateiEXT = PFNGLENABLECLIENTSTATEIEXTPROC(glfwGetProcAddress("glEnableClientStateiEXT"));
 glEnableIndexedEXT = PFNGLENABLEINDEXEDEXTPROC(glfwGetProcAddress("glEnableIndexedEXT"));
 glEnableVertexArrayAttrib = PFNGLENABLEVERTEXARRAYATTRIBPROC(glfwGetProcAddress("glEnableVertexArrayAttrib"));
 glEnableVertexArrayAttribEXT = PFNGLENABLEVERTEXARRAYATTRIBEXTPROC(glfwGetProcAddress("glEnableVertexArrayAttribEXT"));
 glEnableVertexArrayEXT = PFNGLENABLEVERTEXARRAYEXTPROC(glfwGetProcAddress("glEnableVertexArrayEXT"));
 glEnableVertexAttribArray = PFNGLENABLEVERTEXATTRIBARRAYPROC(glfwGetProcAddress("glEnableVertexAttribArray"));
 glEnablei = PFNGLENABLEIPROC(glfwGetProcAddress("glEnablei"));
 glEndConditionalRender = PFNGLENDCONDITIONALRENDERPROC(glfwGetProcAddress("glEndConditionalRender"));
 glEndConditionalRenderNV = PFNGLENDCONDITIONALRENDERNVPROC(glfwGetProcAddress("glEndConditionalRenderNV"));
 glEndPerfMonitorAMD = PFNGLENDPERFMONITORAMDPROC(glfwGetProcAddress("glEndPerfMonitorAMD"));
 glEndPerfQueryINTEL = PFNGLENDPERFQUERYINTELPROC(glfwGetProcAddress("glEndPerfQueryINTEL"));
 glEndQuery = PFNGLENDQUERYPROC(glfwGetProcAddress("glEndQuery"));
 glEndQueryIndexed = PFNGLENDQUERYINDEXEDPROC(glfwGetProcAddress("glEndQueryIndexed"));
 glEndTransformFeedback = PFNGLENDTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glEndTransformFeedback"));
 glEvaluateDepthValuesARB = PFNGLEVALUATEDEPTHVALUESARBPROC(glfwGetProcAddress("glEvaluateDepthValuesARB"));
 glFenceSync = PFNGLFENCESYNCPROC(glfwGetProcAddress("glFenceSync"));
 glFinish = PFNGLFINISHPROC(glfwGetProcAddress("glFinish"));
 glFlush = PFNGLFLUSHPROC(glfwGetProcAddress("glFlush"));
 glFlushMappedBufferRange = PFNGLFLUSHMAPPEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedBufferRange"));
 glFlushMappedNamedBufferRange = PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glFlushMappedNamedBufferRange"));
 glFlushMappedNamedBufferRangeEXT = PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glFlushMappedNamedBufferRangeEXT"));
 glFogCoordFormatNV = PFNGLFOGCOORDFORMATNVPROC(glfwGetProcAddress("glFogCoordFormatNV"));
 glFragmentCoverageColorNV = PFNGLFRAGMENTCOVERAGECOLORNVPROC(glfwGetProcAddress("glFragmentCoverageColorNV"));
 glFramebufferDrawBufferEXT = PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC(glfwGetProcAddress("glFramebufferDrawBufferEXT"));
 glFramebufferDrawBuffersEXT = PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC(glfwGetProcAddress("glFramebufferDrawBuffersEXT"));
 glFramebufferParameteri = PFNGLFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glFramebufferParameteri"));
 glFramebufferReadBufferEXT = PFNGLFRAMEBUFFERREADBUFFEREXTPROC(glfwGetProcAddress("glFramebufferReadBufferEXT"));
 glFramebufferRenderbuffer = PFNGLFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glFramebufferRenderbuffer"));
 glFramebufferSampleLocationsfvARB = PFNGLFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvARB"));
 glFramebufferSampleLocationsfvNV = PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glFramebufferSampleLocationsfvNV"));
 glFramebufferTexture = PFNGLFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glFramebufferTexture"));
 glFramebufferTexture1D = PFNGLFRAMEBUFFERTEXTURE1DPROC(glfwGetProcAddress("glFramebufferTexture1D"));
 glFramebufferTexture2D =
```

```
PFNGLFRAMEBUFFERTEXTURE2DPROC(glfwGetProcAddress("glFramebufferTexture2D"));
 glFramebufferTexture3D = PFNGLFRAMEBUFFERTEXTURE3DPROC(glfwGetProcAddress("glFramebufferTexture3D"));
 glFramebufferTextureARB = PFNGLFRAMEBUFFERTEXTUREARBPROC(glfwGetProcAddress("glFramebufferTextureARB"));
 glFramebufferTextureFaceARB = PFNGLFRAMEBUFFERTEXTUREFACEARBPROC(glfwGetProcAddress("glFramebufferTextureFaceARB"));
 glFramebufferTextureLayer = PFNGLFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glFramebufferTextureLayer"));
 glFramebufferTextureLayerARB = PFNGLFRAMEBUFFERTEXTURELAYERARBPROC(glfwGetProcAddress("glFramebufferTextureLayerARB"));
 glFramebufferTextureMultiviewOVR = PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC(glfwGetProcAddress("glFramebufferTextureMultiviewOVR"));
 glFrontFace = PFNGLFRONTFACEPROC(glfwGetProcAddress("glFrontFace"));
 glGenBuffers = PFNGLGENBUFFERSPROC(glfwGetProcAddress("glGenBuffers"));
 glGenFramebuffers = PFNGLGENFRAMEBUFFERSPROC(glfwGetProcAddress("glGenFramebuffers"));
 glGenPathsNV = PFNGLGENPATHSNVPROC(glfwGetProcAddress("glGenPathsNV"));
 glGenPerfMonitorsAMD = PFNGLGENPERFMONITORSAMDPROC(glfwGetProcAddress("glGenPerfMonitorsAMD"));
 glGenProgramPipelines = PFNGLGENPROGRAMPIPELINESPROC(glfwGetProcAddress("glGenProgramPipelines"));
 glGenQueries = PFNGLGENQUERIESPROC(glfwGetProcAddress("glGenQueries"));
 glGenRenderbuffers = PFNGLGENRENDERBUFFERSPROC(glfwGetProcAddress("glGenRenderbuffers"));
 glGenSamplers = PFNGLGENSAMPLERSPROC(glfwGetProcAddress("glGenSamplers"));
 glGenTextures = PFNGLGENTEXTURESPROC(glfwGetProcAddress("glGenTextures"));
 glGenTransformFeedbacks = PFNGLGENTRANSFORMFEEDBACKSPROC(glfwGetProcAddress("glGenTransformFeedbacks"));
 glGenVertexArrays = PFNGLGENVERTEXARRAYSPROC(glfwGetProcAddress("glGenVertexArrays"));
 glGenerateMipmap = PFNGLGENERATEMIPMAPPROC(glfwGetProcAddress("glGenerateMipmap"));
 glGenerateMultiTexMipmapEXT = PFNGLGENERATEMULTITEXMIPMAPEXTPROC(glfwGetProcAddress("glGenerateMultiTexMipmapEXT"));
 glGenerateTextureMipmap = PFNGLGENERATETEXTUREMIPMAPPROC(glfwGetProcAddress("glGenerateTextureMipmap"));
 glGenerateTextureMipmapEXT = PFNGLGENERATETEXTUREMIPMAPEXTPROC(glfwGetProcAddress("glGenerateTextureMipmapEXT"));
 glGetActiveAtomicCounterBufferiv = PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC(glfwGetProcAddress("glGetActiveAtomicCounterBufferiv"));
 glGetActiveAttrib = PFNGLGETACTIVEATTRIBPROC(glfwGetProcAddress("glGetActiveAttrib"));
 glGetActiveSubroutineName = PFNGLGETACTIVESUBROUTINENAMEPROC(glfwGetProcAddress("glGetActiveSubroutineName"));
 glGetActiveSubroutineUniformName = PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveSubroutineUniformName"));
 glGetActiveSubroutineUniformiv = PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC(glfwGetProcAddress("glGetActiveSubroutineUniformiv"));
 glGetActiveUniform = PFNGLGETACTIVEUNIFORMPROC(glfwGetProcAddress("glGetActiveUniform"));
 glGetActiveUniformBlockName = PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC(glfwGetProcAddress("glGetActiveUniformBlockName"));
 glGetActiveUniformBlockiv = PFNGLGETACTIVEUNIFORMBLOCKIVPROC(glfwGetProcAddress("glGetActiveUniformBlockiv"));
 glGetActiveUniformName = PFNGLGETACTIVEUNIFORMNAMEPROC(glfwGetProcAddress("glGetActiveUniformName"));
 glGetActiveUniformsiv = PFNGLGETACTIVEUNIFORMSIVPROC(glfwGetProcAddress("glGetActiveUniformsiv"));
 glGetAttachedShaders = PFNGLGETATTACHEDSHADERSPROC(glfwGetProcAddress("glGetAttachedShaders"));
 glGetAttribLocation = PFNGLGETATTRIBLOCATIONPROC(glfwGetProcAddress("glGetAttribLocation"));
 glGetBooleanIndexedvEXT = PFNGLGETBOOLEANINDEXEDVEXTPROC(glfwGetProcAddress("glGetBooleanIndexedvEXT"));
 glGetBooleani_v = PFNGLGETBOOLEANI_VPROC(glfwGetProcAddress("glGetBooleani_v"));
 glGetBooleanv = PFNGLGETBOOLEANVPROC(glfwGetProcAddress("glGetBooleanv"));
 glGetBufferParameteri64v = PFNGLGETBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetBufferParameteri64v"));
 glGetBufferParameteriv = PFNGLGETBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetBufferParameteriv"));
 glGetBufferParameterui64vNV = PFNGLGETBUFFERPARAMETERUI64VNVPROC(glfwGetProcAddress("glGetBufferParameterui64vNV"));
 glGetBufferPointerv = PFNGLGETBUFFERPOINTERVPROC(glfwGetProcAddress("glGetBufferPointerv"));
 glGetBufferSubData = PFNGLGETBUFFERSUBDATAPROC(glfwGetProcAddress("glGetBufferSubData"));
 glGetCommandHeaderNV = PFNGLGETCOMMANDHEADERNVPROC(glfwGetProcAddress("glGetCommandHeaderNV"));
 glGetCompressedMultiTexImageEXT = PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedMultiTexImageEXT"));
 glGetCompressedTexImage = PFNGLGETCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetCompressedTexImage"));
 glGetCompressedTextureImage = PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureImage"));
 glGetCompressedTextureImageEXT = PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetCompressedTextureImageEXT"));
 glGetCompressedTextureSubImage = PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetCompressedTextureSubImage"));
 glGetCoverageModulationTableNV = PFNGLGETCOVERAGEMODULATIONTABLENVPROC(glfwGetProcAddress("glGetCoverageModulationTableNV"));
 glGetDebugMessageLog = PFNGLGETDEBUGMESSAGELOGPROC(glfwGetProcAddress("glGetDebugMessageLog"));
 glGetDebugMessageLogARB = PFNGLGETDEBUGMESSAGELOGARBPROC(glfwGetProcAddress("glGetDebugMessageLogARB"));
 glGetDoubleIndexedvEXT = PFNGLGETDOUBLEINDEXEDVEXTPROC(glfwGetProcAddress("glGetDoubleIndexedvEXT"));
 glGetDoublei_v = PFNGLGETDOUBLEI_VPROC(glfwGetProcAddress("glGetDoublei_v"));
 glGetDoublei_vEXT = PFNGLGETDOUBLEI_VEXTPROC(glfwGetProcAddress("glGetDoublei_vEXT"));
 glGetDoublev = PFNGLGETDOUBLEVPROC(glfwGetProcAddress("glGetDoublev"));
 glGetError = PFNGLGETERRORPROC(glfwGetProcAddress("glGetError"));
 glGetFirstPerfQueryIdINTEL =
```

```
PFNGLGETFIRSTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetFirstPerfQueryIdINTEL"));
01639  glGetFloatIndexedvEXT = PFNGLGETFLOATINDEXEDVEXTPROC(glfwGetProcAddress("glGetFloatIndexedvEXT"));
01640  glGetFloati_v = PFNGLGETFLOATI_VPROC(glfwGetProcAddress("glGetFloati_v"));
01641  glGetFloati_vEXT = PFNGLGETFLOATI_VEXTPROC(glfwGetProcAddress("glGetFloati_vEXT"));
01642  glGetFloatv = PFNGLGETFLOATVPROC(glfwGetProcAddress("glGetFloatv"));
01643  glGetFragDataIndex = PFNGLGETFRAGDATAINDEXPROC(glfwGetProcAddress("glGetFragDataIndex"));
01644  glGetFragDataLocation = PFNGLGETFRAGDATALOCATIONPROC(glfwGetProcAddress("glGetFragDataLocation"));
01645  glGetFramebufferAttachmentParameteriv =
PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferAttachmentParameteriv"));
01646  glGetFramebufferParameteriv =
PFNGLGETFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetFramebufferParameteriv"));
01647  glGetFramebufferParameterivEXT =
PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetFramebufferParameterivEXT"));
01648  glGetGraphicsResetStatus =
PFNGLGETGRAPHICSRESETSTATUSPROC(glfwGetProcAddress("glGetGraphicsResetStatus"));
01649  glGetGraphicsResetStatusARB =
PFNGLGETGRAPHICSRESETSTATUSARBPROC(glfwGetProcAddress("glGetGraphicsResetStatusARB"));
01650  glGetImageHandleARB = PFNGLGETIMAGEHANDLEARBPROC(glfwGetProcAddress("glGetImageHandleARB"));
01651  glGetImageHandleNV = PFNGLGETIMAGEHANDLENVPROC(glfwGetProcAddress("glGetImageHandleNV"));
01652  glGetInteger64i_v = PFNGLGETINTEGER64I_VPROC(glfwGetProcAddress("glGetInteger64i_v"));
01653  glGetInteger64v = PFNGLGETINTEGER64VPROC(glfwGetProcAddress("glGetInteger64v"));
01654  glGetIntegerIndexedvEXT =
PFNGLGETINTEGERINDEXEDVEXTPROC(glfwGetProcAddress("glGetIntegerIndexedvEXT"));
01655  glGetIntegeri_v = PFNGLGETINTEGERI_VPROC(glfwGetProcAddress("glGetIntegeri_v"));
01656  glGetIntegerui64i_vNV = PFNGLGETINTEGERUI64I_VNVPROC(glfwGetProcAddress("glGetIntegerui64i_vNV"));
01657  glGetIntegerui64vNV = PFNGLGETINTEGERUI64VNVPROC(glfwGetProcAddress("glGetIntegerui64vNV"));
01658  glGetIntegerv = PFNGLGETINTEGERVPROC(glfwGetProcAddress("glGetIntegerv"));
01659  glGetInternalformatSampleivNV =
PFNGLGETINTERNALFORMATSAMPLEIVNVPROC(glfwGetProcAddress("glGetInternalformatSampleivNV"));
01660  glGetInternalformati64v =
PFNGLGETINTERNALFORMATI64VPROC(glfwGetProcAddress("glGetInternalformati64v"));
01661  glGetInternalformativ = PFNGLGETINTERNALFORMATIVPROC(glfwGetProcAddress("glGetInternalformativ"));
01662  glGetMultiTexEnvfvEXT = PFNGLGETMULTITEXENVFVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvfvEXT"));
01663  glGetMultiTexEnvivEXT = PFNGLGETMULTITEXENVIVEXTPROC(glfwGetProcAddress("glGetMultiTexEnvivEXT"));
01664  glGetMultiTexGendvEXT = PFNGLGETMULTITEXGENDVEXTPROC(glfwGetProcAddress("glGetMultiTexGendvEXT"));
01665  glGetMultiTexGenfvEXT = PFNGLGETMULTITEXGENFVEXTPROC(glfwGetProcAddress("glGetMultiTexGenfvEXT"));
01666  glGetMultiTexGenivEXT = PFNGLGETMULTITEXGENIVEXTPROC(glfwGetProcAddress("glGetMultiTexGenivEXT"));
01667  glGetMultiTexImageEXT = PFNGLGETMULTITEXIMAGEEXTPROC(glfwGetProcAddress("glGetMultiTexImageEXT"));
01668  glGetMultiTexLevelParameterfvEXT =
PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterfvEXT"));
01669  glGetMultiTexLevelParameterivEXT =
PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexLevelParameterivEXT"));
01670  glGetMultiTexParameterIivEXT =
PFNGLGETMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIivEXT"));
01671  glGetMultiTexParameterIuivEXT =
PFNGLGETMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterIuivEXT"));
01672  glGetMultiTexParameterfvEXT =
PFNGLGETMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterfvEXT"));
01673  glGetMultiTexParameterivEXT =
PFNGLGETMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glGetMultiTexParameterivEXT"));
01674  glGetMultisamplefv = PFNGLGETMULTISAMPLEFVPROC(glfwGetProcAddress("glGetMultisamplefv"));
01675  glGetNamedBufferParameteri64v =
PFNGLGETNAMEDBUFFERPARAMETERI64VPROC(glfwGetProcAddress("glGetNamedBufferParameteri64v"));
01676  glGetNamedBufferParameteriv =
PFNGLGETNAMEDBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedBufferParameteriv"));
01677  glGetNamedBufferParameterivEXT =
PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedBufferParameterivEXT"));
01678  glGetNamedBufferParameterui64vNV =
PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC(glfwGetProcAddress("glGetNamedBufferParameterui64vNV"));
01679  glGetNamedBufferPointerv =
PFNGLGETNAMEDBUFFERPOINTERVPROC(glfwGetProcAddress("glGetNamedBufferPointerv"));
01680  glGetNamedBufferPointervEXT =
PFNGLGETNAMEDBUFFERPOINTERVEXTPROC(glfwGetProcAddress("glGetNamedBufferPointervEXT"));
01681  glGetNamedBufferSubData =
PFNGLGETNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glGetNamedBufferSubData"));
01682  glGetNamedBufferSubDataEXT =
PFNGLGETNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glGetNamedBufferSubDataEXT"));
01683  glGetNamedFramebufferAttachmentParameteriv =
PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameteriv"));
01684  glGetNamedFramebufferAttachmentParameterivEXT =
PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferAttachmentParameterivEXT
01685  glGetNamedFramebufferParameteriv =
PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedFramebufferParameteriv"));
01686  glGetNamedFramebufferParameterivEXT =
PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedFramebufferParameterivEXT"));
01687  glGetNamedProgramLocalParameterIivEXT =
PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIivEXT"));
01688  glGetNamedProgramLocalParameterIuivEXT =
PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterIuivEXT"));
01689  glGetNamedProgramLocalParameterdvEXT =
PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterdvEXT"));
01690  glGetNamedProgramLocalParameterfvEXT =
PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC(glfwGetProcAddress("glGetNamedProgramLocalParameterfvEXT"));
01691  glGetNamedProgramStringEXT =
PFNGLGETNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glGetNamedProgramStringEXT"));
01692  glGetNamedProgramivEXT =
PFNGLGETNAMEDPROGRAMIVEXTPROC(glfwGetProcAddress("glGetNamedProgramivEXT"));
```

```
glGetNamedRenderbufferParameteriv = PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetNamedRenderbufferParameteriv"));
glGetNamedRenderbufferParameterivEXT = PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC(glfwGetProcAddress("glGetNamedRenderbufferParameterivEXT"));
glGetNamedStringARB = PFNGLGETNAMEDSTRINGARBPROC(glfwGetProcAddress("glGetNamedStringARB"));
glGetNamedStringivARB = PFNGLGETNAMEDSTRINGIVARBPROC(glfwGetProcAddress("glGetNamedStringivARB"));
glGetNextPerfQueryIdINTEL = PFNGLGETNEXTPERFQUERYIDINTELPROC(glfwGetProcAddress("glGetNextPerfQueryIdINTEL"));
glGetObjectLabel = PFNGLGETOBJECTLABELPROC(glfwGetProcAddress("glGetObjectLabel"));
glGetObjectLabelEXT = PFNGLGETOBJECTLABELEXTPROC(glfwGetProcAddress("glGetObjectLabelEXT"));
glGetObjectPtrLabel = PFNGLGETOBJECTPTRLABELPROC(glfwGetProcAddress("glGetObjectPtrLabel"));
glGetPathCommandsNV = PFNGLGETPATHCOMMANDSNVPROC(glfwGetProcAddress("glGetPathCommandsNV"));
glGetPathCoordsNV = PFNGLGETPATHCOORDSNVPROC(glfwGetProcAddress("glGetPathCoordsNV"));
glGetPathDashArrayNV = PFNGLGETPATHDASHARRAYNVPROC(glfwGetProcAddress("glGetPathDashArrayNV"));
glGetPathLengthNV = PFNGLGETPATHLENGTHNVPROC(glfwGetProcAddress("glGetPathLengthNV"));
glGetPathMetricRangeNV = PFNGLGETPATHMETRICRANGENVPROC(glfwGetProcAddress("glGetPathMetricRangeNV"));
glGetPathMetricsNV = PFNGLGETPATHMETRICSNVPROC(glfwGetProcAddress("glGetPathMetricsNV"));
glGetPathParameterfvNV = PFNGLGETPATHPARAMETERFVNVPROC(glfwGetProcAddress("glGetPathParameterfvNV"));
glGetPathParameterivNV = PFNGLGETPATHPARAMETERIVNVPROC(glfwGetProcAddress("glGetPathParameterivNV"));
glGetPathSpacingNV = PFNGLGETPATHSPACINGNVPROC(glfwGetProcAddress("glGetPathSpacingNV"));
glGetPerfCounterInfoINTEL = PFNGLGETPERFCOUNTERINFOINTELPROC(glfwGetProcAddress("glGetPerfCounterInfoINTEL"));
glGetPerfMonitorCounterDataAMD = PFNGLGETPERFMONITORCOUNTERDATAAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterDataAMD"));
glGetPerfMonitorCounterInfoAMD = PFNGLGETPERFMONITORCOUNTERINFOAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterInfoAMD"));
glGetPerfMonitorCounterStringAMD = PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorCounterStringAMD"));
glGetPerfMonitorCountersAMD = PFNGLGETPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glGetPerfMonitorCountersAMD"));
glGetPerfMonitorGroupStringAMD = PFNGLGETPERFMONITORGROUPSTRINGAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupStringAMD"));
glGetPerfMonitorGroupsAMD = PFNGLGETPERFMONITORGROUPSAMDPROC(glfwGetProcAddress("glGetPerfMonitorGroupsAMD"));
glGetPerfQueryDataINTEL = PFNGLGETPERFQUERYDATAINTELPROC(glfwGetProcAddress("glGetPerfQueryDataINTEL"));
glGetPerfQueryIdByNameINTEL = PFNGLGETPERFQUERYIDBYNAMEINTELPROC(glfwGetProcAddress("glGetPerfQueryIdByNameINTEL"));
glGetPerfQueryInfoINTEL = PFNGLGETPERFQUERYINFOINTELPROC(glfwGetProcAddress("glGetPerfQueryInfoINTEL"));
glGetPointerIndexedvEXT = PFNGLGETPOINTERINDEXEDVEXTPROC(glfwGetProcAddress("glGetPointerIndexedvEXT"));
glGetPointeri_vEXT = PFNGLGETPOINTERI_VEXTPROC(glfwGetProcAddress("glGetPointeri_vEXT"));
glGetPointerv = PFNGLGETPOINTERVPROC(glfwGetProcAddress("glGetPointerv"));
glGetProgramBinary = PFNGLGETPROGRAMBINARYPROC(glfwGetProcAddress("glGetProgramBinary"));
glGetProgramInfoLog = PFNGLGETPROGRAMINFOLOGPROC(glfwGetProcAddress("glGetProgramInfoLog"));
glGetProgramInterfaceiv = PFNGLGETPROGRAMINTERFACEIVPROC(glfwGetProcAddress("glGetProgramInterfaceiv"));
glGetProgramPipelineInfoLog = PFNGLGETPROGRAMPIPELINEINFOLOGPROC(glfwGetProcAddress("glGetProgramPipelineInfoLog"));
glGetProgramPipelineiv = PFNGLGETPROGRAMPIPELINEIVPROC(glfwGetProcAddress("glGetProgramPipelineiv"));
glGetProgramResourceIndex = PFNGLGETPROGRAMRESOURCEINDEXPROC(glfwGetProcAddress("glGetProgramResourceIndex"));
glGetProgramResourceLocation = PFNGLGETPROGRAMRESOURCELOCATIONPROC(glfwGetProcAddress("glGetProgramResourceLocation"));
glGetProgramResourceLocationIndex = PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC(glfwGetProcAddress("glGetProgramResourceLocationIndex"));
glGetProgramResourceName = PFNGLGETPROGRAMRESOURCENAMEPROC(glfwGetProcAddress("glGetProgramResourceName"));
glGetProgramResourcefvNV = PFNGLGETPROGRAMRESOURCEFVNVPROC(glfwGetProcAddress("glGetProgramResourcefvNV"));
glGetProgramResourceiv = PFNGLGETPROGRAMRESOURCEIVPROC(glfwGetProcAddress("glGetProgramResourceiv"));
glGetProgramStageiv = PFNGLGETPROGRAMSTAGEIVPROC(glfwGetProcAddress("glGetProgramStageiv"));
glGetProgramiv = PFNGLGETPROGRAMIVPROC(glfwGetProcAddress("glGetProgramiv"));
glGetQueryBufferObjecti64v = PFNGLGETQUERYBUFFEROBJECTI64VPROC(glfwGetProcAddress("glGetQueryBufferObjecti64v"));
glGetQueryBufferObjectiv = PFNGLGETQUERYBUFFEROBJECTIVPROC(glfwGetProcAddress("glGetQueryBufferObjectiv"));
glGetQueryBufferObjectui64v = PFNGLGETQUERYBUFFEROBJECTUI64VPROC(glfwGetProcAddress("glGetQueryBufferObjectui64v"));
glGetQueryBufferObjectuiv = PFNGLGETQUERYBUFFEROBJECTUIVPROC(glfwGetProcAddress("glGetQueryBufferObjectuiv"));
glGetQueryIndexediv = PFNGLGETQUERYINDEXEDIVPROC(glfwGetProcAddress("glGetQueryIndexediv"));
glGetQueryObjecti64v = PFNGLGETQUERYOBJECTI64VPROC(glfwGetProcAddress("glGetQueryObjecti64v"));
glGetQueryObjectiv = PFNGLGETQUERYOBJECTIVPROC(glfwGetProcAddress("glGetQueryObjectiv"));
glGetQueryObjectui64v = PFNGLGETQUERYOBJECTUI64VPROC(glfwGetProcAddress("glGetQueryObjectui64v"));
glGetQueryObjectuiv = PFNGLGETQUERYOBJECTUIVPROC(glfwGetProcAddress("glGetQueryObjectuiv"));
glGetQueryiv = PFNGLGETQUERYIVPROC(glfwGetProcAddress("glGetQueryiv"));
glGetRenderbufferParameteriv = PFNGLGETRENDERBUFFERPARAMETERIVPROC(glfwGetProcAddress("glGetRenderbufferParameteriv"));
glGetSamplerParameterIiv = PFNGLGETSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glGetSamplerParameterIiv"));
```

```
glGetSamplerParameterIuiv = PFNGLGETSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glGetSamplerParameterIuiv"));
glGetSamplerParameterfv = PFNGLGETSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glGetSamplerParameterfv"));
glGetSamplerParameteriv = PFNGLGETSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glGetSamplerParameteriv"));
glGetShaderInfoLog = PFNGLGETSHADERINFOLOGPROC(glfwGetProcAddress("glGetShaderInfoLog"));
glGetShaderPrecisionFormat = PFNGLGETSHADERPRECISIONFORMATPROC(glfwGetProcAddress("glGetShaderPrecisionFormat"));
glGetShaderSource = PFNGLGETSHADERSOURCEPROC(glfwGetProcAddress("glGetShaderSource"));
glGetShaderiv = PFNGLGETSHADERIVPROC(glfwGetProcAddress("glGetShaderiv"));
glGetStageIndexNV = PFNGLGETSTAGEINDEXNVPROC(glfwGetProcAddress("glGetStageIndexNV"));
glGetString = PFNGLGETSTRINGPROC(glfwGetProcAddress("glGetString"));
glGetStringi = PFNGLGETSTRINGIPROC(glfwGetProcAddress("glGetStringi"));
glGetSubroutineIndex = PFNGLGETSUBROUTINEINDEXPROC(glfwGetProcAddress("glGetSubroutineIndex"));
glGetSubroutineUniformLocation = PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetSubroutineUniformLocation"));
glGetSynciv = PFNGLGETSYNCIVPROC(glfwGetProcAddress("glGetSynciv"));
glGetTexImage = PFNGLGETTEXIMAGEPROC(glfwGetProcAddress("glGetTexImage"));
glGetTexLevelParameterfv = PFNGLGETTEXLEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTexLevelParameterfv"));
glGetTexLevelParameteriv = PFNGLGETTEXLEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTexLevelParameteriv"));
glGetTexParameterIiv = PFNGLGETTEXPARAMETERIIVPROC(glfwGetProcAddress("glGetTexParameterIiv"));
glGetTexParameterIuiv = PFNGLGETTEXPARAMETERIUIVPROC(glfwGetProcAddress("glGetTexParameterIuiv"));
glGetTexParameterfv = PFNGLGETTEXPARAMETERFVPROC(glfwGetProcAddress("glGetTexParameterfv"));
glGetTexParameteriv = PFNGLGETTEXPARAMETERIVPROC(glfwGetProcAddress("glGetTexParameteriv"));
glGetTextureHandleARB = PFNGLGETTEXTUREHANDLEARBPROC(glfwGetProcAddress("glGetTextureHandleARB"));
glGetTextureHandleNV = PFNGLGETTEXTUREHANDLENVPROC(glfwGetProcAddress("glGetTextureHandleNV"));
glGetTextureImage = PFNGLGETTEXTUREIMAGEPROC(glfwGetProcAddress("glGetTextureImage"));
glGetTextureImageEXT = PFNGLGETTEXTUREIMAGEEXTPROC(glfwGetProcAddress("glGetTextureImageEXT"));
glGetTextureLevelParameterfv = PFNGLGETTEXTURELEVELPARAMETERFVPROC(glfwGetProcAddress("glGetTextureLevelParameterfv"));
glGetTextureLevelParameterfvEXT = PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterfvEXT"));
glGetTextureLevelParameteriv = PFNGLGETTEXTURELEVELPARAMETERIVPROC(glfwGetProcAddress("glGetTextureLevelParameteriv"));
glGetTextureLevelParameterivEXT = PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureLevelParameterivEXT"));
glGetTextureParameterIiv = PFNGLGETTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glGetTextureParameterIiv"));
glGetTextureParameterIivEXT = PFNGLGETTEXTUREPARAMETERIIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIivEXT"));
glGetTextureParameterIuiv = PFNGLGETTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glGetTextureParameterIuiv"));
glGetTextureParameterIuivEXT = PFNGLGETTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glGetTextureParameterIuivEXT"));
glGetTextureParameterfv = PFNGLGETTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glGetTextureParameterfv"));
glGetTextureParameterfvEXT = PFNGLGETTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glGetTextureParameterfvEXT"));
glGetTextureParameteriv = PFNGLGETTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glGetTextureParameteriv"));
glGetTextureParameterivEXT = PFNGLGETTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glGetTextureParameterivEXT"));
glGetTextureSamplerHandleARB = PFNGLGETTEXTURESAMPLERHANDLEARBPROC(glfwGetProcAddress("glGetTextureSamplerHandleARB"));
glGetTextureSamplerHandleNV = PFNGLGETTEXTURESAMPLERHANDLENVPROC(glfwGetProcAddress("glGetTextureSamplerHandleNV"));
glGetTextureSubImage = PFNGLGETTEXTURESUBIMAGEPROC(glfwGetProcAddress("glGetTextureSubImage"));
glGetTransformFeedbackVarying = PFNGLGETTRANSFORMFEEDBACKVARYINGPROC(glfwGetProcAddress("glGetTransformFeedbackVarying"));
glGetTransformFeedbacki64_v = PFNGLGETTRANSFORMFEEDBACKI64_VPROC(glfwGetProcAddress("glGetTransformFeedbacki64_v"));
glGetTransformFeedbacki_v = PFNGLGETTRANSFORMFEEDBACKI_VPROC(glfwGetProcAddress("glGetTransformFeedbacki_v"));
glGetTransformFeedbackiv = PFNGLGETTRANSFORMFEEDBACKIVPROC(glfwGetProcAddress("glGetTransformFeedbackiv"));
glGetUniformBlockIndex = PFNGLGETUNIFORMBLOCKINDEXPROC(glfwGetProcAddress("glGetUniformBlockIndex"));
glGetUniformIndices = PFNGLGETUNIFORMINDICESPROC(glfwGetProcAddress("glGetUniformIndices"));
glGetUniformLocation = PFNGLGETUNIFORMLOCATIONPROC(glfwGetProcAddress("glGetUniformLocation"));
glGetUniformSubroutineuiv = PFNGLGETUNIFORMSUBROUTINEUIVPROC(glfwGetProcAddress("glGetUniformSubroutineuiv"));
glGetUniformdv = PFNGLGETUNIFORMDVPROC(glfwGetProcAddress("glGetUniformdv"));
glGetUniformfv = PFNGLGETUNIFORMFVPROC(glfwGetProcAddress("glGetUniformfv"));
glGetUniformi64vARB = PFNGLGETUNIFORMI64VARBPROC(glfwGetProcAddress("glGetUniformi64vARB"));
glGetUniformi64vNV = PFNGLGETUNIFORMI64VNVPROC(glfwGetProcAddress("glGetUniformi64vNV"));
glGetUniformiv = PFNGLGETUNIFORMIVPROC(glfwGetProcAddress("glGetUniformiv"));
glGetUniformui64vARB = PFNGLGETUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetUniformui64vARB"));
glGetUniformui64vNV = PFNGLGETUNIFORMUI64VNVPROC(glfwGetProcAddress("glGetUniformui64vNV"));
glGetUniformuiv = PFNGLGETUNIFORMUIVPROC(glfwGetProcAddress("glGetUniformuiv"));
glGetVertexArrayIndexed64iv = PFNGLGETVERTEXARRAYINDEXED64IVPROC(glfwGetProcAddress("glGetVertexArrayIndexed64iv"));
glGetVertexArrayIndexediv = PFNGLGETVERTEXARRAYINDEXEDIVPROC(glfwGetProcAddress("glGetVertexArrayIndexediv"));
glGetVertexArrayIntegeri_vEXT =
```

```
PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegeri_vEXT"));
  glGetVertexArrayIntegervEXT =
PFNGLGETVERTEXARRAYINTEGERVEXTPROC(glfwGetProcAddress("glGetVertexArrayIntegervEXT"));
  glGetVertexArrayPointeri_vEXT =
PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC(glfwGetProcAddress("glGetVertexArrayPointeri_vEXT"));
  glGetVertexArrayPointervEXT =
PFNGLGETVERTEXARRAYPOINTERVEXTPROC(glfwGetProcAddress("glGetVertexArrayPointervEXT"));
  glGetVertexArrayiv = PFNGLGETVERTEXARRAYIVPROC(glfwGetProcAddress("glGetVertexArrayiv"));
  glGetVertexAttribIiv = PFNGLGETVERTEXATTRIBIIVPROC(glfwGetProcAddress("glGetVertexAttribIiv"));
  glGetVertexAttribIuiv = PFNGLGETVERTEXATTRIBIUIVPROC(glfwGetProcAddress("glGetVertexAttribIuiv"));
  glGetVertexAttribLdv = PFNGLGETVERTEXATTRIBLDVPROC(glfwGetProcAddress("glGetVertexAttribLdv"));
  glGetVertexAttribLi64vNV =
PFNGLGETVERTEXATTRIBLI64VNVPROC(glfwGetProcAddress("glGetVertexAttribLi64vNV"));
  glGetVertexAttribLui64vARB =
PFNGLGETVERTEXATTRIBLUI64VARBPROC(glfwGetProcAddress("glGetVertexAttribLui64vARB"));
  glGetVertexAttribLui64vNV =
PFNGLGETVERTEXATTRIBLUI64VNVPROC(glfwGetProcAddress("glGetVertexAttribLui64vNV"));
  glGetVertexAttribPointerv =
PFNGLGETVERTEXATTRIBPOINTERVPROC(glfwGetProcAddress("glGetVertexAttribPointerv"));
  glGetVertexAttribdv = PFNGLGETVERTEXATTRIBDVPROC(glfwGetProcAddress("glGetVertexAttribdv"));
  glGetVertexAttribfv = PFNGLGETVERTEXATTRIBFVPROC(glfwGetProcAddress("glGetVertexAttribfv"));
  glGetVertexAttribiv = PFNGLGETVERTEXATTRIBIVPROC(glfwGetProcAddress("glGetVertexAttribiv"));
  glGetVkProcAddrNV = PFNGLGETVKPROCADDRNVPROC(glfwGetProcAddress("glGetVkProcAddrNV"));
  glGetnCompressedTexImage =
PFNGLGETNCOMPRESSEDTEXIMAGEPROC(glfwGetProcAddress("glGetnCompressedTexImage"));
  glGetnCompressedTexImageARB =
PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC(glfwGetProcAddress("glGetnCompressedTexImageARB"));
  glGetnTexImage = PFNGLGETNTEXIMAGEPROC(glfwGetProcAddress("glGetnTexImage"));
  glGetnTexImageARB = PFNGLGETNTEXIMAGEARBPROC(glfwGetProcAddress("glGetnTexImageARB"));
  glGetnUniformdv = PFNGLGETNUNIFORMDVPROC(glfwGetProcAddress("glGetnUniformdv"));
  glGetnUniformdvARB = PFNGLGETNUNIFORMDVARBPROC(glfwGetProcAddress("glGetnUniformdvARB"));
  glGetnUniformfv = PFNGLGETNUNIFORMFVPROC(glfwGetProcAddress("glGetnUniformfv"));
  glGetnUniformfvARB = PFNGLGETNUNIFORMFVARBPROC(glfwGetProcAddress("glGetnUniformfvARB"));
  glGetnUniformi64vARB = PFNGLGETNUNIFORMI64VARBPROC(glfwGetProcAddress("glGetnUniformi64vARB"));
  glGetnUniformiv = PFNGLGETNUNIFORMIVPROC(glfwGetProcAddress("glGetnUniformiv"));
  glGetnUniformivARB = PFNGLGETNUNIFORMIVARBPROC(glfwGetProcAddress("glGetnUniformivARB"));
  glGetnUniformui64vARB = PFNGLGETNUNIFORMUI64VARBPROC(glfwGetProcAddress("glGetnUniformui64vARB"));
  glGetnUniformuiv = PFNGLGETNUNIFORMUIVPROC(glfwGetProcAddress("glGetnUniformuiv"));
  glGetnUniformuivARB = PFNGLGETNUNIFORMUIVARBPROC(glfwGetProcAddress("glGetnUniformuivARB"));
  glHint = PFNGLHINTPROC(glfwGetProcAddress("glHint"));
  glIndexFormatNV = PFNGLINDEXFORMATNVPROC(glfwGetProcAddress("glIndexFormatNV"));
  glInsertEventMarkerEXT =
PFNGLINSERTEVENTMARKEREXTPROC(glfwGetProcAddress("glInsertEventMarkerEXT"));
  glInterpolatePathsNV = PFNGLINTERPOLATEPATHSNVPROC(glfwGetProcAddress("glInterpolatePathsNV"));
  glInvalidateBufferData =
PFNGLINVALIDATEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateBufferData"));
  glInvalidateBufferSubData =
PFNGLINVALIDATEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateBufferSubData"));
  glInvalidateFramebuffer =
PFNGLINVALIDATEFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateFramebuffer"));
  glInvalidateNamedFramebufferData =
PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferData"));
  glInvalidateNamedFramebufferSubData =
PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC(glfwGetProcAddress("glInvalidateNamedFramebufferSubData"));
  glInvalidateSubFramebuffer =
PFNGLINVALIDATESUBFRAMEBUFFERPROC(glfwGetProcAddress("glInvalidateSubFramebuffer"));
  glInvalidateTexImage = PFNGLINVALIDATETEXIMAGEPROC(glfwGetProcAddress("glInvalidateTexImage"));
  glInvalidateTexSubImage =
PFNGLINVALIDATETEXSUBIMAGEPROC(glfwGetProcAddress("glInvalidateTexSubImage"));
  glIsBuffer = PFNGLISBUFFERPROC(glfwGetProcAddress("glIsBuffer"));
  glIsBufferResidentNV = PFNGLISBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsBufferResidentNV"));
  glIsCommandListNV = PFNGLISCOMMANDLISTNVPROC(glfwGetProcAddress("glIsCommandListNV"));
  glIsEnabled = PFNGLISENABLEDPROC(glfwGetProcAddress("glIsEnabled"));
  glIsEnabledIndexedEXT = PFNGLISENABLEDINDEXEDEXTPROC(glfwGetProcAddress("glIsEnabledIndexedEXT"));
  glIsEnabledi = PFNGLISENABLEDIPROC(glfwGetProcAddress("glIsEnabledi"));
  glIsFramebuffer = PFNGLISFRAMEBUFFERPROC(glfwGetProcAddress("glIsFramebuffer"));
  glIsImageHandleResidentARB =
PFNGLISIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsImageHandleResidentARB"));
  glIsImageHandleResidentNV =
PFNGLISIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsImageHandleResidentNV"));
  glIsNamedBufferResidentNV =
PFNGLISNAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glIsNamedBufferResidentNV"));
  glIsNamedStringARB = PFNGLISNAMEDSTRINGARBPROC(glfwGetProcAddress("glIsNamedStringARB"));
  glIsPathNV = PFNGLISPATHNVPROC(glfwGetProcAddress("glIsPathNV"));
  glIsPointInFillPathNV = PFNGLISPOINTINFILLPATHNVPROC(glfwGetProcAddress("glIsPointInFillPathNV"));
  glIsPointInStrokePathNV =
PFNGLISPOINTINSTROKEPATHNVPROC(glfwGetProcAddress("glIsPointInStrokePathNV"));
  glIsProgram = PFNGLISPROGRAMPROC(glfwGetProcAddress("glIsProgram"));
  glIsProgramPipeline = PFNGLISPROGRAMPIPELINEPROC(glfwGetProcAddress("glIsProgramPipeline"));
  glIsQuery = PFNGLISQUERYPROC(glfwGetProcAddress("glIsQuery"));
  glIsRenderbuffer = PFNGLISRENDERBUFFERPROC(glfwGetProcAddress("glIsRenderbuffer"));
  glIsSampler = PFNGLISSAMPLERPROC(glfwGetProcAddress("glIsSampler"));
  glIsShader = PFNGLISSHADERPROC(glfwGetProcAddress("glIsShader"));
  glIsStateNV = PFNGLISSTATENVPROC(glfwGetProcAddress("glIsStateNV"));
  glIsSync = PFNGLISSYNCPROC(glfwGetProcAddress("glIsSync"));
  glIsTexture = PFNGLISTEXTUREPROC(glfwGetProcAddress("glIsTexture"));
  glIsTextureHandleResidentARB =
```

```
PFNGLISTEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glIsTextureHandleResidentARB"));
 glIsTextureHandleResidentNV = PFNGLISTEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glIsTextureHandleResidentNV"));
 glIsTransformFeedback = PFNGLISTRANSFORMFEEDBACKPROC(glfwGetProcAddress("glIsTransformFeedback"));
 glIsVertexArray = PFNGLISVERTEXARRAYPROC(glfwGetProcAddress("glIsVertexArray"));
 glLabelObjectEXT = PFNGLLABELOBJECTEXTPROC(glfwGetProcAddress("glLabelObjectEXT"));
 glLineWidth = PFNGLLINEWIDTHPROC(glfwGetProcAddress("glLineWidth"));
 glLinkProgram = PFNGLLINKPROGRAMPROC(glfwGetProcAddress("glLinkProgram"));
 glListDrawCommandsStatesClientNV = PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC(glfwGetProcAddress("glListDrawCommandsStatesClientNV"));
 glLogicOp = PFNGLLOGICOPPROC(glfwGetProcAddress("glLogicOp"));
 glMakeBufferNonResidentNV = PFNGLMAKEBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferNonResidentNV"));
 glMakeBufferResidentNV = PFNGLMAKEBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeBufferResidentNV"));
 glMakeImageHandleNonResidentARB = PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleNonResidentARB"));
 glMakeImageHandleNonResidentNV = PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleNonResidentNV"));
 glMakeImageHandleResidentARB = PFNGLMAKEIMAGEHANDLERESIDENTARBPROC(glfwGetProcAddress("glMakeImageHandleResidentARB"));
 glMakeImageHandleResidentNV = PFNGLMAKEIMAGEHANDLERESIDENTNVPROC(glfwGetProcAddress("glMakeImageHandleResidentNV"));
 glMakeNamedBufferNonResidentNV = PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferNonResidentNV"));
 glMakeNamedBufferResidentNV = PFNGLMAKENAMEDBUFFERRESIDENTNVPROC(glfwGetProcAddress("glMakeNamedBufferResidentNV"));
 glMakeTextureHandleNonResidentARB = PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentARB"));
 glMakeTextureHandleNonResidentNV = PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleNonResidentNV"));
 glMakeTextureHandleResidentARB = PFNGLMAKETEXTUREHANDLERESIDENTARBPROC(glfwGetProcAddress("glMakeTextureHandleResidentARB"));
 glMakeTextureHandleResidentNV = PFNGLMAKETEXTUREHANDLERESIDENTNVPROC(glfwGetProcAddress("glMakeTextureHandleResidentNV"));
 glMapBuffer = PFNGLMAPBUFFERPROC(glfwGetProcAddress("glMapBuffer"));
 glMapBufferRange = PFNGLMAPBUFFERRANGEPROC(glfwGetProcAddress("glMapBufferRange"));
 glMapNamedBuffer = PFNGLMAPNAMEDBUFFERPROC(glfwGetProcAddress("glMapNamedBuffer"));
 glMapNamedBufferEXT = PFNGLMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glMapNamedBufferEXT"));
 glMapNamedBufferRange = PFNGLMAPNAMEDBUFFERRANGEPROC(glfwGetProcAddress("glMapNamedBufferRange"));
 glMapNamedBufferRangeEXT = PFNGLMAPNAMEDBUFFERRANGEEXTPROC(glfwGetProcAddress("glMapNamedBufferRangeEXT"));
 glMatrixFrustumEXT = PFNGLMATRIXFRUSTUMEXTPROC(glfwGetProcAddress("glMatrixFrustumEXT"));
 glMatrixLoad3x2fNV = PFNGLMATRIXLOAD3X2FNVPROC(glfwGetProcAddress("glMatrixLoad3x2fNV"));
 glMatrixLoad3x3fNV = PFNGLMATRIXLOAD3X3FNVPROC(glfwGetProcAddress("glMatrixLoad3x3fNV"));
 glMatrixLoadIdentityEXT = PFNGLMATRIXLOADIDENTITYEXTPROC(glfwGetProcAddress("glMatrixLoadIdentityEXT"));
 glMatrixLoadTranspose3x3fNV = PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixLoadTranspose3x3fNV"));
 glMatrixLoadTransposedEXT = PFNGLMATRIXLOADTRANSPOSEDEXTPROC(glfwGetProcAddress("glMatrixLoadTransposedEXT"));
 glMatrixLoadTransposefEXT = PFNGLMATRIXLOADTRANSPOSEFEXTPROC(glfwGetProcAddress("glMatrixLoadTransposefEXT"));
 glMatrixLoaddEXT = PFNGLMATRIXLOADDEXTPROC(glfwGetProcAddress("glMatrixLoaddEXT"));
 glMatrixLoadfEXT = PFNGLMATRIXLOADFEXTPROC(glfwGetProcAddress("glMatrixLoadfEXT"));
 glMatrixMult3x2fNV = PFNGLMATRIXMULT3X2FNVPROC(glfwGetProcAddress("glMatrixMult3x2fNV"));
 glMatrixMult3x3fNV = PFNGLMATRIXMULT3X3FNVPROC(glfwGetProcAddress("glMatrixMult3x3fNV"));
 glMatrixMultTranspose3x3fNV = PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC(glfwGetProcAddress("glMatrixMultTranspose3x3fNV"));
 glMatrixMultTransposedEXT = PFNGLMATRIXMULTTRANSPOSEDEXTPROC(glfwGetProcAddress("glMatrixMultTransposedEXT"));
 glMatrixMultTransposefEXT = PFNGLMATRIXMULTTRANSPOSEFEXTPROC(glfwGetProcAddress("glMatrixMultTransposefEXT"));
 glMatrixMultdEXT = PFNGLMATRIXMULTDEXTPROC(glfwGetProcAddress("glMatrixMultdEXT"));
 glMatrixMultfEXT = PFNGLMATRIXMULTFEXTPROC(glfwGetProcAddress("glMatrixMultfEXT"));
 glMatrixOrthoEXT = PFNGLMATRIXORTHOEXTPROC(glfwGetProcAddress("glMatrixOrthoEXT"));
 glMatrixPopEXT = PFNGLMATRIXPOPEXTPROC(glfwGetProcAddress("glMatrixPopEXT"));
 glMatrixPushEXT = PFNGLMATRIXPUSHEXTPROC(glfwGetProcAddress("glMatrixPushEXT"));
 glMatrixRotatedEXT = PFNGLMATRIXROTATEDEXTPROC(glfwGetProcAddress("glMatrixRotatedEXT"));
 glMatrixRotatefEXT = PFNGLMATRIXROTATEFEXTPROC(glfwGetProcAddress("glMatrixRotatefEXT"));
 glMatrixScaledEXT = PFNGLMATRIXSCALEDEXTPROC(glfwGetProcAddress("glMatrixScaledEXT"));
 glMatrixScalefEXT = PFNGLMATRIXSCALEFEXTPROC(glfwGetProcAddress("glMatrixScalefEXT"));
 glMatrixTranslatedEXT = PFNGLMATRIXTRANSLATEDEXTPROC(glfwGetProcAddress("glMatrixTranslatedEXT"));
 glMatrixTranslatefEXT = PFNGLMATRIXTRANSLATEFEXTPROC(glfwGetProcAddress("glMatrixTranslatefEXT"));
 glMaxShaderCompilerThreadsARB = PFNGLMAXSHADERCOMPILERTHREADSARBPROC(glfwGetProcAddress("glMaxShaderCompilerThreadsARB"));
 glMemoryBarrier = PFNGLMEMORYBARRIERPROC(glfwGetProcAddress("glMemoryBarrier"));
 glMemoryBarrierByRegion = PFNGLMEMORYBARRIERBYREGIONPROC(glfwGetProcAddress("glMemoryBarrierByRegion"));
 glMinSampleShading = PFNGLMINSAMPLESHADINGPROC(glfwGetProcAddress("glMinSampleShading"));
 glMinSampleShadingARB = PFNGLMINSAMPLESHADINGARBPROC(glfwGetProcAddress("glMinSampleShadingARB"));
 glMultiDrawArrays = PFNGLMULTIDRAWARRAYSPROC(glfwGetProcAddress("glMultiDrawArrays"));
 glMultiDrawArraysIndirect = PFNGLMULTIDRAWARRAYSINDIRECTPROC(glfwGetProcAddress("glMultiDrawArraysIndirect"));
 glMultiDrawArraysIndirectBindlessCountNV = PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessCountNV"));
 glMultiDrawArraysIndirectBindlessNV =
```

```
PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawArraysIndirectBindlessNV"));
 glMultiDrawArraysIndirectCountARB = PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawArraysIndirectCountARB"));
 glMultiDrawElements = PFNGLMULTIDRAWELEMENTSPROC(glfwGetProcAddress("glMultiDrawElements"));
 glMultiDrawElementsBaseVertex = PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC(glfwGetProcAddress("glMultiDrawElementsBaseVertex"));
 glMultiDrawElementsIndirect = PFNGLMULTIDRAWELEMENTSINDIRECTPROC(glfwGetProcAddress("glMultiDrawElementsIndirect"));
 glMultiDrawElementsIndirectBindlessCountNV = PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessCountNV"));
 glMultiDrawElementsIndirectBindlessNV = PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC(glfwGetProcAddress("glMultiDrawElementsIndirectBindlessNV"));
 glMultiDrawElementsIndirectCountARB = PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC(glfwGetProcAddress("glMultiDrawElementsIndirectCountARB"));
 glMultiTexBufferEXT = PFNGLMULTITEXBUFFEREXTPROC(glfwGetProcAddress("glMultiTexBufferEXT"));
 glMultiTexCoordPointerEXT = PFNGLMULTITEXCOORDPOINTEREXTPROC(glfwGetProcAddress("glMultiTexCoordPointerEXT"));
 glMultiTexEnvfEXT = PFNGLMULTITEXENVFEXTPROC(glfwGetProcAddress("glMultiTexEnvfEXT"));
 glMultiTexEnvfvEXT = PFNGLMULTITEXENVFVEXTPROC(glfwGetProcAddress("glMultiTexEnvfvEXT"));
 glMultiTexEnviEXT = PFNGLMULTITEXENVIEXTPROC(glfwGetProcAddress("glMultiTexEnviEXT"));
 glMultiTexEnvivEXT = PFNGLMULTITEXENVIVEXTPROC(glfwGetProcAddress("glMultiTexEnvivEXT"));
 glMultiTexGendEXT = PFNGLMULTITEXGENDEXTPROC(glfwGetProcAddress("glMultiTexGendEXT"));
 glMultiTexGendvEXT = PFNGLMULTITEXGENDVEXTPROC(glfwGetProcAddress("glMultiTexGendvEXT"));
 glMultiTexGenfEXT = PFNGLMULTITEXGENFEXTPROC(glfwGetProcAddress("glMultiTexGenfEXT"));
 glMultiTexGenfvEXT = PFNGLMULTITEXGENFVEXTPROC(glfwGetProcAddress("glMultiTexGenfvEXT"));
 glMultiTexGeniEXT = PFNGLMULTITEXGENIEXTPROC(glfwGetProcAddress("glMultiTexGeniEXT"));
 glMultiTexGenivEXT = PFNGLMULTITEXGENIVEXTPROC(glfwGetProcAddress("glMultiTexGenivEXT"));
 glMultiTexImage1DEXT = PFNGLMULTITEXIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexImage1DEXT"));
 glMultiTexImage2DEXT = PFNGLMULTITEXIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexImage2DEXT"));
 glMultiTexImage3DEXT = PFNGLMULTITEXIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexImage3DEXT"));
 glMultiTexParameterIivEXT = PFNGLMULTITEXPARAMETERIIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIivEXT"));
 glMultiTexParameterIuivEXT = PFNGLMULTITEXPARAMETERIUIVEXTPROC(glfwGetProcAddress("glMultiTexParameterIuivEXT"));
 glMultiTexParameterfEXT = PFNGLMULTITEXPARAMETERFEXTPROC(glfwGetProcAddress("glMultiTexParameterfEXT"));
 glMultiTexParameterfvEXT = PFNGLMULTITEXPARAMETERFVEXTPROC(glfwGetProcAddress("glMultiTexParameterfvEXT"));
 glMultiTexParameteriEXT = PFNGLMULTITEXPARAMETERIEXTPROC(glfwGetProcAddress("glMultiTexParameteriEXT"));
 glMultiTexParameterivEXT = PFNGLMULTITEXPARAMETERIVEXTPROC(glfwGetProcAddress("glMultiTexParameterivEXT"));
 glMultiTexRenderbufferEXT = PFNGLMULTITEXRENDERBUFFEREXTPROC(glfwGetProcAddress("glMultiTexRenderbufferEXT"));
 glMultiTexSubImage1DEXT = PFNGLMULTITEXSUBIMAGE1DEXTPROC(glfwGetProcAddress("glMultiTexSubImage1DEXT"));
 glMultiTexSubImage2DEXT = PFNGLMULTITEXSUBIMAGE2DEXTPROC(glfwGetProcAddress("glMultiTexSubImage2DEXT"));
 glMultiTexSubImage3DEXT = PFNGLMULTITEXSUBIMAGE3DEXTPROC(glfwGetProcAddress("glMultiTexSubImage3DEXT"));
 glNamedBufferData = PFNGLNAMEDBUFFERDATAPROC(glfwGetProcAddress("glNamedBufferData"));
 glNamedBufferDataEXT = PFNGLNAMEDBUFFERDATAEXTPROC(glfwGetProcAddress("glNamedBufferDataEXT"));
 glNamedBufferPageCommitmentARB = PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC(glfwGetProcAddress("glNamedBufferPageCommitmentARB"));
 glNamedBufferPageCommitmentEXT = PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glNamedBufferPageCommitmentEXT"));
 glNamedBufferStorage = PFNGLNAMEDBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedBufferStorage"));
 glNamedBufferStorageEXT = PFNGLNAMEDBUFFERSTORAGEEXTPROC(glfwGetProcAddress("glNamedBufferStorageEXT"));
 glNamedBufferSubData = PFNGLNAMEDBUFFERSUBDATAPROC(glfwGetProcAddress("glNamedBufferSubData"));
 glNamedBufferSubDataEXT = PFNGLNAMEDBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedBufferSubDataEXT"));
 glNamedCopyBufferSubDataEXT = PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC(glfwGetProcAddress("glNamedCopyBufferSubDataEXT"));
 glNamedFramebufferDrawBuffer = PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffer"));
 glNamedFramebufferDrawBuffers = PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC(glfwGetProcAddress("glNamedFramebufferDrawBuffers"));
 glNamedFramebufferParameteri = PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC(glfwGetProcAddress("glNamedFramebufferParameteri"));
 glNamedFramebufferParameteriEXT = PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC(glfwGetProcAddress("glNamedFramebufferParameteriEXT"));
 glNamedFramebufferReadBuffer = PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC(glfwGetProcAddress("glNamedFramebufferReadBuffer"));
 glNamedFramebufferRenderbuffer = PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC(glfwGetProcAddress("glNamedFramebufferRenderbuffer"));
 glNamedFramebufferRenderbufferEXT = PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC(glfwGetProcAddress("glNamedFramebufferRenderbufferEXT"));
 glNamedFramebufferSampleLocationsfvARB = PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvARB"));
 glNamedFramebufferSampleLocationsfvNV = PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC(glfwGetProcAddress("glNamedFramebufferSampleLocationsfvNV"));
 glNamedFramebufferTexture = PFNGLNAMEDFRAMEBUFFERTEXTUREPROC(glfwGetProcAddress("glNamedFramebufferTexture"));
 glNamedFramebufferTexture1DEXT = PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture1DEXT"));
 glNamedFramebufferTexture2DEXT =
```

```
PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture2DEXT"));
 glNamedFramebufferTexture3DEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC(glfwGetProcAddress("glNamedFramebufferTexture3DEXT"));
 glNamedFramebufferTextureEXT =
PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureEXT"));
 glNamedFramebufferTextureFaceEXT =
PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC(glfwGetProcAddress("glNamedFramebufferTextureFaceEXT"));
 glNamedFramebufferTextureLayer =
PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC(glfwGetProcAddress("glNamedFramebufferTextureLayer"));
 glNamedFramebufferTextureLayerEXT =
PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC(glfwGetProcAddress("glNamedFramebufferTextureLayerEXT"));
 glNamedProgramLocalParameter4dEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dEXT"));
 glNamedProgramLocalParameter4dvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4dvEXT"));
 glNamedProgramLocalParameter4fEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fEXT"));
 glNamedProgramLocalParameter4fvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameter4fvEXT"));
 glNamedProgramLocalParameterI4iEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4iEXT"));
 glNamedProgramLocalParameterI4ivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4ivEXT"));
 glNamedProgramLocalParameterI4uiEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uiEXT"));
 glNamedProgramLocalParameterI4uivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameterI4uivEXT"));
 glNamedProgramLocalParameters4fvEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParameters4fvEXT"));
 glNamedProgramLocalParametersI4ivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4ivEXT"));
 glNamedProgramLocalParametersI4uivEXT =
PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC(glfwGetProcAddress("glNamedProgramLocalParametersI4uivEXT"));
 glNamedProgramStringEXT =
PFNGLNAMEDPROGRAMSTRINGEXTPROC(glfwGetProcAddress("glNamedProgramStringEXT"));
 glNamedRenderbufferStorage =
PFNGLNAMEDRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glNamedRenderbufferStorage"));
 glNamedRenderbufferStorageEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageEXT"));
 glNamedRenderbufferStorageMultisample =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisample"));
 glNamedRenderbufferStorageMultisampleCoverageEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleCovera
 glNamedRenderbufferStorageMultisampleEXT =
PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC(glfwGetProcAddress("glNamedRenderbufferStorageMultisampleEXT"));
 glNamedStringARB = PFNGLNAMEDSTRINGARBPROC(glfwGetProcAddress("glNamedStringARB"));
 glNormalFormatNV = PFNGLNORMALFORMATNVPROC(glfwGetProcAddress("glNormalFormatNV"));
 glObjectLabel = PFNGLOBJECTLABELPROC(glfwGetProcAddress("glObjectLabel"));
 glObjectPtrLabel = PFNGLOBJECTPTRLABELPROC(glfwGetProcAddress("glObjectPtrLabel"));
 glPatchParameterfv = PFNGLPATCHPARAMETERFVPROC(glfwGetProcAddress("glPatchParameterfv"));
 glPatchParameteri = PFNGLPATCHPARAMETERIPROC(glfwGetProcAddress("glPatchParameteri"));
 glPathCommandsNV = PFNGLPATHCOMMANDSNVPROC(glfwGetProcAddress("glPathCommandsNV"));
 glPathCoordsNV = PFNGLPATHCOORDSNVPROC(glfwGetProcAddress("glPathCoordsNV"));
 glPathCoverDepthFuncNV =
PFNGLPATHCOVERDEPTHFUNCNVPROC(glfwGetProcAddress("glPathCoverDepthFuncNV"));
 glPathDashArrayNV = PFNGLPATHDASHARRAYNVPROC(glfwGetProcAddress("glPathDashArrayNV"));
 glPathGlyphIndexArrayNV =
PFNGLPATHGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathGlyphIndexArrayNV"));
 glPathGlyphIndexRangeNV =
PFNGLPATHGLYPHINDEXRANGENVPROC(glfwGetProcAddress("glPathGlyphIndexRangeNV"));
 glPathGlyphRangeNV = PFNGLPATHGLYPHRANGENVPROC(glfwGetProcAddress("glPathGlyphRangeNV"));
 glPathGlyphsNV = PFNGLPATHGLYPHSNVPROC(glfwGetProcAddress("glPathGlyphsNV"));
 glPathMemoryGlyphIndexArrayNV =
PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC(glfwGetProcAddress("glPathMemoryGlyphIndexArrayNV"));
 glPathParameterfNV = PFNGLPATHPARAMETERFNVPROC(glfwGetProcAddress("glPathParameterfNV"));
 glPathParameterfvNV = PFNGLPATHPARAMETERFVNVPROC(glfwGetProcAddress("glPathParameterfvNV"));
 glPathParameteriNV = PFNGLPATHPARAMETERINVPROC(glfwGetProcAddress("glPathParameteriNV"));
 glPathParameterivNV = PFNGLPATHPARAMETERIVNVPROC(glfwGetProcAddress("glPathParameterivNV"));
 glPathStencilDepthOffsetNV =
PFNGLPATHSTENCILDEPTHOFFSETNVPROC(glfwGetProcAddress("glPathStencilDepthOffsetNV"));
 glPathStencilFuncNV = PFNGLPATHSTENCILFUNCNVPROC(glfwGetProcAddress("glPathStencilFuncNV"));
 glPathStringNV = PFNGLPATHSTRINGNVPROC(glfwGetProcAddress("glPathStringNV"));
 glPathSubCommandsNV = PFNGLPATHSUBCOMMANDSNVPROC(glfwGetProcAddress("glPathSubCommandsNV"));
 glPathSubCoordsNV = PFNGLPATHSUBCOORDSNVPROC(glfwGetProcAddress("glPathSubCoordsNV"));
 glPauseTransformFeedback =
PFNGLPAUSETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glPauseTransformFeedback"));
 glPixelStoref = PFNGLPIXELSTOREFPROC(glfwGetProcAddress("glPixelStoref"));
 glPixelStorei = PFNGLPIXELSTOREIPROC(glfwGetProcAddress("glPixelStorei"));
 glPointAlongPathNV = PFNGLPOINTALONGPATHNVPROC(glfwGetProcAddress("glPointAlongPathNV"));
 glPointParameterf = PFNGLPOINTPARAMETERFPROC(glfwGetProcAddress("glPointParameterf"));
 glPointParameterfv = PFNGLPOINTPARAMETERFVPROC(glfwGetProcAddress("glPointParameterfv"));
 glPointParameteri = PFNGLPOINTPARAMETERIPROC(glfwGetProcAddress("glPointParameteri"));
 glPointParameteriv = PFNGLPOINTPARAMETERIVPROC(glfwGetProcAddress("glPointParameteriv"));
 glPointSize = PFNGLPOINTSIZEPROC(glfwGetProcAddress("glPointSize"));
 glPolygonMode = PFNGLPOLYGONMODEPROC(glfwGetProcAddress("glPolygonMode"));
 glPolygonOffset = PFNGLPOLYGONOFFSETPROC(glfwGetProcAddress("glPolygonOffset"));
 glPolygonOffsetClampEXT =
```

```
PFNGLPOLYGONOFFSETCLAMPEXTPROC(glfwGetProcAddress("glPolygonOffsetClampEXT"));
 glPopDebugGroup = PFNGLPOPDEBUGGROUPPROC(glfwGetProcAddress("glPopDebugGroup"));
 glPopGroupMarkerEXT = PFNGLPOPGROUPMARKEREXTPROC(glfwGetProcAddress("glPopGroupMarkerEXT"));
 glPrimitiveBoundingBoxARB =
PFNGLPRIMITIVEBOUNDINGBOXARBPROC(glfwGetProcAddress("glPrimitiveBoundingBoxARB"));
 glPrimitiveRestartIndex =
PFNGLPRIMITIVERESTARTINDEXPROC(glfwGetProcAddress("glPrimitiveRestartIndex"));
 glProgramBinary = PFNGLPROGRAMBINARYPROC(glfwGetProcAddress("glProgramBinary"));
 glProgramParameteri = PFNGLPROGRAMPARAMETERIPROC(glfwGetProcAddress("glProgramParameteri"));
 glProgramParameteriARB =
PFNGLPROGRAMPARAMETERIARBPROC(glfwGetProcAddress("glProgramParameteriARB"));
 glProgramPathFragmentInputGenNV =
PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC(glfwGetProcAddress("glProgramPathFragmentInputGenNV"));
 glProgramUniform1d = PFNGLPROGRAMUNIFORM1DPROC(glfwGetProcAddress("glProgramUniform1d"));
 glProgramUniform1dEXT = PFNGLPROGRAMUNIFORM1DEXTPROC(glfwGetProcAddress("glProgramUniform1dEXT"));
 glProgramUniform1dv = PFNGLPROGRAMUNIFORM1DVPROC(glfwGetProcAddress("glProgramUniform1dv"));
 glProgramUniform1dvEXT =
PFNGLPROGRAMUNIFORM1DVEXTPROC(glfwGetProcAddress("glProgramUniform1dvEXT"));
 glProgramUniform1f = PFNGLPROGRAMUNIFORM1FPROC(glfwGetProcAddress("glProgramUniform1f"));
 glProgramUniform1fEXT = PFNGLPROGRAMUNIFORM1FEXTPROC(glfwGetProcAddress("glProgramUniform1fEXT"));
 glProgramUniform1fv = PFNGLPROGRAMUNIFORM1FVPROC(glfwGetProcAddress("glProgramUniform1fv"));
 glProgramUniform1fvEXT =
PFNGLPROGRAMUNIFORM1FVEXTPROC(glfwGetProcAddress("glProgramUniform1fvEXT"));
 glProgramUniform1i = PFNGLPROGRAMUNIFORM1IPROC(glfwGetProcAddress("glProgramUniform1i"));
 glProgramUniform1i64ARB =
PFNGLPROGRAMUNIFORM1I64ARBPROC(glfwGetProcAddress("glProgramUniform1i64ARB"));
 glProgramUniform1i64NV =
PFNGLPROGRAMUNIFORM1I64NVPROC(glfwGetProcAddress("glProgramUniform1i64NV"));
 glProgramUniform1i64vARB =
PFNGLPROGRAMUNIFORM1I64VARBPROC(glfwGetProcAddress("glProgramUniform1i64vARB"));
 glProgramUniform1i64vNV =
PFNGLPROGRAMUNIFORM1I64VNVPROC(glfwGetProcAddress("glProgramUniform1i64vNV"));
 glProgramUniform1iEXT = PFNGLPROGRAMUNIFORM1IEXTPROC(glfwGetProcAddress("glProgramUniform1iEXT"));
 glProgramUniform1iv = PFNGLPROGRAMUNIFORM1IVPROC(glfwGetProcAddress("glProgramUniform1iv"));
 glProgramUniform1ivEXT =
PFNGLPROGRAMUNIFORM1IVEXTPROC(glfwGetProcAddress("glProgramUniform1ivEXT"));
 glProgramUniform1ui = PFNGLPROGRAMUNIFORM1UIPROC(glfwGetProcAddress("glProgramUniform1ui"));
 glProgramUniform1ui64ARB =
PFNGLPROGRAMUNIFORM1UI64ARBPROC(glfwGetProcAddress("glProgramUniform1ui64ARB"));
 glProgramUniform1ui64NV =
PFNGLPROGRAMUNIFORM1UI64NVPROC(glfwGetProcAddress("glProgramUniform1ui64NV"));
 glProgramUniform1ui64vARB =
PFNGLPROGRAMUNIFORM1UI64VARBPROC(glfwGetProcAddress("glProgramUniform1ui64vARB"));
 glProgramUniform1ui64vNV =
PFNGLPROGRAMUNIFORM1UI64VNVPROC(glfwGetProcAddress("glProgramUniform1ui64vNV"));
 glProgramUniform1uiEXT =
PFNGLPROGRAMUNIFORM1UIEXTPROC(glfwGetProcAddress("glProgramUniform1uiEXT"));
 glProgramUniform1uiv = PFNGLPROGRAMUNIFORM1UIVPROC(glfwGetProcAddress("glProgramUniform1uiv"));
 glProgramUniform1uivEXT =
PFNGLPROGRAMUNIFORM1UIVEXTPROC(glfwGetProcAddress("glProgramUniform1uivEXT"));
 glProgramUniform2d = PFNGLPROGRAMUNIFORM2DPROC(glfwGetProcAddress("glProgramUniform2d"));
 glProgramUniform2dEXT = PFNGLPROGRAMUNIFORM2DEXTPROC(glfwGetProcAddress("glProgramUniform2dEXT"));
 glProgramUniform2dv = PFNGLPROGRAMUNIFORM2DVPROC(glfwGetProcAddress("glProgramUniform2dv"));
 glProgramUniform2dvEXT =
PFNGLPROGRAMUNIFORM2DVEXTPROC(glfwGetProcAddress("glProgramUniform2dvEXT"));
 glProgramUniform2f = PFNGLPROGRAMUNIFORM2FPROC(glfwGetProcAddress("glProgramUniform2f"));
 glProgramUniform2fEXT = PFNGLPROGRAMUNIFORM2FEXTPROC(glfwGetProcAddress("glProgramUniform2fEXT"));
 glProgramUniform2fv = PFNGLPROGRAMUNIFORM2FVPROC(glfwGetProcAddress("glProgramUniform2fv"));
 glProgramUniform2fvEXT =
PFNGLPROGRAMUNIFORM2FVEXTPROC(glfwGetProcAddress("glProgramUniform2fvEXT"));
 glProgramUniform2i = PFNGLPROGRAMUNIFORM2IPROC(glfwGetProcAddress("glProgramUniform2i"));
 glProgramUniform2i64ARB =
PFNGLPROGRAMUNIFORM2I64ARBPROC(glfwGetProcAddress("glProgramUniform2i64ARB"));
 glProgramUniform2i64NV =
PFNGLPROGRAMUNIFORM2I64NVPROC(glfwGetProcAddress("glProgramUniform2i64NV"));
 glProgramUniform2i64vARB =
PFNGLPROGRAMUNIFORM2I64VARBPROC(glfwGetProcAddress("glProgramUniform2i64vARB"));
 glProgramUniform2i64vNV =
PFNGLPROGRAMUNIFORM2I64VNVPROC(glfwGetProcAddress("glProgramUniform2i64vNV"));
 glProgramUniform2iEXT = PFNGLPROGRAMUNIFORM2IEXTPROC(glfwGetProcAddress("glProgramUniform2iEXT"));
 glProgramUniform2iv = PFNGLPROGRAMUNIFORM2IVPROC(glfwGetProcAddress("glProgramUniform2iv"));
 glProgramUniform2ivEXT =
PFNGLPROGRAMUNIFORM2IVEXTPROC(glfwGetProcAddress("glProgramUniform2ivEXT"));
 glProgramUniform2ui = PFNGLPROGRAMUNIFORM2UIPROC(glfwGetProcAddress("glProgramUniform2ui"));
 glProgramUniform2ui64ARB =
PFNGLPROGRAMUNIFORM2UI64ARBPROC(glfwGetProcAddress("glProgramUniform2ui64ARB"));
 glProgramUniform2ui64NV =
PFNGLPROGRAMUNIFORM2UI64NVPROC(glfwGetProcAddress("glProgramUniform2ui64NV"));
 glProgramUniform2ui64vARB =
PFNGLPROGRAMUNIFORM2UI64VARBPROC(glfwGetProcAddress("glProgramUniform2ui64vARB"));
 glProgramUniform2ui64vNV =
PFNGLPROGRAMUNIFORM2UI64VNVPROC(glfwGetProcAddress("glProgramUniform2ui64vNV"));
 glProgramUniform2uiEXT =
PFNGLPROGRAMUNIFORM2UIEXTPROC(glfwGetProcAddress("glProgramUniform2uiEXT"));
 glProgramUniform2uiv = PFNGLPROGRAMUNIFORM2UIVPROC(glfwGetProcAddress("glProgramUniform2uiv"));
 glProgramUniform2uivEXT =
PFNGLPROGRAMUNIFORM2UIVEXTPROC(glfwGetProcAddress("glProgramUniform2uivEXT"));
```

```
 glProgramUniform3d = PFNGLPROGRAMUNIFORM3DPROC(glfwGetProcAddress("glProgramUniform3d"));
 glProgramUniform3dEXT = PFNGLPROGRAMUNIFORM3DEXTPROC(glfwGetProcAddress("glProgramUniform3dEXT"));
 glProgramUniform3dv = PFNGLPROGRAMUNIFORM3DVPROC(glfwGetProcAddress("glProgramUniform3dv"));
 glProgramUniform3dvEXT = PFNGLPROGRAMUNIFORM3DVEXTPROC(glfwGetProcAddress("glProgramUniform3dvEXT"));
 glProgramUniform3f = PFNGLPROGRAMUNIFORM3FPROC(glfwGetProcAddress("glProgramUniform3f"));
 glProgramUniform3fEXT = PFNGLPROGRAMUNIFORM3FEXTPROC(glfwGetProcAddress("glProgramUniform3fEXT"));
 glProgramUniform3fv = PFNGLPROGRAMUNIFORM3FVPROC(glfwGetProcAddress("glProgramUniform3fv"));
 glProgramUniform3fvEXT = PFNGLPROGRAMUNIFORM3FVEXTPROC(glfwGetProcAddress("glProgramUniform3fvEXT"));
 glProgramUniform3i = PFNGLPROGRAMUNIFORM3IPROC(glfwGetProcAddress("glProgramUniform3i"));
 glProgramUniform3i64ARB = PFNGLPROGRAMUNIFORM3I64ARBPROC(glfwGetProcAddress("glProgramUniform3i64ARB"));
 glProgramUniform3i64NV = PFNGLPROGRAMUNIFORM3I64NVPROC(glfwGetProcAddress("glProgramUniform3i64NV"));
 glProgramUniform3i64vARB = PFNGLPROGRAMUNIFORM3I64VARBPROC(glfwGetProcAddress("glProgramUniform3i64vARB"));
 glProgramUniform3i64vNV = PFNGLPROGRAMUNIFORM3I64VNVPROC(glfwGetProcAddress("glProgramUniform3i64vNV"));
 glProgramUniform3iEXT = PFNGLPROGRAMUNIFORM3IEXTPROC(glfwGetProcAddress("glProgramUniform3iEXT"));
 glProgramUniform3iv = PFNGLPROGRAMUNIFORM3IVPROC(glfwGetProcAddress("glProgramUniform3iv"));
 glProgramUniform3ivEXT = PFNGLPROGRAMUNIFORM3IVEXTPROC(glfwGetProcAddress("glProgramUniform3ivEXT"));
 glProgramUniform3ui = PFNGLPROGRAMUNIFORM3UIPROC(glfwGetProcAddress("glProgramUniform3ui"));
 glProgramUniform3ui64ARB = PFNGLPROGRAMUNIFORM3UI64ARBPROC(glfwGetProcAddress("glProgramUniform3ui64ARB"));
 glProgramUniform3ui64NV = PFNGLPROGRAMUNIFORM3UI64NVPROC(glfwGetProcAddress("glProgramUniform3ui64NV"));
 glProgramUniform3ui64vARB = PFNGLPROGRAMUNIFORM3UI64VARBPROC(glfwGetProcAddress("glProgramUniform3ui64vARB"));
 glProgramUniform3ui64vNV = PFNGLPROGRAMUNIFORM3UI64VNVPROC(glfwGetProcAddress("glProgramUniform3ui64vNV"));
 glProgramUniform3uiEXT = PFNGLPROGRAMUNIFORM3UIEXTPROC(glfwGetProcAddress("glProgramUniform3uiEXT"));
 glProgramUniform3uiv = PFNGLPROGRAMUNIFORM3UIVPROC(glfwGetProcAddress("glProgramUniform3uiv"));
 glProgramUniform3uivEXT = PFNGLPROGRAMUNIFORM3UIVEXTPROC(glfwGetProcAddress("glProgramUniform3uivEXT"));
 glProgramUniform4d = PFNGLPROGRAMUNIFORM4DPROC(glfwGetProcAddress("glProgramUniform4d"));
 glProgramUniform4dEXT = PFNGLPROGRAMUNIFORM4DEXTPROC(glfwGetProcAddress("glProgramUniform4dEXT"));
 glProgramUniform4dv = PFNGLPROGRAMUNIFORM4DVPROC(glfwGetProcAddress("glProgramUniform4dv"));
 glProgramUniform4dvEXT = PFNGLPROGRAMUNIFORM4DVEXTPROC(glfwGetProcAddress("glProgramUniform4dvEXT"));
 glProgramUniform4f = PFNGLPROGRAMUNIFORM4FPROC(glfwGetProcAddress("glProgramUniform4f"));
 glProgramUniform4fEXT = PFNGLPROGRAMUNIFORM4FEXTPROC(glfwGetProcAddress("glProgramUniform4fEXT"));
 glProgramUniform4fv = PFNGLPROGRAMUNIFORM4FVPROC(glfwGetProcAddress("glProgramUniform4fv"));
 glProgramUniform4fvEXT = PFNGLPROGRAMUNIFORM4FVEXTPROC(glfwGetProcAddress("glProgramUniform4fvEXT"));
 glProgramUniform4i = PFNGLPROGRAMUNIFORM4IPROC(glfwGetProcAddress("glProgramUniform4i"));
 glProgramUniform4i64ARB = PFNGLPROGRAMUNIFORM4I64ARBPROC(glfwGetProcAddress("glProgramUniform4i64ARB"));
 glProgramUniform4i64NV = PFNGLPROGRAMUNIFORM4I64NVPROC(glfwGetProcAddress("glProgramUniform4i64NV"));
 glProgramUniform4i64vARB = PFNGLPROGRAMUNIFORM4I64VARBPROC(glfwGetProcAddress("glProgramUniform4i64vARB"));
 glProgramUniform4i64vNV = PFNGLPROGRAMUNIFORM4I64VNVPROC(glfwGetProcAddress("glProgramUniform4i64vNV"));
 glProgramUniform4iEXT = PFNGLPROGRAMUNIFORM4IEXTPROC(glfwGetProcAddress("glProgramUniform4iEXT"));
 glProgramUniform4iv = PFNGLPROGRAMUNIFORM4IVPROC(glfwGetProcAddress("glProgramUniform4iv"));
 glProgramUniform4ivEXT = PFNGLPROGRAMUNIFORM4IVEXTPROC(glfwGetProcAddress("glProgramUniform4ivEXT"));
 glProgramUniform4ui = PFNGLPROGRAMUNIFORM4UIPROC(glfwGetProcAddress("glProgramUniform4ui"));
 glProgramUniform4ui64ARB = PFNGLPROGRAMUNIFORM4UI64ARBPROC(glfwGetProcAddress("glProgramUniform4ui64ARB"));
 glProgramUniform4ui64NV = PFNGLPROGRAMUNIFORM4UI64NVPROC(glfwGetProcAddress("glProgramUniform4ui64NV"));
 glProgramUniform4ui64vARB = PFNGLPROGRAMUNIFORM4UI64VARBPROC(glfwGetProcAddress("glProgramUniform4ui64vARB"));
 glProgramUniform4ui64vNV = PFNGLPROGRAMUNIFORM4UI64VNVPROC(glfwGetProcAddress("glProgramUniform4ui64vNV"));
 glProgramUniform4uiEXT = PFNGLPROGRAMUNIFORM4UIEXTPROC(glfwGetProcAddress("glProgramUniform4uiEXT"));
 glProgramUniform4uiv = PFNGLPROGRAMUNIFORM4UIVPROC(glfwGetProcAddress("glProgramUniform4uiv"));
 glProgramUniform4uivEXT = PFNGLPROGRAMUNIFORM4UIVEXTPROC(glfwGetProcAddress("glProgramUniform4uivEXT"));
 glProgramUniformHandleui64ARB = PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glProgramUniformHandleui64ARB"));
 glProgramUniformHandleui64NV = PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glProgramUniformHandleui64NV"));
 glProgramUniformHandleui64vARB = PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glProgramUniformHandleui64vARB"));
 glProgramUniformHandleui64vNV = PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glProgramUniformHandleui64vNV"));
 glProgramUniformMatrix2dv = PFNGLPROGRAMUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glProgramUniformMatrix2dv"));
 glProgramUniformMatrix2dvEXT = PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2dvEXT"));
 glProgramUniformMatrix2fv =
```

```
PFNGLPROGRAMUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glProgramUniformMatrix2fv"));
 glProgramUniformMatrix2fvEXT =
PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2fvEXT"));
 glProgramUniformMatrix2x3dv =
PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dv"));
 glProgramUniformMatrix2x3dvEXT =
PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3dvEXT"));
 glProgramUniformMatrix2x3fv =
PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fv"));
 glProgramUniformMatrix2x3fvEXT =
PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x3fvEXT"));
 glProgramUniformMatrix2x4dv =
PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dv"));
 glProgramUniformMatrix2x4dvEXT =
PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4dvEXT"));
 glProgramUniformMatrix2x4fv =
PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fv"));
 glProgramUniformMatrix2x4fvEXT =
PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix2x4fvEXT"));
 glProgramUniformMatrix3dv =
PFNGLPROGRAMUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glProgramUniformMatrix3dv"));
 glProgramUniformMatrix3dvEXT =
PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3dvEXT"));
 glProgramUniformMatrix3fv =
PFNGLPROGRAMUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glProgramUniformMatrix3fv"));
 glProgramUniformMatrix3fvEXT =
PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3fvEXT"));
 glProgramUniformMatrix3x2dv =
PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dv"));
 glProgramUniformMatrix3x2dvEXT =
PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2dvEXT"));
 glProgramUniformMatrix3x2fv =
PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fv"));
 glProgramUniformMatrix3x2fvEXT =
PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x2fvEXT"));
 glProgramUniformMatrix3x4dv =
PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dv"));
 glProgramUniformMatrix3x4dvEXT =
PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4dvEXT"));
 glProgramUniformMatrix3x4fv =
PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fv"));
 glProgramUniformMatrix3x4fvEXT =
PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix3x4fvEXT"));
 glProgramUniformMatrix4dv =
PFNGLPROGRAMUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glProgramUniformMatrix4dv"));
 glProgramUniformMatrix4dvEXT =
PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4dvEXT"));
 glProgramUniformMatrix4fv =
PFNGLPROGRAMUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glProgramUniformMatrix4fv"));
 glProgramUniformMatrix4fvEXT =
PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4fvEXT"));
 glProgramUniformMatrix4x2dv =
PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dv"));
 glProgramUniformMatrix4x2dvEXT =
PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2dvEXT"));
 glProgramUniformMatrix4x2fv =
PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fv"));
 glProgramUniformMatrix4x2fvEXT =
PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x2fvEXT"));
 glProgramUniformMatrix4x3dv =
PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dv"));
 glProgramUniformMatrix4x3dvEXT =
PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3dvEXT"));
 glProgramUniformMatrix4x3fv =
PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fv"));
 glProgramUniformMatrix4x3fvEXT =
PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC(glfwGetProcAddress("glProgramUniformMatrix4x3fvEXT"));
 glProgramUniformui64NV =
PFNGLPROGRAMUNIFORMUI64NVPROC(glfwGetProcAddress("glProgramUniformui64NV"));
 glProgramUniformui64vNV =
PFNGLPROGRAMUNIFORMUI64VNVPROC(glfwGetProcAddress("glProgramUniformui64vNV"));
 glProvokingVertex = PFNGLPROVOKINGVERTEXPROC(glfwGetProcAddress("glProvokingVertex"));
 glPushClientAttribDefaultEXT =
PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC(glfwGetProcAddress("glPushClientAttribDefaultEXT"));
 glPushDebugGroup = PFNGLPUSHDEBUGGROUPPROC(glfwGetProcAddress("glPushDebugGroup"));
 glPushGroupMarkerEXT = PFNGLPUSHGROUPMARKEREXTPROC(glfwGetProcAddress("glPushGroupMarkerEXT"));
 glQueryCounter = PFNGLQUERYCOUNTERPROC(glfwGetProcAddress("glQueryCounter"));
 glRasterSamplesEXT = PFNGLRASTERSAMPLESEXTPROC(glfwGetProcAddress("glRasterSamplesEXT"));
 glReadBuffer = PFNGLREADBUFFERPROC(glfwGetProcAddress("glReadBuffer"));
 glReadPixels = PFNGLREADPIXELSPROC(glfwGetProcAddress("glReadPixels"));
 glReadnPixels = PFNGLREADNPIXELSPROC(glfwGetProcAddress("glReadnPixels"));
 glReadnPixelsARB = PFNGLREADNPIXELSARBPROC(glfwGetProcAddress("glReadnPixelsARB"));
 glReleaseShaderCompiler =
PFNGLRELEASESHADERCOMPILERPROC(glfwGetProcAddress("glReleaseShaderCompiler"));
 glRenderbufferStorage = PFNGLRENDERBUFFERSTORAGEPROC(glfwGetProcAddress("glRenderbufferStorage"));
 glRenderbufferStorageMultisample =
PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC(glfwGetProcAddress("glRenderbufferStorageMultisample"));
```

```
 glRenderbufferStorageMultisampleCoverageNV = PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC(glfwGetProcAddress("glRenderbufferStorageMultisampleCoverageNV"));
 glResolveDepthValuesNV = PFNGLRESOLVEDEPTHVALUESNVPROC(glfwGetProcAddress("glResolveDepthValuesNV"));
 glResumeTransformFeedback = PFNGLRESUMETRANSFORMFEEDBACKPROC(glfwGetProcAddress("glResumeTransformFeedback"));
 glSampleCoverage = PFNGLSAMPLECOVERAGEPROC(glfwGetProcAddress("glSampleCoverage"));
 glSampleMaski = PFNGLSAMPLEMASKIPROC(glfwGetProcAddress("glSampleMaski"));
 glSamplerParameterIiv = PFNGLSAMPLERPARAMETERIIVPROC(glfwGetProcAddress("glSamplerParameterIiv"));
 glSamplerParameterIuiv = PFNGLSAMPLERPARAMETERIUIVPROC(glfwGetProcAddress("glSamplerParameterIuiv"));
 glSamplerParameterf = PFNGLSAMPLERPARAMETERFPROC(glfwGetProcAddress("glSamplerParameterf"));
 glSamplerParameterfv = PFNGLSAMPLERPARAMETERFVPROC(glfwGetProcAddress("glSamplerParameterfv"));
 glSamplerParameteri = PFNGLSAMPLERPARAMETERIPROC(glfwGetProcAddress("glSamplerParameteri"));
 glSamplerParameteriv = PFNGLSAMPLERPARAMETERIVPROC(glfwGetProcAddress("glSamplerParameteriv"));
 glScissor = PFNGLSCISSORPROC(glfwGetProcAddress("glScissor"));
 glScissorArrayv = PFNGLSCISSORARRAYVPROC(glfwGetProcAddress("glScissorArrayv"));
 glScissorIndexed = PFNGLSCISSORINDEXEDPROC(glfwGetProcAddress("glScissorIndexed"));
 glScissorIndexedv = PFNGLSCISSORINDEXEDVPROC(glfwGetProcAddress("glScissorIndexedv"));
 glSecondaryColorFormatNV = PFNGLSECONDARYCOLORFORMATNVPROC(glfwGetProcAddress("glSecondaryColorFormatNV"));
 glSelectPerfMonitorCountersAMD = PFNGLSELECTPERFMONITORCOUNTERSAMDPROC(glfwGetProcAddress("glSelectPerfMonitorCountersAMD"));
 glShaderBinary = PFNGLSHADERBINARYPROC(glfwGetProcAddress("glShaderBinary"));
 glShaderSource = PFNGLSHADERSOURCEPROC(glfwGetProcAddress("glShaderSource"));
 glShaderStorageBlockBinding = PFNGLSHADERSTORAGEBLOCKBINDINGPROC(glfwGetProcAddress("glShaderStorageBlockBinding"));
 glSignalVkFenceNV = PFNGLSIGNALVKFENCENVPROC(glfwGetProcAddress("glSignalVkFenceNV"));
 glSignalVkSemaphoreNV = PFNGLSIGNALVKSEMAPHORENVPROC(glfwGetProcAddress("glSignalVkSemaphoreNV"));
 glSpecializeShaderARB = PFNGLSPECIALIZESHADERARBPROC(glfwGetProcAddress("glSpecializeShaderARB"));
 glStateCaptureNV = PFNGLSTATECAPTURENVPROC(glfwGetProcAddress("glStateCaptureNV"));
 glStencilFillPathInstancedNV = PFNGLSTENCILFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilFillPathInstancedNV"));
 glStencilFillPathNV = PFNGLSTENCILFILLPATHNVPROC(glfwGetProcAddress("glStencilFillPathNV"));
 glStencilFunc = PFNGLSTENCILFUNCPROC(glfwGetProcAddress("glStencilFunc"));
 glStencilFuncSeparate = PFNGLSTENCILFUNCSEPARATEPROC(glfwGetProcAddress("glStencilFuncSeparate"));
 glStencilMask = PFNGLSTENCILMASKPROC(glfwGetProcAddress("glStencilMask"));
 glStencilMaskSeparate = PFNGLSTENCILMASKSEPARATEPROC(glfwGetProcAddress("glStencilMaskSeparate"));
 glStencilOp = PFNGLSTENCILOPPROC(glfwGetProcAddress("glStencilOp"));
 glStencilOpSeparate = PFNGLSTENCILOPSEPARATEPROC(glfwGetProcAddress("glStencilOpSeparate"));
 glStencilStrokePathInstancedNV = PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilStrokePathInstancedNV"));
 glStencilStrokePathNV = PFNGLSTENCILSTROKEPATHNVPROC(glfwGetProcAddress("glStencilStrokePathNV"));
 glStencilThenCoverFillPathInstancedNV = PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathInstancedNV"));
 glStencilThenCoverFillPathNV = PFNGLSTENCILTHENCOVERFILLPATHNVPROC(glfwGetProcAddress("glStencilThenCoverFillPathNV"));
 glStencilThenCoverStrokePathInstancedNV = PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathInstancedNV"));
 glStencilThenCoverStrokePathNV = PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC(glfwGetProcAddress("glStencilThenCoverStrokePathNV"));
 glSubpixelPrecisionBiasNV = PFNGLSUBPIXELPRECISIONBIASNVPROC(glfwGetProcAddress("glSubpixelPrecisionBiasNV"));
 glTexBuffer = PFNGLTEXBUFFERPROC(glfwGetProcAddress("glTexBuffer"));
 glTexBufferARB = PFNGLTEXBUFFERARBPROC(glfwGetProcAddress("glTexBufferARB"));
 glTexBufferRange = PFNGLTEXBUFFERRANGEPROC(glfwGetProcAddress("glTexBufferRange"));
 glTexCoordFormatNV = PFNGLTEXCOORDFORMATNVPROC(glfwGetProcAddress("glTexCoordFormatNV"));
 glTexImage1D = PFNGLTEXIMAGE1DPROC(glfwGetProcAddress("glTexImage1D"));
 glTexImage2D = PFNGLTEXIMAGE2DPROC(glfwGetProcAddress("glTexImage2D"));
 glTexImage2DMultisample = PFNGLTEXIMAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage2DMultisample"));
 glTexImage3D = PFNGLTEXIMAGE3DPROC(glfwGetProcAddress("glTexImage3D"));
 glTexImage3DMultisample = PFNGLTEXIMAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexImage3DMultisample"));
 glTexPageCommitmentARB = PFNGLTEXPAGECOMMITMENTARBPROC(glfwGetProcAddress("glTexPageCommitmentARB"));
 glTexParameterIiv = PFNGLTEXPARAMETERIIVPROC(glfwGetProcAddress("glTexParameterIiv"));
 glTexParameterIuiv = PFNGLTEXPARAMETERIUIVPROC(glfwGetProcAddress("glTexParameterIuiv"));
 glTexParameterf = PFNGLTEXPARAMETERFPROC(glfwGetProcAddress("glTexParameterf"));
 glTexParameterfv = PFNGLTEXPARAMETERFVPROC(glfwGetProcAddress("glTexParameterfv"));
 glTexParameteri = PFNGLTEXPARAMETERIPROC(glfwGetProcAddress("glTexParameteri"));
 glTexParameteriv = PFNGLTEXPARAMETERIVPROC(glfwGetProcAddress("glTexParameteriv"));
 glTexStorage1D = PFNGLTEXSTORAGE1DPROC(glfwGetProcAddress("glTexStorage1D"));
 glTexStorage2D = PFNGLTEXSTORAGE2DPROC(glfwGetProcAddress("glTexStorage2D"));
 glTexStorage2DMultisample = PFNGLTEXSTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage2DMultisample"));
 glTexStorage3D = PFNGLTEXSTORAGE3DPROC(glfwGetProcAddress("glTexStorage3D"));
 glTexStorage3DMultisample = PFNGLTEXSTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTexStorage3DMultisample"));
 glTexSubImage1D = PFNGLTEXSUBIMAGE1DPROC(glfwGetProcAddress("glTexSubImage1D"));
 glTexSubImage2D = PFNGLTEXSUBIMAGE2DPROC(glfwGetProcAddress("glTexSubImage2D"));
 glTexSubImage3D = PFNGLTEXSUBIMAGE3DPROC(glfwGetProcAddress("glTexSubImage3D"));
 glTextureBarrier = PFNGLTEXTUREBARRIERPROC(glfwGetProcAddress("glTextureBarrier"));
 glTextureBarrierNV = PFNGLTEXTUREBARRIERNVPROC(glfwGetProcAddress("glTextureBarrierNV"));
 glTextureBuffer = PFNGLTEXTUREBUFFERPROC(glfwGetProcAddress("glTextureBuffer"));
 glTextureBufferEXT = PFNGLTEXTUREBUFFEREXTPROC(glfwGetProcAddress("glTextureBufferEXT"));
 glTextureBufferRange = PFNGLTEXTUREBUFFERRANGEPROC(glfwGetProcAddress("glTextureBufferRange"));
```

```
glTextureBufferRangeEXT = PFNGLTEXTUREBUFFERRANGEEXTPROC(glfwGetProcAddress("glTextureBufferRangeEXT"));
glTextureImage1DEXT = PFNGLTEXTUREIMAGE1DEXTPROC(glfwGetProcAddress("glTextureImage1DEXT"));
glTextureImage2DEXT = PFNGLTEXTUREIMAGE2DEXTPROC(glfwGetProcAddress("glTextureImage2DEXT"));
glTextureImage3DEXT = PFNGLTEXTUREIMAGE3DEXTPROC(glfwGetProcAddress("glTextureImage3DEXT"));
glTexturePageCommitmentEXT = PFNGLTEXTUREPAGECOMMITMENTEXTPROC(glfwGetProcAddress("glTexturePageCommitmentEXT"));
glTextureParameterIiv = PFNGLTEXTUREPARAMETERIIVPROC(glfwGetProcAddress("glTextureParameterIiv"));
glTextureParameterIivEXT = PFNGLTEXTUREPARAMETERIIVEXTPROC(glfwGetProcAddress("glTextureParameterIivEXT"));
glTextureParameterIuiv = PFNGLTEXTUREPARAMETERIUIVPROC(glfwGetProcAddress("glTextureParameterIuiv"));
glTextureParameterIuivEXT = PFNGLTEXTUREPARAMETERIUIVEXTPROC(glfwGetProcAddress("glTextureParameterIuivEXT"));
glTextureParameterf = PFNGLTEXTUREPARAMETERFPROC(glfwGetProcAddress("glTextureParameterf"));
glTextureParameterfEXT = PFNGLTEXTUREPARAMETERFEXTPROC(glfwGetProcAddress("glTextureParameterfEXT"));
glTextureParameterfv = PFNGLTEXTUREPARAMETERFVPROC(glfwGetProcAddress("glTextureParameterfv"));
glTextureParameterfvEXT = PFNGLTEXTUREPARAMETERFVEXTPROC(glfwGetProcAddress("glTextureParameterfvEXT"));
glTextureParameteri = PFNGLTEXTUREPARAMETERIPROC(glfwGetProcAddress("glTextureParameteri"));
glTextureParameteriEXT = PFNGLTEXTUREPARAMETERIEXTPROC(glfwGetProcAddress("glTextureParameteriEXT"));
glTextureParameteriv = PFNGLTEXTUREPARAMETERIVPROC(glfwGetProcAddress("glTextureParameteriv"));
glTextureParameterivEXT = PFNGLTEXTUREPARAMETERIVEXTPROC(glfwGetProcAddress("glTextureParameterivEXT"));
glTextureRenderbufferEXT = PFNGLTEXTURERENDERBUFFEREXTPROC(glfwGetProcAddress("glTextureRenderbufferEXT"));
glTextureStorage1D = PFNGLTEXTURESTORAGE1DPROC(glfwGetProcAddress("glTextureStorage1D"));
glTextureStorage1DEXT = PFNGLTEXTURESTORAGE1DEXTPROC(glfwGetProcAddress("glTextureStorage1DEXT"));
glTextureStorage2D = PFNGLTEXTURESTORAGE2DPROC(glfwGetProcAddress("glTextureStorage2D"));
glTextureStorage2DEXT = PFNGLTEXTURESTORAGE2DEXTPROC(glfwGetProcAddress("glTextureStorage2DEXT"));
glTextureStorage2DMultisample = PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage2DMultisample"));
glTextureStorage2DMultisampleEXT = PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage2DMultisampleEXT"));
glTextureStorage3D = PFNGLTEXTURESTORAGE3DPROC(glfwGetProcAddress("glTextureStorage3D"));
glTextureStorage3DEXT = PFNGLTEXTURESTORAGE3DEXTPROC(glfwGetProcAddress("glTextureStorage3DEXT"));
glTextureStorage3DMultisample = PFNGLTEXTURESTORAGE3DMULTISAMPLEPROC(glfwGetProcAddress("glTextureStorage3DMultisample"));
glTextureStorage3DMultisampleEXT = PFNGLTEXTURESTORAGE3DMULTISAMPLEEXTPROC(glfwGetProcAddress("glTextureStorage3DMultisampleEXT"));
glTextureSubImage1D = PFNGLTEXTURESUBIMAGE1DPROC(glfwGetProcAddress("glTextureSubImage1D"));
glTextureSubImage1DEXT = PFNGLTEXTURESUBIMAGE1DEXTPROC(glfwGetProcAddress("glTextureSubImage1DEXT"));
glTextureSubImage2D = PFNGLTEXTURESUBIMAGE2DPROC(glfwGetProcAddress("glTextureSubImage2D"));
glTextureSubImage2DEXT = PFNGLTEXTURESUBIMAGE2DEXTPROC(glfwGetProcAddress("glTextureSubImage2DEXT"));
glTextureSubImage3D = PFNGLTEXTURESUBIMAGE3DPROC(glfwGetProcAddress("glTextureSubImage3D"));
glTextureSubImage3DEXT = PFNGLTEXTURESUBIMAGE3DEXTPROC(glfwGetProcAddress("glTextureSubImage3DEXT"));
glTextureView = PFNGLTEXTUREVIEWPROC(glfwGetProcAddress("glTextureView"));
glTransformFeedbackBufferBase = PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC(glfwGetProcAddress("glTransformFeedbackBufferBase"));
glTransformFeedbackBufferRange = PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC(glfwGetProcAddress("glTransformFeedbackBufferRange"));
glTransformFeedbackVaryings = PFNGLTRANSFORMFEEDBACKVARYINGSPROC(glfwGetProcAddress("glTransformFeedbackVaryings"));
glTransformPathNV = PFNGLTRANSFORMPATHNVPROC(glfwGetProcAddress("glTransformPathNV"));
glUniform1d = PFNGLUNIFORM1DPROC(glfwGetProcAddress("glUniform1d"));
glUniform1dv = PFNGLUNIFORM1DVPROC(glfwGetProcAddress("glUniform1dv"));
glUniform1f = PFNGLUNIFORM1FPROC(glfwGetProcAddress("glUniform1f"));
glUniform1fv = PFNGLUNIFORM1FVPROC(glfwGetProcAddress("glUniform1fv"));
glUniform1i = PFNGLUNIFORM1IPROC(glfwGetProcAddress("glUniform1i"));
glUniform1i64ARB = PFNGLUNIFORM1I64ARBPROC(glfwGetProcAddress("glUniform1i64ARB"));
glUniform1i64NV = PFNGLUNIFORM1I64NVPROC(glfwGetProcAddress("glUniform1i64NV"));
glUniform1i64vARB = PFNGLUNIFORM1I64VARBPROC(glfwGetProcAddress("glUniform1i64vARB"));
glUniform1i64vNV = PFNGLUNIFORM1I64VNVPROC(glfwGetProcAddress("glUniform1i64vNV"));
glUniform1iv = PFNGLUNIFORM1IVPROC(glfwGetProcAddress("glUniform1iv"));
glUniform1ui = PFNGLUNIFORM1UIPROC(glfwGetProcAddress("glUniform1ui"));
glUniform1ui64ARB = PFNGLUNIFORM1UI64ARBPROC(glfwGetProcAddress("glUniform1ui64ARB"));
glUniform1ui64NV = PFNGLUNIFORM1UI64NVPROC(glfwGetProcAddress("glUniform1ui64NV"));
glUniform1ui64vARB = PFNGLUNIFORM1UI64VARBPROC(glfwGetProcAddress("glUniform1ui64vARB"));
glUniform1ui64vNV = PFNGLUNIFORM1UI64VNVPROC(glfwGetProcAddress("glUniform1ui64vNV"));
glUniform1uiv = PFNGLUNIFORM1UIVPROC(glfwGetProcAddress("glUniform1uiv"));
glUniform2d = PFNGLUNIFORM2DPROC(glfwGetProcAddress("glUniform2d"));
glUniform2dv = PFNGLUNIFORM2DVPROC(glfwGetProcAddress("glUniform2dv"));
glUniform2f = PFNGLUNIFORM2FPROC(glfwGetProcAddress("glUniform2f"));
glUniform2fv = PFNGLUNIFORM2FVPROC(glfwGetProcAddress("glUniform2fv"));
glUniform2i = PFNGLUNIFORM2IPROC(glfwGetProcAddress("glUniform2i"));
glUniform2i64ARB = PFNGLUNIFORM2I64ARBPROC(glfwGetProcAddress("glUniform2i64ARB"));
glUniform2i64NV = PFNGLUNIFORM2I64NVPROC(glfwGetProcAddress("glUniform2i64NV"));
glUniform2i64vARB = PFNGLUNIFORM2I64VARBPROC(glfwGetProcAddress("glUniform2i64vARB"));
glUniform2i64vNV = PFNGLUNIFORM2I64VNVPROC(glfwGetProcAddress("glUniform2i64vNV"));
glUniform2iv = PFNGLUNIFORM2IVPROC(glfwGetProcAddress("glUniform2iv"));
glUniform2ui = PFNGLUNIFORM2UIPROC(glfwGetProcAddress("glUniform2ui"));
glUniform2ui64ARB = PFNGLUNIFORM2UI64ARBPROC(glfwGetProcAddress("glUniform2ui64ARB"));
```

```
glUniform2ui64NV = PFNGLUNIFORM2UI64NVPROC(glfwGetProcAddress("glUniform2ui64NV"));
glUniform2ui64vARB = PFNGLUNIFORM2UI64VARBPROC(glfwGetProcAddress("glUniform2ui64vARB"));
glUniform2ui64vNV = PFNGLUNIFORM2UI64VNVPROC(glfwGetProcAddress("glUniform2ui64vNV"));
glUniform2uiv = PFNGLUNIFORM2UIVPROC(glfwGetProcAddress("glUniform2uiv"));
glUniform3d = PFNGLUNIFORM3DPROC(glfwGetProcAddress("glUniform3d"));
glUniform3dv = PFNGLUNIFORM3DVPROC(glfwGetProcAddress("glUniform3dv"));
glUniform3f = PFNGLUNIFORM3FPROC(glfwGetProcAddress("glUniform3f"));
glUniform3fv = PFNGLUNIFORM3FVPROC(glfwGetProcAddress("glUniform3fv"));
glUniform3i = PFNGLUNIFORM3IPROC(glfwGetProcAddress("glUniform3i"));
glUniform3i64ARB = PFNGLUNIFORM3I64ARBPROC(glfwGetProcAddress("glUniform3i64ARB"));
glUniform3i64NV = PFNGLUNIFORM3I64NVPROC(glfwGetProcAddress("glUniform3i64NV"));
glUniform3i64vARB = PFNGLUNIFORM3I64VARBPROC(glfwGetProcAddress("glUniform3i64vARB"));
glUniform3i64vNV = PFNGLUNIFORM3I64VNVPROC(glfwGetProcAddress("glUniform3i64vNV"));
glUniform3iv = PFNGLUNIFORM3IVPROC(glfwGetProcAddress("glUniform3iv"));
glUniform3ui = PFNGLUNIFORM3UIPROC(glfwGetProcAddress("glUniform3ui"));
glUniform3ui64ARB = PFNGLUNIFORM3UI64ARBPROC(glfwGetProcAddress("glUniform3ui64ARB"));
glUniform3ui64NV = PFNGLUNIFORM3UI64NVPROC(glfwGetProcAddress("glUniform3ui64NV"));
glUniform3ui64vARB = PFNGLUNIFORM3UI64VARBPROC(glfwGetProcAddress("glUniform3ui64vARB"));
glUniform3ui64vNV = PFNGLUNIFORM3UI64VNVPROC(glfwGetProcAddress("glUniform3ui64vNV"));
glUniform3uiv = PFNGLUNIFORM3UIVPROC(glfwGetProcAddress("glUniform3uiv"));
glUniform4d = PFNGLUNIFORM4DPROC(glfwGetProcAddress("glUniform4d"));
glUniform4dv = PFNGLUNIFORM4DVPROC(glfwGetProcAddress("glUniform4dv"));
glUniform4f = PFNGLUNIFORM4FPROC(glfwGetProcAddress("glUniform4f"));
glUniform4fv = PFNGLUNIFORM4FVPROC(glfwGetProcAddress("glUniform4fv"));
glUniform4i = PFNGLUNIFORM4IPROC(glfwGetProcAddress("glUniform4i"));
glUniform4i64ARB = PFNGLUNIFORM4I64ARBPROC(glfwGetProcAddress("glUniform4i64ARB"));
glUniform4i64NV = PFNGLUNIFORM4I64NVPROC(glfwGetProcAddress("glUniform4i64NV"));
glUniform4i64vARB = PFNGLUNIFORM4I64VARBPROC(glfwGetProcAddress("glUniform4i64vARB"));
glUniform4i64vNV = PFNGLUNIFORM4I64VNVPROC(glfwGetProcAddress("glUniform4i64vNV"));
glUniform4iv = PFNGLUNIFORM4IVPROC(glfwGetProcAddress("glUniform4iv"));
glUniform4ui = PFNGLUNIFORM4UIPROC(glfwGetProcAddress("glUniform4ui"));
glUniform4ui64ARB = PFNGLUNIFORM4UI64ARBPROC(glfwGetProcAddress("glUniform4ui64ARB"));
glUniform4ui64NV = PFNGLUNIFORM4UI64NVPROC(glfwGetProcAddress("glUniform4ui64NV"));
glUniform4ui64vARB = PFNGLUNIFORM4UI64VARBPROC(glfwGetProcAddress("glUniform4ui64vARB"));
glUniform4ui64vNV = PFNGLUNIFORM4UI64VNVPROC(glfwGetProcAddress("glUniform4ui64vNV"));
glUniform4uiv = PFNGLUNIFORM4UIVPROC(glfwGetProcAddress("glUniform4uiv"));
glUniformBlockBinding = PFNGLUNIFORMBLOCKBINDINGPROC(glfwGetProcAddress("glUniformBlockBinding"));
glUniformHandleui64ARB = PFNGLUNIFORMHANDLEUI64ARBPROC(glfwGetProcAddress("glUniformHandleui64ARB"));
glUniformHandleui64NV = PFNGLUNIFORMHANDLEUI64NVPROC(glfwGetProcAddress("glUniformHandleui64NV"));
glUniformHandleui64vARB = PFNGLUNIFORMHANDLEUI64VARBPROC(glfwGetProcAddress("glUniformHandleui64vARB"));
glUniformHandleui64vNV = PFNGLUNIFORMHANDLEUI64VNVPROC(glfwGetProcAddress("glUniformHandleui64vNV"));
glUniformMatrix2dv = PFNGLUNIFORMMATRIX2DVPROC(glfwGetProcAddress("glUniformMatrix2dv"));
glUniformMatrix2fv = PFNGLUNIFORMMATRIX2FVPROC(glfwGetProcAddress("glUniformMatrix2fv"));
glUniformMatrix2x3dv = PFNGLUNIFORMMATRIX2X3DVPROC(glfwGetProcAddress("glUniformMatrix2x3dv"));
glUniformMatrix2x3fv = PFNGLUNIFORMMATRIX2X3FVPROC(glfwGetProcAddress("glUniformMatrix2x3fv"));
glUniformMatrix2x4dv = PFNGLUNIFORMMATRIX2X4DVPROC(glfwGetProcAddress("glUniformMatrix2x4dv"));
glUniformMatrix2x4fv = PFNGLUNIFORMMATRIX2X4FVPROC(glfwGetProcAddress("glUniformMatrix2x4fv"));
glUniformMatrix3dv = PFNGLUNIFORMMATRIX3DVPROC(glfwGetProcAddress("glUniformMatrix3dv"));
glUniformMatrix3fv = PFNGLUNIFORMMATRIX3FVPROC(glfwGetProcAddress("glUniformMatrix3fv"));
glUniformMatrix3x2dv = PFNGLUNIFORMMATRIX3X2DVPROC(glfwGetProcAddress("glUniformMatrix3x2dv"));
glUniformMatrix3x2fv = PFNGLUNIFORMMATRIX3X2FVPROC(glfwGetProcAddress("glUniformMatrix3x2fv"));
glUniformMatrix3x4dv = PFNGLUNIFORMMATRIX3X4DVPROC(glfwGetProcAddress("glUniformMatrix3x4dv"));
glUniformMatrix3x4fv = PFNGLUNIFORMMATRIX3X4FVPROC(glfwGetProcAddress("glUniformMatrix3x4fv"));
glUniformMatrix4dv = PFNGLUNIFORMMATRIX4DVPROC(glfwGetProcAddress("glUniformMatrix4dv"));
glUniformMatrix4fv = PFNGLUNIFORMMATRIX4FVPROC(glfwGetProcAddress("glUniformMatrix4fv"));
glUniformMatrix4x2dv = PFNGLUNIFORMMATRIX4X2DVPROC(glfwGetProcAddress("glUniformMatrix4x2dv"));
glUniformMatrix4x2fv = PFNGLUNIFORMMATRIX4X2FVPROC(glfwGetProcAddress("glUniformMatrix4x2fv"));
glUniformMatrix4x3dv = PFNGLUNIFORMMATRIX4X3DVPROC(glfwGetProcAddress("glUniformMatrix4x3dv"));
glUniformMatrix4x3fv = PFNGLUNIFORMMATRIX4X3FVPROC(glfwGetProcAddress("glUniformMatrix4x3fv"));
glUniformSubroutinesuiv = PFNGLUNIFORMSUBROUTINESUIVPROC(glfwGetProcAddress("glUniformSubroutinesuiv"));
glUniformui64NV = PFNGLUNIFORMUI64NVPROC(glfwGetProcAddress("glUniformui64NV"));
glUniformui64vNV = PFNGLUNIFORMUI64VNVPROC(glfwGetProcAddress("glUniformui64vNV"));
glUnmapBuffer = PFNGLUNMAPBUFFERPROC(glfwGetProcAddress("glUnmapBuffer"));
glUnmapNamedBuffer = PFNGLUNMAPNAMEDBUFFERPROC(glfwGetProcAddress("glUnmapNamedBuffer"));
glUnmapNamedBufferEXT = PFNGLUNMAPNAMEDBUFFEREXTPROC(glfwGetProcAddress("glUnmapNamedBufferEXT"));
glUseProgram = PFNGLUSEPROGRAMPROC(glfwGetProcAddress("glUseProgram"));
glUseProgramStages = PFNGLUSEPROGRAMSTAGESPROC(glfwGetProcAddress("glUseProgramStages"));
glUseShaderProgramEXT = PFNGLUSESHADERPROGRAMEXTPROC(glfwGetProcAddress("glUseShaderProgramEXT"));
glValidateProgram = PFNGLVALIDATEPROGRAMPROC(glfwGetProcAddress("glValidateProgram"));
glValidateProgramPipeline = PFNGLVALIDATEPROGRAMPIPELINEPROC(glfwGetProcAddress("glValidateProgramPipeline"));
glVertexArrayAttribBinding = PFNGLVERTEXARRAYATTRIBBINDINGPROC(glfwGetProcAddress("glVertexArrayAttribBinding"));
glVertexArrayAttribFormat = PFNGLVERTEXARRAYATTRIBFORMATPROC(glfwGetProcAddress("glVertexArrayAttribFormat"));
glVertexArrayAttribIFormat = PFNGLVERTEXARRAYATTRIBIFORMATPROC(glfwGetProcAddress("glVertexArrayAttribIFormat"));
glVertexArrayAttribLFormat = PFNGLVERTEXARRAYATTRIBLFORMATPROC(glfwGetProcAddress("glVertexArrayAttribLFormat"));
glVertexArrayBindVertexBufferEXT = PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC(glfwGetProcAddress("glVertexArrayBindVertexBufferEXT"));
glVertexArrayBindingDivisor = PFNGLVERTEXARRAYBINDINGDIVISORPROC(glfwGetProcAddress("glVertexArrayBindingDivisor"));
```

```
 glVertexArrayColorOffsetEXT = PFNGLVERTEXARRAYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArrayColorOffsetEXT"));
 glVertexArrayEdgeFlagOffsetEXT = PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayEdgeFlagOffsetEXT"));
 glVertexArrayElementBuffer = PFNGLVERTEXARRAYELEMENTBUFFERPROC(glfwGetProcAddress("glVertexArrayElementBuffer"));
 glVertexArrayFogCoordOffsetEXT = PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayFogCoordOffsetEXT"));
 glVertexArrayIndexOffsetEXT = PFNGLVERTEXARRAYINDEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayIndexOffsetEXT"));
 glVertexArrayMultiTexCoordOffsetEXT = PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayMultiTexCoordOffsetEXT"));
 glVertexArrayNormalOffsetEXT = PFNGLVERTEXARRAYNORMALOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayNormalOffsetEXT"));
 glVertexArraySecondaryColorOffsetEXT = PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC(glfwGetProcAddress("glVertexArraySecondaryColorOffsetEXT"));
 glVertexArrayTexCoordOffsetEXT = PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayTexCoordOffsetEXT"));
 glVertexArrayVertexAttribBindingEXT = PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribBindingEXT"));
 glVertexArrayVertexAttribDivisorEXT = PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribDivisorEXT"));
 glVertexArrayVertexAttribFormatEXT = PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribFormatEXT"));
 glVertexArrayVertexAttribIFormatEXT = PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIFormatEXT"));
 glVertexArrayVertexAttribIOffsetEXT = PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribIOffsetEXT"));
 glVertexArrayVertexAttribLFormatEXT = PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLFormatEXT"));
 glVertexArrayVertexAttribLOffsetEXT = PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribLOffsetEXT"));
 glVertexArrayVertexAttribOffsetEXT = PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexAttribOffsetEXT"));
 glVertexArrayVertexBindingDivisorEXT = PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC(glfwGetProcAddress("glVertexArrayVertexBindingDivisorEXT"));
 glVertexArrayVertexBuffer = PFNGLVERTEXARRAYVERTEXBUFFERPROC(glfwGetProcAddress("glVertexArrayVertexBuffer"));
 glVertexArrayVertexBuffers = PFNGLVERTEXARRAYVERTEXBUFFERSPROC(glfwGetProcAddress("glVertexArrayVertexBuffers"));
 glVertexArrayVertexOffsetEXT = PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC(glfwGetProcAddress("glVertexArrayVertexOffsetEXT"));
 glVertexAttrib1d = PFNGLVERTEXATTRIB1DPROC(glfwGetProcAddress("glVertexAttrib1d"));
 glVertexAttrib1dv = PFNGLVERTEXATTRIB1DVPROC(glfwGetProcAddress("glVertexAttrib1dv"));
 glVertexAttrib1f = PFNGLVERTEXATTRIB1FPROC(glfwGetProcAddress("glVertexAttrib1f"));
 glVertexAttrib1fv = PFNGLVERTEXATTRIB1FVPROC(glfwGetProcAddress("glVertexAttrib1fv"));
 glVertexAttrib1s = PFNGLVERTEXATTRIB1SPROC(glfwGetProcAddress("glVertexAttrib1s"));
 glVertexAttrib1sv = PFNGLVERTEXATTRIB1SVPROC(glfwGetProcAddress("glVertexAttrib1sv"));
 glVertexAttrib2d = PFNGLVERTEXATTRIB2DPROC(glfwGetProcAddress("glVertexAttrib2d"));
 glVertexAttrib2dv = PFNGLVERTEXATTRIB2DVPROC(glfwGetProcAddress("glVertexAttrib2dv"));
 glVertexAttrib2f = PFNGLVERTEXATTRIB2FPROC(glfwGetProcAddress("glVertexAttrib2f"));
 glVertexAttrib2fv = PFNGLVERTEXATTRIB2FVPROC(glfwGetProcAddress("glVertexAttrib2fv"));
 glVertexAttrib2s = PFNGLVERTEXATTRIB2SPROC(glfwGetProcAddress("glVertexAttrib2s"));
 glVertexAttrib2sv = PFNGLVERTEXATTRIB2SVPROC(glfwGetProcAddress("glVertexAttrib2sv"));
 glVertexAttrib3d = PFNGLVERTEXATTRIB3DPROC(glfwGetProcAddress("glVertexAttrib3d"));
 glVertexAttrib3dv = PFNGLVERTEXATTRIB3DVPROC(glfwGetProcAddress("glVertexAttrib3dv"));
 glVertexAttrib3f = PFNGLVERTEXATTRIB3FPROC(glfwGetProcAddress("glVertexAttrib3f"));
 glVertexAttrib3fv = PFNGLVERTEXATTRIB3FVPROC(glfwGetProcAddress("glVertexAttrib3fv"));
 glVertexAttrib3s = PFNGLVERTEXATTRIB3SPROC(glfwGetProcAddress("glVertexAttrib3s"));
 glVertexAttrib3sv = PFNGLVERTEXATTRIB3SVPROC(glfwGetProcAddress("glVertexAttrib3sv"));
 glVertexAttrib4Nbv = PFNGLVERTEXATTRIB4NBVPROC(glfwGetProcAddress("glVertexAttrib4Nbv"));
 glVertexAttrib4Niv = PFNGLVERTEXATTRIB4NIVPROC(glfwGetProcAddress("glVertexAttrib4Niv"));
 glVertexAttrib4Nsv = PFNGLVERTEXATTRIB4NSVPROC(glfwGetProcAddress("glVertexAttrib4Nsv"));
 glVertexAttrib4Nub = PFNGLVERTEXATTRIB4NUBPROC(glfwGetProcAddress("glVertexAttrib4Nub"));
 glVertexAttrib4Nubv = PFNGLVERTEXATTRIB4NUBVPROC(glfwGetProcAddress("glVertexAttrib4Nubv"));
 glVertexAttrib4Nuiv = PFNGLVERTEXATTRIB4NUIVPROC(glfwGetProcAddress("glVertexAttrib4Nuiv"));
 glVertexAttrib4Nusv = PFNGLVERTEXATTRIB4NUSVPROC(glfwGetProcAddress("glVertexAttrib4Nusv"));
 glVertexAttrib4bv = PFNGLVERTEXATTRIB4BVPROC(glfwGetProcAddress("glVertexAttrib4bv"));
 glVertexAttrib4d = PFNGLVERTEXATTRIB4DPROC(glfwGetProcAddress("glVertexAttrib4d"));
 glVertexAttrib4dv = PFNGLVERTEXATTRIB4DVPROC(glfwGetProcAddress("glVertexAttrib4dv"));
 glVertexAttrib4f = PFNGLVERTEXATTRIB4FPROC(glfwGetProcAddress("glVertexAttrib4f"));
 glVertexAttrib4fv = PFNGLVERTEXATTRIB4FVPROC(glfwGetProcAddress("glVertexAttrib4fv"));
 glVertexAttrib4iv = PFNGLVERTEXATTRIB4IVPROC(glfwGetProcAddress("glVertexAttrib4iv"));
 glVertexAttrib4s = PFNGLVERTEXATTRIB4SPROC(glfwGetProcAddress("glVertexAttrib4s"));
 glVertexAttrib4sv = PFNGLVERTEXATTRIB4SVPROC(glfwGetProcAddress("glVertexAttrib4sv"));
 glVertexAttrib4ubv = PFNGLVERTEXATTRIB4UBVPROC(glfwGetProcAddress("glVertexAttrib4ubv"));
 glVertexAttrib4uiv = PFNGLVERTEXATTRIB4UIVPROC(glfwGetProcAddress("glVertexAttrib4uiv"));
 glVertexAttrib4usv = PFNGLVERTEXATTRIB4USVPROC(glfwGetProcAddress("glVertexAttrib4usv"));
 glVertexAttribBinding = PFNGLVERTEXATTRIBBINDINGPROC(glfwGetProcAddress("glVertexAttribBinding"));
 glVertexAttribDivisor = PFNGLVERTEXATTRIBDIVISORPROC(glfwGetProcAddress("glVertexAttribDivisor"));
 glVertexAttribDivisorARB = PFNGLVERTEXATTRIBDIVISORARBPROC(glfwGetProcAddress("glVertexAttribDivisorARB"));
 glVertexAttribFormat = PFNGLVERTEXATTRIBFORMATPROC(glfwGetProcAddress("glVertexAttribFormat"));
 glVertexAttribFormatNV = PFNGLVERTEXATTRIBFORMATNVPROC(glfwGetProcAddress("glVertexAttribFormatNV"));
 glVertexAttribI1i = PFNGLVERTEXATTRIBI1IPROC(glfwGetProcAddress("glVertexAttribI1i"));
 glVertexAttribI1iv = PFNGLVERTEXATTRIBI1IVPROC(glfwGetProcAddress("glVertexAttribI1iv"));
```

```
glVertexAttribI1ui = PFNGLVERTEXATTRIBI1UIPROC(glfwGetProcAddress("glVertexAttribI1ui"));
glVertexAttribI1uiv = PFNGLVERTEXATTRIBI1UIVPROC(glfwGetProcAddress("glVertexAttribI1uiv"));
glVertexAttribI2i = PFNGLVERTEXATTRIBI2IPROC(glfwGetProcAddress("glVertexAttribI2i"));
glVertexAttribI2iv = PFNGLVERTEXATTRIBI2IVPROC(glfwGetProcAddress("glVertexAttribI2iv"));
glVertexAttribI2ui = PFNGLVERTEXATTRIBI2UIPROC(glfwGetProcAddress("glVertexAttribI2ui"));
glVertexAttribI2uiv = PFNGLVERTEXATTRIBI2UIVPROC(glfwGetProcAddress("glVertexAttribI2uiv"));
glVertexAttribI3i = PFNGLVERTEXATTRIBI3IPROC(glfwGetProcAddress("glVertexAttribI3i"));
glVertexAttribI3iv = PFNGLVERTEXATTRIBI3IVPROC(glfwGetProcAddress("glVertexAttribI3iv"));
glVertexAttribI3ui = PFNGLVERTEXATTRIBI3UIPROC(glfwGetProcAddress("glVertexAttribI3ui"));
glVertexAttribI3uiv = PFNGLVERTEXATTRIBI3UIVPROC(glfwGetProcAddress("glVertexAttribI3uiv"));
glVertexAttribI4bv = PFNGLVERTEXATTRIBI4BVPROC(glfwGetProcAddress("glVertexAttribI4bv"));
glVertexAttribI4i = PFNGLVERTEXATTRIBI4IPROC(glfwGetProcAddress("glVertexAttribI4i"));
glVertexAttribI4iv = PFNGLVERTEXATTRIBI4IVPROC(glfwGetProcAddress("glVertexAttribI4iv"));
glVertexAttribI4sv = PFNGLVERTEXATTRIBI4SVPROC(glfwGetProcAddress("glVertexAttribI4sv"));
glVertexAttribI4ubv = PFNGLVERTEXATTRIBI4UBVPROC(glfwGetProcAddress("glVertexAttribI4ubv"));
glVertexAttribI4ui = PFNGLVERTEXATTRIBI4UIPROC(glfwGetProcAddress("glVertexAttribI4ui"));
glVertexAttribI4uiv = PFNGLVERTEXATTRIBI4UIVPROC(glfwGetProcAddress("glVertexAttribI4uiv"));
glVertexAttribI4usv = PFNGLVERTEXATTRIBI4USVPROC(glfwGetProcAddress("glVertexAttribI4usv"));
glVertexAttribIFormat = PFNGLVERTEXATTRIBIFORMATPROC(glfwGetProcAddress("glVertexAttribIFormat"));
glVertexAttribIFormatNV = PFNGLVERTEXATTRIBIFORMATNVPROC(glfwGetProcAddress("glVertexAttribIFormatNV"));
glVertexAttribIPointer = PFNGLVERTEXATTRIBIPOINTERPROC(glfwGetProcAddress("glVertexAttribIPointer"));
glVertexAttribL1d = PFNGLVERTEXATTRIBL1DPROC(glfwGetProcAddress("glVertexAttribL1d"));
glVertexAttribL1dv = PFNGLVERTEXATTRIBL1DVPROC(glfwGetProcAddress("glVertexAttribL1dv"));
glVertexAttribL1i64NV = PFNGLVERTEXATTRIBL1I64NVPROC(glfwGetProcAddress("glVertexAttribL1i64NV"));
glVertexAttribL1i64vNV = PFNGLVERTEXATTRIBL1I64VNVPROC(glfwGetProcAddress("glVertexAttribL1i64vNV"));
glVertexAttribL1ui64ARB = PFNGLVERTEXATTRIBL1UI64ARBPROC(glfwGetProcAddress("glVertexAttribL1ui64ARB"));
glVertexAttribL1ui64NV = PFNGLVERTEXATTRIBL1UI64NVPROC(glfwGetProcAddress("glVertexAttribL1ui64NV"));
glVertexAttribL1ui64vARB = PFNGLVERTEXATTRIBL1UI64VARBPROC(glfwGetProcAddress("glVertexAttribL1ui64vARB"));
glVertexAttribL1ui64vNV = PFNGLVERTEXATTRIBL1UI64VNVPROC(glfwGetProcAddress("glVertexAttribL1ui64vNV"));
glVertexAttribL2d = PFNGLVERTEXATTRIBL2DPROC(glfwGetProcAddress("glVertexAttribL2d"));
glVertexAttribL2dv = PFNGLVERTEXATTRIBL2DVPROC(glfwGetProcAddress("glVertexAttribL2dv"));
glVertexAttribL2i64NV = PFNGLVERTEXATTRIBL2I64NVPROC(glfwGetProcAddress("glVertexAttribL2i64NV"));
glVertexAttribL2i64vNV = PFNGLVERTEXATTRIBL2I64VNVPROC(glfwGetProcAddress("glVertexAttribL2i64vNV"));
glVertexAttribL2ui64NV = PFNGLVERTEXATTRIBL2UI64NVPROC(glfwGetProcAddress("glVertexAttribL2ui64NV"));
glVertexAttribL2ui64vNV = PFNGLVERTEXATTRIBL2UI64VNVPROC(glfwGetProcAddress("glVertexAttribL2ui64vNV"));
glVertexAttribL3d = PFNGLVERTEXATTRIBL3DPROC(glfwGetProcAddress("glVertexAttribL3d"));
glVertexAttribL3dv = PFNGLVERTEXATTRIBL3DVPROC(glfwGetProcAddress("glVertexAttribL3dv"));
glVertexAttribL3i64NV = PFNGLVERTEXATTRIBL3I64NVPROC(glfwGetProcAddress("glVertexAttribL3i64NV"));
glVertexAttribL3i64vNV = PFNGLVERTEXATTRIBL3I64VNVPROC(glfwGetProcAddress("glVertexAttribL3i64vNV"));
glVertexAttribL3ui64NV = PFNGLVERTEXATTRIBL3UI64NVPROC(glfwGetProcAddress("glVertexAttribL3ui64NV"));
glVertexAttribL3ui64vNV = PFNGLVERTEXATTRIBL3UI64VNVPROC(glfwGetProcAddress("glVertexAttribL3ui64vNV"));
glVertexAttribL4d = PFNGLVERTEXATTRIBL4DPROC(glfwGetProcAddress("glVertexAttribL4d"));
glVertexAttribL4dv = PFNGLVERTEXATTRIBL4DVPROC(glfwGetProcAddress("glVertexAttribL4dv"));
glVertexAttribL4i64NV = PFNGLVERTEXATTRIBL4I64NVPROC(glfwGetProcAddress("glVertexAttribL4i64NV"));
glVertexAttribL4i64vNV = PFNGLVERTEXATTRIBL4I64VNVPROC(glfwGetProcAddress("glVertexAttribL4i64vNV"));
glVertexAttribL4ui64NV = PFNGLVERTEXATTRIBL4UI64NVPROC(glfwGetProcAddress("glVertexAttribL4ui64NV"));
glVertexAttribL4ui64vNV = PFNGLVERTEXATTRIBL4UI64VNVPROC(glfwGetProcAddress("glVertexAttribL4ui64vNV"));
glVertexAttribLFormat = PFNGLVERTEXATTRIBLFORMATPROC(glfwGetProcAddress("glVertexAttribLFormat"));
glVertexAttribLFormatNV = PFNGLVERTEXATTRIBLFORMATNVPROC(glfwGetProcAddress("glVertexAttribLFormatNV"));
glVertexAttribLPointer = PFNGLVERTEXATTRIBLPOINTERPROC(glfwGetProcAddress("glVertexAttribLPointer"));
glVertexAttribP1ui = PFNGLVERTEXATTRIBP1UIPROC(glfwGetProcAddress("glVertexAttribP1ui"));
glVertexAttribP1uiv = PFNGLVERTEXATTRIBP1UIVPROC(glfwGetProcAddress("glVertexAttribP1uiv"));
glVertexAttribP2ui = PFNGLVERTEXATTRIBP2UIPROC(glfwGetProcAddress("glVertexAttribP2ui"));
glVertexAttribP2uiv = PFNGLVERTEXATTRIBP2UIVPROC(glfwGetProcAddress("glVertexAttribP2uiv"));
glVertexAttribP3ui = PFNGLVERTEXATTRIBP3UIPROC(glfwGetProcAddress("glVertexAttribP3ui"));
glVertexAttribP3uiv = PFNGLVERTEXATTRIBP3UIVPROC(glfwGetProcAddress("glVertexAttribP3uiv"));
glVertexAttribP4ui = PFNGLVERTEXATTRIBP4UIPROC(glfwGetProcAddress("glVertexAttribP4ui"));
glVertexAttribP4uiv = PFNGLVERTEXATTRIBP4UIVPROC(glfwGetProcAddress("glVertexAttribP4uiv"));
glVertexAttribPointer = PFNGLVERTEXATTRIBPOINTERPROC(glfwGetProcAddress("glVertexAttribPointer"));
glVertexBindingDivisor = PFNGLVERTEXBINDINGDIVISORPROC(glfwGetProcAddress("glVertexBindingDivisor"));
glVertexFormatNV = PFNGLVERTEXFORMATNVPROC(glfwGetProcAddress("glVertexFormatNV"));
glViewport = PFNGLVIEWPORTPROC(glfwGetProcAddress("glViewport"));
glViewportArrayv = PFNGLVIEWPORTARRAYVPROC(glfwGetProcAddress("glViewportArrayv"));
glViewportIndexedf = PFNGLVIEWPORTINDEXEDFPROC(glfwGetProcAddress("glViewportIndexedf"));
glViewportIndexedfv = PFNGLVIEWPORTINDEXEDFVPROC(glfwGetProcAddress("glViewportIndexedfv"));
glViewportPositionWScaleNV = PFNGLVIEWPORTPOSITIONWSCALENVPROC(glfwGetProcAddress("glViewportPositionWScaleNV"));
glViewportSwizzleNV = PFNGLVIEWPORTSWIZZLENVPROC(glfwGetProcAddress("glViewportSwizzleNV"));
```

```
  glWaitSync = PFNGLWAITSYNCPROC(glfwGetProcAddress("glWaitSync"));
  glWaitVkSemaphoreNV = PFNGLWAITVKSEMAPHORENVPROC(glfwGetProcAddress("glWaitVkSemaphoreNV"));
  glWeightPathsNV = PFNGLWEIGHTPATHSNVPROC(glfwGetProcAddress("glWeightPathsNV"));
  glWindowRectanglesEXT = PFNGLWINDOWRECTANGLESEXTPROC(glfwGetProcAddress("glWindowRectanglesEXT"));
#endif

  // 使用している GPU のバッファアライメントを調べる
  glGetIntegerv(GL_UNIFORM_BUFFER_OFFSET_ALIGNMENT, &ggBufferAlignment);
}

//
// OpenGL のエラーをチェックする
//
//   OpenGL の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する
//
//   msg エラー発生時に標準エラー出力に出力する文字列. nullptr なら何も出力しない
//
void gg::_ggError(const std::string& name, unsigned int line)
{
  const GLenum error{ glGetError() };

  if (error != GL_NO_ERROR)
  {
    if (!name.empty())
    {
      std::cerr << name;
      if (line > 0) std::cerr << " (" << line << ")";
      std::cerr << ": ";
    }

    switch (error)
    {
    case GL_INVALID_ENUM:
      std::cerr << "An unacceptable value is specified for an enumerated argument" << std::endl;
      break;
    case GL_INVALID_VALUE:
      std::cerr << "A numeric argument is out of range" << std::endl;
      break;
    case GL_INVALID_OPERATION:
      std::cerr << "The specified operation is not allowed in the current state" << std::endl;
      break;
    case GL_OUT_OF_MEMORY:
      std::cerr << "There is not enough memory left to execute the command" << std::endl;
      break;
    case GL_INVALID_FRAMEBUFFER_OPERATION:
      std::cerr << "The specified operation is not allowed current frame buffer" << std::endl;
      break;
    default:
      std::cerr << "An OpenGL error has occured: " << std::hex << std::showbase << error << std::endl;
      break;
    }
  }
}

//
// FBO のエラーをチェックする
//
//   FBO の API を呼び出し直後に実行すればエラーのあるときにメッセージを表示する
//
//   msg エラー発生時に標準エラー出力に出力する文字列. nullptr なら何も出力しない
//
void gg::_ggFBOError(const std::string& name, unsigned int line)
{
  const GLenum status{ glCheckFramebufferStatus(GL_FRAMEBUFFER) };

  if (status != GL_FRAMEBUFFER_COMPLETE)
  {
    if (!name.empty())
    {
      std::cerr << name;
      if (line > 0) std::cerr << " (" << line << ")";
      std::cerr << ": ";
    }

    switch (status)
    {
    case GL_FRAMEBUFFER_UNDEFINED:
      std::cerr << "the default framebuffer does not exist" << std::endl;
      break;
    case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
      std::cerr << "Framebuffer incomplete, duplicate attachment" << std::endl;
      break;
    case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
      std::cerr << "Framebuffer incomplete, missing attachment" << std::endl;
      break;
    case GL_FRAMEBUFFER_UNSUPPORTED:
      std::cerr << "Unsupported framebuffer internal" << std::endl;
```

```
      break;
    case GL_FRAMEBUFFER_INCOMPLETE_MULTISAMPLE:
      std::cerr << "The value of GL_RENDERBUFFER_SAMPLES is not"
        " the same for all attached renderbuffers or,"
        " if the attached images are a mix of renderbuffers and textures,"
        " the value of GL_RENDERBUFFER_SAMPLES is not zero" << std::endl;
      break;
#if !defined(GL_GLES_PROTOTYPES)
    case GL_FRAMEBUFFER_INCOMPLETE_LAYER_TARGETS:
      std::cerr << "Any framebuffer attachment is layered,"
        " and any populated attachment is not layered,"
        " or if all populated color attachments are not from textures"
        " of the same target" << std::endl;
      break;
    case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER:
      std::cerr << "Framebuffer incomplete, missing draw buffer" << std::endl;
      break;
    case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
      std::cerr << "Framebuffer incomplete, missing read buffer" << std::endl;
      break;
#endif
    default:
      std::cerr << "Programming error; will fail on all hardware: " << std::hex << std::showbase << status << std::endl;
      break;
    }
  }
}

//
// 変換行列：行列とベクトルの積 c ← a × b
//
void gg::GgMatrix::projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const
{
  for (int i = 0; i < 4; ++i)
  {
    c[i] = a[0 + i] * b[0] + a[4 + i] * b[1] + a[8 + i] * b[2] + a[12 + i] * b[3];
  }
}

//
// 変換行列：行列と行列の積 c ← a × b
//
void gg::GgMatrix::multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const
{
  for (int i = 0; i < 16; ++i)
  {
    int j = i & 3, k = i & ~3;

    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];
  }
}

//
// 変換行列：単位行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadIdentity()
{
  data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
  data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
  data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
  data()[ 0] = data()[ 5] = data()[10] = data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：平行移動変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
{
  data()[12] = x;
  data()[13] = y;
  data()[14] = z;
  data()[ 0] = data()[ 5] = data()[10] = data()[15] = w;
  data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
  data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
  data()[11] = 0.0f;

  return *this;
}

//
// 変換行列：拡大縮小変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
{
```

```
  data()[ 0] = x;
  data()[ 5] = y;
  data()[10] = z;
  data()[15] = w;
  data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
  data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
  data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;

  return *this;
}

//
// 変換行列：x 軸中心の回転変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadRotateX(GLfloat a)
{
  const GLfloat c{ cosf(a) };
  const GLfloat s{ sinf(a) };

  data()[ 0] = 1.0f; data()[ 1] = 0.0f; data()[ 2] = 0.0f; data()[ 3] = 0.0f;
  data()[ 4] = 0.0f; data()[ 5] = c;    data()[ 6] = s;    data()[ 7] = 0.0f;
  data()[ 8] = 0.0f; data()[ 9] = -s;   data()[10] = c;    data()[11] = 0.0f;
  data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：y 軸中心の回転変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadRotateY(GLfloat a)
{
  const GLfloat c{ cosf(a) };
  const GLfloat s{ sinf(a) };

  data()[ 0] = c;    data()[ 1] = 0.0f; data()[ 2] = -s;   data()[ 3] = 0.0f;
  data()[ 4] = 0.0f; data()[ 5] = 1.0f; data()[ 6] = 0.0f; data()[ 7] = 0.0f;
  data()[ 8] = s;    data()[ 9] = 0.0f; data()[10] = c;    data()[11] = 0.0f;
  data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：z 軸中心の回転変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadRotateZ(GLfloat a)
{
  const GLfloat c{ cosf(a) };
  const GLfloat s{ sinf(a) };

  data()[ 0] = c;    data()[ 1] = s;    data()[ 2] = 0.0f; data()[ 3] = 0.0f;
  data()[ 4] = -s;   data()[ 5] = c;    data()[ 6] = 0.0f; data()[ 7] = 0.0f;
  data()[ 8] = 0.0f; data()[ 9] = 0.0f; data()[10] = 1.0f; data()[11] = 0.0f;
  data()[12] = 0.0f; data()[13] = 0.0f; data()[14] = 0.0f; data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：任意軸中心の回転変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
{
  const GLfloat d{ sqrtf(x * x + y * y + z * z) };

  if (d > 0.0f)
  {
    const GLfloat l{ x / d }, m{ y / d }, n{ z / d };
    const GLfloat l2{ l * l }, m2{ m * m }, n2{ n * n };
    const GLfloat lm{ l * m }, mn{ m * n }, nl{ n * l };
    const GLfloat c{ cosf(a) }, c1{ 1.0f - c };
    const GLfloat s{ sinf(a) };

    data()[ 0] = (1.0f - l2) * c + l2;
    data()[ 1] = lm * c1 + n * s;
    data()[ 2] = nl * c1 - m * s;
    data()[ 3] = 0.0f;

    data()[ 4] = lm * c1 - n * s;
    data()[ 5] = (1.0f - m2) * c + m2;
    data()[ 6] = mn * c1 + l * s;
    data()[ 7] = 0.0f;

    data()[ 8] = nl * c1 + m * s;
    data()[ 9] = mn * c1 - l * s;
    data()[10] = (1.0f - n2) * c + n2;
```

```
    data()[11] = 0.0f;

    data()[12] = 0.0f;
    data()[13] = 0.0f;
    data()[14] = 0.0f;
    data()[15] = 1.0f;
  }

  return *this;
}

//
// 変換行列：転置行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadTranspose(const GLfloat* marray)
{
  data()[ 0] = marray[ 0];
  data()[ 1] = marray[ 4];
  data()[ 2] = marray[ 8];
  data()[ 3] = marray[12];
  data()[ 4] = marray[ 1];
  data()[ 5] = marray[ 5];
  data()[ 6] = marray[ 9];
  data()[ 7] = marray[13];
  data()[ 8] = marray[ 2];
  data()[ 9] = marray[ 6];
  data()[10] = marray[10];
  data()[11] = marray[14];
  data()[12] = marray[ 3];
  data()[13] = marray[ 7];
  data()[14] = marray[11];
  data()[15] = marray[15];

  return *this;
}

//
// 変換行列：逆行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadInvert(const GLfloat* marray)
{
  GLfloat lu[20], * plu[4];

  // j 行の要素の値の絶対値の最大値を plu[j][4] に求める
  for (int j = 0; j < 4; ++j)
  {
    GLfloat max{ fabsf(*(plu[j] = lu + 5 * j) = *(marray++)) };

    for (int i = 0; ++i < 4;)
    {
      GLfloat a{ fabsf(plu[j][i] = *(marray++)) };
      if (a > max) max = a;
    }
    if (max == 0.0f) return *this;
    plu[j][4] = 1.0f / max;
  }

  // ピボットを考慮した LU 分解
  for (int j = 0; j < 4; ++j)
  {
    GLfloat max{ fabsf(plu[j][j] * plu[j][4]) };
    int i = j;

    for (int k = j; ++k < 4;)
    {
      GLfloat a{ fabsf(plu[k][j] * plu[k][4]) };
      if (a > max)
      {
        max = a;
        i = k;
      }
    }
    if (i > j)
    {
      GLfloat* t{ plu[j] };
      plu[j] = plu[i];
      plu[i] = t;
    }
    if (plu[j][j] == 0.0f) return *this;
    for (int k = j; ++k < 4;)
    {
      plu[k][j] /= plu[j][j];
      for (int i = j; ++i < 4;)
      {
        plu[k][i] -= plu[j][i] * plu[k][j];
      }
    }
```

```
  }

  // LU 分解から逆行列を求める
  for (int k = 0; k < 4; ++k)
  {
    // array に単位行列を設定する
    for (int i = 0; i < 4; ++i)
    {
      data()[i * 4 + k] = (plu[i] == lu + k * 5) ? 1.0f : 0.0f;
    }
    // lu から逆行列を求める
    for (int i = 0; i < 4; ++i)
    {
      for (int j = i; ++j < 4;)
      {
        data()[j * 4 + k] -= data()[i * 4 + k] * plu[j][i];
      }
    }
    for (int i = 4; --i >= 0;)
    {
      for (int j = i; ++j < 4;)
      {
        data()[i * 4 + k] -= plu[i][j] * data()[j * 4 + k];
      }
      data()[i * 4 + k] /= plu[i][i];
    }
  }

  return *this;
}

//
// 変換行列：法線変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadNormal(const GLfloat* marray)
{
  data()[ 0] = marray[ 5] * marray[10] - marray[ 6] * marray[ 9];
  data()[ 1] = marray[ 6] * marray[ 8] - marray[ 4] * marray[10];
  data()[ 2] = marray[ 4] * marray[ 9] - marray[ 5] * marray[ 8];
  data()[ 4] = marray[ 9] * marray[ 2] - marray[10] * marray[ 1];
  data()[ 5] = marray[10] * marray[ 0] - marray[ 8] * marray[ 2];
  data()[ 6] = marray[ 8] * marray[ 1] - marray[ 9] * marray[ 0];
  data()[ 8] = marray[ 1] * marray[ 6] - marray[ 2] * marray[ 5];
  data()[ 9] = marray[ 2] * marray[ 4] - marray[ 0] * marray[ 6];
  data()[10] = marray[ 0] * marray[ 5] - marray[ 1] * marray[ 4];
  data()[ 3] = data()[ 7] = data()[11] = data()[12] = data()[13] = data()[14] = 0.0f;
  data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：ビュー変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadLookat(
  GLfloat ex, GLfloat ey, GLfloat ez,
  GLfloat tx, GLfloat ty, GLfloat tz,
  GLfloat ux, GLfloat uy, GLfloat uz
)
{
  // z 軸 = e - t
  const GLfloat zx{ ex - tx };
  const GLfloat zy{ ey - ty };
  const GLfloat zz{ ez - tz };

  // x 軸 = u x z 軸
  const GLfloat xx{ uy * zz - uz * zy };
  const GLfloat xy{ uz * zx - ux * zz };
  const GLfloat xz{ ux * zy - uy * zx };

  // y 軸 = z 軸 x x 軸
  const GLfloat yx{ zy * xz - zz * xy };
  const GLfloat yy{ zz * xx - zx * xz };
  const GLfloat yz{ zx * xy - zy * xx };

  // y 軸の長さをチェック
  GLfloat y{ yx * yx + yy * yy + yz * yz };
  if (y == 0.0f) return *this;

  // x 軸の正規化
  const GLfloat x{ sqrtf(xx * xx + xy * xy + xz * xz) };
  data()[ 0] = xx / x;
  data()[ 4] = xy / x;
  data()[ 8] = xz / x;

  // y 軸の正規化
  y = sqrtf(y);
```

```
  data()[ 1] = yx / y;
  data()[ 5] = yy / y;
  data()[ 9] = yz / y;

  // z 軸の正規化
  const GLfloat z{ sqrtf(zx * zx + zy * zy + zz * zz) };
  data()[ 2] = zx / z;
  data()[ 6] = zy / z;
  data()[10] = zz / z;

  // 平行移動
  data()[12] = -(ex * data()[ 0] + ey * data()[ 4] + ez * data()[ 8]);
  data()[13] = -(ex * data()[ 1] + ey * data()[ 5] + ez * data()[ 9]);
  data()[14] = -(ex * data()[ 2] + ey * data()[ 6] + ez * data()[10]);

  // 残り
  data()[ 3] = data()[ 7] = data()[11] = 0.0f;
  data()[15] = 1.0f;

  return *this;
}

//
// 変換行列：平行投影変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadOrthogonal(
  GLfloat left, GLfloat right,
  GLfloat bottom, GLfloat top,
  GLfloat zNear, GLfloat zFar
)
{
  const GLfloat dx{ right - left };
  const GLfloat dy{ top - bottom };
  const GLfloat dz{ zFar - zNear };

  if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
  {
    data()[ 0] = 2.0f / dx;
    data()[ 5] = 2.0f / dy;
    data()[10] = -2.0f / dz;
    data()[12] = -(right + left) / dx;
    data()[13] = -(top + bottom) / dy;
    data()[14] = -(zFar + zNear) / dz;
    data()[15] = 1.0f;
    data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
    data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
    data()[11] = 0.0f;
  }

  return *this;
}

//
// 変換行列：透視投影変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadFrustum(
  GLfloat left, GLfloat right,
  GLfloat bottom, GLfloat top,
  GLfloat zNear, GLfloat zFar
)
{
  const GLfloat dx{ right - left };
  const GLfloat dy{ top - bottom };
  const GLfloat dz{ zFar - zNear };

  if (dx != 0.0f && dy != 0.0f && dz != 0.0f)
  {
    data()[ 0] = 2.0f * zNear / dx;
    data()[ 5] = 2.0f * zNear / dy;
    data()[ 8] = (right + left) / dx;
    data()[ 9] = (top + bottom) / dy;
    data()[10] = -(zFar + zNear) / dz;
    data()[11] = -1.0f;
    data()[14] = -2.0f * zFar * zNear / dz;
    data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
    data()[ 6] = data()[ 7] = data()[12] = data()[13] =
    data()[15] = 0.0f;
  }

  return *this;
}

//
// 変換行列：画角から透視投影変換行列を設定する
//
gg::GgMatrix& gg::GgMatrix::loadPerspective(
  GLfloat fovy, GLfloat aspect,
```

```
  GLfloat zNear, GLfloat zFar
)
{
  const GLfloat dz{ zFar - zNear };

  if (dz != 0.0f)
  {
    data()[ 5] = 1.0f / tanf(fovy * 0.5f);
    data()[ 0] = data()[ 5] / aspect;
    data()[10] = -(zFar + zNear) / dz;
    data()[11] = -1.0f;
    data()[14] = -2.0f * zFar * zNear / dz;
    data()[ 1] = data()[ 2] = data()[ 3] = data()[ 4] =
    data()[ 6] = data()[ 7] = data()[ 8] = data()[ 9] =
    data()[12] = data()[13] = data()[15] = 0.0f;
  }

  return *this;
}

//
// 四元数：GgQuaternion 型の四元数 p, q の積を r に求める
//
void gg::GgQuaternion::multiply(GLfloat* r, const GLfloat* p, const GLfloat* q) const
{
  r[0] = p[1] * q[2] - p[2] * q[1] + p[0] * q[3] + p[3] * q[0];
  r[1] = p[2] * q[0] - p[0] * q[2] + p[1] * q[3] + p[3] * q[1];
  r[2] = p[0] * q[1] - p[1] * q[0] + p[2] * q[3] + p[3] * q[2];
  r[3] = p[3] * q[3] - p[0] * q[0] - p[1] * q[1] - p[2] * q[2];
}

//
// 四元数：GgQuaternion 型の四元数 q が表す変換行列を m に求める
//
void gg::GgQuaternion::toMatrix(GLfloat* m, const GLfloat* q) const
{
  const GLfloat xx{ q[0] * q[0] * 2.0f };
  const GLfloat yy{ q[1] * q[1] * 2.0f };
  const GLfloat zz{ q[2] * q[2] * 2.0f };
  const GLfloat xy{ q[0] * q[1] * 2.0f };
  const GLfloat yz{ q[1] * q[2] * 2.0f };
  const GLfloat zx{ q[2] * q[0] * 2.0f };
  const GLfloat xw{ q[0] * q[3] * 2.0f };
  const GLfloat yw{ q[1] * q[3] * 2.0f };
  const GLfloat zw{ q[2] * q[3] * 2.0f };

  m[ 0] = 1.0f - yy - zz;
  m[ 1] = xy + zw;
  m[ 2] = zx - yw;
  m[ 4] = xy - zw;
  m[ 5] = 1.0f - zz - xx;
  m[ 6] = yz + xw;
  m[ 8] = zx + yw;
  m[ 9] = yz - xw;
  m[10] = 1.0f - xx - yy;
  m[ 3] = m[ 7] = m[11] = m[12] = m[13] = m[14] = 0.0f;
  m[15] = 1.0f;
}

//
// 四元数：回転変換行列 a が表す四元数を q に求める
//
void gg::GgQuaternion::toQuaternion(GLfloat* q, const GLfloat* a) const
{
  const GLfloat tr{ a[0] + a[5] + a[10] + a[15] };

  if (tr > 0.0f)
  {
    q[3] = sqrtf(tr) * 0.5f;
    q[0] = (a[6] - a[9]) * 0.25f / q[3];
    q[1] = (a[8] - a[2]) * 0.25f / q[3];
    q[2] = (a[1] - a[4]) * 0.25f / q[3];
  }
}

//
// 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
//
void gg::GgQuaternion::slerp(GLfloat* p, const GLfloat* q, const GLfloat* r, GLfloat t) const
{
  const GLfloat qr{ ggDot3(q, r) };
  const GLfloat ss{ 1.0f - qr * qr };

  if (ss == 0.0f)
  {
    if (p != q)
    {
```

```
      p[0] = q[0];
      p[1] = q[1];
      p[2] = q[2];
      p[3] = q[3];
    }
  }
  else
  {
    const GLfloat sp{ sqrtf(ss) };
    const GLfloat ph{ acosf(qr) };
    const GLfloat pt{ ph * t };
    const GLfloat t1{ sinf(pt) / sp };
    const GLfloat t0{ sinf(ph - pt) / sp };

    p[0] = q[0] * t0 + r[0] * t1;
    p[1] = q[1] * t0 + r[1] * t1;
    p[2] = q[2] * t0 + r[2] * t1;
    p[3] = q[3] * t0 + r[3] * t1;
  }
}

//
// 四元数：(x, y, z) を軸とし角度 a 回転する四元数を求める
//
gg::GgQuaternion& gg::GgQuaternion::loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
{
  const GLfloat l(x * x + y * y + z * z);

  if (l != 0.0)
  {
    GLfloat s{ sinf(a *= 0.5f) / sqrtf(l) };

    data()[0] = x * s;
    data()[1] = y * s;
    data()[2] = z * s;
  }
  else
  {
    data()[0] = data()[1] = data()[2] = 0.0f;
  }
  data()[3] = cosf(a);

  return *this;
}

//
// x 軸中心に角度 a 回転する四元数を格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadRotateX(GLfloat a)
{
  const GLfloat t{ a * 0.5f };

  data()[0] = sinf(t);
  data()[3] = cosf(t);
  data()[1] = data()[2] = 0.0f;

  return *this;
}

//
// y 軸中心に角度 a 回転する四元数を格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadRotateY(GLfloat a)
{
  const GLfloat t{ a * 0.5f };

  data()[1] = sinf(t);
  data()[3] = cosf(t);
  data()[0] = data()[2] = 0.0f;

  return *this;
}

//
// z 軸中心に角度 a 回転する四元数を格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadRotateZ(GLfloat a)
{
  const GLfloat t{ a * 0.5f };

  data()[2] = sinf(t);
  data()[3] = cosf(t);
  data()[0] = data()[1] = 0.0f;

  return *this;
}
```

```
//
// 四元数：オイラー角 (heading, pitch, roll) にもとづいて四元数を求める
//
gg::GgQuaternion& gg::GgQuaternion::loadEuler(GLfloat heading, GLfloat pitch, GLfloat roll)
{
  GgQuaternion h, p, r;

  h.loadRotateY(heading);
  p.loadRotateX(pitch);
  r.loadRotateZ(roll);

  *this = h * p * r;

  return *this;
}

//
// 四元数：正規化して格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadNormalize(const GLfloat* a)
{
  data()[0] = a[0];
  data()[1] = a[1];
  data()[2] = a[2];
  data()[3] = a[3];

  ggNormalize4(data());

  return *this;
}

//
// 四元数：共役四元数を格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadConjugate(const GLfloat* a)
{
  // w 要素を反転する
  data()[0] = a[0];
  data()[1] = a[1];
  data()[2] = a[2];
  data()[3] = -a[3];

  return *this;
}

//
// 四元数：逆元を格納する
//
gg::GgQuaternion& gg::GgQuaternion::loadInvert(const GLfloat* a)
{
  // ノルムの二乗を求める
  const GLfloat l(ggDot4(a, a));

  if (l > 0.0f)
  {
    // 共役四元数を求める
    GgQuaternion r;
    r.loadConjugate(a);

    // ノルムの二乗で割る
    data()[0] = r[0] / l;
    data()[1] = r[1] / l;
    data()[2] = r[2] / l;
    data()[3] = r[3] / l;
  }

  return *this;
}

//
// 簡易トラックボール処理：リセット
//
void gg::GgTrackball::reset(const GgQuaternion& q)
{
  // ドラッグ中ではない
  drag = false;

  // 単位クォーターニオンに初期値を与える
  this->load(cq = q);

  // 回転行列を初期化する
  static_cast<GgQuaternion*>(this)->getMatrix(rt);
}

//
// 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
//
```

```
//   Reshape コールバック (resize) の中で実行する
//   (w, h) ウィンドウサイズ
//
void gg::GgTrackball::region(GLfloat w, GLfloat h)
{
  // マウスポインタ位置のウィンドウ内の相対的位置への換算用
  scale[0] = 2.0f / w;
  scale[1] = 2.0f / h;
}

//
// 簡易トラックボール処理：ドラッグ開始時の処理
//
//   マウスボタンを押したときに実行する
//   (x, y) 現在のマウス位置
//
void gg::GgTrackball::begin(GLfloat x, GLfloat y)
{
  // ドラッグ開始
  drag = true;

  // ドラッグ開始点を記録する
  start[0] = x;
  start[1] = y;
}

//
// 簡易トラックボール処理：ドラッグ中の処理
//
//   マウスのドラッグ中に実行する
//   (x, y) 現在のマウス位置
//
void gg::GgTrackball::motion(GLfloat x, GLfloat y)
{
  // ドラッグ中でなければ何もしない
  if (!drag) return;

  // マウスポインタの位置のドラッグ開始位置からの変位
  const GLfloat d[]{ (x - start[0]) * scale[0], (y - start[1]) * scale[1] };

  // マウスポインタの位置のドラッグ開始位置からの距離
  const GLfloat a{ hypotf(d[0], d[1]) };

  // マウスポインタの位置がドラッグ開始位置から移動していれば
  if (a > 0.0)
  {
    // 現在の回転の四元数に作った四元数を掛けて合成する
    this->load(ggRotateQuaternion(d[1], d[0], 0.0f, a * 3.1415926536f) * cq);

    // 合成した四元数から回転の変換行列を求める
    static_cast<GgQuaternion*>(this)->getMatrix(rt);
  }
}

//
// 簡易トラックボール処理：回転角の修正
//
//   現在の回転角を修正する
//   q 修正分の回転角を表す四元数
//
void gg::GgTrackball::rotate(const GgQuaternion& q)
{
  // ドラッグ中なら何もしない
  if (drag) return;

  // 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
  this->load(q * cq);

  // 合成した四元数から回転の変換行列を求める
  static_cast<GgQuaternion*>(this)->getMatrix(rt);

  // 誤差を吸収するために正規化して保存する
  cq = normalize();
}

//
// 簡易トラックボール処理：停止時の処理
//
//   マウスボタンを離したときに実行する
//   (x, y) 現在のマウス位置
//
void gg::GgTrackball::end(GLfloat x, GLfloat y)
{
  // ドラッグ終了点における回転を求める
  motion(x, y);

  // 誤差を吸収するために正規化して保存する
```

```
  cq = normalize();

  // ドラッグ終了
  drag = false;
}

//
// 配列に格納された画像の内容を TGA ファイルに保存する
//
//   name ファイル名
//   buffer 画像データ
//   width 画像の横の画素数
//   height 画像の縦の画素数
//   depth 画像の 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
//
bool gg::ggSaveTga(
  const std::string& name,
  const void* buffer,
  unsigned int width,
  unsigned int height,
  unsigned int depth
)
{
  // ファイルを開く
  std::ofstream file{ name, std::ios::binary };

  // ファイルが開けなかったら戻る
  if (file.fail()) return false;

  // 画像のヘッダ
  const unsigned char type{ static_cast<unsigned char>(depth == 0 ? 0 : depth < 3 ? 3 : 2) };
  const unsigned char alpha{ static_cast<unsigned char>(depth == 2 || depth == 4 ? 8 : 0) };
  const unsigned char header[18]
  {
    0,          // ID length
    0,          // Color map type (none)
    type,       // Image Type (2:RGB, 3:Grayscale)
    0, 0,       // Offset into the color map table
    0, 0,       // Number of color map entries
    0,          // Number of a color map entry bits per pixel
    0, 0,       // Horizontal image position
    0, 0,       // Vertical image position
    static_cast<unsigned char>(width & 0xff),
    static_cast<unsigned char>(width >> 8),
    static_cast<unsigned char>(height & 0xff),
    static_cast<unsigned char>(height >> 8),
    static_cast<unsigned char>(depth * 8),
    alpha       // Image descriptor
  };

  // ヘッダを書き込む
  file.write(reinterpret_cast<const char*>(header), sizeof header);

  // ヘッダの書き込みに失敗したら戻る
  if (file.bad())
  {
    file.close();
    return false;
  }

  // データを書き込む
  const unsigned int size{ width * height * depth };
  if (type == 2)
  {
    // フルカラー
    std::vector<char> temp(size);
    for (GLuint i = 0; i < size; i += depth)
    {
      temp[i + 2] = static_cast<const char*>(buffer)[i + 0];
      temp[i + 1] = static_cast<const char*>(buffer)[i + 1];
      temp[i + 0] = static_cast<const char*>(buffer)[i + 2];
      if (depth == 4) temp[i + 3] = static_cast<const char*>(buffer)[i + 3];
    }
    file.write(temp.data(), size);
  }
  else if (type == 3)
  {
    // グレースケール
    file.write(static_cast<const char*>(buffer), size);
  }

  // フッタを書き込む
  constexpr char footer[] = "\0\0\0\0\0\0\0\0TRUEVISION-XFILE.";
  file.write(footer, sizeof footer);

  // データの書き込みに失敗したら戻る
```

```
  if (file.bad())
  {
    file.close();
    return false;
  }

  // ファイルを閉じる
  file.close();

  // データの書き込みに成功した
  return true;
}

//
// カラーバッファの内容を TGA ファイルに保存する
//
//   name 保存するファイル名
//   戻り値 保存に成功すれば true, 失敗すれば false
//
bool gg::ggSaveColor(const std::string& name)
{
  // 現在のビューポートのサイズを得る
  GLint viewport[4];
  glGetIntegerv(GL_VIEWPORT, viewport);

  // ビューポートのサイズ分のメモリを確保する
  std::vector<GLubyte> buffer(viewport[2] * viewport[3] * 3);

  // 画面表示の完了を待つ
  glFinish();

  // カラーバッファを読み込む
  glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
    GL_RGB, GL_UNSIGNED_BYTE, buffer.data());

  // 読み込んだデータをファイルに書き込む
  return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 3);
}

//
// デプスバッファの内容を TGA ファイルに保存する
//
//   name 保存するファイル名
//   戻り値 保存に成功すれば true, 失敗すれば false
//
bool gg::ggSaveDepth(const std::string& name)
{
  // 現在のビューポートのサイズを得る
  GLint viewport[4];
  glGetIntegerv(GL_VIEWPORT, viewport);

  // ビューポートのサイズ分のメモリを確保する
  std::vector<GLubyte> buffer(viewport[2] * viewport[3]);

  // 画面表示の完了を待つ
  glFinish();

  // デプスバッファを読み込む
  glReadPixels(viewport[0], viewport[1], viewport[2], viewport[3],
    GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, buffer.data());

  // 読み込んだデータをファイルに書き込む
  return ggSaveTga(name, buffer.data(), viewport[2], viewport[3], 1);
}

//
// TGA ファイル (8/16/24/32bit) を読み込む
//
//   name 読み込むファイル名
//   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)
//   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)
//   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)
//   image 読み込んだ画像を格納する vector
//   戻り値 読み込みに成功すれば true, 失敗すれば false
//
bool gg::ggReadImage(
  const std::string& name,
  std::vector<GLubyte>& image,
  GLsizei* pWidth,
  GLsizei* pHeight,
  GLenum* pFormat
)
{
  // ファイルを開く
  std::ifstream file{ name, std::ios::binary };

  // ファイルが開けなかったら戻る
```

```
if (file.fail()) return false;

// ヘッダを読み込む
unsigned char header[18];
file.read(reinterpret_cast<char*>(header), sizeof header);

// ヘッダの読み込みに失敗したら戻る
if (file.bad())
{
  file.close();
  return false;
}

// 深度
const int depth{ header[16] / 8 };
switch (depth)
{
case 1:
  *pFormat = GL_RED;
  break;
case 2:
  *pFormat = GL_RG;
  break;
case 3:
  *pFormat = GL_RGB;
  break;
case 4:
  *pFormat = GL_RGBA;
  break;
default:
  // 取り扱えないフォーマットだったら戻る
  file.close();
  return false;
}

// 画像の縦横の画素数
*pWidth = header[13] << 8 | header[12];
*pHeight = header[15] << 8 | header[14];

// データサイズ
const int size{ *pWidth * *pHeight * depth };
if (size < 2) return false;

// 読み込みに使うメモリを確保する
image.resize(size);

// データを読み込む
if (header[2] & 8)
{
  // RLE
  int p{ 0 };
  char c;
  while (file.get(c))
  {
    if (c & 0x80)
    {
      // run-length packet
      const int count{ (c & 0x7f) + 1 };
      if (p + depth * count > size) break;
      char temp[4];
      file.read(temp, depth);
      for (int i = 0; i < count; ++i)
      {
        for (int j = 0; j < depth;) image[p++] = temp[j++];
      }
    }
    else
    {
      // raw packet
      const int count{ (c + 1) * depth };
      if (p + count > size) break;
      file.read(reinterpret_cast<char*>(image.data() + p), count);
      p += count;
    }
  }
}
else
{
  // 非圧縮
  file.read(reinterpret_cast<char*>(image.data()), size);
}

// 読み込みに失敗したら戻る
if (file.bad())
{
  file.close();
  return false;
```

```
  }

  // ファイルを閉じる
  file.close();

  // ファイルの読み込みに成功した
  return true;
}

//
// テクスチャを作成して画像を読み込む
//
//   image 画像データ, nullptr ならメモリの確保だけを行う
//   width 画像の横の画素数
//   height 画像の縦の画素数
//   format 画像データのフォーマット
//   type 画像のデータ型
//   internal テクスチャの内部フォーマット
//   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
//   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
//   戻り値 テクスチャ名
//
GLuint gg::ggLoadTexture(
  const GLvoid* image,
  GLsizei width,
  GLsizei height,
  GLenum format,
  GLenum type,
  GLenum internal,
  GLenum wrap,
  bool swizzle
)
{
  // テクスチャオブジェクト
  const GLuint tex{ [] { GLuint tex; glGenTextures(1, &tex); return tex; } () };
  glBindTexture(GL_TEXTURE_2D, tex);

  // 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
  glPixelStorei(GL_UNPACK_ALIGNMENT, format == GL_RGBA ? 4 : 1);

  // テクスチャを割り当てる
  glTexImage2D(GL_TEXTURE_2D, 0, internal, width, height, 0, format, type, image);

  // バイリニア（ミップマップなし），エッジでクランプ
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap);

  if (swizzle)
  {
    // テクスチャのサンプリング時に赤とと青を入れ替える
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_R, GL_BLUE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_SWIZZLE_B, GL_RED);
  }

  // テクスチャ名を返す
  return tex;
}

//
// 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
//
//   name TGA ファイル名
//   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)
//   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)
//   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.
//   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
//   戻り値 テクスチャ名
//
GLuint gg::ggLoadImage(
  const std::string& name,
  GLsizei* pWidth,
  GLsizei* pHeight,
  GLenum internal,
  GLenum wrap
)
{
  // 画像データ
  std::vector<GLubyte> image;

  // 画像サイズ
  GLsizei width, height;

  // 画像フォーマット
  GLenum format;

```

```
  // 画像を読み込む
  ggReadImage(name, image, &width, &height, &format);

  // 画像が読み込めなかったら戻る
  if (image.empty()) return 0;

  // 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
  if (internal == 0) internal = format;

  // テクスチャに読み込む
  const GLuint tex(ggLoadTexture(image.data(), width, height,
    format, GL_UNSIGNED_BYTE, internal, wrap, true));

  // 画像サイズを返す
  if (pWidth) *pWidth = width;
  if (pHeight) *pHeight = height;

  // テクスチャ名を返す
  return tex;
}

//
// 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
//
//   width 高さマップのグレースケール画像 hmap の横の画素数
//   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
//   stride データの間隔
//   hmap グレースケール画像のデータ
//   nz 法線の z 成分の割合
//   internal テクスチャの内部フォーマット
//   nmap 法線マップを格納する vector
//
void gg::ggCreateNormalMap(
  const GLubyte* hmap,
  GLsizei width,
  GLsizei height,
  GLenum format,
  GLfloat nz,
  GLenum internal,
  std::vector<GgVector>& nmap
)
{
  // メモリサイズ
  const GLsizei size{ width * height };

  // 法線マップのメモリを確保する
  nmap.resize(size);

  // 画素のバイト数
  GLint stride;
  switch (format)
  {
  case GL_RED:
    stride = 1;
    break;
  case GL_RG:
    stride = 2;
    break;
  case GL_RGB:
    stride = 3;
    break;
  case GL_RGBA:
    stride = 4;
    break;
  default:
    stride = 1;
    break;
  }

  // 法線マップの作成
  for (GLsizei i = 0; i < size; ++i)
  {
    const int x{ i % width };
    const int y{ i - x };
    const int u0{ (y + (x - 1 + width) % width) * stride };
    const int u1{ (y + (x + 1) % width) * stride };
    const int v0{ ((y - width + size) % size + x) * stride };
    const int v1{ ((y + width) % size + x) * stride };

    // 隣接する画素との値の差を法線の成分に用いる
    nmap[i][0] = static_cast<GLfloat>(hmap[u1] - hmap[u0]);
    nmap[i][1] = static_cast<GLfloat>(hmap[v1] - hmap[v0]);
    nmap[i][2] = nz;
    nmap[i][3] = hmap[i * stride];

    // 法線ベクトルを正規化する
    ggNormalize3(nmap[i]);
```

```
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ったら戻る
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
```

```
//
//   name 読み込むファイル名
//   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
//   戻り値 テクスチャの作成に成功すれば true, 失敗すれば false
//
void gg::GgColorTexture::load(
  const std::string& name,
  GLenum internal,
  GLenum wrap
)
{
  // 画像データ
  std::vector<GLubyte> image;

  // 画像サイズ
  GLsizei width, height;

  // 画像フォーマット
  GLenum format;

  // 画像を読み込む
  ggReadImage(name, image, &width, &height, &format);

  // 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
  if (internal == 0) internal = format;

  // テクスチャを作成する
  texture = std::make_shared<GgTexture>(image.data(), width, height,
    format, GL_UNSIGNED_BYTE, internal, wrap, true);
}

//
// 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
//
//   name 画像ファイル名
//   width テクスチャとして用いる画像データの横幅
//   height テクスチャとして用いる画像データの高さ
//   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)
//   nz 法線マップの z 成分の値
//   internal テクスチャの内部フォーマット
//
void gg::GgNormalTexture::load(
  const std::string& name,
  GLfloat nz,
  GLenum internal
)
{
  // 高さマップ
  std::vector<GLubyte> hmap;

  // 画像サイズ
  GLsizei width, height;

  // 画像フォーマット
  GLenum format;

  // 高さマップの画像を読み込む
  ggReadImage(name, hmap, &width, &height, &format);

  // 法線マップ
  std::vector<GgVector> nmap;

  // 法線マップを作成する
  ggCreateNormalMap(hmap.data(), width, height, format, nz, internal, nmap);
}


//
// OBJ ファイルの読み込みに使うデータ型と関数
//
namespace gg
{
  // GLfloat 型の 2 要素のベクトル
  using  vec2 = std::array<GLfloat, 2>;

  // GLfloat 型の 3 要素のベクトル
  using vec3 = std::array<GLfloat, 3>;

  // 三角形データ
  struct fidx
  {
    GLuint p[3];        // 頂点座標番号
    GLuint n[3];        // 頂点法線番号
    GLuint t[3];        // テクスチャ座標番号
    bool smooth;        // スムーズシェーディングの有無
  };

```

```
// ポリゴングループ
struct fgrp
{
  GLuint nextgroup;   // 次のポリゴングループの最初の三角形番号
  GLuint mtlno;       // このポリゴングループの材質番号

  // コンストラクタ
  fgrp(GLuint nextgroup, GLuint mtlno) :
    nextgroup(nextgroup),
    mtlno(mtlno)
  {
  }
};

// デフォルトの材質
constexpr GgSimpleShader::Material defaultMaterial
{
  { 0.1f, 0.1f, 0.1f, 1.0f },
  { 0.6f, 0.6f, 0.6f, 0.0f },
  { 0.3f, 0.3f, 0.3f, 0.0f },
  60.0f
};

// デフォルトの材質名
constexpr char defaultMaterialName[] = "_default_";

//
// Alias OBJ 形式の MTL ファイルを読み込む
//
//   mtlpath MTL ファイルのパス名
//   mtl 読み込んだ材質名をキー，材質番号を値にした map
//   material 材質データ
//
static bool ggLoadMtl(const std::string& mtlpath,
  std::map<std::string, GLuint>& mtl,
  std::vector<GgSimpleShader::Material>& material)
{
  // MTL ファイルが無ければ戻る
  std::ifstream mtlfile{ mtlpath, std::ios::binary };
  if (!mtlfile)
  {
#if defined(DEBUG)
    std::cerr << "Warning: Can't open MTL file: " << mtlpath << std::endl;
#endif
    return false;
  }

  // 一行読み込み用のバッファ
  std::string mtlline;

  // 材質名（ループの外に置く）
  std::string mtlname{ defaultMaterialName };

  // 現在の材質番号を登録する
  mtl[mtlname] = static_cast<GLuint>(material.size());

  // 現在の材質にデフォルトの材質を設定する
  material.emplace_back(defaultMaterial);

  // 材質データを読み込む
  while (std::getline(mtlfile, mtlline))
  {
    // 空行は読み飛ばす
    if (mtlline == "") continue;

    // 最後の文字が '\r' なら
    if (*(mtlline.end() - 1) == '\r')
    {
      // 最後の文字を削除する
      mtlline.erase(mtlline.end() - 1, mtlline.end());

      // 空行になったら読み飛ばす
      if (mtlline == "") continue;
    }

    // 読み込んだ行を文字列ストリームにする
    std::istringstream mtlstr{ mtlline };

    // オペレータ
    std::string mtlop;

    // 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
    mtlstr >> mtlop;

    // '#' で始まる場合はコメントとして行末まで読み飛ばす
    if (mtlop[0] == '#') continue;

```

```
      if (mtlop == "newmtl")
      {
        // 新規作成する材質名を取り出す
        mtlstr >> mtlname;

        // 材質名が既に存在するかどうか調べる
        const auto m{ mtl.find(mtlname) };
        if (m == mtl.end())
        {
          // 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
          mtl[mtlname] = static_cast<GLuint>(material.size());

          // 新規作成する材質にデフォルトの材質を設定しておく
          material.emplace_back(defaultMaterial);
        }

#if defined(DEBUG)
        std::cerr << "newmtl: " << mtlname << std::endl;
#endif
      }
      else if (mtlop == "Ka")
      {
        // 環境光の反射係数を登録する
        mtlstr
          >> material.back().ambient[0]
          >> material.back().ambient[1]
          >> material.back().ambient[2];
      }
      else if (mtlop == "Kd")
      {
        // 拡散反射係数を登録する
        mtlstr
          >> material.back().diffuse[0]
          >> material.back().diffuse[1]
          >> material.back().diffuse[2];
      }
      else if (mtlop == "Ks")
      {
        // 鏡面反射係数を登録する
        mtlstr
          >> material.back().specular[0]
          >> material.back().specular[1]
          >> material.back().specular[2];
      }
      else if (mtlop == "Ns")
      {
        // 輝き係数を登録する
        GLfloat shininess;
        mtlstr >> shininess;
        material.back().shininess = shininess * 0.1f;
      }
      else if (mtlop == "d")
      {
        // 不透明度を登録する
        mtlstr >> material.back().ambient[3];
      }
    }

    // MTL ファイルの読み込みに失敗したら戻る
    if (mtlfile.bad())
    {
#if defined(DEBUG)
      std::cerr << "Warning: Can't read MTL file: " << mtlpath << std::endl;
#endif

      // MTL ファイルを閉じる
      mtlfile.close();

      // MTL ファイルが読み込みに失敗したので戻る
      return false;
    }

    // MTL ファイルを閉じる
    mtlfile.close();

    // MTL ファイルの読み込みに成功した
    return true;
  }

  //
  // Alias OBJ 形式のファイルを解析する
  //
  //   name Alias OBJ 形式のファイルのファイル名
  //   group 同じ材質を割り当てるポリゴングループ
  //   mtl 読み込んだ材質名をキーにした map
  //   pos 頂点の位置
  //   norm 頂点の法線
```

```
//   tex 頂点のテクスチャ座標
//   face 三角形のデータ
//
static bool ggParseObj(
  const std::string& name,
  std::vector<fgrp>& group,
  std::vector<GgSimpleShader::Material>& material,
  std::vector<vec3>& pos,
  std::vector<vec3>& norm,
  std::vector<vec2>& tex,
  std::vector<fidx>& face,
  bool normalize
)
{
  // ファイルパスからディレクトリ名を取り出す
  const std::string path{ name };
  const size_t base{ path.find_last_of("/\\") };
  const std::string dirname{ (base == std::string::npos) ? "" : path.substr(0, base + 1) };

  // OBJ ファイルを読み込む
  std::ifstream file{ path };

  // 読み込みに失敗したら戻る
  if (file.fail())
  {
#if defined(DEBUG)
    std::cerr << "Error: Can't open OBJ file: " << path << std::endl;
#endif
    return false;
  }

  // ポリゴングループの最初の三角形番号
  GLsizei startgroup(static_cast<GLsizei>(group.size()));

  // スムーズシェーディングのスイッチ
  bool smooth{ false };

  // 材質のテーブル
  std::map<std::string, GLuint> mtl;

  // 現在の材質名（ループの外で宣言する）
  std::string mtlname;

  // 座標値の最小値・最大値
  vec3 bmin{ FLT_MAX }, bmax{ -FLT_MAX };

  // 一行読み込み用のバッファ
  std::string line;

  // データの読み込み
  while (std::getline(file, line))
  {
    // 空行は読み飛ばす
    if (line == "") continue;

    // 最後の文字が '\r' なら
    if (*(line.end() - 1) == '\r')
    {
      // 最後の文字を削除する
      line.erase(line.end() - 1, line.end());

      // 空行になったら読み飛ばす
      if (line == "") continue;
    }

    // 一行を文字列ストリームに入れる
    std::istringstream str(line);

    // 最初のトークンを命令 (op) とみなす
    std::string op;
    str >> op;

    if (op[0] == '#') continue;

    if (op == "v")
    {
      // 頂点位置
      vec3 v;

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

      // 頂点位置を記録する
      pos.emplace_back(v);

      // 頂点位置の最小値と最大値を求める (AABB)
      for (int i = 0; i < 3; ++i)
```

```
        {
          bmin[i] = std::min(bmin[i], v[i]);
          bmax[i] = std::max(bmax[i], v[i]);
        }
      }
      else if (op == "vt")
      {
        // テクスチャ座標
        vec2 t;

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

        // テクスチャ座標を記録する
        tex.emplace_back(t);
      }
      else if (op == "vn")
      {
        // 頂点法線
        vec3 n;

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

        // 頂点法線を記録する
        norm.emplace_back(n);
      }
      else if (op == "f")
      {
        // 三角形データ
        fidx f;

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

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

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

          // テクスチャ座標と法線の番号は未定義を表す 0 にしておく
          f.p[i] = f.t[i] = f.n[i] = 0;

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

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

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

        // 三角形データを登録する
        face.emplace_back(f);
      }
      else if (op == "s")
      {
        // '1' だったらスムースシェーディング有効
        std::string s;
        str >> s;
        smooth = s == "1";
      }
      else if (op == "usemtl")
      {
        // 次のポリゴングループの最初の三角形番号
        const GLsizei nextgroup(static_cast<GLsizei>(face.size()));

        // ポリゴングループに三角形が存在すれば
        if (nextgroup > startgroup)
        {
          // ポリゴングループの三角形数と材質番号を記録する
          group.emplace_back(nextgroup, mtl[mtlname]);

          // 次のポリゴングループの開始番号を保存しておく
          startgroup = nextgroup;
        }

```

```
        // 次に usemtl が来るまで材質名を保持する
        str >> mtlname;

        // 材質の存在チェック
        if (mtl.find(mtlname) == mtl.end())
        {
#if defined(DEBUG)
          std::cerr << "Warning: Undefined material: " << mtlname << std::endl;
#endif

          // デフォルトの材質を割り当てておく
          mtlname = defaultMaterialName;
        }
#if defined(DEBUG)
        else std::cerr << "usemtl: " << mtlname << std::endl;
#endif
      }
      else if (op == "mtllib")
      {
        // MTL ファイルのパス名を作る
        str >> std::ws;
        std::string mtlpath;
        std::getline(str, mtlpath);

        // MTL ファイルを読み込む
        ggLoadMtl(dirname + mtlpath, mtl, material);
      }
    }

    // OBJ ファイルの読み込みに失敗したら戻る
    if (file.bad())
    {
#if defined(DEBUG)
      std::cerr << "Error: Can't read OBJ file: " << path << std::endl;
#endif
      file.close();
      return false;
    }

    // ファイルを閉じる
    file.close();

    // 最後のポリゴン グループの次の三角形番号
    const GLsizei nextgroup(static_cast<GLsizei>(face.size()));
    if (nextgroup > startgroup)
    {
      // 最後のポリゴン グループの三角形数と材質を記録する
      group.emplace_back(nextgroup, static_cast<GLuint>(mtl[mtlname]));
    }

    // 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
    for (auto& f : face)
    {
      if (!f.smooth)
      {
        // 三頂点のそれぞれについて
        for (int i = 0; i < 3; ++i)
        {
          // 新しい頂点座標を生成する (std::array の要素は emplace_back できない)
          pos.emplace_back(pos[f.p[i] - 1]);
          f.p[i] = static_cast<GLuint>(pos.size());

          if (f.t[i] > 0)
          {
            // 新しいテクスチャ座標を生成する
            tex.emplace_back(tex[f.t[i] - 1]);
            f.t[i] = static_cast<GLuint>(tex.size());
          }

          if (f.n[i] > 0)
          {
            // 新しい法線を生成する
            norm.emplace_back(norm[f.n[i] - 1]);
            f.n[i] = static_cast<GLuint>(norm.size());
          }
        }
      }
    }

    // 法線データがなければ算出しておく
    if (norm.empty())
    {
      // 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
      norm.resize(pos.size(), { 0.0f, 0.0f, 0.0f });

      // 面の法線の算出と頂点法線の算出
      for (auto& f : face)
```

```
      {
        // 頂点座標番号
        const GLuint v0{ f.p[0] - 1 };
        const GLuint v1{ f.p[1] - 1 };
        const GLuint v2{ f.p[2] - 1 };

        // v1 - v0, v2 - v0 を求める
        const GLfloat d1[]{ pos[v1][0] - pos[v0][0], pos[v1][1] - pos[v0][1], pos[v1][2] - pos[v0][2] };
        const GLfloat d2[]{ pos[v2][0] - pos[v0][0], pos[v2][1] - pos[v0][1], pos[v2][2] - pos[v0][2] };

        // 外積により面法線を求める
        vec3 n;
        ggCross(n.data(), d1, d2);

        if (f.smooth)
        {
          // スムースシェーディングを行うときは
          for (int i = 0; i < 3; ++i)
          {
            // 面法線を頂点法線に積算する
            norm[v0][i] += n[i];
            norm[v1][i] += n[i];
            norm[v2][i] += n[i];

            // 面の各頂点の法線番号は頂点番号と同じにする
            f.n[i] = f.p[i];
          }
        }
        else
        {
          // 面法線を最初の頂点に保存する
          norm[v0] = n;
          f.n[0] = f.p[0];

          // 2 頂点追加
          for (int i = 1; i < 3; ++i)
          {
            norm.emplace_back(n);
            f.n[i] = static_cast<GLuint>(norm.size());
          }
        }
      }

      // 頂点の法線ベクトルを正規化する
      for (auto& n : norm) ggNormalize3(n.data());
    }

    // 図形の正規化
    if (normalize)
    {
      // 図形の大きさ
      const GLfloat sx{ bmax[0] - bmin[0] };
      const GLfloat sy{ bmax[1] - bmin[1] };
      const GLfloat sz{ bmax[2] - bmin[2] };

      // 図形のスケール
      GLfloat s{ sx };
      if (sy > s) s = sy;
      if (sz > s) s = sz;
      const GLfloat scale{ s != 0.0f ? 2.0f / s : 1.0f };

      // 図形の中心位置
      const GLfloat cx{ (bmax[0] + bmin[0]) * 0.5f };
      const GLfloat cy{ (bmax[1] + bmin[1]) * 0.5f };
      const GLfloat cz{ (bmax[2] + bmin[2]) * 0.5f };

      // 図形の大きさと位置を正規化する
      for (auto& p : pos)
      {
        p[0] = (p[0] - cx) * scale;
        p[1] = (p[1] - cy) * scale;
        p[2] = (p[2] - cz) * scale;
      }
    }

#if defined(DEBUG)
    std::cerr
      << "[" << name << "]\n(Parsed) Group: " << group.size() << ", Material: " << mtl.size()
      << ", Pos: " << pos.size() << ", Norm: " << norm.size() << ", Tex: " << tex.size()
      << ", Face: " << face.size() << "\n";
#endif

    // OBJ ファイルの読み込み成功
    return true;
  }
```

```
}

//
// 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)
//
//   name 読み込むOBJ ファイル名
//   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
//   material 読み込んだデータのポリゴン グループごとの材質
//   vert 読み込んだデータの頂点属性
//   normalize true ならサイズを正規化する
//   戻り値 読み込みに成功したら true
//
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)
{
  // 読み込み用の一時記憶領域
  std::vector<fgrp> tgroup;
  std::vector<vec3> tpos;
  std::vector<vec3> tnorm;
  std::vector<vec2> ttex;
  std::vector<fidx> tface;

  // OBJ ファイルを解析する
  if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;

  // 頂点属性データのメモリを確保する
  vert.reserve(vert.size() + tface.size() * 3);

  // ポリゴン グループデータのメモリを確保する
  group.reserve(group.size() + tgroup.size());
  material.reserve(material.size() + tgroup.size());

  // ポリゴン グループの最初の三角形番号
  GLuint startgroup{ 0 };

  // ポリゴン グループデータの作成
  for (auto& g : tgroup)
  {
    // 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
    std::array<GLuint, 3> v;

    // このポリゴン グループの最初の頂点番号を保存する
    v[0] = static_cast<GLuint>(vert.size());

    // 三角形ごとの頂点データの作成
    for (GLuint j = startgroup; j < g.nextgroup; ++j)
    {
      // 処理対象の三角形
      auto& f = tface[j];

      // 三頂点のそれぞれについて
      for (int i = 0; i < 3; ++i)
      {
        // テクスチャ座標
        vec2 tex{ 0.0f, 0.0f };
        if (f.t[i] > 0) tex = ttex[f.t[i] - 1];

        // 頂点法線
        vec3 norm{ 0.0f, 0.0f, 0.0f };
        if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];

        // 頂点属性の追加
        vert.emplace_back(tpos[f.p[i] - 1].data(), norm.data());
      }
    }

    // このポリゴン グループの頂点数を保存する
    v[1] = static_cast<GLuint>(vert.size()) - v[0];
    v[2] = g.mtlno;

    // 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
    group.emplace_back(v);

    // 次のポリゴン グループの最初の三角形番号を求める
    startgroup = g.nextgroup;
  }

#if defined(DEBUG)
  std::cerr
    << "(Stored) Group: " << group.size() << ", Material: " << material.size()
    << ", Vertex: " << vert.size() << "\n";
#endif

  // OBJ ファイルの読み込み成功
```

```
  return true;
}

//
// 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)
//
//   name 読み込むOBJ ファイル名
//   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
//   material 読み込んだデータのポリゴン グループごとの材質
//   vert 読み込んだデータの頂点属性
//   face 読み込んだデータの三角形の頂点インデックス
//   normalize true ならサイズを正規化する
//   戻り値 読み込みに成功したら true
//
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)
{
  // 読み込み用の一時記憶領域
  std::vector<fgrp> tgroup;
  std::vector<vec3> tpos;
  std::vector<vec3> tnorm;
  std::vector<vec2> ttex;
  std::vector<fidx> tface;

  // OBJ ファイルを解析する
  if (!ggParseObj(name, tgroup, material, tpos, tnorm, ttex, tface, normalize)) return false;

  // 頂点属性データの最初の頂点番号
  const GLuint vertbase{ static_cast<GLuint>(vert.size()) };

  // 頂点属性データのメモリを確保する
  vert.resize(vertbase + tpos.size());

  // 三角形データのメモリを確保する
  face.reserve(face.size() + tface.size());

  // ポリゴン グループデータのメモリを確保する
  group.reserve(group.size() + tgroup.size());
  material.reserve(material.size() + tgroup.size());

  // ポリゴン グループの最初の三角形番号
  GLuint startgroup{ 0 };

  // ポリゴン グループデータの作成
  for (auto& g : tgroup)
  {
    // このポリゴン グループの最初の頂点番号
    const GLuint first{ static_cast<GLuint>(face.size()) };

    // 三角形ごとの頂点データの作成
    for (GLuint j = startgroup; j < g.nextgroup; ++j)
    {
      // 処理対象の三角形
      auto& f{ tface[j] };

      // 三頂点のそれぞれについて
      for (int i = 0; i < 3; ++i)
      {
        // 追加する三角形データの頂点番号
        const GLuint q{ f.p[i] - 1 + vertbase };

        // 三角形データの追加
        face.emplace_back(q);

        // テクスチャ座標番号
        vec2 tex{ 0.0f, 0.0f };
        if (f.t[i] > 0) tex = ttex[f.t[i] - 1];

        // 頂点法線番号
        vec3 norm{ 0.0f, 0.0f, 0.0f };
        if (f.n[i] > 0) norm = tnorm[f.n[i] - 1];

        // 頂点の格納
        vert[q] = GgVertex(tpos[f.p[i] - 1].data(), norm.data());
      }
    }

    // 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
    group.emplace_back(std::array<GLuint, 3>{ first, static_cast<GLuint>(face.size()) - first, g.mtlno
  });

    // 次のポリゴン グループの最初の三角形番号を求める
    startgroup = g.nextgroup;
```

```
  }

#if defined(DEBUG)
  std::cerr
    << "(Stored) Group: " << group.size() << ", Material: " << material.size()
    << ", Vertex: " << vert.size() << ", Face: " << face.size() << "\n";
#endif

  // OBJ ファイルの読み込み成功
  return true;
}

//
// シェーダオブジェクトのコンパイル結果を表示する
//
static GLboolean printShaderInfoLog(GLuint shader, const std::string& str)
{
  // コンパイル結果を取得する
  GLint status;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
#if defined(DEBUG)
  if (status == GL_FALSE) std::cerr << "Compile Error in " << str << std::endl;
#endif

  // シェーダのコンパイル時のログの長さを取得する
  GLsizei bufSize;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &bufSize);

  if (bufSize > 1)
  {
    // シェーダのコンパイル時のログの内容を取得する
    std::vector<GLchar> infoLog(bufSize);
    GLsizei length;
    glGetShaderInfoLog(shader, bufSize, &length, infoLog.data());
#if defined(DEBUG)
    std::cerr << infoLog.data();
#endif
  }

  // コンパイル結果を返す
  return static_cast<GLboolean>(status);
}

//
// プログラムオブジェクトのリンク結果を表示する
//
static GLboolean printProgramInfoLog(GLuint program)
{
  // リンク結果を取得する
  GLint status;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
#if defined(DEBUG)
  if (status == GL_FALSE) std::cerr << "Link Error." << std::endl;
#endif

  // シェーダのリンク時のログの長さを取得する
  GLsizei bufSize;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &bufSize);

  // シェーダのリンク時のログの内容を取得する
  if (bufSize > 1)
  {
    std::vector<GLchar> infoLog(bufSize);
    GLsizei length;
    glGetProgramInfoLog(program, bufSize, &length, infoLog.data());
#if defined(DEBUG)
    std::cerr << infoLog.data() << std::endl;
#endif
  }

  // リンク結果を返す
  return static_cast<GLboolean>(status);
}

//
// 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
//
//   vsrc バーテックスシェーダのソースプログラムの文字列
//   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)
//   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)
//   nvarying フィードバックする varying 変数の数 (0 なら不使用)
//   varyings フィードバックする varying 変数のリスト (nullptr なら不使用)
//   vtext 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
//   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
//   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
//   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)
//
```

```
GLuint gg::ggCreateShader(
  const std::string& vsrc,
  const std::string& fsrc,
  const std::string& gsrc,
  GLint nvarying,
  const char* const* varyings,
  const std::string& vtext,
  const std::string& ftext,
  const std::string& gtext)
{
  // シェーダプログラムの作成
  const GLuint program{ glCreateProgram() };

  if (program > 0)
  {
    bool status = true;

    if (!vsrc.empty())
    {
      // バ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
      const GLuint vertShader{ glCreateShader(GL_VERTEX_SHADER) };
      const GLchar* vsrcp{ vsrc.c_str() };
      glShaderSource(vertShader, 1, &vsrcp, nullptr);
      glCompileShader(vertShader);

      // バ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
      if (printShaderInfoLog(vertShader, vtext))
        glAttachShader(program, vertShader);
      else
        status = false;
      glDeleteShader(vertShader);
    }

    if (!fsrc.empty())
    {
      // 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
      const GLuint fragShader{ glCreateShader(GL_FRAGMENT_SHADER) };
      const GLchar* fsrcp{ fsrc.c_str() };
      glShaderSource(fragShader, 1, &fsrcp, nullptr);
      glCompileShader(fragShader);

      // 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
      if (printShaderInfoLog(fragShader, ftext))
        glAttachShader(program, fragShader);
      else
        status = false;
      glDeleteShader(fragShader);
    }

    if (!gsrc.empty())
    {
#if defined(GL_GLES_PROTOTYPES)
#  if defined(DEBUG)
      std::cerr << gtext << ": The geometry is not supported." << std::endl;
      status = false;
#  endif
#else
      // 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
      const GLuint geomShader{ glCreateShader(GL_GEOMETRY_SHADER) };
      const GLchar* gsrcp{ gsrc.c_str() };
      glShaderSource(geomShader, 1, &gsrcp, nullptr);
      glCompileShader(geomShader);

      // 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
      if (printShaderInfoLog(geomShader, gtext))
        glAttachShader(program, geomShader);
      else
        status = false;
      glDeleteShader(geomShader);
#endif
    }

    // 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
    if (status)
    {
      // feedback に使う varying 変数を指定する
      if (nvarying > 0)
        glTransformFeedbackVaryings(program, nvarying, varyings, GL_SEPARATE_ATTRIBS);

      // シェーダプログラムをリンクする
      glLinkProgram(program);

      // リンクに成功したらプログラムオブジェクト名を返す
      if (printProgramInfoLog(program) != GL_FALSE) return program;
    }
  }

```

```
  // プログラムオブジェクトが作成できなかった
  glDeleteProgram(program);
  return 0;
}

//
// シェーダのソースファイルを読み込んだ vector を返す
//
//   name ソースファイル名
//   src 読み込んだソースファイルの文字列
//   戻り値 読み込みの成功すれば true, 失敗したら false
//
static bool readShaderSource(const std::string& name, std::string& src)
{
  // ファイル名が nullptr ならそのまま戻る
  if (name.empty()) return true;

  // ソースファイルを開く
  std::ifstream file{ name, std::ios::binary };
  if (file.fail())
  {
    // ファイルが開けなければエラーで戻る
#if defined(DEBUG)
    std::cerr << "Error: Can't open source file: " << name << std::endl;
#endif
    return false;
  }

  // ファイル全体を文字列として読み込む
  std::istreambuf_iterator<char> it{ file };
  std::istreambuf_iterator<char> last;
  src = std::string(it, last);

  // ファイルがうまく読み込めなければ戻る
  if (file.bad())
  {
#if defined(DEBUG)
    std::cerr << "Error: Could not read souce file: " << name << std::endl;
#endif
    file.close();
    return false;
  }

  // ファイルを閉じて戻る
  file.close();
  return true;
}

//
// 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
//
//   vert バーテックスシェーダのソースファイル名
//   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)
//   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)
//   nvarying フィードバックする varying 変数の数 (0 なら不使用)
//   varyings 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 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)
//
GLuint gg::ggLoadShader(
  const std::string& vert,
  const std::string& frag,
  const std::string& geom,
  GLint nvarying,
  const char* const* varyings
)
{
  // 読み込んだシェーダのソースプログラム
  std::string vsrc, fsrc, gsrc;

  // 読み込んだ結果
  bool status;

  // シェーダのソースファイルを読み込む
  status = readShaderSource(vert, vsrc);
  status = readShaderSource(frag, fsrc) && status;
  status = readShaderSource(geom, gsrc) && status;

  // 全てのソースファイルが読み込めていなかったらエラー
  if (!status) return 0;

  // プログラムオブジェクトを作成する
  return ggCreateShader(vsrc, fsrc, gsrc, nvarying, varyings, vert, frag, geom);
}

#if !defined(__APPLE__) && !defined(GL_GLES_PROTOTYPES)
//
// 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
```

```
//
//   csrc コンピュートシェーダのソースプログラムの文字列
//   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)
//
GLuint gg::ggCreateComputeShader(const std::string& csrc, const std::string& ctext)
{
  // シェーダプログラムの作成
  const GLuint program{ glCreateProgram() };

  if (program > 0)
  {
    // ソースプログラムの文字列が空だったら 0 を返す
    if (csrc.empty()) return 0;

    // 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
    const GLuint compShader{ glCreateShader(GL_COMPUTE_SHADER) };
    const GLchar* csrcp{ csrc.c_str() };
    glShaderSource(compShader, 1, &csrcp, nullptr);
    glCompileShader(compShader);

    // 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
    if (printShaderInfoLog(compShader, ctext)) glAttachShader(program, compShader);
    glDeleteShader(compShader);

    // シェーダプログラムをリンクする
    glLinkProgram(program);

    // プログラムオブジェクトが作成できなければ 0 を返す
    if (printProgramInfoLog(program) == GL_FALSE)
    {
      glDeleteProgram(program);
      return 0;
    }
  }

  // プログラムオブジェクトを返す
  return program;
}

//
// 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
//
//   comp コンピュートシェーダのソースファイル名
//   戻り値 プログラムオブジェクトのプログラム名 (作成できなければ 0)
//
GLuint gg::ggLoadComputeShader(const std::string& comp)
{
  // シェーダのソースファイルを読み込む
  std::string csrc;
  if (readShaderSource(comp, csrc))
  {
    // プログラムオブジェクトを作成する
    return ggCreateComputeShader(csrc.data(), comp);
  }

  // プログラムオブジェクト作成失敗
  return 0;
}
#endif

//
// 点：データ作成
//
void gg::GgPoints::load(const GgVector* pos, GLsizei count, GLenum usage)
{
  // 頂点バッファオブジェクトを作成する
  position = std::make_shared<GgBuffer<GgVector>>(GL_ARRAY_BUFFER, pos,
 static_cast<GLsizei>(sizeof(GgVector)), count, usage);

  // 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
  glVertexAttribPointer(0, static_cast<GLint>(pos->size()), GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(0);
}

//
// 点：描画
//
void gg::GgPoints::draw(GLint first, GLsizei count) const
{
  // 頂点配列オブジェクトを指定する
  GgShape::draw(first, count);

  // 図形を描画する
  glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
}

//
```

```
// 三角形：データ作成
//
void gg::GgTriangles::load(const GgVertex* vert, GLsizei count, GLenum usage)
{
  // 頂点バッファオブジェクトを作成する
  vertex = std::make_unique<GgBuffer<GgVertex>>(GL_ARRAY_BUFFER, vert, static_cast<GLsizei>(sizeof(GgVertex)), count, usage);

  // 頂点の位置は index == 0 の in 変数から入力する
  glVertexAttribPointer(0, static_cast<GLint>(vert->position.size()), GL_FLOAT, GL_FALSE,
    sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, position));
  glEnableVertexAttribArray(0);

  // 頂点の法線は index == 1 の in 変数から入力する
  glVertexAttribPointer(1, static_cast<GLint>(vert->normal.size()), GL_FLOAT, GL_FALSE,
    sizeof(GgVertex), static_cast<const char*>(0) + offsetof(GgVertex, normal));
  glEnableVertexAttribArray(1);
}

//
// 三角形：描画
//
void gg::GgTriangles::draw(GLint first, GLsizei count) const
{
  // 頂点配列オブジェクトを指定する
  GgShape::draw(first, count);

  // 図形を描画する
  glDrawArrays(getMode(), first, count > 0 ? count : getCount() - first);
}

//
// オブジェクト：描画
//
void gg::GgElements::draw(GLint first, GLsizei count) const
{
  // 頂点配列オブジェクトを指定する
  GgShape::draw(first, count);

  // 図形を描画する
  glDrawElements(getMode(), count > 0 ? count : getIndexCount() - first,
    GL_UNSIGNED_INT, static_cast<GLuint*>(0) + first);
}

//
// 点群を立方体状に生成する
//
std::unique_ptr<gg::GgPoints> gg::ggPointsCube(GLsizei count, GLfloat length, GLfloat cx, GLfloat cy, GLfloat cz)
{
  // メモリを確保する
  std::vector<GgVector> pos(count);

  // 点を生成する
  for (GLsizei v = 0; v < count; ++v)
  {
    const GgVector p
    {
      (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cx,
      (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cy,
      (static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 0.5f) * length + cz,
      1.0f
    };

    pos.emplace_back(p);
  }

  // 点データの GgPoints オブジェクトを作成して返す
  return std::make_unique<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
}

//
// 点群を球状に生成する
//
std::unique_ptr<gg::GgPoints> gg::ggPointsSphere(GLsizei count, GLfloat radius,
  GLfloat cx, GLfloat cy, GLfloat cz)
{
  // メモリを確保する
  std::vector<GgVector> pos(count);

  // 点を生成する
  for (GLsizei v = 0; v < count; ++v)
  {
    const GLfloat r{ radius * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) };
    const GLfloat t{ 6.28318530718f * static_cast<GLfloat>(rand()) / (static_cast<GLfloat>(RAND_MAX) + 1.0f) };
    const GLfloat cp{ 2.0f * static_cast<GLfloat>(rand()) / static_cast<GLfloat>(RAND_MAX) - 1.0f };
```

```
    const GLfloat sp{ sqrtf(1.0f - cp * cp) };
    const GLfloat ct{ cosf(t) };
    const GLfloat st{ sinf(t) };
    const GgVector p{ r * sp * ct + cx, r * sp * st + cy, r * cp + cz, 1.0f };

    pos.emplace_back(p);
  }

  // 点データの GgPoints オブジェクトを作成して返す
  return std::make_unique<gg::GgPoints>(pos.data(), static_cast<GLsizei>(pos.size()), GL_POINTS);
}

//
// 矩形状に 2 枚の三角形を生成する
//
std::unique_ptr<gg::GgTriangles> gg::ggRectangle(GLfloat width, GLfloat height)
{
  // 頂点属性
  std::array<gg::GgVertex, 4> vert;

  // 頂点位置と法線を求める
  for (int v = 0; v < 4; ++v)
  {
    const GLfloat x{ ((v & 1) * 2 - 1) * width };
    const GLfloat y{ ((v & 2) - 1) * height };

    vert[v] = gg::GgVertex{ x, y, 0.0f, 0.0f, 0.0f, 1.0f };
  }

  // 矩形の GgTrianges オブジェクトを作成する
  return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
 GL_TRIANGLE_STRIP);
}

//
// 楕円状に三角形を生成する
//
std::unique_ptr<gg::GgTriangles> gg::ggEllipse(GLfloat width, GLfloat height, GLuint slices)
{
  // 楕円のスケール
  constexpr GLfloat scale{ 0.5f };

  // 作業用のメモリ
  std::vector<gg::GgVertex> vert;

  // 頂点位置と法線を求める
  for (GLuint v = 0; v < slices; ++v)
  {
    const GLfloat t{ 6.28318530717f * static_cast<GLfloat>(v) / static_cast<GLfloat>(slices) };
    const GLfloat x{ cosf(t) * width * scale };
    const GLfloat y{ sinf(t) * height * scale };

    vert.emplace_back(x, y, 0.0f, 0.0f, 0.0f, 1.0f);
  }

  // GgTriangles オブジェクトを作成する
  return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
 GL_TRIANGLE_FAN);
}

//
// Wavefront OBJ ファイルを読み込む (Arrays 形式)
//
std::unique_ptr<gg::GgTriangles> gg::ggArraysObj(const std::string& name, bool normalize)
{
  std::vector<std::array<GLuint, 3>> group;
  std::vector<gg::GgSimpleShader::Material> material;
  std::vector<gg::GgVertex> vert;

  // ファイルを読み込む
  if (!ggLoadSimpleObj(name, group, material, vert, normalize)) return nullptr;

  // GgTriangles オブジェクトを作成する
  return std::make_unique<gg::GgTriangles>(vert.data(), static_cast<GLsizei>(vert.size()),
 GL_TRIANGLES);
}

//
// Wavefront OBJ ファイル を読み込む (Elements 形式)
//
std::unique_ptr<gg::GgElements> gg::ggElementsObj(const std::string& name, bool normalize)
{
  std::vector<std::array<GLuint, 3>> group;
  std::vector<gg::GgSimpleShader::Material> material;
  std::vector<gg::GgVertex> vert;
  std::vector<GLuint> face;

```

```
  // ファイルを読み込む
  if (!ggLoadSimpleObj(name, group, material, vert, face, normalize)) return nullptr;

  // GgElements オブジェクトを作成する
  return std::make_unique<gg::GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
    face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
}

//
// メッシュ形状を作成する (Elements 形式)
//
std::unique_ptr<gg::GgElements> gg::ggElementsMesh(GLuint slices, GLuint stacks, const GLfloat(*pos)[3], const GLfloat(*norm)[3])
{
  // 頂点属性
  std::vector<gg::GgVertex> vert((slices + 1) * (stacks + 1));

  // 頂点の法線を求める
  for (GLuint j = 0; j <= stacks; ++j)
  {
    for (GLuint i = 0; i <= slices; ++i)
    {
      // 処理対象の頂点番号
      const GLuint k{ j * (slices + 1) + i };

      // 頂点の法線
      gg::GgVector tnorm;
      tnorm[3] = 0.0f;

      if (norm)
      {
        tnorm[0] = norm[k][0];
        tnorm[1] = norm[k][1];
        tnorm[2] = norm[k][2];
      }
      else
      {
        // 処理対象の頂点の周囲の頂点番号
        const GLuint kim{ i > 0 ? k - 1 : k };
        const GLuint kip{ i < slices ? k + 1 : k };
        const GLuint kjm{ j > 0 ? k - slices - 1 : k };
        const GLuint kjp{ j < stacks ? k + slices + 1 : k };

        // 接線ベクトル
        const std::array<GLfloat, 3> t
        {
          pos[kip][0] - pos[kim][0],
          pos[kip][1] - pos[kim][1],
          pos[kip][2] - pos[kim][2]
        };

        // 従接線ベクトル
        const std::array<GLfloat, 3> b
        {
          pos[kjp][0] - pos[kjm][0],
          pos[kjp][1] - pos[kjm][1],
          pos[kjp][2] - pos[kjm][2]
        };

        // 法線
        tnorm[0] = t[1] * b[2] - t[2] * b[1];
        tnorm[1] = t[2] * b[0] - t[0] * b[2];
        tnorm[2] = t[0] * b[1] - t[1] * b[0];

        // 法線の正規化
        ggNormalize3(tnorm);
      }

      // 頂点の位置
      const gg::GgVector tpos{ pos[k][0], pos[k][1], pos[k][2], 1.0f };

      // 頂点属性の保存
      vert.emplace_back(tpos, tnorm);
    }
  }

  // 頂点のインデックス (三角形データ)
  std::vector<GLuint> face;

  // 頂点のインデックスを求める
  for (GLuint j = 0; j < stacks; ++j)
  {
    for (GLuint i = 0; i < slices; ++i)
    {
      // 処理対象のマス
      const GLuint k{ (slices + 1) * j + i };

```

```
      // マスの上半分の三角形
      face.emplace_back(k);
      face.emplace_back(k + slices + 2);
      face.emplace_back(k + 1);

      // マスのお下半分の三角形
      face.emplace_back(k);
      face.emplace_back(k + slices + 1);
      face.emplace_back(k + slices + 2);
    }
  }

  // GgElements オブジェクトを作成する
  return std::make_unique<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
    face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);
}

//
// 球状に三角形データを生成する (Elements 形式)
//
std::unique_ptr<gg::GgElements> gg::ggElementsSphere(GLfloat radius, int slices, int stacks)
{
  // 頂点の位置と法線
  std::vector<GLfloat> p, n;

  // 頂点の位置と法線を求める
  for (int j = 0; j <= stacks; ++j)
  {
    const GLfloat t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };
    const GLfloat ph{ 3.1415926536f * t };
    const GLfloat y{ cosf(ph) };
    const GLfloat r{ sinf(ph) };

    for (int i = 0; i <= slices; ++i)
    {
      const GLfloat s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };
      const GLfloat th{ -2.0f * 3.1415926536f * s };
      const GLfloat x{ r * cosf(th) };
      const GLfloat z{ r * sinf(th) };

      // 頂点の座標値
      p.emplace_back(x * radius);
      p.emplace_back(y * radius);
      p.emplace_back(z * radius);

      // 頂点の法線
      n.emplace_back(x);
      n.emplace_back(y);
      n.emplace_back(z);
    }
  }

  // GgElements オブジェクトを作成する
  return ggElementsMesh(slices, stacks, reinterpret_cast<GLfloat(*)[3]>(p.data()),
    reinterpret_cast<GLfloat(*)[3]>(p.data()));
}

//
// 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
//
//   r 光源の強度の環境光成分の赤成分
//   g 光源の強度の環境光成分の緑成分
//   b 光源の強度の環境光成分の青成分
//   a 光源の強度の環境光成分の不透明度, デフォルトは 1
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の環境光成分を設定する
    light->ambient[0] = r;
    light->ambient[1] = g;
    light->ambient[2] = b;
    light->ambient[3] = a;
  }
  unmap();
}

//
```

```
// 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
//
//   ambient 光源の強度の環境光成分
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadAmbient(const GgVector& ambient,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

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

//
// 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
//
//   r 光源の強度の拡散反射光成分の赤成分
//   g 光源の強度の拡散反射光成分の緑成分
//   b 光源の強度の拡散反射光成分の青成分
//   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
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の拡散反射光成分を設定する
    light->diffuse[0] = r;
    light->diffuse[1] = g;
    light->diffuse[2] = b;
    light->diffuse[3] = a;
  }
  unmap();
}

//
// 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
//
//   ambient 光源の強度の拡散反射光成分
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadDiffuse(const GgVector& diffuse,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の強度の拡散反射光成分を設定する
    light->diffuse = diffuse;
  }
  unmap();
}

//
// 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
//
//   r 光源の強度の鏡面反射光成分の赤成分
//   g 光源の強度の鏡面反射光成分の緑成分
//   b 光源の強度の鏡面反射光成分の青成分
//   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
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
```

```
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の鏡面反射光成分を設定する
    light->specular[0] = r;
    light->specular[1] = g;
    light->specular[2] = b;
    light->specular[3] = a;
  }
  unmap();
}

//
// 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
//
//   ambient 光源の強度の鏡面反射光成分
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadSpecular(const GgVector& specular,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の強度の鏡面反射光成分を設定する
    light->specular = specular;
  }
  unmap();
}

//
// 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が位置は変更しない
//
//   material 光源の特性の GgSimpleShader::Light 構造体
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadColor(const Light& color,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の色を設定する
    light->ambient = color.ambient;
    light->diffuse = color.diffuse;
    light->specular = color.specular;
  }
  unmap();
}

//
// 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
//
//   x 光源の位置の x 座標
//   y 光源の位置の y 座標
//   z 光源の位置の z 座標
//   w 光源の位置の w 座標, デフォルトは 1
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadPosition(GLfloat x, GLfloat y, GLfloat z, GLfloat w,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の位置を設定する
    light->position[0] = x;
    light->position[1] = y;
```

```
    light->position[2] = z;
    light->position[3] = w;
  }
  unmap();
}

//
// 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
//
//   position 光源の位置
//   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
//   count 値を設定する光源データの数, デフォルトは 1
//
void gg::GgSimpleShader::LightBuffer::loadPosition(const GgVector& position,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Light* const light{ reinterpret_cast<Light*>(start + getStride() * i) };

    // 光源の位置を設定する
    light->position = position;
  }
  unmap();
}

//
// 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
//
//   r 環境光に対する反射係数の赤成分
//   g 環境光に対する反射係数の緑成分
//   b 環境光に対する反射係数の青成分
//   a 環境光に対する反射係数の不透明度, デフォルトは 1
//   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
//   count 値を設定する材質データの数, デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadAmbient(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };

    // 環境光に対する反射係数を設定する
    material->ambient[0] = r;
    material->ambient[1] = g;
    material->ambient[2] = b;
    material->ambient[3] = a;
  }
  unmap();
}

//
// 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
//
//   r 拡散反射係数の赤成分
//   g 拡散反射係数の緑成分
//   b 拡散反射係数の青成分
//   a 拡散反射係数の不透明度, デフォルトは 1
//   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
//   count 値を設定する材質データの数, デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };

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

```

```
//
// 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
//
//   r 環境光に対する反射係数と拡散反射係数の赤成分
//   g 環境光に対する反射係数と拡散反射係数の緑成分
//   b 環境光に対する反射係数と拡散反射係数の青成分
//   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
//   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
//   count 値を設定する材質データの数，デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };

    // 環境光に対する反射係数と拡散反射係数を設定する
    material->ambient[0] = material->diffuse[0] = r;
    material->ambient[1] = material->diffuse[1] = g;
    material->ambient[2] = material->diffuse[2] = b;
    material->ambient[3] = material->diffuse[3] = a;
  }
  unmap();
}

//
// 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
//
//   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
//   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
//   count 値を設定する材質データの数，デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadAmbientAndDiffuse(const GLfloat* color,
  GLint first, GLsizei count) const
{
  // ambient 要素のバイトオフセット
  constexpr GLint ambientOffset{ offsetof(Material, ambient) };

  // ambient 要素のバイト数
  constexpr size_t ambientSize{ sizeof(Material::diffuse) };

  // diffuse 要素のバイトオフセット
  constexpr GLint diffuseOffset{ offsetof(Material, diffuse) };

  // diffuse 要素のバイト数
  constexpr size_t diffuseSize{ sizeof(Material::diffuse) };

  // 元のデータの先頭
  const char* source{ reinterpret_cast<const char*>(color) };

  // first 番目のブロックから count 個の ambient 要素と diffuse 要素に値を設定する
  bind();
  for (GLsizei i = 0; i < count; ++i)
  {
    // 格納先
    const GLsizeiptr destination{ getStride() * (first + i) };

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

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

//
// 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
//
//   r 鏡面反射係数の赤成分
//   g 鏡面反射係数の緑成分
//   b 鏡面反射係数の青成分
//   a 鏡面反射係数の不透明度，デフォルトは 1
//   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
//   count 値を設定する材質データの数，デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadSpecular(GLfloat r, GLfloat g, GLfloat b, GLfloat a,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
```

```
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };

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

//
// 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
//
//   shininess 輝き係数
//   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
//   count 値を設定する材質データの数, デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadShininess(GLfloat shininess,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
    material->shininess = shininess;
  }
  unmap();
}

//
// 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
//
//   shininess 輝き係数
//   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
//   count 値を設定する材質データの数, デフォルトは 1
//
void gg::GgSimpleShader::MaterialBuffer::loadShininess(const GLfloat* shininess,
  GLint first, GLsizei count) const
{
  // 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
  char* const start{ static_cast<char*>(map(first, count)) };
  for (GLsizei i = 0; i < count; ++i)
  {
    // バッファオブジェクトの i 番目のブロックのポインタ
    Material* const material{ reinterpret_cast<Material*>(start + getStride() * i) };
    material->shininess = shininess[i];
  }
  unmap();
}

//
// 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
//
bool gg::GgSimpleShader::load(const std::string& vert, const std::string& frag, const std::string&
 geom,
  GLint nvarying, const char* const* varyings)
{
  if (!GgPointShader::load(vert, frag, geom, nvarying, varyings)) return false;

  mnLoc = glGetUniformLocation(get(), "mn");
  lightIndex = glGetUniformBlockIndex(get(), "Light");
  glUniformBlockBinding(get(), lightIndex, 0);
  materialIndex = glGetUniformBlockIndex(get(), "Material");
  glUniformBlockBinding(get(), materialIndex, 1);

  return true;
}

//
// Wavefront OBJ 形式のデータ：コンストラクタ
//
gg::GgSimpleObj::GgSimpleObj(const std::string& name, bool normalize)
{
  // 作業用のメモリ
  std::vector<GgSimpleShader::Material> mat;
  std::vector<GgVertex> vert;
  std::vector<GLuint> face;

  // グループのデータのメモリを確保する
  group = std::make_shared<std::vector<std::array<GLuint, 3>>>();

  // ファイルを読み込む
  if (ggLoadSimpleObj(name, *group, mat, vert, face, normalize))
```

```
  {
    // 頂点バッファオブジェクトを作成する
    data = std::make_shared<GgElements>(vert.data(), static_cast<GLsizei>(vert.size()),
      face.data(), static_cast<GLsizei>(face.size()), GL_TRIANGLES);

    // 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
    glBindVertexArray(0);

    // 材質データを設定する
    material = std::make_shared<GgSimpleShader::MaterialBuffer>(mat.data(),
  static_cast<GLsizei>(mat.size()));
  }
}

//
// Wavefront OBJ 形式のデータ：図形の描画
//
void gg::GgSimpleObj::draw(GLint first, GLsizei count) const
{
  // 保持しているグループの数
  const GLsizei ng{ static_cast<GLsizei>(group->size()) };

  // 描画する最後のグループの次
  GLsizei last(count <= 0 ? ng : first + count);
  if (last > ng) last = ng;

  for (GLsizei i = first; i < last; ++i)
  {
    // グループのデータ
    const auto& g{ (*group)[i] };

    // 材質を設定する
    material->select(g[2]);

    // 図形を描画する
    data->draw(g[0], g[1]);
  }
}
```

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 &gtext="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)
```

```
#      define DEBUG
#    endif
#    define GLFW3_CONFIGURATION "Debug"
#  else
#    if !defined(NDEBUG)
#      define NDEBUG
#    endif
#    define GLFW3_CONFIGURATION "Release"
#  endif
// プラットフォームを調べる
#  if defined(_WIN64)
#    define GLFW3_PLATFORM "x64"
#  else
#    define GLFW3_PLATFORM "Win32"
#  endif
#endif

// OpenGL 3.2 の API のエントリポイント
#if !defined(GL3_PROTOTYPES) && !defined(GL_GLES_PROTOTYPES)
extern PFNGLACTIVEPROGRAMEXTPROC glActiveProgramEXT;
extern PFNGLACTIVESHADERPROGRAMPROC glActiveShaderProgram;
extern PFNGLACTIVETEXTUREPROC glActiveTexture;
extern PFNGLAPPLYFRAMEBUFFERATTACHMENTCMAAINTELPROC glApplyFramebufferAttachmentCMAAINTEL;
extern PFNGLATTACHSHADERPROC glAttachShader;
extern PFNGLBEGINCONDITIONALRENDERNVPROC glBeginConditionalRenderNV;
extern PFNGLBEGINCONDITIONALRENDERPROC glBeginConditionalRender;
extern PFNGLBEGINPERFMONITORAMDPROC glBeginPerfMonitorAMD;
extern PFNGLBEGINPERFQUERYINTELPROC glBeginPerfQueryINTEL;
extern PFNGLBEGINQUERYINDEXEDPROC glBeginQueryIndexed;
extern PFNGLBEGINQUERYPROC glBeginQuery;
extern PFNGLBEGINTRANSFORMFEEDBACKPROC glBeginTransformFeedback;
extern PFNGLBINDATTRIBLOCATIONPROC glBindAttribLocation;
extern PFNGLBINDBUFFERBASEPROC glBindBufferBase;
extern PFNGLBINDBUFFERPROC glBindBuffer;
extern PFNGLBINDBUFFERRANGEPROC glBindBufferRange;
extern PFNGLBINDBUFFERSBASEPROC glBindBuffersBase;
extern PFNGLBINDBUFFERSRANGEPROC glBindBuffersRange;
extern PFNGLBINDFRAGDATALOCATIONINDEXEDPROC glBindFragDataLocationIndexed;
extern PFNGLBINDFRAGDATALOCATIONPROC glBindFragDataLocation;
extern PFNGLBINDFRAMEBUFFERPROC glBindFramebuffer;
extern PFNGLBINDIMAGETEXTUREPROC glBindImageTexture;
extern PFNGLBINDIMAGETEXTURESPROC glBindImageTextures;
extern PFNGLBINDMULTITEXTUREEXTPROC glBindMultiTextureEXT;
extern PFNGLBINDPROGRAMPIPELINEPROC glBindProgramPipeline;
extern PFNGLBINDRENDERBUFFERPROC glBindRenderbuffer;
extern PFNGLBINDSAMPLERPROC glBindSampler;
extern PFNGLBINDSAMPLERSPROC glBindSamplers;
extern PFNGLBINDTEXTUREPROC glBindTexture;
extern PFNGLBINDTEXTURESPROC glBindTextures;
extern PFNGLBINDTEXTUREUNITPROC glBindTextureUnit;
extern PFNGLBINDTRANSFORMFEEDBACKPROC glBindTransformFeedback;
extern PFNGLBINDVERTEXARRAYPROC glBindVertexArray;
extern PFNGLBINDVERTEXBUFFERPROC glBindVertexBuffer;
extern PFNGLBINDVERTEXBUFFERSPROC glBindVertexBuffers;
extern PFNGLBLENDBARRIERKHRPROC glBlendBarrierKHR;
extern PFNGLBLENDBARRIERNVPROC glBlendBarrierNV;
extern PFNGLBLENDCOLORPROC glBlendColor;
extern PFNGLBLENDEQUATIONIARBPROC glBlendEquationiARB;
extern PFNGLBLENDEQUATIONIPROC glBlendEquationi;
extern PFNGLBLENDEQUATIONPROC glBlendEquation;
extern PFNGLBLENDEQUATIONSEPARATEIARBPROC glBlendEquationSeparateiARB;
extern PFNGLBLENDEQUATIONSEPARATEIPROC glBlendEquationSeparatei;
extern PFNGLBLENDEQUATIONSEPARATEPROC glBlendEquationSeparate;
extern PFNGLBLENDFUNCIARBPROC glBlendFunciARB;
extern PFNGLBLENDFUNCIPROC glBlendFunci;
extern PFNGLBLENDFUNCPROC glBlendFunc;
extern PFNGLBLENDFUNCSEPARATEIARBPROC glBlendFuncSeparateiARB;
extern PFNGLBLENDFUNCSEPARATEIPROC glBlendFuncSeparatei;
extern PFNGLBLENDFUNCSEPARATEPROC glBlendFuncSeparate;
extern PFNGLBLENDPARAMETERINVPROC glBlendParameteriNV;
extern PFNGLBLITFRAMEBUFFERPROC glBlitFramebuffer;
extern PFNGLBLITNAMEDFRAMEBUFFERPROC glBlitNamedFramebuffer;
extern PFNGLBUFFERADDRESSRANGENVPROC glBufferAddressRangeNV;
extern PFNGLBUFFERDATAPROC glBufferData;
extern PFNGLBUFFERPAGECOMMITMENTARBPROC glBufferPageCommitmentARB;
extern PFNGLBUFFERSTORAGEPROC glBufferStorage;
extern PFNGLBUFFERSUBDATAPROC glBufferSubData;
extern PFNGLCALLCOMMANDLISTNVPROC glCallCommandListNV;
extern PFNGLCHECKFRAMEBUFFERSTATUSPROC glCheckFramebufferStatus;
extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSEXTPROC glCheckNamedFramebufferStatusEXT;
extern PFNGLCHECKNAMEDFRAMEBUFFERSTATUSPROC glCheckNamedFramebufferStatus;
extern PFNGLCLAMPCOLORPROC glClampColor;
extern PFNGLCLEARBUFFERDATAPROC glClearBufferData;
extern PFNGLCLEARBUFFERFIPROC glClearBufferfi;
extern PFNGLCLEARBUFFERFVPROC glClearBufferfv;
extern PFNGLCLEARBUFFERIVPROC glClearBufferiv;
extern PFNGLCLEARBUFFERSUBDATAPROC glClearBufferSubData;
```

```
extern PFNGLCLEARBUFFERUIVPROC glClearBufferuiv;
extern PFNGLCLEARCOLORPROC glClearColor;
extern PFNGLCLEARDEPTHFPROC glClearDepthf;
extern PFNGLCLEARDEPTHPROC glClearDepth;
extern PFNGLCLEARNAMEDBUFFERDATAEXTPROC glClearNamedBufferDataEXT;
extern PFNGLCLEARNAMEDBUFFERDATAPROC glClearNamedBufferData;
extern PFNGLCLEARNAMEDBUFFERSUBDATAEXTPROC glClearNamedBufferSubDataEXT;
extern PFNGLCLEARNAMEDBUFFERSUBDATAPROC glClearNamedBufferSubData;
extern PFNGLCLEARNAMEDFRAMEBUFFERFIPROC glClearNamedFramebufferfi;
extern PFNGLCLEARNAMEDFRAMEBUFFERFVPROC glClearNamedFramebufferfv;
extern PFNGLCLEARNAMEDFRAMEBUFFERIVPROC glClearNamedFramebufferiv;
extern PFNGLCLEARNAMEDFRAMEBUFFERUIVPROC glClearNamedFramebufferuiv;
extern PFNGLCLEARPROC glClear;
extern PFNGLCLEARSTENCILPROC glClearStencil;
extern PFNGLCLEARTEXIMAGEPROC glClearTexImage;
extern PFNGLCLEARTEXSUBIMAGEPROC glClearTexSubImage;
extern PFNGLCLIENTATTRIBDEFAULTEXTPROC glClientAttribDefaultEXT;
extern PFNGLCLIENTWAITSYNCPROC glClientWaitSync;
extern PFNGLCLIPCONTROLPROC glClipControl;
extern PFNGLCOLORFORMATNVPROC glColorFormatNV;
extern PFNGLCOLORMASKIPROC glColorMaski;
extern PFNGLCOLORMASKPROC glColorMask;
extern PFNGLCOMMANDLISTSEGMENTSNVPROC glCommandListSegmentsNV;
extern PFNGLCOMPILECOMMANDLISTNVPROC glCompileCommandListNV;
extern PFNGLCOMPILESHADERINCLUDEARBPROC glCompileShaderIncludeARB;
extern PFNGLCOMPILESHADERPROC glCompileShader;
extern PFNGLCOMPRESSEDMULTITEXIMAGE1DEXTPROC glCompressedMultiTexImage1DEXT;
extern PFNGLCOMPRESSEDMULTITEXIMAGE2DEXTPROC glCompressedMultiTexImage2DEXT;
extern PFNGLCOMPRESSEDMULTITEXIMAGE3DEXTPROC glCompressedMultiTexImage3DEXT;
extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE1DEXTPROC glCompressedMultiTexSubImage1DEXT;
extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE2DEXTPROC glCompressedMultiTexSubImage2DEXT;
extern PFNGLCOMPRESSEDMULTITEXSUBIMAGE3DEXTPROC glCompressedMultiTexSubImage3DEXT;
extern PFNGLCOMPRESSEDTEXIMAGE1DPROC glCompressedTexImage1D;
extern PFNGLCOMPRESSEDTEXIMAGE2DPROC glCompressedTexImage2D;
extern PFNGLCOMPRESSEDTEXIMAGE3DPROC glCompressedTexImage3D;
extern PFNGLCOMPRESSEDTEXSUBIMAGE1DPROC glCompressedTexSubImage1D;
extern PFNGLCOMPRESSEDTEXSUBIMAGE2DPROC glCompressedTexSubImage2D;
extern PFNGLCOMPRESSEDTEXSUBIMAGE3DPROC glCompressedTexSubImage3D;
extern PFNGLCOMPRESSEDTEXTUREIMAGE1DEXTPROC glCompressedTextureImage1DEXT;
extern PFNGLCOMPRESSEDTEXTUREIMAGE2DEXTPROC glCompressedTextureImage2DEXT;
extern PFNGLCOMPRESSEDTEXTUREIMAGE3DEXTPROC glCompressedTextureImage3DEXT;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE1DEXTPROC glCompressedTextureSubImage1DEXT;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE1DPROC glCompressedTextureSubImage1D;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DEXTPROC glCompressedTextureSubImage2DEXT;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE2DPROC glCompressedTextureSubImage2D;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DEXTPROC glCompressedTextureSubImage3DEXT;
extern PFNGLCOMPRESSEDTEXTURESUBIMAGE3DPROC glCompressedTextureSubImage3D;
extern PFNGLCONSERVATIVERASTERPARAMETERFNVPROC glConservativeRasterParameterfNV;
extern PFNGLCONSERVATIVERASTERPARAMETERINVPROC glConservativeRasterParameteriNV;
extern PFNGLCOPYBUFFERSUBDATAPROC glCopyBufferSubData;
extern PFNGLCOPYIMAGESUBDATAPROC glCopyImageSubData;
extern PFNGLCOPYMULTITEXIMAGE1DEXTPROC glCopyMultiTexImage1DEXT;
extern PFNGLCOPYMULTITEXIMAGE2DEXTPROC glCopyMultiTexImage2DEXT;
extern PFNGLCOPYMULTITEXSUBIMAGE1DEXTPROC glCopyMultiTexSubImage1DEXT;
extern PFNGLCOPYMULTITEXSUBIMAGE2DEXTPROC glCopyMultiTexSubImage2DEXT;
extern PFNGLCOPYMULTITEXSUBIMAGE3DEXTPROC glCopyMultiTexSubImage3DEXT;
extern PFNGLCOPYNAMEDBUFFERSUBDATAPROC glCopyNamedBufferSubData;
extern PFNGLCOPYPATHNVPROC glCopyPathNV;
extern PFNGLCOPYTEXIMAGE1DPROC glCopyTexImage1D;
extern PFNGLCOPYTEXIMAGE2DPROC glCopyTexImage2D;
extern PFNGLCOPYTEXSUBIMAGE1DPROC glCopyTexSubImage1D;
extern PFNGLCOPYTEXSUBIMAGE2DPROC glCopyTexSubImage2D;
extern PFNGLCOPYTEXSUBIMAGE3DPROC glCopyTexSubImage3D;
extern PFNGLCOPYTEXTUREIMAGE1DEXTPROC glCopyTextureImage1DEXT;
extern PFNGLCOPYTEXTUREIMAGE2DEXTPROC glCopyTextureImage2DEXT;
extern PFNGLCOPYTEXTURESUBIMAGE1DEXTPROC glCopyTextureSubImage1DEXT;
extern PFNGLCOPYTEXTURESUBIMAGE1DPROC glCopyTextureSubImage1D;
extern PFNGLCOPYTEXTURESUBIMAGE2DEXTPROC glCopyTextureSubImage2DEXT;
extern PFNGLCOPYTEXTURESUBIMAGE2DPROC glCopyTextureSubImage2D;
extern PFNGLCOPYTEXTURESUBIMAGE3DEXTPROC glCopyTextureSubImage3DEXT;
extern PFNGLCOPYTEXTURESUBIMAGE3DPROC glCopyTextureSubImage3D;
extern PFNGLCOVERAGEMODULATIONNVPROC glCoverageModulationNV;
extern PFNGLCOVERAGEMODULATIONTABLENVPROC glCoverageModulationTableNV;
extern PFNGLCOVERFILLPATHINSTANCEDNVPROC glCoverFillPathInstancedNV;
extern PFNGLCOVERFILLPATHNVPROC glCoverFillPathNV;
extern PFNGLCOVERSTROKEPATHINSTANCEDNVPROC glCoverStrokePathInstancedNV;
extern PFNGLCOVERSTROKEPATHNVPROC glCoverStrokePathNV;
extern PFNGLCREATEBUFFERSPROC glCreateBuffers;
extern PFNGLCREATECOMMANDLISTSNVPROC glCreateCommandListsNV;
extern PFNGLCREATEFRAMEBUFFERSPROC glCreateFramebuffers;
extern PFNGLCREATEPERFQUERYINTELPROC glCreatePerfQueryINTEL;
extern PFNGLCREATEPROGRAMPIPELINESPROC glCreateProgramPipelines;
extern PFNGLCREATEPROGRAMPROC glCreateProgram;
extern PFNGLCREATEQUERIESPROC glCreateQueries;
extern PFNGLCREATERENDERBUFFERSPROC glCreateRenderbuffers;
extern PFNGLCREATESAMPLERSPROC glCreateSamplers;
extern PFNGLCREATESHADERPROC glCreateShader;
```

```
extern PFNGLCREATESHADERPROGRAMEXTPROC glCreateShaderProgramEXT;
extern PFNGLCREATESHADERPROGRAMVPROC glCreateShaderProgramv;
extern PFNGLCREATESTATESNVPROC glCreateStatesNV;
extern PFNGLCREATESYNCFROMCLEVENTARBPROC glCreateSyncFromCLeventARB;
extern PFNGLCREATETEXTURESPROC glCreateTextures;
extern PFNGLCREATETRANSFORMFEEDBACKSPROC glCreateTransformFeedbacks;
extern PFNGLCREATEVERTEXARRAYSPROC glCreateVertexArrays;
extern PFNGLCULLFACEPROC glCullFace;
extern PFNGLDEBUGMESSAGECALLBACKARBPROC glDebugMessageCallbackARB;
extern PFNGLDEBUGMESSAGECALLBACKPROC glDebugMessageCallback;
extern PFNGLDEBUGMESSAGECONTROLARBPROC glDebugMessageControlARB;
extern PFNGLDEBUGMESSAGECONTROLPROC glDebugMessageControl;
extern PFNGLDEBUGMESSAGEINSERTARBPROC glDebugMessageInsertARB;
extern PFNGLDEBUGMESSAGEINSERTPROC glDebugMessageInsert;
extern PFNGLDELETEBUFFERSPROC glDeleteBuffers;
extern PFNGLDELETECOMMANDLISTSNVPROC glDeleteCommandListsNV;
extern PFNGLDELETEFRAMEBUFFERSPROC glDeleteFramebuffers;
extern PFNGLDELETENAMEDSTRINGARBPROC glDeleteNamedStringARB;
extern PFNGLDELETEPATHSNVPROC glDeletePathsNV;
extern PFNGLDELETEPERFMONITORSAMDPROC glDeletePerfMonitorsAMD;
extern PFNGLDELETEPERFQUERYINTELPROC glDeletePerfQueryINTEL;
extern PFNGLDELETEPROGRAMPIPELINESPROC glDeleteProgramPipelines;
extern PFNGLDELETEPROGRAMPROC glDeleteProgram;
extern PFNGLDELETEQUERIESPROC glDeleteQueries;
extern PFNGLDELETERENDERBUFFERSPROC glDeleteRenderbuffers;
extern PFNGLDELETESAMPLERSPROC glDeleteSamplers;
extern PFNGLDELETESHADERPROC glDeleteShader;
extern PFNGLDELETESTATESNVPROC glDeleteStatesNV;
extern PFNGLDELETESYNCPROC glDeleteSync;
extern PFNGLDELETETEXTURESPROC glDeleteTextures;
extern PFNGLDELETETRANSFORMFEEDBACKSPROC glDeleteTransformFeedbacks;
extern PFNGLDELETEVERTEXARRAYSPROC glDeleteVertexArrays;
extern PFNGLDEPTHFUNCPROC glDepthFunc;
extern PFNGLDEPTHMASKPROC glDepthMask;
extern PFNGLDEPTHRANGEARRAYVPROC glDepthRangeArrayv;
extern PFNGLDEPTHRANGEFPROC glDepthRangef;
extern PFNGLDEPTHRANGEINDEXEDPROC glDepthRangeIndexed;
extern PFNGLDEPTHRANGEPROC glDepthRange;
extern PFNGLDETACHSHADERPROC glDetachShader;
extern PFNGLDISABLECLIENTSTATEIEXTPROC glDisableClientStateiEXT;
extern PFNGLDISABLECLIENTSTATEINDEXEDEXTPROC glDisableClientStateIndexedEXT;
extern PFNGLDISABLEINDEXEDEXTPROC glDisableIndexedEXT;
extern PFNGLDISABLEIPROC glDisablei;
extern PFNGLDISABLEPROC glDisable;
extern PFNGLDISABLEVERTEXARRAYATTRIBEXTPROC glDisableVertexArrayAttribEXT;
extern PFNGLDISABLEVERTEXARRAYATTRIBPROC glDisableVertexArrayAttrib;
extern PFNGLDISABLEVERTEXARRAYEXTPROC glDisableVertexArrayEXT;
extern PFNGLDISABLEVERTEXATTRIBARRAYPROC glDisableVertexAttribArray;
extern PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC glDispatchComputeGroupSizeARB;
extern PFNGLDISPATCHCOMPUTEINDIRECTPROC glDispatchComputeIndirect;
extern PFNGLDISPATCHCOMPUTEPROC glDispatchCompute;
extern PFNGLDRAWARRAYSINDIRECTPROC glDrawArraysIndirect;
extern PFNGLDRAWARRAYSINSTANCEDARBPROC glDrawArraysInstancedARB;
extern PFNGLDRAWARRAYSINSTANCEDBASEINSTANCEPROC glDrawArraysInstancedBaseInstance;
extern PFNGLDRAWARRAYSINSTANCEDEXTPROC glDrawArraysInstancedEXT;
extern PFNGLDRAWARRAYSINSTANCEDPROC glDrawArraysInstanced;
extern PFNGLDRAWARRAYSPROC glDrawArrays;
extern PFNGLDRAWBUFFERPROC glDrawBuffer;
extern PFNGLDRAWBUFFERSPROC glDrawBuffers;
extern PFNGLDRAWCOMMANDSADDRESSNVPROC glDrawCommandsAddressNV;
extern PFNGLDRAWCOMMANDSNVPROC glDrawCommandsNV;
extern PFNGLDRAWCOMMANDSSTATESADDRESSNVPROC glDrawCommandsStatesAddressNV;
extern PFNGLDRAWCOMMANDSSTATESNVPROC glDrawCommandsStatesNV;
extern PFNGLDRAWELEMENTSBASEVERTEXPROC glDrawElementsBaseVertex;
extern PFNGLDRAWELEMENTSINDIRECTPROC glDrawElementsIndirect;
extern PFNGLDRAWELEMENTSINSTANCEDARBPROC glDrawElementsInstancedARB;
extern PFNGLDRAWELEMENTSINSTANCEDBASEINSTANCEPROC glDrawElementsInstancedBaseInstance;
extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXBASEINSTANCEPROC
       glDrawElementsInstancedBaseVertexBaseInstance;
extern PFNGLDRAWELEMENTSINSTANCEDBASEVERTEXPROC glDrawElementsInstancedBaseVertex;
extern PFNGLDRAWELEMENTSINSTANCEDEXTPROC glDrawElementsInstancedEXT;
extern PFNGLDRAWELEMENTSINSTANCEDPROC glDrawElementsInstanced;
extern PFNGLDRAWELEMENTSPROC glDrawElements;
extern PFNGLDRAWRANGEELEMENTSBASEVERTEXPROC glDrawRangeElementsBaseVertex;
extern PFNGLDRAWRANGEELEMENTSPROC glDrawRangeElements;
extern PFNGLDRAWTRANSFORMFEEDBACKINSTANCEDPROC glDrawTransformFeedbackInstanced;
extern PFNGLDRAWTRANSFORMFEEDBACKPROC glDrawTransformFeedback;
extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMINSTANCEDPROC glDrawTransformFeedbackStreamInstanced;
extern PFNGLDRAWTRANSFORMFEEDBACKSTREAMPROC glDrawTransformFeedbackStream;
extern PFNGLDRAWVKIMAGENVPROC glDrawVkImageNV;
extern PFNGLEDGEFLAGFORMATNVPROC glEdgeFlagFormatNV;
extern PFNGLENABLECLIENTSTATEIEXTPROC glEnableClientStateiEXT;
extern PFNGLENABLECLIENTSTATEINDEXEDEXTPROC glEnableClientStateIndexedEXT;
extern PFNGLENABLEINDEXEDEXTPROC glEnableIndexedEXT;
extern PFNGLENABLEIPROC glEnablei;
extern PFNGLENABLEPROC glEnable;
extern PFNGLENABLEVERTEXARRAYATTRIBEXTPROC glEnableVertexArrayAttribEXT;
```

```
extern PFNGLENABLEVERTEXARRAYATTRIBPROC glEnableVertexArrayAttrib;
extern PFNGLENABLEVERTEXARRAYEXTPROC glEnableVertexArrayEXT;
extern PFNGLENABLEVERTEXATTRIBARRAYPROC glEnableVertexAttribArray;
extern PFNGLENDCONDITIONALRENDERNVPROC glEndConditionalRenderNV;
extern PFNGLENDCONDITIONALRENDERPROC glEndConditionalRender;
extern PFNGLENDPERFMONITORAMDPROC glEndPerfMonitorAMD;
extern PFNGLENDPERFQUERYINTELPROC glEndPerfQueryINTEL;
extern PFNGLENDQUERYINDEXEDPROC glEndQueryIndexed;
extern PFNGLENDQUERYPROC glEndQuery;
extern PFNGLENDTRANSFORMFEEDBACKPROC glEndTransformFeedback;
extern PFNGLEVALUATEDEPTHVALUESARBPROC glEvaluateDepthValuesARB;
extern PFNGLFENCESYNCPROC glFenceSync;
extern PFNGLFINISHPROC glFinish;
extern PFNGLFLUSHMAPPEDBUFFERRANGEPROC glFlushMappedBufferRange;
extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEEXTPROC glFlushMappedNamedBufferRangeEXT;
extern PFNGLFLUSHMAPPEDNAMEDBUFFERRANGEPROC glFlushMappedNamedBufferRange;
extern PFNGLFLUSHPROC glFlush;
extern PFNGLFOGCOORDFORMATNVPROC glFogCoordFormatNV;
extern PFNGLFRAGMENTCOVERAGECOLORNVPROC glFragmentCoverageColorNV;
extern PFNGLFRAMEBUFFERDRAWBUFFEREXTPROC glFramebufferDrawBufferEXT;
extern PFNGLFRAMEBUFFERDRAWBUFFERSEXTPROC glFramebufferDrawBuffersEXT;
extern PFNGLFRAMEBUFFERPARAMETERIPROC glFramebufferParameteri;
extern PFNGLFRAMEBUFFERREADBUFFEREXTPROC glFramebufferReadBufferEXT;
extern PFNGLFRAMEBUFFERRENDERBUFFERPROC glFramebufferRenderbuffer;
extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glFramebufferSampleLocationsfvARB;
extern PFNGLFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glFramebufferSampleLocationsfvNV;
extern PFNGLFRAMEBUFFERTEXTURE1DPROC glFramebufferTexture1D;
extern PFNGLFRAMEBUFFERTEXTURE2DPROC glFramebufferTexture2D;
extern PFNGLFRAMEBUFFERTEXTURE3DPROC glFramebufferTexture3D;
extern PFNGLFRAMEBUFFERTEXTUREARBPROC glFramebufferTextureARB;
extern PFNGLFRAMEBUFFERTEXTUREFACEARBPROC glFramebufferTextureFaceARB;
extern PFNGLFRAMEBUFFERTEXTURELAYERARBPROC glFramebufferTextureLayerARB;
extern PFNGLFRAMEBUFFERTEXTURELAYERPROC glFramebufferTextureLayer;
extern PFNGLFRAMEBUFFERTEXTUREMULTIVIEWOVRPROC glFramebufferTextureMultiviewOVR;
extern PFNGLFRAMEBUFFERTEXTUREPROC glFramebufferTexture;
extern PFNGLFRONTFACEPROC glFrontFace;
extern PFNGLGENBUFFERSPROC glGenBuffers;
extern PFNGLGENERATEMIPMAPPROC glGenerateMipmap;
extern PFNGLGENERATEMULTITEXMIPMAPEXTPROC glGenerateMultiTexMipmapEXT;
extern PFNGLGENERATETEXTUREMIPMAPEXTPROC glGenerateTextureMipmapEXT;
extern PFNGLGENERATETEXTUREMIPMAPPROC glGenerateTextureMipmap;
extern PFNGLGENFRAMEBUFFERSPROC glGenFramebuffers;
extern PFNGLGENPATHSNVPROC glGenPathsNV;
extern PFNGLGENPERFMONITORSAMDPROC glGenPerfMonitorsAMD;
extern PFNGLGENPROGRAMPIPELINESPROC glGenProgramPipelines;
extern PFNGLGENQUERIESPROC glGenQueries;
extern PFNGLGENRENDERBUFFERSPROC glGenRenderbuffers;
extern PFNGLGENSAMPLERSPROC glGenSamplers;
extern PFNGLGENTEXTURESPROC glGenTextures;
extern PFNGLGENTRANSFORMFEEDBACKSPROC glGenTransformFeedbacks;
extern PFNGLGENVERTEXARRAYSPROC glGenVertexArrays;
extern PFNGLGETACTIVEATOMICCOUNTERBUFFERIVPROC glGetActiveAtomicCounterBufferiv;
extern PFNGLGETACTIVEATTRIBPROC glGetActiveAttrib;
extern PFNGLGETACTIVESUBROUTINENAMEPROC glGetActiveSubroutineName;
extern PFNGLGETACTIVESUBROUTINEUNIFORMIVPROC glGetActiveSubroutineUniformiv;
extern PFNGLGETACTIVESUBROUTINEUNIFORMNAMEPROC glGetActiveSubroutineUniformName;
extern PFNGLGETACTIVEUNIFORMBLOCKIVPROC glGetActiveUniformBlockiv;
extern PFNGLGETACTIVEUNIFORMBLOCKNAMEPROC glGetActiveUniformBlockName;
extern PFNGLGETACTIVEUNIFORMNAMEPROC glGetActiveUniformName;
extern PFNGLGETACTIVEUNIFORMPROC glGetActiveUniform;
extern PFNGLGETACTIVEUNIFORMSIVPROC glGetActiveUniformsiv;
extern PFNGLGETATTACHEDSHADERSPROC glGetAttachedShaders;
extern PFNGLGETATTRIBLOCATIONPROC glGetAttribLocation;
extern PFNGLGETBOOLEANINDEXEDVEXTPROC glGetBooleanIndexedvEXT;
extern PFNGLGETBOOLEANI_VPROC glGetBooleani_v;
extern PFNGLGETBOOLEANVPROC glGetBooleanv;
extern PFNGLGETBUFFERPARAMETERI64VPROC glGetBufferParameteri64v;
extern PFNGLGETBUFFERPARAMETERIVPROC glGetBufferParameteriv;
extern PFNGLGETBUFFERPARAMETERUI64VNVPROC glGetBufferParameterui64vNV;
extern PFNGLGETBUFFERPOINTERVPROC glGetBufferPointerv;
extern PFNGLGETBUFFERSUBDATAPROC glGetBufferSubData;
extern PFNGLGETCOMMANDHEADERNVPROC glGetCommandHeaderNV;
extern PFNGLGETCOMPRESSEDMULTITEXIMAGEEXTPROC glGetCompressedMultiTexImageEXT;
extern PFNGLGETCOMPRESSEDTEXIMAGEPROC glGetCompressedTexImage;
extern PFNGLGETCOMPRESSEDTEXTUREIMAGEEXTPROC glGetCompressedTextureImageEXT;
extern PFNGLGETCOMPRESSEDTEXTUREIMAGEPROC glGetCompressedTextureImage;
extern PFNGLGETCOMPRESSEDTEXTURESUBIMAGEPROC glGetCompressedTextureSubImage;
extern PFNGLGETCOVERAGEMODULATIONTABLENVPROC glGetCoverageModulationTableNV;
extern PFNGLGETDEBUGMESSAGELOGARBPROC glGetDebugMessageLogARB;
extern PFNGLGETDEBUGMESSAGELOGPROC glGetDebugMessageLog;
extern PFNGLGETDOUBLEINDEXEDVEXTPROC glGetDoubleIndexedvEXT;
extern PFNGLGETDOUBLEI_VEXTPROC glGetDoublei_vEXT;
extern PFNGLGETDOUBLEI_VPROC glGetDoublei_v;
extern PFNGLGETDOUBLEVPROC glGetDoublev;
extern PFNGLGETERRORPROC glGetError;
extern PFNGLGETFIRSTPERFQUERYIDINTELPROC glGetFirstPerfQueryIdINTEL;
extern PFNGLGETFLOATINDEXEDVEXTPROC glGetFloatIndexedvEXT;
```

```
extern PFNGLGETFLOATI_VEXTPROC glGetFloati_vEXT;
extern PFNGLGETFLOATI_VPROC glGetFloati_v;
extern PFNGLGETFLOATVPROC glGetFloatv;
extern PFNGLGETFRAGDATAINDEXPROC glGetFragDataIndex;
extern PFNGLGETFRAGDATALOCATIONPROC glGetFragDataLocation;
extern PFNGLGETFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetFramebufferAttachmentParameteriv;
extern PFNGLGETFRAMEBUFFERPARAMETERIVEXTPROC glGetFramebufferParameterivEXT;
extern PFNGLGETFRAMEBUFFERPARAMETERIVPROC glGetFramebufferParameteriv;
extern PFNGLGETGRAPHICSRESETSTATUSARBPROC glGetGraphicsResetStatusARB;
extern PFNGLGETGRAPHICSRESETSTATUSPROC glGetGraphicsResetStatus;
extern PFNGLGETIMAGEHANDLEARBPROC glGetImageHandleARB;
extern PFNGLGETIMAGEHANDLENVPROC glGetImageHandleNV;
extern PFNGLGETINTEGER64I_VPROC glGetInteger64i_v;
extern PFNGLGETINTEGER64VPROC glGetInteger64v;
extern PFNGLGETINTEGERINDEXEDVEXTPROC glGetIntegerIndexedvEXT;
extern PFNGLGETINTEGERI_VPROC glGetIntegeri_v;
extern PFNGLGETINTEGERUI64I_VNVPROC glGetIntegerui64i_vNV;
extern PFNGLGETINTEGERUI64VNVPROC glGetIntegerui64vNV;
extern PFNGLGETINTEGERVPROC glGetIntegerv;
extern PFNGLGETINTERNALFORMATI64VPROC glGetInternalformati64v;
extern PFNGLGETINTERNALFORMATIVPROC glGetInternalformativ;
extern PFNGLGETINTERNALFORMATSAMPLEIVNVPROC glGetInternalformatSampleivNV;
extern PFNGLGETMULTISAMPLEFVPROC glGetMultisamplefv;
extern PFNGLGETMULTITEXENVFVEXTPROC glGetMultiTexEnvfvEXT;
extern PFNGLGETMULTITEXENVIVEXTPROC glGetMultiTexEnvivEXT;
extern PFNGLGETMULTITEXGENDVEXTPROC glGetMultiTexGendvEXT;
extern PFNGLGETMULTITEXGENFVEXTPROC glGetMultiTexGenfvEXT;
extern PFNGLGETMULTITEXGENIVEXTPROC glGetMultiTexGenivEXT;
extern PFNGLGETMULTITEXIMAGEEXTPROC glGetMultiTexImageEXT;
extern PFNGLGETMULTITEXLEVELPARAMETERFVEXTPROC glGetMultiTexLevelParameterfvEXT;
extern PFNGLGETMULTITEXLEVELPARAMETERIVEXTPROC glGetMultiTexLevelParameterivEXT;
extern PFNGLGETMULTITEXPARAMETERFVEXTPROC glGetMultiTexParameterfvEXT;
extern PFNGLGETMULTITEXPARAMETERIIVEXTPROC glGetMultiTexParameterIivEXT;
extern PFNGLGETMULTITEXPARAMETERIUIVEXTPROC glGetMultiTexParameterIuivEXT;
extern PFNGLGETMULTITEXPARAMETERIVEXTPROC glGetMultiTexParameterivEXT;
extern PFNGLGETNAMEDBUFFERPARAMETERI64VPROC glGetNamedBufferParameteri64v;
extern PFNGLGETNAMEDBUFFERPARAMETERIVEXTPROC glGetNamedBufferParameterivEXT;
extern PFNGLGETNAMEDBUFFERPARAMETERIVPROC glGetNamedBufferParameteriv;
extern PFNGLGETNAMEDBUFFERPARAMETERUI64VNVPROC glGetNamedBufferParameterui64vNV;
extern PFNGLGETNAMEDBUFFERPOINTERVEXTPROC glGetNamedBufferPointervEXT;
extern PFNGLGETNAMEDBUFFERPOINTERVPROC glGetNamedBufferPointerv;
extern PFNGLGETNAMEDBUFFERSUBDATAEXTPROC glGetNamedBufferSubDataEXT;
extern PFNGLGETNAMEDBUFFERSUBDATAPROC glGetNamedBufferSubData;
extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVEXTPROC
       glGetNamedFramebufferAttachmentParameterivEXT;
extern PFNGLGETNAMEDFRAMEBUFFERATTACHMENTPARAMETERIVPROC glGetNamedFramebufferAttachmentParameteriv;
extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVEXTPROC glGetNamedFramebufferParameterivEXT;
extern PFNGLGETNAMEDFRAMEBUFFERPARAMETERIVPROC glGetNamedFramebufferParameteriv;
extern PFNGLGETNAMEDPROGRAMIVEXTPROC glGetNamedProgramivEXT;
extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERDVEXTPROC glGetNamedProgramLocalParameterdvEXT;
extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERFVEXTPROC glGetNamedProgramLocalParameterfvEXT;
extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIIVEXTPROC glGetNamedProgramLocalParameterIivEXT;
extern PFNGLGETNAMEDPROGRAMLOCALPARAMETERIUIVEXTPROC glGetNamedProgramLocalParameterIuivEXT;
extern PFNGLGETNAMEDPROGRAMSTRINGEXTPROC glGetNamedProgramStringEXT;
extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVEXTPROC glGetNamedRenderbufferParameterivEXT;
extern PFNGLGETNAMEDRENDERBUFFERPARAMETERIVPROC glGetNamedRenderbufferParameteriv;
extern PFNGLGETNAMEDSTRINGARBPROC glGetNamedStringARB;
extern PFNGLGETNAMEDSTRINGIVARBPROC glGetNamedStringivARB;
extern PFNGLGETNCOMPRESSEDTEXIMAGEARBPROC glGetnCompressedTexImageARB;
extern PFNGLGETNCOMPRESSEDTEXIMAGEPROC glGetnCompressedTexImage;
extern PFNGLGETNEXTPERFQUERYIDINTELPROC glGetNextPerfQueryIdINTEL;
extern PFNGLGETNTEXIMAGEARBPROC glGetnTexImageARB;
extern PFNGLGETNTEXIMAGEPROC glGetnTexImage;
extern PFNGLGETNUNIFORMDVARBPROC glGetnUniformdvARB;
extern PFNGLGETNUNIFORMDVPROC glGetnUniformdv;
extern PFNGLGETNUNIFORMFVARBPROC glGetnUniformfvARB;
extern PFNGLGETNUNIFORMFVPROC glGetnUniformfv;
extern PFNGLGETNUNIFORMI64VARBPROC glGetnUniformi64vARB;
extern PFNGLGETNUNIFORMIVARBPROC glGetnUniformivARB;
extern PFNGLGETNUNIFORMIVPROC glGetnUniformiv;
extern PFNGLGETNUNIFORMUI64VARBPROC glGetnUniformui64vARB;
extern PFNGLGETNUNIFORMUIVARBPROC glGetnUniformuivARB;
extern PFNGLGETNUNIFORMUIVPROC glGetnUniformuiv;
extern PFNGLGETOBJECTLABELEXTPROC glGetObjectLabelEXT;
extern PFNGLGETOBJECTLABELPROC glGetObjectLabel;
extern PFNGLGETOBJECTPTRLABELPROC glGetObjectPtrLabel;
extern PFNGLGETPATHCOMMANDSNVPROC glGetPathCommandsNV;
extern PFNGLGETPATHCOORDSNVPROC glGetPathCoordsNV;
extern PFNGLGETPATHDASHARRAYNVPROC glGetPathDashArrayNV;
extern PFNGLGETPATHLENGTHNVPROC glGetPathLengthNV;
extern PFNGLGETPATHMETRICRANGENVPROC glGetPathMetricRangeNV;
extern PFNGLGETPATHMETRICSNVPROC glGetPathMetricsNV;
extern PFNGLGETPATHPARAMETERFVNVPROC glGetPathParameterfvNV;
extern PFNGLGETPATHPARAMETERIVNVPROC glGetPathParameterivNV;
extern PFNGLGETPATHSPACINGNVPROC glGetPathSpacingNV;
extern PFNGLGETPERFCOUNTERINFOINTELPROC glGetPerfCounterInfoINTEL;
extern PFNGLGETPERFMONITORCOUNTERDATAAMDPROC glGetPerfMonitorCounterDataAMD;
```

```
extern PFNGLGETPERFMONITORCOUNTERINFOAMDPROC glGetPerfMonitorCounterInfoAMD;
extern PFNGLGETPERFMONITORCOUNTERSAMDPROC glGetPerfMonitorCountersAMD;
extern PFNGLGETPERFMONITORCOUNTERSTRINGAMDPROC glGetPerfMonitorCounterStringAMD;
extern PFNGLGETPERFMONITORGROUPSAMDPROC glGetPerfMonitorGroupsAMD;
extern PFNGLGETPERFMONITORGROUPSTRINGAMDPROC glGetPerfMonitorGroupStringAMD;
extern PFNGLGETPERFQUERYDATAINTELPROC glGetPerfQueryDataINTEL;
extern PFNGLGETPERFQUERYIDBYNAMEINTELPROC glGetPerfQueryIdByNameINTEL;
extern PFNGLGETPERFQUERYINFOINTELPROC glGetPerfQueryInfoINTEL;
extern PFNGLGETPOINTERINDEXEDVEXTPROC glGetPointerIndexedvEXT;
extern PFNGLGETPOINTERI_VEXTPROC glGetPointeri_vEXT;
extern PFNGLGETPOINTERVPROC glGetPointerv;
extern PFNGLGETPROGRAMBINARYPROC glGetProgramBinary;
extern PFNGLGETPROGRAMINFOLOGPROC glGetProgramInfoLog;
extern PFNGLGETPROGRAMINTERFACEIVPROC glGetProgramInterfaceiv;
extern PFNGLGETPROGRAMIVPROC glGetProgramiv;
extern PFNGLGETPROGRAMPIPELINEINFOLOGPROC glGetProgramPipelineInfoLog;
extern PFNGLGETPROGRAMPIPELINEIVPROC glGetProgramPipelineiv;
extern PFNGLGETPROGRAMRESOURCEFVNVPROC glGetProgramResourcefvNV;
extern PFNGLGETPROGRAMRESOURCEINDEXPROC glGetProgramResourceIndex;
extern PFNGLGETPROGRAMRESOURCEIVPROC glGetProgramResourceiv;
extern PFNGLGETPROGRAMRESOURCELOCATIONINDEXPROC glGetProgramResourceLocationIndex;
extern PFNGLGETPROGRAMRESOURCELOCATIONPROC glGetProgramResourceLocation;
extern PFNGLGETPROGRAMRESOURCENAMEPROC glGetProgramResourceName;
extern PFNGLGETPROGRAMSTAGEIVPROC glGetProgramStageiv;
extern PFNGLGETQUERYBUFFEROBJECTI64VPROC glGetQueryBufferObjecti64v;
extern PFNGLGETQUERYBUFFEROBJECTIVPROC glGetQueryBufferObjectiv;
extern PFNGLGETQUERYBUFFEROBJECTUI64VPROC glGetQueryBufferObjectui64v;
extern PFNGLGETQUERYBUFFEROBJECTUIVPROC glGetQueryBufferObjectuiv;
extern PFNGLGETQUERYINDEXEDIVPROC glGetQueryIndexediv;
extern PFNGLGETQUERYIVPROC glGetQueryiv;
extern PFNGLGETQUERYOBJECTI64VPROC glGetQueryObjecti64v;
extern PFNGLGETQUERYOBJECTIVPROC glGetQueryObjectiv;
extern PFNGLGETQUERYOBJECTUI64VPROC glGetQueryObjectui64v;
extern PFNGLGETQUERYOBJECTUIVPROC glGetQueryObjectuiv;
extern PFNGLGETRENDERBUFFERPARAMETERIVPROC glGetRenderbufferParameteriv;
extern PFNGLGETSAMPLERPARAMETERFVPROC glGetSamplerParameterfv;
extern PFNGLGETSAMPLERPARAMETERIIVPROC glGetSamplerParameterIiv;
extern PFNGLGETSAMPLERPARAMETERIUIVPROC glGetSamplerParameterIuiv;
extern PFNGLGETSAMPLERPARAMETERIVPROC glGetSamplerParameteriv;
extern PFNGLGETSHADERINFOLOGPROC glGetShaderInfoLog;
extern PFNGLGETSHADERIVPROC glGetShaderiv;
extern PFNGLGETSHADERPRECISIONFORMATPROC glGetShaderPrecisionFormat;
extern PFNGLGETSHADERSOURCEPROC glGetShaderSource;
extern PFNGLGETSTAGEINDEXNVPROC glGetStageIndexNV;
extern PFNGLGETSTRINGIPROC glGetStringi;
extern PFNGLGETSTRINGPROC glGetString;
extern PFNGLGETSUBROUTINEINDEXPROC glGetSubroutineIndex;
extern PFNGLGETSUBROUTINEUNIFORMLOCATIONPROC glGetSubroutineUniformLocation;
extern PFNGLGETSYNCIVPROC glGetSynciv;
extern PFNGLGETTEXIMAGEPROC glGetTexImage;
extern PFNGLGETTEXLEVELPARAMETERFVPROC glGetTexLevelParameterfv;
extern PFNGLGETTEXLEVELPARAMETERIVPROC glGetTexLevelParameteriv;
extern PFNGLGETTEXPARAMETERFVPROC glGetTexParameterfv;
extern PFNGLGETTEXPARAMETERIIVPROC glGetTexParameterIiv;
extern PFNGLGETTEXPARAMETERIUIVPROC glGetTexParameterIuiv;
extern PFNGLGETTEXPARAMETERIVPROC glGetTexParameteriv;
extern PFNGLGETTEXTUREHANDLEARBPROC glGetTextureHandleARB;
extern PFNGLGETTEXTUREHANDLENVPROC glGetTextureHandleNV;
extern PFNGLGETTEXTUREIMAGEEXTPROC glGetTextureImageEXT;
extern PFNGLGETTEXTUREIMAGEPROC glGetTextureImage;
extern PFNGLGETTEXTURELEVELPARAMETERFVEXTPROC glGetTextureLevelParameterfvEXT;
extern PFNGLGETTEXTURELEVELPARAMETERFVPROC glGetTextureLevelParameterfv;
extern PFNGLGETTEXTURELEVELPARAMETERIVEXTPROC glGetTextureLevelParameterivEXT;
extern PFNGLGETTEXTURELEVELPARAMETERIVPROC glGetTextureLevelParameteriv;
extern PFNGLGETTEXTUREPARAMETERFVEXTPROC glGetTextureParameterfvEXT;
extern PFNGLGETTEXTUREPARAMETERFVPROC glGetTextureParameterfv;
extern PFNGLGETTEXTUREPARAMETERIIVEXTPROC glGetTextureParameterIivEXT;
extern PFNGLGETTEXTUREPARAMETERIIVPROC glGetTextureParameterIiv;
extern PFNGLGETTEXTUREPARAMETERIUIVEXTPROC glGetTextureParameterIuivEXT;
extern PFNGLGETTEXTUREPARAMETERIUIVPROC glGetTextureParameterIuiv;
extern PFNGLGETTEXTUREPARAMETERIVEXTPROC glGetTextureParameterivEXT;
extern PFNGLGETTEXTUREPARAMETERIVPROC glGetTextureParameteriv;
extern PFNGLGETTEXTURESAMPLERHANDLEARBPROC glGetTextureSamplerHandleARB;
extern PFNGLGETTEXTURESAMPLERHANDLENVPROC glGetTextureSamplerHandleNV;
extern PFNGLGETTEXTURESUBIMAGEPROC glGetTextureSubImage;
extern PFNGLGETTRANSFORMFEEDBACKI64_VPROC glGetTransformFeedbacki64_v;
extern PFNGLGETTRANSFORMFEEDBACKIVPROC glGetTransformFeedbackiv;
extern PFNGLGETTRANSFORMFEEDBACKI_VPROC glGetTransformFeedbacki_v;
extern PFNGLGETTRANSFORMFEEDBACKVARYINGPROC glGetTransformFeedbackVarying;
extern PFNGLGETUNIFORMBLOCKINDEXPROC glGetUniformBlockIndex;
extern PFNGLGETUNIFORMDVPROC glGetUniformdv;
extern PFNGLGETUNIFORMFVPROC glGetUniformfv;
extern PFNGLGETUNIFORMI64VARBPROC glGetUniformi64vARB;
extern PFNGLGETUNIFORMI64VNVPROC glGetUniformi64vNV;
extern PFNGLGETUNIFORMINDICESPROC glGetUniformIndices;
extern PFNGLGETUNIFORMIVPROC glGetUniformiv;
extern PFNGLGETUNIFORMLOCATIONPROC glGetUniformLocation;
```

```
extern PFNGLGETUNIFORMSUBROUTINEUIVPROC glGetUniformSubroutineuiv;
extern PFNGLGETUNIFORMUI64VARBPROC glGetUniformui64vARB;
extern PFNGLGETUNIFORMUI64VNVPROC glGetUniformui64vNV;
extern PFNGLGETUNIFORMUIVPROC glGetUniformuiv;
extern PFNGLGETVERTEXARRAYINDEXED64IVPROC glGetVertexArrayIndexed64iv;
extern PFNGLGETVERTEXARRAYINDEXEDIVPROC glGetVertexArrayIndexediv;
extern PFNGLGETVERTEXARRAYINTEGERI_VEXTPROC glGetVertexArrayIntegeri_vEXT;
extern PFNGLGETVERTEXARRAYINTEGERVEXTPROC glGetVertexArrayIntegervEXT;
extern PFNGLGETVERTEXARRAYIVPROC glGetVertexArrayiv;
extern PFNGLGETVERTEXARRAYPOINTERI_VEXTPROC glGetVertexArrayPointeri_vEXT;
extern PFNGLGETVERTEXARRAYPOINTERVEXTPROC glGetVertexArrayPointervEXT;
extern PFNGLGETVERTEXATTRIBDVPROC glGetVertexAttribdv;
extern PFNGLGETVERTEXATTRIBFVPROC glGetVertexAttribfv;
extern PFNGLGETVERTEXATTRIBIIVPROC glGetVertexAttribIiv;
extern PFNGLGETVERTEXATTRIBIUIVPROC glGetVertexAttribIuiv;
extern PFNGLGETVERTEXATTRIBIVPROC glGetVertexAttribiv;
extern PFNGLGETVERTEXATTRIBLDVPROC glGetVertexAttribLdv;
extern PFNGLGETVERTEXATTRIBLI64VNVPROC glGetVertexAttribLi64vNV;
extern PFNGLGETVERTEXATTRIBLUI64VARBPROC glGetVertexAttribLui64vARB;
extern PFNGLGETVERTEXATTRIBLUI64VNVPROC glGetVertexAttribLui64vNV;
extern PFNGLGETVERTEXATTRIBPOINTERVPROC glGetVertexAttribPointerv;
extern PFNGLGETVKPROCADDRNVPROC glGetVkProcAddrNV;
extern PFNGLHINTPROC glHint;
extern PFNGLINDEXFORMATNVPROC glIndexFormatNV;
extern PFNGLINSERTEVENTMARKEREXTPROC glInsertEventMarkerEXT;
extern PFNGLINTERPOLATEPATHSNVPROC glInterpolatePathsNV;
extern PFNGLINVALIDATEBUFFERDATAPROC glInvalidateBufferData;
extern PFNGLINVALIDATEBUFFERSUBDATAPROC glInvalidateBufferSubData;
extern PFNGLINVALIDATEFRAMEBUFFERPROC glInvalidateFramebuffer;
extern PFNGLINVALIDATENAMEDFRAMEBUFFERDATAPROC glInvalidateNamedFramebufferData;
extern PFNGLINVALIDATENAMEDFRAMEBUFFERSUBDATAPROC glInvalidateNamedFramebufferSubData;
extern PFNGLINVALIDATESUBFRAMEBUFFERPROC glInvalidateSubFramebuffer;
extern PFNGLINVALIDATETEXIMAGEPROC glInvalidateTexImage;
extern PFNGLINVALIDATETEXSUBIMAGEPROC glInvalidateTexSubImage;
extern PFNGLISBUFFERPROC glIsBuffer;
extern PFNGLISBUFFERRESIDENTNVPROC glIsBufferResidentNV;
extern PFNGLISCOMMANDLISTNVPROC glIsCommandListNV;
extern PFNGLISENABLEDINDEXEDEXTPROC glIsEnabledIndexedEXT;
extern PFNGLISENABLEDIPROC glIsEnabledi;
extern PFNGLISENABLEDPROC glIsEnabled;
extern PFNGLISFRAMEBUFFERPROC glIsFramebuffer;
extern PFNGLISIMAGEHANDLERESIDENTARBPROC glIsImageHandleResidentARB;
extern PFNGLISIMAGEHANDLERESIDENTNVPROC glIsImageHandleResidentNV;
extern PFNGLISNAMEDBUFFERRESIDENTNVPROC glIsNamedBufferResidentNV;
extern PFNGLISNAMEDSTRINGARBPROC glIsNamedStringARB;
extern PFNGLISPATHNVPROC glIsPathNV;
extern PFNGLISPOINTINFILLPATHNVPROC glIsPointInFillPathNV;
extern PFNGLISPOINTINSTROKEPATHNVPROC glIsPointInStrokePathNV;
extern PFNGLISPROGRAMPIPELINEPROC glIsProgramPipeline;
extern PFNGLISPROGRAMPROC glIsProgram;
extern PFNGLISQUERYPROC glIsQuery;
extern PFNGLISRENDERBUFFERPROC glIsRenderbuffer;
extern PFNGLISSAMPLERPROC glIsSampler;
extern PFNGLISSHADERPROC glIsShader;
extern PFNGLISSTATENVPROC glIsStateNV;
extern PFNGLISSYNCPROC glIsSync;
extern PFNGLISTEXTUREHANDLERESIDENTARBPROC glIsTextureHandleResidentARB;
extern PFNGLISTEXTUREHANDLERESIDENTNVPROC glIsTextureHandleResidentNV;
extern PFNGLISTEXTUREPROC glIsTexture;
extern PFNGLISTRANSFORMFEEDBACKPROC glIsTransformFeedback;
extern PFNGLISVERTEXARRAYPROC glIsVertexArray;
extern PFNGLLABELOBJECTEXTPROC glLabelObjectEXT;
extern PFNGLLINEWIDTHPROC glLineWidth;
extern PFNGLLINKPROGRAMPROC glLinkProgram;
extern PFNGLLISTDRAWCOMMANDSSTATESCLIENTNVPROC glListDrawCommandsStatesClientNV;
extern PFNGLLOGICOPPROC glLogicOp;
extern PFNGLMAKEBUFFERNONRESIDENTNVPROC glMakeBufferNonResidentNV;
extern PFNGLMAKEBUFFERRESIDENTNVPROC glMakeBufferResidentNV;
extern PFNGLMAKEIMAGEHANDLENONRESIDENTARBPROC glMakeImageHandleNonResidentARB;
extern PFNGLMAKEIMAGEHANDLENONRESIDENTNVPROC glMakeImageHandleNonResidentNV;
extern PFNGLMAKEIMAGEHANDLERESIDENTARBPROC glMakeImageHandleResidentARB;
extern PFNGLMAKEIMAGEHANDLERESIDENTNVPROC glMakeImageHandleResidentNV;
extern PFNGLMAKENAMEDBUFFERNONRESIDENTNVPROC glMakeNamedBufferNonResidentNV;
extern PFNGLMAKENAMEDBUFFERRESIDENTNVPROC glMakeNamedBufferResidentNV;
extern PFNGLMAKETEXTUREHANDLENONRESIDENTARBPROC glMakeTextureHandleNonResidentARB;
extern PFNGLMAKETEXTUREHANDLENONRESIDENTNVPROC glMakeTextureHandleNonResidentNV;
extern PFNGLMAKETEXTUREHANDLERESIDENTARBPROC glMakeTextureHandleResidentARB;
extern PFNGLMAKETEXTUREHANDLERESIDENTNVPROC glMakeTextureHandleResidentNV;
extern PFNGLMAPBUFFERPROC glMapBuffer;
extern PFNGLMAPBUFFERRANGEPROC glMapBufferRange;
extern PFNGLMAPNAMEDBUFFEREXTPROC glMapNamedBufferEXT;
extern PFNGLMAPNAMEDBUFFERPROC glMapNamedBuffer;
extern PFNGLMAPNAMEDBUFFERRANGEEXTPROC glMapNamedBufferRangeEXT;
extern PFNGLMAPNAMEDBUFFERRANGEPROC glMapNamedBufferRange;
extern PFNGLMATRIXFRUSTUMEXTPROC glMatrixFrustumEXT;
extern PFNGLMATRIXLOAD3X2FNVPROC glMatrixLoad3x2fNV;
extern PFNGLMATRIXLOAD3X3FNVPROC glMatrixLoad3x3fNV;
```

```
extern PFNGLMATRIXLOADDEXTPROC glMatrixLoaddEXT;
extern PFNGLMATRIXLOADFEXTPROC glMatrixLoadfEXT;
extern PFNGLMATRIXLOADIDENTITYEXTPROC glMatrixLoadIdentityEXT;
extern PFNGLMATRIXLOADTRANSPOSE3X3FNVPROC glMatrixLoadTranspose3x3fNV;
extern PFNGLMATRIXLOADTRANSPOSEDEXTPROC glMatrixLoadTransposedEXT;
extern PFNGLMATRIXLOADTRANSPOSEFEXTPROC glMatrixLoadTransposefEXT;
extern PFNGLMATRIXMULT3X2FNVPROC glMatrixMult3x2fNV;
extern PFNGLMATRIXMULT3X3FNVPROC glMatrixMult3x3fNV;
extern PFNGLMATRIXMULTDEXTPROC glMatrixMultdEXT;
extern PFNGLMATRIXMULTFEXTPROC glMatrixMultfEXT;
extern PFNGLMATRIXMULTTRANSPOSE3X3FNVPROC glMatrixMultTranspose3x3fNV;
extern PFNGLMATRIXMULTTRANSPOSEDEXTPROC glMatrixMultTransposedEXT;
extern PFNGLMATRIXMULTTRANSPOSEFEXTPROC glMatrixMultTransposefEXT;
extern PFNGLMATRIXORTHOEXTPROC glMatrixOrthoEXT;
extern PFNGLMATRIXPOPEXTPROC glMatrixPopEXT;
extern PFNGLMATRIXPUSHEXTPROC glMatrixPushEXT;
extern PFNGLMATRIXROTATEDEXTPROC glMatrixRotatedEXT;
extern PFNGLMATRIXROTATEFEXTPROC glMatrixRotatefEXT;
extern PFNGLMATRIXSCALEDEXTPROC glMatrixScaledEXT;
extern PFNGLMATRIXSCALEFEXTPROC glMatrixScalefEXT;
extern PFNGLMATRIXTRANSLATEDEXTPROC glMatrixTranslatedEXT;
extern PFNGLMATRIXTRANSLATEFEXTPROC glMatrixTranslatefEXT;
extern PFNGLMAXSHADERCOMPILERTHREADSARBPROC glMaxShaderCompilerThreadsARB;
extern PFNGLMEMORYBARRIERBYREGIONPROC glMemoryBarrierByRegion;
extern PFNGLMEMORYBARRIERPROC glMemoryBarrier;
extern PFNGLMINSAMPLESHADINGARBPROC glMinSampleShadingARB;
extern PFNGLMINSAMPLESHADINGPROC glMinSampleShading;
extern PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawArraysIndirectBindlessCountNV;
extern PFNGLMULTIDRAWARRAYSINDIRECTBINDLESSNVPROC glMultiDrawArraysIndirectBindlessNV;
extern PFNGLMULTIDRAWARRAYSINDIRECTCOUNTARBPROC glMultiDrawArraysIndirectCountARB;
extern PFNGLMULTIDRAWARRAYSINDIRECTPROC glMultiDrawArraysIndirect;
extern PFNGLMULTIDRAWARRAYSPROC glMultiDrawArrays;
extern PFNGLMULTIDRAWELEMENTSBASEVERTEXPROC glMultiDrawElementsBaseVertex;
extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSCOUNTNVPROC glMultiDrawElementsIndirectBindlessCountNV;
extern PFNGLMULTIDRAWELEMENTSINDIRECTBINDLESSNVPROC glMultiDrawElementsIndirectBindlessNV;
extern PFNGLMULTIDRAWELEMENTSINDIRECTCOUNTARBPROC glMultiDrawElementsIndirectCountARB;
extern PFNGLMULTIDRAWELEMENTSINDIRECTPROC glMultiDrawElementsIndirect;
extern PFNGLMULTIDRAWELEMENTSPROC glMultiDrawElements;
extern PFNGLMULTITEXBUFFEREXTPROC glMultiTexBufferEXT;
extern PFNGLMULTITEXCOORDPOINTEREXTPROC glMultiTexCoordPointerEXT;
extern PFNGLMULTITEXENVFEXTPROC glMultiTexEnvfEXT;
extern PFNGLMULTITEXENVFVEXTPROC glMultiTexEnvfvEXT;
extern PFNGLMULTITEXENVIEXTPROC glMultiTexEnviEXT;
extern PFNGLMULTITEXENVIVEXTPROC glMultiTexEnvivEXT;
extern PFNGLMULTITEXGENDEXTPROC glMultiTexGendEXT;
extern PFNGLMULTITEXGENDVEXTPROC glMultiTexGendvEXT;
extern PFNGLMULTITEXGENFEXTPROC glMultiTexGenfEXT;
extern PFNGLMULTITEXGENFVEXTPROC glMultiTexGenfvEXT;
extern PFNGLMULTITEXGENIEXTPROC glMultiTexGeniEXT;
extern PFNGLMULTITEXGENIVEXTPROC glMultiTexGenivEXT;
extern PFNGLMULTITEXIMAGE1DEXTPROC glMultiTexImage1DEXT;
extern PFNGLMULTITEXIMAGE2DEXTPROC glMultiTexImage2DEXT;
extern PFNGLMULTITEXIMAGE3DEXTPROC glMultiTexImage3DEXT;
extern PFNGLMULTITEXPARAMETERFEXTPROC glMultiTexParameterfEXT;
extern PFNGLMULTITEXPARAMETERFVEXTPROC glMultiTexParameterfvEXT;
extern PFNGLMULTITEXPARAMETERIEXTPROC glMultiTexParameteriEXT;
extern PFNGLMULTITEXPARAMETERIIVEXTPROC glMultiTexParameterIivEXT;
extern PFNGLMULTITEXPARAMETERIUIVEXTPROC glMultiTexParameterIuivEXT;
extern PFNGLMULTITEXPARAMETERIVEXTPROC glMultiTexParameterivEXT;
extern PFNGLMULTITEXRENDERBUFFEREXTPROC glMultiTexRenderbufferEXT;
extern PFNGLMULTITEXSUBIMAGE1DEXTPROC glMultiTexSubImage1DEXT;
extern PFNGLMULTITEXSUBIMAGE2DEXTPROC glMultiTexSubImage2DEXT;
extern PFNGLMULTITEXSUBIMAGE3DEXTPROC glMultiTexSubImage3DEXT;
extern PFNGLNAMEDBUFFERDATAEXTPROC glNamedBufferDataEXT;
extern PFNGLNAMEDBUFFERDATAPROC glNamedBufferData;
extern PFNGLNAMEDBUFFERPAGECOMMITMENTARBPROC glNamedBufferPageCommitmentARB;
extern PFNGLNAMEDBUFFERPAGECOMMITMENTEXTPROC glNamedBufferPageCommitmentEXT;
extern PFNGLNAMEDBUFFERSTORAGEEXTPROC glNamedBufferStorageEXT;
extern PFNGLNAMEDBUFFERSTORAGEPROC glNamedBufferStorage;
extern PFNGLNAMEDBUFFERSUBDATAEXTPROC glNamedBufferSubDataEXT;
extern PFNGLNAMEDBUFFERSUBDATAPROC glNamedBufferSubData;
extern PFNGLNAMEDCOPYBUFFERSUBDATAEXTPROC glNamedCopyBufferSubDataEXT;
extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERPROC glNamedFramebufferDrawBuffer;
extern PFNGLNAMEDFRAMEBUFFERDRAWBUFFERSPROC glNamedFramebufferDrawBuffers;
extern PFNGLNAMEDFRAMEBUFFERPARAMETERIEXTPROC glNamedFramebufferParameteriEXT;
extern PFNGLNAMEDFRAMEBUFFERPARAMETERIPROC glNamedFramebufferParameteri;
extern PFNGLNAMEDFRAMEBUFFERREADBUFFERPROC glNamedFramebufferReadBuffer;
extern PFNGLNAMEDFRAMEBUFFERRENDERBUFFEREXTPROC glNamedFramebufferRenderbufferEXT;
extern PFNGLNAMEDFRAMEBUFFERRENDERBUFFERPROC glNamedFramebufferRenderbuffer;
extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVARBPROC glNamedFramebufferSampleLocationsfvARB;
extern PFNGLNAMEDFRAMEBUFFERSAMPLELOCATIONSFVNVPROC glNamedFramebufferSampleLocationsfvNV;
extern PFNGLNAMEDFRAMEBUFFERTEXTURE1DEXTPROC glNamedFramebufferTexture1DEXT;
extern PFNGLNAMEDFRAMEBUFFERTEXTURE2DEXTPROC glNamedFramebufferTexture2DEXT;
extern PFNGLNAMEDFRAMEBUFFERTEXTURE3DEXTPROC glNamedFramebufferTexture3DEXT;
extern PFNGLNAMEDFRAMEBUFFERTEXTUREEXTPROC glNamedFramebufferTextureEXT;
extern PFNGLNAMEDFRAMEBUFFERTEXTUREFACEEXTPROC glNamedFramebufferTextureFaceEXT;
extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYEREXTPROC glNamedFramebufferTextureLayerEXT;
```

```
extern PFNGLNAMEDFRAMEBUFFERTEXTURELAYERPROC glNamedFramebufferTextureLayer;
extern PFNGLNAMEDFRAMEBUFFERTEXTUREPROC glNamedFramebufferTexture;
extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DEXTPROC glNamedProgramLocalParameter4dEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETER4DVEXTPROC glNamedProgramLocalParameter4dvEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FEXTPROC glNamedProgramLocalParameter4fEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETER4FVEXTPROC glNamedProgramLocalParameter4fvEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IEXTPROC glNamedProgramLocalParameterI4iEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4IVEXTPROC glNamedProgramLocalParameterI4ivEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIEXTPROC glNamedProgramLocalParameterI4uiEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERI4UIVEXTPROC glNamedProgramLocalParameterI4uivEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERS4FVEXTPROC glNamedProgramLocalParameters4fvEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4IVEXTPROC glNamedProgramLocalParametersI4ivEXT;
extern PFNGLNAMEDPROGRAMLOCALPARAMETERSI4UIVEXTPROC glNamedProgramLocalParametersI4uivEXT;
extern PFNGLNAMEDPROGRAMSTRINGEXTPROC glNamedProgramStringEXT;
extern PFNGLNAMEDRENDERBUFFERSTORAGEEXTPROC glNamedRenderbufferStorageEXT;
extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLECOVERAGEEXTPROC
       glNamedRenderbufferStorageMultisampleCoverageEXT;
extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEEXTPROC glNamedRenderbufferStorageMultisampleEXT;
extern PFNGLNAMEDRENDERBUFFERSTORAGEMULTISAMPLEPROC glNamedRenderbufferStorageMultisample;
extern PFNGLNAMEDRENDERBUFFERSTORAGEPROC glNamedRenderbufferStorage;
extern PFNGLNAMEDSTRINGARBPROC glNamedStringARB;
extern PFNGLNORMALFORMATNVPROC glNormalFormatNV;
extern PFNGLOBJECTLABELPROC glObjectLabel;
extern PFNGLOBJECTPTRLABELPROC glObjectPtrLabel;
extern PFNGLPATCHPARAMETERFVPROC glPatchParameterfv;
extern PFNGLPATCHPARAMETERIPROC glPatchParameteri;
extern PFNGLPATHCOMMANDSNVPROC glPathCommandsNV;
extern PFNGLPATHCOORDSNVPROC glPathCoordsNV;
extern PFNGLPATHCOVERDEPTHFUNCNVPROC glPathCoverDepthFuncNV;
extern PFNGLPATHDASHARRAYNVPROC glPathDashArrayNV;
extern PFNGLPATHGLYPHINDEXARRAYNVPROC glPathGlyphIndexArrayNV;
extern PFNGLPATHGLYPHINDEXRANGENVPROC glPathGlyphIndexRangeNV;
extern PFNGLPATHGLYPHRANGENVPROC glPathGlyphRangeNV;
extern PFNGLPATHGLYPHSNVPROC glPathGlyphsNV;
extern PFNGLPATHMEMORYGLYPHINDEXARRAYNVPROC glPathMemoryGlyphIndexArrayNV;
extern PFNGLPATHPARAMETERFNVPROC glPathParameterfNV;
extern PFNGLPATHPARAMETERFVNVPROC glPathParameterfvNV;
extern PFNGLPATHPARAMETERINVPROC glPathParameteriNV;
extern PFNGLPATHPARAMETERIVNVPROC glPathParameterivNV;
extern PFNGLPATHSTENCILDEPTHOFFSETNVPROC glPathStencilDepthOffsetNV;
extern PFNGLPATHSTENCILFUNCNVPROC glPathStencilFuncNV;
extern PFNGLPATHSTRINGNVPROC glPathStringNV;
extern PFNGLPATHSUBCOMMANDSNVPROC glPathSubCommandsNV;
extern PFNGLPATHSUBCOORDSNVPROC glPathSubCoordsNV;
extern PFNGLPAUSETRANSFORMFEEDBACKPROC glPauseTransformFeedback;
extern PFNGLPIXELSTOREFPROC glPixelStoref;
extern PFNGLPIXELSTOREIPROC glPixelStorei;
extern PFNGLPOINTALONGPATHNVPROC glPointAlongPathNV;
extern PFNGLPOINTPARAMETERFPROC glPointParameterf;
extern PFNGLPOINTPARAMETERFVPROC glPointParameterfv;
extern PFNGLPOINTPARAMETERIPROC glPointParameteri;
extern PFNGLPOINTPARAMETERIVPROC glPointParameteriv;
extern PFNGLPOINTSIZEPROC glPointSize;
extern PFNGLPOLYGONMODEPROC glPolygonMode;
extern PFNGLPOLYGONOFFSETCLAMPEXTPROC glPolygonOffsetClampEXT;
extern PFNGLPOLYGONOFFSETPROC glPolygonOffset;
extern PFNGLPOPDEBUGGROUPPROC glPopDebugGroup;
extern PFNGLPOPGROUPMARKEREXTPROC glPopGroupMarkerEXT;
extern PFNGLPRIMITIVEBOUNDINGBOXARBPROC glPrimitiveBoundingBoxARB;
extern PFNGLPRIMITIVERESTARTINDEXPROC glPrimitiveRestartIndex;
extern PFNGLPROGRAMBINARYPROC glProgramBinary;
extern PFNGLPROGRAMPARAMETERIARBPROC glProgramParameteriARB;
extern PFNGLPROGRAMPARAMETERIPROC glProgramParameteri;
extern PFNGLPROGRAMPATHFRAGMENTINPUTGENNVPROC glProgramPathFragmentInputGenNV;
extern PFNGLPROGRAMUNIFORM1DEXTPROC glProgramUniform1dEXT;
extern PFNGLPROGRAMUNIFORM1DPROC glProgramUniform1d;
extern PFNGLPROGRAMUNIFORM1DVEXTPROC glProgramUniform1dvEXT;
extern PFNGLPROGRAMUNIFORM1DVPROC glProgramUniform1dv;
extern PFNGLPROGRAMUNIFORM1FEXTPROC glProgramUniform1fEXT;
extern PFNGLPROGRAMUNIFORM1FPROC glProgramUniform1f;
extern PFNGLPROGRAMUNIFORM1FVEXTPROC glProgramUniform1fvEXT;
extern PFNGLPROGRAMUNIFORM1FVPROC glProgramUniform1fv;
extern PFNGLPROGRAMUNIFORM1I64ARBPROC glProgramUniform1i64ARB;
extern PFNGLPROGRAMUNIFORM1I64NVPROC glProgramUniform1i64NV;
extern PFNGLPROGRAMUNIFORM1I64VARBPROC glProgramUniform1i64vARB;
extern PFNGLPROGRAMUNIFORM1I64VNVPROC glProgramUniform1i64vNV;
extern PFNGLPROGRAMUNIFORM1IEXTPROC glProgramUniform1iEXT;
extern PFNGLPROGRAMUNIFORM1IPROC glProgramUniform1i;
extern PFNGLPROGRAMUNIFORM1IVEXTPROC glProgramUniform1ivEXT;
extern PFNGLPROGRAMUNIFORM1IVPROC glProgramUniform1iv;
extern PFNGLPROGRAMUNIFORM1UI64ARBPROC glProgramUniform1ui64ARB;
extern PFNGLPROGRAMUNIFORM1UI64NVPROC glProgramUniform1ui64NV;
extern PFNGLPROGRAMUNIFORM1UI64VARBPROC glProgramUniform1ui64vARB;
extern PFNGLPROGRAMUNIFORM1UI64VNVPROC glProgramUniform1ui64vNV;
extern PFNGLPROGRAMUNIFORM1UIEXTPROC glProgramUniform1uiEXT;
extern PFNGLPROGRAMUNIFORM1UIPROC glProgramUniform1ui;
extern PFNGLPROGRAMUNIFORM1UIVEXTPROC glProgramUniform1uivEXT;
```

```
extern PFNGLPROGRAMUNIFORM1UIVPROC glProgramUniform1uiv;
extern PFNGLPROGRAMUNIFORM2DEXTPROC glProgramUniform2dEXT;
extern PFNGLPROGRAMUNIFORM2DPROC glProgramUniform2d;
extern PFNGLPROGRAMUNIFORM2DVEXTPROC glProgramUniform2dvEXT;
extern PFNGLPROGRAMUNIFORM2DVPROC glProgramUniform2dv;
extern PFNGLPROGRAMUNIFORM2FEXTPROC glProgramUniform2fEXT;
extern PFNGLPROGRAMUNIFORM2FPROC glProgramUniform2f;
extern PFNGLPROGRAMUNIFORM2FVEXTPROC glProgramUniform2fvEXT;
extern PFNGLPROGRAMUNIFORM2FVPROC glProgramUniform2fv;
extern PFNGLPROGRAMUNIFORM2I64ARBPROC glProgramUniform2i64ARB;
extern PFNGLPROGRAMUNIFORM2I64NVPROC glProgramUniform2i64NV;
extern PFNGLPROGRAMUNIFORM2I64VARBPROC glProgramUniform2i64vARB;
extern PFNGLPROGRAMUNIFORM2I64VNVPROC glProgramUniform2i64vNV;
extern PFNGLPROGRAMUNIFORM2IEXTPROC glProgramUniform2iEXT;
extern PFNGLPROGRAMUNIFORM2IPROC glProgramUniform2i;
extern PFNGLPROGRAMUNIFORM2IVEXTPROC glProgramUniform2ivEXT;
extern PFNGLPROGRAMUNIFORM2IVPROC glProgramUniform2iv;
extern PFNGLPROGRAMUNIFORM2UI64ARBPROC glProgramUniform2ui64ARB;
extern PFNGLPROGRAMUNIFORM2UI64NVPROC glProgramUniform2ui64NV;
extern PFNGLPROGRAMUNIFORM2UI64VARBPROC glProgramUniform2ui64vARB;
extern PFNGLPROGRAMUNIFORM2UI64VNVPROC glProgramUniform2ui64vNV;
extern PFNGLPROGRAMUNIFORM2UIEXTPROC glProgramUniform2uiEXT;
extern PFNGLPROGRAMUNIFORM2UIPROC glProgramUniform2ui;
extern PFNGLPROGRAMUNIFORM2UIVEXTPROC glProgramUniform2uivEXT;
extern PFNGLPROGRAMUNIFORM2UIVPROC glProgramUniform2uiv;
extern PFNGLPROGRAMUNIFORM3DEXTPROC glProgramUniform3dEXT;
extern PFNGLPROGRAMUNIFORM3DPROC glProgramUniform3d;
extern PFNGLPROGRAMUNIFORM3DVEXTPROC glProgramUniform3dvEXT;
extern PFNGLPROGRAMUNIFORM3DVPROC glProgramUniform3dv;
extern PFNGLPROGRAMUNIFORM3FEXTPROC glProgramUniform3fEXT;
extern PFNGLPROGRAMUNIFORM3FPROC glProgramUniform3f;
extern PFNGLPROGRAMUNIFORM3FVEXTPROC glProgramUniform3fvEXT;
extern PFNGLPROGRAMUNIFORM3FVPROC glProgramUniform3fv;
extern PFNGLPROGRAMUNIFORM3I64ARBPROC glProgramUniform3i64ARB;
extern PFNGLPROGRAMUNIFORM3I64NVPROC glProgramUniform3i64NV;
extern PFNGLPROGRAMUNIFORM3I64VARBPROC glProgramUniform3i64vARB;
extern PFNGLPROGRAMUNIFORM3I64VNVPROC glProgramUniform3i64vNV;
extern PFNGLPROGRAMUNIFORM3IEXTPROC glProgramUniform3iEXT;
extern PFNGLPROGRAMUNIFORM3IPROC glProgramUniform3i;
extern PFNGLPROGRAMUNIFORM3IVEXTPROC glProgramUniform3ivEXT;
extern PFNGLPROGRAMUNIFORM3IVPROC glProgramUniform3iv;
extern PFNGLPROGRAMUNIFORM3UI64ARBPROC glProgramUniform3ui64ARB;
extern PFNGLPROGRAMUNIFORM3UI64NVPROC glProgramUniform3ui64NV;
extern PFNGLPROGRAMUNIFORM3UI64VARBPROC glProgramUniform3ui64vARB;
extern PFNGLPROGRAMUNIFORM3UI64VNVPROC glProgramUniform3ui64vNV;
extern PFNGLPROGRAMUNIFORM3UIEXTPROC glProgramUniform3uiEXT;
extern PFNGLPROGRAMUNIFORM3UIPROC glProgramUniform3ui;
extern PFNGLPROGRAMUNIFORM3UIVEXTPROC glProgramUniform3uivEXT;
extern PFNGLPROGRAMUNIFORM3UIVPROC glProgramUniform3uiv;
extern PFNGLPROGRAMUNIFORM4DEXTPROC glProgramUniform4dEXT;
extern PFNGLPROGRAMUNIFORM4DPROC glProgramUniform4d;
extern PFNGLPROGRAMUNIFORM4DVEXTPROC glProgramUniform4dvEXT;
extern PFNGLPROGRAMUNIFORM4DVPROC glProgramUniform4dv;
extern PFNGLPROGRAMUNIFORM4FEXTPROC glProgramUniform4fEXT;
extern PFNGLPROGRAMUNIFORM4FPROC glProgramUniform4f;
extern PFNGLPROGRAMUNIFORM4FVEXTPROC glProgramUniform4fvEXT;
extern PFNGLPROGRAMUNIFORM4FVPROC glProgramUniform4fv;
extern PFNGLPROGRAMUNIFORM4I64ARBPROC glProgramUniform4i64ARB;
extern PFNGLPROGRAMUNIFORM4I64NVPROC glProgramUniform4i64NV;
extern PFNGLPROGRAMUNIFORM4I64VARBPROC glProgramUniform4i64vARB;
extern PFNGLPROGRAMUNIFORM4I64VNVPROC glProgramUniform4i64vNV;
extern PFNGLPROGRAMUNIFORM4IEXTPROC glProgramUniform4iEXT;
extern PFNGLPROGRAMUNIFORM4IPROC glProgramUniform4i;
extern PFNGLPROGRAMUNIFORM4IVEXTPROC glProgramUniform4ivEXT;
extern PFNGLPROGRAMUNIFORM4IVPROC glProgramUniform4iv;
extern PFNGLPROGRAMUNIFORM4UI64ARBPROC glProgramUniform4ui64ARB;
extern PFNGLPROGRAMUNIFORM4UI64NVPROC glProgramUniform4ui64NV;
extern PFNGLPROGRAMUNIFORM4UI64VARBPROC glProgramUniform4ui64vARB;
extern PFNGLPROGRAMUNIFORM4UI64VNVPROC glProgramUniform4ui64vNV;
extern PFNGLPROGRAMUNIFORM4UIEXTPROC glProgramUniform4uiEXT;
extern PFNGLPROGRAMUNIFORM4UIPROC glProgramUniform4ui;
extern PFNGLPROGRAMUNIFORM4UIVEXTPROC glProgramUniform4uivEXT;
extern PFNGLPROGRAMUNIFORM4UIVPROC glProgramUniform4uiv;
extern PFNGLPROGRAMUNIFORMHANDLEUI64ARBPROC glProgramUniformHandleui64ARB;
extern PFNGLPROGRAMUNIFORMHANDLEUI64NVPROC glProgramUniformHandleui64NV;
extern PFNGLPROGRAMUNIFORMHANDLEUI64VARBPROC glProgramUniformHandleui64vARB;
extern PFNGLPROGRAMUNIFORMHANDLEUI64VNVPROC glProgramUniformHandleui64vNV;
extern PFNGLPROGRAMUNIFORMMATRIX2DVEXTPROC glProgramUniformMatrix2dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2DVPROC glProgramUniformMatrix2dv;
extern PFNGLPROGRAMUNIFORMMATRIX2FVEXTPROC glProgramUniformMatrix2fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2FVPROC glProgramUniformMatrix2fv;
extern PFNGLPROGRAMUNIFORMMATRIX2X3DVEXTPROC glProgramUniformMatrix2x3dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2X3DVPROC glProgramUniformMatrix2x3dv;
extern PFNGLPROGRAMUNIFORMMATRIX2X3FVEXTPROC glProgramUniformMatrix2x3fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2X3FVPROC glProgramUniformMatrix2x3fv;
extern PFNGLPROGRAMUNIFORMMATRIX2X4DVEXTPROC glProgramUniformMatrix2x4dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2X4DVPROC glProgramUniformMatrix2x4dv;
```

```
extern PFNGLPROGRAMUNIFORMMATRIX2X4FVEXTPROC glProgramUniformMatrix2x4fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX2X4FVPROC glProgramUniformMatrix2x4fv;
extern PFNGLPROGRAMUNIFORMMATRIX3DVEXTPROC glProgramUniformMatrix3dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3DVPROC glProgramUniformMatrix3dv;
extern PFNGLPROGRAMUNIFORMMATRIX3FVEXTPROC glProgramUniformMatrix3fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3FVPROC glProgramUniformMatrix3fv;
extern PFNGLPROGRAMUNIFORMMATRIX3X2DVEXTPROC glProgramUniformMatrix3x2dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3X2DVPROC glProgramUniformMatrix3x2dv;
extern PFNGLPROGRAMUNIFORMMATRIX3X2FVEXTPROC glProgramUniformMatrix3x2fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3X2FVPROC glProgramUniformMatrix3x2fv;
extern PFNGLPROGRAMUNIFORMMATRIX3X4DVEXTPROC glProgramUniformMatrix3x4dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3X4DVPROC glProgramUniformMatrix3x4dv;
extern PFNGLPROGRAMUNIFORMMATRIX3X4FVEXTPROC glProgramUniformMatrix3x4fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX3X4FVPROC glProgramUniformMatrix3x4fv;
extern PFNGLPROGRAMUNIFORMMATRIX4DVEXTPROC glProgramUniformMatrix4dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4DVPROC glProgramUniformMatrix4dv;
extern PFNGLPROGRAMUNIFORMMATRIX4FVEXTPROC glProgramUniformMatrix4fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4FVPROC glProgramUniformMatrix4fv;
extern PFNGLPROGRAMUNIFORMMATRIX4X2DVEXTPROC glProgramUniformMatrix4x2dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4X2DVPROC glProgramUniformMatrix4x2dv;
extern PFNGLPROGRAMUNIFORMMATRIX4X2FVEXTPROC glProgramUniformMatrix4x2fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4X2FVPROC glProgramUniformMatrix4x2fv;
extern PFNGLPROGRAMUNIFORMMATRIX4X3DVEXTPROC glProgramUniformMatrix4x3dvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4X3DVPROC glProgramUniformMatrix4x3dv;
extern PFNGLPROGRAMUNIFORMMATRIX4X3FVEXTPROC glProgramUniformMatrix4x3fvEXT;
extern PFNGLPROGRAMUNIFORMMATRIX4X3FVPROC glProgramUniformMatrix4x3fv;
extern PFNGLPROGRAMUNIFORMUI64NVPROC glProgramUniformui64NV;
extern PFNGLPROGRAMUNIFORMUI64VNVPROC glProgramUniformui64vNV;
extern PFNGLPROVOKINGVERTEXPROC glProvokingVertex;
extern PFNGLPUSHCLIENTATTRIBDEFAULTEXTPROC glPushClientAttribDefaultEXT;
extern PFNGLPUSHDEBUGGROUPPROC glPushDebugGroup;
extern PFNGLPUSHGROUPMARKEREXTPROC glPushGroupMarkerEXT;
extern PFNGLQUERYCOUNTERPROC glQueryCounter;
extern PFNGLRASTERSAMPLESEXTPROC glRasterSamplesEXT;
extern PFNGLREADBUFFERPROC glReadBuffer;
extern PFNGLREADNPIXELSARBPROC glReadnPixelsARB;
extern PFNGLREADNPIXELSPROC glReadnPixels;
extern PFNGLREADPIXELSPROC glReadPixels;
extern PFNGLRELEASESHADERCOMPILERPROC glReleaseShaderCompiler;
extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLECOVERAGENVPROC glRenderbufferStorageMultisampleCoverageNV;
extern PFNGLRENDERBUFFERSTORAGEMULTISAMPLEPROC glRenderbufferStorageMultisample;
extern PFNGLRENDERBUFFERSTORAGEPROC glRenderbufferStorage;
extern PFNGLRESOLVEDEPTHVALUESNVPROC glResolveDepthValuesNV;
extern PFNGLRESUMETRANSFORMFEEDBACKPROC glResumeTransformFeedback;
extern PFNGLSAMPLECOVERAGEPROC glSampleCoverage;
extern PFNGLSAMPLEMASKIPROC glSampleMaski;
extern PFNGLSAMPLERPARAMETERFPROC glSamplerParameterf;
extern PFNGLSAMPLERPARAMETERFVPROC glSamplerParameterfv;
extern PFNGLSAMPLERPARAMETERIIVPROC glSamplerParameterIiv;
extern PFNGLSAMPLERPARAMETERIPROC glSamplerParameteri;
extern PFNGLSAMPLERPARAMETERIUIVPROC glSamplerParameterIuiv;
extern PFNGLSAMPLERPARAMETERIVPROC glSamplerParameteriv;
extern PFNGLSCISSORARRAYVPROC glScissorArrayv;
extern PFNGLSCISSORINDEXEDPROC glScissorIndexed;
extern PFNGLSCISSORINDEXEDVPROC glScissorIndexedv;
extern PFNGLSCISSORPROC glScissor;
extern PFNGLSECONDARYCOLORFORMATNVPROC glSecondaryColorFormatNV;
extern PFNGLSELECTPERFMONITORCOUNTERSAMDPROC glSelectPerfMonitorCountersAMD;
extern PFNGLSHADERBINARYPROC glShaderBinary;
extern PFNGLSHADERSOURCEPROC glShaderSource;
extern PFNGLSHADERSTORAGEBLOCKBINDINGPROC glShaderStorageBlockBinding;
extern PFNGLSIGNALVKFENCENVPROC glSignalVkFenceNV;
extern PFNGLSIGNALVKSEMAPHORENVPROC glSignalVkSemaphoreNV;
extern PFNGLSPECIALIZESHADERARBPROC glSpecializeShaderARB;
extern PFNGLSTATECAPTURENVPROC glStateCaptureNV;
extern PFNGLSTENCILFILLPATHINSTANCEDNVPROC glStencilFillPathInstancedNV;
extern PFNGLSTENCILFILLPATHNVPROC glStencilFillPathNV;
extern PFNGLSTENCILFUNCPROC glStencilFunc;
extern PFNGLSTENCILFUNCSEPARATEPROC glStencilFuncSeparate;
extern PFNGLSTENCILMASKPROC glStencilMask;
extern PFNGLSTENCILMASKSEPARATEPROC glStencilMaskSeparate;
extern PFNGLSTENCILOPPROC glStencilOp;
extern PFNGLSTENCILOPSEPARATEPROC glStencilOpSeparate;
extern PFNGLSTENCILSTROKEPATHINSTANCEDNVPROC glStencilStrokePathInstancedNV;
extern PFNGLSTENCILSTROKEPATHNVPROC glStencilStrokePathNV;
extern PFNGLSTENCILTHENCOVERFILLPATHINSTANCEDNVPROC glStencilThenCoverFillPathInstancedNV;
extern PFNGLSTENCILTHENCOVERFILLPATHNVPROC glStencilThenCoverFillPathNV;
extern PFNGLSTENCILTHENCOVERSTROKEPATHINSTANCEDNVPROC glStencilThenCoverStrokePathInstancedNV;
extern PFNGLSTENCILTHENCOVERSTROKEPATHNVPROC glStencilThenCoverStrokePathNV;
extern PFNGLSUBPIXELPRECISIONBIASNVPROC glSubpixelPrecisionBiasNV;
extern PFNGLTEXBUFFERARBPROC glTexBufferARB;
extern PFNGLTEXBUFFERPROC glTexBuffer;
extern PFNGLTEXBUFFERRANGEPROC glTexBufferRange;
extern PFNGLTEXCOORDFORMATNVPROC glTexCoordFormatNV;
extern PFNGLTEXIMAGE1DPROC glTexImage1D;
extern PFNGLTEXIMAGE2DMULTISAMPLEPROC glTexImage2DMultisample;
extern PFNGLTEXIMAGE2DPROC glTexImage2D;
```

```
extern PFNGLTEXIMAGE3DMULTISAMPLEPROC glTexImage3DMultisample;
extern PFNGLTEXIMAGE3DPROC glTexImage3D;
extern PFNGLTEXPAGECOMMITMENTARBPROC glTexPageCommitmentARB;
extern PFNGLTEXPARAMETERFPROC glTexParameterf;
extern PFNGLTEXPARAMETERFVPROC glTexParameterfv;
extern PFNGLTEXPARAMETERIIVPROC glTexParameterIiv;
extern PFNGLTEXPARAMETERIPROC glTexParameteri;
extern PFNGLTEXPARAMETERIUIVPROC glTexParameterIuiv;
extern PFNGLTEXPARAMETERIVPROC glTexParameteriv;
extern PFNGLTEXSTORAGE1DPROC glTexStorage1D;
extern PFNGLTEXSTORAGE2DMULTISAMPLEPROC glTexStorage2DMultisample;
extern PFNGLTEXSTORAGE2DPROC glTexStorage2D;
extern PFNGLTEXSTORAGE3DMULTISAMPLEPROC glTexStorage3DMultisample;
extern PFNGLTEXSTORAGE3DPROC glTexStorage3D;
extern PFNGLTEXSUBIMAGE1DPROC glTexSubImage1D;
extern PFNGLTEXSUBIMAGE2DPROC glTexSubImage2D;
extern PFNGLTEXSUBIMAGE3DPROC glTexSubImage3D;
extern PFNGLTEXTUREBARRIERNVPROC glTextureBarrierNV;
extern PFNGLTEXTUREBARRIERPROC glTextureBarrier;
extern PFNGLTEXTUREBUFFEREXTPROC glTextureBufferEXT;
extern PFNGLTEXTUREBUFFERPROC glTextureBuffer;
extern PFNGLTEXTUREBUFFERRANGEEXTPROC glTextureBufferRangeEXT;
extern PFNGLTEXTUREBUFFERRANGEPROC glTextureBufferRange;
extern PFNGLTEXTUREIMAGE1DEXTPROC glTextureImage1DEXT;
extern PFNGLTEXTUREIMAGE2DEXTPROC glTextureImage2DEXT;
extern PFNGLTEXTUREIMAGE3DEXTPROC glTextureImage3DEXT;
extern PFNGLTEXTUREPAGECOMMITMENTEXTPROC glTexturePageCommitmentEXT;
extern PFNGLTEXTUREPARAMETERFEXTPROC glTextureParameterfEXT;
extern PFNGLTEXTUREPARAMETERFPROC glTextureParameterf;
extern PFNGLTEXTUREPARAMETERFVEXTPROC glTextureParameterfvEXT;
extern PFNGLTEXTUREPARAMETERFVPROC glTextureParameterfv;
extern PFNGLTEXTUREPARAMETERIEXTPROC glTextureParameteriEXT;
extern PFNGLTEXTUREPARAMETERIIVEXTPROC glTextureParameterIivEXT;
extern PFNGLTEXTUREPARAMETERIIVPROC glTextureParameterIiv;
extern PFNGLTEXTUREPARAMETERIPROC glTextureParameteri;
extern PFNGLTEXTUREPARAMETERIUIVEXTPROC glTextureParameterIuivEXT;
extern PFNGLTEXTUREPARAMETERIUIVPROC glTextureParameterIuiv;
extern PFNGLTEXTUREPARAMETERIVEXTPROC glTextureParameterivEXT;
extern PFNGLTEXTUREPARAMETERIVPROC glTextureParameteriv;
extern PFNGLTEXTURERENDERBUFFEREXTPROC glTextureRenderbufferEXT;
extern PFNGLTEXTURESTORAGE1DEXTPROC glTextureStorage1DEXT;
extern PFNGLTEXTURESTORAGE1DPROC glTextureStorage1D;
extern PFNGLTEXTURESTORAGE2DEXTPROC glTextureStorage2DEXT;
extern PFNGLTEXTURESTORAGE2DMULTISAMPLEEXTPROC glTextureStorage2DMultisampleEXT;
extern PFNGLTEXTURESTORAGE2DMULTISAMPLEPROC glTextureStorage2DMultisample;
extern PFNGLTEXTURESTORAGE2DPROC glTextureStorage2D;
extern PFNGLTEXTURESTORAGE3DEXTPROC glTextureStorage3DEXT;
extern PFNGLTEXTURESTORAGE3DMULTISAMPLEEXTPROC glTextureStorage3DMultisampleEXT;
extern PFNGLTEXTURESTORAGE3DMULTISAMPLEPROC glTextureStorage3DMultisample;
extern PFNGLTEXTURESTORAGE3DPROC glTextureStorage3D;
extern PFNGLTEXTURESUBIMAGE1DEXTPROC glTextureSubImage1DEXT;
extern PFNGLTEXTURESUBIMAGE1DPROC glTextureSubImage1D;
extern PFNGLTEXTURESUBIMAGE2DEXTPROC glTextureSubImage2DEXT;
extern PFNGLTEXTURESUBIMAGE2DPROC glTextureSubImage2D;
extern PFNGLTEXTURESUBIMAGE3DEXTPROC glTextureSubImage3DEXT;
extern PFNGLTEXTURESUBIMAGE3DPROC glTextureSubImage3D;
extern PFNGLTEXTUREVIEWPROC glTextureView;
extern PFNGLTRANSFORMFEEDBACKBUFFERBASEPROC glTransformFeedbackBufferBase;
extern PFNGLTRANSFORMFEEDBACKBUFFERRANGEPROC glTransformFeedbackBufferRange;
extern PFNGLTRANSFORMFEEDBACKVARYINGSPROC glTransformFeedbackVaryings;
extern PFNGLTRANSFORMPATHNVPROC glTransformPathNV;
extern PFNGLUNIFORM1DPROC glUniform1d;
extern PFNGLUNIFORM1DVPROC glUniform1dv;
extern PFNGLUNIFORM1FPROC glUniform1f;
extern PFNGLUNIFORM1FVPROC glUniform1fv;
extern PFNGLUNIFORM1I64ARBPROC glUniform1i64ARB;
extern PFNGLUNIFORM1I64NVPROC glUniform1i64NV;
extern PFNGLUNIFORM1I64VARBPROC glUniform1i64vARB;
extern PFNGLUNIFORM1I64VNVPROC glUniform1i64vNV;
extern PFNGLUNIFORM1IPROC glUniform1i;
extern PFNGLUNIFORM1IVPROC glUniform1iv;
extern PFNGLUNIFORM1UI64ARBPROC glUniform1ui64ARB;
extern PFNGLUNIFORM1UI64NVPROC glUniform1ui64NV;
extern PFNGLUNIFORM1UI64VARBPROC glUniform1ui64vARB;
extern PFNGLUNIFORM1UI64VNVPROC glUniform1ui64vNV;
extern PFNGLUNIFORM1UIPROC glUniform1ui;
extern PFNGLUNIFORM1UIVPROC glUniform1uiv;
extern PFNGLUNIFORM2DPROC glUniform2d;
extern PFNGLUNIFORM2DVPROC glUniform2dv;
extern PFNGLUNIFORM2FPROC glUniform2f;
extern PFNGLUNIFORM2FVPROC glUniform2fv;
extern PFNGLUNIFORM2I64ARBPROC glUniform2i64ARB;
extern PFNGLUNIFORM2I64NVPROC glUniform2i64NV;
extern PFNGLUNIFORM2I64VARBPROC glUniform2i64vARB;
extern PFNGLUNIFORM2I64VNVPROC glUniform2i64vNV;
extern PFNGLUNIFORM2IPROC glUniform2i;
extern PFNGLUNIFORM2IVPROC glUniform2iv;
```

```
extern PFNGLUNIFORM2UI64ARBPROC glUniform2ui64ARB;
extern PFNGLUNIFORM2UI64NVPROC glUniform2ui64NV;
extern PFNGLUNIFORM2UI64VARBPROC glUniform2ui64vARB;
extern PFNGLUNIFORM2UI64VNVPROC glUniform2ui64vNV;
extern PFNGLUNIFORM2UIPROC glUniform2ui;
extern PFNGLUNIFORM2UIVPROC glUniform2uiv;
extern PFNGLUNIFORM3DPROC glUniform3d;
extern PFNGLUNIFORM3DVPROC glUniform3dv;
extern PFNGLUNIFORM3FPROC glUniform3f;
extern PFNGLUNIFORM3FVPROC glUniform3fv;
extern PFNGLUNIFORM3I64ARBPROC glUniform3i64ARB;
extern PFNGLUNIFORM3I64NVPROC glUniform3i64NV;
extern PFNGLUNIFORM3I64VARBPROC glUniform3i64vARB;
extern PFNGLUNIFORM3I64VNVPROC glUniform3i64vNV;
extern PFNGLUNIFORM3IPROC glUniform3i;
extern PFNGLUNIFORM3IVPROC glUniform3iv;
extern PFNGLUNIFORM3UI64ARBPROC glUniform3ui64ARB;
extern PFNGLUNIFORM3UI64NVPROC glUniform3ui64NV;
extern PFNGLUNIFORM3UI64VARBPROC glUniform3ui64vARB;
extern PFNGLUNIFORM3UI64VNVPROC glUniform3ui64vNV;
extern PFNGLUNIFORM3UIPROC glUniform3ui;
extern PFNGLUNIFORM3UIVPROC glUniform3uiv;
extern PFNGLUNIFORM4DPROC glUniform4d;
extern PFNGLUNIFORM4DVPROC glUniform4dv;
extern PFNGLUNIFORM4FPROC glUniform4f;
extern PFNGLUNIFORM4FVPROC glUniform4fv;
extern PFNGLUNIFORM4I64ARBPROC glUniform4i64ARB;
extern PFNGLUNIFORM4I64NVPROC glUniform4i64NV;
extern PFNGLUNIFORM4I64VARBPROC glUniform4i64vARB;
extern PFNGLUNIFORM4I64VNVPROC glUniform4i64vNV;
extern PFNGLUNIFORM4IPROC glUniform4i;
extern PFNGLUNIFORM4IVPROC glUniform4iv;
extern PFNGLUNIFORM4UI64ARBPROC glUniform4ui64ARB;
extern PFNGLUNIFORM4UI64NVPROC glUniform4ui64NV;
extern PFNGLUNIFORM4UI64VARBPROC glUniform4ui64vARB;
extern PFNGLUNIFORM4UI64VNVPROC glUniform4ui64vNV;
extern PFNGLUNIFORM4UIPROC glUniform4ui;
extern PFNGLUNIFORM4UIVPROC glUniform4uiv;
extern PFNGLUNIFORMBLOCKBINDINGPROC glUniformBlockBinding;
extern PFNGLUNIFORMHANDLEUI64ARBPROC glUniformHandleui64ARB;
extern PFNGLUNIFORMHANDLEUI64NVPROC glUniformHandleui64NV;
extern PFNGLUNIFORMHANDLEUI64VARBPROC glUniformHandleui64vARB;
extern PFNGLUNIFORMHANDLEUI64VNVPROC glUniformHandleui64vNV;
extern PFNGLUNIFORMMATRIX2DVPROC glUniformMatrix2dv;
extern PFNGLUNIFORMMATRIX2FVPROC glUniformMatrix2fv;
extern PFNGLUNIFORMMATRIX2X3DVPROC glUniformMatrix2x3dv;
extern PFNGLUNIFORMMATRIX2X3FVPROC glUniformMatrix2x3fv;
extern PFNGLUNIFORMMATRIX2X4DVPROC glUniformMatrix2x4dv;
extern PFNGLUNIFORMMATRIX2X4FVPROC glUniformMatrix2x4fv;
extern PFNGLUNIFORMMATRIX3DVPROC glUniformMatrix3dv;
extern PFNGLUNIFORMMATRIX3FVPROC glUniformMatrix3fv;
extern PFNGLUNIFORMMATRIX3X2DVPROC glUniformMatrix3x2dv;
extern PFNGLUNIFORMMATRIX3X2FVPROC glUniformMatrix3x2fv;
extern PFNGLUNIFORMMATRIX3X4DVPROC glUniformMatrix3x4dv;
extern PFNGLUNIFORMMATRIX3X4FVPROC glUniformMatrix3x4fv;
extern PFNGLUNIFORMMATRIX4DVPROC glUniformMatrix4dv;
extern PFNGLUNIFORMMATRIX4FVPROC glUniformMatrix4fv;
extern PFNGLUNIFORMMATRIX4X2DVPROC glUniformMatrix4x2dv;
extern PFNGLUNIFORMMATRIX4X2FVPROC glUniformMatrix4x2fv;
extern PFNGLUNIFORMMATRIX4X3DVPROC glUniformMatrix4x3dv;
extern PFNGLUNIFORMMATRIX4X3FVPROC glUniformMatrix4x3fv;
extern PFNGLUNIFORMSUBROUTINESUIVPROC glUniformSubroutinesuiv;
extern PFNGLUNIFORMUI64NVPROC glUniformui64NV;
extern PFNGLUNIFORMUI64VNVPROC glUniformui64vNV;
extern PFNGLUNMAPBUFFERPROC glUnmapBuffer;
extern PFNGLUNMAPNAMEDBUFFEREXTPROC glUnmapNamedBufferEXT;
extern PFNGLUNMAPNAMEDBUFFERPROC glUnmapNamedBuffer;
extern PFNGLUSEPROGRAMPROC glUseProgram;
extern PFNGLUSEPROGRAMSTAGESPROC glUseProgramStages;
extern PFNGLUSESHADERPROGRAMEXTPROC glUseShaderProgramEXT;
extern PFNGLVALIDATEPROGRAMPIPELINEPROC glValidateProgramPipeline;
extern PFNGLVALIDATEPROGRAMPROC glValidateProgram;
extern PFNGLVERTEXARRAYATTRIBBINDINGPROC glVertexArrayAttribBinding;
extern PFNGLVERTEXARRAYATTRIBFORMATPROC glVertexArrayAttribFormat;
extern PFNGLVERTEXARRAYATTRIBIFORMATPROC glVertexArrayAttribIFormat;
extern PFNGLVERTEXARRAYATTRIBLFORMATPROC glVertexArrayAttribLFormat;
extern PFNGLVERTEXARRAYBINDINGDIVISORPROC glVertexArrayBindingDivisor;
extern PFNGLVERTEXARRAYBINDVERTEXBUFFEREXTPROC glVertexArrayBindVertexBufferEXT;
extern PFNGLVERTEXARRAYCOLOROFFSETEXTPROC glVertexArrayColorOffsetEXT;
extern PFNGLVERTEXARRAYEDGEFLAGOFFSETEXTPROC glVertexArrayEdgeFlagOffsetEXT;
extern PFNGLVERTEXARRAYELEMENTBUFFERPROC glVertexArrayElementBuffer;
extern PFNGLVERTEXARRAYFOGCOORDOFFSETEXTPROC glVertexArrayFogCoordOffsetEXT;
extern PFNGLVERTEXARRAYINDEXOFFSETEXTPROC glVertexArrayIndexOffsetEXT;
extern PFNGLVERTEXARRAYMULTITEXCOORDOFFSETEXTPROC glVertexArrayMultiTexCoordOffsetEXT;
extern PFNGLVERTEXARRAYNORMALOFFSETEXTPROC glVertexArrayNormalOffsetEXT;
extern PFNGLVERTEXARRAYSECONDARYCOLOROFFSETEXTPROC glVertexArraySecondaryColorOffsetEXT;
extern PFNGLVERTEXARRAYTEXCOORDOFFSETEXTPROC glVertexArrayTexCoordOffsetEXT;
```

```
extern PFNGLVERTEXARRAYVERTEXATTRIBBINDINGEXTPROC glVertexArrayVertexAttribBindingEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBDIVISOREXTPROC glVertexArrayVertexAttribDivisorEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBFORMATEXTPROC glVertexArrayVertexAttribFormatEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBIFORMATEXTPROC glVertexArrayVertexAttribIFormatEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBIOFFSETEXTPROC glVertexArrayVertexAttribIOffsetEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBLFORMATEXTPROC glVertexArrayVertexAttribLFormatEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBLOFFSETEXTPROC glVertexArrayVertexAttribLOffsetEXT;
extern PFNGLVERTEXARRAYVERTEXATTRIBOFFSETEXTPROC glVertexArrayVertexAttribOffsetEXT;
extern PFNGLVERTEXARRAYVERTEXBINDINGDIVISOREXTPROC glVertexArrayVertexBindingDivisorEXT;
extern PFNGLVERTEXARRAYVERTEXBUFFERPROC glVertexArrayVertexBuffer;
extern PFNGLVERTEXARRAYVERTEXBUFFERSPROC glVertexArrayVertexBuffers;
extern PFNGLVERTEXARRAYVERTEXOFFSETEXTPROC glVertexArrayVertexOffsetEXT;
extern PFNGLVERTEXATTRIB1DPROC glVertexAttrib1d;
extern PFNGLVERTEXATTRIB1DVPROC glVertexAttrib1dv;
extern PFNGLVERTEXATTRIB1FPROC glVertexAttrib1f;
extern PFNGLVERTEXATTRIB1FVPROC glVertexAttrib1fv;
extern PFNGLVERTEXATTRIB1SPROC glVertexAttrib1s;
extern PFNGLVERTEXATTRIB1SVPROC glVertexAttrib1sv;
extern PFNGLVERTEXATTRIB2DPROC glVertexAttrib2d;
extern PFNGLVERTEXATTRIB2DVPROC glVertexAttrib2dv;
extern PFNGLVERTEXATTRIB2FPROC glVertexAttrib2f;
extern PFNGLVERTEXATTRIB2FVPROC glVertexAttrib2fv;
extern PFNGLVERTEXATTRIB2SPROC glVertexAttrib2s;
extern PFNGLVERTEXATTRIB2SVPROC glVertexAttrib2sv;
extern PFNGLVERTEXATTRIB3DPROC glVertexAttrib3d;
extern PFNGLVERTEXATTRIB3DVPROC glVertexAttrib3dv;
extern PFNGLVERTEXATTRIB3FPROC glVertexAttrib3f;
extern PFNGLVERTEXATTRIB3FVPROC glVertexAttrib3fv;
extern PFNGLVERTEXATTRIB3SPROC glVertexAttrib3s;
extern PFNGLVERTEXATTRIB3SVPROC glVertexAttrib3sv;
extern PFNGLVERTEXATTRIB4BVPROC glVertexAttrib4bv;
extern PFNGLVERTEXATTRIB4DPROC glVertexAttrib4d;
extern PFNGLVERTEXATTRIB4DVPROC glVertexAttrib4dv;
extern PFNGLVERTEXATTRIB4FPROC glVertexAttrib4f;
extern PFNGLVERTEXATTRIB4FVPROC glVertexAttrib4fv;
extern PFNGLVERTEXATTRIB4IVPROC glVertexAttrib4iv;
extern PFNGLVERTEXATTRIB4NBVPROC glVertexAttrib4Nbv;
extern PFNGLVERTEXATTRIB4NIVPROC glVertexAttrib4Niv;
extern PFNGLVERTEXATTRIB4NSVPROC glVertexAttrib4Nsv;
extern PFNGLVERTEXATTRIB4NUBPROC glVertexAttrib4Nub;
extern PFNGLVERTEXATTRIB4NUBVPROC glVertexAttrib4Nubv;
extern PFNGLVERTEXATTRIB4NUIVPROC glVertexAttrib4Nuiv;
extern PFNGLVERTEXATTRIB4NUSVPROC glVertexAttrib4Nusv;
extern PFNGLVERTEXATTRIB4SPROC glVertexAttrib4s;
extern PFNGLVERTEXATTRIB4SVPROC glVertexAttrib4sv;
extern PFNGLVERTEXATTRIB4UBVPROC glVertexAttrib4ubv;
extern PFNGLVERTEXATTRIB4UIVPROC glVertexAttrib4uiv;
extern PFNGLVERTEXATTRIB4USVPROC glVertexAttrib4usv;
extern PFNGLVERTEXATTRIBBINDINGPROC glVertexAttribBinding;
extern PFNGLVERTEXATTRIBDIVISORARBPROC glVertexAttribDivisorARB;
extern PFNGLVERTEXATTRIBDIVISORPROC glVertexAttribDivisor;
extern PFNGLVERTEXATTRIBFORMATNVPROC glVertexAttribFormatNV;
extern PFNGLVERTEXATTRIBFORMATPROC glVertexAttribFormat;
extern PFNGLVERTEXATTRIBI1IPROC glVertexAttribI1i;
extern PFNGLVERTEXATTRIBI1IVPROC glVertexAttribI1iv;
extern PFNGLVERTEXATTRIBI1UIPROC glVertexAttribI1ui;
extern PFNGLVERTEXATTRIBI1UIVPROC glVertexAttribI1uiv;
extern PFNGLVERTEXATTRIBI2IPROC glVertexAttribI2i;
extern PFNGLVERTEXATTRIBI2IVPROC glVertexAttribI2iv;
extern PFNGLVERTEXATTRIBI2UIPROC glVertexAttribI2ui;
extern PFNGLVERTEXATTRIBI2UIVPROC glVertexAttribI2uiv;
extern PFNGLVERTEXATTRIBI3IPROC glVertexAttribI3i;
extern PFNGLVERTEXATTRIBI3IVPROC glVertexAttribI3iv;
extern PFNGLVERTEXATTRIBI3UIPROC glVertexAttribI3ui;
extern PFNGLVERTEXATTRIBI3UIVPROC glVertexAttribI3uiv;
extern PFNGLVERTEXATTRIBI4BVPROC glVertexAttribI4bv;
extern PFNGLVERTEXATTRIBI4IPROC glVertexAttribI4i;
extern PFNGLVERTEXATTRIBI4IVPROC glVertexAttribI4iv;
extern PFNGLVERTEXATTRIBI4SVPROC glVertexAttribI4sv;
extern PFNGLVERTEXATTRIBI4UBVPROC glVertexAttribI4ubv;
extern PFNGLVERTEXATTRIBI4UIPROC glVertexAttribI4ui;
extern PFNGLVERTEXATTRIBI4UIVPROC glVertexAttribI4uiv;
extern PFNGLVERTEXATTRIBI4USVPROC glVertexAttribI4usv;
extern PFNGLVERTEXATTRIBIFORMATNVPROC glVertexAttribIFormatNV;
extern PFNGLVERTEXATTRIBIFORMATPROC glVertexAttribIFormat;
extern PFNGLVERTEXATTRIBIPOINTERPROC glVertexAttribIPointer;
extern PFNGLVERTEXATTRIBL1DPROC glVertexAttribL1d;
extern PFNGLVERTEXATTRIBL1DVPROC glVertexAttribL1dv;
extern PFNGLVERTEXATTRIBL1I64NVPROC glVertexAttribL1i64NV;
extern PFNGLVERTEXATTRIBL1I64VNVPROC glVertexAttribL1i64vNV;
extern PFNGLVERTEXATTRIBL1UI64ARBPROC glVertexAttribL1ui64ARB;
extern PFNGLVERTEXATTRIBL1UI64NVPROC glVertexAttribL1ui64NV;
extern PFNGLVERTEXATTRIBL1UI64VARBPROC glVertexAttribL1ui64vARB;
extern PFNGLVERTEXATTRIBL1UI64VNVPROC glVertexAttribL1ui64vNV;
extern PFNGLVERTEXATTRIBL2DPROC glVertexAttribL2d;
extern PFNGLVERTEXATTRIBL2DVPROC glVertexAttribL2dv;
extern PFNGLVERTEXATTRIBL2I64NVPROC glVertexAttribL2i64NV;
```

```
extern PFNGLVERTEXATTRIBL2I64VNVPROC glVertexAttribL2i64vNV;
extern PFNGLVERTEXATTRIBL2UI64NVPROC glVertexAttribL2ui64NV;
extern PFNGLVERTEXATTRIBL2UI64VNVPROC glVertexAttribL2ui64vNV;
extern PFNGLVERTEXATTRIBL3DPROC glVertexAttribL3d;
extern PFNGLVERTEXATTRIBL3DVPROC glVertexAttribL3dv;
extern PFNGLVERTEXATTRIBL3I64NVPROC glVertexAttribL3i64NV;
extern PFNGLVERTEXATTRIBL3I64VNVPROC glVertexAttribL3i64vNV;
extern PFNGLVERTEXATTRIBL3UI64NVPROC glVertexAttribL3ui64NV;
extern PFNGLVERTEXATTRIBL3UI64VNVPROC glVertexAttribL3ui64vNV;
extern PFNGLVERTEXATTRIBL4DPROC glVertexAttribL4d;
extern PFNGLVERTEXATTRIBL4DVPROC glVertexAttribL4dv;
extern PFNGLVERTEXATTRIBL4I64NVPROC glVertexAttribL4i64NV;
extern PFNGLVERTEXATTRIBL4I64VNVPROC glVertexAttribL4i64vNV;
extern PFNGLVERTEXATTRIBL4UI64NVPROC glVertexAttribL4ui64NV;
extern PFNGLVERTEXATTRIBL4UI64VNVPROC glVertexAttribL4ui64vNV;
extern PFNGLVERTEXATTRIBLFORMATNVPROC glVertexAttribLFormatNV;
extern PFNGLVERTEXATTRIBLFORMATPROC glVertexAttribLFormat;
extern PFNGLVERTEXATTRIBLPOINTERPROC glVertexAttribLPointer;
extern PFNGLVERTEXATTRIBP1UIPROC glVertexAttribP1ui;
extern PFNGLVERTEXATTRIBP1UIVPROC glVertexAttribP1uiv;
extern PFNGLVERTEXATTRIBP2UIPROC glVertexAttribP2ui;
extern PFNGLVERTEXATTRIBP2UIVPROC glVertexAttribP2uiv;
extern PFNGLVERTEXATTRIBP3UIPROC glVertexAttribP3ui;
extern PFNGLVERTEXATTRIBP3UIVPROC glVertexAttribP3uiv;
extern PFNGLVERTEXATTRIBP4UIPROC glVertexAttribP4ui;
extern PFNGLVERTEXATTRIBP4UIVPROC glVertexAttribP4uiv;
extern PFNGLVERTEXATTRIBPOINTERPROC glVertexAttribPointer;
extern PFNGLVERTEXBINDINGDIVISORPROC glVertexBindingDivisor;
extern PFNGLVERTEXFORMATNVPROC glVertexFormatNV;
extern PFNGLVIEWPORTARRAYVPROC glViewportArrayv;
extern PFNGLVIEWPORTINDEXEDFPROC glViewportIndexedf;
extern PFNGLVIEWPORTINDEXEDFVPROC glViewportIndexedfv;
extern PFNGLVIEWPORTPOSITIONWSCALENVPROC glViewportPositionWScaleNV;
extern PFNGLVIEWPORTPROC glViewport;
extern PFNGLVIEWPORTSWIZZLENVPROC glViewportSwizzleNV;
extern PFNGLWAITSYNCPROC glWaitSync;
extern PFNGLWAITVKSEMAPHORENVPROC glWaitVkSemaphoreNV;
extern PFNGLWEIGHTPATHSNVPROC glWeightPathsNV;
extern PFNGLWINDOWRECTANGLESEXTPROC glWindowRectanglesEXT;
#endif


// 標準ライブラリ
#include <cmath>
#include <array>
#include <vector>
#include <string>
#include <memory>

namespace gg
{
  enum BindingPoints
  {
    LightBindingPoint = 0,
    MaterialBindingPoint,
  };

  extern GLint ggBufferAlignment;

  extern void ggInit();

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

#if defined(DEBUG)
#  define ggError() gg::_ggError(__FILE__, __LINE__)
#else
#  define ggError()
#endif

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

#if defined(DEBUG)
#  define ggFBOError() gg::_ggFBOError(__FILE__, __LINE__)
#else
#  define ggFBOError()
#endif

  inline void ggCross(GLfloat* c, const GLfloat* a, const GLfloat* b)
  {
    c[0] = a[1] * b[2] - a[2] * b[1];
    c[1] = a[2] * b[0] - a[0] * b[2];
    c[2] = a[0] * b[1] - a[1] * b[0];
  }

  inline GLfloat ggDot3(const GLfloat* a, const GLfloat* b)
  {
    return a[0] * b[0] + a[1] * b[1] + a[2] * b[2];
```

```
}

inline GLfloat ggLength3(const GLfloat* a)
{
  return sqrtf(ggDot3(a, a));
}

inline GLfloat ggDistance3(const GLfloat* a, const GLfloat* b)
{
  const GLfloat c[]{ a[0] - b[0], a[1] - b[1], a[2] - b[2], 0.0f };
  return ggLength3(c);
}

inline void ggNormalize3(const GLfloat* a, GLfloat* b)
{
  const GLfloat l{ ggLength3(a) };
  b[0] = a[0] / l;
  b[1] = a[1] / l;
  b[2] = a[2] / l;
}

inline void ggNormalize3(GLfloat* a)
{
  const GLfloat l{ ggLength3(a) };
  a[0] /= l;
  a[1] /= l;
  a[2] /= l;
}

inline GLfloat ggDot4(const GLfloat* a, const GLfloat* b)
{
  return a[0] * b[0] + a[1] * b[1] + a[2] * b[2] + a[3] * b[3];
}

inline GLfloat ggLength4(const GLfloat* a)
{
  return sqrtf(ggDot4(a, a));
}

inline GLfloat ggDistance4(const GLfloat* a, const GLfloat* b)
{
  const GLfloat c[]{ a[0] - b[0], a[1] - b[1], a[2] - b[2], a[3] - b[3] };
  return ggLength4(c);
}

inline void ggNormalize4(const GLfloat* a, GLfloat* b)
{
  const GLfloat l{ ggLength4(a) };
  b[0] = a[0] / l;
  b[1] = a[1] / l;
  b[2] = a[2] / l;
  b[3] = a[3] / l;
}

inline void ggNormalize4(GLfloat* a)
{
  const GLfloat l{ ggLength4(a) };
  a[0] /= l;
  a[1] /= l;
  a[2] /= l;
  a[3] /= l;
}

class GgVector : public std::array<GLfloat, 4>
{
public:

  GgVector()
  {
  }

  constexpr GgVector(GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3) :
    std::array<GLfloat, 4>{ v0, v1, v2, v3 }
  {
  }

  constexpr GgVector(GLfloat c) :
    GgVector{ c, c, c, c }
  {
  }

  constexpr GgVector(const GLfloat* a) :
    GgVector{ a[0], a[1], a[2], a[3] }
  {
  }

  GgVector operator+(const GgVector& v) const
```

```
    {
      return GgVector{ data()[0] + v[0], data()[1] + v[1], data()[2] + v[2], data()[3] + v[3] };
    }

    GgVector operator+(GLfloat c) const
    {
      return GgVector{ data()[0] + c, data()[1] + c, data()[2] + c, data()[3] + c };
    }

    GgVector& operator+=(const GgVector& v)
    {
      data()[0] += v[0];
      data()[1] += v[1];
      data()[2] += v[2];
      data()[3] += v[3];
      return *this;
    }

    GgVector& operator+=(GLfloat c)
    {
      data()[0] += c;
      data()[1] += c;
      data()[2] += c;
      data()[3] += c;
      return *this;
    }

    GgVector operator-(const GgVector& v) const
    {
      return GgVector{ data()[0] - v[0], data()[1] - v[1], data()[2] - v[2], data()[3] - v[3] };
    }

    GgVector operator-(GLfloat c) const
    {
      return GgVector{ data()[0] - c, data()[1] - c, data()[2] - c, data()[3] - c };
    }

    GgVector& operator-=(const GgVector& v)
    {
      data()[0] -= v[0];
      data()[1] -= v[1];
      data()[2] -= v[2];
      data()[3] -= v[3];
      return *this;
    }

    GgVector& operator-=(GLfloat c)
    {
      data()[0] -= c;
      data()[1] -= c;
      data()[2] -= c;
      data()[3] -= c;
      return *this;
    }

    GgVector operator*(const GgVector& v) const
    {
      return GgVector{ data()[0] * v[0], data()[1] * v[1], data()[2] * v[2], data()[3] * v[3] };
    }

    GgVector operator*(GLfloat c) const
    {
      return GgVector{ data()[0] * c, data()[1] * c, data()[2] * c, data()[3] * c };
    }

    GgVector& operator*=(const GgVector& v)
    {
      data()[0] *= v[0];
      data()[1] *= v[1];
      data()[2] *= v[2];
      data()[3] *= v[3];
      return *this;
    }

    GgVector& operator*=(GLfloat c)
    {
      data()[0] *= c;
      data()[1] *= c;
      data()[2] *= c;
      data()[3] *= c;
      return *this;
    }

    GgVector operator/(const GgVector& v) const
    {
      return GgVector{ data()[0] / v[0], data()[1] / v[1], data()[2] / v[2], data()[3] / v[3] };
    }
```

```
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] };
```

```
}

inline GgVector operator*(GLfloat a, const GgVector& b)
{
  return GgVector{ a * b[0], a * b[1], a * b[2], a * b[3] };
}

inline GgVector operator/(GLfloat a, const GgVector& b)
{
  return GgVector{ a / b[0], a / b[1], a / b[2], a / b[3] };
}

inline GgVector ggCross(const GgVector& a, const GgVector& b)
{
  GgVector c;
  ggCross(c.data(), a.data(), b.data());
  return c;
}

inline GLfloat ggDot3(const GgVector& a, const GgVector& b)
{
  return ggDot3(a.data(), b.data());;
}

inline GLfloat ggLength3(const GgVector& a)
{
  return ggLength3(a.data());
}

inline GLfloat ggDistance3(const GgVector& a, const GgVector& b)
{
  return ggDistance3(a.data(), b.data());
}

inline GgVector ggNormalize3(const GgVector& a)
{
  GgVector b;
  ggNormalize3(a.data(), b.data());
  b.data()[3] = 0.0f;
  return b;
}

inline void ggNormalize3(GgVector* a)
{
  ggNormalize3(a->data());
  a->data()[3] = 0.0f;
}

inline GLfloat ggDot4(const GgVector& a, const GgVector& b)
{
  return ggDot4(a.data(), b.data());
}

inline GLfloat ggLength4(const GgVector& a)
{
  return ggLength4(a.data());
}

inline GLfloat ggDistance4(const GgVector& a, const GgVector& b)
{
  return ggDistance4(a.data(), b.data());
}

inline GgVector ggNormalize4(const GgVector& a)
{
  GgVector b;
  ggNormalize4(a.data(), b.data());
  return b;
}

inline void ggNormalize4(GgVector* a)
{
  ggNormalize4(a->data());
}

class GgMatrix : public std::array<GLfloat, 16>
{
  // 行列 a とベクトル b の積をベクトル c に代入する
  void projection(GLfloat* c, const GLfloat* a, const GLfloat* b) const;

  // 行列 a と行列 b の積を行列 c に代入する
  void multiply(GLfloat* c, const GLfloat* a, const GLfloat* b) const;

public:

  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
    ) :
      std::array<GLfloat, 16>{ m00, m10, m20, m30, m01, m11, m21, m31, m02, m12, m22, m32, m03, m13,
m23, m33 }
    {
    }

    constexpr GgMatrix(GLfloat c) :
      GgMatrix{ c, c, c, c, c, c, c, c, c, c, c, c, c, c, c, c }
    {
    }

    GgMatrix(const GLfloat* a)
    {
      operator=(a);
    }

    GgMatrix& operator=(const GLfloat* a)
    {
      std::copy(a, a + size(), data());
      return *this;
    }

    GgMatrix operator+(const GLfloat* a) const
    {
      GgMatrix t;
      for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] + a[i];
      return t;
    }

    GgMatrix operator+(const GgMatrix& m) const
    {
      return operator+(m.data());
    }

    GgMatrix& operator+=(const GLfloat* a)
    {
      for (std::size_t i = 0; i < size(); ++i) data()[i] += a[i];
      return *this;
    }

    GgMatrix& operator+=(const GgMatrix& m)
    {
      return operator+=(m.data());
    }

    GgMatrix operator-(const GLfloat* a) const
    {
      GgMatrix t;
      for (std::size_t i = 0; i < size(); ++i) t[i] = data()[i] - a[i];
      return t;
    }

    GgMatrix operator-(const GgMatrix& m) const
    {
      return operator-(m.data());
    }

    GgMatrix& operator-=(const GLfloat* a)
    {
      for (std::size_t i = 0; i < size(); ++i) data()[i] -= a[i];
      return *this;
    }

    GgMatrix& operator-=(const GgMatrix& m)
    {
      return operator-=(m.data());
    }

    GgMatrix operator*(const GLfloat* a) const
    {
      GgMatrix t;
      multiply(t.data(), data(), a);
      return t;
    }

    GgMatrix operator*(const GgMatrix& m) const
    {
      return operator*(m.data());
    }

```

```
    GgMatrix& operator*=(const GLfloat* a)
    {
      *this = operator*(a);
      return *this;
    }

    GgMatrix& operator*=(const GgMatrix& m)
    {
      return operator*=(m.data());
    }

    GgMatrix operator/(const GLfloat* a) const
    {
      GgMatrix i, t;
      i.loadInvert(a);
      multiply(t.data(), data(), i.data());
      return t;
    }

    GgMatrix operator/(const GgMatrix& m) const
    {
      return operator/(m.data());
    }

    GgMatrix& operator/=(const GLfloat* a)
    {
      *this = operator/(a);
      return *this;
    }

    GgMatrix& operator/=(const GgMatrix& m)
    {
      return operator/=(m.data());
    }

    GgMatrix& loadIdentity();

    GgMatrix& loadTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);

    GgMatrix& loadTranslate(const GLfloat* t)
    {
      return loadTranslate(t[0], t[1], t[2]);
    }

    GgMatrix& loadTranslate(const GgVector& t)
    {
      return loadTranslate(t[0], t[1], t[2], t[3]);
    }

    GgMatrix& loadScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f);

    GgMatrix& loadScale(const GLfloat* s)
    {
      return loadScale(s[0], s[1], s[2]);
    }

    GgMatrix& loadScale(const GgVector& s)
    {
      return loadScale(s[0], s[1], s[2], s[3]);
    }

    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)
    {
      return loadRotate(r[0], r[1], r[2], a);
    }

    GgMatrix& loadRotate(const GgVector& r, GLfloat a)
    {
      return loadRotate(r[0], r[1], r[2], a);
    }

    GgMatrix& loadRotate(const GLfloat* r)
    {
      return loadRotate(r[0], r[1], r[2], r[3]);
    }

    GgMatrix& loadRotate(const GgVector& r)
    {
      return loadRotate(r[0], r[1], r[2], r[3]);
```

```
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
```

```
    {
      GgMatrix m;
      return operator*(m.loadRotateY(a));
    }

    GgMatrix rotateZ(GLfloat a) const
    {
      GgMatrix m;
      return operator*(m.loadRotateZ(a));
    }

    GgMatrix rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
    {
      GgMatrix m;
      return operator*(m.loadRotate(x, y, z, a));
    }

    GgMatrix rotate(const GLfloat* r, GLfloat a) const
    {
      return rotate(r[0], r[1], r[2], a);
    }

    GgMatrix rotate(const GgVector& r, GLfloat a) const
    {
      return rotate(r[0], r[1], r[2], a);
    }

    GgMatrix rotate(const GLfloat* r) const
    {
      return rotate(r[0], r[1], r[2], r[3]);
    }

    GgMatrix rotate(const GgVector& r) const
    {
      return rotate(r[0], r[1], r[2], r[3]);
    }

    GgMatrix lookat(
      GLfloat ex, GLfloat ey, GLfloat ez,
      GLfloat tx, GLfloat ty, GLfloat tz,
      GLfloat ux, GLfloat uy, GLfloat uz
    ) const
    {
      GgMatrix m;
      return operator*(m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz));
    }

    GgMatrix lookat(const GLfloat* e, const GLfloat* t, const GLfloat* u) const
    {
      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
    }

    GgMatrix lookat(const GgVector& e, const GgVector& t, const GgVector& u) const
    {
      return lookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
    }

    GgMatrix orthogonal(
      GLfloat left, GLfloat right,
      GLfloat bottom, GLfloat top,
      GLfloat zNear, GLfloat zFar
    ) const
    {
      GgMatrix m;
      return operator*(m.loadOrthogonal(left, right, bottom, top, zNear, zFar));
    }

    GgMatrix frustum(
      GLfloat left, GLfloat right,
      GLfloat bottom, GLfloat top,
      GLfloat zNear, GLfloat zFar
    ) const
    {
      GgMatrix m;
      return operator*(m.loadFrustum(left, right, bottom, top, zNear, zFar));
    }

    GgMatrix perspective(
      GLfloat fovy, GLfloat aspect,
      GLfloat zNear, GLfloat zFar
    ) const
    {
      GgMatrix m;
      return operator*(m.loadPerspective(fovy, aspect, zNear, zFar));
    }

    GgMatrix transpose() const
```

```
    {
      GgMatrix t;
      return t.loadTranspose(*this);
    }

    GgMatrix invert() const
    {
      GgMatrix t;
      return t.loadInvert(*this);
    }

    GgMatrix normal() const
    {
      GgMatrix t;
      return t.loadNormal(*this);
    }

    void projection(GLfloat* c, const GLfloat* v) const
    {
      projection(c, data(), v);
    }

    void projection(GLfloat* c, const GgVector& v) const
    {
      projection(c, v.data());
    }

    void projection(GgVector& c, const GLfloat* v) const
    {
      projection(c.data(), v);
    }

    void projection(GgVector& c, const GgVector& v) const
    {
      projection(c.data(), v.data());
    }

    GgVector operator*(const GgVector& v) const
    {
      GgVector c;
      projection(c, v);
      return c;
    }

    const GLfloat* get() const
    {
      return data();
    }

    void get(GLfloat* a) const
    {
      std::copy(data(), data() + size(), a);
    }
  };

  inline GgMatrix ggIdentity()
  {
    GgMatrix t;
    return t.loadIdentity();
  };

  inline GgMatrix ggTranslate(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
  {
    GgMatrix m;
    return m.loadTranslate(x, y, z, w);
  }

  inline GgMatrix ggTranslate(const GLfloat* t)
  {
    GgMatrix m;
    return m.loadTranslate(t[0], t[1], t[2]);
  }

  inline GgMatrix ggTranslate(const GgVector& t)
  {
    GgMatrix m;
    return m.loadTranslate(t[0], t[1], t[2], t[3]);
  }

  inline GgMatrix ggScale(GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f)
  {
    GgMatrix m;
    return m.loadScale(x, y, z, w);
  }

  inline GgMatrix ggScale(const GLfloat* s)
  {
```

```
  GgMatrix m;
  return m.loadScale(s[0], s[1], s[2]);
}

inline GgMatrix ggScale(const GgVector& s)
{
  GgMatrix m;
  return m.loadScale(s[0], s[1], s[2], s[3]);
}

inline GgMatrix ggRotateX(GLfloat a)
{
  GgMatrix m;
  return m.loadRotateX(a);
}

inline GgMatrix ggRotateY(GLfloat a)
{
  GgMatrix m;
  return m.loadRotateY(a);
}

inline GgMatrix ggRotateZ(GLfloat a)
{
  GgMatrix m;
  return m.loadRotateZ(a);
}

inline GgMatrix ggRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
{
  GgMatrix m;
  return m.loadRotate(x, y, z, a);
}

inline GgMatrix ggRotate(const GLfloat* r, GLfloat a)
{
  GgMatrix m;
  return m.loadRotate(r[0], r[1], r[2], a);
}

inline GgMatrix ggRotate(const GgVector& r, GLfloat a)
{
  GgMatrix m;
  return m.loadRotate(r[0], r[1], r[2], a);
}

inline GgMatrix ggRotate(const GLfloat* r)
{
  GgMatrix m;
  return m.loadRotate(r[0], r[1], r[2], r[3]);
}

inline GgMatrix ggRotate(const GgVector& r)
{
  GgMatrix m;
  return m.loadRotate(r[0], r[1], r[2], r[3]);
}

inline GgMatrix ggLookat(
  GLfloat ex, GLfloat ey, GLfloat ez,     // 視点の位置
  GLfloat tx, GLfloat ty, GLfloat tz,     // 目標点の位置
  GLfloat ux, GLfloat uy, GLfloat uz      // 上方向のベクトル
)
{
  GgMatrix m;
  return m.loadLookat(ex, ey, ez, tx, ty, tz, ux, uy, uz);
}

inline GgMatrix ggLookat(
  const GLfloat* e,                       // 視点の位置
  const GLfloat* t,                       // 目標点の位置
  const GLfloat* u                        // 上方向のベクトル
)
{
  GgMatrix m;
  return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
}

inline GgMatrix ggLookat(
  const GgVector& e,                      // 視点の位置
  const GgVector& t,                      // 目標点の位置
  const GgVector& u                       // 上方向のベクトル
)
{
  GgMatrix m;
  return m.loadLookat(e[0], e[1], e[2], t[0], t[1], t[2], u[0], u[1], u[2]);
}
```

```
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     }
```

```
    GgQuaternion& load(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
    {
      data()[0] = x;
      data()[1] = y;
      data()[2] = z;
      data()[3] = w;
      return *this;
    }

    GgQuaternion& load(const GLfloat* a)
    {
      return load(a[0], a[1], a[2], a[3]);
    }

    GgQuaternion& load(const GgVector& v)
    {
      static_cast<GgVector>(*this) = v;
      return *this;
    }

    GgQuaternion& load(const GgQuaternion& q)
    {
      *this = q;
      return *this;
    }

    GgQuaternion& loadAdd(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
    {
      data()[0] += x;
      data()[1] += y;
      data()[2] += z;
      data()[3] += w;
      return *this;
    }

    GgQuaternion& loadAdd(const GLfloat* a)
    {
      return loadAdd(a[0], a[1], a[2], a[3]);
    }

    GgQuaternion& loadAdd(const GgVector& v)
    {
      return loadAdd(v[0], v[1], v[2], v[3]);
    }

    GgQuaternion& loadAdd(const GgQuaternion& q)
    {
      return loadAdd(q[0], q[1], q[2], q[3]);
    }

    GgQuaternion& loadSubtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
    {
      data()[0] -= x;
      data()[1] -= y;
      data()[2] -= z;
      data()[3] -= w;
      return *this;
    }

    GgQuaternion& loadSubtract(const GLfloat* a)
    {
      return loadSubtract(a[0], a[1], a[2], a[3]);
    }

    GgQuaternion& loadSubtract(const GgVector& v)
    {
      return loadSubtract(v[0], v[1], v[2], v[3]);
    }

    GgQuaternion& loadSubtract(const GgQuaternion& q)
    {
      return loadSubtract(q[0], q[1], q[2], q[3]);
    }

    GgQuaternion& loadMultiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
    {
      const GLfloat a[]{ x, y, z, w };
      return loadMultiply(a);
    }

    GgQuaternion& loadMultiply(const GLfloat* a)
    {
      return load(multiply(a));
    }

    GgQuaternion& loadMultiply(const GgVector& v)
```

```
03673     {
03674       return loadMultiply(v.data());
03675     }
03676
03683     GgQuaternion& loadMultiply(const GgQuaternion& q)
03684     {
03685       return loadMultiply(q.data());
03686     }
03687
03697     GgQuaternion& loadDivide(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
03698     {
03699       const GLfloat a[]{ x, y, z, w };
03700       return loadDivide(a);
03701     }
03702
03709     GgQuaternion& loadDivide(const GLfloat* a)
03710     {
03711       return load(divide(a));
03712     }
03713
03720     GgQuaternion& loadDivide(const GgVector& v)
03721     {
03722       return loadDivide(v.data());
03723     }
03724
03731     GgQuaternion& loadDivide(const GgQuaternion& q)
03732     {
03733       return loadDivide(q.data());
03734     }
03735
03745     GgQuaternion add(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03746     {
03747       GgQuaternion s{ *this };
03748       return s.loadAdd(x, y, z, w);
03749     }
03750
03757     GgQuaternion add(const GLfloat* a) const
03758     {
03759       return add(a[0], a[1], a[2], a[3]);
03760     }
03761
03768     GgQuaternion add(const GgVector& v) const
03769     {
03770       return add(v[0], v[1], v[2], v[3]);
03771     }
03772
03779     GgQuaternion add(const GgQuaternion& q) const
03780     {
03781       return add(q[0], q[1], q[2], q[3]);
03782     }
03783
03793     GgQuaternion subtract(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03794     {
03795       GgQuaternion s{ *this };
03796       return s.loadSubtract(x, y, z, w);
03797     }
03798
03805     GgQuaternion subtract(const GLfloat* a) const
03806     {
03807       return subtract(a[0], a[1], a[2], a[3]);
03808     }
03809
03816     GgQuaternion subtract(const GgVector& v) const
03817     {
03818       return subtract(v[0], v[1], v[2], v[3]);
03819     }
03820
03827     GgQuaternion subtract(const GgQuaternion& q) const
03828     {
03829       return subtract(q[0], q[1], q[2], q[3]);
03830     }
03831
03841     GgQuaternion multiply(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
03842     {
03843       const GLfloat a[]{ x, y, z, w };
03844       return multiply(a);
03845     }
03846
03853     GgQuaternion multiply(const GLfloat* a) const
03854     {
03855       GgQuaternion s;
03856       multiply(s.data(), data(), a);
03857       return s;
03858     }
03859
03866     GgQuaternion multiply(const GgVector& v) const
03867     {
```

```
    return multiply(v.data());
  }

  GgQuaternion multiply(const GgQuaternion& q) const
  {
    return multiply(q.data());
  }

  GgQuaternion divide(GLfloat x, GLfloat y, GLfloat z, GLfloat w) const
  {
    const GLfloat a[]{ x, y, z, w };
    return divide(a);
  }

  GgQuaternion divide(const GLfloat* a) const
  {
    GgQuaternion s, ia;
    ia.loadInvert(a);
    multiply(s.data(), data(), ia.data());
    return s;
  }

  GgQuaternion divide(const GgVector& v) const
  {
    return divide(v.data());
  }

  GgQuaternion divide(const GgQuaternion& q) const
  {
    return divide(q.data());
  }

  // 演算子
  GgQuaternion& operator=(const GLfloat* a)
  {
    return load(a);
  }
  GgQuaternion& operator=(const GgVector& v)
  {
    return load(v);
  }
  GgQuaternion& operator+=(const GLfloat* a)
  {
    return loadAdd(a);
  }
  GgQuaternion& operator+=(const GgVector& v)
  {
    return loadAdd(v);
  }
  GgQuaternion& operator+=(const GgQuaternion& q)
  {
    return loadAdd(q);
  }
  GgQuaternion& operator-=(const GLfloat* a)
  {
    return loadSubtract(a);
  }
  GgQuaternion& operator-=(const GgVector& v)
  {
    return loadSubtract(v);
  }
  GgQuaternion& operator-=(const GgQuaternion& q)
  {
    return operator-=(q.data());
  }
  GgQuaternion& operator*=(const GLfloat* a)
  {
    return loadMultiply(a);
  }
  GgQuaternion& operator*=(const GgVector& v)
  {
    return loadMultiply(v);
  }
  GgQuaternion& operator*=(const GgQuaternion& q)
  {
    return operator*=(q.data());
  }
  GgQuaternion& operator/=(const GLfloat* a)
  {
    return loadDivide(a);
  }
  GgQuaternion& operator/=(const GgVector& v)
  {
    return loadDivide(v);
  }
  GgQuaternion& operator/=(const GgQuaternion& q)
  {
```

```
    return operator/=(q.data());
  }
  GgQuaternion operator+(const GLfloat* a) const
  {
    return add(a);
  }
  GgQuaternion operator+(const GgVector& v) const
  {
    return add(v);
  }
  GgQuaternion operator+(const GgQuaternion& q) const
  {
    return operator+(q.data());
  }
  GgQuaternion operator-(const GLfloat* a) const
  {
    return add(a);
  }
  GgQuaternion operator-(const GgVector& v) const
  {
    return add(v);
  }
  GgQuaternion operator-(const GgQuaternion& q) const
  {
    return operator-(q.data());
  }
  GgQuaternion operator*(const GLfloat* a) const
  {
    return multiply(a);
  }
  GgQuaternion operator*(const GgVector& v) const
  {
    return multiply(v);
  }
  GgQuaternion operator*(const GgQuaternion& q) const
  {
    return operator*(q.data());
  }
  GgQuaternion operator/(const GLfloat* a) const
  {
    return divide(a);
  }
  GgQuaternion operator/(const GgVector& v) const
  {
    return divide(v);
  }
  GgQuaternion operator/(const GgQuaternion& q) const
  {
    return operator/(q.data());
  }

  GgQuaternion& loadMatrix(const GLfloat* a)
  {
    toQuaternion(data(), a);
    return *this;
  }

  GgQuaternion& loadMatrix(const GgMatrix& m)
  {
    return loadMatrix(m.get());
  }

  GgQuaternion& loadIdentity()
  {
    return load(0.0f, 0.0f, 0.0f, 1.0f);
  }

  GgQuaternion& loadRotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a);

  GgQuaternion& loadRotate(const GLfloat* v, GLfloat a)
  {
    return loadRotate(v[0], v[1], v[2], a);
  }

  GgQuaternion& loadRotate(const GLfloat* v)
  {
    return loadRotate(v[0], v[1], v[2], v[3]);
  }

  GgQuaternion& loadRotateX(GLfloat a);

  GgQuaternion& loadRotateY(GLfloat a);

  GgQuaternion& loadRotateZ(GLfloat a);

  GgQuaternion rotate(GLfloat x, GLfloat y, GLfloat z, GLfloat a) const
  {
```

```
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());
```

```
04355     }
04356
04364     GgQuaternion slerp(GLfloat* a, GLfloat t) const
04365     {
04366       GgQuaternion p;
04367       slerp(p.data(), data(), a, t);
04368       return p;
04369     }
04370
04378     GgQuaternion slerp(const GgQuaternion& q, GLfloat t) const
04379     {
04380       GgQuaternion p;
04381       slerp(p.data(), data(), q.data(), t);
04382       return p;
04383     }
04384
04390     GgQuaternion normalize() const
04391     {
04392       GgQuaternion q;
04393       q.loadNormalize(data());
04394       return q;
04395     }
04396
04402     GgQuaternion conjugate() const
04403     {
04404       GgQuaternion q;
04405       q.loadConjugate(data());
04406       return q;
04407     }
04408
04414     GgQuaternion invert() const
04415     {
04416       GgQuaternion q;
04417       q.loadInvert(data());
04418       return q;
04419     }
04420
04426     void get(GLfloat* a) const
04427     {
04428       a[0] = data()[0];
04429       a[1] = data()[1];
04430       a[2] = data()[2];
04431       a[3] = data()[3];
04432     }
04433
04439     void getMatrix(GLfloat* a) const
04440     {
04441       toMatrix(a, data());
04442     }
04443
04449     void getMatrix(GgMatrix& m) const
04450     {
04451       getMatrix(m.data());
04452     }
04453
04459     GgMatrix getMatrix() const
04460     {
04461       GgMatrix m;
04462       getMatrix(m);
04463       return m;
04464     }
04465
04471     void getConjugateMatrix(GLfloat* a) const
04472     {
04473       GgQuaternion c;
04474       c.loadConjugate(data());
04475       toMatrix(a, c.data());
04476     }
04477
04483     void getConjugateMatrix(GgMatrix& m) const
04484     {
04485       getConjugateMatrix(m.data());
04486     }
04487
04493     GgMatrix getConjugateMatrix() const
04494     {
04495       GgMatrix m;
04496       getConjugateMatrix(m);
04497       return m;
04498     }
04499   };
04500
04510   inline GgQuaternion ggQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat w)
04511   {
04512     GgQuaternion q;
04513     return q.load(x, y, z, w);
04514   }
```

```
inline GgQuaternion ggQuaternion(const GLfloat* a)
{
  return ggQuaternion(a[0], a[1], a[2], a[3]);
}

inline GgQuaternion ggIdentityQuaternion()
{
  GgQuaternion q;
  return q.loadIdentity();
}

inline GgQuaternion ggMatrixQuaternion(const GLfloat* a)
{
  GgQuaternion q;
  return q.loadMatrix(a);
}

inline GgQuaternion ggMatrixQuaternion(const GgMatrix& m)
{
  return ggMatrixQuaternion(m.get());
}

inline GgMatrix ggQuaternionMatrix(const GgQuaternion& q)
{
  GLfloat m[16];
  q.getMatrix(m);
  return GgMatrix{ m };
}

inline GgMatrix ggQuaternionTransposeMatrix(const GgQuaternion& q)
{
  GLfloat m[16];
  q.getMatrix(m);
  GgMatrix t;
  return t.loadTranspose(m);
}

inline GgQuaternion ggRotateQuaternion(GLfloat x, GLfloat y, GLfloat z, GLfloat a)
{
  GgQuaternion q;
  return q.loadRotate(x, y, z, a);
}

inline GgQuaternion ggRotateQuaternion(const GLfloat* v, GLfloat a)
{
  return ggRotateQuaternion(v[0], v[1], v[2], a);
}

inline GgQuaternion ggRotateQuaternion(const GLfloat* v)
{
  return ggRotateQuaternion(v[0], v[1], v[2], v[3]);
}

inline GgQuaternion ggEulerQuaternion(GLfloat heading, GLfloat pitch, GLfloat roll)
{
  GgQuaternion q;
  return q.loadEuler(heading, pitch, roll);
}

inline GgQuaternion ggEulerQuaternion(const GLfloat* e)
{
  return ggEulerQuaternion(e[0], e[1], e[2]);
}

inline GgQuaternion ggSlerp(const GLfloat* a, const GLfloat* b, GLfloat t)
{
  GgQuaternion r;
  return r.loadSlerp(a, b, t);
}

inline GgQuaternion ggSlerp(const GgQuaternion& q, const GgQuaternion& r, GLfloat t)
{
  return ggSlerp(q.data(), r.data(), t);
}

inline GgQuaternion ggSlerp(const GgQuaternion& q, const GLfloat* a, GLfloat t)
{
  return ggSlerp(q.data(), a, t);
}

inline GgQuaternion ggSlerp(const GLfloat* a, const GgQuaternion& q, GLfloat t)
{
  return ggSlerp(a, q.data(), t);
}

inline GLfloat ggNorm(const GgQuaternion& q)
```

```
{
  return q.norm();
}

inline GgQuaternion ggNormalize(const GgQuaternion& q)
{
  return q.normalize();
}

inline GgQuaternion ggConjugate(const GgQuaternion& q)
{
  return q.conjugate();
}

inline GgQuaternion ggInvert(const GgQuaternion& q)
{
  return q.invert();
}

class GgTrackball : public GgQuaternion
{
  bool drag;         // ドラッグ中か否か
  GLfloat start[2]; // ドラッグ開始位置
  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
  GgQuaternion cq;  // 回転の初期値 (四元数)
  GgMatrix rt;      // 回転の変換行列

public:

  GgTrackball(const GgQuaternion& q = ggIdentityQuaternion())
  {
    reset(q);
  }

  virtual ~GgTrackball()
  {
  }

  GgTrackball& operator=(const GgQuaternion& q)
  {
    this->load(q);
    return *this;
  }

  void region(GLfloat w, GLfloat h);

  void region(int w, int h)
  {
    region(static_cast<GLfloat>(w), static_cast<GLfloat>(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
  {
    return start;
  }

  const GLfloat& getStart(int direction) const
  {
    return start[direction];
  }

  void getStart(GLfloat* position) const
  {
    position[0] = start[0];
    position[1] = start[1];
  }

  const GLfloat* getScale() const
  {
    return scale;
  }

  const GLfloat getScale(int direction) const
  {
    return scale[direction];
  }

```

```
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     {
```

```
      size[0] = getWidth();
      size[1] = getHeight();
    }

    const GLsizei* getSize() const
    {
      return size;
    }

    const GLuint& getTexture() const
    {
      return texture;
    }
  };

  class GgColorTexture
  {
    // テクスチャ
    std::shared_ptr<GgTexture> texture;

  public:

    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
    )
    {
      load(image, width, height, format, type, internal, wrap, swizzle);
    }

    GgColorTexture(
      const std::string& name,
      GLenum internal = 0,
      GLenum wrap = GL_CLAMP_TO_EDGE
    )
    {
      load(name, internal, wrap);
    }

    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
    )
    {
      // テクスチャを作成する
      texture = std::make_shared<GgTexture>(image, width, height, format, type, internal, wrap,
swizzle);
    }

    void load(
      const std::string& name,
      GLenum internal = 0,
      GLenum wrap = GL_CLAMP_TO_EDGE
    );
  };

  class GgNormalTexture
  {
    // テクスチャ
    std::shared_ptr<GgTexture> texture;

  public:

    GgNormalTexture()
    {
    }

```

```
    GgNormalTexture(
      const GLubyte* image,
      GLsizei width,
      GLsizei height,
      GLenum format = GL_RED,
      GLfloat nz = 1.0f,
      GLenum internal = GL_RGBA
    )
    {
      // 法線マップのテクスチャを作成する
      load(image, width, height, format, nz, internal);
    }

    GgNormalTexture(
      const std::string& name,
      GLfloat nz = 1.0f,
      GLenum internal = GL_RGBA
    )
    {
      // 法線マップのテクスチャを作成する
      load(name, nz, internal);
    }

    virtual ~GgNormalTexture()
    {
    }

    void load(
      const GLubyte* hmap,
      GLsizei width,
      GLsizei height,
      GLenum format = GL_RED,
      GLfloat nz = 1.0f,
      GLenum internal = GL_RGBA
    )
    {
      // 法線マップ
      std::vector<GgVector> nmap;

      // 法線マップを作成する
      ggCreateNormalMap(hmap, width, height, format, nz, internal, nmap);

      // テクスチャを作成する
      texture = std::make_shared<GgTexture>(nmap.data(), width, height, GL_RGBA, GL_FLOAT, internal,
GL_REPEAT);
    }

    void load(
      const std::string& name,
      GLfloat nz = 1.0f,
      GLenum internal = GL_RGBA
    );
  };

  template <typename T>
  class GgBuffer
  {
    // ターゲット
    const GLenum target;

    // 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
    const GLsizei stride;

    // データの数
    const GLsizei count;

    // バッファオブジェクト
    const GLuint buffer;

  public:

    GgBuffer<T>(
      GLenum target,
      const T* data,
      GLsizei stride,
      GLsizei count,
      GLenum usage
      ) :
      target{ target },
      stride{ stride },
      count{ count },
      buffer{ [] { GLuint buffer; glGenBuffers(1, &buffer); return buffer; } () }
    {
      // 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
      glBindBuffer(target, buffer);
      glBufferData(target, getStride()* count, data, usage);
    }
```

```
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)
```

```
      const GLsizeiptr begin{ getStride() * first };
      const GLsizeiptr range{ getStride() * count };
      T* const source{ glMapBufferRange(target, begin, range, GL_MAP_READ_BIT) };
      std::copy(source, source + count, data);
      glUnmapBuffer(target);
#else
      glGetBufferSubData(target, getStride() * first, getStride() * count, data);
#endif
    }

    void copy(GLuint src_buffer, GLint src_first = 0, GLint dst_first = 0, GLsizei count = 0) const
    {
      // count が 0 なら全データを複写する
      if (count == 0) count = getCount();
      if (src_first + count > getCount()) count = getCount() - src_first;
      if (dst_first + count > getCount()) count = getCount() - dst_first;

      // データの間隔
      const GLsizeiptr stride{ getStride() };

      glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
      glBindBuffer(GL_COPY_WRITE_BUFFER, buffer);
      glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
        stride * src_first, stride * dst_first, stride * count);
      glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
      glBindBuffer(GL_COPY_READ_BUFFER, 0);
    }
  };

  template <typename T>
  class GgUniformBuffer
  {
    // ユニフォームバッファオブジェクト
    std::shared_ptr<GgBuffer<T>> uniform;

  public:

    GgUniformBuffer<T>()
    {
    }

    GgUniformBuffer<T>(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
    {
      load(data, count, usage);
    }

    GgUniformBuffer<T>(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
    {
      load(data, count, usage);
    }

    virtual ~GgUniformBuffer<T>()
    {
    }

    const GLuint& getTarget() const
    {
      return uniform->getTarget();
    }

    GLsizeiptr getStride() const
    {
      return uniform->getStride();
    }

    const GLsizei& getCount() const
    {
      return uniform->getCount();
    }

    const GLuint& getBuffer() const
    {
      return uniform->getBuffer();
    }

    void bind() const
    {
      uniform->bind();
    }

    void unbind() const
    {
      uniform->unbind();
    }

    void* map() const
    {
```

```
      return uniform->map();
    }

    void* map(GLint first, GLsizei count) const
    {
      return uniform->map(first, count);
    }

    void unmap() const
    {
      uniform->unmap();
    }

    void load(const T* data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
    {
      // バッファオブジェクト上のデータの間隔
      const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) * ggBufferAlignment };

      // ユニフォームバッファオブジェクトを確保する
      uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);

      // 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
      if (data) send(data, 0, sizeof(T), 0, count);
    }

    void load(const T& data, GLsizei count, GLenum usage = GL_STATIC_DRAW)
    {
      // バッファオブジェクト上のデータの間隔
      const GLsizei stride{ (((static_cast<GLint>(sizeof(T)) - 1) / ggBufferAlignment) + 1) * ggBufferAlignment };

      // ユニフォームバッファオブジェクトを確保する
      uniform = std::make_shared<GgBuffer<T>>(GL_UNIFORM_BUFFER, nullptr, stride, count, usage);

      // 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
      fill(&data, 0, sizeof(T), 0, count);
    }

    void send(
      const GLvoid* data,
      GLint offset = 0,
      GLsizei size = sizeof(T),
      GLint first = 0,
      GLsizei count = 0
    ) const
    {
      // count が 0 なら全データを転送する
      if (count == 0) count = getCount();
      if (first + count > getCount()) count = getCount() - first;

      // 転送元のデータの先頭
      const char* source{ reinterpret_cast<const char*>(data) };

      // ターゲット
      const GLuint target{ getTarget() };

      // データの間隔
      const GLsizeiptr stride{ getStride() };

      // 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
      bind();
      for (GLsizei i = 0; i < count; ++i)
      {
        glBufferSubData(target, stride * (first + i) + offset, size, source + size * i);
      }
    }

    void fill(
      const GLvoid* data,
      GLint offset = 0,
      GLsizei size = sizeof(T),
      GLint first = 0,
      GLsizei count = 0
    ) const
    {
      // count が 0 なら全データを転送する
      if (count == 0) count = getCount();
      if (first + count > getCount()) count = getCount() - first;

      // ターゲット
      const GLuint target{ getTarget() };

      // データの間隔
      const GLsizeiptr stride{ getStride() };

      // 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
```

```
      bind();
      for (GLsizei i = 0; i < count; ++i)
      {
        glBufferSubData(target, stride * (first + i) + offset, size, data);
      }
    }

    void read(
      GLvoid* data,
      GLint offset = 0,
      GLsizei size = sizeof(T),
      GLint first = 0,
      GLsizei count = 0
    ) const
    {
      // count が 0 なら全データを転送する
      if (count == 0) count = getCount();
      if (first + count > getCount()) count = getCount() - first;

      // 抽出先のデータの先頭
      char* const destination{ reinterpret_cast<char*>(data) };

      // ターゲット
      const GLuint target{ getTarget() };

      // データの間隔
      const GLsizeiptr stride{ getStride() };

      // 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
      bind();
#if defined(GL_GLES_PROTOTYPES)
      const char* const source{ glMapBufferRange(target, stride * first, stride * count,
 GL_MAP_READ_BIT) };
      for (GLsizei i = 0; i < count; ++i)
      {
        const char* const begin{ source + stride * i + offset };
        const char* const end{ begin + size };
        std::copy(begin, end, destination + sizeof(T) * i);
      }
      glUnmapBuffer(target);
#else
      for (GLsizei i = 0; i < count; ++i)
      {
        glGetBufferSubData(target, stride * (first + i) + offset, size, destination + sizeof(T) * i);
      }
#endif
    }

    void copy(
      GLuint src_buffer,
      GLint src_first = 0,
      GLint dst_first = 0,
      GLsizei count = 0
    ) const
    {
      // count が 0 なら全データを複写する
      if (count == 0) count = getCount();
      if (src_first + count > getCount()) count = getCount() - src_first;
      if (dst_first + count > getCount()) count = getCount() - dst_first;

      // データの間隔
      const GLsizeiptr stride{ getStride() };

      // 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
      glBindBuffer(GL_COPY_READ_BUFFER, src_buffer);
      glBindBuffer(GL_COPY_WRITE_BUFFER, getBuffer());
      for (GLsizei i = 0; i < count; ++i)
      {
        glCopyBufferSubData(GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER,
          stride * (src_first + i), stride * (dst_first + i), sizeof(T));
      }
      glBindBuffer(GL_COPY_WRITE_BUFFER, 0);
      glBindBuffer(GL_COPY_READ_BUFFER, 0);
    }
  };

  class GgShape
  {
    // 頂点配列オブジェクト
    const GLuint vao;

    // 基本図形の種類
    GLenum mode;

  public:

    GgShape(GLenum mode = 0) :
```

```
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
06290     GgVertex()
06291     {
06292     }
06293
06300     GgVertex(const GgVector& pos, const GgVector& norm) :
06301       position(pos),
06302       normal(norm)
06303     {
06304     }
06305
06316     GgVertex(
06317       GLfloat px, GLfloat py, GLfloat pz,
06318       GLfloat nx, GLfloat ny, GLfloat nz
06319     ) :
06320       position{ px, py, pz, 1.0f },
06321       normal{ nx, ny, nz, 0.0f }
06322     {
06323     }
06324
06331     GgVertex(const GLfloat* pos, const GLfloat* norm) :
06332       GgVertex(pos[0], pos[1], pos[2], norm[0], norm[1], norm[2])
06333     {
06334     }
06335   };
06336
06340   class GgTriangles
06341     : public GgShape
06342   {
06343     // 頂点属性
06344     std::unique_ptr<GgBuffer<GgVertex>> vertex;
06345
06346   public:
06347
06353     GgTriangles(GLenum mode = GL_TRIANGLES) :
06354       GgShape(mode)
06355     {
06356     }
06357
06366     GgTriangles(
06367       const GgVertex* vert,
06368       GLsizei count,
06369       GLenum mode = GL_TRIANGLES,
06370       GLenum usage = GL_STATIC_DRAW
06371     ) :
06372       GgTriangles(mode)
06373     {
06374       load(vert, count, usage);
06375     }
06376
06380     virtual ~GgTriangles()
06381     {
06382     }
06383
06389     const GLsizei& getCount() const
06390     {
06391       return vertex->getCount();
06392     }
06393
06399     const GLuint& getBuffer() const
06400     {
06401       return vertex->getBuffer();
06402     }
06403
06411     void send(const GgVertex* vert, GLint first = 0, GLsizei count = 0) const
06412     {
06413       vertex->send(vert, first, count);
06414     }
06415
06423     void load(const GgVertex* vert, GLsizei count, GLenum usage = GL_STATIC_DRAW);
06424
06431     virtual void draw(GLint first = 0, GLsizei count = 0) const;
06432   };
06433
06437   class GgElements
06438     : public GgTriangles
06439   {
06440     // インデックスを格納する頂点バッファオブジェクト
06441     std::unique_ptr<GgBuffer<GLuint>> index;
06442
06443   public:
06444
06450     GgElements(GLenum mode = GL_TRIANGLES) :
06451       GgTriangles(mode)
```

```
    {
    }

    GgElements(
      const GgVertex* vert,
      GLsizei countv,
      const GLuint* face,
      GLsizei countf,
      GLenum mode = GL_TRIANGLES,
      GLenum usage = GL_STATIC_DRAW
    ) :
      GgElements(mode)
    {
      load(vert, countv, face, countf, usage);
    }

    virtual ~GgElements()
    {
    }

    const GLsizei& getIndexCount() const
    {
      return index->getCount();
    }

    const GLuint& getIndexBuffer() const
    {
      return index->getBuffer();
    }

    void send(
      const GgVertex* vert,
      GLuint firstv,
      GLsizei countv,
      const GLuint* face = nullptr,
      GLuint firstf = 0,
      GLsizei countf = 0
    ) const
    {
      GgTriangles::send(vert, firstv, countv);
      if (face != nullptr && countf > 0) index->send(face, firstf, countf);
    }

    void load(
      const GgVertex* vert,
      GLsizei countv,
      const GLuint* face,
      GLsizei countf,
      GLenum usage = GL_STATIC_DRAW
    )
    {
      // 頂点バッファオブジェクトを作成する
      GgTriangles::load(vert, countv, usage);

      // インデックスの頂点バッファオブジェクトを作成する
      index = std::make_unique<GgBuffer<GLuint>>(GL_ELEMENT_ARRAY_BUFFER, face,
static_cast<GLsizei>(sizeof(GLuint)), countf, usage);
    }

    virtual void draw(GLint first = 0, GLsizei count = 0) const;
  };

  extern std::unique_ptr<GgPoints> ggPointsCube(
    GLsizei countv,
    GLfloat length = 1.0f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f
  );

  extern std::unique_ptr<GgPoints> ggPointsSphere(
    GLsizei countv,
    GLfloat radius = 0.5f,
    GLfloat cx = 0.0f,
    GLfloat cy = 0.0f,
    GLfloat cz = 0.0f
  );

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

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

```
);

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

extern std::unique_ptr<GgElements> ggElementsObj(
  const std::string& name,
  bool normalize = false
);

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

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

class GgShader
{
  // プログラム名
  const GLuint program;

public:

  GgShader(
    const std::string& vert,
    const std::string& frag = "",
    const std::string& geom = "",
    int nvarying = 0,
    const char* const* varyings = nullptr
  ) :
    program(ggLoadShader(vert, frag, geom, nvarying, varyings))
  {
  }

  GgShader(
    const std::array<std::string, 3>& files,
    int nvarying = 0,
    const char* const* varyings = nullptr
  ) :
    GgShader(files[0], files[1], files[2], nvarying, varyings)
  {
  }

  GgShader(const GgShader& o) = delete;

  virtual ~GgShader()
  {
    // 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
    glUseProgram(0);
    glDeleteProgram(program);
  }

  GgShader& operator=(const GgShader& o) = delete;

  void use() const
  {
    glUseProgram(program);
  }

  void unuse() const
  {
    glUseProgram(0);
  }

  GLuint get() const
  {
    return program;
  }
};

class GgPointShader
{
  // シェーダー
  std::shared_ptr<GgShader> shader;

  // 投影変換行列の uniform 変数の場所
  GLint mpLoc;

```

```
    // モデルビュー変換行列の uniform 変数の場所
    GLint mvLoc;

  public:

    GgPointShader() :
      mpLoc{ -1 },
      mvLoc{ -1 }
    {
    }

    GgPointShader(
      const std::string& vert,
      const std::string& frag = "",
      const std::string& geom = "",
      GLint nvarying = 0,
      const char* const* varyings = nullptr
    ) :
      GgPointShader()
    {
      load(vert, frag, geom, nvarying, varyings);
    }

    GgPointShader(
      const std::array<std::string, 3>& files,
      int nvarying = 0,
      const char* const* varyings = nullptr
    ) :
      GgPointShader(files[0], files[1], files[2], nvarying, varyings)
    {
    }

    virtual ~GgPointShader()
    {
    }

    bool load(
      const std::string& vert,
      const std::string& frag = "",
      const std::string& geom = "",
      GLint nvarying = 0,
      const char* const* varyings = nullptr
    )
    {
      // シェーダを作成する
      shader = std::make_shared<GgShader>(vert, frag, geom, nvarying, varyings);

      // プログラム名を取り出す
      const GLuint program(shader->get());

      // プログラムオブジェクトが作成できていなければ戻る
      if (program == 0) return false;

      // 変換行列の uniform 変数の場所
      mpLoc = glGetUniformLocation(program, "mp");
      mvLoc = glGetUniformLocation(program, "mv");

      // プログラムオブジェクトの作成に成功した
      return true;
    }

    bool load(
      const std::array<std::string, 3>& files,
      GLint nvarying = 0,
      const char* const* varyings = nullptr
    )
    {
      return load(files[0], files[1], files[2], nvarying, varyings);
    }

    virtual void loadProjectionMatrix(const GLfloat* mp) const
    {
      glUniformMatrix4fv(mpLoc, 1, GL_FALSE, mp);
    }

    virtual void loadProjectionMatrix(const GgMatrix& mp) const
    {
      loadProjectionMatrix(mp.get());
    }

    virtual void loadModelviewMatrix(const GLfloat* mv) const
    {
      glUniformMatrix4fv(mvLoc, 1, GL_FALSE, mv);
    }

    virtual void loadModelviewMatrix(const GgMatrix& mv) const
    {
```

```
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   };
07039
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,
```

```
      GLint nvarying = 0,
      const char* const* varyings = nullptr
    ) :
      GgSimpleShader(files[0], files[1], files[2], nvarying, varyings)
    {
    }

    GgSimpleShader(const GgSimpleShader& o) :
      GgPointShader(o),
      materialIndex{ o.materialIndex },
      lightIndex{ o.lightIndex },
      mnLoc{ o.mnLoc }
    {
    }

    virtual ~GgSimpleShader()
    {
    }

    GgSimpleShader& operator=(const GgSimpleShader& o)
    {
      if (&o != this)
      {
        GgPointShader::operator=(o);
        materialIndex = o.materialIndex;
        lightIndex = o.lightIndex;
        mnLoc = o.mnLoc;
      }

      return *this;
    }

    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
    )
    {
      return load(files[0], files[1], files[2], nvarying, varyings);
    }

    virtual void loadModelviewMatrix(const GLfloat* mv, const GLfloat* mn) const
    {
      GgPointShader::loadModelviewMatrix(mv);
      glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
    }

    virtual void loadModelviewMatrix(const GgMatrix& mv, const GgMatrix& mn) const
    {
      loadModelviewMatrix(mv.get(), mn.get());
    }

    virtual void loadModelviewMatrix(const GLfloat* mv) const
    {
      loadModelviewMatrix(mv, GgMatrix(mv).normal().get());
    }

    virtual void loadModelviewMatrix(const GgMatrix& mv) const
    {
      loadModelviewMatrix(mv.get());
    }

    virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv, const GLfloat* mn) const
    {
      GgPointShader::loadMatrix(mp, mv);
      glUniformMatrix4fv(mnLoc, 1, GL_FALSE, mn);
    }

    virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv, const GgMatrix& mn) const
    {
      loadMatrix(mp.get(), mv.get(), mn.get());
    }

    virtual void loadMatrix(const GLfloat* mp, const GLfloat* mv) const
    {
      loadMatrix(mp, mv, GgMatrix(mv).normal());
    }

    virtual void loadMatrix(const GgMatrix& mp, const GgMatrix& mv) const
```

```
07259     {
07260       loadMatrix(mp, mv, mv.normal());
07261     }
07262
07266     struct Light
07267     {
07268       GgVector ambient;
07269       GgVector diffuse;
07270       GgVector specular;
07271       GgVector position;
07272     };
07273
07277     class LightBuffer
07278       : public GgUniformBuffer<Light>
07279     {
07280     public:
07281
07289       LightBuffer(
07290         const Light* light = nullptr,
07291         GLsizei count = 1,
07292         GLenum usage = GL_STATIC_DRAW
07293       ) :
07294         GgUniformBuffer<Light>(light, count, usage)
07295       {
07296       }
07297
07305       LightBuffer(
07306         const Light& light,
07307         GLsizei count = 1,
07308         GLenum usage = GL_STATIC_DRAW
07309       ) :
07310         GgUniformBuffer<Light>(light, count, usage)
07311       {
07312       }
07313
07317       virtual ~LightBuffer()
07318       {
07319       }
07320
07331       void loadAmbient(
07332         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07333         GLint first = 0, GLsizei count = 1
07334       ) const;
07335
07343       void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07344       {
07345         // first 番目のブロックから count 個の ambient 要素に値を設定する
07346         send(ambient, offsetof(Light, ambient), sizeof(Light::ambient), first, count);
07347       }
07348
07356       void loadAmbient(const GgVector& ambient, GLint first = 0, GLsizei count = 1) const;
07357
07368       void loadDiffuse(
07369         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07370         GLint first = 0, GLsizei count = 1
07371       ) const;
07372
07380       void loadDiffuse(const GgVector& diffuse, GLint first = 0, GLsizei count = 1) const;
07381
07389       void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07390       {
07391         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07392         send(diffuse, offsetof(Light, diffuse), sizeof(Light::diffuse), first, count);
07393       }
07394
07405       void loadSpecular(
07406         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07407         GLint first = 0, GLsizei count = 1
07408       ) const;
07409
07417       void loadSpecular(const GgVector& specular, GLint first = 0, GLsizei count = 1) const;
07418
07426       void loadSpecular(const GLfloat* specular, GLint first = 0, GLsizei count = 1) const
07427       {
07428         // first 番目のブロックから count 個の specular 要素に値を設定する
07429         send(specular, offsetof(Light, specular), sizeof(Light::specular), first, count);
07430       }
07431
07439       void loadColor(const Light& color, GLint first = 0, GLsizei count = 1) const;
07440
07451       void loadPosition(
07452         GLfloat x, GLfloat y, GLfloat z, GLfloat w = 1.0f,
07453         GLint first = 0, GLsizei count = 1
07454       ) const;
07455
07463       void loadPosition(const GgVector& position, GLint first = 0, GLsizei count = 1) const;
07464
```

```
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
07485       void loadPosition(const GgVector* position, GLint first = 0, GLsizei count = 1) const
07486       {
07487         loadPosition(position->data(), first, count);
07488       }
07489
07497       void load(const Light* light, GLint first = 0, GLsizei count = 1) const
07498       {
07499         send(light, 0, sizeof(Light), first, count);
07500       }
07501
07509       void load(const Light& light, GLint first = 0, GLsizei count = 1) const
07510       {
07511         load(&light, first, count);
07512       }
07513
07519       void select(GLint i = 0) const
07520       {
07521         // バッファオブジェクトの i 番目のブロックの位置
07522         const GLintptr offset(static_cast<GLintptr>(getStride()) * i);
07523         glBindBufferRange(getTarget(), LightBindingPoint, getBuffer(), offset, sizeof(Light));
07524       }
07525     };
07526
07530     struct Material
07531     {
07532       GgVector ambient;
07533       GgVector diffuse;
07534       GgVector specular;
07535       GLfloat shininess;
07536     };
07537
07541     class MaterialBuffer
07542       : public GgUniformBuffer<Material>
07543     {
07544     public:
07545
07553       MaterialBuffer(
07554         const Material* material = nullptr,
07555         GLsizei count = 1,
07556         GLenum usage = GL_STATIC_DRAW
07557       ) :
07558         GgUniformBuffer<Material>(material, count, usage)
07559       {
07560       }
07561
07569       MaterialBuffer(
07570         const Material& material,
07571         GLsizei count = 1,
07572         GLenum usage = GL_STATIC_DRAW
07573       ) :
07574         GgUniformBuffer<Material>(material, count, usage)
07575       {
07576       }
07577
07581       virtual ~MaterialBuffer()
07582       {
07583       }
07584
07595       void loadAmbient(
07596         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07597         GLint first = 0, GLsizei count = 1
07598       ) const;
07599
07607       void loadAmbient(const GLfloat* ambient, GLint first = 0, GLsizei count = 1) const
07608       {
07609         // first 番目のブロックから count 個のブロックの ambient 要素に値を設定する
07610         send(ambient, offsetof(Material, ambient), sizeof(Material::ambient), first, count);
07611       }
07612
07623       void loadDiffuse(
07624         GLfloat r, GLfloat g, GLfloat b, GLfloat a = 1.0f,
07625         GLint first = 0, GLsizei count = 1
07626       ) const;
07627
07635       void loadDiffuse(const GLfloat* diffuse, GLint first = 0, GLsizei count = 1) const
07636       {
07637         // first 番目のブロックから count 個の diffuse 要素に値を設定する
07638         send(diffuse, offsetof(Material, diffuse), sizeof(Material::diffuse), first, count);
07639       }
07640
07651       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,
```

```
      const LightBuffer* light,
      GLint i = 0
    ) const
    {
      // 光源を指定してプログラムオブジェクトを指定する
      use(light, i);

      // 変換行列を設定する
      loadMatrix(mp, mv, mn);
    }

    void use(
      const GgMatrix& mp,
      const GgMatrix& mv,
      const GgMatrix& mn,
      const LightBuffer& light,
      GLint i = 0
    ) const
    {
      use(mp.get(), mv.get(), mn.get(), &light, i);
    }

    void use(
      const GLfloat* mp,
      const GLfloat* mv,
      const LightBuffer* light,
      GLint i = 0
    ) const
    {
      use(mp, mv, GgMatrix(mv).normal().get(), light, i);
    }

    void use(
      const GgMatrix& mp,
      const GgMatrix& mv,
      const LightBuffer& light,
      GLint i = 0
    ) const
    {
      use(mp, mv, mv.normal(), light, i);
    }

    void use(const GLfloat* mp, const LightBuffer* light, GLint i = 0) const
    {
      // 光源を指定してプログラムオブジェクトを指定する
      use(light, i);

      // 投影変換行列を設定する
      loadProjectionMatrix(mp);
    }

    void use(const GgMatrix& mp, const LightBuffer& light, GLint i = 0) const
    {
      // 光源を指定してプログラムオブジェクトを指定する
      use(mp.get(), &light, i);
    }
  };

  extern 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
  );

  extern 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
  );

  class GgSimpleObj
  {
    // 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
    std::shared_ptr <std::vector<std::array<GLuint, 3>>> group;

    // ポリゴングループごとの材質のユニフォームバッファ
    std::shared_ptr<GgSimpleShader::MaterialBuffer> material;

    // この図形の形状データ
    std::shared_ptr<GgElements> data;

  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]

```
#include "GgApp.h"

//
// GLFW のエラー表示
//
static void glfwErrorCallback(int error, const char* description)
{
#if defined(__aarch64__)
  if (error == 65544) return;
#endif
  throw std::runtime_error(description);
}

//
// GgApp クラスのコンストラクタ
//
GgApp::GgApp(int major, int minor)
{
  // GLFW のエラー処理関数を登録する
  glfwSetErrorCallback(glfwErrorCallback);

  // GLFW を初期化する
  if (glfwInit() == GL_FALSE) throw std::runtime_error("Can't initialize GLFW");

  // OpenGL の major 番号が指定されていれば
  if (major > 0)
  {
    // OpenGL のバージョンを指定する
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, major);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, minor);

#if defined(GL_GLES_PROTOTYPES)
    // OpenGL ES 3 のコンテキストを指定する
    glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_ES_API);
    glfwWindowHint(GLFW_CONTEXT_CREATION_API, GLFW_EGL_CONTEXT_API);
#else
    // OpenGL Version 3.2 以降なら
    if (major * 10 + minor >= 32)
    {
      // Core Profile を選択する (macOS の都合)
      glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    }
#endif
  }

#if defined(GG_USE_OCULUS_RIFT)
  // Oculus Rift では SRGB でレンダリングする
  glfwWindowHint(GLFW_SRGB_CAPABLE, GL_TRUE);
#endif

#if defined(IMGUI_VERSION)
  // ImGui のバージョンをチェックする
  IMGUI_CHECKVERSION();

  // ImGui のコンテキストを作成する
  ImGui::CreateContext();
#endif
}

//
// デストラクタ
//
GgApp::~GgApp()
{
#if defined(IMGUI_VERSION)
  // Shutdown Platform/Renderer bindings
  ImGui_ImplOpenGL3_Shutdown();
  ImGui_ImplGlfw_Shutdown();
  ImGui::DestroyContext();
#endif

  // プログラム終了時に GLFW を終了する
  glfwTerminate();
}

//
// マウスや矢印キーによる平行移動量を初期化する
//
void GgApp::Window::HumanInterface::resetTranslation()
{
```

```
  // 平行移動量を初期化する
  for (auto& t : translation)
  {
    std::fill(t.begin(), t.end(), GgVector{ 0.0f, 0.0f, 0.0f, 1.0f });
  }

  // 矢印キーの設定値を初期化する
  std::fill(arrow.begin(), arrow.end(), std::array<int, 2>{ 0, 0 });

  // マウスホイールの回転量を初期化する
  std::fill(wheel.begin(), wheel.end(), 0.0f);
}

//
// 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)
//
void GgApp::Window::HumanInterface::calcTranslation(int button, const std::array<GLfloat, 3>&
 velocity)
{
  // マウスの相対変位
  assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
  const auto dx{ (mouse[0] - rotation[button].getStart(0)) * rotation[button].getScale(0) };
  const auto dy{ (rotation[button].getStart(1) - mouse[1]) * rotation[button].getScale(1) };

  // 平行移動量
  auto& t{ translation[button] };

  // 平行移動量の更新
  t[1][0] = dx * velocity[0] + t[0][0];
  t[1][1] = dy * velocity[1] + t[0][1];

  // 回転量の更新
  rotation[button].motion(mouse[0], mouse[1]);
}

//
// ウィンドウのサイズ変更時の処理
//
void GgApp::Window::resize(GLFWwindow* window, int width, int height)
{
  // このインスタンスの this ポインタを得る
  auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };

  if (instance)
  {
    // ウィンドウのサイズを保存する
    instance->size[0] = width;
    instance->size[1] = height;

    // トラックボール処理の範囲を設定する
    for (auto& current_if : instance->interfaceData)
    {
      for (auto& t : current_if.rotation)
      {
        t.region(width, height);
      }
    }

    // ビューポートを更新する
    instance->updateViewport();

    // ユーザー定義のコールバック関数の呼び出し
    if (instance->resizeFunc) (*instance->resizeFunc)(instance, width, height);
  }
}

//
// キーボードをタイプした時の処理
//
void GgApp::Window::keyboard(GLFWwindow* window, int key, int scancode, int action, int mods)
{
#if defined(IMGUI_VERSION)
  // 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
  if (ImGui::GetIO().WantCaptureKeyboard) return;
#endif

  // このインスタンスの this ポインタを得る
  auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };

  if (instance && action)
  {
    // ユーザー定義のコールバック関数の呼び出し
    if (instance->keyboardFunc) (*instance->keyboardFunc)(instance, key, scancode, action, mods);

    // 対象のユーザインタフェース
    auto& current_if{ instance->interfaceData[instance->interfaceNo] };

```

```
    switch (key)
    {
    case GLFW_KEY_HOME:

      // トラックボールを初期化する
      instance->resetRotation();
      [[fallthrough]];

    case GLFW_KEY_END:

      // 平行移動量を初期化する
      instance->resetTranslation();
      break;

    case GLFW_KEY_UP:

      if (mods & GLFW_MOD_SHIFT)
        current_if.arrow[1][1]++;
      else if (mods & GLFW_MOD_CONTROL)
        current_if.arrow[2][1]++;
      else if (mods & GLFW_MOD_ALT)
        current_if.arrow[3][1]++;
      else
        current_if.arrow[0][1]++;
      break;

    case GLFW_KEY_DOWN:

      if (mods & GLFW_MOD_SHIFT)
        current_if.arrow[1][1]--;
      else if (mods & GLFW_MOD_CONTROL)
        current_if.arrow[2][1]--;
      else if (mods & GLFW_MOD_ALT)
        current_if.arrow[3][1]--;
      else
        current_if.arrow[0][1]--;
      break;

    case GLFW_KEY_RIGHT:

      if (mods & GLFW_MOD_SHIFT)
        current_if.arrow[1][0]++;
      else if (mods & GLFW_MOD_CONTROL)
        current_if.arrow[2][0]++;
      else if (mods & GLFW_MOD_ALT)
        current_if.arrow[3][0]++;
      else
        current_if.arrow[0][0]++;
      break;

    case GLFW_KEY_LEFT:

      if (mods & GLFW_MOD_SHIFT)
        current_if.arrow[1][0]--;
      else if (mods & GLFW_MOD_CONTROL)
        current_if.arrow[2][0]--;
      else if (mods & GLFW_MOD_ALT)
        current_if.arrow[3][0]--;
      else
        current_if.arrow[0][0]--;
      break;

    default:
      break;
    }

    current_if.lastKey = key;
  }
}

//
// マウスボタンを操作したときの処理
//
void GgApp::Window::mouse(GLFWwindow* window, int button, int action, int mods)
{
#if defined(IMGUI_VERSION)
  // 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
  if (ImGui::GetIO().WantCaptureMouse) return;
#endif

  // このインスタンスの this ポインタを得る
  auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };

  // マウスボタンの状態を記録する
  assert(button >= GLFW_MOUSE_BUTTON_1 && button < GLFW_MOUSE_BUTTON_1 + GG_BUTTON_COUNT);
  instance->status[button] = action != GLFW_RELEASE;

```

```
  if (instance)
  {
    // ユーザー定義のコールバック関数の呼び出し
    if (instance->mouseFunc) (*instance->mouseFunc)(instance, button, action, mods);

    // 対象のユーザインタフェース
    auto& current_if{ instance->interfaceData[instance->interfaceNo] };

    // マウスの現在位置を得る
    const auto x{ current_if.mouse[0] };
    const auto y{ current_if.mouse[1] };

    if (x < 0 || x >= instance->size[0] || y < 0 || y >= instance->size[1]) return;

    if (action)
    {
      // ドラッグ開始
      current_if.rotation[button].begin(x, y);
    }
    else
    {
      // ドラッグ終了
      current_if.translation[button][0] = current_if.translation[button][1];
      current_if.rotation[button].end(x, y);
    }
  }
}

//
// マウスホイールを操作した時の処理
//
void GgApp::Window::wheel(GLFWwindow* window, double x, double y)
{
#if defined(IMGUI_VERSION)
  // 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
  if (ImGui::GetIO().WantCaptureMouse) return;
#endif

  // このインスタンスの this ポインタを得る
  auto* const instance{ static_cast<Window*>(glfwGetWindowUserPointer(window)) };

  if (instance)
  {
    // ユーザー定義のコールバック関数の呼び出し
    if (instance->wheelFunc) (*instance->wheelFunc)(instance, x, y);

    // 対象のユーザインタフェース
    auto& current_if{ instance->interfaceData[instance->interfaceNo] };

    // マウスホイールの回転量の保存
    current_if.wheel[0] += static_cast<GLfloat>(x);
    current_if.wheel[1] += static_cast<GLfloat>(y);

    // マウスによる平行移動量の z 値の更新
    const auto z{ current_if.wheel[1] * instance->velocity[2] };
    for (auto& t : current_if.translation) t[1][2] = z;
  }
}

//
// Window クラスのコンストラクタ
//
GgApp::Window::Window(const std::string& title, int width, int height, int fullscreen, GLFWwindow*
    share) :
  window{ nullptr },
  size{ width, height },
  fboSize{ width, height },
#if defined(IMGUI_VERSION)
  menubarHeight{ 0 },
#endif
  aspect{ 1.0f },
  velocity{ 1.0f, 1.0f, 0.1f },
  status{ false },
  interfaceNo{ 0 },
  userPointer{ nullptr },
  resizeFunc{ nullptr },
  keyboardFunc{ nullptr },
  mouseFunc{ nullptr },
  wheelFunc{ nullptr }
{
  // ディスプレイの情報
  GLFWmonitor* monitor{ nullptr };

  // フルスクリーン表示
  if (fullscreen > 0)
  {
    // 接続されているモニタの数を数える
```

```
    int mcount;
    auto** const monitors{ glfwGetMonitors(&mcount) };

    // セカンダリモニタがあればそれを使う
    if (fullscreen > mcount) fullscreen = mcount;
    monitor = monitors[fullscreen - 1];

    // モニタのモードを調べる
    const auto* mode{ glfwGetVideoMode(monitor) };

    // ウィンドウのサイズをディスプレイのサイズにする
    width = mode->width;
    height = mode->height;
  }

  // GLFW のウィンドウを作成する
  window = glfwCreateWindow(width, height, title.c_str(), monitor, share);

  // ウィンドウが作成できなければエラー
  if (!window) throw std::runtime_error("Unable to open the GLFW window.");

  // 現在のウィンドウを処理対象にする
  glfwMakeContextCurrent(window);

  // ゲームグラフィックス特論の都合による初期化を行う
  ggInit();

  // このインスタンスの this ポインタを記録しておく
  glfwSetWindowUserPointer(window, this);

  // キーボードを操作した時の処理を登録する
  glfwSetKeyCallback(window, keyboard);

  // マウスボタンを操作したときの処理を登録する
  glfwSetMouseButtonCallback(window, mouse);

  // マウスホイール操作時に呼び出す処理を登録する
  glfwSetScrollCallback(window, wheel);

  // ウィンドウのサイズ変更時に呼び出す処理を登録する
  glfwSetFramebufferSizeCallback(window, resize);

  // 垂直同期タイミングに合わせる
  glfwSwapInterval(1);

  // 実際のフレームバッファのサイズを取得する
  glfwGetFramebufferSize(window, &width, &height);

  // ビューポートと投影変換行列を初期化する
  resize(window, width, height);

#if defined(IMGUI_VERSION)
  // 最初のウィンドウを開いたとき
  static bool firstTime{ true };
  if (firstTime)
  {
    // Setup Platform/Renderer bindings
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init(nullptr);

    // 実行済みであることを記録する
    firstTime = false;
  }
#endif
}

//
// 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
//
GgApp::Window::operator bool()
{
  // イベントを取り出す
  glfwPollEvents();

  // ウィンドウを閉じるべきなら false を返す
  if (shouldClose()) return false;

  // 対象のユーザインタフェース
  auto& current_if{ interfaceData[interfaceNo] };

#if defined(IMGUI_VERSION)
  // ImGui の新規フレームを作成する
  ImGui_ImplOpenGL3_NewFrame();
  ImGui_ImplGlfw_NewFrame();

  // ImGui の状態を取り出す
  const ImGuiIO& io{ ImGui::GetIO() };
```

```
00435
00436   // 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
00437   if (io.WantCaptureMouse) return true;
00438
00439   // マウスの位置を更新する
00440   current_if.mouse = std::array<GLfloat, 2>{ io.MousePos.x, io.MousePos.y };
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_OCULUS_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
```

```
      adapter->GetDesc(&desc);
      memcpy(&luid, &desc.AdapterLuid, sizeof luid);
      adapter->Release();
    }

    factory->Release();
  }
#    endif

  return luid;
}

//
// グラフィックスカードの LUID の比較
//
int GgApp::Oculus::Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs)
{
  return memcmp(&lhs, &rhs, sizeof(ovrGraphicsLuid));
}
#  endif

//
// コンストラクタ
//
GgApp::Oculus::Oculus() :
  session{ nullptr },
  oculusFbo{ 0 },
  screen{ -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, },
  mirrorFbo{ 0 },
  window{ nullptr },
#  if OVR_PRODUCT_VERSION > 0
  frameIndex{ 0LL },
  oculusDepth{ 0 },
  mirrorWidth{ 1280 },
  mirrorHeight{ 640 },
#  endif
  mirrorTexture{ nullptr }
{
}

//
// Oculus Rift のセッションを作成する
//
GgApp::Oculus& GgApp::Oculus::initialize(const Window& window)
{
  // Oculus Rift のコンテキスト
  static Oculus oculus;

  // 既に Oculus Rift のセッションが作成されていたら参照を返す
  if (oculus.session) return oculus;

  // 最初に呼び出したときだけ実行する
  static bool firstTime{ true };
  if (firstTime)
  {
    // Oculus Rift (LibOVR) を初期化する
    ovrInitParams initParams{ ovrInit_RequestVersion, OVR_MINOR_VERSION, NULL, 0, 0 };
    if (OVR_FAILURE(ovr_Initialize(&initParams)))
      throw std::runtime_error("Can't initialize LibOVR");

    // アプリケーションの終了時に LibOVR を終了する
    atexit(ovr_Shutdown);

    // 実行済みであることを記録する
    firstTime = false;
  }

  // Oculus Rift のセッションを作成する
  ovrGraphicsLuid luid;
  if (OVR_FAILURE(ovr_Create(&oculus.session, &luid)))
    throw std::runtime_error("Can't create Oculus Rift session");

#  if OVR_PRODUCT_VERSION > 0
  // 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
  if (Compare(luid, GetDefaultAdapterLuid()))
    throw std::runtime_error("Graphics adapter is not default");
#  endif

  // session が無効ならエラー
  if (!oculus.session) std::runtime_error("Unable to use the Oculus Rift.");

  // ミラー表示を行うウィンドウを設定する
  oculus.window = &window;

  // Oculus Rift の情報を取り出す
  oculus.hmdDesc = ovr_GetHmdDesc(oculus.session);

```

```
#  if defined(_DEBUG)
  // Oculus Rift の情報を表示する
  std::cerr
    << "\nProduct name: " << oculus.hmdDesc.ProductName
    << "\nResolution:   " << oculus.hmdDesc.Resolution.w << " x " << oculus.hmdDesc.Resolution.h
    << "\nDefault Fov:  (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].LeftTan
    << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].DownTan
    << ") - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].RightTan
    << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Left].UpTan
    << ")\n              (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].LeftTan
    << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].DownTan
    << ") - (" << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].RightTan
    << "," << oculus.hmdDesc.DefaultEyeFov[ovrEye_Right].UpTan
    << ")\nMaximum Fov:  (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].LeftTan
    << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].DownTan
    << ") - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].RightTan
    << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Left].UpTan
    << ")\n              (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].LeftTan
    << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].DownTan
    << ") - (" << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].RightTan
    << "," << oculus.hmdDesc.MaxEyeFov[ovrEye_Right].UpTan
    << ")\n" << std::endl;
#  endif

  // Oculus Rift に転送する描画データを作成する
#  if OVR_PRODUCT_VERSION > 0
  oculus.layerData.Header.Type = ovrLayerType_EyeFov;
#  else
  oculus.layerData.Header.Type = ovrLayerType_EyeFovDepth;
#  endif

  // OpenGL なので左下が原点
  oculus.layerData.Header.Flags = ovrLayerFlag_TextureOriginAtBottomLeft;

  // Oculus Rift のレンダリングに使う FBO を作成する
  glGenFramebuffers(ovrEye_Count, oculus.oculusFbo);

  // 全ての目について
  for (int eye = 0; eye < ovrEye_Count; ++eye)
  {
    // Oculus Rift の視野を取得する
    const auto& fov{ oculus.hmdDesc.DefaultEyeFov[ovrEyeType(eye)] };

    // Oculus Rift 用の FBO のサイズを求める
    const auto textureSize{ ovr_GetFovTextureSize(oculus.session, ovrEyeType(eye), fov, 1.0f) };

    // Oculus Rift のスクリーンのサイズを保存する
    oculus.screen[eye][0] = -fov.LeftTan;
    oculus.screen[eye][1] = fov.RightTan;
    oculus.screen[eye][2] = -fov.DownTan;
    oculus.screen[eye][3] = fov.UpTan;

#  if OVR_PRODUCT_VERSION > 0

    // 描画データに視野を設定する
    oculus.layerData.Fov[eye] = fov;

    // 描画データにビューポートを設定する
    oculus.layerData.Viewport[eye].Pos = OVR::Vector2i(0, 0);
    oculus.layerData.Viewport[eye].Size = textureSize;

    // 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
    const ovrTextureSwapChainDesc colorDesc
    {
      ovrTexture_2D,                    // Type
      OVR_FORMAT_R8G8B8A8_UNORM_SRGB,   // Format
      1,                                // ArraySize
      textureSize.w,                    // Width
      textureSize.h,                    // Height
      1,                                // MipLevels
      1,                                // SampleCount
      ovrFalse,                         // StaticImage
      0, 0
    };

    // 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
    oculus.layerData.ColorTexture[eye] = nullptr;
    if (OVR_SUCCESS(ovr_CreateTextureSwapChainGL(oculus.session, &colorDesc, &oculus.layerData.ColorTexture[eye])))
    {
      // 作成したテクスチャチェインの長さを取得する
      int length(0);
      if (OVR_SUCCESS(ovr_GetTextureSwapChainLength(oculus.session, oculus.layerData.ColorTexture[eye], &length)))
      {
        // テクスチャチェインの個々の要素について
        for (int i = 0; i < length; ++i)
```

```
        {
          // テクスチャのパラメータを設定する
          GLuint texId{ 0 };
          ovr_GetTextureSwapChainBufferGL(oculus.session, oculus.layerData.ColorTexture[eye], i, &texId);
          glBindTexture(GL_TEXTURE_2D, texId);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
          glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        }
      }

      // 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
      glGenTextures(1, oculus.oculusDepth + eye);
      glBindTexture(GL_TEXTURE_2D, oculus.oculusDepth[eye]);
      glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    }

#  else

    // 描画データに視野を設定する
    oculus.layerData.EyeFov.Fov[eye] = fov;

    // 描画データにビューポートを設定する
    oculus.layerData.EyeFov.Viewport[eye].Pos = OVR::Vector2i(0, 0);
    oculus.layerData.EyeFov.Viewport[eye].Size = textureSize;

    // 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
    ovrSwapTextureSet* colorTexture{ nullptr };
    ovr_CreateSwapTextureSetGL(oculus.session, GL_SRGB8_ALPHA8, textureSize.w, textureSize.h, &colorTexture);
    oculus.layerData.EyeFov.ColorTexture[eye] = colorTexture;

    // 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
    ovrSwapTextureSet* depthTexture{ nullptr };
    ovr_CreateSwapTextureSetGL(oculus.session, GL_DEPTH_COMPONENT32F, textureSize.w, textureSize.h, &depthTexture);
    oculus.layerData.EyeFovDepth.DepthTexture[eye] = depthTexture;

    // Oculus Rift のレンズ補正等の設定値を取得する
    oculus.eyeRenderDesc[eye] = ovr_GetRenderDesc(oculus.session, ovrEyeType(eye), fov);

#  endif
  }

#  if OVR_PRODUCT_VERSION > 0

  // 姿勢のトラッキングにおける床の高さを 0 に設定する
  ovr_SetTrackingOriginType(oculus.session, ovrTrackingOrigin_FloorLevel);

  // ミラー表示用の FBO を作成する
  const GLsizei* size{ oculus.window->getSize() };
  const ovrMirrorTextureDesc mirrorDesc
  {
    OVR_FORMAT_R8G8B8A8_UNORM_SRGB, // Format
    oculus.mirrorWidth = size[0],   // Width
    oculus.mirrorHeight = size[1],  // Height
    0                               // Flags
  };

  // 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
  if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, &mirrorDesc, &oculus.mirrorTexture)))
  {
    // 作成したテクスチャのテクスチャ名を得る
    GLuint texId{ 0 };
    if (OVR_SUCCESS(ovr_GetMirrorTextureBufferGL(oculus.session, oculus.mirrorTexture, &texId)))
    {
      // 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
      glGenFramebuffers(1, &oculus.mirrorFbo);
      glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
      glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texId, 0);
      glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
      glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
    }
  }

#  else

  // 作成したテクスチャのテクスチャ名を得る
  if (OVR_SUCCESS(ovr_CreateMirrorTextureGL(oculus.session, GL_SRGB8_ALPHA8, width, height, reinterpret_cast<ovrTexture**>(&mirrorTexture))))
```

```
  {
    // 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
    oculus.mirrorFbo = 0;
    glGenFramebuffers(1, &oculus.mirrorFbo);
    glBindFramebuffer(GL_READ_FRAMEBUFFER, oculus.mirrorFbo);
    glFramebufferTexture2D(GL_READ_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
 mirrorTexture->OGL.TexId, 0);
    glFramebufferRenderbuffer(GL_READ_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, 0);
    glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
  }

#  endif

  // 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
  glEnable(GL_FRAMEBUFFER_SRGB);

  // フロントバッファに描く
  glDrawBuffer(GL_FRONT);

  // Oculus Rift への表示では垂直同期タイミングに合わせない
  glfwSwapInterval(0);

  return oculus;
}

//
// Oculus Rift のセッションを破棄する
//
void GgApp::Oculus::terminate()
{
  // session が無効なら何もしない
  if (!session) return;

  // ミラー表示用の FBO を作っていたら削除する
  if (mirrorFbo)
  {
    glDeleteFramebuffers(1, &mirrorFbo);
    mirrorFbo = 0;
  }

  // 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
  if (mirrorTexture)
  {
#  if OVR_PRODUCT_VERSION > 0
    ovr_DestroyMirrorTexture(session, mirrorTexture);
#  else
    glDeleteTextures(1, &mirrorTexture->OGL.TexId);
    ovr_DestroyMirrorTexture(session, reinterpret_cast<ovrTexture*>(mirrorTexture));
#  endif
    mirrorTexture = nullptr;
  }

  // 全ての目について
  for (int eye = 0; eye < ovrEye_Count; ++eye)
  {
    // Oculus Rift へのレンダリング用の FBO を削除する
    glDeleteFramebuffers(1, oculusFbo + eye);
    oculusFbo[eye] = 0;

#  if OVR_PRODUCT_VERSION > 0

    // 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
    if (layerData.ColorTexture[eye])
    {
      ovr_DestroyTextureSwapChain(session, layerData.ColorTexture[eye]);
      layerData.ColorTexture[eye] = nullptr;
    }

    // デプスバッファとして使ったテクスチャを削除する
    glDeleteTextures(1, oculusDepth + eye);
    oculusDepth[eye] = 0;

#  else

    // 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
    auto* const colorTexture(layerData.EyeFov.ColorTexture[eye]);
    for (int i = 0; i < colorTexture->TextureCount; ++i)
    {
      const auto* const ctex(reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[i]));
      glDeleteTextures(1, &ctex->OGL.TexId);
    }
    ovr_DestroySwapTextureSet(session, colorTexture);

    // デプスバッファとして使ったテクスチャを削除する
    auto* const depthTexture(layerData.EyeFovDepth.DepthTexture[eye]);
    for (int i = 0; i < depthTexture->TextureCount; ++i)
    {
```

```
      const auto* const dtex(reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[i]));
      glDeleteTextures(1, &dtex->OGL.TexId);
    }
    ovr_DestroySwapTextureSet(session, depthTexture);

#  endif
  }

  // Oculus Rift のセッションを破棄する
  ovr_Destroy(session);
  session = nullptr;

  // カラースペースを元に戻す
  glDisable(GL_FRAMEBUFFER_SRGB);

  // バックバッファに描く
  glDrawBuffer(GL_BACK);

  // 垂直同期タイミングに合わせる
  glfwSwapInterval(1);
}

//
// Oculus Rift による描画開始
//
bool GgApp::Oculus::begin()
{
#  if OVR_PRODUCT_VERSION > 0

  // セッションの状態を取得する
  ovrSessionStatus sessionStatus;
  ovr_GetSessionStatus(session, &sessionStatus);

  // 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
  if (sessionStatus.ShouldQuit) window->setClose(GLFW_TRUE);

  // Oculus Rift に表示されていないときは戻る
  if (!sessionStatus.IsVisible) return false;

  // 現在の状態をトラッキングの原点にする
  if (sessionStatus.ShouldRecenter) ovr_RecenterTrackingOrigin(session);

  // 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
  const ovrEyeRenderDesc eyeRenderDesc[]
  {
    ovr_GetRenderDesc(session, ovrEyeType(0), hmdDesc.DefaultEyeFov[0]),
    ovr_GetRenderDesc(session, ovrEyeType(1), hmdDesc.DefaultEyeFov[1])
  };

  // 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
  const ovrPosef hmdToEyePose[]
  {
    eyeRenderDesc[0].HmdToEyePose,
    eyeRenderDesc[1].HmdToEyePose
  };

  // 視点の姿勢情報を取得する
  ovr_GetEyePoses(session, frameIndex, ovrTrue, hmdToEyePose, layerData.RenderPose,
 &layerData.SensorSampleTime);

#  else

  // フレームのタイミング計測開始
  const auto ftiming(ovr_GetPredictedDisplayTime(session, 0));

  // sensorSampleTime の取得は可能な限り ovr_GetTrackingState() の近くで行う
  layerData.EyeFov.SensorSampleTime = ovr_GetTimeInSeconds();

  // ヘッドトラッキングの状態を取得する
  const auto hmdState(ovr_GetTrackingState(session, ftiming, ovrTrue));

  // 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
  const ovrVector3f hmdToEyeViewOffset[]
  {
    eyeRenderDesc[0].HmdToEyeViewOffset,
    eyeRenderDesc[1].HmdToEyeViewOffset
  };

  // 視点の姿勢情報を求める
  ovr_CalcEyePoses(hmdState.HeadPose.ThePose, hmdToEyeViewOffset, eyePose);

#  endif

  return true;
}

//
```

```
// Oculus Rift の描画する目の指定
//
void GgApp::Oculus::select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation)
{
#  if OVR_PRODUCT_VERSION > 0

  // Oculus Rift にレンダリングする FBO に切り替える
  if (layerData.ColorTexture[eye])
  {
    // 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
    int curIndex;
    ovr_GetTextureSwapChainCurrentIndex(session, layerData.ColorTexture[eye], &curIndex);

    // FBO のカラーバッファに使うテクスチャを取得する
    GLuint curTexId;
    ovr_GetTextureSwapChainBufferGL(session, layerData.ColorTexture[eye], curIndex, &curTexId);

    // FBO を設定する
    glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, curTexId, 0);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, oculusDepth[eye], 0);

    // ビューポートを設定する
    const auto& vp{ layerData.Viewport[eye] };
    glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);
  }

  // Oculus Rift の片目の位置と回転を取得する
  const auto& p{ layerData.RenderPose[eye].Position };
  const auto& o{ layerData.RenderPose[eye].Orientation };

#  else

  // 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
  auto* const colorTexture{ layerData.EyeFov.ColorTexture[eye] };
  colorTexture->CurrentIndex = (colorTexture->CurrentIndex + 1) % colorTexture->TextureCount;
  auto* const depthTexture{ layerData.EyeFovDepth.DepthTexture[eye] };
  depthTexture->CurrentIndex = (depthTexture->CurrentIndex + 1) % depthTexture->TextureCount;

  // レンダーターゲットを切り替える
  glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
  const auto& ctex{ reinterpret_cast<ovrGLTexture*>(&colorTexture->Textures[colorTexture->CurrentIndex]) };
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ctex->OGL.TexId, 0);
  const auto& dtex{ reinterpret_cast<ovrGLTexture*>(&depthTexture->Textures[depthTexture->CurrentIndex]) };
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, dtex->OGL.TexId, 0);

  // ビューポートを設定する
  const auto& vp{ layerData.EyeFov.Viewport[eye] };
  glViewport(vp.Pos.x, vp.Pos.y, vp.Size.w, vp.Size.h);

  // Oculus Rift の片目の位置と回転を取得する
  const auto& p{ eyePose[eye].Position };
  const auto& o{ eyePose[eye].Orientation };

#  endif

  // Oculus Rift のスクリーンの大きさを返す
  screen[0] = this->screen[eye][0];
  screen[1] = this->screen[eye][1];
  screen[2] = this->screen[eye][2];
  screen[3] = this->screen[eye][3];

  // Oculus Rift の位置を返す
  position[0] = p.x;
  position[1] = p.y;
  position[2] = p.z;

  // Oculus Rift の方向を返す
  orientation[0] = o.x;
  orientation[1] = o.y;
  orientation[2] = o.z;
  orientation[3] = o.w;
}

//
// Time Warp 処理に使う投影変換行列の成分の設定 (DK1, DK2)
//
void GgApp::Oculus::timewarp(const GgMatrix& projection)
{
#  if OVR_PRODUCT_VERSION < 1
  // TimeWarp に使う変換行列の成分を設定する
  auto& posTimewarpProjectionDesc{ layerData.EyeFovDepth.ProjectionDesc };
  posTimewarpProjectionDesc.Projection22 = (projection.get()[4 * 2 + 2] + projection.get()[4 * 3 + 2]) * 0.5f;
  posTimewarpProjectionDesc.Projection23 = projection.get()[4 * 2 + 3] * 0.5f;
```

```
  posTimewarpProjectionDesc.Projection32 = projection.get()[4 * 3 + 2];
#  endif
}

//
// 図形の描画を完了する (CV1 以降)
//
void GgApp::Oculus::commit(int eye)
{
#  if OVR_PRODUCT_VERSION > 0
  // GL_COLOR_ATTACHMENT0 に割り当てられたテクスチャが wglDXUnlockObjectsNV() によって
  // 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
  // FBO に結合されるのを避ける
  glBindFramebuffer(GL_FRAMEBUFFER, oculusFbo[eye]);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, 0, 0);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, 0, 0);

  // 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
  ovr_CommitTextureSwapChain(session, layerData.ColorTexture[eye]);
#  endif
}

//
// フレームを転送する
//
bool GgApp::Oculus::submit(bool mirror)
{
  // エラーチェック
  ggError();

#  if OVR_PRODUCT_VERSION > 0
  // 描画データを Oculus Rift に転送する
  const auto* const layers{ &layerData.Header };
  if (OVR_FAILURE(ovr_SubmitFrame(session, frameIndex++, nullptr, &layers, 1))) return false;
#  else
  // Oculus Rift 上の描画位置と拡大率を求める
  ovrViewScaleDesc viewScaleDesc;
  viewScaleDesc.HmdSpaceToWorldScaleInMeters = 1.0f;
  viewScaleDesc.HmdToEyeViewOffset[0] = eyeRenderDesc[0].HmdToEyeViewOffset;
  viewScaleDesc.HmdToEyeViewOffset[1] = eyeRenderDesc[1].HmdToEyeViewOffset;

  // 描画データを更新する
  layerData.EyeFov.RenderPose[0] = eyePose[0];
  layerData.EyeFov.RenderPose[1] = eyePose[1];

  // 描画データを Oculus Rift に転送する
  const auto* const layers{ &layerData.Header };
  if (OVR_FAILURE(ovr_SubmitFrame(session, 0, &viewScaleDesc, &layers, 1))) return false;
#  endif

  // ミラー表示
  if (mirror)
  {
#  if OVR_PRODUCT_VERSION > 0
    const auto& sx1{ mirrorWidth };
    const auto& sy1{ mirrorHeight };
#  else
    const auto& sx1{ mirrorTexture->OGL.Header.TextureSize.w };
    const auto& sy1{ mirrorTexture->OGL.Header.TextureSize.h };
#  endif

    // ミラー表示のウィンドウのサイズ
    GLsizei size[2];
    window->getSize(size);

    // ミラー表示の表示領域
    GLint dx0{ 0 }, dx1{ size[0] }, dy0{ 0 }, dy1{ size[1] };

    // ミラー表示がウィンドウからはみ出ないようにする
    if ((size[0] *= sy1) < (size[1] *= sx1))
    {
      const GLint ty1{ size[0] / sx1 };
      dy0 = (dy1 - ty1) / 2;
      dy1 = dy0 + ty1;
    }
    else
    {
      const GLint tx1{ size[1] / sy1 };
      dx0 = (dx1 - tx1) / 2;
      dx1 = dx0 + tx1;
    }

    // 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
    glBindFramebuffer(GL_READ_FRAMEBUFFER, mirrorFbo);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glBlitFramebuffer(0, sy1, sx1, 0, dx0, dy0, dx1, dy1, GL_COLOR_BUFFER_BIT, GL_NEAREST);
    glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
```

```
01119
01120     // 残っている OpenGL コマンドを実行する
01121     glFlush();
01122   }
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]

```
#pragma once


// Dear ImGui を使うなら
#define GG_USE_IMGUI

// Oculus Rift を使うなら
//#define GG_USE_OCULUS_RIFT

// 使用するマウスのボタン数
#if !defined(GG_BUTTON_COUNT)
#  define GG_BUTTON_COUNT 3
#endif

// 使用するユーザインタフェースの数
#if !defined(GG_INTERFACE_COUNT)
#  define GG_INTERFACE_COUNT 5
#endif

// 補助プログラム
#include "gg.h"
using namespace gg;

// 標準ライブラリ
#include <cassert>
#include <stdexcept>
#include <iostream>

// ImGui の組み込み
#if defined(GG_USE_IMGUI)
#  include "imgui.h"
#  include "imgui_impl_glfw.h"
#  include "imgui_impl_opengl3.h"
#endif

// Oculus Rift SDK ライブラリ (LibOVR) の組み込み
#if defined(GG_USE_OCULUS_RIFT)
#  if defined(_MSC_VER)
#    define GLFW_EXPOSE_NATIVE_WIN32
#    define GLFW_EXPOSE_NATIVE_WGL
#    include <GLFW/glfw3native.h>
#    define OVR_OS_WIN32
#    undef APIENTRY
#    pragma comment(lib, "LibOVR.lib")
#  endif
#  include <OVR_CAPI_GL.h>
#  include <Extras/OVR_Math.h>
#  if OVR_PRODUCT_VERSION > 0
#    include <dxgi.h> // GetDefaultAdapterLuid のため
#    pragma comment(lib, "dxgi.lib")
#  endif
#endif

class GgApp
{
public:

  GgApp(int major = 0, int minor = 1);

  //
  // コピーコンストラクタは封じる
  //
  GgApp(const GgApp& w) = delete;

  virtual ~GgApp();

  //
  // 代入演算子は封じる
  //
  GgApp& operator=(const GgApp& w) = delete;

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

  class Window
  {
    // ウィンドウの識別子
    GLFWwindow* window;

    // ビューポートの横幅と高さ
    std::array<GLsizei, 2> size;

    // フレームバッファの横幅と高さ
```

```
    std::array<GLsizei, 2> fboSize;

#if defined(IMGUI_VERSION)
    // メニューバーの高さ
    GLsizei menubarHeight;
#endif

    // ビューポートのアスペクト比
    GLfloat aspect;

    // マウスの移動速度[X/Y/Z]
    std::array<GLfloat, 3> velocity;

    // マウスボタンの状態
    std::array<bool, GG_BUTTON_COUNT> status;

    // ユーザインタフェースのデータ構造
    struct HumanInterface
    {
      // 最後にタイプしたキー
      int lastKey;

      // 矢印キー
      std::array<std::array<int, 2>, 4> arrow;

      // マウスの現在位置
      std::array<GLfloat, 2> mouse;

      // マウスホイールの回転量
      std::array<GLfloat, 2> wheel;

      // 平行移動量[ボタン][直前/更新][X/Y/Z]
      std::array<std::array<GgVector, 2>, GG_BUTTON_COUNT> translation;

      // トラックボール
      std::array<GgTrackball, GG_BUTTON_COUNT> rotation;

      // コンストラクタ
      HumanInterface() :
        lastKey{ 0 },
        arrow{},
        mouse{},
        wheel{},
        translation{}
      {
        resetTranslation();
      }

      //
      // マウスや矢印キーによる平行移動量を初期化する
      //
      void resetTranslation();

      //
      // 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)
      //
      void calcTranslation(int button, const std::array<GLfloat, 3>& velocity);
    };

    // ヒューマンインタフェースデバイスのデータ
    std::array<HumanInterface, GG_INTERFACE_COUNT> interfaceData;

    // ヒューマンインタフェースデバイスの番号
    int interfaceNo;

    //
    // ユーザー定義のコールバック関数へのポインタ
    //
    void* userPointer;
    void (*resizeFunc)(const Window* window, int width, int height);
    void (*keyboardFunc)(const Window* window, int key, int scancode, int action, int mods);
    void (*mouseFunc)(const Window* window, int button, int action, int mods);
    void (*wheelFunc)(const Window* window, double x, double y);

    //
    // ウィンドウのサイズ変更時の処理
    //
    static void resize(GLFWwindow* window, int width, int height);

    //
    // キーボードをタイプした時の処理
    //
    static void keyboard(GLFWwindow* window, int key, int scancode, int action, int mods);

    //
    // マウスボタンを操作したときの処理
    //
```

```
    static void mouse(GLFWwindow* window, int button, int action, int mods);

    //
    // マウスホイールを操作した時の処理
    //
    static void wheel(GLFWwindow* window, double x, double y);

  public:

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

    //
    // コピーコンストラクタは封じる
    //
    Window(const Window& w) = delete;

    virtual ~Window()
    {
      // ウィンドウが作成されていなければ戻る
      if (!window) return;

      // ウィンドウを破棄する
      glfwDestroyWindow(window);
    }

    //
    // 代入演算子は封じる
    //
    Window& operator=(const Window& w) = delete;

    auto* get() const
    {
      return window;
    }

    void setClose(int flag = GLFW_TRUE) const
    {
      glfwSetWindowShouldClose(window, flag);
    }

    bool shouldClose() const
    {
      // ウィンドウを閉じるべきなら true を返す
      return glfwWindowShouldClose(window) != GLFW_FALSE;
    }

    explicit operator bool();

    void swapBuffers() const;

    void restoreViewport() const
    {
      glViewport(0, 0, fboSize[0], fboSize[1]);
    }

    void updateViewport();

#if defined(IMGUI_VERSION)
    void setMenubarHeight(GLsizei height)
    {
      // メニューバーの高さを保存する
      menubarHeight = height;

      // ビューポートを復帰する
      updateViewport();
    }
#endif

    auto getWidth() const
    {
      return size[0];
    }

    auto getHeight() const
    {
      return size[1];
    }

    auto getFboWidth() const
    {
      return fboSize[0];
    }

    auto getFboHeight() const
    {
      return fboSize[1];
```

```
    }

    const auto& getSize() const
    {
      return size;
    }

    void getSize(GLsizei* size) const
    {
      size[0] = getWidth();
      size[1] = getHeight();
    }

    const auto getFboSize() const
    {
      return fboSize.data();
    }

    void getFboSize(GLsizei* fboSize) const
    {
      fboSize[0] = getFboWidth();
      fboSize[1] = getFboHeight();
    }

    auto getAspect() const
    {
      return aspect;
    }

    bool getKey(int key) const
    {
#if defined(IMGUI_VERSION)
      // 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
      if (ImGui::GetIO().WantCaptureKeyboard) return false;
#endif

      return glfwGetKey(window, key) != GLFW_RELEASE;
    }

    void selectInterface(int no)
    {
      assert(static_cast<size_t>(no) < interfaceData.size());
      interfaceNo = no;
    }

    void setVelocity(GLfloat vx, GLfloat vy, GLfloat vz = 0.1f)
    {
      velocity = std::array<GLfloat, 3>{ vx, vy, vz };
    }

    int getLastKey()
    {
      auto& current_if{ interfaceData[interfaceNo] };
      const int key{ current_if.lastKey };
      current_if.lastKey = 0;
      return key;
    }

    auto getArrow(int direction = 0, int mods = 0) const
    {
      const auto& current_if{ interfaceData[interfaceNo] };
      return static_cast<GLfloat>(current_if.arrow[mods & 3][direction & 1]);
    }

    auto getArrowX(int mods = 0) const
    {
      return getArrow(0, mods);
    }

    auto getArrowY(int mods = 0) const
    {
      return getArrow(1, mods);
    }

    void getArrow(GLfloat* arrow, int mods = 0) const
    {
      arrow[0] = getArrowX(mods);
      arrow[1] = getArrowY(mods);
    }

    auto getShiftArrowX() const
    {
      return getArrow(0, 1);
    }

    auto getShiftArrowY() const
    {
```

```
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 getAltlArrow(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
```

```
    {
      return userPointer;
    }

    void setUserPointer(void* pointer)
    {
      userPointer = pointer;
    }

    void setResizeFunc(void (*func)(const Window* window, int width, int height))
    {
      resizeFunc = func;
    }

    void setKeyboardFunc(void (*func)(const Window* window, int key, int scancode, int action, int
  mods))
    {
      keyboardFunc = func;
    }

    void setMouseFunc(void (*func)(const Window* window, int button, int action, int mods))
    {
      mouseFunc = func;
    }

    void setWheelFunc(void (*func)(const Window* window, double x, double y))
    {
      wheelFunc = func;
    }
  };

#if defined(GG_USE_OCULUS_RIFT)
  class Oculus
  {
    // Oculus Rift のセッション
    ovrSession session;

    // Oculus Rift の状態
    ovrHmdDesc hmdDesc;

    // Oculus Rift へのレンダリングに使う FBO
    GLuint oculusFbo[ovrEye_Count];

    // Oculus Rift のスクリーンのサイズ
    GLfloat screen[ovrEye_Count][4];

    // ミラー表示用の FBO
    GLuint mirrorFbo;

    // Oculus Rift のミラー表示を行うウィンドウ
    const Window* window;

#  if OVR_PRODUCT_VERSION > 0

    // Oculus Rift に送る描画データ
    ovrLayerEyeFov layerData;

    // Oculus Rift にレンダリングするフレームの番号
    long long frameIndex;

    // Oculus Rift へのレンダリングに使う FBO のデプステクスチャ
    GLuint oculusDepth[ovrEye_Count];

    // ミラー表示用の FBO のサイズ
    int mirrorWidth, mirrorHeight;

    // ミラー表示用の FBO のカラーテクスチャ
    ovrMirrorTexture mirrorTexture;

    //
    // グラフィックスカードのデフォルトの LUID を得る
    //
    static ovrGraphicsLuid GetDefaultAdapterLuid();

    //
    // グラフィックスカードの LUID の比較
    //
    static int Compare(const ovrGraphicsLuid& lhs, const ovrGraphicsLuid& rhs);

#  else

    // Oculus Rift に送る描画データ
    ovrLayer_Union layerData;

    // Oculus Rift のレンダリング情報
    ovrEyeRenderDesc eyeRenderDesc[ovrEye_Count];

```

```
    // Oculus Rift の視点情報
    ovrPosef eyePose[ovrEye_Count];

    // ミラー表示用の FBO のカラーテクスチャ
    ovrGLTexture* mirrorTexture;

#  endif

    //
    // コンストラクタ
    //
    Oculus();

    //
    // デストラクタ
    //
    virtual ~Oculus() = default;

  public:

    // シングルトンなのでコピー・ムーブ禁止
    Oculus(const Oculus&) = delete;
    Oculus& operator=(const Oculus&) = delete;
    Oculus(Oculus&&) = delete;
    Oculus& operator=(Oculus&&) = delete;

    static Oculus& initialize(const Window& window);

    void terminate();

    bool begin();

    void select(int eye, GLfloat* screen, GLfloat* position, GLfloat* orientation);

    void timewarp(const GgMatrix& projection);

    void commit(int eye);

    bool submit(bool mirror = true);
  };
#endif
};
```

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 };
```

```
  // 図形の描画の設定を行う
  Draw draw{ menu };

  // ビュー変換行列を設定する
  const GgMatrix mv{ ggLookat(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f) };

  // ウィンドウが開いている間繰り返す
  while (window)
  {
    // ウィンドウを消去する
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // メニューを表示する
    menu.draw();

    // オブジェクトの回転の変換行列を設定する
    const auto& mr{ window.getRotationMatrix(0) };

    // 視点の平行移動の変換行列を設定する
    const auto& mt{ window.getTranslationMatrix(1) };

    // 投影変換行列を設定する
    const GgMatrix&& mp{ ggPerspective(0.5f, window.getAspect(), 1.0f, 15.0f) };

    // 図形を描画する
    draw.draw(mp, mt * mv * mr);

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

  return 0;
}
```

9.19 main.cpp ファイル

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

main.cpp の依存先関係図:

マクロ定義

- #define HEADER_STR "ゲームグラフィックス特論"

関数

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

9.19.1 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 行目に定義があります。

9.19.2 関数詳解

9.19.2.1 main()

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

main.cpp の 19 行目に定義があります。

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 <atlstr.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 //
00017 // メインプログラム
00018 //
00019 int main(int argc, const char* const* argv) try
00020 {
00021   // アプリケーションのオブジェクトを生成する
00022 #if defined(GL_GLES_PROTOTYPES)
00023   GgApp app(3, 1);
00024 #else
00025   GgApp app(4, 1);
00026 #endif
00027
00028   // アプリケーションを実行する
00029   return app.main(argc, argv);
```

```
}
catch (const std::runtime_error &e)
{
  // エラーメッセージを表示する
#if defined(_MSC_VER)
  MessageBox(NULL, CString(e.what()), TEXT(HEADER_STR), MB_ICONERROR);
#elif defined(__APPLE__)
  // the following code is copied from http://blog.jorgearimany.com/2010/05/messagebox-from-windows-to-mac.html
  // convert the strings from char* to CFStringRef
  CFStringRef msg_ref = CFStringCreateWithCString(NULL, e.what(), kCFStringEncodingUTF8);

  // result code from the message box
  CFOptionFlags result;

  //launch the message box
  CFUserNotificationDisplayAlert(
    0,                                  // no timeout
    kCFUserNotificationNoteAlertLevel, // change it depending message_type flags ( MB_ICONASTERISK.... etc.)
    NULL,                               // icon url, use default, you can change it depending message_type flags
    NULL,                               // not used
    NULL,                               // localization of strings
    CFSTR(HEADER_STR),                 // header text
    msg_ref,                           // message text
    NULL,                               // default "ok" text in button
    NULL,                               // alternate button title
    NULL,                               // other button title, null--> no other button
    &result                             // response flags
  );

  // Clean up the strings
  CFRelease(msg_ref);
#else
  std::cerr << HEADER_STR << ": " << e.what() << '\n';
#endif

  // プログラムを終了する
  return EXIT_FAILURE;
}
```

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]

```
#include "Menu.h"

//
// コンストラクタ
//
Menu::Menu(const Config& config) :
  defaults{ config },
  settings{ config },
  model{ std::make_unique<GgSimpleObj>(config.model, true) },
  light{ std::make_unique<GgSimpleShader::LightBuffer>(config.light) },
  shader{ std::make_unique<GgSimpleShader>(config.shader) }
{
#if defined(IMGUI_VERSION)
  //
  // ImGui の初期設定
  //

 // ファイルダイアログ (Native File Dialog Extended) を初期化する
  NFD_Init();

  // Dear ImGui の入力デバイス
  //io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;     // キーボードコントロールを使う
  //io.ConfigFlags |= ImGuiConfigFlags_NavEnableGamepad;      // ゲームパッドを使う

  // Dear ImGui のスタイル
  //ImGui::StyleColorsDark();                                  // 暗めのスタイル
  //ImGui::StyleColorsClassic();                               // 以前のスタイル

  // 日本語を表示できるメニューフォントを読み込む
  if (!ImGui::GetIO().Fonts->AddFontFromFileTTF(config.menuFont.c_str(), config.menuFontSize, nullptr,
    ImGui::GetIO().Fonts->GetGlyphRangesJapanese()))
  {
    // メニューフォントが読み込めなかったらエラーにする
    throw std::runtime_error("Cannot find any menu fonts.");
  }
#endif
}

//
// デストラクタ
//
Menu::~Menu()
{
}

//
// ファイルパスを取得する
//
bool Menu::getFilePath(std::string& path, const nfdfilteritem_t* filter)
{
  // ファイルダイアログから得るパス
  nfdchar_t* filepath{ nullptr };

  // ファイルダイアログを開く
  if (NFD_OpenDialog(&filepath, filter, 1, nullptr) == NFD_OKAY)
  {
    path = TCharToUtf8(filepath);
    return true;
  }

  return false;
```

```
}

//
// 描画する
//
void Menu::draw()
{
#if defined(IMGUI_VERSION)
  //
  // ImGui によるユーザインタフェース
  //

  // ImGui のフレームを準備する
  ImGui::NewFrame();

  // メニューの表示位置を決定する
  ImGui::SetNextWindowPos(ImVec2(4, 4), ImGuiCond_Once);
  ImGui::SetNextWindowSize(ImVec2(320, 98), ImGuiCond_Once);

  // メニュー表示開始
  ImGui::Begin(u8"コントロールパネル");

  // 光源
  if (ImGui::SliderFloat3(u8"光源位置", settings.light.position.data(), -10.0f, 10.0f, "%.2f"))
    light->loadPosition(settings.light.position);
  if (ImGui::ColorEdit3(u8"光源色", settings.light.diffuse.data(), ImGuiColorEditFlags_Float))
    light->loadDiffuse(settings.light.diffuse);

  // メニュー表示終了
  ImGui::End();

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

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   // シェーダ
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
00048   Menu(const Config& config);
00049
00050   // コピーコンストラクタは封じる
00051   Menu(const Menu& menu) = delete;
00052
00056   virtual ~Menu();
00057
00058   // 代入演算子は封じる
00059   Menu& operator=(const Menu& menu) = delete;
00060
00064   void draw();
00065 };
```

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

U	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

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
strings	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

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
string	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

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
strings	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

T	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
U	構成ファイルから取得する配列の要素の数値の数

引数

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
vector	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

T	構成ファイルから取得する数値のデータ型

引数

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
scalar	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

object	構成ファイルの JSON オブジェクト
key	取得する JSON オブジェクトのキー
vector	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

U	構成ファイルに設定する配列の要素の文字列の数

引数

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
strings	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

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
string	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

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
strings	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

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

引数

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
vector	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

T	構成ファイルに設定する数値のデータ型

引数

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
scalar	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

object	構成ファイルの JSON オブジェクト
key	設定する JSON オブジェクトのキー
vector	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]

```
#pragma once


// JSON
#include "picojson.h"

template <typename T>
bool getValue(const picojson::object& object,
  const std::string& key, T& scalar)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか数値でなかったら戻る
  if (value == object.end() || !value->second.is<double>()) return false;

  // 数値として格納する
  scalar = static_cast<T>(value->second.get<double>());

  return true;
}

template <typename T, std::size_t U>
bool getValue(const picojson::object& object,
  const std::string& key, std::array<T, U>& vector)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか配列でなかったら戻る
  if (value == object.end() || !value->second.is<picojson::array>()) return false;

  // 配列を取り出す
  const auto& array{ value->second.get<picojson::array>() };

  // 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
  const auto n{ std::min(U, array.size()) };

  // 配列の要素について
  for (std::size_t i = 0; i < n; ++i)
  {
    // 要素が数値なら格納する
    if (array[i].is<double>()) vector[i] = static_cast<T>(array[i].get<double>());
  }

  return true;
}

inline
bool getVector(const picojson::object& object,
  const std::string& key, GgVector& vector)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか配列でなかったら戻る
  if (value == object.end() || !value->second.is<picojson::array>()) return false;

  // 配列を取り出す
  const auto& array{ value->second.get<picojson::array>() };

  // 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
  const auto n{ std::min(sizeof(GgVector) / sizeof(GLfloat), array.size()) };

  // 配列の要素について
  for (std::size_t i = 0; i < n; ++i)
  {
    // 要素が数値なら格納する
    if (array[i].is<double>()) vector[i] = static_cast<GLfloat>(array[i].get<double>());
  }

  return true;
}

inline
bool getString(const picojson::object& object,
  const std::string& key, std::string& string)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか文字列でなかったら戻る
```

```
  if (value == object.end() || !value->second.is<std::string>()) return false;

  // 文字列として格納する
  string = value->second.get<std::string>();

  return true;
}

template <std::size_t U>
bool getString(const picojson::object& object,
  const std::string& key, std::array<std::string, U>& strings)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか配列でなかったら戻る
  if (value == object.end() || !value->second.is<picojson::array>()) return false;

  // 配列を取り出す
  const auto& array{ value->second.get<picojson::array>() };

  // 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
  const auto n{ std::min(U, array.size()) };

  // 配列の要素について
  for (std::size_t i = 0; i < n; ++i)
  {
    // 要素が文字列なら文字列として格納する
    strings[i] = array[i].is<std::string>() ? array[i].get<std::string>() : "";
  }

  return true;
}

inline
bool getString(const picojson::object& object,
  const std::string& key, std::vector<std::string>& strings)
{
  // key に一致するオブジェクトを探す
  const auto&& value{ object.find(key) };

  // オブジェクトが無いか配列でなかったら戻る
  if (value == object.end() || !value->second.is<picojson::array>()) return false;

  // 配列を取り出す
  const auto& array{ value->second.get<picojson::array>() };

  // 配列のすべての要素について
  for (auto& element : array)
  {
    // 要素が文字列なら文字列として格納する
    strings.emplace_back(element.is<std::string>() ? element.get<std::string>() : "");
  }

  return true;
}

template <typename T>
void setValue(picojson::object& object,
  const std::string& key, const T& scalar)
{
  object.emplace(key, picojson::value(static_cast<double>(scalar)));
}

template <typename T, std::size_t U>
void setValue(picojson::object& object,
  const std::string& key, const std::array<T, U>& vector)
{
  // picojson の配列
  picojson::array array;

  // 配列のすべての要素について
  for (const auto& element : vector)
  {
    // 要素を picojson::array に追加する
    array.emplace_back(picojson::value(static_cast<double>(element)));
  }

  // オブジェクトに追加する
  object.emplace(key, array);
}

inline
void setVector(picojson::object& object,
  const std::string& key, const GgVector& vector)
{
  // picojson の配列
```

```
  picojson::array array;

  // ベクトルのすべての要素について
  for (const auto& element : vector)
  {
    // 要素を picojson::array に追加する
    array.emplace_back(picojson::value(static_cast<double>(element)));
  }

  // オブジェクトに追加する
  object.emplace(key, array);
}

inline
void setString(picojson::object& object,
  const std::string& key, const std::string& string)
{
  object.emplace(key, picojson::value(string));
}

template <std::size_t U>
void setString(picojson::object& object,
  const std::string& key, const std::array<std::string, U>& strings)
{
  // picojson の配列
  picojson::array array;

  // 配列のすべての要素について
  for (const auto& string : strings)
  {
    // 要素を picojson::array に追加する
    array.emplace_back(picojson::value(string));
  }

  // オブジェクトに追加する
  object.emplace(key, array);
}

inline
void setString(picojson::object& object,
  const std::string& key, const std::vector<std::string>& strings)
{
  // picojson の配列
  picojson::array array;

  // 配列のすべての要素について
  for (auto& string : strings)
  {
    // 要素を picojson::array に追加する
    array.emplace_back(picojson::value(string));
  }

  // オブジェクトに追加する
  object.emplace(key, array);
}
```

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
CONFIG_FILE
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
getAltlArrow
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
getAltlArrow, 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